複合粒度アーキテクチャ上での並列実行方式

佐藤充, 小池汎平, 田中英彦
{msato,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学部

1 はじめに

複合粒度アーキテクチャとは, 粗粒度プロセッサと細粒度プロセッサという異なる 2 つのプロセッサをもつアーキテクチャである. このようなアーキテクチャ上では, 通信や同期といった並列処理に伴う細粒度処理を, それに最適化された細粒度プロセッサに振り分けることによって, 粗粒度処理と細粒度処理を並列に実行することができる. しかし, このようなアーキテクチャ上では, 粗粒度プロセッサと細粒度プロセッサのバランスをとるのが難しく, コンパイル方法や実行方式も確立されてはいない. 本研究では, 並列プログラムを複合粒度アーキテクチャ上で効率的に実行するためのコード分割および実行方式について考察する.

2 複合粒度アーキテクチャ

複合粒度アーキテクチャとは, 図 1に示すように, ひとつのノードに 2 種類の異なるプロセッサを搭載したアーキテクチャである [1].

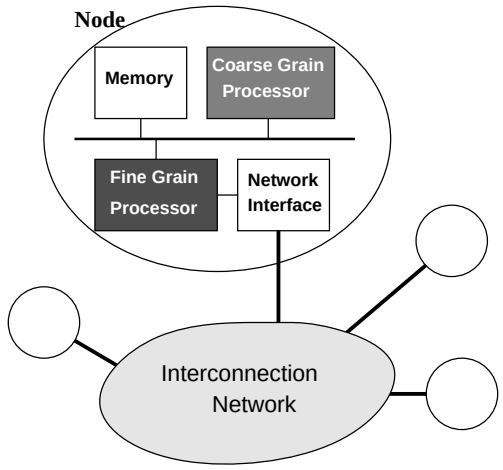

図 1: 複合粒度アーキテクチャ

粗粒度プロセッサとは逐次処理に最適化されたプロセッサである. このプロセッサでは従来の RISC プロセッサと同様に, キャッシュシステム, レジスタファイルなどの時間的・空間的局所性を利用した高速化を行なうことができる. 一方, 細粒度プロセッサとは通信・同期などの細粒度処理を高速に行なうプロセッサであり, 短時間スレッドを数多くこなすための, マルチコンテクスト機能などが備えられている.

このような複合粒度アーキテクチャは, 例えばJUMP-1で実装されている [2].

3 実行対象

複合粒度アーキテクチャは, 汎用並列処理を目的としたアーキテクチャである. すなわち, 行列演算などの定型的なアプリケーションを効率的に実行するだけではなく, AI 問題のような非定型的アプリケーションをも効率的に実行することを目的としている.

定型的アプリケーションの場合, その実行を効率的に行なうためには, アプリケーションに大きく依存した最適化を行なわなければならない場合が多く, 一般的に応用することは困難である.

また, 非定型的アプリケーションの多くは, 並列度が明らかでなく, 複雑である. このようなアプリケーションを並列に実行する場合は, アプリケーションに内在する並列度をうまく抽出し実行する細粒度の処理が適している.

以上のようなことから, 本研究では非定型的アプリケーションを細粒度並列実行によって処理する方式について検討する.

4 各プロセッサの処理の切りわけ

一般に細粒度並列処理は, 人間が並列度を抽出し明示的に同期をとる処理方式と比べて, 同期オーバーヘッドが大きく, その分処理の効率が悪いとされている.

しかし, 複合粒度アーキテクチャでは, 同期には細粒度プロセッサを用いることができるので, 粗粒度プロセッサで主処理を行なっている間に細粒度プロセッサで同期処理を行なうことができる.

また, 負荷分散やスケジューリングなどといった並列管理処理は, 逐次実行では必要のない並列オーバーヘッドに属するものである. これを主処理を行なう粗粒度プロセッサで行なうと, 全体の処理時間にオーバーヘッドが加わり, 処理効率が低下する.

Parallel Execution Method on Composite Grain Architecture
Mitsuru SATO, Hanpei KOIKE, Hidehiko TANAKA
Faculty of Engineering, University of Tokyo

従って, 各プロセッサに対する最適な処理の切りわけは, 粗粒度プロセッサに主処理だけを担当させ, その他の並列化によって生じる同期処理や並列管理処理を細粒度プロセッサが担当するという方式であると考えられる.

5 低負荷時実行モデル

以上のようなことを考えると, 複合粒度アーキテクチャ上での実行は図 2のようになる.

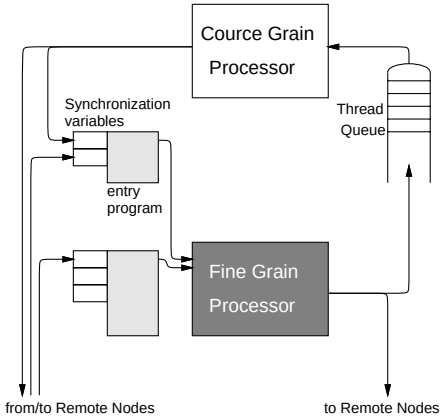

図 2: 複合粒度アーキテクチャ上での実行

図 2では, 細粒度の処理を行なうことを基本にしている. ここでの実行単位はスレッドであり, プログラムを静的に解析した結果を元に作られた処理の最小単位である.

細粒度プロセッサには, 各待ち合わせ変数に対応したプログラムが用意されている. 待ち合わせ変数に対する (リモート) アクセスが生じると, 登録されたプログラムが起動し, 待ち合わせなどの処理を行なう. その結果, 実行可能になったスレッドは, ローカルな粗粒度プロセッサの実行キューに入れられたり, 負荷分散戦略によってはリモートノードに送られる.

粗粒度プロセッサではスケジューリングなどの処理は行なわず, 細粒度プロセッサによって用意された実行キューから実行可能なスレッドを取り出し, 次々と実行する.

ただし, このようにスレッドという最小単位をスケジューリングの単位とすると, 実行時のローカリティが非常に低くなる. 粗粒度プロセッサは時間的・空間的ローカリティを利用した高速化を行なっているので, このままでは粗粒度プロセッサの実行効率が低下する. そのため TAM[3] で用いられいているように frame という概念を導入して, スケジューリングの単位を大きくするという工夫が必要になる.

6 高負荷状態での実行

上記実行モデルは, 実行可能なスレッドの数が少ない時, すなわち負荷の小さい時にうまく動作するように考えられたモデルである. しかしこのモデルでは, 実行可能なスレッドの数が増え, 負荷が上昇した時には必ずしも効率の良いものではない. すなわち, 周囲の負荷が上昇した場合, 新たに実行可能となったスレッドはもはやスケジューリングの必要がないにも関わらず, 一旦細粒度プロセッサを通してのみ実行可能状態になるため, プログラムのローカリティを充分に利用できないのである.

これを解決するためには, 高負荷状態でのスケジューリングを停止し, 現在実行中のスレッドから生成された実行可能なスレッドは連続して実行するような実行機構が必要になる.

このような実行機構として, 例えば低負荷時と高負荷時のコードを切替える手法がある [4]. これは, 低負荷時と高負荷時では動作が大きく違うので, それぞれに応じたコードを準備し, 負荷状態によって切替えて実行するという方法である. この場合, コード入れ換えのコストやコード量の増加などの問題が生じる. また高負荷状態から低負荷状態へ遷移した場合のコード切替えのオーバーヘッドも問題となる.

7 まとめ

粗粒度プロセッサと細粒度プロセッサという 2 つの異なる種類のプロセッサを融合させたアーキテクチャである複合粒度アーキテクチャ上でのプログラム実行方式について考察した.

今後は, さらに細かい実装について考察し, 実機またはシミュレータ上に適当な言語処理系の実装を行ない, その上で評価を行なっていく予定である.

謝辞

本研究の一部は文部省科学研究費 (重点領域研究 (1) 課題番号 04235130「超並列原理に基づく情報処理基本体系」) による。

参考文献

[1] 松本尚. 局所処理と非局所処理を分離並列処理するアーキテクチャ. 第 43 回情報処理学会全国大会講演論文集, 第 6 巻, pp. 115–116, October 1991.

[2] 平木敬, 松本尚. JUMP-1 MBP コアの命令設計. 第 49 回情報処理学会全国大会講演論文集, 第 6 巻, pp. 1–2, September 1994.

[3] David E.Culler, Anurag Sah, Klaus Erik Schauser, Thorsten von Eicken, and John Wawrzynek. Fine-grain Parallelism with Minimal Hardware Support: A Compiler-Controlled Threaded Abstract Machine. In *4th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 164–175, April 1991.

[4] 日高康雄, 小池汎平, 田中英彦. PIE64 における複合粒度並列処理を用いた最適粒度制御 –細粒度並列と粗粒度並列; 二つの異質な世界の分離と融合–. 並列処理シンポジウム JSPP'95 論文集, pp. 115–122, 1995.