Vol.2020-HPC-177 No.17
2020/12/22

小天体重力場計算の並列処理による高速化

河野 郁也[1,a)]　中里 直人[1]　平田 成[1]　松本 晃治[2]

概要：太陽系小天体の三次元形状モデルデータを読み込み，重力ポテンシャルや引力をはじめとした物理量を計算する GFandSlope をアクセラレータ用コードとして実装し，GPU 上で性能を評価した．逐次実行では，高解像度モデルデータの入力や入力パラメータ変更ごとの再計算による急激な処理時間の増大は避けられなかったが，本実装により CPU 上での計算と比較して数百〜千倍の飛躍的な処理時間短縮に成功し，小惑星研究をより円滑に進められるものと期待される．

1. はじめに

太陽系小天体とは，太陽系の天体のうち惑星，準惑星，衛星以外の小惑星や彗星を含む天体を指す．小天体・小惑星を対象とした研究・探査活動は，2000 年代初頭の日本の小惑星探査機はやぶさ (MUSES-C) による小惑星イトカワ (25143 Itokawa) に対する探査や，最近実施された後継機のはやぶさ 2 による小惑星リュウグウ (162173 Ryugu) の探査をはじめ，積極的に行われている．太陽系の天体は，46 億年前の太陽系形成時に微惑星の衝突合体によって誕生したと考えられている．地球のような惑星は，その後の内部構造の分化や表面の地質活動によって，形成初期の状態が残されていない．一方，小惑星を含む太陽系小天体の多くは，あまり分化せずに形成当時の状態を保持している．そのため小惑星研究は，太陽系および生命の誕生や進化を解明する手掛かりを得ることに繋がると期待される．

天体表面の重力場は天体の形状，自転遠心力，内部構造によって決まる．大きく成長した惑星は自己重力によって球形に近い形状を持つが，太陽系小天体は自己重力が弱く，不規則な形状を持っているものが多い．また，自転遠心力による寄与や内部構造の不均質性の影響もあり，小天体表面の重力場は複雑な様相を示す．その実態と天体表面での地質活動への影響などを調べるために，天体の形状モデルを使った数値シミュレーションが用いられる．例えば Richardson ら [1] は，エロスやイダなどの複数の小惑星において天体形状を多面体でしたモデルを用いて，自転速度などの初期条件を変えて表面における重力勾配などの物理的情報を計算している．また Kanamaru ら [2] は，Mascon Gravity Modeling という手法を開発して，複数の 3 次元メッシュの組み合わせで形状を表現した小惑星イトカワにおいて重力場計算を行っている．この手法では各メッシュに質点を置くことで，天体内部の密度を考慮した計算を可能にしている．

一方，会津大学では Mitsuta ら [3] によって重力場計算コードである GFandSlope[4] が開発されており，Richardson らのように天体形状を多面体で近似したモデルに対する計算を行っている．GFandSlope が用いられた小天体解析の例としては，Mitsuta ら [3] による小惑星リュウグウ表面からの衝突放出物の軌道計算や，Watanabe ら [5] によるリュウグウ表面における重力勾配の計算などがある．

GFandSlope の重力場計算アルゴリズムは，Werner と Scheeres が 1996 年に発表した多面体の貼り合わせによって小天体を構成して計算を行う理論モデル [6] に基づいて実装されており，論文は「並列処理によって高い恩恵を受けられる計算法である」との期待で締めくくられている．

ところが GFandSlope は，GPU などのアクセラレータを利用した並列化・高速化を見越していない C++による逐次計算コードとなっており，現在も CPU 上での逐次処理により利用されている．重力場計算部分は細かな処理は多いものの，本質的には N 体計算の部類である．これまでの研究では数万ポリゴン程度の低解像度に対する計算で使われているため支障は少なかったが，今後モデル解像度や物理的な初期条件を多様に変えて繰り返し計算を行っていくには計算時間の面で力不足となることは明白である．N 体計算は演算強度が高く，それほど高いメモリバンド幅が要求されないため GPU での計算に適している．それゆえに，GFandSlope についても GPU の演算性能を十分に活かして高速化できることが期待できる．

我々は，GFandSlope のコード全体を GPU を用いて計算できる形に実装を改めて，CUDA, OpenCL で重力場計

[1] 会津大学
[2] 国立天文台
[a)] f-kono@u-aizu.ac.jp

算の並列化を行った．本稿では，作成した GFandSlope の GPU 版コードについて NVIDIA Tesla V100 GPU を中心に用いた性能評価について報告する．

2. 小天体モデルへの重力場計算の適用

始めに本研究での高速化の対象である重力場計算コード GFandSlope の概要と，主要な計算について述べる．

2.1 入力形式，モデル

図 1 は，入力ファイル例の 1 つとして存在する小惑星イトカワのモデルを読み込んで，可視化したものである．このモデルは，49152 個の三角形状ポリゴン (頂点数 25350, 辺の数 73728) をイトカワの表面に沿って貼り合わせることで構成されている．

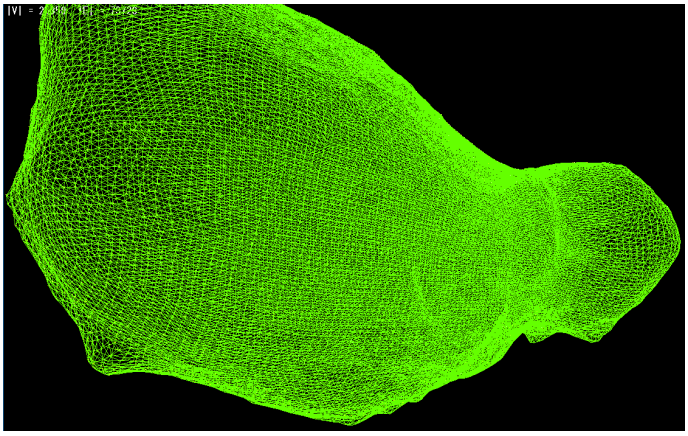

図 1 可視化した入力モデル (イトカワ)

入力ファイルは，モデルを構成するための頂点の列で与えられており，さらにモデル上で辺を構成する 2 頂点，そして三角形を構成する 3 辺の組み合わせが記述されている．紙風船のような物体の表面のみを覆った閉曲面であるため，内部の構造は持っていない．

2.2 計算全体の流れ

図 2 は，GFandSlope のプログラム全体の処理を表したものである．図 1 で示した頂点，辺，面の構造情報を持った入力ファイルを読み込んで，データ配列を初期化する．その後，小天体を構成する全ての面 (三角形) に対して計算を実行する．

各面における面積や重心座標，法線ベクトルといった細かな基本的計算を多量に含むが，全体に対して最もコストの大きいものは重力場（ポテンシャル）計算となる．ある面と他の全ての面 (および辺) との重力相互作用を計算するため，ポリゴン数の二乗に比例する演算が必要となり、本質的には N 体計算と同等となる。

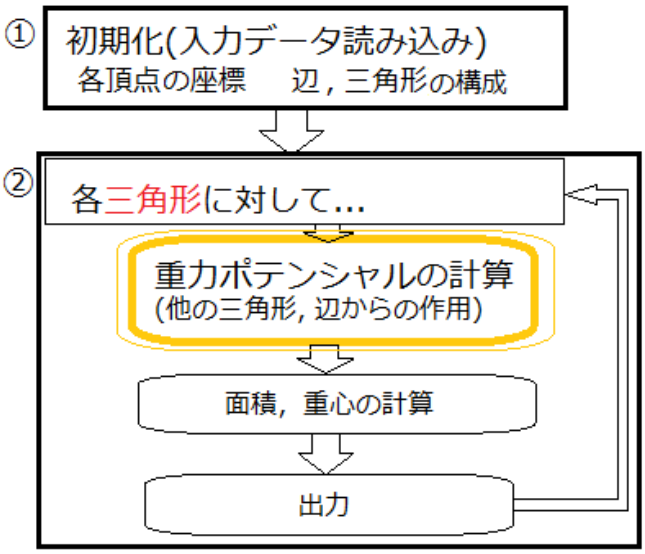

図 2 GFandSlope の計算全体の概略図

2.3 重力場計算部分の支配方程式

重力場計算部分では，小天体を構成する 1 つ 1 つの面 (三角形) に対して，次の式 (1)〜(3) を解いてポテンシャル，引力，ラプラシアンをそれぞれ得る．

$$U = \frac{1}{2}G\sigma \sum_{e\in edges} \mathbf{r}_e \cdot \mathbf{E}_e \cdot \mathbf{r}_e \cdot L_e - \frac{1}{2}G\sigma \sum_{f\in faces} \mathbf{r}_f \cdot \mathbf{F}_f \cdot \mathbf{r}_f \cdot \omega_f \tag{1}$$

$$\nabla U = -G\sigma \sum_{e\in edges} \mathbf{E}_e \cdot \mathbf{r}_e \cdot L_e + G\sigma \sum_{f\in faces} \mathbf{F}_f \cdot \mathbf{r}_f \cdot \omega_f \tag{2}$$

$$\nabla^2 U = -G\sigma \sum_{f\in faces} \omega_f \tag{3}$$

各式中における G は重力定数, σ は密度定数を表し，$\mathbf{r}$ は計算を行う面の重心を基準とした，各辺・面への位置ベクトルとする．各式を俯瞰すると，小天体を構成する各辺 $e \in edges$ および，各面 $f \in faces$ との間で計算を実行してその総和の差分を取っている．各項を詳しく見ると，$\mathbf{E}$, $\mathbf{F}$ および L, ω という変数が現れている．

今，三角形の貼り合わせによって小天体の表面に沿った閉曲面を構成している．即ち，図 3 のようにモデル内の任意の辺は 2 つの面に共有されていることになる．図中では，頂点 V_1, V_2 を結ぶ辺が面 A, B に共有されている．故に $\mathbf{E}$ および $\mathbf{F}$ は共通辺を管理するために導入される変数で，3×3 の正方行列である．

$\mathbf{E}$ は辺との相互作用の計算に現れるが，式 (4) のように任意の 2 つの面が共有する辺 (V_1, V_2) に対して，面の法線ベクトル $\hat{\mathbf{n}}_A, \hat{\mathbf{n}}_B$ と，それらと垂直な共通辺の法線ベクトル $\hat{\mathbf{n}}_{12}^A, \hat{\mathbf{n}}_{21}^B$ の直積をとった結果を保持している．

$$\mathbf{E}_e = \mathbf{E}_{12} = \hat{\mathbf{n}}_A(\hat{\mathbf{n}}_{12}^A)^T + \hat{\mathbf{n}}_B(\hat{\mathbf{n}}_{21}^B)^T \tag{4}$$

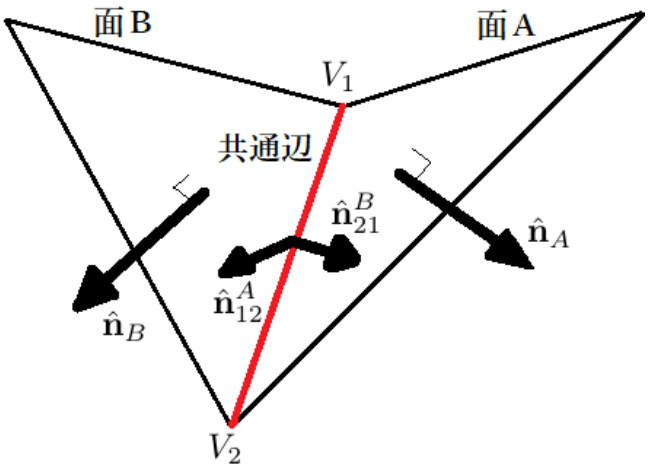

図 3 2 つの三角形に共有される小天体モデル上の任意の辺

一方 F は他の面 ($f \in faces$) との相互作用の計算に現れるが，こちらは式 (5) のように面 f の法線ベクトル同士で直積をとったものを保持する．

$$\mathbf{F}_f = \hat{\mathbf{n}}_f(\hat{\mathbf{n}}_{12}^f)^T \tag{5}$$

L および ω については，他の辺および面との間のポテンシャルの式を導出する際に現れる積分項を代数演算に置き換えるために生じるものである．面 f を構成する辺 $e = (V_i, V_j)$ に対する積分 L_e は式 (6) の形で計算する．

$$L_e = \int_e \frac{1}{r} ds = \log \frac{r_i + r_j + e_{ij}}{r_i + r_j - e_{ij}} \tag{6}$$

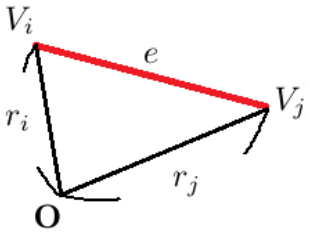

図 4 辺のポテンシャル計算に現れる L_e に関わる変数の図形的表示

r_i および r_j は，図 4 のように基準点を **O** とした，頂点 V_i, V_j の位置ベクトル $\mathbf{r}_i$, $\mathbf{r}_j$ の大きさ (基準点からの距離) である．また e_{ij} は，辺 e の長さ (頂点 V_i, V_j 間の距離) を表す．

また，ω_f は図 5 のように，小天体を構成する各面を単位球面上へ射影したときの符号付き面積 (頂点 V_i, V_j, V_k から構成される面 f に対する立体角) となる．図中の $\hat{\mathbf{n}}_f$ は面に対する法線ベクトルであり，$\hat{\mathbf{r}}$ は単位球上に射影した時の微小領域 $d\omega$ に対する法線ベクトルである．各面に対するポテンシャルを計算する時に式 (7) の第 1 式のように領域全体に渡る積分を行うことになるが，球面幾何学とベクトル代数の考え方を用いた式変形 [6] を行うことで，第

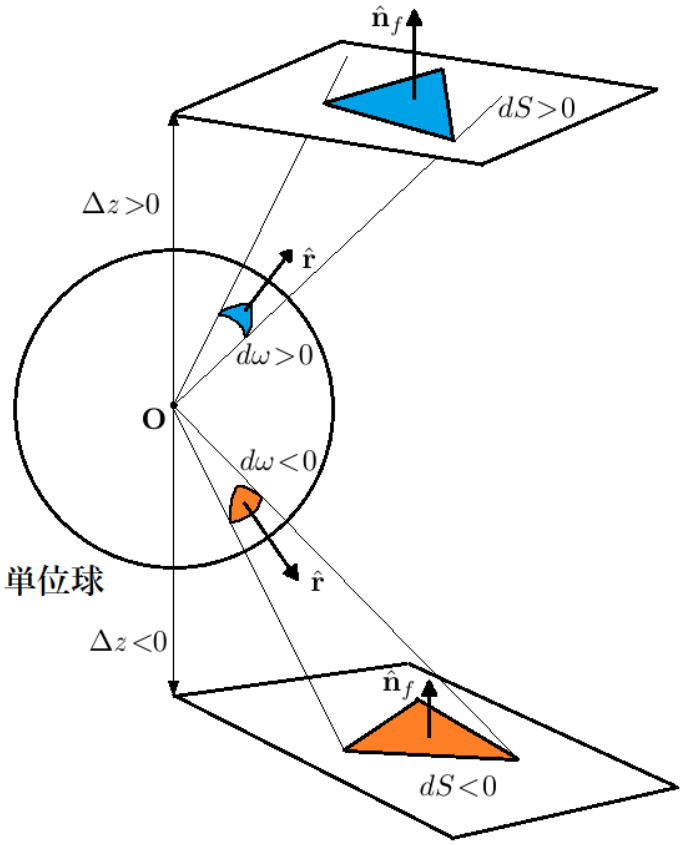

図 5 小天体モデルを構成する三角形の単位球面上への射影

2 式の 3 次ベクトル演算の形で表したものを用いる．

$$\begin{aligned}\omega_f &= \iint_{triangle} \frac{\Delta z}{r^3} ds \qquad (7)\\ &= 2\arctan \frac{\mathbf{r}_i \cdot \mathbf{r}_j \times \mathbf{r}_k}{r_i r_j r_k + r_i(\mathbf{r}_j \cdot \mathbf{r}_k) + r_j(\mathbf{r}_k \cdot \mathbf{r}_i) + r_k(\mathbf{r}_i \cdot \mathbf{r}_j)}\end{aligned}$$

3. 重力場計算の並列化

2 章で述べた，式 (1)〜(3) からなる重力場計算を高速に行うために，GPU カーネル化を行う．ただし，元の GFandSlope のコード全体は C++で書かれており，特に C++の Standard Template Library(STL) やクラス，およびその他ライブラリを併用しているため，並列化，さらには GPU 計算を見越した変更を想定していない実装となっている．

そのため，頂点や辺，三角形を保持するデータ構造を中心に見直し，コード全体を C 言語に移植して GPU カーネル化が行いやすいようにした．予備実験として，作成した (修正した C++および移植した C) コードに対して OpenMP による並列化も行う．

3.1 データ構造の書き換えと OpenMP によるマルチスレッド化

ファイルから入力した小天体のモデルデータは，頂点を構成する vector 配列，辺を表す頂点対を持つ vector 配列，

そして三角形を構成する頂点の 3 つ組を持つ vector 配列の形で保持される．共通辺と三角形は map を用いて対応付けられ，計算中は三角形や辺の配列を経由して頂点の配列にアクセスを繰り返す．

重力場計算部分のコードは図 6 のように，一般的な N 体計算に見られる二重ループ構造となっているため，OpenMP によるループ並列化は容易に検討することができる．OpenMP による並列化自体は C++コードに対しても行うことができるが，GFandSlope のループ処理は全体に渡って STL コンテナに対するイテレータを用いて行われているため，そのままでは並列化のためのループ分割が行えない．そのため，イテレータによるループ処理を全て添字によるランダムアクセスへ置換した．

```
//各三角形
 for(|P|){
  //全ての辺
    for(|E|){ … }
  //他の三角形
    for(|P|){ … }
 }
```

図 6 重力場計算部分のコード構造

GPU カーネル化を見越した C 言語への移植においては，クラス構造の解体をはじめ，STL コンテナで記述されたデータ構造を通常の配列で実装をやり直した．2 章で述べた三角形の共通辺管理を含む辺の扱いはややトリッキーな実装となっているため，C 言語化にあたって単純な配列参照となるように変更した．

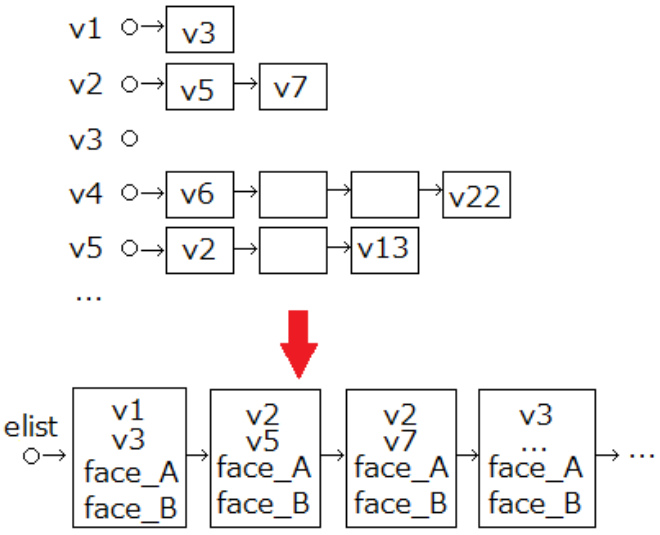

図 7 移植コード上での辺および隣接三角形を保持するデータ構造

図 7 は，C 言語コード上のデータ構造の例として，辺情報を保持する配列の作成過程を示したものである．移植コードでは，辺および三角形情報を読み込む際に頂点ごとの隣接リスト (図上部) を構築しておく．その後リストを辿って，各辺を構成する頂点とそれを共有する三角形を保持する構造体を作り，それらの総数を調べた上で AOS 形式となるように辺の配列 (図下部) を作成した．

頂点データは，元の C++コードでは複数のライブラリの複合で扱われる 3 次ベクトルクラスで保持されているため，x, y, z の 3 要素を持つ構造体として定義し直して，重力場計算に必要なベクトル演算のみを改めて実装した．

このようにして，C++で書かれていたデータ構造ができるだけ単純な配列参照となるように C 言語で書き直した．その結果，OpenMP 指示文を重力場計算中の `for` ループに与えるだけで並列化できる．ここでは，元のコードの構造として三角形を参照する外側ループに IO 部分が関わっていたため，他の三角形および辺を参照する内側のループに対して指示文を設定した．C 言語移植にあたっては，GPU カーネル化のためにこの IO 処理を計算の前に独立して行うように変更した．

3.2 GPU カーネル化 (CUDA/OpenCL)

前節の変更を施した GFandSlope の C コードを，GPU が利用できるように拡張する．C 言語への移植を通してデータ構造と共に並列化可能な処理が明確に抽出されたため，GPU カーネル化も容易である．応用の都合上，計算機環境を選ばず利用できるようにするために CUDA 版と OpenCL 版をそれぞれ作成した．両者のコード内容は，利用する API が異なるだけで本質的に同一である．

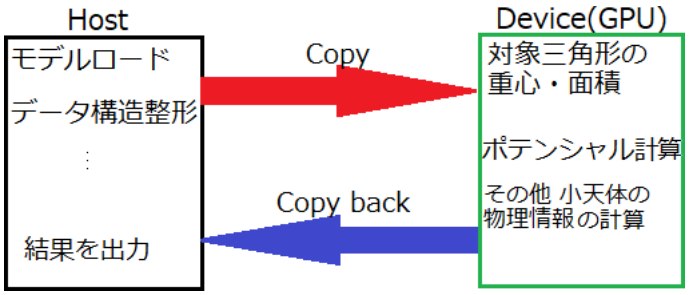

図 8 GPU 版 GFandSlope の処理構造

図 3 は，GPU 実装版におけるホスト側とデバイス側の担当する処理を示したものである．ホスト側は，入力ファイルに基づくデータ構造の初期化を行った後にそれをデバイスへ転送し，デバイスから受け取った計算結果を出力するのみである．

デバイス上で実行される計算カーネルとして実装するのは，2 章の図 2 に示した ② の部分に相当する．小天体モデルを構成する三角形の数は，解像度が上がるにつれて数万から数百万まで大きくなる．そのため，各三角形に対す

る重力計算を 1 つの GPU スレッドに割り当てて計算するようにした．三角形ごとに面積や重心座標をはじめ，計算したポテンシャルを元に算出した重力勾配など多数の物理的情報を含むため，それぞれのスレッドで計算結果を構造体にパックし，グローバルメモリを経由して構造体配列としてホストへ全ての計算結果を返す．

C 言語移植の時点でコードを大きく修正しているため，改めて GPU カーネル化のために必要な修正は極めて少ない．デバイスへ渡される入力データは読み込みしか行わないため，定数メモリや共有メモリの利用は有効であると考えられるが，現時点の実装ではデータ量が非常に大きく載せることができないため，直にグローバルメモリに配置している．一方で $\mathbf{E}$, $\mathbf{F}$ および L, ω のように構造の共通したごく一時的に使われる数多くの変数については，デバイス上でのレジスタスピルを避けるため，再利用できる変数はそうすることでデバイス上で宣言する変数の数を最小化している．

4. 性能評価

並列化した C 言語版 GFandSlope に対して，諸計算機上でベンチマークを実行してその性能を評価する．本稿内で作成した並列化コードは以下の通りである．

- OpenMP (C,C++)
- CUDA, OpenCL

OpenMP 版は予備評価として，ループ並列化のためにイテレータ部分のみを書き換えた C++コードと移植した C コードをループ並列化したものを実行する．

4.1 計算機環境

OpenMP 版コードは, Intel(R) Xeon(R) Gold 5122 CPU (3.60GHz, 4 コア/4 スレッド) を用いて逐次およびスレッド並列によって計算した結果を示す．コンパイラは gcc (ver.5.4.0) を使用している．GPU 用に作成したコードは，HPC をはじめ AI やその他分野の計算のために研究機関で普及が進んでいると考えられる NVIDIA Tesla V100 を中心に表 1 に示した GPU を用いて計算した結果を示す．参考として，CUDA 実装とほぼ同等のものとして用意した OpenCL 版コードを使ってゲーミングデバイスとして比較的安価で入手・利用できる AMD Radeon RX Vega64 を用いた結果も示す．

GPU	Tesla K20	Tesla V100	Radeon RX Vega64
GPU コア数	2496	5120	4096
周波数	706MHz	1370MHz	1247MHz
メモリサイズ	5GB	32GB	8GB
理論性能 (DP)	1.17Tflops	7.0Tflops	791.6GFlops
理論性能 (SP)	3.51Tflops	14.0Tflops	12.66TFlops

表 1 CUDA/OpenCL コードの性能評価に用いる GPU

4.2 ベンチマーク条件

入力として，小惑星イトカワ (25143 Itokawa) のモデルを使用する．OpenMP コードには図 1 でも示した 49152 の三角形状ポリゴンからなるイトカワモデルを用いて計算部分の時間を計測する．GPU コードに対しては，これに加えて全 4 種類の異なるポリゴン数 (最大 314 万ポリゴン) のイトカワモデルを用いて計算を行う．入力および初期化にかかる時間は計算全体の時間に対して極めて短いため，アプリケーション上考慮しなくても問題ないと考えられる．

浮動小数点演算については，元の実装に合わせて倍精度で行っている．単精度の場合についても後で別途評価する．

修正を行う前の C++コードを，49152 ポリゴンのイトカワモデルを入力して上記の Xeon CPU で実行したときの所要時間は 4900 秒 (約 1 時間 20 分) であった．これと比較してどこまで計算時間が短縮されるかが今後の応用のために重要である．

4.3 計算結果

表 2 は，元の C++コードを OpenMP 化のためにループ部分のみを書き換えたものと，GPU カーネル化のために C 言語へ移植したものを OpenMP 並列化したコードについて 49152 ポリゴン (50k) のイトカワモデルを使って計算した時の実行時間をまとめたものである．表中のスレッド数 1 の行は，OpenMP による並列化をしていない逐次実行の場合の結果を示している．

スレッド数	OpenMP/C++	OpenMP/C
1	4900.84	1323.48
2	2531.59	873.59
3	1809.39	689.89
4	1436.64	631.27

表 2 OpenMP 並列化による計算の実行時間 (s)

C++コードのループ部分の変更は計算時間にほとんど影響を与えなかったが，OpenMP によるループ並列化をできるようにしたことでマルチスレッド実行による速度向上の効果が見られる．一方移植した C コードは，逐次処理の時点で C++の 3〜4 倍高速になっている．GPU カーネル化を想定して外側ループで行われていた IO 処理をループ外へ追い出したことや，配列や構造体などの C 言語の基本的なデータ構造で簡素に移植したことが影響していると考えられる．

一方で，C コードのマルチスレッド実行による効果は C++コードと比べて半減している．頂点や辺，面それぞれに紐づいたデータは明にメモリ上近い場所に配置されるようにしているが，全体で見ると SOA 構造となっている．計算全体を通して，面や辺のデータを元に頂点データへア

クセスすることが頻繁に起こっているため，頂点データを参照する際に不連続アクセスとなって，性能向上のボトルネックに繋がっていると考えられる．

表 3 は CUDA/OpenCL コードを入力モデルのポリゴン数を変化させて GPU 上で実行した時の計算時間である．CUDA コードは NVIDIA GPU2 種について, OpenCL コードは V100 および AMD Radeon GPU で計算を行った結果である．

計算環境＼ポリゴン数	50k	200k	780k	3.1M
V100 (CUDA)	0.45s	7.18s	1m50s	30m
K20 (CUDA)	3.98s	64.17s	16m55s	4h30m
V100 (CL)	0.72s	7.26s	1m56s	30m
Radeon (CL)	4.44s	69.50s	18m30s	4h57m

表 3 CUDA/OpenCL による GPU 上での計算 (倍精度演算) の実行時間

いずれの計算環境においても，計算時間は小天体データのポリゴン数の二乗に比例するように増加している．計算環境ごとに結果を見ると，V100 GPU 上で CUDA コードを実行した場合は最も高速であり，50k ポリゴンの入力では 0.45 秒とプログラム実行開始後直ちに完了できている．高解像度な 3.1M ポリゴンを使用しても 30 分で計算が完了しており，CPU で実行した場合と比べると劇的に高速化されていることが分かる．元の C++実装を Xeon CPU でシングルスレッドにて実行した結果と比べると，V100 GPU を用いれば約 10000 倍高速に計算できるということになる．

一方，OpenCL コードを用いて V100 GPU 上で計算を実行しても CUDA コードよりわずかに実行時間が長い程度であり，ポリゴン数の十分大きなモデルではその差はほぼ無いことが分かる．ところが Radeon GPU で実行したときの時間は，旧世代の Tesla シリーズである K20 GPU で CUDA コードを実行した場合よりも長い傾向にある．Radeon シリーズはゲーミング用途に設計されており，倍精度演算器の実装が省略されている．そのため理論演算性能が単精度演算の場合に比べて極めて低いため，倍精度演算が必要な計算には適当でないと考えられる．

5. 考察

一連のコード修正による GPU カーネル化の実現によって，従来の逐次計算のみにしか対応していなかった GFandSlope による小天体モデルの重力場計算は目覚ましい処理時間短縮を達成した．ここでは GPU による高速化によって新たに期待できる小惑星研究のための実アプリケーションとしての展望や，それに合わせた本実装の更なる高速化へ向けた検討について議論する．

5.1 演算性能換算

最も短時間で計算を終えた NVIDIA Tesla V100 GPU での計算について，実行時間から演算性能を算出して評価する．まず, 計算カーネルにおける浮動小数点演算数を計数した．除算, 平方根は 2 演算として扱っている．GFandSlope の演算数は，入力した小天体モデルを構成するポリゴンおよび辺の数 N_p, N_e に依存する．1 つのポリゴンに対して，他辺との演算数は 175，他ポリゴンとの演算数は 174 である．また，相互作用の計算前後に 197 の浮動小数点演算がある．ゆえに，計算カーネル上の総演算数 N_{op} は式 (8) で表される．

$$N_{op} = (197 + N_e * 175 + N_p * 174) * N_p \qquad (8)$$

3.1M モデルの正確なポリゴンと辺の数はそれぞれ $N_p = 3145728$, $N_e = 4718592$ である．計算時間が丁度 30 分であったため，式 (9) のようにして演算性能を得る．

$$N_{perf} = N_{op}/1800(sec.)/10^{12} = 2.4\text{TFlops} \qquad (9)$$

V100 GPU の倍精度理論性能の公表値が 7TFlops である．現状の実装では明示的に FMA 命令を利用していないため，期待できるピーク性能は高々 3.5TFlops である．これと比較すると，ピークの約 70% の性能が出ていると考えられる．

5.2 単精度浮動小数点数への切り替え

ここまでの評価は，全て倍精度演算で計算を行ったものである．これは元の GFandSlope の実装に準拠したものである．しかし，許容できる範囲で精度を下げることによって計算時間を短縮できる可能性もある．

そこで，我々の GPU 用コードにおいて全ての浮動小数点数を倍精度から単精度に置き換えた上で，計算時間と数値精度を比較した．表 4 は，V100 GPU 上で単精度演算で計算した場合の実行時間をまとめたものである．

計算環境＼ポリゴン数	50k	200k	780k	3.1M
V100 (CUDA)	0.22s	4.33s	52.34s	13m32s
V100 (CL)	0.21s	3.28s	45.93s	11m24s

表 4 CUDA/OpenCL による GPU 上での計算 (単精度演算) の実行時間

計算時間については，ポリゴン数の 2 乗に比例することは勿論であるが，いずれの計算環境においても倍精度変数を用いた時と比べて半分以下となっている．

また倍精度演算による計算結果との数値誤差を入力ポリゴン数ごとに調べる．ここでは GFandSlope の計算の中で最も重要となる，重力ポテンシャルの値について絶対誤差および相対誤差を取って比較した．

ポリゴン数	50k	200k	780k	3.1M
有効桁数	5 桁	4 桁	3 桁	2 桁
相対誤差 (%)	0.001	0.02	0.18	1.04

表 5 倍精度/単精度演算による重力ポテンシャルの値の数値誤差

表 5 は，計算の出力結果から N_p 個の各ポリゴンごとの重力ポテンシャルの誤差の平均値を調べたものである．絶対誤差については，数値が一致している有効桁数として与えている．50k ポリゴンにおいては 1000 分の 1% の相対誤差であり，実の数値も有効桁数 5 桁まで一致している．ポリゴン数を大きくすると明らかに数値精度は低下しており，小天体を構成する 1 つ 1 つの面が小さくなることで，座標計算などにおいて 2 数の差が小さい演算を行う状況が増えていると考えられる．特に 3.1M ポリゴンを用いた場合の精度の低下は顕著で，1% の相対誤差が見られる．

50k ポリゴンの場合の隣接ポリゴンにおける重力ポテンシャルの差は単精度演算の場合の相対誤差より十分大きい．したがって，この程度のポリゴン数のモデルに対しては単精度演算を用いることで計算時間を短縮する選択も可能であると考えられる．一方，高解像度モデルに対しては計算精度と計算時間のトレードオフを行う必要も出てくると考えられる．そのような場合は重力計算の一部分について倍精度演算の併用を行うなどして，数値精度を維持できる方式を検討する必要がある．

5.3 アプリケーションの拡張と最適化

GPU 用コードの作成によって，GFandSlope を用いてポリゴン数の多い高解像度の小天体モデルに対する重力場計算も現実的な所要時間で解けるようになった．計算時間が従来と比べて大幅に短縮されたことにより，さらに精度の高い小天体解析のために逐次計算アルゴリズムの修正を検討する余地がある．

2 章で示した重力場計算法は，密度 σ がモデル全体に渡って均一であるという前提で行われる．実際の小天体においてそのようなことはもちろん無いため，1 つの小天体を複数のポリゴンの集合に分けて別々に計算を行うといった手法で内部構造を持ったモデルに対する重力場計算が行えるように拡張を行う予定である．

今後，対象モデルを高解像度したり，並列化前の計算が複雑化したりすることで計算量が増大するため，GPU を用いても計算時間が長くなる可能性がある．また，初期条件として与えるパラメータを様々に変えて計算を繰り返し行うため，GPU カーネルについても最適化を進めて 1 回の計算時間を短縮する必要がある．例えば N 体計算でよく用いられるツリー法 [7] を実装することで，面 1 つあたりの計算量を減らすことが考えられる．GFandSlope では，面 1 つに対して他の面および辺との重力作用を考慮する必要があるが，どの辺もモデル内のいずれかの面に接しているため，既に計算している各面の重心に粒子があると見て，そこへ近傍の辺と面を対応させることで，一般の N 体計算におけるツリー法に帰着できると考えられる．

一方，今回の C 言語，GPU コード化にあたってデータ構造の変更を図ったが，計算中に頻繁にアクセスされる面，辺，頂点の配列は全体として見ると SOA 構造になっている．面や辺の配列から頂点の配列へのアクセスが多いため，メモリの参照アドレスが飛び飛びとなりキャッシュの有効活用はあまりできていないと考えられる．これについては，今後のアルゴリズムの修正と同時に再度検討を行う．

6. おわりに

本稿では，小天体の構造を解析するために用いられる重力場計算プログラム GFandSlope を GPU を用いた並列計算によって高速化し，その性能評価について報告した．NVIDIA Tesla V100 GPU 上で，作成した GPU コードを用いて 49152 ポリゴンで構成された小惑星イトカワのモデルにおける重力場計算を行ったところ，僅か 0.45 秒で実行することができ，一般的な CPU で逐次実行した場合と比べて数百〜千倍の高速化に成功した．

元のプログラムは JAXA/ISAS のアーカイブにて公開されているが，長らく逐次処理にしか対応していなかったため，今回の結果により飛躍的に計算時間を短縮し，高精度化のための重力場計算アルゴリズムの拡張を検討できる段階に至った．逐次アルゴリズムの拡張に併せて更に計算量が増大し，かつ高速に解く必要が生じると考えられるため，現在の実装を元に GPU カーネル側のアルゴリズムの変更および，データ構造の最適化を検討していく予定である．

謝辞 本研究は文部科学省特色ある共同研究拠点の整備の推進事業 JPMXP0619217839 の助成を受けたものです。

参考文献

[1] Richardson, J. E. and Bowling, T. J.: Investigating the combined effects of shape, density, and rotation on small body surface slopes and erosion rates, *Icarus*, Vol. 234, pp. 53 – 65 (online), DOI: https://doi.org/10.1016/j.icarus.2014.02.015 (2014).

[2] Kanamaru, M., Sasaki, S. and Wieczorek, M.: Density distribution of asteroid 25143 Itokawa based on smooth terrain shape, *Planetary and Space Science*, Vol. 174, pp. 32 – 42 (online), DOI: https://doi.org/10.1016/j.pss.2019.05.002 (2019).

[3] Mitsuta, T., Hirata, N., Wada, K., Yano, H. and Arakawa, M.: Distribution of Impact Ejecta around a Small Asteroid: Implication to Artificial Impact Experiment in Hayabusa-2 Mission to the Asteroid 1999JU3., *LPI Contributions*, pp. 6476– (2012).

[4] Hirata, N.: GFandSlope, The Data ARchive and Transmission System (DARTS) at JAXA/ISAS (online), available from ⟨https://www.darts.isas.jaxa.jp/pub/hayabusa2/paper/Watanabe_2019/⟩ (accessed 2020-10-16).

[5] Watanabe, S., Hirabayashi, M., Hirata, N., Hirata, N.

et al.: Hayabusa2 arrives at the carbonaceous asteroid 162173 Ryugu—A spinning top–shaped rubble pile, *Science*, Vol. 364, No. 6437, pp. 268–272 (online), DOI: 10.1126/science.aav8032 (2019).

[6] Werner, R. and Scheeres, D.: Exterior Gravitation of a Polyhedron Derived and Compared with Harmonic and Mascon Gravitation Representations of Asteroid 4769 Castalia, *Celestial Mechanics and Dynamical Astronomy*, Vol. 65, pp. 313–344 (online), DOI: 10.1007/BF00053511 (1996).

[7] Barnes, J. and Hut, P.: A hierarchical O(N log N) force-calculation algorithm, *Nature*, Vol. 324, pp. 446–449 (1986).