スレッドレベル並列処理プロセッサの検討

1L－4

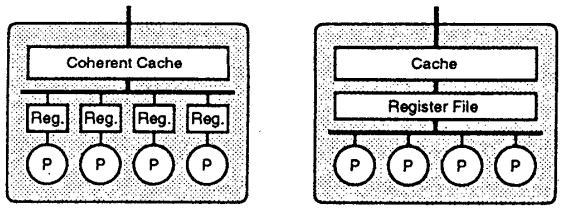
鳥居 淳†　本村 真人‡　井上 俊明‡　小長谷 明彦†
NEC †C&C研究所 ‡マイクロエレクトロニクス研究所

1 はじめに

近年のマイクロプロセッサは、スーパスカラ技術などの命令レベルの並列性を活用することで性能向上を実現してきた。しかしながら命令レベルの並列性自身の限界や、命令発行の論理の複雑化によって、従来技術による性能向上は限界に達しつつある。一方、従来の単一命令流(スレッド)を複数の命令流に分割し、これらを並列に実行する処理方法であるスレッドレベル並列処理についての検討はまだまだ不十分である。このスレッドレベル並列処理は並列性を十分に持った問題の場合、その問題に含まれる大局的な並列性を自然に取り出せるという利点を持つ。本研究では、このスレッドレベル並列処理で問題となるスレッド生成、消滅、同期オーバヘッドについて、アーキテクチャ・サポートを提案し、これらの効果についてシミュレーションにより初期評価を行なった。この結果スレッドレベル並列処理の有用性と今後の課題が明らかになった。

2 スレッドレベル並列処理

並列に処理を行なうことが可能なスレッドが存在する場合、これらを並列に実行するためには、新たにスレッドを生成する必要がある。実行速度を向上させるためには、このスレッド生成のコストに比して並列処理単位が大きくなくてはならない。しかしながら、1チップに複数PE (Processing Element)を集積した場合、PE間の交信コストの低減が可能となり、比較的粒度の小さいスレッドも並列処理可能になると考えられる。したがって、本研究では図1に示すような1チップに複数のPEを集積したプロセッサの検討を行なった。



a）スレッドレベル並列プロセッサ　b）スーパスカラプロセッサ

図 1: 1チップのプロセッサイメージ

2.1 スレッド生成のアーキテクチャサポート

1チップに複数PEを集積しPE間交信のコストが低くなった利点を最大限に活用するため、スレッド生成やスレッドの終了を1ないし数サイクルで行なう命令を用意し、アーキテクチャ的にスレッド生成をサポートすることにする。さらに、スレッドレベル並列処理では、プログラムから並列実行可能な部分をスレッドとして分割して実行を行なうため、並列実行可能なスレッド数はアルゴリズムに依存し、利用できるPE数を上回る可能性が生じる。このような場合のスレッド実行モデルとして以下の3種類のモデルが考えられる。

1. 無限長のスレッドキューを設ける
2. 有限長のスレッドキューを設ける。スレッドキューがいっぱいの時は逐次処理をする。
3. スレッドキューがないため、これ以上のスレッド生成を行なわずに逐次処理をする。

スレッドが生成できない場合に逐次処理を行なう方法としては、ハードウェアによってスレッド生成命令をサブルーチン呼び出す命令として実行するか、ソフトウェアによってスレッド生成可否を判定し、できない場合には逐次処理ルーチンを呼び出す方法が考えられる。また、スレッドキューを持たない3のモデルでは、各PEの状態をプロセッサ内で一元的にハードウェア管理するか、ソフトウェア管理するかによってスレッド生成の方法が異なる。これらを踏まえ、本研究では表1に示した4つのプロセッサモデルを比較した。

表 1: スレッド生成プロセッサモデル

	スレッドキュー	逐次化	PE状態ハードウェア管理
A	キュー長さ64	－	有
B	キュー長さ4	ハードウェア	有
C	なし	ハードウェア	有
D	なし	ソフトウェア	無

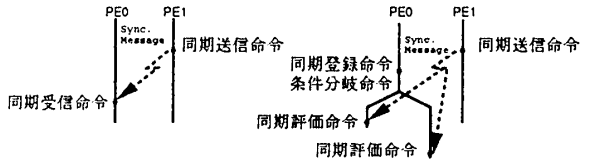
2.2 同期のアーキテクチャサポート

同期機構についてはこれまでに種々の提案がなされているが、本研究では軽い同期機構として同期メッセージの送受信によって同期をとる方式を用いた。スレッド生成と同様に、同期メッセージの送受信は1命令で実現する。表2に同期機構の詳細仕様について示す。なお、スレッド生成モデルA、Bについては同期受信命令実行時に同期メッセージが未到着の場合は、スレッドスイッチが生じるものとする。

表 2: 同期のアーキテクチャサポート

メッセージの格納場所	受信側プロセッサ専用バッファ
送り側、受け側の特定	受信PE番号、命令アドレス、(同期ID)
その他	同期登録命令、同期評価命令を導入(図2)

An experimental evaluation of thread-level parallel processor architecture
Sunao Torii, Masato Motomura, Toshiaki Inoue,Akihiko Konagaya
NEC C&C Res. Labs. Microelectronics Res. Labs.



a）通常の同期　　b）遅延評価同期

図 2: Send-Receive 同期

3 評価

提案したアーキテクチャの評価を行なうために、いくつかのアプリケーションについて各モデル毎のコードを生成し、トレースシミュレーションを行なった。なお、評価コードは、MIPS R3000 の命令セットに、スレッド管理命令、同期命令を加えたものとした。

表 3: トレースシミュレーションの条件

プロセッサモデル	A ～ C	D
スレッド生成成功時	2	2
スレッド生成失敗時	1	-
同期失敗スレッド切替時	3	Interlock
キャッシュヒット率	100%	100%

LU 分解、クイックソート、深さ優先探索の 3 つのアプリケーションを用いた評価結果を図 3に示す。

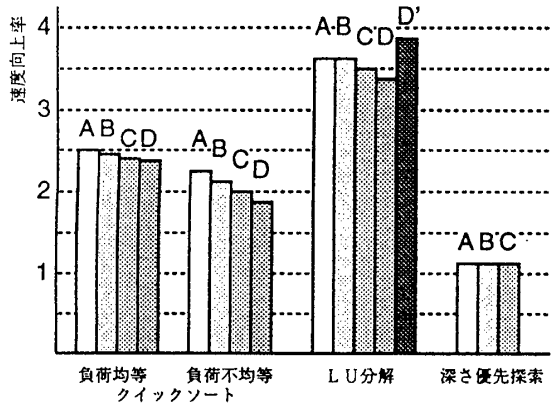


図 3: アプリケーションを用いた評価結果

クイックソートでは与える数列によってスレッド毎の負荷がほぼ均等になる場合と不均等になる場合があるので、2 種類の 100 要素の数列について評価した。また、LU 分解は 100 × 100 の要素のものについて評価し、モデル D についてはユーザコードレベルで LU 分解に特化したスレッドスケジューリングを行なった結果を D' として示した。深さ優先探索はプロセッサモデル D ではスレッド粒度が小さ過ぎて並列化できないため、モデル A ～ C について行なった。

クイックソートにおいては、負荷が不均等な場合において、ハードウェアサポートによるスレッドの動的スケジューリングの効果が表れている。しかしながら、負荷がほぼ均等の場合にはハードウェアによるスケジューリングの優位性はそれほど表れない。LU 分解についても同じような傾向を示している。

LU 分解では D' のモデルがもっとも性能が高くなっているが、D' のモデルでは PE 数に応じた最適なスレッド数を生成するためであり、他のモデルではループの 1 反復を 1 スレッドとしており、PE 数をスレッド数が上回るためこのような結果となる。このような手法はクイックソートなどのコード生成時に静的にスレッドの粒度がわからない場合に適用することは難しい。

深さ優先探索は、問題にほとんど並列性がないため、スレッドキューの有無は性能に影響しなかった。モデル D ではソフトウェアで PE 管理をするためのオーバヘッドによって、細粒度の並列化による性能向上は難しい。

4 まとめと今後の課題

今回の結果から、以下のことが明らかになった。

- ハードウェアサポートを幅広く行なうことにより、粒度の細かいスレッドに対しても並列性を活用でき、負荷分散もある程度行なえる。しかしながら、粒度の大きいスレッドに対してはハードウェアサポートによる差異はほとんどみられない。一方、細粒度の並列処理を行なうためには、命令レベル並列処理との併用も検討する必要がある。
- ユーザがスケジューリングを行った場合、データパラレル的な単純な並列性が存在し、静的に負荷が予測できる場合には性能的には有利となるが、複雑な並列処理では、ユーザに対する負担も大きくなる。

今回の検討ではキャッシュを考慮していないが、今後、これを含めたレベルでのシミュレーションを行ない、より正確な評価を行なう必要がある。スレッド毎にローカルなメモリ空間が必要となるので、スレッド数が増えた時のメモリ使用効率の問題も検討する必要がある。今後、命令セットアーキテクチャに基づいたハードウェアアーキテクチャの検討と評価、OS のスレッド処理、マルチタスク処理との整合性、他のハードウェアサポートの検討を行ない、スレッドレベル並列性の有効性を検証する予定である。

謝辞

本研究を行なうにあたって有益な助言を頂きました、C&C 研究所、マイクロエレクトロニクス研究所の諸氏に深く感謝します。

参考文献

[1] 井上他 “スレッドレベル並列処理アーキテクチャの検討”, 情報処理学会研究報告 計算機アーキテクチャ 107-11,pp81-88(1994).

[2] John L. Hennessy, David A. Patterson 著 富田眞治、村上 和彰、新實 治男訳：“コンピュータ・アーキテクチャ 設計・実現・評価の定量的アプローチ” 日経 BP 社 (1992).

[3] 岡本他 “超並列計算機 RWC-1 における同期機構”, 情報処理学会研究報告 計算機アーキテクチャ 101-2, pp9-16(1993).