B-002

階層統合型粗粒度タスク並列処理における再帰プログラムの並列 Java コード生成

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

遠藤 佑太† 吉田 明正†
Yuta Endo 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，再帰メソッドを伴う 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を実現する並列 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 言語の再帰プログラムの並列処理に関する研究は，トレース間並列処理 [1, 2] が提案されているが，ハードウェアトランザクションメモリ環境を前提としている．

それに対して本稿では，マルチコアプロセッサ上でダイナミックスケジューリングを伴い，再帰レベルを考慮した階層統合型粗粒度タスク並列処理の Java コード生成手法を提案する．再帰呼び出しを伴うマージソートのプログラムを用いて，Sun Fire T1000 上で性能評価を行っている．

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 [3] で用いられている並列性抽出技術を利用し，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（MT）[4] を導入し，全階層の MT を統一的にコアに割り当てるダイナミックスケジューリングルーチン [3, 4] を生成する．このようなダイナミックスケジューリング方式を採用することにより，例えば図 1 で表現されるプログラムの場合，4 コアのプロセッサ上で実行したイメージは図 2 のようになり，全階層の MT 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），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）[3] に分割する．図 1 は，Java プログラムを階層的に MT 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解析により生成される階層型マクロタスクグラフ（MTG）である．

次に，階層統合型実行制御 [4] を適用する場合，全階層の MT を統一的に取り扱うため，階層開始 MT を導入する．階層開始 MT の導入により，当該階層の MT の実行が可能になったことが保証され，全階層の MT を同時に取り扱うことが可能となる．図 1 の MT1_2 や MT2_1 は階層開始 MT 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 MT 間並列性を最大限に引き出すために，各 MT の最早実行可能条件 [3] を解析する．例えば，図 1 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〜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 MT の最早実行可能条件は，図 1 のような階層型マクロタスクグラフ（MTG）[3] によって表すことが可能である．

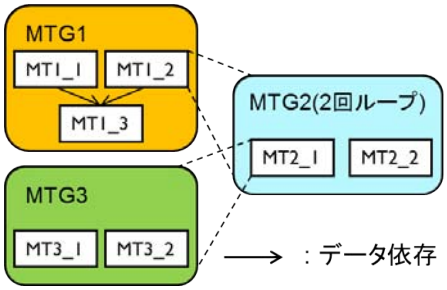

図 1 階層型マクロタスクグラフ（MTG）.

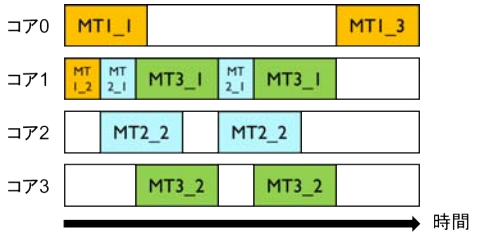

図 2 階層統合型粗粒度タスク並列処理の実行イメージ.

3 再帰プログラムの並列 Java コード生成

本章では，再帰呼び出しを伴う階層統合型粗粒度タスク並列処理を実現する並列 Java コードについて述べる．並列 Java コードは，本研究室で開発している並列化コンパイラ [5] により自動生成可能であるが，本稿で提案する並列 Java コードについては，現在並列化コンパイラに実装中である．この並列 Java コードは，Java のマルチスレッドにより記述されており，通常の Java コンパイラでコンパイルすることにより，JVM 上で並列実行できる．

3.1 スケジューラを含む並列 Java コードの構成

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 Func 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．この際，各 MT の最早実行可能条件の判定は，eeccheck() メソッドにより行う．各 MT のコードは，Main_p クラス及びユーザ定義クラスである Other クラ

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

スにクラスメソッドとして実装される．

3.2 再帰呼び出しの並列 Java コード

再帰メソッドを複数の箇所で同時に呼び出す可能性を考慮し，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．

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，同時に動的に生成される識別子を付加する．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．返却値が存在する場合には変数情報管理インスタンスに保存した値を返却する．この一連の動作は，再帰呼び出しされる MTG（メソッド本体）及びその内部 MTG でのみ行われる．

再帰メソッドの呼び出し側と呼び出される側の管理を，変数情報管理インスタンスに付加された動的生成識別子を MT の番号として行う．再帰メソッド内の MT の最早実行可能条件は，通常の階層レベルに加えてこの動的生成識別子も考慮して扱われる．これによりマルチスレッド上において適切に再帰呼び出しを行うことが可能となる．

3.3 再帰レベルを考慮した並列 Java コード

再帰呼び出しを行う際，最初に再帰呼び出しを行う MT の再帰レベルを 0 とする．呼び出された MT の再帰レベルは 1 となり，以降の MT の再帰レベルは呼び出し側の再帰レベルに 1 を足したものになる．

本手法では，再帰メソッドを呼び出す際，同時に再帰レベルとなる引数を渡す．再帰呼び出し時に一定のレベルに達している場合は，その再帰メソッド内の MT を実行せず，変数情報管理インスタンスの生成も行わないようにしている．これにより，並列性を十分満たした場合には，その再帰レベル以上の再帰呼び出しは 1 コア上で実行することにより，ダイナミックスケジューリングのオーバーヘッドを軽減している．

3.4 マージソートプログラムの並列処理

本稿では再帰プログラムの例としてマージソート [6] を取り上げる．図 3 はその MTG である．図 3 の MTG において，メインメソッドの MT1 はユーザ定義クラス Other の階層開始 MT を呼び出している．この時点での再帰レベルは 0 である．

この階層開始 MT により再帰 MT の MT0 の開始条件が満たされる．MT0 は与えられたデータ列の要素が 2 つ以上の場合，マージソートを開始する処理を行う．MT1 でデータ列を二分割し，MT2 と MT3 でそれぞれのデータ列のマージソートを行うために再帰呼び出しを行う．呼び出した再帰 MT の再帰レベルは，現在の再帰レベルに 1 を足したものとなる．MT4 から MT7 にかけてデータ列の分割とソートを行う．MT8 の終了通知後，上位レベルから分割した同レベルのデータ列とマージし，呼び出し元の MT4 や MT5 の処理に戻る．全てのデータ列のマージソートが完了することでメインメソッドの MT2 が実行され，MT2 が終了通知を出すことでプログラムは終了となる．

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．

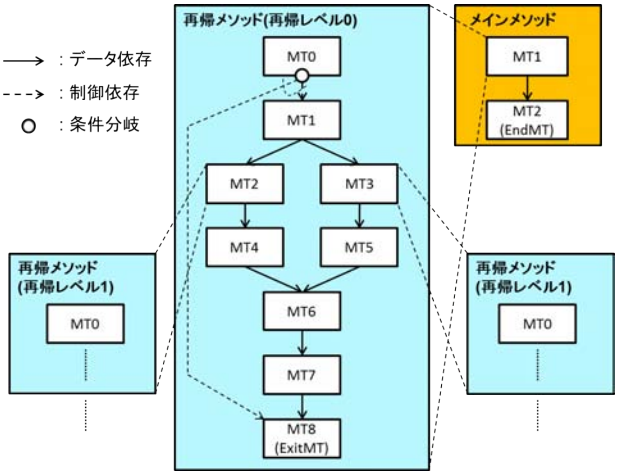

図 3 マージソートプログラムの MTG.

本性能評価では，3.4 節のマージソートプログラムを用いる．マージソートのデータ数は 100 万個とし，実行する再帰メソッドの MT の再帰レベルは最大 3 とする．なお，粗粒度タスク並列性を高めるため，MT4，MT5，MT7 を 8 つの MT に分割したコードに対して並列 Java コードを生成する．この並列 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，1 コアと比べて 2 コアを用いた場合に 1.92 倍，4 コアを用いた場合に 3.94 倍，8 コアを用いた場合に 6.83 倍の速度向上が得られた．これにより，再帰メソッドを伴う Java プログラムに階層統合型粗粒度タスク並列処理を適用することにより，高い実行性能を達成できることが確かめられた．

5 おわりに

本稿では，再帰メソッドを伴う 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を実現する並列 Java コードの生成手法を提案した．本手法では，再帰呼び出しをする度に生成される MT に動的生成識別子を付加し，動的に生成される変数情報管理クラスと対応させることで効率よく並列処理を行うことが可能である．

マルチコアプロセッサ Sun Fire T1000 上でマージソートプログラムを実行したところ，高い実効性能を達成することができ，階層統合型粗粒度タスク並列処理が効果的に実現されていることがわかる．

参考文献

[1] Borys J. Bradel, Tarek S. Abdelrahman. A study of potential parallelism among traces in Java programs. *Science of Computer Programming*, 74, pp.296-313, 2009.

[2] Borys J. Bradel, Tarek S. Abdelrahman. The Use of Hardware Transactional Memory for the Trace-Based Parallelization of Recursive Java Programs. *PPPJ '09*, August 27-28, 2009.

[3] 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.

[4] 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.

[5] 吉田明正, 小澤智弘. マルチコアプロセッサ上での Java 階層統合型粗粒度タスク並列処理 . 電子情報通信学会技術研究報告, CPSY 2011-28, 2011.

[6] 石畑清. 岩波講座 ソフトウェア科学 3 アルゴリズムとデータ構造 . 岩波書店, 1989.