Vol.2011-ARC-197 No.28
Vol.2011-HPC-132 No.28
2011/11/29

動的タスクスケジューリングによるCPU/GPUヘテロジニアス環境でのFMMの最適化

福田圭祐[†1] 丸山直也[†1,†2] 松岡聡[†1,†2]

FMM は、N 体問題を O(N) 時間で近似的に計算するアルゴリズムであり、他の N 体問題のアルゴリズムと比較してスケーラブルであることから近年着目されている．一方で，FMM は異なる計算特性や依存性を持つ複数の計算フェーズからなるアルゴリズムであり、それらを複数の異種プロセッサを持つ環境上で効率よく実行する方法は明らかではない．本稿では、筆者らの過去の発表[18] に引き続き、FMM の実装である kifmm3d の複数のフェーズを GPU 化し、速度向上と実装上の課題について検討した．さらにそれらの実装を元に、動的タスクスケジューリングシステムである StarPU[5] を用いて CPU/GPU からなるヘテロジニアス環境上でプロセッサ資源を有効に利用する試みの初期実装および検討を行い，CUDA スレッドの起動や速度に関する不具合，フェーズ分割および filter の実装に関する検討事項など，様々な技術的問題点や検討事項に関する知見を得た．

Towards Optimizations of FMM on CPU-GPU Heterogeneous Environments using Dynamic Task Scheduling Runtimes

Keisuke Fukuda,[†1] Naoya Maruyama[†1,†2] and Satoshi Matsuoka[†1,†2]

FMM is an O(N) approximative algorithm for N-body problems and recognized more scalable and promising than other N-body computation methods. Effectively utilizing heterogeneous systmes in FMM, however, is a challenging issue because FMM consists of several phases with different performance characteristics that call for careful load balancing for optimal performance. This paper extends our previous work[18] that partially ported the CPU implementation of kifmm3d to CUDA, and presents a complete CUDA implementation. To exploit heterogeneous processing elements, we further extend the implementation with StarPU, which allows dynamic task scheduling on CPU-GPU heterogeneous environments. We have found several technical issues and challenges, such as failing CUDA kernel invocations, phase splitting and implementation of filters, to achieve a good load balancing.

1. はじめに

Fast Multipole Method(FMM) は，Greengard ら[8] によって提案された，N 体問題向けの指定精度の $O(N)$ 計算時間アルゴリズムである．N 体問題向けのアルゴリズムとしては他に，直接計算や $O(N \log N)$ 計算量の Barnes-Hut アルゴリズム，全対全通信を要する FFT ををベースとした方法等が知られている．小規模〜中規模サイズの問題に関しては Barnes-Hut 等の手法の方が効率が良い場合が多いが，大規模問題に関しては計算オーダーが非常に重要となり，かつボトルネックとなる通信量が比較的少ないことが有効なアルゴリズムの条件となる．近年の計算環境と問題の大規模化に伴い，各種の N 体問題の計算においては FMM の重要性が高まっていくと考えられる．

計算環境に関しては，近年 GPU を搭載したスーパーコンピューターが増加している．東京工業大学学術国際情報センターに設置された TSUBAME2.0 はその最たる例であり，理論性能 2.4PFlops を誇る．また，Linpack ベンチマークによる世界ランキング Top500 では，上位 5 台のうち TSUBAME を含めて 3 台が CPU と GPU を併用したものであった[1]．このような環境では，計算性能の大部分を GPU が占める場合が多い．従来の CPU 向けの分散・並列化だけではなく CPU-GPU 異種混在環境を考慮した最適化を行う必要がある．

しかし，既存の並列化・最適化手法だけでは，CPU-GPU 混在環境においては十分でない．CPU ではコアあたり 1 スレッドを前提とし，単ノード上では数〜数十スレッド程度による並列化がなされることが多い．一方，GPU は数百の SP(Streaming Processor) から構成され，数千〜数万スレッドによる並列化がしばしば行われる．さらに，GPU 向けのデータ構造への変換やメインメモリから GPU メモリへのデータ転送のコスト等も無視できない場合がある．

特に，本研究で取り上げる FMM は複数の計算フェーズからなるアルゴリズムであり，それぞれのフェーズがデータ構造・計算方法・並列性・メモリアクセス等の点から異なった特

†1 東京工業大学 Tokyo Institute of Technology
†2 JST/CREST

© 2011 Information Processing Society of Japan

Vol.2011-ARC-197 No.28
Vol.2011-HPC-132 No.28
2011/11/29

徴を持っている．また，入力となる粒子の配置により，各フェーズの計算時間の割合は変動する．CPU-GPU 混在環境で FMM を効率よく実行するためには，このような計算フェーズの特徴と CPU-GPU 間のデータ変換・転送のコストを考慮して CPU-GPU 間での計算の割り当てを動的に決定する必要があるが，その方法論は明らかではない．

また，各種プロセッサは速いスピードで開発が進められており，処理速度や計算特性の点で将来的にも同じ傾向が続くとは限らない．現に，CPU と同じダイ上に両方の種類のプロセッサを搭載した Intel 社の Sandy Bridge や，x86 メニーコアとなる MIC 等の製品が近い将来に科学技術計算において利用可能となると考えられ，異種プロセッサ混在環境を取り巻く環境は大きく変化し続けている．CPU ／ GPU に限らず，プロセッサ特性，性能，ノード構成，ノード数に適応して，自動的に処理を割り当てて最良の性能を得ることができるような手法が必要である．

そのような手法としては，まず手動によるチューニングが考えられる．しかし，対象とする計算の種類・入力データ・実行環境によって様々に変動する負荷のバランスを調整し，プロセッサを無駄なく使うような最適化を常に行うのは現実的ではない．静的な自動チューニング手法を用いることも考えられるが，入力データや実行時の変化により計算時間の動的な変動が見られるような場合には適さない．そこで，実行時にタスクの配分を調整し，動的な調整を行うようなタスクスケジューリングランタイムの利用により，比較的低い労力でさまざまな環境に対応した効率的なアプリケーションの実行が可能になると考えられる．そのようなライブラリやランタイムは複数開発されているが，本稿では GPU に対応したランタイムの 1 つである StarPU[4] を用いる．

本稿では，Ying らによる FMM の亜種である KIFMM[16] の参照実装 kifmm3d に変更を加え，U-list，Upward フェーズを GPU 化した我々の HPC 研究会発表[18] に引き続き，V-list フェーズ・Downward フェーズについても GPU 化を行った．そして，同プログラムを StarPU 上に初期実装した．CPU 版のプログラムを用いて task 間の依存性に基づいてプログラムが適切にスケジューリングされ実行されることを確認し，以下のような技術的課題と検討事項を明らかにした．

- StarPU の組み込みによって，GPU カーネルや CUFFT カーネルが高確率で起動に失敗する現象が起こっており，原因の究明と改善が必要である．
- 現段階で確証はないが，StarPU，CUDA, StarPU 外でのマルチスレッドの 3 者を混合して使う場合，特にスレッドの扱いに考慮を要すると思われる．
- 様々な計算特性からなる複数のフェーズを持つ FMM に対して，History-base のパフォーマンスモデルがどの程度適用可能であるか明らかではない
- プロセッサ間のデータのやりとりに用いる filter の枠組みが FMM にどの程度適用可能であるか明らかでない．特に木構造を GPU 向けに変換するような場合である．

2. 背　　景

本節では背景として，本稿で対象とする Fast Multipome Method と、タスクスケジューリングランタイムである StarPU について概要を述べる．

2.1 Fast Multipole Method

本節では，FMM の概要と，並列処理の観点から見た FMM の特徴，そして各フェーズの計算特性の違いについて述べる．本項で述べるアルゴリズムの数学的詳細，計算フェーズについての詳細についてはいずれも文献[8),16)] を参照されたい．

2.1.1 FMM の概要

FMM の基本的な考え方は，空間を再帰的に木構造に分割し，遠くの粒子を”まとめて”計算することで，全粒子同士の相互作用を近似的に $O(N)$ 時間で計算するということである．特に，KIFMM においては計算は，木構築・Upward・U-list・V-list・X-list・W-list・Downward の各フェーズからなる．

まず木構築フェーズでは，中心点を用いて空間を 8 等分[⋆1] し，分割されたそれぞれの空間について，粒子の個数が q 個を超える場合は再帰的に 8 等分を繰り返す．こうして分割された空間は，それぞれの葉 (または box と呼ばれる) がたかだか q 個の粒子を保持する 8 分木として表現される (図 **2.1.1** [⋆2])．

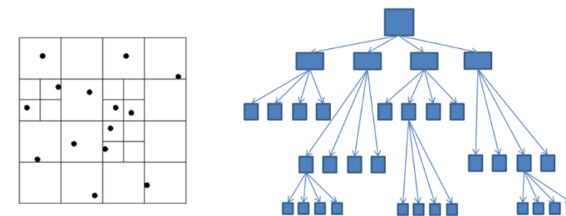

図 1　空間の再帰的分割と木表現

⋆1 1 次元空間においては 2 等分，2 次元空間においては 4 等分である
⋆2 簡単のため，図は 2 次元の場合を示した

© 2011 Information Processing Society of Japan

次に，それぞれの box に属する粒子の情報を，親の節点 (Multipole と呼ばれる) から節点へと頂点へ向かって再帰的に集約していく．この数学的操作は Multipole Expansion，計算フェーズは Upward Computation と呼ばれる．なお，Greengard らによる FMM と Ying らによる KIFMM の主な違いはこの数学的操作にある．FMM では球面調和関数を用いてカーネル関数を数学的に級数展開することによって粒子をまとめることを可能にしている．一方，KIFMM ではポテンシャル理論の成果を用いて，一定の条件を満たす場合に空間を包み込む曲面上に分布した density による作用と内部の点の density による作用の和が同じになるという定理を用いている．

次に，計算する粒子間の距離に応じて 4 種類からなる評価フェーズがある．

まず，近傍の粒子同士の相互作用については近似計算を用いることができないので，直接計算を行う．このフェーズは U-list フェーズと呼ばれる．元の空間上で隣り合っている box 同士の全粒子同士の直接計算が行われる (図 **2.1.1**).

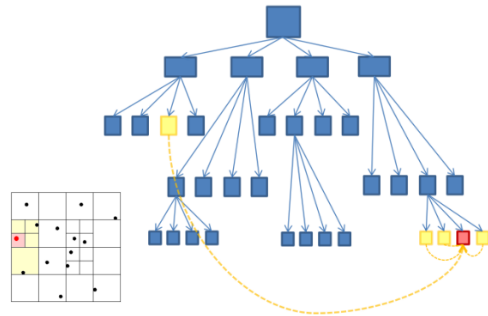

図 2 U-list フェーズの計算例

次に，V-list と呼ばれる計算フェーズでは，親が同じレベルであるような節または box 同士 (ただし互いが U-list である場合をのぞく) の計算が行われる．図 **2.1.1** は，V-list フェーズによって節から節へ相互作用が計算され，それが後述の Downward フェーズによって粒子へ伝播される様子を示している．このフェーズでは，KIFMM においては大量かつ細粒度の FFT が行われる．

その他に W-list，X-list と呼ばれる計算フェーズが存在する．これらのフェーズは，粒子の分散が不均一であり，木構造が平衡木でない場合の処理を行うフェーズである．本研究では均一分散のみを扱い，これらのフェーズは対象としなかった．これは将来の課題である．

最後に，Downward フェーズと呼ばれるフェーズによって，節同士で行われた相互作用の結果が親ノードから子ノードへと展開され，最終的に末端の個々の粒子の値へと伝播される．

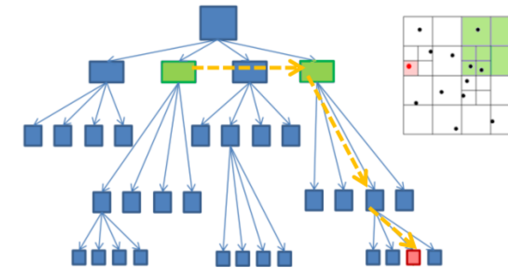

図 3 V-list フェーズの計算例

2.2 StarPU の概要

CPU と GPU を併用したヘテロジニアス環境上において，FMM のようなアプリケーションを効率よく実行するためには，プロセッサ資源を無駄なく利用する最適化が欠かせない．そのための方法としては，まず手動によるチューニングが考えられる．しかし，対象とする計算の種類・入力データ・実行環境によって様々に変動する負荷のバランスを調整し，プロセッサを無駄なく使うような最適化を常に人間が行うのは現実的ではない．また静的な自動チューニングは，一般的な線形代数演算のように，計算にかかる時間が入力データの内容によらない場合によく用いられ，BLAS・FFT 等を異なるプロセッサに対して高度に最適化するような実装が知られている[3)12)]．しかし，本項で取り上げる FMM のように計算フェーズ間の負荷のバランスが実行時にしかわからず，また実行中にもタイムステップを経る粒子移動によって変動するような場合には，このような静的な自動チューニングは適さない．そこで，実行時に計算の配分を調整し最適な負荷バランスを達成するような動的タスクスケジューリングランタイムの利用より，比較的低い労力でさまざまな環境に対応した効率的なアプリケーションの実行が可能になると考えられる．動的タスクスケジューリングを行うライブラリやランタイムは複数開発されているが，本稿では GPU に対応したランタイムの 1 つである StarPU を用いる．

© 2011 Information Processing Society of Japan

StarPU は，1 つは task と呼ばれる処理単位を基本とし，それらを異種・複数プロセッサに配分するタスクスケジューリングシステムである．内部的には大まかには 2 つの構成要素からなる．実行時に各プロセッサの負荷を考慮し，場合によってはモデルに基づいてパフォーマンス予測を行い task 実行をスケジューリングするタスク管理システムと，task が各プロセッサで実行されるためのメモリの転送や一貫性管理等の処理を受け持つメモリ管理システムである．

まずタスク管理システムについて述べる．前述のように，StarPU では処理を”task”と呼ばれるかたまりで管理する．この処理には，codelet と呼ばれる構造体を設定する．これは，必要な処理を行う各種プロセッサ向けの関数を集めたものである．たとえば，CPU 用，CUDA 用，OpenCL 用などが考えられる．そして，codelet を保持した task に，task 同士の依存関係，必要なデータの指示，そしてパフォーマンスモデルを与えてランタイムエンジンに登録する．

次にメモリ管理システムについて簡単に述べる．GPU をはじめとして PCI-express バス等で接続されるプロセッサは，CPU とは独立した階層的なメモリを持つ．これらのメモリ上に配置（もしくは複製）されたデータを適切に管理することは，不可能ではないにしても大変難しく，またコストもかかる．そこで StarPU では”filter”と呼ばれる概念を導入することによってこの処理を簡略化することを狙っている．プログラマがデータについて知識を持っていること仮定し，メモリの部分切り出し等のデータ変換を BLAS 等のインターフェースを組み合わせた filter を用いることで行う．これにより，分割されたデータ同士が共通部分を持たない（disjoint）であることを保証し，同時にプログラマによるメモリ管理にライブラリが必要以上に干渉することを避けている．また，逆変換が可能であり，メインメモリへの書き戻しも透過的に行えることも利点である．

StarPU は，例えば LU 分解のパフォーマンスにおいて MAGMA ライブラリ[14] に対して 25 %程度劣るだけであることが示されている[5]．これは，MADAM ライブラリが人手によるチューニングと静的自動チューニングを組み合わせた非常に高速なライブラリであり，StarPU は動的にタスクスケジューリングを行うライブラリであることを考えると非常に高い性能であるといえる．さらに，StarPU は 1GPU と 3CPU というように，ヘテロジニアスな構成においても動的負荷分散を行い，利用可能な資源の利用効率を最適化することができる．さらに，MAGMA や PLASMA[3] と StarPU を組み合わせてタイル型 Choleskey 分解を行う研究等も行われている[2]．

StarPU システムの詳細については[5] を参照されたい．

3. V-list/Downward フェーズの GPU 実装

本節では，kifmm3d の V-list フェーズと Downward フェーズの GPU 化について述べる．

3.1 V-list フェーズ

本研究ではまず V-list フェーズの GPU 化を行った．V-list は，FMM の木構造におけるノードごとに並列性を持つ処理であり，それ自体は高い並列性を持ち並列化が容易であると言える．本稿での実験粒子数においては，ノード数は 3 万前後であり，GPU 化にあたっても十分な並列数であると言える．

しかし，V-list はノードごとに FFT を含むフェーズであることが課題となる．具体的には，P^3 程度の要素数の FFT と逆 FFT を行う必要がある (ただし，P はプログラム開始時にユーザーが指定する精度に関するパラメーターであり，4,6,8 などの値を取る．本実験では 4 を使用している）．これは FFT のサイズとしては小さく，全体としてはこれをノード数だけ（つまり 3 万回前後）行う必要がある．CUDA において FFT を行うには，通常は既存のライブラリを利用し，著名なものとしては，NVIDIA 社による CUFFT や非常に高速であることが知られている NukadaFFT[12] 等が知られている．しかし，これらは CPU 側から呼ばれることが想定されており，かつ大きなサイズの FFT を計算することが想定されている．このため，計算としてはノード数分の十分な並列性を保持しているにも関わらず，FFT 呼び出しのたびに CUDA のカーネルを終了し CPU 側から for 文を用いて CUFFT 関数を呼び出すことになる．V-list フェーズには 2 回の FFT に呼び出しが含まれているので，GPU 化に際して障害となる．本稿ではこの手法を採用したため，並列数から想定されるほどの性能向上を達成することはできなかった．これを解決するためには，CUDA において device 関数として呼び出すことができ，我々が実装した前後の関数との親和性のためにノードごとに 1 つの ThreadBlock を使用し，小規模な FFT に最適化した実装を用意する必要がある．これは将来の研究課題とした．

3.2 Downward フェーズ

次に，Downward フェーズの GPU 化について述べる．Downward フェーズは，大きく分けて木構造を辿りながら下方向に計算行うフェーズ（L2L とも呼ばれる）と，最後に木の葉から末端の粒子までの計算を行うフェーズ（L2T とも呼ばれる）に分けられる．今回は，後者のみを GPU 化した．前者については，将来的に本実装を MPI 化することを予定している点から，木の分割に際して CPU 側での処理が必須になることから見送った．

L2T フェーズは，並列数も高く比較的単純なカーネルであるが，レジスタの使用数が多

© 2011 Information Processing Society of Japan

く，また CPU で行う GPU のためのデータ変換コストがフェーズ全体に比して大きいことが影響し，これについてもさほどの性能向上は見られなかった．これについては，CPU 側でのデータ変換部分も含め，さらなる最適化の余地があると考えている．

3.3 既存 GPU 実装と CPU 実装

CPU による並列化実装，Upward フェーズの GPU 実装，U-list フェーズの GPU 実装については我々の HPC129 における発表[18] において実装済みであるのでそれを用いた．CPU 上での並列化には OpenMP を用い，計算フェーズ内の依存性に留意しつつプログラムに存在する for ループを並列化した．

4. kifmm3d の StarPU 上への実装

本節では，kifmm3d を StarPU 上で実行するためのプログラムの変更点について述べる．

StarPU は，ユーザー空間で動作するライブラリとして実装されている．実行したい処理を starpu_task という構造体として作成し，関数ポインタを設定することにより関数をタスクとして実行できる（なお，CUDA カーネル関数であっても，ラッパーとなる CPU 関数を用意し，間接的に呼び出すことになる）．starpu_task 構造体には，CPU 用の関数と CUDA 用の関数の両方を設定することができ，どちらの関数をどのように実行するかを設定することができる．これにより，どちらの関数を実行するかをランタイムの判断に委ねることができる．今回は，CUDA 実装を用意してある U-list，Upward，V-list，Downward については両方の関数を設定してランタイムに判断をゆだね，W-list と X-list については CPU 関数のみを指定した．ソースコードの記述量については，本実装では各フェーズごとに既存の関数を呼び出すだけの機能を持った task を作成し launch する作業を W,X を含めた 6 つ全てのフェーズについて行い，全体としては 200 行程度のコード追加となった．

task 間には依存関係を設定することができるので，本稿では以下のような依存関係を設定した．まず，U-list は周辺粒子との直接計算であるから，木構造上での処理（Upward，V-list，W-list, X-list, Downward）とは独立である．ただし，個々の粒子への計算結果の書き込み時に，Downward との競合が存在している．よって，$Ulist \rightarrow Downward$ という依存関係を設定した（つまり，U-list の方が必ず先に実行され，その終了を待って Downward が実行される）．次に，木構造の処理については，フェーズ間の自明な依存性にしたがって $Upward \rightarrow Vlist \rightarrow Wlist \rightarrow Xlist \rightarrow Downward$ とした．

なお，データ管理機構と性能モデルに関しては，次項以降で述べるような検討事項があるため，将来の課題とした．

表 1 評価環境 1

	CPU	GPU
Type	Intel 6 core Xeon X5670 × 2	NVIDIA Tesla M2050 × 3
Freq	2.93GHz	1.15GHz
Cores	6 x 2 (with HT)	14SM/448 cores
Memory	54GB	3GB

表 2 評価環境 2

	CPU	GPU
Type	Intel core i7 (4 cores) × 1	NVIDIA GTX580 × 1
Freq	2.67GHz	1.536 GHz
Cores	4 x 2 (with HT)	16SM/512 cores
Memory	6GB	1.5GB

5. 評　価

5.1 GPU 実装の評価

CPU 実装と GPU 実装の**表 1**, **表 2** の 2 種類の評価環境における実行時間の評価結果**図 5.1** に示す．なお，CPU の OpenMP による並列化については，実験環境上の CPU1 コアに対して 1 スレッドが割り当てられるように numactl コマンドを用いて制御し，HyperThreading の有無に関わらず物理コアあたり 1 つのスレッドが割り当てられるように調整した．入力データとしては，粒子数 100 万の均一分布を用いた．

© 2011 Information Processing Society of Japan

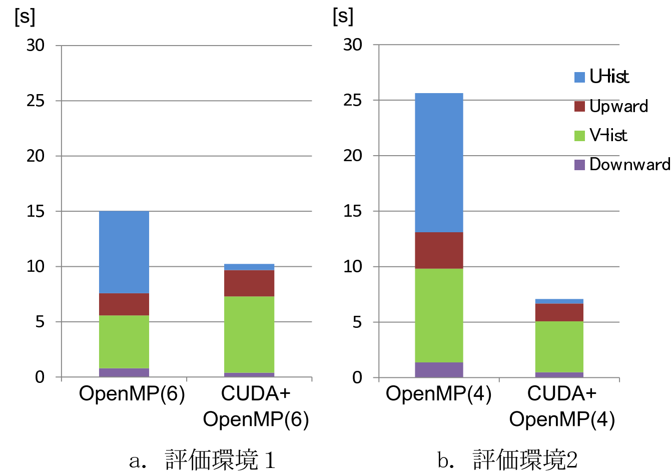

図 4 2 つの評価環境における GPU と OpenMP 実装の評価

2 つの評価環境は，CPU・GPU ともに異なる型の製品を採用しており，それぞれにパフォーマンスが異なることが確認できる．まず GPU については，コンシューマー向けのの上位モデルである GTX580 が全般的に上回っていることが確認できる．特に，ここで評価している全てのフェーズにおいて，4-core の Core i7 を上回るパフォーマンスを発揮し，大きな速度向上を果している．しかしこれは CPU が遅いことによるもので，評価環境 1 においては逆の様子が見られる．6 コアの Xeon プロセッサは高いパフォーマンスを発揮し，U-list 以外のフェーズにおいて全て GPU を上回っている．このことからも，CPU・GPU 製品によってもそれぞれのフェーズの計算時間は異なり，最適な負荷のバランスも異なることがわかる．ただし，ソースコード上の細かいチューニングによって高いパフォーマンスを達成すること自体は本稿の趣旨ではないため，CPU/GPU 共に実装には最適化の余地が残されいてることを注釈しておく．

5.2 StarPU 実装の評価

前述のように，CUDA 関数を用いた実装は高頻度で CUDA カーネルの起動に失敗し正しく動作させることができなかったので，正しく動作する CPU 関数のみを用いて予備評価を行った．StarPU によって task のオーバーラップ実行が行われるかを確認するため，これらの関数から OpenMP による並列化を取り除いて各フェーズを逐次実行とし，実行時間を計測した．

表 3 CPU 版逐次実行と StarPU の比較 (単位:秒)

	逐次実行	StarPU 実行
U-list	48.20	48.70
Upward	10.42	10.46
V-list	30.34	30.72
Downward	3.96	3.96
全体	93.16	53.10

表 3 は、全フェーズを逐次に実行する場合と，StarPU により実行する場合の比較である．入力データは 5.1 と同じものを用いた．なお，StarPU は評価環境 1 では正しく動作しなかったため，評価環境 2 を用いた．

ここで，各フェーズの実行にかかる時間は同じであるが，StarPU 実行においては U-list とその他のフェーズが依存関係の解析から自動的に並列に実行され，全体の実行時間が短縮されていることが確認できた．

次項以降では，実装上問題となった諸課題について述べる．

6. StarPU 上へ kifmm3d を実装する際の課題と検討事項

本節では，まず実装を行う上で障害となった問題点について述べ，その次に今後への課題と検討事項を述べる．

6.1 今回見られた実装上の障害

6.1.1 StarPU ランタイムの導入による全体的な速度低下

StarPU を利用するためには，最初にランタイムの初期化を行う必要があるが，StarPU によるタスク管理を実装する以前の時点で本稿のプログラムを構成する CUDA カーネルの実行速度が落ちるという問題が発生した．カーネルにより若干の違いはあるが，CUFFT 呼び出しも含めて平均的に 2 倍前後遅くなった．これが StarPU システムの問題によるものか，本稿実装の問題であるかは現時点で明らかではない．

6.1.2 StarPU 経由での CUDA カーネル時の実行失敗

StarPU ランタイム経由で CUDA カーネルを実行する際に，頻繁にカーネル起動に実行に失敗するという現象が見られる．特にカーネル呼び出し後の cudaThreadSynchronize 関数の戻り値チェックでも補足できない場合もある．本稿では，カーネル実行時間を計測して

ⓒ 2011 Information Processing Society of Japan

一定以上短い場合，書き出しメモリ領域をチェックした上で失敗とみなして再実行するという実装を行った（出力メモリ領域は書き出し専用になるように修正した）．これにより失敗の頻度をある程度削減できたが，完全になくすことはできず性能評価を行うには至らなかった．前項の点と同様に，これが StarPU 側の問題なのか，本稿プログラムの問題なのかは現時点では不明である．なおプログラムを変更して CPU 関数のみで StarPU による実行を構成した場合，正しく実行されることを確認している (表 3).

また同様の問題として，StarPU はホームディレクトリに実行環境の性能モデルデータを持つのでそれが存在しない場合まず最初に計測を行うが，計測を行ったプロセスのその後の計算で CUFFT の実行が失敗するという現象も見られた．これは一度モデルデータが生成されれば二度目以降は行われないため上記に比べれば致命的な問題とはならないものの，原因究明と解決が必要である．

今回の実装にあたっては，CUDA ランタイムとスレッドの関係の衝突を疑い OpenMP を無効にするなどの対策を取ったが，現象の改善には至らなかった．これらの問題点については今後も StarPU 関係者とも情報を交換しながら問題の解決に努めていく．

6.2 今後への課題と検討事項

6.2.1 各種バグの原因究明と修正

今後の課題して，前述のように GPU カーネルが正しく実行されない問題があるので，この原因究明を行い，正しく動作するように修正する必要がある．現時点では，本プログラムは単体では正しく動作しており，また StarPU も一定の実績があるため，どちらの問題であるか等の原因については明らかでは無い．

6.2.2 パフォーマンスを得るためのフェーズの分割

今回は各フェーズを 1 つの単位として StarPU の task を作成し，それらを依存関係で接続して実行した．しかし実際には，このような方法では分割の粒度が粗すぎると考えられる．本稿での結果では，例えば評価環境 2 において U-list の CPU 実行が約 12.5 秒，GPU 実行が約 0.4 秒であり，V-list ー Downward から構成される一連の木探索フェーズについてはそれぞれ約 13.1 秒，約 6.66 秒である．近傍粒子同士の直接計算を行う U-list フェーズと一連の木探索フェーズが互いに独立である事を踏まえると，U-list を GPU で実行する場合，木探索フェーズを CPU で実行してオーバーラップしたとしても大きな性能向上は期待できない．逆に，U-list を CPU で実行する場合は一連の木探索フェーズを GPU で実行しても大幅な性能低下が起こってしまう．

よって，性能向上を図るためには個々のフェーズを分割してより細かい依存関係を設定することが必要であると考えられる．例えば，評価環境 1 において CPU 実行で 4.7 秒・GPU 実行で 6.9 秒を要する V-list フェーズは，フェーズ内では木のノード数の並列性があり，容易に分割が行える．分割した V-list フェーズを CPU と GPU で分担すれば実行時間の削減が期待できる．

6.2.3 タスクスケジューリングとパフォーマンスモデル

現状では，パフォーマンスモデルを与えていないため，CPU 版と GPU 版の呼び出しはシステムによって特に基準なく選ばれてしまう．パフォーマンスモデルは，StarPU が task を投入する際の参考とされるため，高効率なスケジューリングのためには必須である．

StarPU におけるパフォーマンスモデルの与え方には 2 通りの方法がある．第 1 は History base と呼ばれる過去の実行実績からシステムが自動的に実行時間を推定する方法であり，第 2 は人間がパフォーマンスモデルを構築して与える方法である．参考文献[5] では，線形代数アルゴリズムの実装であれば History base により十分な精度で実行時間を予測できることが述べられている．History-base の方法によって十分な精度が得られるのであれば労力の観点からそちらのほうが望ましいが，精度が不十分な場合は手動モデルを導入する必要がある．kifmm3d は計算特性の異なる複数のフェーズから構成されているため，history base のパフォーマンスモデルがどの程度適用可能であるかは明らかではなく，今後の課題である．

6.2.4 データ変換のタイミングの検討

参考文献[5] で挙げられている StarPU の適用事例において，データの変換は”部分的な切り出し”や”転置”のような，線形変換によって実装できるものが主であった．しかし，kifmm3d においては木構造に含まれているデータを GPU で処理しやすいように変形処理を行う必要があり，特に木の探索を含む．また，その他多数の係数等を引数として渡す必要もあり，これらの処理を既存の StarPU の filter の枠組みでどのように実現するかは明らかではない．

具体的な方法としては，第一にデータ構造の変換そのものを処理の一部としてしまい，CUDA 処理としながらも CPU によるデータ変換も含めてしまう方法，第二にユーザー定義のフィルタとして実装する方法が考えられる．

また，前述のように CUDA カーネルがうまく起動しないことの原因が CUDA ランタイムと OpenMP 等のスレッディング機構の相性の問題である場合，スレッドを用いずに完全に StarPU のみで並列処理を行う必要がある．データ変換には無視できない時間がかかるため，データ変換処理を CPU でのみ実行可能な task の 1 つとして依存関係に組み込み，分割／並列処理する必要がある可能性も考えられる．

© 2011 Information Processing Society of Japan

6.2.5 マルチ GPU

StarPU ではタスク単位で独立したメモリ領域管理を行うため，task ごとに別々の CPU/GPU での実行が容易に可能である．特に，TSUBAME2.0 の単ノードは CPU 2 個・GPU 3 個からなるため，フェーズ間の負荷の不均衡を適切に配分して，プロセッサリソースを有効に活用できる可能性がある．

6.2.6 粒子の不均一分散

本稿においては，粒子の分散に均一分散を仮定し，X-list フェーズ，W-list フェーズについては GPU 化を行なっていない．しかし StarPU による負荷分散機能が大いに発揮されるのは，むしろ不均一な粒子分布におけるフェーズ間の負荷の偏りと予測不能性である．これに対する StarPU の有効性を検証するため，X-list，W-list フェーズについても GPU 化を行い，StarPU による負荷分散が有効に働くかどうかを検証することが必要である．

6.2.7 マルチノード化

ここまで，単ノード上でのタスクスケジューリングとロードバランシングについて述べたが，大規模なシミュレーションを行うためには MPI によるノード間分散処理が欠かせない．StarPU はノード内での処理を想定して開発されており，MPI を用いたノード間での負荷分散は実装されていない．よって，StarPU と MPI によるデータ分散を組み合わせて全体で最適な負荷分散を実現する必要があり，その具体的な手法はこの大きな課題である．

7. 関 連 研 究

Lashuk らは，GPU 上での KIFMM の実装に取り組み，逐次実装に対して 30 倍を超える大幅な高速化を得ている[11]．Chandramowlishwaran らは，KIFMM を複数種類のマルチコア CPU 上で高速化する研究を行った[6),7]．また，それらの成果を総合し，Vuduc らは，KIFMM においては高度に Nehalem 等のプロセッサ上で高度に最適化された CPU 実装は GPU 実装に匹敵するとの見解を示した[15]．本稿での GPU 実装については，Lashuk らの成果を参考にした部分が多い．さらに，kifmm3d を実際のアプリケーションに応用した例として Rahimian らの研究[13] が挙げられる．

横田らは，FMM を応用して GPU クラスタ上での乱流解析を実装した[17]．

FMM の他に N 体問題の高速化法として広く用いられているアルゴリズムに，$O(N \log N)$ 計算量である Barnes-Hut の Tree アルゴリズムがある．Jetley らは，Barnes-Hut アルゴリズムの実装である ChaNGa を GPU 上に移植し，その性能を調べた[10]．Hamada らは，安価に構築した大規模な GPU クラスタ上で Barnes-Hut アルゴリズムを実装し，2009 年の ACM ゴードン・ベル賞を受賞している[9]．

8. まとめと今後

本稿では，FMM の派生アルゴリズム KIFMM の実装 kifmm3d のフェーズを GPU 化し，それらを動的タスクスケジューリングエンジン StarPU の下で動かすことを試み，その経過を報告した．実装の課程で，多数の技術的な問題点が見つかると同時に，効率的なアルゴリズムの実装のために検討すべき点も得られた．今後は，未だ GPU 化をしていないフェーズもすべて GPU 化すると同時に StarPU 上での kifmm3d の実装における種々の技術的課題を解決し，動的なタスクスケジューリングによる最適実行を目指して研究を進めていく．特に，入力データである粒子の分布が様々に変化する場合に，人間による個々のケースへのチューニングを行うことなく，自動的にリソースを最大限有効活用できるようになることが目標である．

さらに将来的には，単ノードの実行にとどまらず MPI を併用した分散環境での大規模なシミュレーションの最適実行に関する研究も行っていく．

参 考 文 献

1) Top500 supercomputing sites, September 2010. `http://www.top500.org/`.
2) E.Agullo, C.Augonnet, J.Dongarra, H.Ltaief, R.Namyst, J.Roman, S.Thibault, and S.Tomov. Dynamically scheduled cholesky factorization on multicore architectures with gpu accelerators. 2010.
3) Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov. Numerical linear algebra on emerging architectures: The plasma and magma projects. *Journal of Physics: Conference Series*, Vol. 180, No.1, p. 012037, 2009.
4) Cédric Augonnet and Raymond Namyst. Euro-par 2008 workshops - parallel processing. In Eduardo César, Michael Alexander, Achim Streit, JesperLarsson Träff, Christophe Cérin, Andreas Knüpfer, Dieter Kranzlmüller, and Shantenu Jha, editors, *Euro-Par 2008 Workshops - Parallel Processing*, chapter A Unified Runtime System for Heterogeneous Multi-core Architectures, pp. 174–183. Springer-Verlag, Berlin, Heidelberg, 2009.
5) Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurr. Comput. : Pract. Exper.*, Vol.23, pp. 187–198, February 2011.

© 2011 Information Processing Society of Japan

6) A.Chandramowlishwaran, S.Williams, L.Oliker, I.Lashuk, G.Biros, and R.Vuduc. Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pp. 1 –12, april 2010.
7) Aparna Chandramowlishwarany, Kamesh Madduri, and Richard Vuduc. Diagnosis, tuning, and redesign for multicore performance: A case study of the fast multipole method. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–12, Washington, DC, USA, 2010. IEEE Computer Society.
8) L.Greengard and V.Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, Vol.73, pp. 325–348, December 1987.
9) Tsuyoshi Hamada, Tetsu Narumi, Rio Yokota, Kenji Yasuoka, Keigo Nitadori, and Makoto Taiji. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pp. 62:1–62:12, New York, NY, USA, 2009. ACM.
10) Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V. Kalé, and ThomasR. Quinn. Scaling hierarchical n-body simulations on gpu clusters. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–11, Washington, DC, USA, 2010. IEEE Computer Society.
11) Ilya Lashuk, Aparna Chandramowlishwaran, Harper Langston, Tuan-Anh Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, Denis Zorin, and George Biros. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pp. 58:1–58:12, New York, NY, USA, 2009. ACM.
12) Akira Nukada and Satoshi Matsuoka. Auto-tuning 3-d fft library for cuda gpus. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pp. 30:1–30:10, New York, NY, USA, 2009. ACM.
13) Abtin Rahimian, Ilya Lashuk, Shravan Veerapaneni, Aparna Chandramowlishwaran, Dhairya Malhotra, Logan Moon, Rahul Sampath, Aashay Shringarpure, Jeffrey Vetter, Richard Vuduc, Denis Zorin, and George Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–11, Washington, DC, USA, 2010. IEEE Computer Society.
14) Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid gpu accelerated manycore systems. *Parallel Comput.*, Vol.36, pp. 232–240, June 2010.
15) Richard Vuduc, Aparna Chandramowlishwaran, Jee Choi, Murat Guney, and Aashay Shringarpure. On the limits of gpu acceleration. In *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, HotPar'10, pp. 13–13, Berkeley, CA, USA, 2010. USENIX Association.
16) Lexing Ying, George Biros, and Denis Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, Vol. 196, No.2, pp. 591 – 626, 2004.
17) R. Yokota, T. Narumi, R. Sakamaki, S. Kameoka, S. Obi, and K. Yasuoka. Fast multipole methods on a cluster of gpus for the meshless simulation of turbulence. *Computer Physics Communications*, Vol. 180, No.11, pp. 2066 – 2078, 2009.
18) 福田圭祐, 丸山直也, 松岡聡. Cpu/gpu ヘテロジニアス環境における fmm の最適化. 情報処理学会第 129 回 HPC 研究会, Vol. 129, , Mary 2010.

© 2011 Information Processing Society of Japan