1J-01

ループ終了判定命令を考慮したSoftware Pipeliningの多重ループへの適応

畠山　雄樹†　　宮内　新‡　　中野　秀洋‡
†東京都市大学大学院工学研究科　　‡情報工学専攻

1　はじめに

Software Pipelining[1] はループの命令レベル並列性を向上させる手法の一つである．基本的な Software Pipelining はループ回数が未知の場合や少数の場合には適用できず，また，多重ループで効果が減衰するという問題がある．前者の問題にはループの終了判定命令を考慮 [2] して Software Pipelining を行うことで，後者の問題には Prologue と Epilogue を Overlap して実行 [3] することで解決が図られている．しかし後者の手法をそのまま前者の手法に適用することはできず，コンパイラにも実装されていないため，適用する場合は手作業で行う必要がある。

2　提案手法

本手法では Prologue や Overlap 部にも外部ループの終了判定命令等を追加することを提案する．その際に Overlap 部の作り方の変更を行う．本手法を適用することでループ終了条件を考慮した手法 [2] においても多重ループの最適化が可能になる．提案手法の概要を図 1 に示す．左側が提案手法の適用前であり，右側が適用後である．それぞれ同じ名前のブロックが左と右で対応している．名前の違う Overlap と Kernel0 と Kernel1 は左の Kernel と内容が対応している．[3] と比較した際の提案手法における命令の変更や追加を行った命令が存在するブロックは太枠で示している．

2.1　命令追加・変更

Overlap 部の作り方の変更

従来手法 [3] では Overlap ブロックは *S-1* 回の Kernel と同等の命令列で構成されている．ここで *S* は Software Pipelining の Stage 数を表す．本手法ではループ回数が少ない場合にも対応するために，Overlap 部の作り方を変更する．Overlap ブロックは 1 つの Kernel の命令で構成し，このブロックを *S-1* 個確保する．

外ループ終了判定命令等の追加

Prologue 部と Overlap 部の下に外ループの終了判定命令等と Jump 命令を追加する．*N-1* 個の外ループの終了判定命令等を追加した後，Jump 命令を追加する．ここで *N* は最適化するループの多重度を表す．最大値はループの深さである．これを *S-1* 回繰り返し全ての Prologue・Overlap ブロックに適用する．

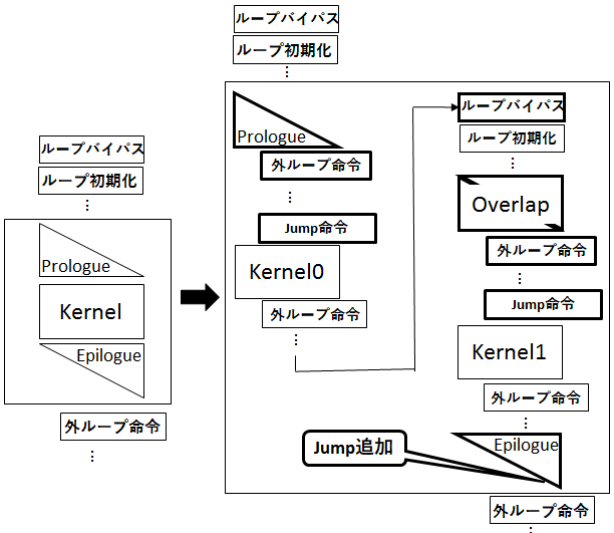

図 1: 提案手法の概要

ループバイパス命令の追加

従来手法 [3] では最内ループが 0 回となることを想定していない．本手法では最内ループが 0 回の時にも対応するため，最初の Overlap ブロックの上部にループバイパス命令を *N-1* 個追加する．分岐先は Kernel1 の下の外ループ命令ブロックである．

Epilogue への Jump 命令の追加

最後の Epilogue ブロックの末尾に Jump 命令を追加する．Jump 先は最後の Epilogue 命令の *N* 個先のブロックである．これは本来ある外ループ命令等が存在するブロックを実行しないためである．

2.2　動作

内ループ継続時

内ループが続く場合の処理の流れを図 2 に示す．Prologue の場合は次の Prologue，もしくは Kernel0 に分岐

†Adaptation of Software Pipelining considering the loop control instruction to nested loops
†Yuki Hatakeyama　‡Arata Miyauchi　‡Hidehiro Nakano
†Computer Engineering, Tokyo City University

する．Overlap の場合は次の Overlap，または Kernel1 に分岐する．

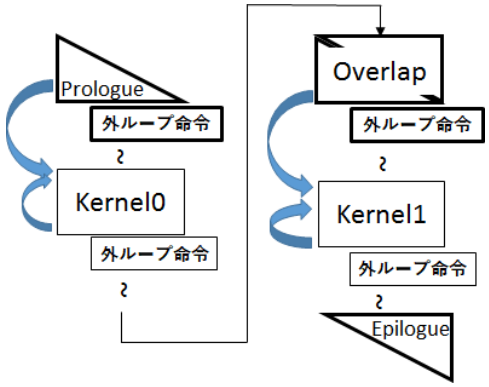

図 2: 内ループ継続時の分岐

内ループ終了・外ループ継続時

内ループが終了した場合は外ループ命令ブロックに遷移する．その流れを図 3 に示す．外ループが継続中の場合はループバイパス命令が存在するブロックに分岐する．

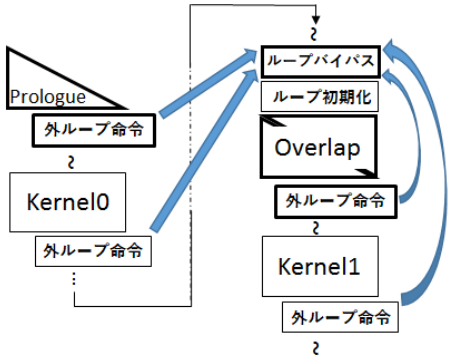

図 3: 内ループ終了・外ループ継続の分岐

内ループ終了・外ループ終了時

外ループがすべて終了した際には図 1 の”Jump 命令”のブロック，もしくは Kernel0 と Kernel1 の下にある外ループ命令の最後のブロックから，Epilogue の最初のブロックに遷移する．

3 実装

COINS[4] へ実装を行った．Epilogue の下にある *N-1* 個のブロックを外ループの終了判定命令等が存在するブロックとみなし，これらをコピー・変更し各 Prologue，Overlap ブロックの下に追加するようにした．ループバイパス命令，ループ初期化命令は Prologue の上のブロックを *2(N-1)* 個コピー・変更し最初の Overlap ブロックの上に追加した．

4 評価

本手法の有効性を確かめるため従来手法 [2] との比較を行った．対象としたプログラムを図 4 に示す．COINS の最適化コマンドは-O3 を使用した．プログラムのループ回数は全て 10 とし，100 万回関数を呼び出し，その実行時間を測定した．測定は 100 回試行し，その平均値を評価した．計測には clock() 関数を使用した．実験環境は core i7-5500 2.40GHz の CPU，Ubuntu 14.04 の OS 64bit，8GB の RAM である．また，提案手法は内側の 2 重ループに適用した．

```
float prod(float a[][A], float b[][A], int n, int m, int l){
  int i,j,k; float s = 0.0;
    for(i = 0;i<n; i++){
      for(j = 0; j<m; j++){
        for(k = 0; k<l; k++)
        s +=a[i][k]*b[k][j];
      }
    }
  return s;
}
```

図 4: ベンチマークプログラム

実験結果は従来手法 [2] が 1.030[s]，提案手法が 1.005[s] となった．値を比較したところ，実行効率が約 2.5 %向上した．これは Overlap 部を実行することで Prologue 部と Epilogue 部を実行するより命令レベル並列性が増したため，実行効率が向上したと考えられる．

5 結論

本手法は前者の手法の多重ループへの最適化手法を提案し，従来の手法と比べ実行効率の向上が見られた．今回提案したものは Prologue ブロックをすべて実行する前提であるため，今後の課題として，途中で Prologue をすべて処理せずに抜けた場合の対応や，命令単位でのスケジューリング，依存関係の検出があげられる．

参考文献

[1] Lam, M.S. ”Software Pipelining:AnEffective Schduling Technique for VLIW Machines”, PLDI’88, pp.318-328 (1988)

[2] 中田育夫 ”ソフトウェア・パイプライニングの一実現法”, 情報処理学会論文誌:プログラミング, Vol 47 No.SIG 16(PRO 31) (2006)

[3] 中田育夫 ”ソフトウェア・パイプライニングにおける多重ループの最適化”, 並列処理シンポジウム JSPP ’95， P185, 平成 7 年 5 月 (1995)

[4] ”コンパイラの基盤技術と実践”，朝倉書店，2008