命令レベル並列性を利用したOpenMPによるプロセッサシミュレータの並列実行

大島 聡史 † 檜田 敏克 †† 吉瀬 謙二 †† 片桐 孝洋 †† 本多 弘樹 †† 弓場 敏嗣 ††

† 電気通信大学 電気通信学部 †† 電気通信大学 大学院情報システム学研究科

1 はじめに

マイクロプロセッサの研究用、あるいは教育用として、様々なプロセッサシミュレータが利用されている。プロセッサに実装するアイディアの複雑化やベンチマークの巨大化等により、シミュレーションに要する時間は増加する傾向にある。そのため、様々な手法がシミュレータの高速化に用いられている [1]。並列化によるシミュレータの高速化も期待されており、わずかながら成功例もある [2]。

このような背景から、本稿ではプロセッサ数が 2 から 4 程度の SMP 型並列計算機を対象に、命令レベルの並列性を利用してプロセッサシミュレータを並列化し、高速実行させることを目指す。

これ以降、並列化したシミュレータのことを、並列シミュレータと呼ぶ。

2 並列シミュレータの提案

2.1 命令レベルの並列化

CPU が命令を実行する際、互いにデータ依存関係や制御依存関係の無い命令が連続、あるいは近接している場合がある。このような命令は、実行順序を入れ替えたり同時に実行したりしても、その前後のレジスタやメモリの値に差は生じない。スーパースカラ・プロセッサなどでは、このことを利用して命令の実行順序を入れ替え、全体の実行時間が短くなるようにしている [3]。プロセッサシミュレータにおいてもこれを利用し、複数の CPU に命令を割り当てて同時に実行することで、シミュレーション全体の実行時間を短縮することが可能になると考えられる。

2.2 複数の CPU へ命令を割り当てる方法

図 1 を例に、実際に複数の CPU へ命令を割り当てる方法を述べる。

(a) ケース 1 は、命令間に依存関係が無い。この場合は各命令を実行する CPU や実行順序に制限が無いため、各 CPU の実行する命令数が均等になるように命令を割り当てる。

(b) ケース 2 では、命令 1 と命令 2 は互いに依存関係が無いため、ケース 1 と同様に各 CPU に順番に割り当てる。命令 3 は CPU2 に割り当てられた命令 2 にのみデータ依存関係が有る。この場合は、命令 3 も CPU2 に割り当てることで、命令 2 とのデータ依存関係を保証する。

(c) ケース 3 も、命令 1 と命令 2 は互いに依存関係が無いため、各 CPU に順番に割り当てる。命令 3 は、命令 1 と命令 2 という複数の CPU に割り当てられた命令にデータ依存関係が有る。この場合は、命令 3 を実行する前に全 CPU 間で同期をとり、先行する依存関係のある命令が先に終了することを保証する。命令 3 は任意の CPU に割り当てる。

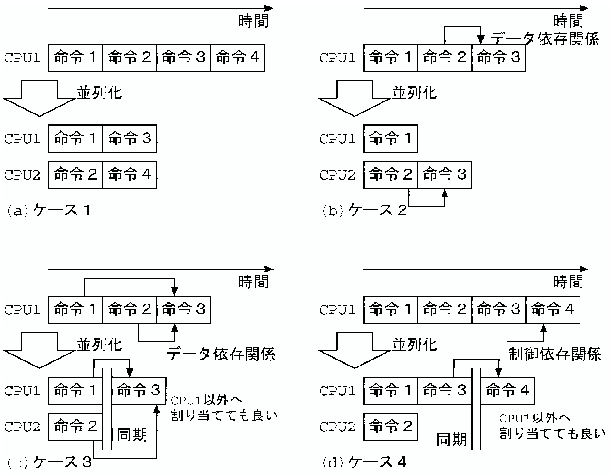

図 1: 複数の CPU へ命令を割り当てる方法

CPU 数が 2 を超える場合も、割り当てようとする命令とデータ依存関係に有る命令が一つの CPU にのみ割り当てられている場合はケース 2、複数の CPU に割り当てられている場合はケース 3 のように命令を割り当てる。

(d) ケース 4 は、制御依存関係が有る場合である。命令 3 が分岐命令である時などにこのケースになる。命令 1 から命令 3 までは依存関係が無いため、各 CPU に順番に割り当てる。しかし命令 3 はその後の命令と制御依存関係が有るため、命令 3 の後は全 CPU 間で同期をとって命令の実行順序を制御する。命令 4 は任意の CPU に割り当てる。

なお、同期を入れた後の命令は、同期を入れる前の命令との依存関係を気にすることなく、任意の CPU に割り当てることができる。しかし、同期処理は逐次実行時には不要な処理であるため、並列処理におけるオーバーヘッドとなる。そのため、並列シミュレータを高速に実行するには、同期回数を増やさずに多くの命令を並列に実行する必要がある。

3 並列シミュレータの実装

3.1 実装環境

本研究では、既存のプロセッサシミュレータを並列化して並列シミュレータを構築する。

ベースとなるシミュレータには、SimCore/Alpha Functional Simulator[4][5](以降 SimCore と略す) を利用する。SimCore は、C++で書かれた Alpha プロセッサの機能レベルシミュレータ (命令単位でシミュレーションを行うシミュレータ) である。高い可読性とシンプルな構造を特徴としているため、並列化に適している。

並列化のための API には、OpenMP[6] を利用する。プロセッサシミュレータに新たな機能を実装する際に

Parallelization of a Processor Simulator with OpenMP Exploiting Instruction Level Parallelism

† Satoshi OHSHIMA

†† Toshikatsu HIDA, Kenji KISE, Takahiro KATAGIRI, Hiroki HONDA, Toshitsugu YUBA

Department of Computer Science,The University of Electro-Communications (†)

Graduate School of Information Systems,The University of Electro-Communications (††)

はコードを変更するため、可読性が高く構造がわかりやすいことが望ましい。OpenMP は、並列化の際に元のプログラムの可読性を維持しやすいという特徴があるため、プロセッサシミュレータを扱うのに適している。

コンパイラに Intel C++ Compiler7.1[7] を利用することで、OpenMP を用いた C++プログラムの並列化を行う。

3.2 実装方法

SimCore は、テキスト形式のファイルをシミュレーション用の実行ファイルとして利用する (図 2-a 参照)。この実行ファイルを利用して、命令間の依存関係の解析と各 CPU への命令の割り当てを行う。命令の割り当て結果をファイルとしてまとめ、これを並列実行用の情報ファイル、もしくは単に情報ファイルと呼ぶことにする。並列にシミュレーションを行う際には、実行ファイルとともに情報ファイルをシミュレータに与える (図 2-b 参照)。情報ファイルには、各命令をどの CPU が実行すれば良いかと、その命令に続いて何命令が同期をとらずに実行できるかという情報を記述する。

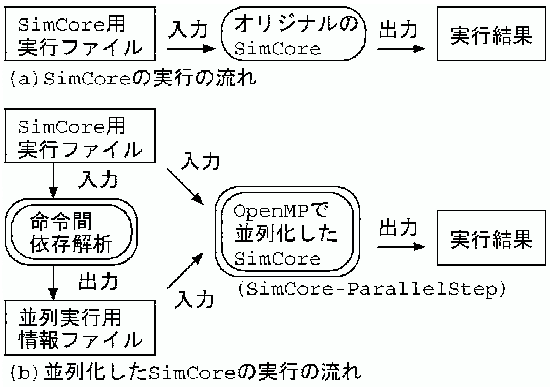

図 2: SimCore の動作におけるモジュール間の関係

SimCore に並列実行用の情報ファイルを読む込む機能と、情報ファイルに従って並列実行する機能を付け加えることで、並列シミュレータを構築する。以降、SimCore にこれまでに述べてきた手法を実装した並列シミュレータを、SimCore-ParallelStep と呼ぶ。

4 SimCore-ParallelStep の評価

提案手法の評価として、並列化に適した構造を持つ単純なシミュレーションを 2CPU で実行し、逐次実行した場合と実行時間を比較する。

ここで行うシミュレーションは、同期をとらずに連続実行できる命令が並んだブロックを、何度もループさせるという単純なものである。各ループの最後に一度同期をとっている。並列実行用の情報ファイルは手作業で作成する。1 ループあたりの命令数を変化させて、同期間隔と実行速度の関係も調べる。ここでいう 1 ループあたりの命令数とは、各 CPU が 1 ループで実行する命令数の合計であり、2CPU で実行する場合は、各 CPU は 1 ループあたりこの半分の命令を実行する。1 ループあたりの命令数が多いほど、同期間隔が広がることになる。

逐次実行の実行時間は、オリジナルの SimCore を利用して測定する。並列実行については、SimCore-ParallelStep を利用して測定する。

実行時間の測定に用いたのは、PentiumIII Xeon 700MHz を 4 個搭載した SMP 型並列計算機である。逐次実行と並列実行の実行時間の比を図 3 に示す。同期間の命令数毎に、逐次実行時の実行時間を 1 とした並列実行時の実行時間の割合を表している。

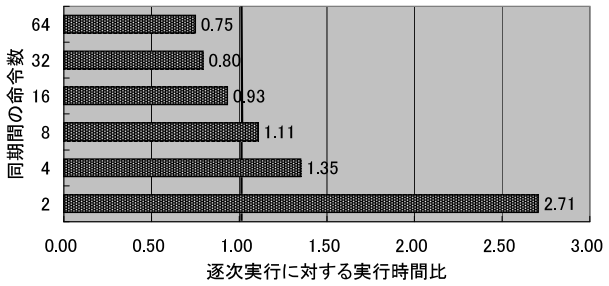

図 3: 逐次実行時間に対する SimCore-ParallelStep の実行時間比

図 3 を見てわかるように、同期間の命令数が少ない場合は逐次実行の方が高速である。例えば同期間の命令数が 2 命令のプログラムを並列に実行すると、実行時間は 2.71 倍に増えてしまう。これは、各 CPU がシミュレーションする命令数に対して、CPU 間の同期の回数が多すぎるためと推測できる。同期間の命令数が増えると同期の頻度が低下して実行時間が短くなり、この実験では 8 命令から 16 命令の段階で、並列実行の実行時間が逐次実行を下回る。つまり、プログラム全体における平均の同期間隔がある程度大きくなれば、2CPU の並列実行で逐次実行よりも高速にシミュレーションを行えることがわかる。

5 おわりに

命令レベル並列性を利用した、プロセッサシミュレータの並列化手法を提案した。また、実際に機能レベルプロセッサシミュレータの並列化を行い、高速化の可能性を示した。

今回は実行時間の測定に並列実行しやすいプログラムを用いたが、今後は現実のアプリケーションに近いプログラムを用いて測定を行っていく。その際、今回の実験では並列実行用の情報ファイルを手作業で作成したが、実際のアプリケーションに対して手作業で情報ファイルを作成することは現実的ではない。今後、情報ファイルを自動生成するプログラムの作成を進め、本研究の提案する手法の有効性を検証することを考えている。また、使用する CPU の数を増やしたり、SMP 型並列計算機以外の環境へ提案手法を適用することも検討する。CPU へ命令を割り当てる方法を改良したり、実行環境に合わせて割り当て方を変えることも考える。

参考文献

[1] 中島浩、ハイエンド・コンピュータ研究のためのシミュレーション技術、ハイエンドコンピューティング技術に関する調査資料 II(http://.www.icot.or.jp/)、2001 年 3 月

[2] 高崎透, 中田尚, 中島浩、高性能マイクロプロセッサシミュレータの並列化による高速化の構想、情報処理学会研究報告 2003-ARC-155, pp.45-50、2003 年 11 月

[3] マイク・ジョンソン、スーパースカラ・プロセッサ、日経 BP 出版センター、1994

[4] Kenji Kise, Hiroki Honda, and Toshitsugu Yuba、SimAlpha Version 1.0: Simpleand Readable Alpha Processor Simulator、in The Eighth Asia-Pacific Computer Systems Architecture Conference (ACSAC'2003)、September 2003

[5] SimCore/Alpha Functional Simulator、http://www.yuba.is.uec.ac.jp/~kis/

[6] OpenMP、http://www.openmp.org/

[7] Intel C++ Compiler、http://www.intel.co.jp/