6J-02

GPUシステムにおける階層統合型粗粒度並列処理の並列化コンパイラ

Parallelizing Compiler for Layer-Unified Coarse Grain Parallel Processing on GPU System

渡辺 智之 †
Tomoyuki Watanabe

吉田 明正 †‡
Akimasa Yoshida

1 はじめに

マルチコアシステムにおける並列処理手法として，ループ並列性や粗粒度タスク並列性を利用する階層統合型粗粒度並列処理手法 [1][2] が提案されている．GPUを搭載したマルチコアシステムにおいては，階層型粗粒度並列処理 [3] が提案されている．

本手法では，階層統合型粗粒度並列処理 [4] において，処理時間の大きいマクロタスクの実行に対して，コアの代わりにGPUを使用し，プログラム全体の実行時間の短縮を目指す．本研究では，LLVM/Clangを用いた並列化コンパイラを開発しており，その並列化コンパイラを用いてCUDAを伴う並列Cコードを自動生成する．性能評価では，Tesla K80を搭載したXeonサーバ上で，粒子法とヤコビ法のプログラムに対して，並列化コンパイラにより生成した並列コードを用いて性能評価を行った結果，提案手法の有効性が確認された．

2 GPU上での階層統合型粗粒度並列処理

階層統合型粗粒度並列処理では，階層型マクロタスクグラフ（MTG）[1] を生成し，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種類から構成される粗粒度タスク（マクロタスク，MT）を階層的に定義する．その後，最早実行可能条件 [1] を満たした全階層のマクロタスクを，ダイナミックスケジューラが統一的にコアに割り当てて実行する．例えば，図1のプログラムは，4コア＋1GPUの場合，図2のように実行される．

本研究では，開発した並列化コンパイラを用いることで，CUDAコードを伴う階層統合型粗粒度並列処理コードを自動生成する．

2.1 任意GPU割当てを伴うMTスケジューリング

本スケジューラは，CPU用のレディキューとGPU用のレディキュー（共通GPUキュー）を使用する．各コアはGPUがアイドル状態であれば，共通GPUキューを優先し，そうでなければCPUキューよりMTを取り出して実行する．それゆえ，GPUを効率的に利用することができる．

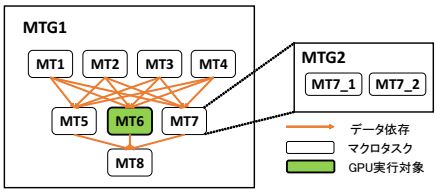

図 1: 階層型マクロタスクグラフ（MTG）．

† 明治大学大学院先端数理科学研究科
Graduate School of Advanced Mathematical Sciences, Meiji University
‡ 明治大学総合数理学部
School of Interdisciplinary Mathematical Sciences, Meiji University

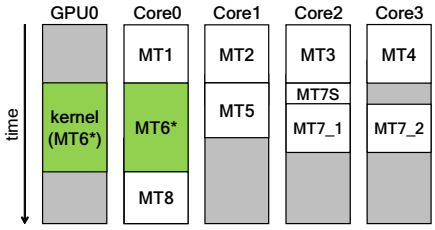

図 2: 4コア＋1GPU上での階層統合型粗粒度並列処理の実行イメージ．

```
1  MT 間共有変数，スケジューラ用変数の宣言;
2  double *data;/*ホスト用*/
3  double *d_data[devMAX];/*デバイス用*/
4  void init(){
5    /*ホスト・デバイスにデータ確保，初期化*/
6    data = (double *)malloc(sizeof(double)*N);
7    cudaMalloc((void **)&d_data[dev], sizeof(double)*N);
8  }
9  /*マクロタスクのコード（GPU)*/
10 void MT6(){
11   cudaSetDevice(dev); /*デバイス番号を設定*/
12   cudaMemcpy(d_data[dev], data, sizeof(double)*N,
13     cudaMemcpyHostToDevice);
14   kernel<<<BLOCK+1, THREAD>>>(d_data); /*GPU で実行*/
15   cudaDeviceSynchronize(); /*同期*/
16   cudaMemcpy(data, d_data[dev], sizeof(double)*N,
17     cudaMemcpyDeviceToHost);
18 }
19 ...
20 /*最早実行可能条件*/
21 void EEC(int mt){
22   mt の最早実行可能条件をチェック;
23 }
24 /*ダイナミックスケジューラ*/
25 void SCHEDULER(int num){
26   while(終了条件を満たすまで){
27     if(レディキューの投入MT 数>=1)
28       レディキューから 1MT を取り出す;
29     CPU または GPU で MT を実行;
30     EEC()を満たしたMT をレディキューに投入;
31   }
32 }
33 int main(){
34   init();
35   #pragma omp parallel
36   SCHEDULER(omp_get_thread_num());
37   return 0;
38 }
```

図 3: 階層統合型粗粒度並列処理コード．

2.2 指定GPU割当てを伴うMTスケジューリング

GPU実行対象MTに対して，使用するGPUのデバイス番号をあらかじめユーザが指定し，スケジューリングを行う．これにより，MT間のデータ授受がデバイス上で行われ，データの局所性を利用した実行が可能である．

3 階層統合型粗粒度並列処理のための並列化コンパイラ

本章では，開発した並列化コンパイラと生成された並列コードについて述べる．

3.1 並列化コンパイラ

本研究の並列化コンパイラはLLVM/Clang[5] を用いて開発されている．LibTooling[6] からClangが解析した抽象構文木の情報にアクセスすることにより，並列化指示文を挿入したCコードからCUDAを伴う並列Cコードを自動生成する．LLVM/Clangは3.7.1を用いた．

表 1: 並列コンパイラのための並列化指示文.

表記	意味
#pragma para mt(MT 番号){}	MT 番号のマクロタスクの定義
#pragma para mt(MT 番号) inner{}	内部にサブマクロタスクを含む MT 番号のマクロタスクの定義
#pragma para mt(MT 番号) kernel{}	GPU 実行対象とする MT 番号のマクロタスクの定義
#pragma para eec(out:MT 番号) eec(in:論理式)	out:で MT 番号を指定し，最早実行可能条件の論理式を in:で指定する．
#pragma para copy(変数名，初期値，型，サイズ，HostToDevice|DeviceToHost)	GPU で使用するデータの転送と転送方向
#pragma para group(デバイス番号，MT 番号)	指定 GPU 割当てにおいて，MT 実行に使用するデバイス番号を指定

表 2: ヤコビ法による任意 GPU 割当ての性能評価.

構成	実行時間 [s]	1 コア実行に対する速度向上率 [倍]
1 コア	112.1	1.0
4 コア	28.3	4.0
4 コア＋ 1GPU	15.2	7.4
4 コア＋ 2GPU	7.7	14.6
4 コア＋ 4GPU	3.9	28.7

3.2 並列化指示文

階層統合型粗粒度並列処理コードを生成するために，入力対象になる C コードに対して表 1 の並列化指示文を記述することにより，本並列化コンパイラで CUDA を伴う並列 C コードを生成する．マクロタスクとして定義する領域に#pragma para mt(MT 番号){} の指示文を記述する．最早実行可能条件は，#pragma para eec(out:MT 番号) eec(in:論理式) により記述する．GPU 実行対象マクロタスクの定義は，#pragma para mt(MT 番号) kernel{} を記述する．

3.3 並列化コンパイラ生成の並列コード

並列化コンパイラで生成した階層統合型粗粒度並列処理コードの構成を図 3 に示す.

図 3 の main() 関数では，init() により初期化を行い，OpenMP の指示文によりコア数分のスレッドを生成し，SCHEDULER() 関数を実行する．

4 GPU システム上における階層統合型粗粒度並列処理の性能評価

本章では，NVIDIA Tesla K80 を伴う Xeon サーバにおいてヤコビ法プログラムと粒子法プログラムを用いて性能評価を行う．

4.1 性能評価の環境

本サーバは，Intel Xeon E5-2680（12 コア）の CPU を 2 個，Tesla K80 を 2 個，メモリ 64GB を搭載している．各 Tesla K80 は 2496CUDA コアの GK210 を 2 台搭載しており，GK210 デバイスを 4 台使用する．OS は CentOS6.9，処理系は GCC4.4.7，CUDA Toolkit 9.1 である．

4.2 ヤコビ法を用いた任意 GPU 割当ての性能評価

性能評価プログラムとして連立 1 次方程式反復解法のヤコビ法を用いた．行列サイズは 40,960 × 40,960 としている．本性能評価では，任意 GPU 割当てにより実行した結果を表 2 に示す.

まず，CPU の 1 コアの実行時間は 112.1[s]，CPU の 4 コアによる実行時間は 28.3[s] となり 4.0 倍の速度向上が得られている．4 コア＋ 4GPU では 3.9[s] となり，28.7 倍の速度向上が得られた．

表 3: 粒子法プログラムによる指定 GPU 割当ての性能評価.

構成	実行時間 [s]	1 コア実行に対する速度向上率 [倍]
1 コア	2139.9	1.0
4 コア	718.3	3.0
4 コア＋ 1GPU	144.5	14.8
4 コア＋ 2GPU	83.0	25.8
4 コア＋ 4GPU	60.4	35.4

4.3 粒子法を用いた指定 GPU 割当ての性能評価

次に，粒子法プログラム [7] を用いて，指定 GPU 割当てにより実行した結果を表 3 に示す.

本性能評価では，粒子数が 68,416 個で取り扱う．CPU の 1 コアの実行時間は 2,139.9[s]，4 コア＋ 4GPU で 60.4[s] となり，35.4 倍の速度向上率が得られた．

5 おわりに

本稿では，CUDA コードを伴う階層統合型粗粒度並列処理コードを生成する並列化コンパイラを開発した．

ヤコビ法プログラムにおける性能評価では，Xeon サーバ上で実行した結果，最大 28.7 倍の速度向上率が得られた．また，粒子法プログラムにおいては最大 35.1 倍の速度向上率が得られ，提案する並列化コンパイラにより生成された並列コードの有効性が確かめられた．

参考文献

[1] 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 pp.2732-2740，2004.
[2] Yoshida, A., Ochi, Y., Yamanouchi, N.: 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.
[3] 林明宏，和田康孝，渡辺岳志，関口威，間瀬正啓，白子準，木村啓二，笠原博徳: ヘテロジニアスマルチコア向けソフトウェア開発フレームワークおよび API，情報処理学会論文誌，Vol.5，No.1 pp.68-79，2012.
[4] 渡辺智之，吉田明正: マルチコア/GPU 環境における階層統合型粗粒度タスク並列処理，情報処理学会研究報告，Vol.2018-ARC-232，No.25 pp.1-7，2018.
[5] The LLVM Compiler Infrastructure Project, https://llvm.org/，2019.
[6] LibTooling - Clang 10 documentation, https://clang.llvm.org/docs/LibTooling.html, 2019.
[7] 越塚誠一，柴田和也，室谷浩平: 粒子法入門，丸善出版株式会社，2014.