B-001

階層統合型粗粒度タスク並列処理のための共有データ管理手法

Shared Data Management Scheme for Layer-Unified Coarse Grain Task Parallel Processing

越智　佑樹 †　　中田　大貴 †　　吉田 明正 †
Yuuki Ochi　Daiki Nakata　Akimasa Yoshida

1　はじめに

マルチコアプロセッサ上での Java プログラムの並列処理手法として提案されている階層統合型粗粒度タスク並列処理では，階層的に粗粒度タスク間の並列性を抽出し，Java 実装されたダイナミックスケジューラが全階層の粗粒度タスクをコアに割り当て，階層を超えた並列性を利用することが可能である．本稿では，階層統合型粗粒度タスク並列処理において，動的な共有データ管理と静的な共有データ管理を併用することにより，マクロタスクスケジューリングおよびマクロタスク実行の際のデータアクセス時間を軽減する共有データ管理手法を提案する．Java 言語における並列処理としてはループ並列処理 [1]，トレース間の並列処理 [2]，zJava システムによるポインタベースの動的データを扱う並列処理 [3] が提案されている．本稿では，並列化指示文付 Java プログラムを入力とし，並列化コンパイラを用いて並列 Java コードを生成し，本手法を実装した並列 Java コードを用いて Sun Fire T1000 上で性能評価を行った．

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 [5][6] では，まず粗粒度タスク並列処理 [4] 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を導入し，全階層のマクロタスク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を生成する．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．

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 [4] による実行では，プログラムを階層的に，基本ブロック，繰返しブロック（for 文や while 文等），サブルーチンブロック（メソッド呼出し）の 3 種類のマクロタスク（MT）に分割する．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すると，図 2 のような階層型マクロタスクグラフ（MTG）が生成される．

次に，階層統合型実行制御 [5] 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できる．

2.2　階層統合型実行制御の最早実行可能条件

マクロタスク生成後，各階層のマクロタスク間の制御フローとデータ依存を解析し，階層型マクロフローグラフ [4] を生成する．次に，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 [4] 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．

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

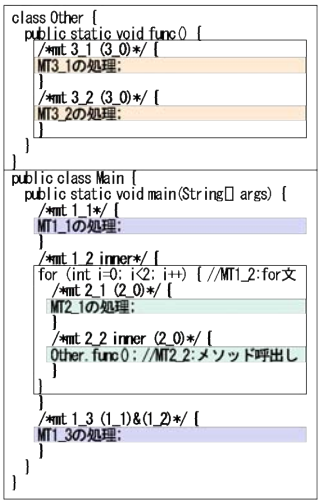

図 1　並列化指示文を伴う Java プログラム．

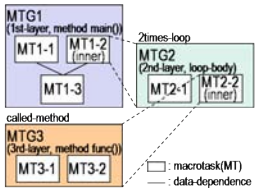

図 2　階層型マクロタスクグラフ（MTG）．

3　階層統合型粗粒度タスク並列処理における共有データ管理

本節では，マルチコアプロセッサ上で階層統合型粗粒度タスク並列処理を実現するための並列 Java コードの構成について述べる．

3.1　共有データ管理

階層統合型粗粒度タスク並列処理では，異なる階層のマクロタスクを統一的に扱うため，メソッド内部のマクロタスクとメソッド呼び出し側のマクロタスクもダイナミックスケジューラが一元管理する．この際，MTG(メソッド）内部のマクロタスク間共有変数を管理する変数管理クラス VARmanage クラスと MTG(メソッド）内部のマクロタスク制御用変数を管理する MTmanage クラスを作成し，共有データを管理している．

3.2 動的配列による共有データ管理

再帰呼び出しや実行時にループ回数が決まる繰り返し分の内部において，メソッド呼び出しを行うにはそのメソッドに対応するプログラムの場合には VARmanage クラスと MTmanage クラスのインスタンスはそれぞれ実行回数分必要となる．この場合のインスタンスはメソッド呼び出しの際に階層開始マクロタスクを実行し，動的に VARmanage クラス MTmanage クラスのインスタンスを生成すればよい．この方法では，実行時に動的に要素を追加していくことが可能となり不定回数のメソッド呼び出しを含むプログラムにも対応することができる．

Java 言語で実現する場合，ArrayList を用いるのが一般的であるが，この場合，データアクセスの度にデータの追加（add），取得（get），置換 (set) の操作が実行される．このため逐次プログラムでは不必要のないデータアクセスのオーバーヘッドが生じる．

3.3 静的・動的配列による共有データ管理

メソッド同時呼び出し回数が静的に決まっている場合や，収束ループ内でのメソッド呼び出しの場合には VARmanage と MTmanage のインスタンス数を静的に決定しインスタンスを再利用することが可能となる．静的な共有データ管理としては Java 言語ではそれぞれの VARmanage クラスと MTmanage クラスにおいて配列としてそれぞれの要素を用意しておく．動的な共有データ管理に対して静的な共有データ管理の場合には 3.2 節で述べた動的配列によるオーバーヘッドが軽減される．また階層開始マクロタスクにおけるインスタンス生成のオーバーヘッドが軽減される．静的な共有データ管理が可能である部分は静的な共有データ管理を適用し，静的データ管理を適用できない部分のみに動的なデータ管理を適用し，オーバーヘッドを軽減し，全体の処理速度を向上させる．

3.4 並列 Java コードの構成

階層統合型粗粒度タスク並列処理による実行を行う場合，本研究室で開発している並列化コンパイラに，並列化指示文を伴う Java プログラム（例，図 1）を入力し，並列 Java コードを生成する．並列 Java コードによる実行では，各スレッドは，コア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，ダイナミックスケジューリング用共通データ MTmanage クラスのための Data クラス，ユーザ定義クラスとメソッドのための VARManage クラスである MTG 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．eeccheck() メソッドでは，引数で与えられたマクロタスクが最早実行可能条件を満たしているかを判定している．

4 マルチコアプロセッサ上での性能評価

本性能評価では，マルチコアプロセッサ Sun Fire T1000 を用いる．T1000 は，UltraSPARC T1（1.0GHz, 8 コア）と 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，SPECfp95 のベンチマークの SWIM プログラムから，f2j ツールを用いて Fortran から Java に変換した Java プログラムを用いる．この Java プログラムに，本研究室で開発した並列化コンパイラを用いて，階層統合型並列処理の並列 Java コード（ループ分割数は 32）を生成する．その後，本手法を実装した並列 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 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．

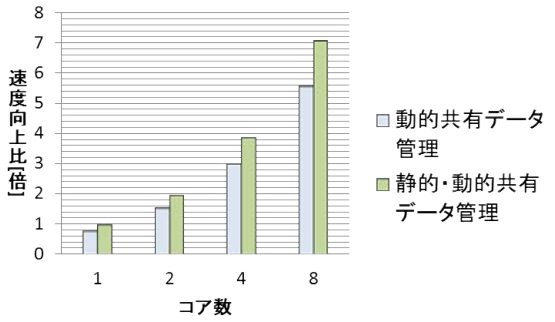

図 3　逐次実行に対しての速度向上比率.

図 3 は並列化コンパイラで生成した動的共有データ管理と，提案する静的・動的共有データ管理における逐次プログラムに対する速度向上比を表している．並列実行結果は，動的共有データ管理手法によるコードでは逐次プログラムに対して T1000 では 4 コアで 3.0 倍，8 コアで 5.56 倍であったのに対して静的・動的な共有データ管理を行ったコードでは 4 コアで 3.85 倍，8 コアで 7.09 倍の速度向上が得られている．本結果より動的共有データ管理手法に対して静的・動的共有データ管理手法を取り入れることによって 1.27 倍の速度向上を得ることができた．

5 おわりに

本稿では，階層統合型粗粒度タスク並列処理における共有データ管理手法を提案した．本手法では，動的な共有データ管理手法と静的なデータ管理手法を併用することにより，データアクセス時間の軽減が可能となる．

本手法を実装した並列 Java コードを，Sun Fire T1000 上で実行したところ，SWIM 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，共有データのデータアクセスオーバーヘッドの軽減により効率的な並列処理が実現された．本稿で提案した手法は並列化コンパイラへと実装中である．

参考文献

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

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

[3] Bryan Chan, Tarek S. Abdelrahman. Run-Time Support for the Automatic Parallelization of Java Programs. *The Journal of Supercomputing*, 28, 91-117, 2004.

[4] 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.

[5] 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.

[6] 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.