階層統合型粗粒度タスク並列処理における再帰メソッドの並列Javaコード生成

Parallel Java Code Generation of Recursive Methods in Layer-Unified Coarse Grain Task Parallel Processing

小澤 智弘† 吉田 明正†
Tomohiro Ozawa Akimasa Yoshida

1 はじめに

本稿では，マルチコアプロセッサを対象とした階層統合型粗粒度タスク並列処理のための並列Javaコード生成手法を提案する．階層統合型粗粒度タスク並列処理では，階層ごとにタスク間並列性を抽出した後，ダイナミックスケジューラが全階層のタスクをプロセッサコアに割り当て，階層を越えたタスク間並列性を利用することが可能である．Javaプログラムの並列処理については，ループ並列化コンパイラ [1]，HPFのような配列分散を取り入れたHPJava[2]，ランタイムサポートによりスレッド間並列性を利用するzJava[3]，トランザクションメモリ上でのトレースベースの並列処理 [4] が提案されているが，複数階層の粗粒度タスク間の並列性を利用することは困難である．そこで，本稿では粗粒度タスク並列処理を実現するJavaコードを自動生成する方法を提案する．再帰メソッドを含む指示文付Javaプログラムに対して提案手法を用いた並列化コンパイラは実装されており，Sun Fire T1000上での性能評価により有効性が確かめられた．

2 階層統合型粗粒度タスク並列処理

階層統合型粗粒度タスク並列処理では，まず，粗粒度タスク並列処理手法 [5] で用いられている並列性抽出技術を用いて，階層型マクロタスクグラフ（MTG）を生成する．次に，階層開始マクロタスク [6] を導入し，全階層のマクロタスクを統一的にコア（あるいはプロセッサ）に割り当てるダイナミックスケジューリングルーチンを生成する [6, 7]．例えば，図1のJavaプログラムは，図2の階層型マクロタスクグラフに変換される．このプログラムを4コアのプロセッサ上で実行したイメージは図3のようになり，全階層のマクロタスク間並列性が最大限に利用される．

2.1 マクロタスクと階層開始マクロタスク

粗粒度タスク並列処理による実行では，プログラムを階層的に，基本ブロック，繰り返しブロック（for文等のループ），あるいは，サブルーチンブロック（メソッド呼び出し）の3種類のマクロタスク（MT）に分割する [6]．例えば，図1のJavaプログラムを階層的にマクロタスクに分割し，制御依存とデータ依存を考慮した最早実行可能条件を解析することにより，図2のような階層型マクロタスクグラフ（MTG）を生成できる．

次に，階層統合型実行制御 [6] を適用する場合，全階層のマクロタスクを統一的に取り扱うため，階層開始マクロタスクを導入する．階層開始マクロタスクの導入により，当該階層のマクロタスクの実行が可能になったことが保証され，全階層のマクロタスクを同時に取り扱うことが可能となる．例えば，図2の繰返し文のMT1_2の場合，内部に第2階層MT（MT2_1，MT2_2）を含んでおり，MT1_2は第2階層用の階層開始マクロタスクMT1_2Sとして扱われる．

2.2 階層統合型実行制御の最早実行可能条件

制御依存とデータ依存を考慮したマクロタスク間並列性を最大限に引き出すために，各マクロタスクの最早実行可能条件 [5] を解析する．例えば，図2のMT1_3の最早実行可能条件は，1_1 ∧ 1_2と求められ，MT1_3はMT1_1とMT1_2の実行終了後に実行可能となることを表している．各マクロタスクの最早実行可能条件は，図2のような階層型マクロタスクグラフ（MTG）[5] によって表すことが可能である．

```
class Other {
  public static void
        func(int n) {
    /*mt*/{
    if(n<=1) { //MT3_1:IF文
      /*mt*/{ MT3_2の処理; }
    }else{
      /*mt*/{ MT3_3の処理; }
      /*mt inner*/{
      func(n-1); //MT3_4
      }

      /*mt*/{ MT3_5の処理; }
      /*mt inner*/{
      func(n-2); //MT3_6
      }
    }
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    /*mt inner*/ {
    Other.func(2); //MT1_1:メソッド呼び出し
    }
    /*mt inner*/ {
    for (int i=0; i<2; i++) { //MT1_2:for文
      /*mt*/ {
      MT2_1の処理;
      }
      /*mt*/ {
      MT2_2の処理;
      }
    }
    /*mt*/ {
    MT1_3の処理;
    }
  }
}
```

図1 再帰メソッドを含む指示文付Javaプログラム．

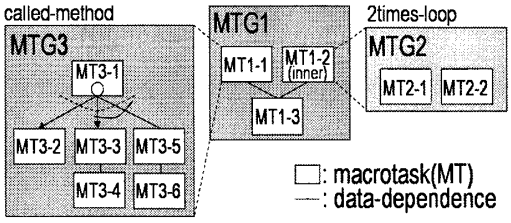

図2 階層型マクロタスクグラフ（MTG）．

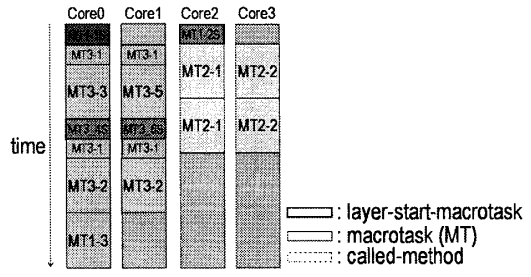

図3 階層統合型粗粒度タスク並列処理の実行イメージ．

3 コンパイラによる並列Javaコード生成

本節では，繰返し文や再帰メソッド呼び出しを含むJavaプログラムに対して，マルチスレッドを伴う並列Javaコードを生成する方法を提案する．

†東邦大学理学部情報科学科
Department of Information Science, Toho University

3.1 並列化指示文

本手法では，対象となるJavaプログラムにおいて，階層統合型粗粒度タスク並列処理を実現する部分に並列化指示文を記述する．マクロタスクは階層的に定義することが可能であり，for文やwhile文等の繰返し文内部や，メソッド内部においても，並列化指示文（/*mt*/）を入れ子にすることにより記述できる．この場合，上位マクロタスクにおいては，/*mt inner*/のようにinnerを並列化指示文に加える．図1のプログラムは，本コンパイラの並列化指示文を加えたソースプログラムの一例である．

3.2 スケジューラを含む並列Javaコードの構成

図1のプログラムは，開発した並列化コンパイラに入力すると，階層統合型粗粒度タスク並列処理の並列Javaコード[7]が生成される．各スレッドコードでは，コア（あるいはプロセッサ）上でマクロタスクの処理を終える度に，スケジューリング処理部でスケジューリングを行い，自コアに新たに割り当てられたマクロタスクの処理を行う．なお，レディマクロタスクキューのアクセスに対しては排他制御を行う．

並列Javaコードは，変数および引数の情報を管理する変数情報管理クラス，階層統合型ダイナミックスケジューリングの共通データのためのDataクラス，ユーザ定義クラスとメソッドのためのOtherクラス，並列Javaコードのmain()メソッドを含むMain_pクラスから構成される．Main_pクラスにおいて，内部クラスのSchedulerクラスが定義されており，scheduler()メソッドが呼び出される．scheduler()メソッドでは，レディMTキューからMTを取り出して実行し，新たなレディMTをレディMTキューに投入する処理を，EndMTが終了するまで繰り返す．

3.3 再帰メソッドに対応する並列Javaコード

再帰呼び出しを含むメソッドでは，そのメソッド内に定義されたマクロタスクを動的に生成し，管理する必要がある．そこで本手法では，変数情報管理クラスを用いて，メソッドの引数，ローカル変数を管理している．変数情報管理クラスには呼び出すメソッドの引数，ローカル変数の情報を事前に登録しておく．

再帰メソッド内のマクロタスクは，再帰の深さ(レベル)毎に，その最早実行可能条件を管理しており，異なる再帰レベルに属するマクロタスクが並列に実行される場合にも対応している．再帰メソッド内のマクロタスクの最早実行可能条件は，通常の階層レベルに加えて再帰レベルも考慮して扱われる．

再帰メソッドを呼び出す際に，動的に変数情報管理インスタンスを生成して識別子を付加する．生成後に変数情報管理インスタンスに引数のコピーを行う．そして再帰メソッド上で使われる変数は変数情報管理インスタンスを介して操作を行う．

3.4 並列化コンパイラの実装

本節では，3.2節の並列Javaコードを生成する並列化コンパイラについて述べる．開発したコンパイラはJava言語を用いて開発されており，構文解析においては，LALR(1)のコンパイラコンパイラであるJay/JFlexを用いており，抽象構文木を作成する．その後，並列化指示文の情報を用いて，ダイナミックスケジューリングコードを伴う並列Javaコードを生成する．この時，メソッド内のマクロタスク間のデータ依存解析は自動的に行われており，最早実行可能条件も生成される．本コンパイラの対象となる入力Javaプログラムは，現段階で，JDK1.2の文法で記述できるものとする．クラスメソッド内部では階層統合型粗粒度タスク並列処理が可能であるが，インスタンスメソッド内部は1つのマクロタスクとして扱われる．

4 マルチコアプロセッサT1000上での性能評価

本性能評価では，マルチコアプロセッサUltraSPARC T1をベースに構築されたSun Fire T1000を用いた．T1000は，UltraSPARC T1プロセッサ（8core, 1.0GHz），8GBのメモリを備えており，OSはSolaris 10，JavaコンパイラはJDK1.6となっている．

本性能評価では，図1のJavaプログラムを用いる．それぞれのMTは一定の負荷がかかるダミー処理を行っている．また，性能評価では図1のOther.func(2)をOther.func(10)に変更し，MT1_2のループ回数を10回としている．単一のデータ依存エッジのみで接続されたマクロタスクは1つのマクロタスクとした．このJavaプログラムに，3.1節の並列化指示文を加え，開発した並列化コンパイラを用いて，並列Javaコードを生成する．その後，並列Javaコードを JDK1.6のjavacでコンパイルし，マルチコアプロセッサSun Fire T1000のJVMで実行した．JVMでは-Xintオプションをつけ，JITコンパイルは適用していない．

実行結果は，4コアを用いた場合に3.78倍，8コアを用いた場合に6.79倍の速度向上が得られた．それゆえ，本コンパイラは，再帰メソッドを含むJavaプログラムに対して，階層統合型粗粒度タスク並列処理の並列Javaコードを効率よく生成しており，並列Javaコード生成手法の有効性が確かめられた．

5 おわりに

本稿では，階層統合型粗粒度タスク並列処理の並列Javaコード生成手法を提案した．本手法では，再帰メソッドを含むプログラムに並列化指示文を挿入し，開発した並列化コンパイラを用いることにより並列Javaコードを自動生成することが可能である．

並列コンパイラにより生成された並列Javaコードを，Sun Fire T1000上で実行したところ，再帰メソッドを含むプログラムにおいて高い実効性能を達成することができ，並列Javaコード生成が効果的に行われていることがわかる．

参考文献

[1] A.J.C. Bik, D.B. Gannon. Javar a prototype Java restructuring compiler. *Concurrency: Practice and Experience*, Vol. 9, No. 11, 1997.

[2] S.B Lim, H. Lee, B. Carpenter, G. Fox. Runtime support for scalable programming in Java. *J. Supercomputing*, 43, pp.165-182, 2008.

[3] B. Chan, T.S. Abdelrahman. Run-Time Support for the Automatic Parallelization of Java Programs. *J. Supercomputing*, 28, pp.91-117, 2004.

[4] B.J. Bradel, T.S. Abdelrahman. The Use of Hardware Transactional Memory for the Trace-Based Parallelization of Recursive Java Programs. *Proc. PPPJ'09*, pp.27-28, 2009.

[5] 笠原博徳, 小幡元樹, 石坂一久. 共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理. 情報処理学会論文誌, Vol. 42, No. 4, 2001.

[6] 吉田明正. 粗粒度タスク並列処理のための階層統合型実行制御手法. 情報処理学会論文誌, Vol. 45, No. 12, 2004.

[7] 吉田明正, 小澤智弘. 階層統合型粗粒度タスク並列処理のための並列Javaコード生成手法 . 情報処理学会研究報告, 2008-HPC-116-32, 2008.