データ同化における衛星熱解析のGPGPUによる高速化試行

高木 亮治*、秋田剛†

Application of GPGPU to thermal analysis used in data assimilation

by

Ryoji Takaki* and Takeshi Akita†

Abstract

A thermal mathmatical model plays an important role in operations on orbit as well as spacecraft thermal designs. The thermal mathematical model has some uncertain thermal characteristic parameters, which discourage make up efficiency and accuracy of the model. A particle filter which is one of successive data assimilation methods hase been applied to construct spacecraft thermal mathematical models. This method conducts a lot of ensemble computations, which require large computatilnal power. Recently, General Purpose computing in Graphics Processing Unit (GPGPU) has been attracted attention in high performance computing. Therefore GPGPU is applied to increase the computational speed of thermal analysis used in the particle filter. This paper shows the speed-up results by using GPGPU as well as the application method of GPGPU.

1. はじめに

衛星開発および運用では、適切な熱設計を行うことが重要であり、精度の高い熱数学モデルを構築する必要がある。熱数学モデルは熱伝導係数、熱容量、輻射係数など様々な物理パラメータが必要となり、これらのパラメータの値は基礎的な試験で取得された値を用いるが、接触熱抵抗のように実機の製作工程に依存するなど、値が正確に予測できないものがある。そのため、最終的には熱真空試験を行い、試験結果と熱数学モデルのコリレーションをとる。このように熱数学モデルに使われる物理的なパラメータは熱真空試験結果を用いて推定することになるが、このパラメータ推定には不確定性が強く、経験者による試行錯誤が必要となる。これらの試行錯誤には多大な労力と時間が必要とされ、衛星開発期間の短縮やコスト削減が求められるなか、より効率的で精度の高い推定方法が望まれている。

近年、物理現象に対して数学モデル（とその数値シミュレーション）および観測データを統一的に融合する手法としてデータ同化[1)]と呼ばれる手法が提案されている。データ同化を適用することで熱真空試験結果と熱数学モデルのコリレーションを高効率かつ高精度で行うことが可能と考えられ、衛星熱設計へのデータ同化手法の適用が試みられ、その有効性が確認されつつある[2, 3, 4)]。これらの試みでは、取り扱う物理が熱伝導や輻射を伴った熱現象であり非線形的な現象である。そのためデータ同化手法のうち、非線形システムを取り扱うことが可能なアンサンブルカルマンフィルターや粒子フィルター[5, 6)]と呼ばれる手法が使われている。これらの手法は多数の実現値 (アンサンブル) を用いて統計処理を行うため、多量の解析 (ここでは熱解析) を実施することになる。流体解析等と比較して熱解析は計算負荷が比較的軽いが、多量のアンサンブル解析を実施するのは容易ではなく、高性能な計算環境が必要となる。

一方、高性能な計算環境として GPGPU (General Purpose computing on Graphics Processing Unit) が現在注目を集めている。GPGPU は高い演算性能を低コストで得られる新しい並列計算用ハードウェアとして注目を集めており、世界トップクラスの性能を有するスーパーコンピュータシステムで採用されるなど、その利用が進められている。

本報告では、逐次データ同化手法である粒子フィルターを用いた衛星熱解析を実施する際に必要となる多量の熱解析を、GPGPU を用いて高速に実施することを試みたのでその結果について報告する。

2. 衛星の熱数学モデル

衛星の熱数学モデルは、衛星を構成部品である構体パネルや搭載機器などをいくつかの要素に分割し、各要素単位に熱特性（温度、比熱、熱伝導係数、輻射特性など）を代表する節点を設けることで構築される。太陽輻射、アルベド、地球赤外放射などの外部からの熱入力源や搭載機器からの発熱などによる内部熱入力もそれぞれ節点として考えることができ、これら節点間の熱交換を記述することで支配方程式が求められる。

$$C_i \frac{dT_i}{dt} = Q_i - \sum_{j=1}^{N_n} C_{ij} \left(T_i - T_j\right) - \sum_{j=1}^{N_n} \sigma R_{ij} \left(T_i^4 - T_j^4\right) \qquad (1)$$

ここで、C_i, T_i, Q_i は節点 i の熱容量 $[J/K]$、温度 $[K]$、内外の熱入力 $[W]$ である。C_{ij} は節点 i, j 間の熱コンダクタンス $[W/K]$、R_{ij} は輻射係数 $[m^2]$、σ は Stefan-Boltzmann 係数（$5.669 \times 10^{-8}[W/m^2/K^4]$）である。$N_n$ は総節点数であり、N_n 個の支配方程式を連立させて解くことで各節点での温度を求めることができる。熱コンダクタンスは節点 i, j が同一物体内の場合は物体の熱伝導率で表される。一方、節点 i, j が異種物体である場合は、接触熱伝達率で表される。一般に接触熱伝達率は接触圧力など衛星組み立て、運用時の様々な外的要因によって大きく変化する可能性があり、一般には実機を用いた熱真空試験データを使って値を推定する必要がある。

*宇宙航空研究開発機構 宇宙科学研究所/情報・計算工学センター
†宇宙航空研究開発機構 情報・計算工学センター

2.1 データ同化を用いた熱数学モデルのパラメータ推定手法

データ同化 (data assimilation)[1] は 1990 年代中頃から気象学や海洋学の分野で発達した手法であり、物理シミュレーションモデルと実際の観測を統合する手法 (方法論) である。物理シミュレーションモデルには、モデルの不完全性や初期条件、境界条件が正確にわからないなどの不確かさが存在するため、物理シミュレーションのみでは適切に物理現象を再現できない場合がある。一方観測データは物理的、社会的制約のために得られる情報が十分でないことが多い。データ同化では物理シミュレーションモデルに実際の観測データの情報を組み込むことで、実際の現象をより良く再現する信頼性の高い物理モデルを構築することを目的とする。データ同化は、既に気象予報の精度向上などの目的で応用されているほか、更に様々な分野での応用が検討されている。

データ同化では、まず取り扱う対象を支配する変数を状態変数ベクトル x_t とし、x_t を用いてシステムモデル (一般に物理現象を表現するモデル) と観測モデル (観測される情報を表現するモデル) を以下の様に記述する。これらを状態空間モデルと呼ぶ。

$$x_t = f(x_{t-1}) + v_t \tag{2}$$

$$y_t = h(x_t) + w_t \tag{3}$$

ここで v_t はシステムノイズと呼ばれ、システムモデルの不確かさを表現する変数である。また w_t は観測ノイズと呼ばれる。実際の観測では、現象の一部が観測され、しかも観測時に非線形変換を受ける場合もある。逐次データ同化では観測値 y_t を取得する度に x_t の条件付確率分布または値の推定を行う。条件付確率分布では 3 種類の分布（予測分布、フィルター分布、平滑化分布) が重要な役割を果たし、逐次型データ同化ではこれらを時間ステップ毎に求めていく事になる。ちなみに、予測分布は $t-1$ までのデータに基づく t の状態 (昨日までのデータに基づく今日の状態) の分布、フィルター分布は t までのデータに基づく t の状態 (今日までのデータに基づく今日の状態) の分布、平滑化分布は T までのデータを用いた t の状態 (数年後、データを全て取得したもとで振り返った今日の状態) の分布である。

逐次型のデータ同化では、これらの条件付き確率分布を求めることになるが、対象となるシステムの特性に応じて様々な手法がある。非線形システムにおいては、確率分布を多数の実現値 (アンサンブル) で近似する Ensemble Kalman Filter (EnKF) や Particle Filter (PF:粒子フィルター) が利用される。PF は確率分布のアンサンブル近似に基づく手法の一つであるが、システム自体やシステムの状態と観測との関係に対する線形性および Gauss 分布の仮定を必要としないため、適用範囲が非常に広い。しかしながら、これらの方法は多数のアンサンブルを用いて確率分布を表現する必要があり、多量のアンサンブルの計算、つまり熱解析を行う必要がある。ここでは、多量の熱解析を高速に実施するために GPGPU を用いた並列計算を試みた。以下では GPU の概略に触れた後、熱解析の多量計算を GPGPU を用いて如何に高速化を行ったかに関して述べる。データ同化を用いた熱数学モデルのパラメータ推定手法の詳細については文献 [2] を参照のこと。

3. GPGPU による高速化

近年、高い演算性能を低コストで得られる新しい並列計算用ハードウェアとして GPU が注目を集めている。GPU はもともと画像処理用の演算装置であったが、相対的に簡単な構造を持っているため、CPU の性能向上率を上回る GPU の性能向上や Nvidia 社により GPU の開発環境 CUDA(Compute Unified Device Architecture)[7] が一般に公開されるなど、GPU を一般的な計算、特に科学技術計算に利用する GPGPU（GPU による汎用計算）が注目されるようになり、GPGPU を用いた計算科学の研究や利用技術自体に関する研究が盛んに行われている。さらに世界トップクラスの性能を有するスーパーコンピュータの多くに採用され、スーパーコンピュータのランキング Top500(2010 年 11 月時点) ではトップ 10 のリストの中で 1 位、3 位、4 位のシステムが GPU を利用するシステムである。

3.1 GPU の概要

GPU を用いて計算を実施する場合、GPU の特徴を踏まえた上で利用することが必要である。CPU と比較した場合の GPU の特徴としては、まず計算コアの数が圧倒的に GPU が多い事である。Intel 社の CPU である Xeon X7560 では 8 コアが搭載されているが、Nvidia 社の GPU である Tesla C2070 では 448 CUDA コアを有する。CPU のコアと GPGPU のコア (CUDA コア) が同じ性能・機能を有するわけではなく、例えば、コアのクロック周波数を比較すると、CPU は 2GHz から 3GHz の高クロックであるのに対して GPU では 1GHz 程度と低いクロックである。また、GPU のコアは、それぞれのコアが独立に計算を行うのではなく、同じ計算 (命令) を行う SIMD(Single Instruction Multiple Data) となっている。GPU は単純なコアを沢山搭載することで演算性能を高めており、Xeon X7560 のピーク性能が 72.5Gflops であるのに対し、Tesla C2050 では 1.03Tflops(単精度)、515Gflops(倍精度) となっている。ちなみに単一コア（Xeon X7560 のコアと Tesla C2050 の CUDA コア）のピーク性能を比較すると、X7560 は 10.64Gflops に対して C2050 は 1.15Gflops となり、GPU ではより多くの並列度が必要となる。また GPU では単精度と倍精度でピーク性能が違うことや、単純で高並列な計算は得意であるが、分岐が多いなど複雑で低並列な計算は苦手であるといった特徴も有する。

メモリに関しては、GPU はグラフィック処理用に開発された高速なメモリ GDDR SDRAM を搭載しているが、搭載容量は CPU に比べて多くはなく、大規模な計算でメモリを多く必要とする場合は注意が必要となる。GPU は CPU とは独立にメモリを持っており、GPU で計算を行う場合は、計算で使うデータを CPU のメモリから GPU のメモリに移動するなど CPU と GPU の間でデータ通信を行う必要がある。計算に必要なデータを CPU から GPU に転送し、GPU で計算を実行した後、結果を GPU から CPU に書き戻す必要がある。CPU-GPU 間のデータ通信は一般的に PCI-Express バスが使われるが、GPU 内での通信性能と比較すると低い通信性能となり、頻繁に CPU と GPU でデータのやりとりを行う場合は GPU の高い演算性能を活かせない場合もある。GPU は高い演算性能を持っているが上記のような特性を持っているため、それらの特性を理解した使い方が必要となる。

3.2 CUDA

GPGPU のプログラミング環境としては Nvidia 社が提供する CUDA が一般に広く使われている。CUDA は C/C++言語をベースに、GPU を利用するために独自の拡張を行ったプログラミング言語であり、コンパイラ (nvcc)、実行時ライブラリ、数値計算ライブラリ、ドキュメントなどが提供されている。CUDA は Nvidia 社製 GPU 専用であるが、GPU への低レベルでのアクセス手段を提供することから、適切に利用することで GPU の持つ高い性能を利用することが可能である。

CUDA で記述されたプログラムは以下の特徴を持つ。まず、プログラムは CPU で実行する部分と GPU で実行する部分を明示的に記述する必要がある。CUDA では、GPU で処理を実行する単位は関数であり、これを「GPU カーネル」もしくは「カーネル関数」と呼び、__global__ という関数指示子を記載する。通常はプログ

ラムの中で計算負荷が大きく、並列化可能な部分を関数として抽出し、GPU で実行させることで高速化を図る。カーネル関数の呼び出しは「<<<」と「>>>」を用いて記述する。なお、関数指示子には__global__ (CPU から呼び出されて GPU 上で実行する関数)、__device__ (GPU から呼び出されて GPU 上で実行する関数)、__host__ (CPU から呼び出されて CPU 上で実行する関数) がある。

また、GPU は CPU とは独立したメモリを持ち、CPU から GPU 上のメモリへのアクセスは制限がある。また、その逆に GPU から CPU 上のメモリへはアクセスできない。すなわち CPU から GPU へはデータ転送を行う必要があり、そのための API（cudaMemcpy などの API 関数）が用意されており、GPU 上での処理を行う前後にこれらの API 関数を用いてデータの転送を行う必要がある。

CUDA で記述されたプログラムの処理の流れは以下のようなイメージになる。

1. cudaSetDevice(0) : 使用する GPU の ID を指定する。
2. cudaMalloc() : GPU 上のメモリを確保する。
3. cudaMemcpy(,,,cudaMemcpyHostToDevice) : CPU から GPU へデータ転送を行う
4. カーネル関数<<< , >>>() : カーネル関数を呼び出し、GPU での処理を行う。
5. cudaMemcpy(,,,cudaMemcpyDeviceToHost) : GPU から CPU へデータ (計算結果) 転送を行う
6. cudaFree() : 確保した GPU 上のメモリの解放を行う。

3.3 CUDA による並列処理

GPU を用いた計算では GPU カーネルが GPU で実行される。その際に GPU 上の多数のプロセッサ（CUDA コア）それぞれにおいて同一の GPU カーネルが実行される。CUDA コア上で実行される各インスタンスは個別の ID を持ち、その ID を用いてそれぞれが担当するデータを特定し、GPU カーネルで記述された処理を実行することができる。

図 1 に Nvidia 社製の GPU、Tesla C2050 のハードウェア構成の模式図を示す。Tesla C2050 は Fermi アーキテクチャを採用した GPU で、CUDA コアと呼ばれる演算器が最小単位となり、CUDA コアが 16 個× 2 を一つのまとまりとして Streaming Multiprocessor(SM) と呼ぶ。C2050 では 14 個の SM が実装されており、CUDA コアはトータルで 448 個 ($16 \times 2 \times 14 = 448$) となる。多量の CUDA コア（C2050 の場合 448 個）がフラットに実装されているのではなく、CUDA コア、SM と言った階層構造を持っているのが GPU の特徴となっている。これは演算器だけではなく、メモリに関しても同様に階層構造が存在する。

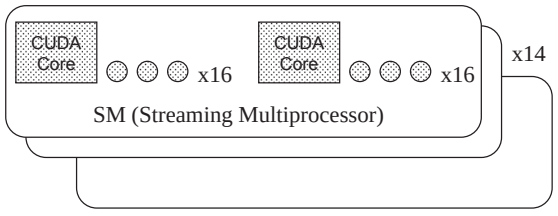


図 1: Tesla C2050 のハードウェア構成

CUDA コアは一つのスレッドが実行できる単位である。同じ SM に存在する CUDA コアは全て同じ演算を実行する SIMD もしくはベクトル処理的な実行形態となり、通常のマルチコア CPU における「コア」とは異なっている。一般的なマルチコア CPU における「コア」に相当するものは GPU では SM となる。CUDA コアは SIMD コアであり、複数の CUDA コアがそれぞれ異なるデータに対して同じ演算を行うデータ並列処理が CUDA での基本的な並列処理となる。また、CUDA コアは分岐予測器や Out of Order 機能を持たないシンプルな演算単位であり、同一クロックの CPU コアと比べると演算性能が低い。GPU では多数の CUDA コアが使える高並列度の問題でなくては高い性能を発揮することができないため、その様な使い方が必要となる。

既存のマルチコア CPU の場合、コア数以上にスレッドを生成した並列処理を行うと、Time sharing 実行となり、コア数以下のスレッドを用いた並列処理に比べて性能が低下する。一方、GPU では、ハードウェアの特性として処理の切り替えが高速に実行できるため、あるスレッドがメモリアクセスで待ち状態になった場合、実行待機状態のスレッドに切り替えることでメモリレイテンシを隠蔽することが可能となり、CUDA コア数よりも多くの数のスレッドを使うことで高い演算性能が得られようになっている。CUDA においては図 2 で示すように処理の単位としてスレッド、ブロック、グリッドがある。スレッドは CUDA コアに割り付けられる処理、ブロックは SM に割り付けられる処理、グリッドはカーネルの実行単位である。図ではスレッド、ブロックとも 1 次元で表現しているが、ブロックは 2 次元空間、スレッドは 3 次元空間に割り付けることができる。同一グリッドでは、ブロック毎のスレッド数は同じでなくてはならない、また同一ブロック内のスレッドは全て同じ SM に割り当てられるなどの制限がある。ブロック内のスレッドは 32 スレッドを一つの単位（Warp と呼ばれる）として割り当てられる。

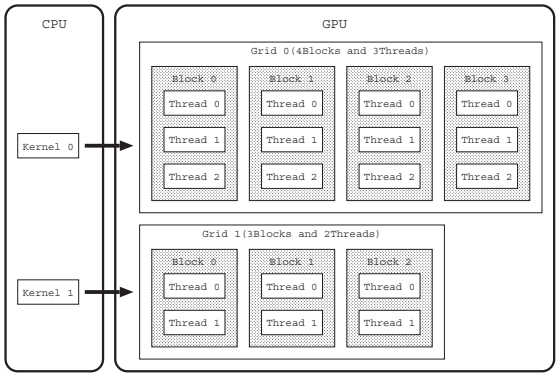


図 2: CUDA における処理の階層構造

GPU および CUDA はこれまで述べてきたような特性を持っており、GPU による高速化ではどのように対象となるプログラムの並列化を行うかが重要となる。特に、高い性能を得るためには高い並列度を確保する必要がある。ここで、GPU を用いた高速化対象としているのは粒子フィルターを衛星熱解析に適用したプログラムである。粒子フィルターでは多数のアンサンブル計算を行う必要があり、言わば大量のパラメトリック計算を行うことになる。そのため並列化手法としては二通りの手法とそれらの組み合わせが考えられる。1) 各アンサンブルの解析自体を領域分割による並列化により高速化する方法、2) 各アンサンブルの解析自体は並列化せずに、多数のアンサンブル計算をそのまま並列に実行する方法、3) 上記の二つの手法を組み合わせて各アンサンブルの計算を並列化し、さらにそれを同時にパラメトリック計算を行う方法である。大量のパラメトリック計算を実施するという粒子フィルターの特性と、プログラミングの容易さから、ここでは 2) の多数のアンサンブル計算を並列実行することとした。

GPU を用いたプログラムの高速化では、プログラム中の計算負荷が高い部分 (計算時間が多くかかるところ) を抽出し、その部分を GPU で実行するやり方もあるが、ここでは熱解析プログラムを全て GPU 上で実行することとした。もともと Fortran90/95 で書かれた熱解析プログラムを CUDA3.2 を用いて書き直した。も

表 1: 計算機

	CPU	GPU
計算機 A	Xeon X5650 x 2	Tesla C2050 x1
計算機 B	Corei7 x 1	GTS x 4

表 2: CPU および GPU の仕様

	周波数	コア数	性能
Xeon X5650	2.66GHz	6	63.84Gflops
Corei7	3.33GHz	6	79.92Gflops
Tesla C2050	1.15GHz	448	515.2Gflops
GTS450	1.57GHz	192	301.4Gflops

とのプログラムでは MPI および OpenMP を用いたハイブリッド並列を行っていたが、ここでは GPU の性能を評価することを目的とするため、CUDA で書き直したプログラムは MPI 並列は行っていない。また複数の GPU を利用するために OpenMP でのスレッド並列を行い、OpenMP の各スレッドがそれぞれ GPU の制御を行うようにした。なお、最新の CUDA4.0[8] では複数の GPU を 1 スレッドで制御することが可能となっている。CUDA で書き直したプログラムでは、複数の GPU 上の CUDA コアおよびホスト CPU のコア (GPU の制御を担当するコアを除く) が分担して大量のアンサンブル計算を実行するような並列化を行った。計算は全て倍精度で行うこととした。

4. 性能評価

文献 [3] で用いられている小型衛星モデルを対象とした熱解析で性能評価を行った。用いた熱数学モデルは節点数が 16 点の小さなモデルである。小型の実衛星規模 (4,000 節点程度) のデータでも代表的な性能評価を行ったが、傾向は変わらなかったため、ここでは小さなモデルでの結果について報告する。実時間 1,000 秒分の解析を実行した場合の計算時間で比較を行った。

性能評価に用いた計算機および搭載 CPU および GPU のスペックを表 1、2 にまとめる。ここでの性能評価は主に計算機 A で行った。

計算機 A では GPU が 1 個、CPU が 2 個搭載されている。問題規模を同じ（総粒子数を同じ）にして GPU を使った場合 (GPU)、CPU を 1 個使った場合 (1CPU)、2 個使った場合 (2CPU) で計算時間の比較を行った。表 3 に総粒子数が 8,928 および 16,128 の結果を示す。総粒子数は粒子フィルターの粒子数でアンサンブル計算のアンサンブル数に該当する。CPU の場合、コア数=スレッド数とし、各スレッドが複数の粒子を担当することになる。一方、GPU では総スレッド数 (=ブロック数×ブロック当たりのスレッド数) が総粒子数と同じとなるようにし、各スレッドが 1 粒子の計算をすることになる。この表からブロック数とスレッド数を適切に設定すれば GPU を用いた計算が CPU を用いるよりも 2 倍程度高速であることがわかる。なお、スレッド数およびブロック数はある程度試行錯誤的に決定した。GPU の場合、総スレッド数が CUDA コア数よりも大きな値にすることが大事で、総スレッド数が小さい場合は CPU の方が速い結果となった。

この結果を GPU と CPU の理論ピーク性能で比較してみると、GPU はピーク性能が 515Gflops に対して CPU は 64Gflops(1CPU あたり）となり、GPU と CPU のピーク性能比は 8 となる。ピーク性能比を考慮すると GPU は絶対性能としては CPU より高いが、実行効率では CPU の方が高く、その差は 4 倍程度と考えられる。GPU の実行効率が低い原因はプログラムのチューニング、特にメモリアクセスのチューニングを実施していないためと考えられる。GPU では複数の階層構造を持つメモリが搭載されており、高速性能を発揮させるには高速なアクセス性能を持つメモリを使うことが重要である。今回の試行では、全スレッドからアクセス可能なグローバルメモリを使っているため、メモリアクセスが高速ではなく、ここが性能ネックになっていると考えられる。高速化に向けた今後の課題と考えている。

計算機 B では GPU が 4 個、CPU が 1 個搭載されており、複数 GPU を利用した場合の性能評価を行った。総粒子数は 64,512 とした。結果を表 4 に示す。複数 GPU を使う場合でも CPU に比べて絶対性能は高いことがわかるが、ピーク性能比を考慮した場合、実行効率では CPU が良いと考えられる。

次に CPU と GPU を同時に使った場合の比較を計算機 A で行った。その結果を表 5 を示す。なお、総粒子数はそれぞれのケースで完全に一致していないが、その影響は小さいと思われる。

ケース 1 は GPU だけを用いた計算で総粒子数は 22,016 である。ケース 2 は GPU と 1CPU の両方を用いた計算である。ここで使用した計算機 A は 1CPU に 6 コアを搭載しており、OpenMP で 6 スレッドを起動し、1 スレッドが GPU の処理を制御し、残りの 5 スレッドは計算を行った。CPU と GPU では性能に差があるため、CPU、GPU で同じ計算時間となるように、それぞれに割り当てる粒子数を手動で調整した。GPU に割り当てた粒子数は 16,128、CPU に割り当てた粒子数は 5,880 となり、総粒子数は 22,008 である。割り当てられた粒子数を見ても GPU が 2 倍程度 CPU よりも高速であることがわかる。ケース 3 は 1CPU だけ (6 コアを使用) を用いた計算で総粒子数は 22,016 である。ケース A,B,C は搭載された演算器を全て利用する（CPU × 2, GPU）条件で比較を行った。ケース A は CPU だけで、粒子数は 29,056。ケース B は GPU の粒子数が 16,128、CPU は 11 コアを使い、粒子数は 12,936 で総粒子数は 29,064 である。ケース C は CPU だけ (CPU × 2) で、粒子数は 29,064 である。当然の結果ではあるが、GPU と CPU を組み合わせて使う場合が最も速く、ケース 1,2,3 の場合は GPU だけの場合の約 1.5 倍、CPU だけの場合の 3 倍、ケース A,B,C の場合は CPU だけ、GPU だけに比べて 2 倍程度高速であることがわかる。

ケース B' はケース B と同じ粒子数 (29,056) を GPU と CPU1 個で分担して計算した場合、ケース C' はほぼ同じ粒子数 (29,052) で CPU1 個 (6 コア) を用いて計算した結果である。ケース C' に CPU を追加した場合がケース C になり、1.99 倍高速化されたことになる。一方ケース C' に GPU を追加した場合がケース B' になり、3.1 倍高速化されたことになる。1CPU マシンに CPU を追加するのか、GPU を追加するのかを考えた場合、今回のケースでは GPU を追加した方が良い結果となった。

4.1 スレッド数、ブロック数の特性

前に述べたように、CUDA のハードウェアモデルには階層構造（スレッド、ブロック）があり、これらの値をどの様に決めれば良いかという問題がある。一般にこれらのパラメータの最適な値はアプリケーション毎に異なり、ある程度試行錯誤的に決めてやる必要がある。そのため、スレッド数およびブロック数の決め方の指針を得るため、それぞれの値を変化させた時の性能の変化について調べた。計算規模（この場合は粒子数）を拡大するにつれて、計算リソースを増やす弱スケーリングで調査を行った。各スレッドは常に 1 粒子の計算を担当し、ブロック数、スレッド数に応じて粒子数を増減させた。この様な場合、理想的な並列計算では、常に計算時間は一定となるが、実際は並列処理等のオーバーヘッドのため、スレッド数の増加にともない、計算時間は増加する。

まず、ブロック数の影響を調べた。スレッド数を 1 としてブロック数を変化させた場合を図 3 に示す。

表 3: GPU と CPU の性能比較

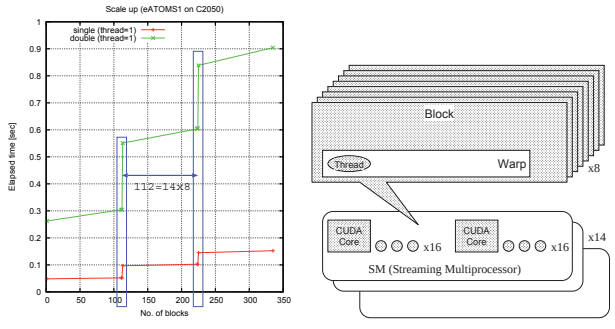
	ブロック数	スレッド数	粒子数/スレッド	総粒子数	計算時間 [秒]
GPU	558	16/ブロック	1	8,928	5.754
1CPU	-	6	1,488	8,928	13.04
2CPU	-	12	744	8,928	6.570
GPU	1,008	16/ブロック	1	16,128	10.33
1CPU	-	6	1,488	16,128	23.60
2CPU	-	12	744	16,128	11.87

表 4: 複数 GPU と CPU の性能比較

	デバイス数	ブロック数	スレッド数	粒子数/スレッド	総粒子数	計算時間 [秒]
GPU	4	1,008	16/ブロック	1	64,512	27.82
CPU	1	-	6	10,752	64,512	78.41

表 5: GPU,GPU+CPU,CPU の性能比較

		ブロック数	スレッド数	粒子数/スレッド	総粒子数	計算時間 [秒]
Case 1	GPU	1,376	16/ブロック	1	22,016	14.57
Case 2	GPU	1,008	16/ブロック	1	16,128	10.32
	CPU(1)	-	5	1,176	5,880	10.32
Case 3	CPU(1)	-	6	3,668	22,008	32.27
Case A	GPU	1,816	16/ブロック	1	29,056	19.08
Case B	GPU	1,008	16/ブロック	1	16,128	10.34
	CPU(2)	-	11	1,176	12,936	10.36
Case C	CPU(2)	-	12	2,422	29,064	21.38
Case B'	GPU	1,326	16/ブロック	1	21,216	13.73
	CPU(1)	-	5	1,568	7,840	13.73
Case C'	CPU(1)	-	6	4,842	29,052	42.44



(a) 計算時間 (b) 処理イメージ

図 3: スレッド数を 1 としてブロック数を変化させた場合

表 6: C2050 のハードウェア制限

項目	ハードウェア制限
Warp サイズ	32
最大スレッド数/ブロック	1,024
最大スレッド数/SM	1,536
最大 Warp 数/SM	48
最大ブロック数/SM	8

図では、単精度計算と倍精度計算の結果を示している。1 粒子の計算（1 スレッド、1 ブロックとなり、CUDA コアの性能となる）で比較すると倍精度計算は単精度計算に比べて約 5.4 倍程度遅いことがわかる。また、どちらのケースでも計算時間はブロック数の変化に対して、112 ブロックを単位に計算時間が段階的に長くなるという離散的な傾向を示している。Tesla C2050 のハードウェアにはいくつかの処理単位やハードウェア制限があり、それらの値を表 6 に示す。

これらの制限と図 3(b) より、ブロック数 112 は C2050 が搭載する SM の数 (14) と SM 当たりの最大ブロック数 8 の積であることがわかる。つまり本ケースでは、1 スレッドを有する Warp が一つだけブロックに割り当てられ、そのブロックが最大で 8 ブロックまで一つの SM に割り当てられる。SM は 14 個なので、C2050 には最大で 112 ブロックでハードウェアが一杯になる。つまり、同時に計算されるのは 112 ブロックまでで、それが処理単位となりブロック数の増加に伴って 112 ブロックで段階的に計算時間が長くなる傾向を示すと考えられる。これは、SM の搭載数が異なる GPU(例えば、GTS450) の結果と比較するとより明確になる。図 4 に C2050 と GTS450 の結果を示す。C2050 では SM は 14 個搭載されているが、GTS450 では SM は 4 個搭載されている。そのため、GTS450 では $8 \times 4 = 32$ ブロックが処理単位となっており、ブロック数の増加にともない 32 ブロックで段階的に計算時間が離散的に変化している。

C2050 において 112 ブロック (GTS450 の場合は 32 ブロック) の処理単位の中でブロック数の増加にともない、計算時間が若干増加しているのは、メモリアクセスの影響が考えられる。この図で示すように単精度と倍精度の比較では、倍精度の増加傾向が強いこと、また図 5 で示すように、固定スレッド数を増やした場合は、スレッド数が多い程増加傾向が強いことからメモリアクセスが主な原因と考えられる (「倍精度」、「スレッド数が多い」はどちらもメモリアクセスが増加するため)。

次にスレッド数の影響を調べた。今度はブロック数を 1 としてスレッド数を変化させた場合を図 6 に示す。

図には単精度計算と倍精度計算の結果を示している。どちらも同じようにスレッド数の増加とともに計算時間が増加している。また、特定のスレッド数で傾向が変化し、そのスレッド数は単精度が 32、倍精度が 16 と異なることがわかる。表 6 の制限より、図 6(b) で示すように各 SM には 1 ブロックが割り当てられ、スレッ

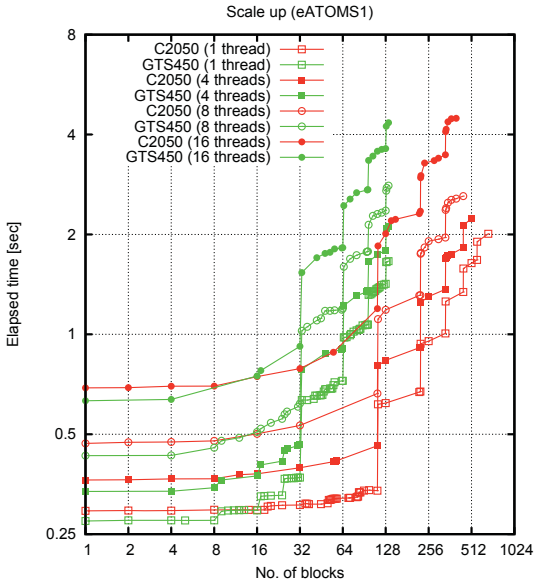
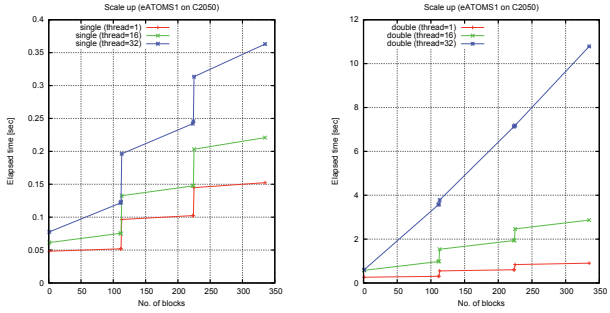
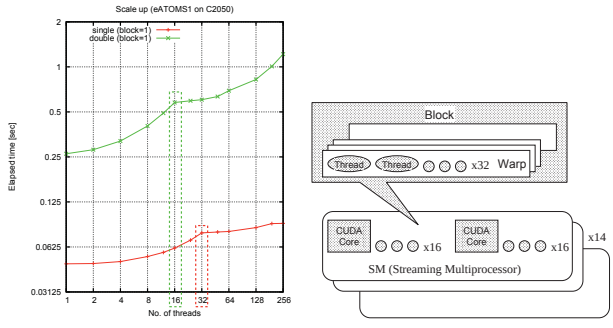


図 4: C2050 と GTS450 の比較



(a) 単精度 (b) 倍精度

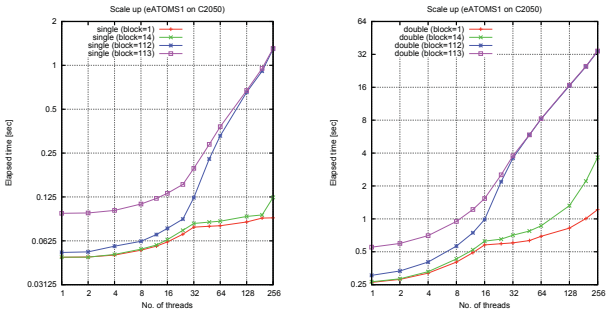
図 5: スレッド数を固定してブロック数を変化させた場合



(a) 計算時間 (b) 処理イメージ

図 6: スレッド数を 1 としてブロック数を変化させた場合

ド数の増加とともに Warp 内のスレッド数が変化することとなる。SM 内の CUDA コアは全部で 32 個あり、これが単精度の場合の処理単位に相当する。倍精度の場合は、2 つの CUDA コアを組み合わせて倍精度演算を行うために、実質的に 16 個となりこれが倍精度の場合の処理単位となる。スレッド数を増やしていくと、上記の理由により単精度演算では 32 スレッド、倍精度演算では 16 スレッドで SM が埋まってしまうため、図で示すような傾向を示すと考えられる。



(a) 単精度 (b) 倍精度

図 7: ブロック数を固定してスレッド数を変化させた場合

図 7 に固定したブロック数を増やした場合の計算時間の傾向を示す。ブロック数が異なっていても同じ様に 32 スレッド（単精度）、16 スレッド（倍精度）で傾向が変化することがわかる。

5. おわりに

高精度衛星熱数学モデルを構築する手段として粒子フィルターの適用を試みているが、そこでは大量の解析を高速に行う必要がある。そのため、近年高性能計算機として注目を集めている GPU を用いて解析の高速化を試みた。粒子フィルターにおけるアンサンブル計算を GPU における高並列度処理にマッピングすることで GPU を用いた解析を行った。GPU を用いることで CPU よりも高速な計算が行えることを確認した。特に CPU と GPU の両者を同時に用いることで、CPU だけを用いるよりも 2 倍程度高速化することができた。GPU が持つ潜在能力はまだ十分活かしきれていないため、高速メモリの活用など更なるチューニングが必要であり、今後の課題である。

参考文献

1) 中村和幸, 上野玄太, 樋口知之. データ同化：その概念と計算アルゴリズム. 統計数理, Vol. 53, No. 2, pp. 211–229, 2005.

2) 高木亮治, 秋田剛, 嶋英志. 宇宙機熱数学モデルにおけるパラメータ推定への粒子フィルターの摘要. 第 42 回流体力学講演会/航空宇宙数値シミュレーション技術シンポジウム 2010 講演論文集, pp. 735–740, 2010.

3) 秋田剛, 高木亮治, 嶋英志. アンサンブルカルマンフィルタを用いた衛星熱数学モデルの接触熱伝導率推定法. 宇宙技術, Vol. 9, pp. 1–8, 2010.

4) 秋田剛, 高木亮治, 嶋英志, 石村康生. アンサンブルカルマンフィルタの適応型熱解析への適. 第 42 回流体力学講演会/航空宇宙数値シミュレーション技術シンポジウム 2010 講演論文集, pp. 729–734, 2010.

5) 樋口知之. 粒子フィルタ. 電子情報通信学会誌, Vol. 88, No. 12, pp. 989–994, 2005.

6) 中野慎也, 上野玄太, 中村和幸, 樋口知之. Merging particle filter とその特性. 統計数理, Vol. 56, No. 2, pp. 225–234, 2008.

7) CUDA Zone http://www.nvidia.co.jp/object/cuda_home_new_jp.html.

8) CUDA Toolkit 4.0 http://developer.nvidia.com/cuda-toolkit-40.