4J-3

マルチスレッドコードにおけるインライン展開適用の手法

増澤 英樹　阿久津 徳寿　大津 金光　横田 隆史　馬場 敬信†
宇都宮大学工学部情報工学科‡

1　はじめに

シングルスレッドコード実行においての計算機の速度向上には限界が存在する。また、ソースコードは必ずしも入手可能とは限らない。そこで、我々はシングルスレッドコードをバイナリレベルでマルチスレッド化および最適化するシステムを構築している。

このシステムにおいてより高い実行性能を持つマルチスレッドコードを生成するためには、効果的なスレッドコード最適化技術[4]が必要となる。

スレッドコード最適化技術のひとつであるインライン展開は、関数呼び出しのコスト削減および命令畳込み、コードの書き換えなどの他の最適化の適用範囲拡大につながり、実行速度向上が期待できることでよく知られている手法である。

本研究では、マルチスレッドコード向けのループにおけるインライン展開手法を検討する。さらに、実際のプログラムに適用することで、本手法の有効性についての評価を行う。

2　マルチスレッド化とインライン展開

我々が研究しているバイナリレベルでマルチスレッド化および最適化するシステムにおいて、マルチスレッド化は主に速度向上の可能性が高いループ構造を対象としている。マルチスレッド化をループ単位で行う際、ループイテレーション間依存データの情報が必要となる。ループイテレーション間依存がある場合には、計算順序を守るためにスレッド間で同期を行う必要がある。

マルチスレッド化を行う際、対象となるループ中に関数呼び出し命令があり、呼び出し先の関数に依存がある場合には、インライン展開されたコードを変更し同期を行うほうが効果的である。例えば、図1の(b)のようにマルチスレッド化を行う際、インライン展開を行わないで関数Bのコードを変更し関数B'としてしまうと、関数Cから呼ばれた時に計算結果が正しくなくなるため関数Bのコードを変更できない。図1の(c)のようにループ中の関数呼び出し命令の前後に同期を行うようコードを変更すると、依存箇所でない部分も同期待ちを行わなければならなくなるので無駄になる。従って、図1の(d)のようにインライン展開を行い、展開されたコードに対して変更をすれば、同期待ち時間は最少になり最適なマルチスレッドコードになる。

また、ループにおけるインライン展開は、シングルスレッド実行においてもループのイテレーション数×インライン展開による命令数の削減により大幅な速度向上が期待できる。

実際に、ループにおけるインライン展開手法を高級言語のFORTRANでは、KAPコンパイラ[3]の最適化

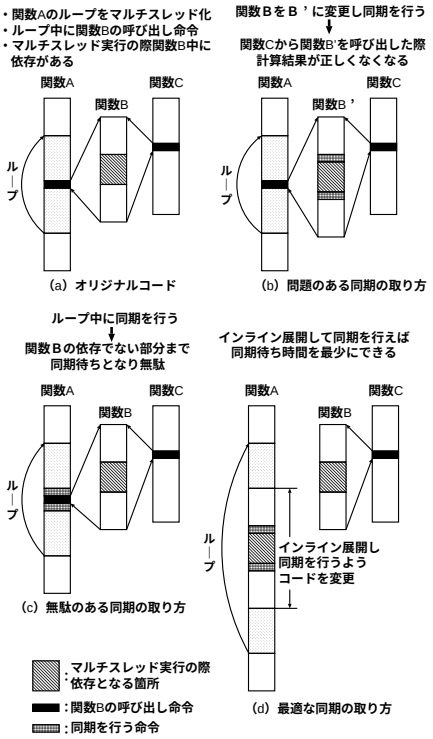

図 1. マルチスレッドコードにおける同期の取り方

オプションの一つとして実現しているが、本研究では、この手法を参考にしてバイナリレベルにおいてマルチスレッド化を考慮しつつ、一般に速度向上をするインライン展開手法を検討する。

3　本研究でのインライン展開手法

本研究でのインライン展開手法は、マルチスレッド化を前提としているため、ループ構造の中にある関数呼び出し命令をインライン展開する。

インライン展開をした結果、コードサイズが大きくなり過ぎるとキャッシュミスを起こし、速度低下する可能性がある。よって、図2のようにループ部分のコードサイズとインライン展開する関数のコードサイズの和がキャッシュのサイズ以下になるようにし、ループ中にキャッシュミスを起こさないようにする。

ループイテレーション間依存が多く、マルチスレッド化しても速度向上が見込めないループや、ループのイテレーション数が少なくてマルチスレッド化ができないループでは、シングルスレッド実行での速度向上に期待する。

4　評価

4.1　評価環境

評価環境は、SimpleScalarをベースとしたスレッドパイプライニングモデルシミュレータであるSIMCA[2]を用いた。評価対象アプリケーションとし

Inline expansion method for multithreadcode
† Hideki Masuzawa, Noritoshi Akutsu, Kanemitsu Ootu, Takashi Yokota and Takanobu Baba
‡ Department of Information Science, Faculty of Engineering, Utsunomiya University

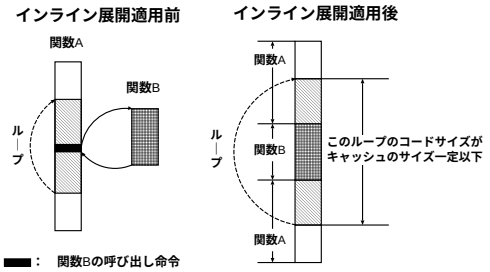

図 2. 本研究におけるインライン展開手法

て、SPECfp2000 の 168.wupwise と 183.equake を用いる。入力データセットには、168.wupwise については test を、183.equake については test、train を使用する。

評価対象である SPECfp2000 のプログラムは FORTRAN で記述されているので、f2c（Fortran to C トランスレータ）を用いて C コードへ変換し、SIMCA 用 gcc クロスコンパイラ（最適化オプション-O3）を用いてバイナリコードを生成する。

このバイナリコードに対して本手法のインライン展開を適用する。インライン展開の対象としたのは 168.wupwise の関数 zgemm_() と関数 zdotc_()、183.equake の関数 main() に含まれるそれぞれのループである。今回の対象アプリケーションは、168.wupwise においてはループのイテレーション数が少ないためマルチスレッド化ができず、183.equake においてはループイテレーション間依存による同期待ちの時間が長いためマルチスレッド化による速度向上が見込めないためシングルスレッド実行による評価となった。しかし、これらのループを選んだのは、ループの全イテレーション数が多いのでインライン展開すれば大幅なコスト削減につながり有意な速度向上を得られると考えたためである。また、本研究のインライン展開手法を評価する際、シングルスレッド実行でも検討可能と考えたためである。

評価は SIMCA 用 gcc クロスコンパイラ（最適化オプション-O3）を用いて得られたコードから本手法のインライン展開されたコードの速度向上率を求めることで行う。速度向上率は以下の式によって求められる値とする。

$$\text{速度向上率} = \frac{\text{本手法適用前の実行サイクル数}}{\text{本手法適用後の実行サイクル数}} \quad (1)$$

4.2 評価結果

図 3 に対象アプリケーション 168.wupwise の関数 zgemm_()、関数 zdoct_()、および 183.equake の関数 main() の入力データセット test、train のループにおける速度向上率を示している。グラフの横軸は対象アプリケーションを示し、縦軸は SIMCA 用 gcc クロスコンパイラ（最適化オプション- O3）を用いて得られたコードの実行速度を 1 としたときの本手法のインライン展開されたコードの速度向上率を示している。

本手法のインライン展開により、168.wupwise の関数 zgemm_() に含まれるループでは 1.20 倍、関数 zdotc_() に含まれるループでは 1.04 倍、183.equake の関数 main() に含まれるループに入力データセットが test の場合は 1.08 倍、入力データセットが train の場合は

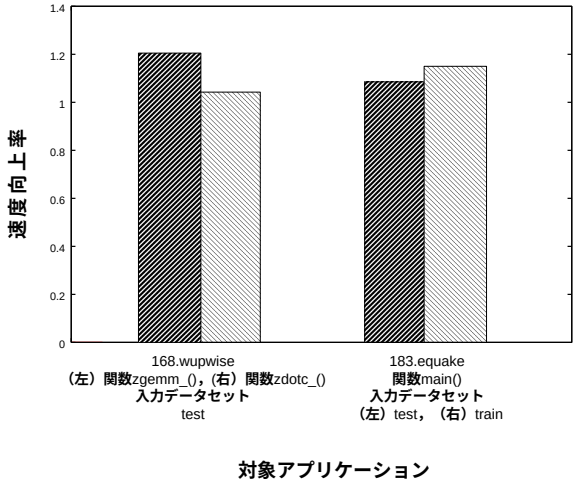

図 3. 対象アプリケーションの速度向上率

1.15 倍の速度向上を得た。これらの結果により、本稿で示したループにおけるインライン展開手法により最大 20 %と有意な速度向上を得られることが確認できた。

5 おわりに

本稿ではループにおけるインライン展開に着目し、バイナリレベルにおいての本手法の有効性の検討を行った。

今回の評価対象アプリケーションではループのイテレーション数や、依存の関係によりマルチスレッド化を行えなかったが、シングルスレッド実行において本手法が有効であることが分かった。マルチスレッド実行においても本手法のインライン展開を適用すれば、コストの削減につながるはずなので速度向上すると考えられる。このことから、本手法は、シングルスレッドコードおよびマルチスレッドコードにおいて有効であると考えられる。

今後の課題として、マルチスレッド化できるループに本手法を適用し、より多くのアプリケーションで検討を行うことが挙げられる。

謝辞　本研究は、一部日本学術振興会科学研究費補助金（基盤研究 (B)14380135、同 (C)16500023、若手研究 (B)17700047）の援助による。

参考文献

[1] 大津 金光、小野　喬史、横田 隆史、馬場 敬信、“バイナリレベルマルチスレッド化コード生成手法とその評価、” 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム、Vol.44、 No.SIG-1 (HPS 6)、 pp.70-80、 2003．

[2] J. Huang，“The SImulator for Multithreaded Computer Architecture (Release 1.2)，” http://www.cs.umn.edu/Research/Agassiz/Tools/SIMCA/simca.html.

[3] KAP/Pro Toolset，http://developer.intel.com/software/products/kappro/

[4] 阿久津　徳寿、大津 金光、横田 隆史、馬場 敬信、“マルチスレッドコード向け命令レベル最適化ツールの開発、” 情報処理学会第 67 回全国大会 5ZB-6、2005.