5L-02

POWER8 上でのローカルタスク協調実行を伴う タスク駆動型粗粒度並列処理

Task-Driven Coarse Grain Parallel Processing with Local-Task Cooperative Execution on POWER8

山端 大揮 †　　岡 宏樹 ‡　　吉田 明正 †
Daiki Yamahata　　Hiroki Oka　　Akimasa Yoshida

1 はじめに

マルチコア上での粗粒度並列処理手法として，計算プラットフォームに依存しないタスク駆動型実行による粗粒度並列処理が提案されている．タスク駆動型粗粒度並列処理では，Java Fork/Join Framework を利用した粗粒度タスクのスケジューリングが導入されており，開発した並列化コンパイラにより並列 Java コードを生成する [1]．本手法は複数階層の粗粒度タスク間の並列性のみならず，ローカルタスク（ループイタレーション集合あるいは再帰呼び出し文）間の並列性を利用することが可能である [2]．

近年，並列サーバーのプロセッサとして POWER プロセッサが注目されており，SMT（同時マルチスレッド化）を伴って高い実効性能を達成している．POWER プロセッサの並列処理手法としては，ループ並列処理やマルチグレイン並列処理 [3] 等があげられるが，本稿では新たな並列処理手法としてタスク駆動型粗粒度並列処理 [1] を適用する．本手法による実行では，ベンチマークプログラム等に対して開発した並列化コンパイラがタスク駆動型粗粒度並列処理の並列コードを生成し，POWER8 プロセッサ上での並列コードの実行結果からその有効性を確認した．

2 タスク駆動型粗粒度並列処理

本稿で利用するタスク駆動型粗粒度並列処理 [1] とは，Java Fork/Join Framework 環境において，階層統合型の粗粒度並列処理 [4] を実現するための，データ依存と制御依存を考慮した粗粒度タスクの並列実行手法である．

2.1 Fork/Join Framework によるタスク駆動型実行

このタスク駆動型実行では，入力対象となるプログラムの構造に対応した階層を定義し，各階層のマクロタスク間のデータ依存と制御依存を解析して，最早実行可能条件 [4] の形で並列性を実現する．これは図 1 のようなマクロタスクグラフとして表現され，4 コア (4 スレッド) 上で並列実行したイメージは図 2 のようになる．

その後，マクロタスクの終了状態と分岐状態を管理し，新たに実行可能になるマクロタスクを fork し，ワーカーキューに投入される．そして，Fork/Join Framework のスケジューラがワーカーキューのマクロタスクを取り出し，ワーカースレッドで実行する．このとき，必要に応じてワークスティーリングが行われる．

2.2 ローカルタスク協調実行による並列処理

タスク駆動型粗粒度並列処理において高い並列性を引き出すために，並列可能ループや再帰メソッドのような

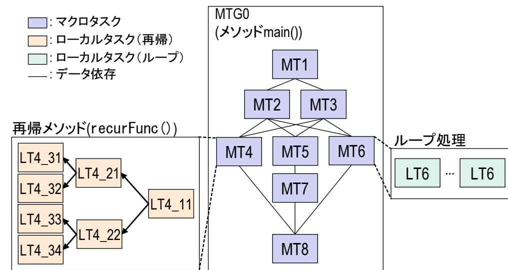

図 1　階層型マクロタスクグラフ

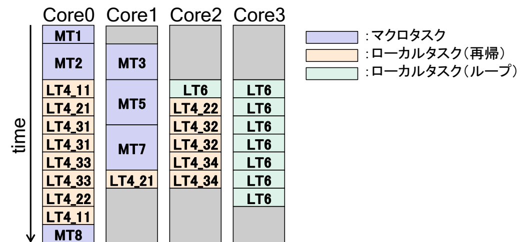

図 2　4 コアでのタスク駆動型実行の並列処理イメージ

マクロタスクを複数のタスクに分割するリストラクチャリングを行う．その際，マクロタスク実行管理のオーバーヘッドを軽減するために，Fork/Join Framework のスケジューラを用いたローカルタスク協調実行を導入する [2]．ローカルタスク協調実行では，タスク駆動型粗粒度並列処理のスケジューラがマクロタスクの実行を管理し，Fork/Join Framework のスケジューラがマクロタスク内部のローカルタスクを低オーバーヘッドで実行管理する．この際，並列可能ループの一部となる部分ループ，あるいは，再帰メソッド内部の再帰呼び出し文がローカルタスクとして定義される．

2.3 並列化コンパイラ

本研究で開発した並列化コンパイラ [1] は，並列化指示文を加えた Java プログラムを入力ソースファイルとして，Java Fork/Join Framework を利用したタスク駆動型粗粒度並列処理コードを出力する．タスク駆動型実行による粗粒度並列処理を実現する際，Java プログラムにマクロタスクを定義する並列化指示文を挿入し，タスク駆動型粗粒度並列処理用の並列化コンパイラによって，図 1 のようなマクロタスクグラフを生成し，図 3 のような並列 Java コードを出力する．

3 POWER8 上での粗粒度並列処理の性能評価

本性能評価では，表 1 に示す Java Grande Forum Benchmark Suite Version 2.0[5]，及び再帰プログラムを用いた性能評価を行う．性能評価には POWER8 プロセッサを搭載した並列サーバー IBM Power S812L を利用した．本並列サーバーは，POWER8 プロセッサ（3.02GHz, 12 コア，SMT=8），メモリバンド幅 192GB/s，メモリ 128GB，OS RHEL 7 FOR POWER，Java 処理系 JDK1.8 の構成である．

†明治大学総合数理学部ネットワークデザイン学科
Department of Network Design, School of Interdisciplinary Mathematical Sciences, Meiji University
‡明治大学大学院先端数理科学研究科
Graduate School of Advancd Mathmatical Sciences, Meiji University

Copyright ©2019 Information Processing Society of Japan.
All Rights Reserved.

表 1　性能評価プログラム

プログラムの特性	Crypt	Series	MonteCarlo	フィボナッチ	マージソート
プログラムの種類	暗号化処理	フーリエ級数	モンテカルロ法	—	—
データセット	C(N=5000 万)	B(N=10 万)	A(N=1 万)	—	(N=5000 万)
並列化対象のソースコード長	308	505	553	64	95
タスク駆動型並列 Java コード長	584	761	856	415	607
逐次実行時間 [ms]	3,194	143,911	5,109	19,508	10,814

```
01: class Mainp { //並列メイン
02:     static class Layer0 extends RecursiveTask<Void> { //ForkJoin開始
03:         Layer0() { //コンストラクタ
04:             Dataクラスのフィールド変数の初期化;
05:         }
06:         protected void compute() {
07:             ForkTemplate_mainクラスのmtStartのforkを行う;
08:             helpQuiesce()でタスク処理へ移行;
09:             //joinを行う;
10:         }
11:     }
12:     public static class ForkTemplate_main extends RecursiveTask<Void> {
13:         マクロタスク間共有変数等の宣言;
14:         ForkTemplate_main(MT識別情報) { //コンストラクタ
15:             MT識別情報をフィールド変数に設定;
16:         }
17:         protected void compute() { 該当するマクロタスクを実行; }
18:         public void mtStart(){ //mtStart
19:             マクロタスク実行管理テーブル更新;
20:             後続マクロタスクのforkを試みる;
21:         }
22:         public void mt1() { ... }
23:         ...
24:         public void mt8() { ... }
25:     }
26:     static int recurFunc(int n) { RecurFuncをfork; }
27:     private static class RecurFunc extends RecursiveTask<Integer> { ... }
28:     public static void main(String[] args) {
29:         ForkJoinPool pool = new ForkJoinPool(ワーカースレッド数);
30:         Layer0 layer0 = new Layer0();
31:         pool.invoke(layer0); //ForkJoin開始
32:     }
33: }
```

図 3　タスク駆動型並列 Java コード

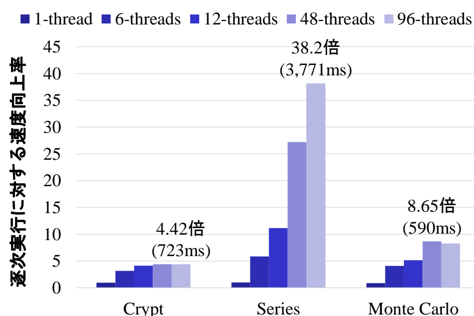

図 4　Java Grande Benchmark Suite による性能評価

3.1　ベンチマークプログラムを用いた性能評価

本節では，Java Grande Forum Benchmark Suite Version 2.0 より，暗号化処理を行う Crypt，フーリエ級数展開を行う Series，モンテカルロ法を用いた計算を行う MonteCarlo の 3 つのベンチマークプログラムを用いて性能評価を行う．これらのプログラムは並列可能ループを含んでおり，ローカルタスク協調実行を伴う並列処理を適用した．

POWER8 プロセッサ上での速度向上率 (逐次実行比) を図 4 に示す．この図では，それぞれのベンチマークプログラムに対して，スレッド数を 1 から 96 まで変化させており，最大の速度向上率はそれぞれ 4.42 倍，38.2 倍，8.65 倍を達成した．

3.2　再帰プログラムを用いた性能評価

次に，フィボナッチ数 fib(43) を 8 回求めるプログラムと，5000 万要素配列のマージソートプログラムを用いて性能評価を行う．これらのプログラムには再帰メソッドを含んでおり，ローカルタスク協調実行を伴う並列処理を適用した．

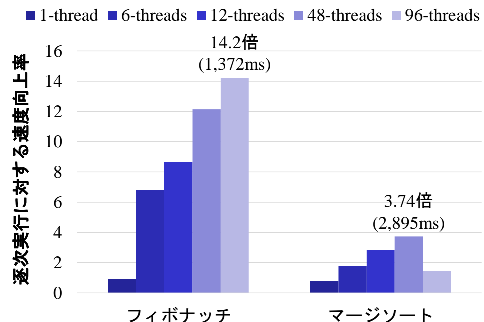

図 5　再帰プログラムによる性能評価

POWER8 プロセッサ上での速度向上率 (逐次実行比) は図 5 の通りであり，それぞれの再帰プログラムに対して，最大の速度向上率が 14.2 倍，3.74 倍となった．

4　おわりに

本稿では，粗粒度並列処理手法としてローカルタスク協調実行を伴うタスク駆動型粗粒度並列処理を適用し，その並列化コンパイラを用いてタスク駆動型並列 Java コードを生成した．性能評価では，マルチコアプロセッサ POWER8 の 12 コア上で並列実行を行っており，ローカルタスク協調実行を伴うタスク駆動型粗粒度並列処理において，ベンチマークプログラムで最大 38.2 倍，再帰プログラムで最大 14.2 倍の速度向上が得られた．

以上の結果から，POWER8 プロセッサにおいて，Java Fork/Join Framework 実装によるローカルタスク協調実行を伴うタスク駆動型粗粒度並列処理の有効性，並びにその並列化コンパイラの有用性が確認された．

本研究の一部は，JSPS 科研費基盤研究 (C) 課題番号 16K00174 の助成により行われた．

参考文献

[1] A.Yoshida, A.Kamiyama, H.Oka. A Task-driven Parallel Code Generation Scheme for Coarse Grain Parallelization on Android Platform, Journal of Information Processing, Vol.25, pp.426-437, 2017.

[2] 岡宏樹，吉田明正．メニーコア上でのローカルタスク協調実行を伴う Java プログラムのタスク駆動型粗粒度並列処理，研究報告システム・アーキテクチャ(ARC), Vol.2018-ARC-232, No.24, pp.1-9, 2018.

[3] 奥村万里子，柴崎大侑，桑島昂平，見神広紀，木村啓二，門下康平，中野恵一，笠原博徳．OSCAR コンパイラを用いた医用画像フィルタリングのマルチグレイン並列処理，情報処理学会研究報告, Vol.2016-HPC-153, No.13, pp.1-7, 2016.

[4] A.Yoshida, Y.Ochi, N.Yamanouchi. Parallel Java Code Generation for Layer-unified Coarse Grain Task Parallel Processing, IPSJ Transactions on Advanced Computing Systems, Vol.7, No.4, pp.56-66, 2014.

[5] EPCC. The Java Grande Forum Benchmark Suite, *https://www.epcc.ed.ac.uk/research/computing/performance-characterisation-and-benchmarking/java-grande-benchmark-suite*, 2018.

Copyright ©2019 Information Processing Society of Japan.
All Rights Reserved.