C-014

GPU アクセラレータを伴うマルチコア上での ダイナミックスケジューリングを用いた階層統合型粗粒度並列処理

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

渡辺 智之† 吉田 明正†
Tomoyuki Watanabe 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 [1][2] を利用する方法として，階層統合型実行制御を伴う粗粒度並列処理手法 [1][3] 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 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．

本稿では，複数の GPU アクセラレータを伴うマルチコア環境において，ダイナミックスケジューリングを用いて粗粒度タスク間の並列性を利用しつつ，ユーザの指定した処理時間の大きい粗粒度タスクを GPU 実行して高速化を実現する方法を提案する．関連研究としては静的スケジューリングを用いて CPU と GPU を併用する並列処理手法が提案されている．性能評価では，Intel Xeon E5-2680 および 2 台の NVIDIA Tesla K80（GK210 を計 4 台）からなる並列システムを使用し，ヤコビ法と粒子法プログラムによる性能評価の結果から提案手法の有効性が確認された．

2 GPU アクセラレータを伴う階層統合型粗粒度並列処理

粗粒度並列処理 [1] を GPU アクセラレータを伴う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を解析することにより粗粒度並列性を抽出する．階層統合型粗粒度並列処理の場合には，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].

次に，各マクロタスクの最早実行可能条件を伴うダイナミックスケジューリングコードを生成する．このダイナミック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が実行され新たな実行可能マクロタスクをレディキューに投入する．

その後，アイドル状態のスレッドは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．この際，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（CUDA の kernel コード）を用意する [4].

3 OpenMP/CUDA 実装による並列コード

GPU アクセラレータを伴うマルチコア上におけるダイナミックスケジューリングを用いた階層統合型粗粒度並列処理の並列コードの構成を図 1 に示す．

```
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(dev);  //GPUの実行デバイスをdev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GPUキューorCPUキューに投入;
32:         GPUキューorCPUキューから取り出す;
33:         GPUキューの場合はGPUデバイス番号の指定;
34:         MTjに相当する関数MTj()を実行;
35:         MTjの終了・分岐通知;
36:   }
37: }
38: /*main関数*/
39: void main(){
40:     データの初期化;
41:     #pragma omp parallel
42:     {
43:       SCHEDULER(omp_get_thread_num());
44:     }
45: }
```

図 1 OpenMP/CUDA 実装による並列コード.

まず，main() 関数において，コア数分のスレッドを生成し，SCHEDULER() 関数を実行する．この関数では最早実行可能条件を満たした 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マクロタスクに対応する関数（マクロタスクコード）を実行する．ここで，レディキューとして GPU キュー，CPU キューの 2 つのキューを用意する．GPU キューは実行するマクロタスクが GPU 指定とされている場合に投入され，CPU キューは実行するマクロタスクが CPU 指定とされている場合に投入される．

例えば，図 2(a) のマクロタスクグラフを 4 コア＋4GPU のシステムで実行する場合を取り上げる．

MT2 の実行が終了した段階で，レディキューは図 2(b) のようになっている．この時，図 2(c) の Core3 で動作しているスケジューラは，MT7 を GPU キューから取り出して GPU3 で実行する．但し，Core3 は GPU3 の終了まで待機している．その後，Core2 と Core1 は，それぞれ MT6 と MT8 を実行する．このように 2 つのレディキュー（GPU 用，CPU 用）にマクロタスクが投入されている場合，GPU キューを優先する．

†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

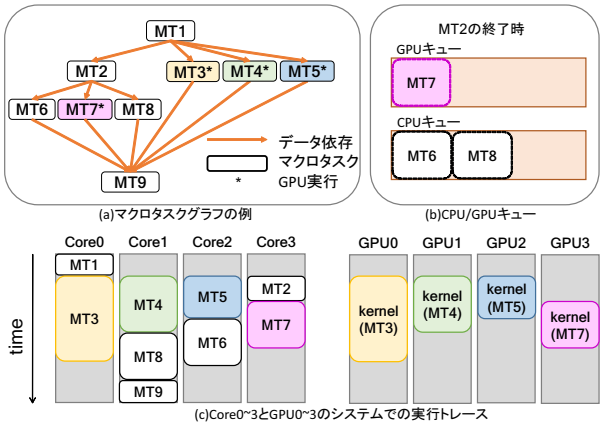

図 2　CPU/GPU 上でのマクロタスクの実行トレース.

4　Tesla K80 搭載マルチコアサーバでの性能評価

本章では，GPU 搭載のマルチコアサーバにおいて連立 1 次方程式反復解法のヤコビ法プログラム，粒子法プログラム [6] を用いて性能評価を行う.

4.1　性能評価の環境

本性能評価では，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，Tesla K80 を 2 個，メモリ 64GB を搭載している [5]．各 Tesla K80 には 2496CUDA コアからなる GK210 を 2 台搭載しており，本サーバでは GK210 デバイスを 4 台使用することが可能である．OS は CentOS 6.9，処理系は GCC 4.4.7，CUDA Toolkit 8.0 である.

4.2　ヤコビ法プログラムを用いた性能評価

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．ヤコビ法の反復計算は，収束条件を満たすまで繰り返され，行列サイズは 40000 × 40000 とした．TeslaK80 搭載マルチコアサーバで実行した結果を図 3 に示す．まず，CPU の 1 コアの実行時間は 88.7[s]，CPU の 4 コアによる実行時間は 22.2[s] となり 4.0 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に CPU 実行ではなく GPU 実行を試みる．GPU 実行には 4 台の GPU デバイス（GK210）を使用する．4 コア＋ 1GPU で並列実行を行うと実行時間は 8.9[s]，4 コア＋ 2GPU の場合の実行時間は 4.5[s]，4 コア＋ 4GPU で 2.3[s] という実行結果となり，これらの結果は 1 コアのみの場合と比べて 10.0 倍，19.7 倍，38.6 倍の速度向上が得られている．各 GPU デバイスでは 2496CUDA コアが利用可能であり，kernel() 関数の実行には 10 ブロック× 1000 スレッドを指定している.

4.3　粒子法プログラムを用いた性能評価

2 つ目の性能評価プログラムとして粒子法プログラム [6] を用いる．粒子法プログラムは粒子数を 19,136 個とし，陽解法で実行を行った．粒子法プログラムも同様に TeslaK80 搭載マルチコアサーバで実行し，その結果を図 4 に示す．まず，CPU の 1 コアの実行時間は 330.6[s]，CPU の 4 コアによる実行時間は 103.2[s] となり 3.2 倍の速度向上が確認できる.

次に処理時間の大きいマクロタスクに CPU 実行ではなく GPU 実行を試みる．GPU 実行には GK210 からなる GPU デバイスを利用する．4 コア＋ 1GPU による実行時間は 22.8[s] となり，1 コアのみの場合と比べて 14.5 倍の速度向上が得られた．本プログラムでは 2496CUDA

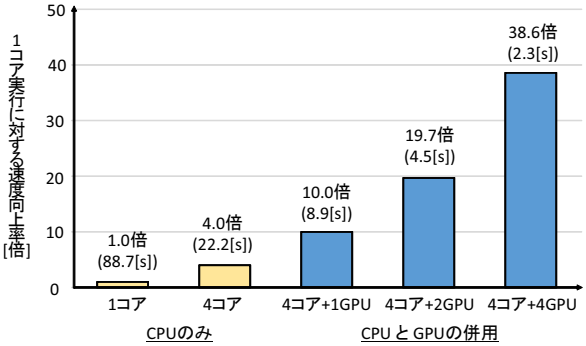

図 3　ヤコビ法プログラムによる性能評価.

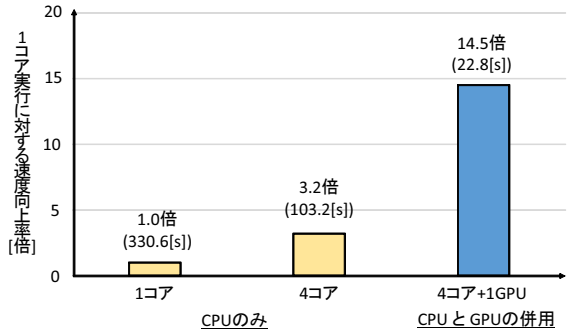

図 4　粒子法プログラムによる性能評価.

コアの GPU デバイス 1 台が利用可能であり，kernel() 関数の実行には 39 ブロック× 576 スレッドを指定している.

5　おわりに

本稿では，ダイナミックスケジューリングを用いた階層統合型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の生成手法を提案した.

提案手法は，OpenMP/CUDA 処理系を用いて実装されており，ヤコビ法プログラムにおける性能評価では，Xeon E5-2680 の 4 コアと Tesla K80 の 4GPU デバイスからなるサーバ上で実行したところ，最大で 38.6 倍の速度向上が得られた．また，粒子法プログラムでは 4 コアと 1GPU を用いて 14.5 倍の速度向上が得られており，本手法の有効性が確認された.

今後の課題としては，提案する並列コードを自動生成する並列化コンパイラの開発が挙げられる.

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 pp.2732-2740，2004.

[2] 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.

[3] 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.

[4] 安部拓未，吉田明正: 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情報処理学会第 79 回全国大会，3A-05，2017.

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

[6] 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.