INF560

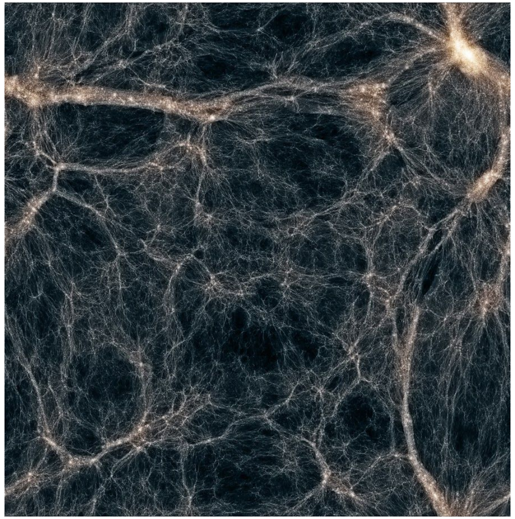
Towards Paralleled N-Body Interaction

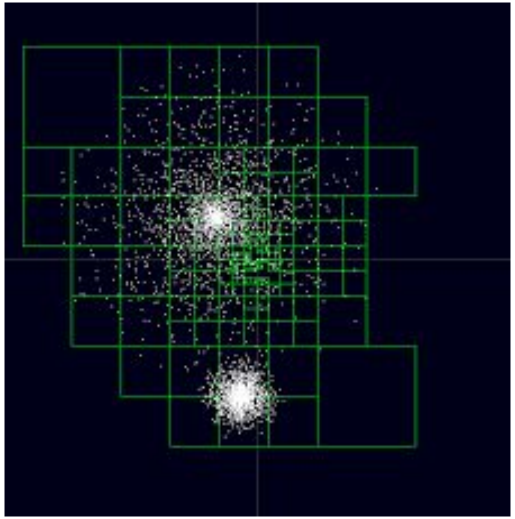
Zhengqing Liu, Jiongyan Zhang

16/03/2022

Outline

1. Background





N-Body Simulation

2. Brute Force

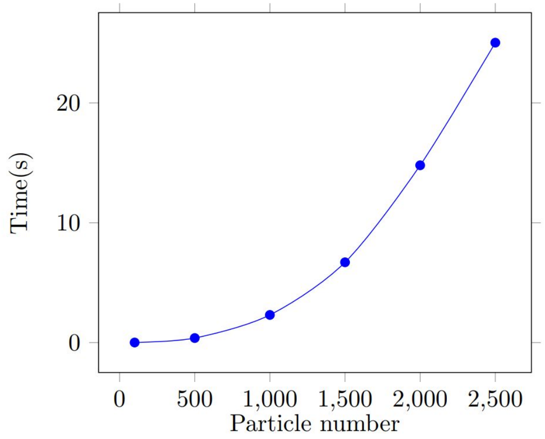
2.1 Sequential Brute Force

- Two essential tasks (for-loops)
  - Force computation
  - Position movement

```
for(j=0; j<nparticles; j++) {
  particle_t*p = &particles[j];
  /* compute the force of particle j on particle i */
  compute_force(&particles[i], p->x_pos, p->y_pos, p->mass);
}

for(i=0; i<nparticles; i++) {
  move_particle(&particles[i], step);
}
```

$O(n^2)$

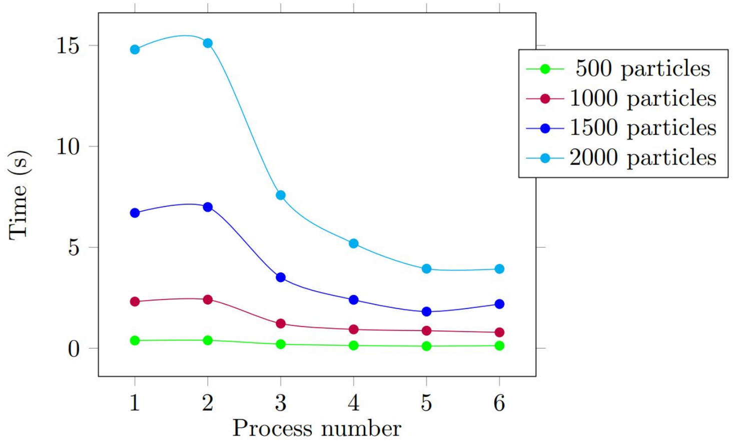


2.2 MPI Brute Force

4 processes, 10 particles
- each slave: 10/3 = 3
- master = 10 % 3 =1

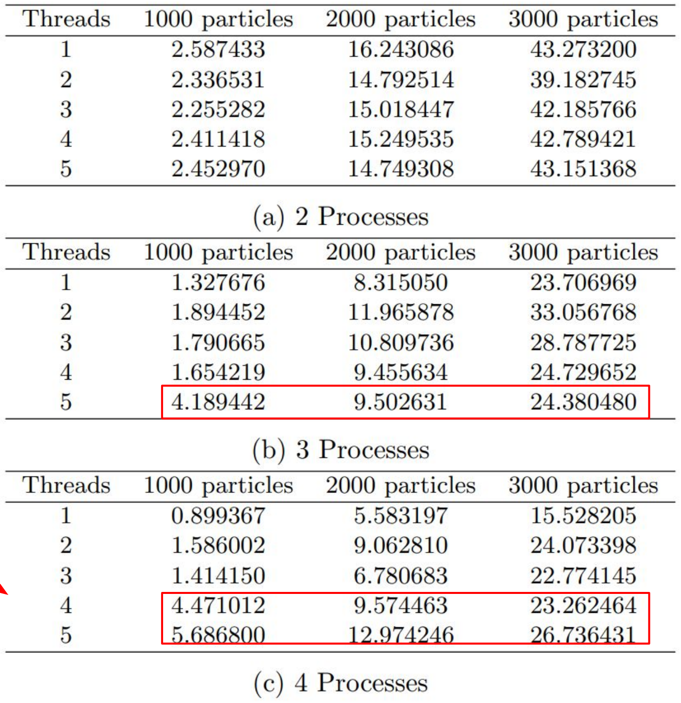
- Master and Slave process
  - master - distribute tasks, move position, sync, do the *modulus* force computation
  - slave - force computation (equally)
- Collective communication (Bcast, gatherv) - new MPI datatype
- Sync max_acc, max_speed using MPI_Send

```
/* Create MPI type for collective communication */
MPI_Datatype particle_mpi_t;
int blocklens[1] = {7};
MPI_Aint offsets[1] = {0};
MPI_Datatype types[1] =  {MPI_DOUBLE};
MPI_Type_create_struct(1, blocklens, offsets, types, &particle_mpi_t);
MPI_Type_commit(&particle_mpi_t);
```



2.3 MPI + OpenMP Brute Force

- Fine-grain hybrid programming
  - Distribute tasks among MPI ranks
  - Omp *parallel for* region for each subpart
    - Loop scheduling
- Outcome
  - Trivial and unstable (even worse) effects than pure MPI
  - Obvious *slow-down* with many threads and processes
    - Amdahl’s law + launching and maintaining costs on thr/proc



Threads	1000 particles	2000 particles	3000 particles
1	2.587433	16.243086	43.273200
2	2.336531	14.792514	39.182745
3	2.255282	15.018447	42.185766
4	2.411418	15.249535	42.789421
5	2.452970	14.749308	43.151368

(a) 2 Processes

Threads	1000 particles	2000 particles	3000 particles
1	1.327676	8.315050	23.706969
2	1.894452	11.965878	33.056768
3	1.790665	10.809736	28.787725
4	1.654219	9.455634	24.729652
5	4.189442	9.502631	24.380480

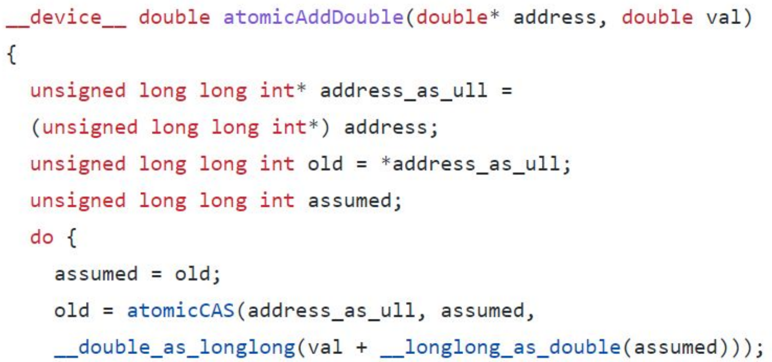
(b) 3 Processes

Threads	1000 particles	2000 particles	3000 particles
1	0.899367	5.583197	15.528205
2	1.586002	9.062810	24.073398
3	1.414150	6.780683	22.774145
4	4.471012	9.574463	23.262464
5	5.686800	12.974246	26.736431

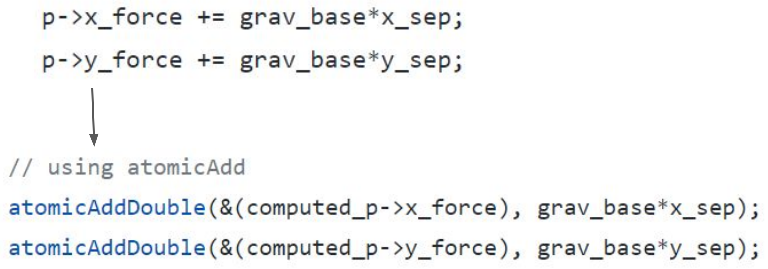
(c) 4 Processes

2.3 MPI + OpenMP + CUDA Brute Force

1. **extern "C" void cuda_compute_force(int i, int nparticles, particle_t * p)**
2. **__global__ void __compute_force__ (int * i, int * nparticles, particle_t * d_p)**



```
__device__ double atomicAddDouble(double* address, double val)
{
  unsigned long long int* address_as_ull =
  (unsigned long long int*) address;
  unsigned long long int old = *address_as_ull;
  unsigned long long int assumed;
  do {
    assumed = old;
    old = atomicCAS(address_as_ull, assumed,
    __double_as_longlong(val + __longlong_as_double(assumed)));
```



```
p->x_force += grav_base*x_sep;
p->y_force += grav_base*y_sep;
```

$\downarrow$

```
// using atomicAdd
atomicAddDouble(&(computed_p->x_force), grav_base*x_sep);
atomicAddDouble(&(computed_p->y_force), grav_base*y_sep);
```

Processes	100 particles	500 particles	1000 particles
2 (No CUDA)	0.007055	0.385060	2.336531
2	0.631886	13.081082	78.902757
3	1.409775	22.233090	92.376449
4	1.477007	22.345278	109.797773

3. Barnes Hut

3.1 Parallelism strategy

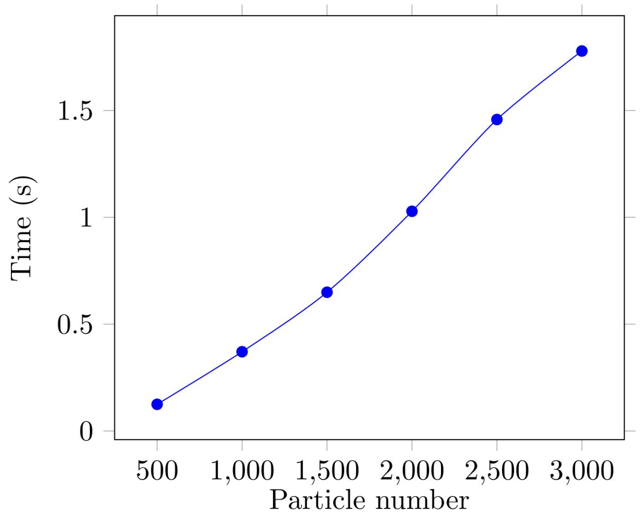
The main idea is to create a for-loop for computing the force.

1. Flatten the quad-tree: *Linear* quad-tree	→	Fail
2. Recursion to non-recursion: Iteration method	→	Not good
3. Tricky coding: Derive the *address* of particles	→	**Succeed**

3.2 Non-parallelism performance

Barnes Hut Algorithm

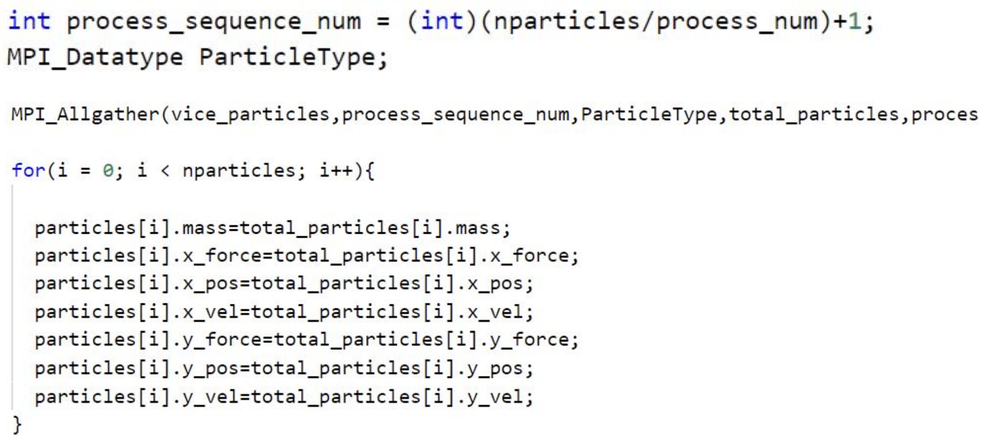
Better efficiency in computing the n-body problem. It does not require go through all the particles again and again.



3.3 MPI Barnes Hut

Gradually assign cores to the program

- Slightly accelerate the program, and with more cores assigned, speed rises as well.
- Too many cores may lead to the performance decreasing.

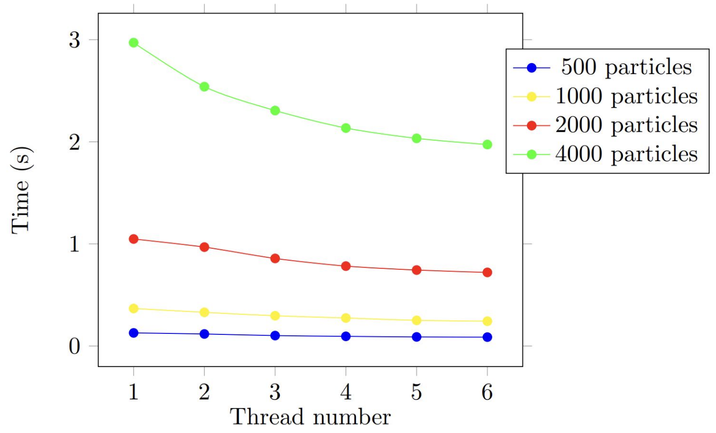


```
int process_sequence_num = (int)(nparticles/process_num)+1;
MPI_Datatype ParticleType;

MPI_Allgather(vice_particles,process_sequence_num,ParticleType,total_particles,proces

for(i = 0; i < nparticles; i++){

  particles[i].mass=total_particles[i].mass;
  particles[i].x_force=total_particles[i].x_force;
  particles[i].x_pos=total_particles[i].x_pos;
  particles[i].x_vel=total_particles[i].x_vel;
  particles[i].y_force=total_particles[i].y_force;
  particles[i].y_pos=total_particles[i].y_pos;
  particles[i].y_vel=total_particles[i].y_vel;
}
```

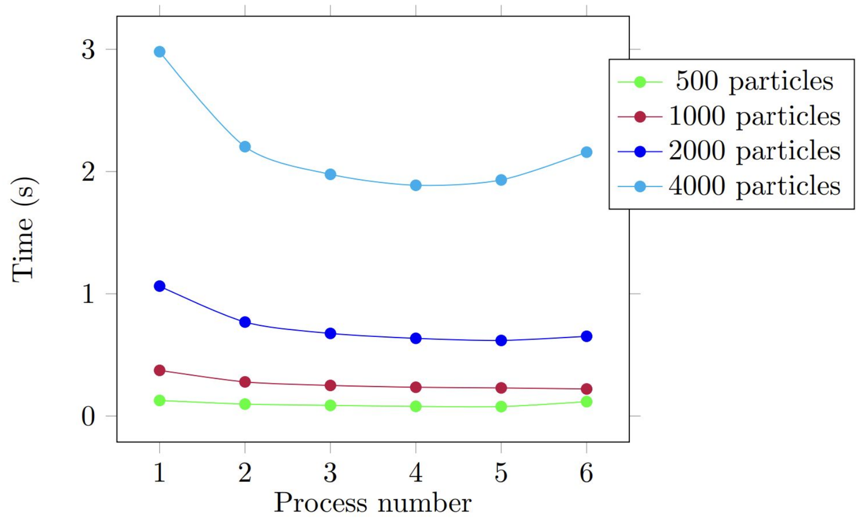


3.4 OpenMP Barnes Hut

Gradually assign threads to the program

- Shared memory system requires less communication cost compared with MPI.
- However, the speedup decreases as well.
- Context switch, memory synchronization, etc., take the main influence.

```
#pragma omp parallel for schedule(dynamic)
for(i=0;i<nparticles;i++) {
  compute_force_in_node(particles[i].node);
}
```

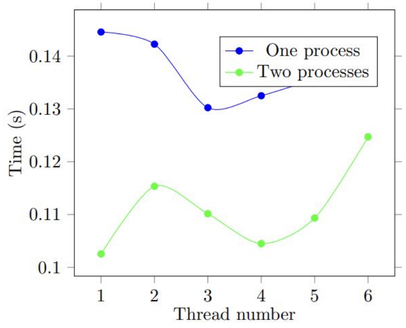


3.5 Hybrid Barnes Hut (1)

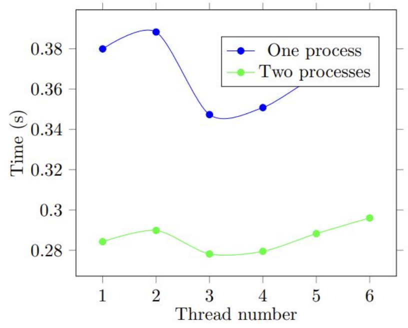
Hybrid structure is to use the nested approach, making each process capable of using multiple threads to carry out computing tasks.

6 THREADS, 3 PROCESSES, 4 AMOUNTS:

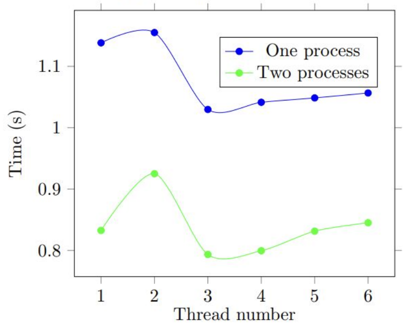
- Different input parameters to test the hybrid structure
- The result shows that the MPI does help to decrease the execution time, while 2 threads lead to a time peak.



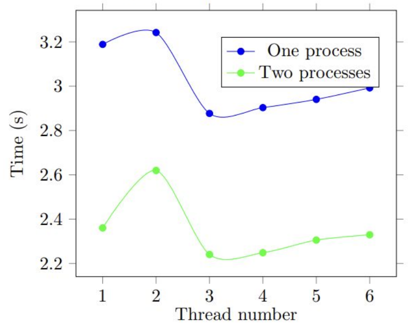
(a) 500 particles



(b) 1000 particles



(c) 2000 particles

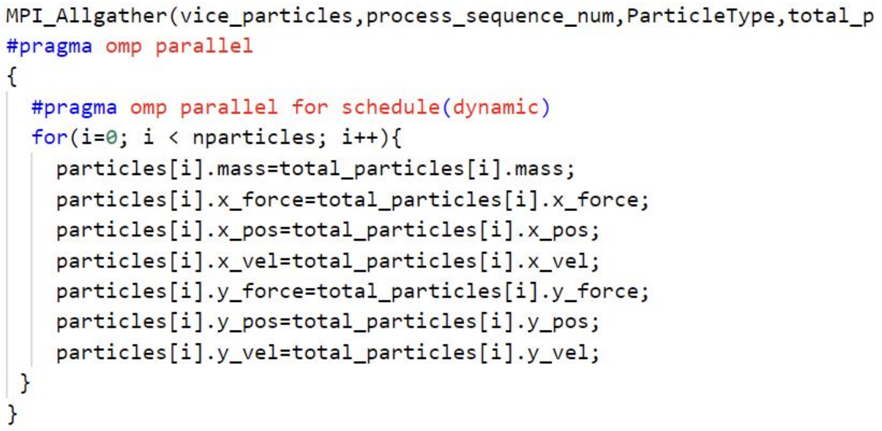


(d) 4000 particles

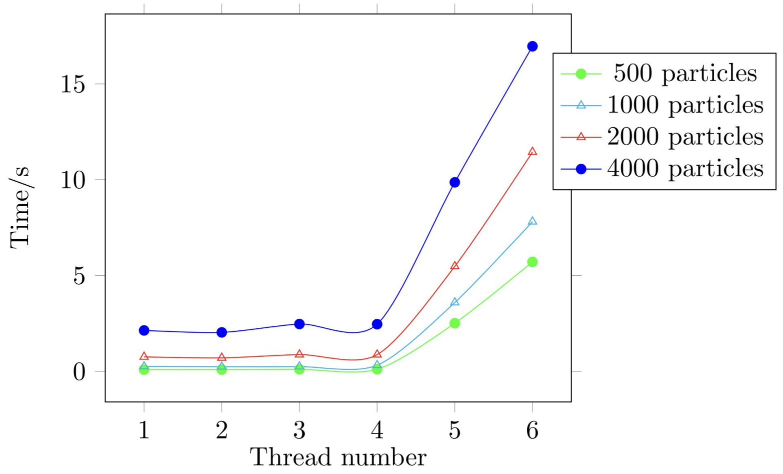
3.5 Hybrid Barnes Hut (2)

Abnormality while utilising >=3 processes

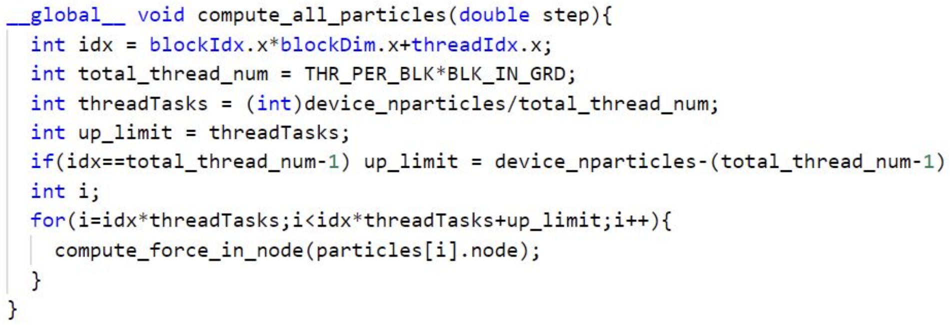
performance degrades sharply (mainly with >=5 threads) in any particle number level.



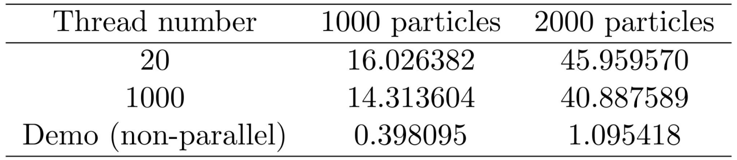
```
MPI_Allgather(vice_particles,process_sequence_num,ParticleType,total_p
#pragma omp parallel
{
  #pragma omp parallel for schedule(dynamic)
  for(i=0; i < nparticles; i++){
    particles[i].mass=total_particles[i].mass;
    particles[i].x_force=total_particles[i].x_force;
    particles[i].x_pos=total_particles[i].x_pos;
    particles[i].x_vel=total_particles[i].x_vel;
    particles[i].y_force=total_particles[i].y_force;
    particles[i].y_pos=total_particles[i].y_pos;
    particles[i].y_vel=total_particles[i].y_vel;
  }
}
```



3.6 CUDA Barnes Hut



```
__global__ void compute_all_particles(double step){
  int idx = blockIdx.x*blockDim.x+threadIdx.x;
  int total_thread_num = THR_PER_BLK*BLK_IN_GRD;
  int threadTasks = (int)device_nparticles/total_thread_num;
  int up_limit = threadTasks;
  if(idx==total_thread_num-1) up_limit = device_nparticles-(total_thread_num-1)
  int i;
  for(i=idx*threadTasks;i<idx*threadTasks+up_limit;i++){
    compute_force_in_node(particles[i].node);
  }
}
```



Thread number	1000 particles	2000 particles
20	16.026382	45.959570
1000	14.313604	40.887589
Demo (non-parallel)	0.398095	1.095418

The idea is to establish a **quad-tree** in GPU.

The CUDA model has a very bad effect on the algorithm.

1. The GPU is not suitable for the operations on a complicated structure, which is limited by the device performance;
2. The Barnes Hut algorithm is not a computing intensively program.

Summary

In this work, we target at paralleling N-body simulation (Brute Force and Barnes Hut) with MPI, OpenMP and CUDA.

- Brute Force
  - MPI largely speeds up the computation
  - OpenMP provides slight enhancement in a fine-grain mode
  - CUDA slows down the task at several magnitude of orders
- Barnes Hut
  - Handle the complex tree structure via memory share or structure synchronization
  - MPI: using space to avoid transfer, leading to acceleration
  - OpenMP: shared memory to avoid transfer to speed up
  - Hybrid: be careful of the device limitation
  - CUDA: not suitable for the complex structure and the non-computing-intensive algorithm

✔ What we learned/acquired?
  - Multiple parallelism paradigms
  - Various factors which influence the parallelism efficiency
  - Proper collaboration with tutors and mates
- Which skills should we keep reinforcing in the future?
  - Habits/styles of Coding
  - Analyze deeper influence on parallelism

Reference

[1] Sverre J Aarseth and Sverre Johannes Aarseth. Gravitational N-body simulations: tools and algorithms. Cambridge University Press, 2003.
[2] Maida Arnautovi ́c et al. “Parallelization of the ant colony optimization for the shortest path problem using OpenMP and CUDA”. In: 2013 36th International Convention on Information and Communication Technology, Electronics and Mi-croelectronics (MIPRO). IEEE. 2013, pp. 1273–1277.
[3] Marco Bertini. GPU Programming Basics. 2017. url: https://www.micc.unifi.it/bertini/download/gpu-programming-basics/2017/gpu_cuda_5.pdf.
[4] John Smith and S-F Chang. “Quad-tree segmentation for texture-based image query”. In: Proceedings of the second ACM international conference on Multimedia. 1994, pp. 279–286.
[5] Michele Trenti and Piet Hut. “N-body simulations (gravitational)”. In: Scholarpedia 3.5 (2008), p. 3930.
[6] Laurens Van Der Maaten. “Barnes-hut-sne”. In: arXiv preprint arXiv:1301.3342 Add to Citavi project by ArXiv ID (2013).
[7] Wikipedia. N-body Simulation. [Online; accessed 13-March-2022]. 2022. url: https://en.wikipedia.org/wiki/N-body_simulation.

ÉCOLE POLYTECHNIQUE

IP PARIS

THANKS

Zhengqing Liu, Jiongyan Zhang

16/03/2022