3A-05

GPU アクセラレータを伴うマルチコア上での階層統合型粗粒度並列処理

Layer-Unified Coarse Grain Parallel Processing on Multicore Systems with GPU Accelerators

安部 拓未 †　　吉田 明正 †
Takumi Abe　　Akimasa Yoshida

1　はじめに

GPU アクセラレータを伴うマルチコアシステムは，近年，様々な分野で広く普及している．マルチコアシステムにおいて複数階層の粗粒度タスク並列性を利用する方法として，階層統合型実行制御 [1] を伴う粗粒度並列処理手法 [2] が提案されている．この粗粒度並列処理手法では，OpenMP あるいは Java マルチスレッドを用いて，ダイナミックスケジューリングコードを含む並列コードを生成する．CPU/GPU 環境での並列処理に関する研究 [3] は行われているが，階層統合型実行制御のように複数階層のマクロタスクを統合的に扱うことはできない．

そこで，本稿では，マルチコアと複数 GPU からなる並列システムにおいて，階層統合型実行制御により粗粒度タスク間の並列性を利用しつつ，処理時間の大きい粗粒度タスクの実行に GPU を用いて高速化を実現する手法を提案する．性能評価では，Intel Xeon E5-2680 および NVIDIA Tesla K80 からなる並列システム上で，連立 1 次方程式反復解法であるヤコビ法プログラムを用いて並列実行を行い，その有効性を確認した．

2　階層統合型実行制御を伴う粗粒度並列処理

階層統合型の粗粒度並列処理 [1] をマルチコア上で実現するためには，まず，対象プログラムをマクロタスク（MT）と呼ばれる粗粒度タスクに階層的に分割し，各マクロタスクの最早実行可能条件 [4] を解析することにより，粗粒度タスク並列性を抽出する．この際，階層統合型実行制御を用いる場合には，階層開始マクロタスクを導入して，複数階層のマクロタスクを同一レベルのダイナミックスケジューリング対象として扱う [1]．抽出した粗粒度タスク並列性は，図 1 のようなマクロタスクグラフ（MTG）として表現される．

次に，各マクロタスクの最早実行可能条件を考慮したダイナミックスケジューリングコードを作成し，そのスケジューリングコードは OpenMP により生成された全スレッドで実行される．各マクロタスクはスレッド上で実行が終了すると，スケジューリングコードは新たな実行可能マクロタスクをレディキューに投入し，その後，レディキューの先頭からマクロタスクを取り出して実行する．

3　CPU/GPU 環境における粗粒度並列処理

本章では，マルチコア CPU と GPU を併用した環境において粗粒度並列処理を実現する方法について述べる．

3.1　OpenMP/CUDA による並列コード

階層統合型実行制御を伴う粗粒度並列処理の並列コードの構成を図 3 に示す．

†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

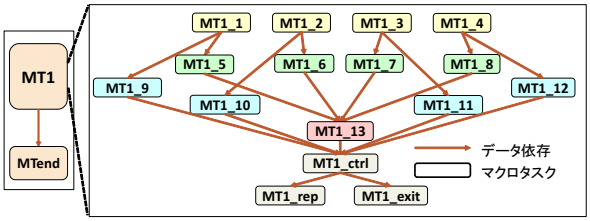

図 1　ヤコビ法のマクロタスクグラフ (MTG).

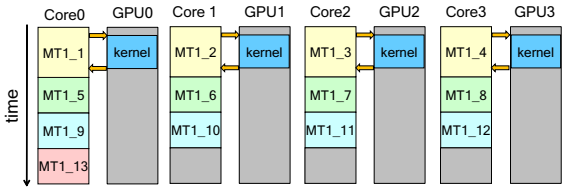

図 2　CPU/GPU 環境における粗粒度並列処理.

まず，36〜43 行目の main() 関数において，CORE 数分のスレッドを生成する．ここで各スレッドは 28〜35 行目の scheduler() 関数を実行する．この関数では，最早実行可能条件（EEC）を満たした MT をレディキューに投入し，レディキューから取り出した MT に相当する関数を実行する．例えば，図 1 のヤコビ法プログラムの場合，MT1_1 のコードは 10〜18 行目の MT1_1()，MT1_5 のコードは 20〜22 行目の MT1_5() に記述されている．

ここで，各マクロタスクに対応する関数のコードは，通常，CPU コードであるが，提案手法では，実行すべきマクロタスクを GPU 指定とする場合には，そのマクロタスク用の GPU コード（予め用意された kernel コードを呼び出す）を用意する．このようなユーザ指定による選択的なコードにより，マクロタスクごとに CPU（コア）実行，あるいは，GPU 実行を行うことが可能である．但し，本手法では GPU コードを用意する場合には，複数の GPU デバイス内の 1 つを指定する方式を取る．

例えば，前述の MT1_1 の実行において GPU を用いる場合，MT1_1() のコードは，13 行目の GPU デバイス設定，14 行目の CPU 変数から GPU 変数への転送，15 行目の kernel() 関数の呼び出し，16 行目の GPU 変数から CPU 変数への転送，17 行目の GPU デバイス内の同期の処理から構成される．

CPU/GPU 環境での実行イメージは，図 2 に示される通りであり，Core0〜Core3 において各ダイナミックスケジューラを伴うスレッドコードが実行され，指定された MT1_1〜MT1_4 の実行においては，GPU0〜GPU3 上で kernel() コードを呼び出して GPU 実行を行う．

4　GPU を伴うマルチコア上での性能評価

本章では，GPU 搭載の並列システムにおいて，連立 1 次方程式反復解法のヤコビ法プログラムを用いて，性能評価を行う．

Copyright ©2017 Information Processing Society of Japan.
All Rights Reserved.

```
01: #define BLOCK   //GPUのブロック数
02: #define THREAD  //GPUのスレッド数
03: #define CORE    //CPUのコア数
04: スケジューラ用変数の宣言;
05: MT間共有変数の宣言;
06: /*kernel関数*/
07: void kernel(){
08:     GPUコード;
09: }
10: /*MT1_1のコード*/
11: void MT1_1(){
12:     /*GPU指定の場合*/
13:     cudaSetDevice(0);  //GPUの実行デバイスを0に設定
14:     cudaMemcpy(&GPU変数, &CPU変数, …);  //CPUからGPUへメモリ転送
15:     kernel<<<BLOCK, THREAD>>>();  //kerenel関数呼び出し
16:     cudaMemcpy(&CPU変数, &GPU変数, …);  //GPUからCPUへメモリ転送
17:     cudaThreadSynchronize();  //スレッド同期
18: }
19: …
20: Void MT1_5(){
21:     CPUコード;
22: }
23: …
24: /*最早実行可能条件*/
25: void EEC(int mt){
26:     mtの最早実行可能条件をチェック;
27: }
28: /*ダイナミックスケジューラ*/
29: void scheduler(){
30:   while (全MTが終了するまで){
31:         各MTiのEECを満たしたらMTiをレディキューに投入;
32:         レディキューから取り出したMTjに相当する関数MTj()を実行;
33:         MTjの終了・分岐通知;
34:   }
35: }
36: /*main関数*/
37: void main(){
38:     データの初期化;
39:     #pragma omp parallel
40:     {
41:       scheduler(omp_get_thread_num());
42:     }
43: }
```

図 3　OpenMP/CUDA 実装による並列コード.

4.1　性能評価の環境

本性能評価では，表 1 に示す Dell PowerEdge R730 サーバで並列実行を行う．本サーバは，Xeon E5-2680（12 コア）の CPU を 2 個と Tesla K80 を 2 個搭載している [5]．各 Tesla K80 には 2496CUDA コアからなる GK210 を 2 基搭載しており，本サーバでは GK210 デバイスを 4 基使用することが可能である．なお，性能評価に用いる並列コードは，図 3 のように OpenMP/CUDA を用いて実装されている．

4.2　ヤコビ法のアルゴリズムと並列コード

本節では，性能評価プログラムとして用いたヤコビ法のアルゴリズムについて述べる．ヤコビ法プログラムの反復計算部分は，(手順 1) 新しい解を求める，(手順 2) 古い解と新しい解の差分の 2 乗和を計算する，(手順 3) 新しい解を古い解にコピーする，から構成される．これらの反復計算は，収束条件を満たすまで繰り返される．

ここで，ヤコビ法の (手順 1) から (手順 3) はそれぞれ，1 つのループとして記述することが可能であるが，粗粒度並列性を引き出すため，それらを事前にコア数と同数に分割している．それゆえ，ヤコビ法の (手順 1) は図 1 の MT1_1〜MT1_4，(手順 2) は MT1_5〜MT1_8，(手順 3) は MT1_9〜MT1_12 に対応している．

4.3　CPU/GPU 搭載サーバにおける性能評価

提案する粗粒度並列処理コードを，CPU/GPU 搭載サーバで実行した結果を図 4 に示す．評価に用いたヤコビ法の行列サイズは 16000 × 16000 である．

まず，CPU の 1 コアの実行時間は 11.085[s] であり，CPU の 4 コアでの粗粒度並列処理による実行時間は 2.783[s] となり，3.9 倍の速度向上が得られている．

次に，処理時間の大きいマクロタスクの実行には，CPU 実行ではなく，GPU 実行を試みる．図 1 の MT1_1〜MT1_4 は GPU による並列処理が可能であり，それぞれのマクロタスクに対して，GK210 からなる 4 つの GPU デバイスを指定する．ここで，1 コア+1GPU で並列実行を行うと，実行時間は 1.011[s] となり，1 コアのみの場合と比べて，GPU 実行の効果により 10.9 倍の速度向上が得られた．

表 1　並列システム Dell PowerEdge R730 の構成.

CPU	Intel Xeon E5-2680 v3, 2.5GHz，12 コア× 2 個
メモリ	64GB
GPU	NVIDIA Tesla K80 × 2 個
OS	CentOS 6.7
処理系	GCC 4.4.7，CUDA Toolkit 7.5

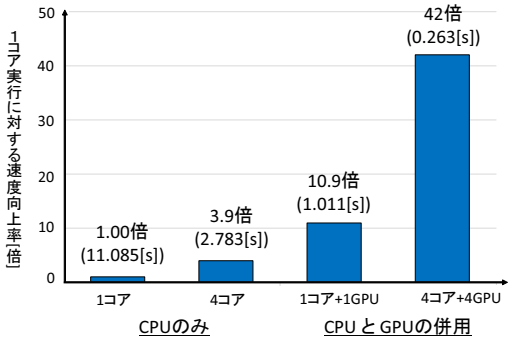

図 4　Tesla K80 を伴う Xeon E5-2680 システム上での性能評価.

さらに，4 コア＋4GPU で並列実行を行うと，その実行時間は 0.263[s] となり，1 コアのみの場合と比べて，マルチコアと GPU の両方の並列性の利用により 42 倍の速度向上を得た．この場合，各 GPU デバイスは 2496CUDA コアが利用可能であり，kernel() 関数の実行には 10 ブロック× 400 スレッドを指定している．

5　おわりに

本稿では，階層統合型実行制御の粗粒度並列処理において，CPU/GPU 実行を可能にする並列コード生成手法を提案した．本並列コードを用いることにより，マルチコア上で粗粒度並列性を利用しつつ，処理時間の大きいマクロタスクの実行に GPU を用いることで，マクロタスクの実行時間を短縮することが可能である．

提案手法は，OpenMP/CUDA 処理系を用いて実装されており，ヤコビ法を用いた性能評価では，Xeon E5-2680 の 4 コアと Tesla K80 の 4GPU デバイスからなる並列サーバ上で実行したところ，最大で 42 倍の速度向上が得られ，本手法の有効性が確認された．

今後の課題としては，実行時に GPU の利用状況を考慮して，動的に CPU 実行と GPU 実行を選択できる並列コードを開発することがあげられる．

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.12，2004.

[2] 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.

[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] 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] NVIDIA TESLA K80 GPU ACCELERATOR, *https://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf*, 2015.

Copyright ©2017 Information Processing Society of Japan.
All Rights Reserved.