SuperCon2019 本選問題

1　多体問題

多体問題は天体同士の重力相互作用、原子同士の静電相互作用に代表されるような、粒子同士の相互作用の問題である。2 体問題の場合は解析解が存在するが、3 体以上になると解析的に解くことはできないため、コンピュータ上で数値解析的に解くことになる。ただし、N 個の粒子同士の相互作用を素朴に計算すると N^2 に比例する計算量を要する。初期宇宙のシミュレーションやウィルスの分子シミュレーションなどでは N が何億にも及ぶため、現実的な時間で計算を行うためには多体問題の高速な近似解法が必要になる。

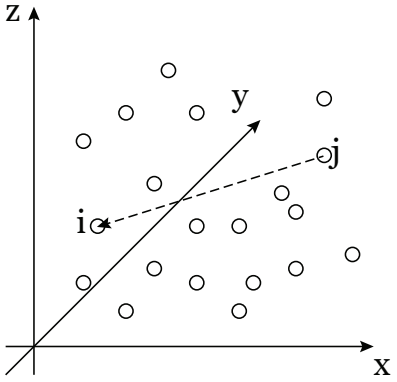

図 1: 点 i と点 j の相互作用

次に多体問題の最も簡単な例を示す。図 1 に示すように、x, y, z で表される 3 次元空間上に点 i と点 j があるとする。点 i の座標を $\vec{x}_i = (x_i, y_i, z_i)$、質量を m_i とし、点 j の座標を $\vec{x}_j = (x_j, y_j, z_j)$、質量を m_j とすると、点 j が点 i に及ぼす引力はその間のベクトル $\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$ を用いて

$$f_{ij} = \frac{Gm_i m_j}{|\vec{x}_{ij}|^2} \tag{1}$$

のように、それぞれの質量に比例し点 i と点 j の距離の二乗に反比例する関数で表される。ただし、$G = 6.67430 \times 10^{-11}[m^3/kg \cdot s^2]$ は重力定数である。また、実際には引力はスカラー量ではなく、点 j から点 i へ向かうベクトルの逆方向を向いている。そこで、点 i から点 j へ向かうベクトルの単位ベクトル-$\vec{x}_{ij}/|\vec{x}_{ij}|$ をかけると

$$\vec{f}_{ij} = -\frac{Gm_i m_j \vec{x}_{ij}}{|\vec{x}_{ij}|^3} \tag{2}$$

と表せる。これを全ての点 j について総和をとると点 i に働く力の総和が

$$\vec{f_i} = \sum_{j=1}^{N} \vec{f_{ij}} = \sum_{j=1}^{N} -\frac{Gm_i m_j \vec{x}_{ij}}{|\vec{x}_{ij}|^3} \tag{3}$$

のように求まる。ここで、ニュートンの運動の第 2 法則 $\vec{f_i} = m_i \vec{a}_i$ を用いると加速度

$$\vec{a}_i = -\sum_{j=1}^{N} \frac{Gm_j \vec{x}_{ij}}{|\vec{x}_{ij}|^3} \tag{4}$$

が計算できる。加速度 $\vec{a}_i$ は速度 $\vec{v}_i$ の時間微分 $\vec{a}_i = d\vec{v}_i/dt$ なので、時刻 t の速度 $\vec{v}_i(t)$ と加速度 $\vec{a}_i(t)$ から微小に進んだ時刻 $t+h$ における速度を

$$\vec{v}_i(t+h) \approx \vec{v}_i(t) + \vec{a}_i(t)h \tag{5}$$

のように求めることができる。座標と速度にも同様の関係があるため

$$\vec{x}_i(t+h) \approx \vec{x}_i(t) + \vec{v}_i(t+h)h \tag{6}$$

のように式 (5) で求めた速度を使って更新することができる。このとき h が十分に小さければ、いずれの式も微分の定義に近づくため、加速度の情報から正確に速度と座標を更新することができることが分かる。式 (4) を N 個全ての点 i と点 j の組み合わせについて計算し、式 (5) と (6) を全ての点 i について十分小さな h を用いて繰り返し解くことで、万有引力の法則にもとづく無数の天体の運動を正確に再現することができる。

2　多体問題の近似解法

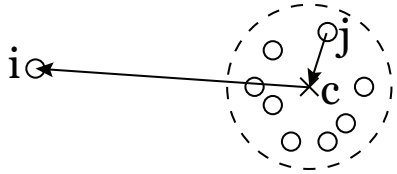

図 2: 複数の点の影響をまとめて近似する様子

多体問題の近似解法では複数の点 j が一つの点 i に及ぼす力をまとめて計算する。複数の点 j が図 2 の点線で描かれた球の中にあり、その球の中心を c とするとベクトル $\vec{x}_{ij}$ は $\vec{x}_{ic} + \vec{x}_{cj}$ に分解できる。このとき、$\vec{F}_{ij} = -G\vec{x}_{ij}/|\vec{x}_{ij}|^3$ を定義すると、式 (4) は

$$\vec{a}_i = \sum_{j-1}^{N} \vec{F}_{ij} m_j \tag{7}$$

と書ける。ここで、付録 A に示すような式変形を用いると点線で描かれた球の中にある N_j 個の点 j が点 i に及ぼす加速度は

$$M_n = \sum_{j=1}^{N_j} \frac{\vec{x}_{cj}^{\,n} m_j}{n!} \tag{8}$$

$$\vec{a}_i \approx \sum_{n=0}^{p} \vec{F}_{ic}^{(n)} M_n \tag{9}$$

のように一気にまとめて計算することができる。ただし、$\vec{F}^{(n)}$ は $\vec{F}$ の n 階微分、! は階乗を表す。ただし、この近似が成り立つのは $|\vec{x}_{ic}| \gg |\vec{x}_{cj}|$ を満たすとき、つまり図 2 の点線で囲まれた領域の大きさに対して、点線で囲まれた領域の中心から点 i までの距離が十分遠いときである。つまり、点 i からの距離が遠くなるほど、点 j は大きなくくりでまとめて計算しても同じ近似精度になる。では、全ての点 i と点 j の相互作用を最も効率よく計算するにはどうしたら良いだろうか？

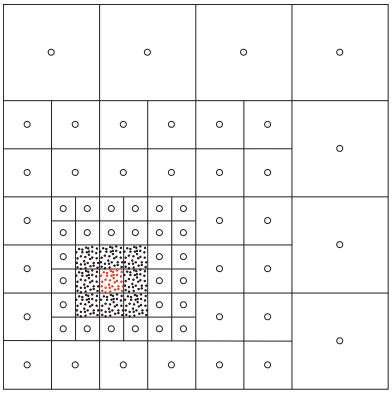

図 3: ある点 i の集合に対して、点 j をまとめて近似できる範囲の様子

図 3 に効率の良い計算方法の一例を示す。ここでは、分布している点を階層的な区画に分け、赤い点で示されるそれぞれの点 i に及ぼされる加速度をまとめて計算することを考える。近傍にある黒い点は、式 (8)、(9) の近似が成り立たないので式 (7) を直接計算する。その周りにある区画の中心にある白丸は図 2 の点 c の位置を表しており、その区画の中にある j はまとめて計算できることを表している。図 3 では赤い点で示したごく一部の点の加速度を求める様子を表しているが、実際には同様の計算を全ての点について行う必要がある。このためには、計算する点 i の位置によって点 j をまとめる範囲を柔軟に変えなければならない。これを実現するためのデータ構造を高速に作ることが多体問題の近似解法のカギとなる。

3 問題

上記のような多体問題の性質を用いることで N 個の点の相互作用の計算量を N^2 に比例するものから $N \log N$ に比例するものへと低減することができる。スーパーコン 2019 の本選では、このような多体問題の近似解法を用いて、ある一定の近似精度が得られている条件下で最速のものを勝ちとする。

3.1 計算方法

1. 与えられたコードを用いて $N = 100,000,000$ 個の点の初期配置と質量を計算する
2. 作成した近似解法を用いて全ての点に働く加速度を計算する
3. 計算は 1 ステップの加速度を求める部分のみ計算し、座標・速度の更新は行わない

4. こちらであらかじめ計算した、式 (7) の倍精度を用いた直接計算の加速度と比較する

5. 式 (10) を用いて全ての点を使って計算した誤差が下記に指定した値以下であれば有効な計算とする

6. そのとき近似解法の部分にかかった時間のみを計測し、その時間を競う

誤差の計算には式 (10) を用いる。式 (7) の直接計算から求めた点 i における加速度を $\vec{a}_i = \{[a_x]_i, [a_y]_i, [a_z]_i\}$、近似計算によって求めた加速度を $\hat{\vec{a}}_i = \{[\hat{a}_x]_i, [\hat{a}_y]_i, [\hat{a}_z]_i\}$ とすると

$$\text{誤差} = \sqrt{\frac{\sum_{i=1}^{N}([a_x]_i - [\hat{a}_x]_i)^2 + ([a_y]_i - [\hat{a}_y]_i)^2 + ([a_z]_i - [\hat{\vec{a}}_z]_i)^2}{\sum_{i=1}^{N}([a_x]_i)^2 + ([a_y]_i)^2 + ([a_z]_i)^2}} \tag{10}$$

として計算する。

3.2 練習と本番環境に関する注意点

- 練習ではこちらで提供するプログラムを使って初期条件を生成し、式 (10) を用いて全ての点を使って計算した誤差が 10^{-5} 以下のものを有効とする
- 本番ではこれとは異なる初期条件をこちらで指定し、式 (10) を用いて全ての点を使って計算した誤差が 10^{-4} 以下のものを有効とする

3.3 言語と実行環境

プログラムは C(C99), C++(C++14), CUDA(sm_60) のいずれかで書く。インラインアセンブリや PTX(CUDA の中間アセンブリ言語) を直接記述すること、出力されたものを改変することは認めない。gcc が標準的に提供している OpenMP によるスレッド並列化も用いてよい。プログラムの分割はファイル数は特に指定しない。使用できる外部のヘッダはシステムが標準で提供するもののみとし、ライブラリの使用は認めない。なお、標準ヘッダーファイルは改変してはならない。これらの注意に反するプログラムは失格とする。

本選の実行環境は表 1 に示す TSUBAME3.0 を用いる。コンパイラは TSUBAME3.0 で標準的に提供されている `gcc4.8.5` および `cuda9.2.148` を用いる。コンパイルオプションは C を用いる場合は

```
gcc -O -std=c99 プログラム名.c -lm
```

プログラムが C++で書かれている場合は

```
g++ -O -std=c++1y プログラム名.cpp
```

プログラムが CUDA と C++で書かれている場合は

```
nvcc --std c++11 -O -arch=sm_60 プログラム名.cu
```

とし、OpenMP を用いる場合は適宜`-fopenmp` のオプションをつける。

表 1: TSUBAME3.0 の仕様

CPU	Intel Xeon E5-2680 V4 Processor(Broadwell-EP, 14 コア, 2.4GHz) × 2
GPU	Tesla P100 (5.3TFLOPS@FP64, 10.6TFLOPS@FP32, 21.2TFLOPS@FP16) × 4
RAM	256GiB
SSD	Intel DC P3500 2TB (NVMe, PCI-E 3.0 x4, R2700/W1800)
ネットワーク	Intel Omni-Path 100Gb/s × 4

3.4 プログラムの提出方法

８月２２日（木）１３：００までにホームディレクトリの下にサブディレクトリ
`~/final`
を作り，そこに一つだけ
`final.cu`
という名前のファイルとして保存しておくこと．
ただし，サブディレクトリ`~/final`のパーミッションは
700
としておくこと．運営側で root 権限で回収します．
ただし，運営側は拡張子が`.c`のコードは
```
gcc -O -std=c99 -fopenmp final.c -lm
```
拡張子が`.cpp`のコードは
```
g++ -O -std=c++1y -fopenmp final.cpp
```
拡張子が`.cu`のコードは
```
nvcc --std c++11 -O -arch=sm_60 -Xcompiler "-fopenmp" final.cu
```
でコンパイルしたときの性能を評価します．

3.5 選考基準

- 練習ではこちらで提供する 4 種類の初期分布 ($N = 10^5, 10^6, 10^7, 10^8$) を用いて近似計算を行い、式 (10) を用いて計算した誤差が 10^{-5} 以下であること
- 本番ではこちらで提供する練習とは異なる 1 種類の初期分布 ($N = 10^8$) を用いて近似計算を行い、式 (10) を用いて計算した誤差が 10^{-4} 以下であること
- この誤差の基準を満たすもののうち最速のものを勝ちとする
- 誤差を計算する際の比較対象となる加速度はこちらから提供するものを用いること
- 時間を計測するのは加速度の近似計算の部分のみとし、初期配置の生成や誤差の計算の部分は計測時間に含めない
- 式 (9) は大学数学の範囲なので、こちらでサンプルプログラムを提供する

- 木構造の生成や相互作用リストの生成用のサンプルプログラムもこちらで提供する
- これらの近似計算用のサンプルプログラムは自由に書き換えて良い
- 近似計算には必ずしも式 (8)、(9) を使う必要はなく、誤差が基準値以下にさえなればどのような近似を用いても良い
- 近似計算自体には答え合わせ用の加速度のデータはいかなる形でも用いてはならない
- 木構造を生成したり、点をソーティングする時間も近似計算の時間に含まれる
- 一回の計算で使用する GPU は一つまでとする

3.6 解法のヒント

多体問題の近似解法の精度は式 (8)、(9) の p を大きくすると高くなるが、式 (9) を一回解くための計算時間は増加する。また、図 3 の区画をより小さく割っていくと p が同じままでも近似精度を向上できるが、今度は式 (9) を計算する回数が増加して結局計算時間が増加する。式 (10) を用いて計算した誤差が 10^{-5} 以下という制約の中で計算時間を短縮するにはこれらのバランスを考慮する必要がある。また、空間の階層的分割には Z 階数曲線などを用いると効率的である。これはランダムに配置された点のソーティングの問題に帰着する。近傍の点の相互作用は必ず式 (7) の直接計算を行うので、その部分を徹底的にチューニングするというのも一つの手である。

付録 A 近似解法の計算式の導出

多変数関数のテイラー展開

$$\begin{aligned} f(x+h_x, y+h_y, z+h_z) &= f(x,y,z) + \frac{df(x,y,z)}{dx}h_x + \frac{df(x,y,z)}{dy}h_y + \frac{df(x,y,z)}{dz}h_z \\ &+ \frac{1}{2}\frac{d^2f(x,y,z)}{dx^2}h_x^2 + \frac{1}{2}\frac{d^2f(x,y,z)}{dy^2}h_y^2 + \frac{1}{2}\frac{d^2f(x,y,z)}{dz^2}h_z^2 \\ &+ \frac{d^2f(x,y,z)}{dxdy}h_xh_y + \frac{d^2f(x,y,z)}{dydz}h_yh_z + \frac{d^2f(x,y,z)}{dzdx}h_zh_x \\ &+ \cdots \end{aligned}$$

に対して、次のような略記法を導入すると

$$\begin{aligned} \frac{f^{(0)}(\vec{x})}{0!}\vec{h}^0 &= f(x,y,z) \\ \frac{f^{(1)}(\vec{x})}{1!}\vec{h}^1 &= \frac{df(x,y,z)}{dx}h_x + \frac{df(x,y,z)}{dy}h_y + \frac{df(x,y,z)}{dz}h_z \\ \frac{f^{(2)}(\vec{x})}{2!}\vec{h}^2 &= \frac{1}{2}\frac{d^2f(x,y,z)}{dx^2}h_x^2 + \frac{1}{2}\frac{d^2f(x,y,z)}{dy^2}h_y^2 + \frac{1}{2}\frac{d^2f(x,y,z)}{dz^2}h_z^2 \\ &+ \frac{d^2f(x,y,z)}{dxdy}h_xh_y + \frac{d^2f(x,y,z)}{dydz}h_yh_z + \frac{d^2f(x,y,z)}{dzdx}h_zh_x \end{aligned}$$

となる．この一般形は以下のように書け

$$\begin{aligned} f\left(x+h_x, y+h_y, z+h_z\right) &= \sum_{n_x=0}^{\infty}\sum_{n_y=0}^{\infty}\sum_{n_z=0}^{\infty}\frac{1}{n_x!}\frac{1}{n_y!}\frac{1}{n_z!}\frac{d^{n_x}}{dx^{n_x}}\frac{d^{n_y}}{dy^{n_y}}\frac{d^{n_z}}{dz^{n_z}}f(x,y,z) \\ &\simeq \sum_{n_x=0}^{P}\sum_{n_y=0}^{P-n_x}\sum_{n_z=0}^{P-n_x-n_y}\frac{1}{n_x!}\frac{1}{n_y!}\frac{1}{n_z!}\frac{d^{n_x}}{dx^{n_x}}\frac{d^{n_y}}{dy^{n_y}}\frac{d^{n_z}}{dz^{n_z}}f(x,y,z) \end{aligned}$$

それは，以下のように簡略化できる

$$f(\vec{x}+\vec{h}) = \sum_{n=0}^{p}\frac{f^{(n)}(\vec{x})}{\vec{n}!}\vec{h}^n \tag{11}$$

ただし、p はテイラー展開の次数である。これらの微分は偏微分なので正確には ∂ 記号を用いるべきであるが、ここでは高校数学の範囲内の表記法で d を用いている。上記のテイラー展開の $f=\vec{F}_{ij}$、$\vec{x}=\vec{x}_{ic}$、$\vec{h}=\vec{x}_{cj}$ とおくことで

$$\vec{F}_{ij}(\vec{x}_{ic}+\vec{x}_{cj}) = \sum_{n=0}^{p}\frac{\vec{F}_{ic}^{(n)}(\vec{x}_{ic})}{\vec{n}!}\vec{x}_{cj}^{n} \tag{12}$$

のように書ける。これを図 2 の点線で囲まれた球の中の N_j 個の点 j について積算すると点 i での加速度は

$$\vec{a}_i(\vec{x}_i) = \sum_{j=1}^{N_j}\sum_{n=0}^{p}\frac{\vec{F}_{ic}^{(n)}(\vec{x}_{ic})}{\vec{n}!}\vec{x}_{cj}^{n}m_j \tag{13}$$

となる。ここで、$\vec{F}_{ic}^{(n)}(\vec{x}_{ic}) =$ は j に依存しないので $\sum_{j=1}^{N_j}$ の中で何度も計算する必要はなく

$$M_n = \sum_{j=1}^{N_j} \frac{\vec{x}_{cj}^{\,n} m_j}{\vec{n}!} \tag{14}$$

$$\vec{a}_i(\vec{x}_i) = \sum_{n=0}^{p} \vec{F}_{ic}^{(n)}(\vec{x}_{ic}) M_n \tag{15}$$

のように 2 段階に分けることで計算量を大幅に低減できる。ただし、$\vec{F}_{ic}^{(n)}(\vec{x}_{ic})$ は分母に $|\vec{x}_{ic}|^n$ がくるような形になるため、式 (12) のテイラー展開が収束するためには

$$\frac{|\vec{x}_{cj}|}{|\vec{x}_{ic}|} < 1 \tag{16}$$

の条件を満たす必要がある。これは図 2 において、点 i と c の距離が点 j と c の距離よりも大きくなくてはならないことを表している。