Acceleration of Gravitation Field Analysis for Asteroids by GPU Computation

Fumiya Kono\(^1\), Naohito Nakasato\(^1\), Naru Hirata\(^1\), Koji Matsumoto\(^2\)

\(^1\) University of Aizu, \(^2\) National Astronomical Observatory of Japan (NAOJ)

IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-onChip (MCSoC-2021), December 20, 2021
• Asteroids have been actively investigated to elucidate the origin of the solar system and life.
  • Initiated by an exploration for Itokawa by Hayabusa probe in the early of 2000.
  • Hayabusa2 brought back a capsule containing carbon-rich fragments from Ryugu in 2020.

• Why asteroids?
  • The solar system bodies are considered to have consisted by repeating collision and merging of planetesimals during the evolution of the solar system.
  • The structure of planets has been changed by their planetary differentiation and geological activity on the surface.
Gravitation Field Analysis for Asteroids

- Numerical analysis for the gravitation field on the asteroid based on observation data
  - Evaluate the structure of asteroids.
  - Investigate geological activity on the asteroid surface.
- The calculation code GFandSlope has been developed in University of Aizu and used in preceding researches.
  - Trajectories of impact ejecta on the Ryugu (Mitsuta et al., 2012)
  - Gravitation slope on the surface of Ryugu (Watanabe et al., 2019)

However, the existing GFandSlope code requires much calculation time (several hours to days), which is a disadvantage in term of repeatedly calculations with
  - with different initial conditions, and
  - different resolution of input models.
Gravitation Field Analysis for Asteroids

- GFandSlope evaluates a complex body (asteroid) on the gravitation field.
- Calculation of physical properties on each face of approximated polyhedron.
  - Gravitation potential, attraction, Laplacian
- The algorithm was proposed by Werner and Scheeres (1996).
  - A theory to approximate a polyhedron model by polygon faces for gravitation calculation.

The original GFandSlope is only available in sequential C++ code.
- The main calculation is a kind of \( N \)-body simulation.
- Difficulty in large computation time for high resolution models

```plaintext
1: for f in faces do
2:   for e in edges do
3:     Calculation between f and e
4:   end for
5: for f' in faces do
6:   Calculation between f and f'
7: end for
8: end for
```
This research finds out parallel computation environment to accelerate GFandSlope code, which assists researchers working on space science.

This presentation focuses on computations with GPUs widely used for scientific researches.
Visualization of an Input Model Treated by GFandSlope

- An asteroid 25143 Itokawa
  - Input data is consisted of vertices, edges, and faces for a polyhedron.

- A shape model constructed by triangle faces covering on the surface
  - 25350 vertices, 73728 edges, 49152 triangle faces
Calculating Potential by GFandSlope

Calculation of gravitation field for each face (solve Eqs. (1) to (3) for all faces) on the model

- **Potential**
  \[
  U = \frac{1}{2} G \sigma \sum_{e \in \text{edges}} r_e \cdot E_e \cdot r_e \cdot L_e - \frac{1}{2} G \sigma \sum_{f \in \text{faces}} r_f \cdot F_f \cdot r_f \cdot \omega_f
  \] (1)

- **Attraction**
  \[
  \nabla U = -G \sigma \sum_{e \in \text{edges}} E_e \cdot r_e \cdot L_e + G \sigma \sum_{f \in \text{faces}} F_f \cdot r_f \cdot \omega_f
  \] (2)

- **Laplacian**
  \[
  \nabla^2 U = -G \sigma \sum_{f \in \text{faces}} \omega_f
  \] (3)

*G*: Gravitation acceleration, *σ*: Density (constant),
*r*: Vectors to other edges *e* and faces *f* from the centroid of the target face
Calculating Potential by GFandSlope (Variables E and F)

Each edge on the model is **shared by two faces**.

E and F are the of $3 \times 3$-matrix which are consisted by taking outer products of...

E: **normal vectors of an edge** $e$ and the target face.

F: **normal vectors of a face** $f$ and the target face.

![Diagram of faces and edges](image)

$$E_e = E_{12} = \hat{n}_A(\hat{n}_{12}^A)^T + \hat{n}_B(\hat{n}_{21}^B)^T$$

$$F_f = \hat{n}_f(\hat{n}_{12}^f)^T$$
Calculating Potential by GFandSlope (Factors $L$ and $\omega$)

$L (\omega)$: Substitution of integrals to derive equations for potential

- Integral along an edge $e = (i, j)$ on a face $f$
  
  $$L_e = \int_e \frac{1}{r} ds = \log \frac{r_i + r_j + e_{ij}}{r_i + r_j - e_{ij}}$$

- Steradian for a face $f$ (a signed area by projecting onto the unit sphere surface)

$$\omega_f = \int_{\text{triangle}} \frac{\Delta z}{r^3} ds = 2 \arctan\left(\frac{r_i \cdot r_j \times r_k}{r_i r_j r_k + r_i (r_j \cdot r_k) + r_j (r_k \cdot r_i) + r_k (r_i \cdot r_j)}\right)$$
The original GFandSlope is implemented in C++ which is incompatible with parallel frameworks (CUDA/OpenCL) in terms of...

- STL containers (vector, list) for the data structure with accesses to them by *iterators*
- Use of *external libraries* such as spice (NASA), vector3 (Zenitani)
- Implementation based on *object oriented programming* (e.g. Class)
- *Mingled code structure* of IO, gravitation calculation and other procedures

We ported to C with *modification of entire (data/code) structure*
- Class is replaced to Structure.
- Model data (vertices, edges, and faces) is managed by Arrays instead of STL containers.
Porting the Original GFandSlope for Acceleration

- Coordinates \((x, y, z)\) are treated as Structure.
- Replace external libraries (spice, and vector3) with original implementation.
  - They cannot be imported to GPU kernel codes.
  - Actually, they have been used only for fundamental arithmetic.
    - Replace with operations of structures corresponding to geometric vectors \(\mathbb{R}^3\)
- Moved IO to outside of the computation loop.
- As soon as the model is loaded, data structure to manage vertices and edges in faces are constructed.

**Preliminary Evaluation on CPU:** Intel(R) Xeon(R) Gold 5122 CPU @ 3.60GHz
(compiled with gcc ver. 5.4.0, 49152 faces of Itokawa model)

- Original C++: 4900s. \(\sim\) 1h 20min. \(\Rightarrow\) 1436s. (OpenMP, 4 thread)
- Ported C: 1323s. \(\Rightarrow\) 631s. (OpenMP, 4 threads)
GPU Implementation and Evaluation

- Implementation CUDA/OpenCL codes based on ported C code
  - Extract functions for the gravitation calculation (every GPU thread is assigned calculation for one face).
  - Others (IO and other preliminary calculations) are executed on CPU (data required by the calculation are transferred between CPU and GPU).
  - All floating point operations are processed in **double precision** (based on the original C++).

<table>
<thead>
<tr>
<th></th>
<th>Tesla K20</th>
<th>Tesla V100</th>
<th>GeForce RTX 3090</th>
</tr>
</thead>
<tbody>
<tr>
<td># of GPU cores</td>
<td>2496</td>
<td>5120</td>
<td>10496</td>
</tr>
<tr>
<td>Frequency</td>
<td>706MHz</td>
<td>1370MHz</td>
<td>1395MHz</td>
</tr>
<tr>
<td>Memory size</td>
<td>5GB</td>
<td>32GB</td>
<td>24GB</td>
</tr>
<tr>
<td>Peak Perf. (Double Precision)</td>
<td>1.17TFlops</td>
<td>7.0TFlops</td>
<td>556.0GFlops</td>
</tr>
<tr>
<td>Peak Perf. (Single Precision)</td>
<td>3.51TFlops</td>
<td>14.0TFlops</td>
<td>35.58TFlops</td>
</tr>
</tbody>
</table>

- How the computation time can be decreased compared with the original C++ code (4900s)?
- Measurement by using Itokawa model with different resolutions
  - # of faces: 50K, 200K, 780K, 3.1M
### Computation Time of Parallelized GFandSlope Code on GPU (DP)

<table>
<thead>
<tr>
<th>Devices \ $N_f$</th>
<th>50K</th>
<th>200K</th>
<th>780K</th>
<th>3.1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100 (CUDA)</td>
<td>0.45s</td>
<td>7.07s</td>
<td>1m52s</td>
<td>29m23s</td>
</tr>
<tr>
<td>K20 (CUDA)</td>
<td>3.54s</td>
<td>56.85s</td>
<td>14m58s</td>
<td>3h59m</td>
</tr>
<tr>
<td>RTX3090 (CUDA)</td>
<td>5.12s</td>
<td>1m18s</td>
<td>20m38s</td>
<td>5h30m</td>
</tr>
<tr>
<td>V100 (OpenCL)</td>
<td>0.71s</td>
<td>7.17s</td>
<td>1m54s</td>
<td>29m27s</td>
</tr>
<tr>
<td>RTX3090 (OpenCL)</td>
<td>6.14s</td>
<td>1m22s</td>
<td>20m54s</td>
<td>5h32m</td>
</tr>
</tbody>
</table>

**Table 1:** Computation time of CUDA and OpenCL codes in DP on GPUs

- Computation has been finished in 0.45sec. on Tesla V100 for 49152 faces model.
  - It is about **10000x** faster than the original C++ code on CPU(4900s.).
- Computation time also increases $O(n^2)$ for model resolutions.
- RTX3090 is inferior to K20 an old Tesla model due to lower double precision peak.
Alternation by Single Precision Floating Operations (SP)

- Calculation in single precision for all floating points in the gravitation calculation
- At least 2x speed-up is expected \textbf{if numerical accuracy does not drop} (especially 64x speed-up is expected in RTX3090).
  - If RTX is valuable for single precision operations, the installation cost gets also cheaper.

\begin{table}[h]
\centering
\begin{tabular}{l|cccc}
\hline
\text{Devices} & \text{$N_f$} & 50K  & 200K & 780K & 3.1M  \\
\hline
V100 (CUDA)   &   0.23s &  3.77s & 49.35s & 12m45s  \\
K20 (CUDA)    &   1.65s & 24.75s & 6m38s  & 1h44m    \\
RTX3090 (CUDA)&   0.17s &  2.27s & 35.00s & 9m22s    \\
\hline
V100 (OpenCL) &   0.19s &  2.53s & 40.85s & 10m29s  \\
RTX3090 (OpenCL)& 0.15s &  1.72s & 25.62s & 6m49s    \\
\hline
\end{tabular}
\caption{Computation time of CUDA and OpenCL codes in SP on GPUs}
\end{table}

Computation time of each environment are decreased as their ratio of peak performance between SP and DP.
Alternation by Single Precision Floating Operations (SP)

Comparison of relative errors (potential) in SP with DP

- With the smallest model (50K), errors concentrate on around $5.0 \times 10^{-6}$.
- Higher resolution model causes greater errors.
  - With 3.1M model, errors of 40% are more than 0.01.
  - Not practical in terms of numerical errors though computations in SP are faster than in DP.
Substitution some floating point operations with double precision

Accumulation which appears in Eqs.(1) to (3).

<table>
<thead>
<tr>
<th>Devices \ Model</th>
<th>50K</th>
<th>200K</th>
<th>780K</th>
<th>3.1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100</td>
<td>0.30s</td>
<td>5.19s</td>
<td>1m7s</td>
<td>17m55s</td>
</tr>
<tr>
<td>RTX3090</td>
<td>1.87s</td>
<td>28.14s</td>
<td>7m29s</td>
<td>2h6m</td>
</tr>
</tbody>
</table>

Table 3: Computation time of CUDA codes (SP2) on GPUs

The computation time is trade-off with precision (faster than DP, but slower than SP).
Operations Partially in Double Precision (SP2)

Comparison of relative errors (potential) in SP2 with DP

- Increase of errors due to scaling model resolution is mitigated.
- Most errors distribute around $10^{-6}$, which is enough small for practice
- 1.64x faster than in DP (18 min. on Tesla V100 for 3.1M model)
Discussion

- Succeeded in acceleration of GFandSlope code on GPU (for **uniform density** models)
  - The algorithm in GFandSlope will be extended to take density distribution into account.
  - The number of operations on GPUs also increases as extension of gravitation calculation.
  - It is inevitable to increase computation time for much higher resolution model (**an asteroid Ryugu is consisted of more than 10M faces**).

- Further **acceleration for parallel implementation** is needed.
  - Barnes-Hut tree algorithm (a way of mitigating computation complexity for $N$-body simulation)
    - Not suitable for GFandSlope due to calculations between faces and edges on the model
  - Converting models into a set of particles (e.g. Point cloud)
  - Decomposition to connecting tetrahedral elements as point mass (Kanamaru et al., 2019)
  - Solution as Point in polygon problems by winding number algorithm
We have accelerated gravitation calculation in GFandSlope code used for analysis of asteroids, which are 10000x faster than the existing implementation.

- Calculation is feasible easily on 1 GPU without dependency on external libraries.
- This works contributes to boost researches in the field of space science.
  - Numerical experiments with combination of different initial conditions and resolution models

The achievement of this research enables to improve an algorithm of GFandSlope for further practical analysis.

- Calculation for models with density distribution which requires much more numerical operations in the algorithm.
- We discuss optimization of GPU kernel codes in terms of parallel computation to process high resolution model faster.