C-033

Android 搭載ヘテロジニアスマルチコアにおける Fork/Join Framework を用いた粗粒度並列処理

Coarse Grain Parallel Processing Using Fork/Join Framework for Android Heterogeneous Multicore

岡 宏樹 †　　　吉田 明正 †
Hiroki Oka　　　Akimasa Yoshida

1　はじめに

Android 搭載マルチコアにおける Java プログラムの並列処理環境として, Android API level 21 より Fork/Join Framework が導入されている．Fork/Join Framework はワークスティーリングを伴うスケジューラが利用できるようになっているが，一般的なプログラムに適用して粗粒度タスク間の並列性を引き出すことは困難であった．このような問題点を解決するために, 指示文付 Java プログラムを入力として, 並列化コンパイラにより Fork/Join Framework を用いた粗粒度並列処理コードを自動生成する方法が提案されている [1][2]．本稿では，Fork/Join Framework を用いた粗粒度並列処理コードを，ヘテロジニアスマルチコア Samsung Exynos 7420 を搭載した Android スマートフォン Galaxy S6 に適用して性能評価を行い，その有効性を確認する．

2　タスク駆動型実行による粗粒度並列処理

本稿で利用するタスク駆動型実行による粗粒度並列処理 [1] は，Java Fork/Join Framework 環境で階層統合型の粗粒度並列処理 [3] 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 を用いたタスク駆動型実行

このタスク駆動型実行では，入力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 [3] の形で並列性を実現する．これはマクロタスクグラフ (図 1(b)) として表現される．その後，マクロタスクの終了状態と分岐状態を管理し，当該マクロタスクの状態変化により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て，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　並列化コンパイラによる並列 Java コード生成

本稿では，前節で述べた実行手法を利用した粗粒度並列処理用の並列化コンパイラを開発した．図 1(a) は粗粒度並列処理用の並列化コンパイラ [2] に入力される指示文付き Java コードの一部である．図 1(a) のように並列化したい処理の直前に指示文を挿入することで，図 1(b) のようなマクロタスクグラフを生成し，図 1(c) のような構成の並列 Java コードを出力することができる．

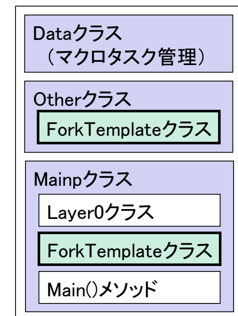
3　Android プラットフォームでの粗粒度並列処理

本稿では，2 章の並列 Java コードを Android プラットフォームで実行する方法について述べる．

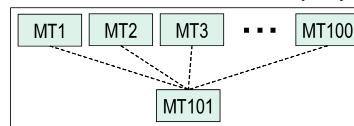
†明治大学総合数理学部ネットワークデザイン学科
Department of Network Design, School of Interdisciplinary Mathematical Sciences, Meiji University

```
/*mt fork decomp=100
  private(i,x,x2,y,y2)
  reduction(+:sum)*/
{
  for(i=0;i<N;i++){
    x=i*step;
    x2=(i+1)*step;
    y=4.0/(1.0+x*x);
    y2=4.0/(1.0+x2*x2);
    sum+=(y+y2)*step/2.0;
  }
}
```

(a) 指示文付Javaプログラム



(c) 並列化コンパイラで生成されたタスク駆動型並列Javaプログラム



(b) マクロタスクグラフ

図 1　並列化前及び並列化後の Java コード．

3.1　並列 Java コードの実行方法

並列化コンパイラによって生成された並列 Java コードは Android Studio 上で MainActivity.java の onClick() メソッドに埋め込むことで，Android プラットフォーム上で実行可能となる．

Android Studio の Build により，java ファイルは class ファイルを経て dex(Dalvik Executable) ファイルに変換される．ART ランタイムの Android 端末では AOT(Ahead-Of-Time) コンパイル方式が採用されており，dex ファイルは Android 端末に Load する際にネイティブコードに変換される．

3.2　ヘテロジニアスマルチコアへの適用

本研究ではこのタスク駆動型並列 Java コードをヘテロジニアス Android プラットフォームへ適用させた．従来のようなブロック分割，または，サイクリック分割によるタスクの割り当てをヘテロジニアスなプラットフォームに適用すると，コアの処理速度の違いによってその処理時間に大きな差が生じてしまい，コアの性能を最大限に活用することができない [4]．しかし，タスク駆動型のようなダイナミックスケジューリング方式ではコアの処理状況に応じてタスクの割り当てを行うため，全コアの性能を最大限に活用することができる．

4　Android プラットフォームでの粗粒度並列処理の性能評価

本性能評価では，ベンチマークプログラムを用いた性能評価とスケジューリングオーバーヘッドの性能評価を行う．

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

表 1　性能評価に用いる Android 搭載マシン.

マシン	Samsung Galaxy S6	NVIDIA Shield タブレット
プロセッサ	Exynos 7420	Tegra K1
CPU コア	Cortex-A57(2.1GHz) +A53(1.5GHz) (4 コア+4 コア)	Cortex-A15 r3 (2.3GHz)(4 コア)
メモリ	3GB	2GB
OS	Android 6.0.1 (API level 23)	Android 6.0 (API level 23)

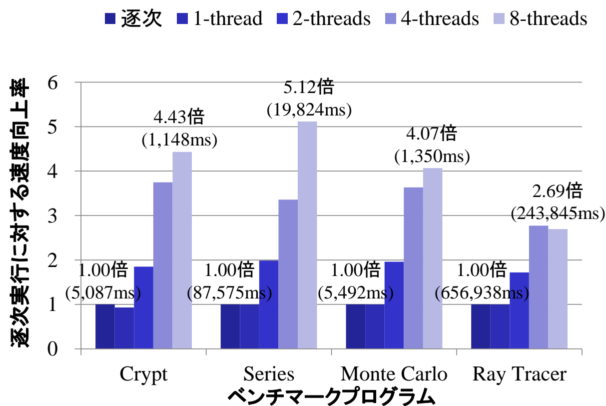


図 2　Galaxy S6 上でのベンチマークによる性能評価.

4.1　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[5] 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 Monte Carlo，光線追跡法を用いた計算を行う Ray Tracer の 4 つのベンチマークプログラムを用いて性能評価を行う．評価に用いたマシンは表 1 にある 2 つのマシンである．

まず，8 コア搭載の Galaxy S6 上での並列処理における速度向上率 (逐次実行比) を図 2 に示す．それぞれのベンチマークプログラムの 8 スレッド実行時の速度向上率が 4.43 倍，5.12 倍，4.07 倍，2.69 倍となっている．この測定結果から，タスク駆動型並列 Java コードがヘテロジニアス Android プラットフォームにおいて最大 5.12 倍という高い速度向上率を達成できることが確かめられた．

次に，ホモジニアス 4 コア搭載の Shield タブレット上での並列処理における速度向上率 (逐次実行比) を図 3 に示す．この測定結果から，Android タブレット Shield タブレット上で並列処理を行った場合，最大 3.94 倍という高い速度向上率を達成できることが確かめられた．

4.2　スケジューリングオーバーヘッドの性能評価

表 2 は積分による円周率計算 (図 1(a) はその主要部分) の計算時間と逐次比を示した表である．図 1(b) はそのマクロタスクグラフとなっており，MT1〜MT100 で分割された積分計算を並列に行い，MT101 でその合計を計算している．積分計算のマクロタスク数を 100 個に固定，各マクロタスクの積分領域の個数は N/100 となっており，積分分割数 N の大きさを 10^5，10^6，10^7 としている．

N=10^6，10^7 の場合ではそれぞれ 8 コアで 2.85 倍，3.48 倍の速度向上率が得られており，マクロタスクの処理時間に対してスケジューリングオーバーヘッドは小さい．

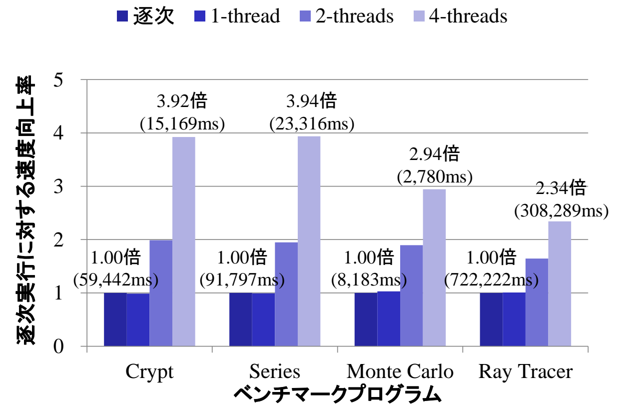


図 3　Shield 上でのベンチマークによる性能評価.

表 2　Galaxy S6 上でのスケジューリングオーバーヘッドの性能評価.

積分分割数 N	逐次	1 コア	4 コア	8 コア
N=10^5	5,114 (1.00 倍)	8,043 (0.64 倍)	5,692 (0.90 倍)	8,821 (0.58 倍)
N=10^6	50,596 (1.00 倍)	56,072 (0.90 倍)	17,806 (2.84 倍)	17,758 (2.85 倍)
N=10^7	336,476 (1.00 倍)	338,932 (0.99 倍)	117,437 (2.87 倍)	96,620 (3.48 倍)

※単位は [μs]．(　) 内は逐次比.

5　おわりに

本稿では，ヘテロジニアスマルチコア上での Java Fork/Join Framework を用いた粗粒度並列処理を実現するために，タスク駆動型並列 Java コード生成手法を提案し，その並列化コンパイラを開発した．

Java Grande Forum Benchmark Suite による性能評価では，Android スマートフォン Galaxy S6 の 8 スレッド実行では最大 5.12 倍，Android タブレット Shield タブレットの 4 スレッド実行では最大 3.94 倍の速度向上が得られた．

以上の結果から，ヘテロジニアス及びホモジニアス Android プラットフォーム両方において，Java Fork/Join Framework を用いたタスク駆動型実行による粗粒度並列処理の有効性が確認された．

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] 吉田明正, 神山彰．Android プラットフォームにおける Java Fork/Join Framework を用いた粗粒度並列処理，情報処理学会研究報告 Vol.2016-ARC-218 No.11, 2016.

[2] 神山彰，吉田明正．Java Fork/Join Framework を用いた粗粒度並列処理コードの自動生成，情報処理学会研究報告 Vol.2015-ARC-214 No.6，2015.

[3] 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 56-66，2014.

[4] Tulika Mitra．Heterogeneous Multi-core Architectures，IPSJ Transactions on System LSI Design Methodology Vol.8 51-62，2015.

[5] EPCC．The Java Grande Forum Benchmark Suite, *https://www2.epcc.ed.ac.uk/computing/research_activities/java_grande/*，2016.

Copyright © 2016 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.