联系人 胡航：HUHANG123@SJTU.EDU.CN
研究海报 CNP20223

Porting and Optimizing CUBE – Large Scale Cosmological N-body simulation on NVIDIA V100

Authors: Hang Hu[1], Shenggan Cheng[1], Minhua Wen[1], Hao-ran Yu[2], James Lin[1]

Institution: Center for HPC, Shanghai Jiao Tong University[1]

School of Physics and Astronomy, Shanghai Jiao Tong University[2]

Background

Large scale N-body cosmic simulation has great scientific significance for studying cosmos evolution and finding specific cosmic structure. The main method of cosmological simulation is the Particle-Mesh (PM) algorithm combined with Particle-Particle (PP) algorithm. When simulating scale is large, the calculation and memory consumption of this method are very huge. This poster presents one optimizing method for CUBE (a typical N-body cosmological simulation program).

Program Structure

CUBE solves the gravitational force using the PMPM algorithm, with optional extended-PP force modules for increased accuracy. The PMPM algorithm solves this problem by splitting the gravitational force F_G into a short-range force F_s and a long-range force F_l, $F_G = F_s + F_l$. The long-range force F_l is calculated by 3D FFT, the short-range force F_s is calculated by the gravity law. As Figure 1 shows, CUBE divides the simulation volume L^3 into many cubic sub-volumes $V_{tile} = (L / N_{image})^3$, called images. Each image is divided into nnt^3 images, and each tile is further divided into $N_{fine}{}^3$ fine cells, 4^3 fine cells are combined as a coarse cell.

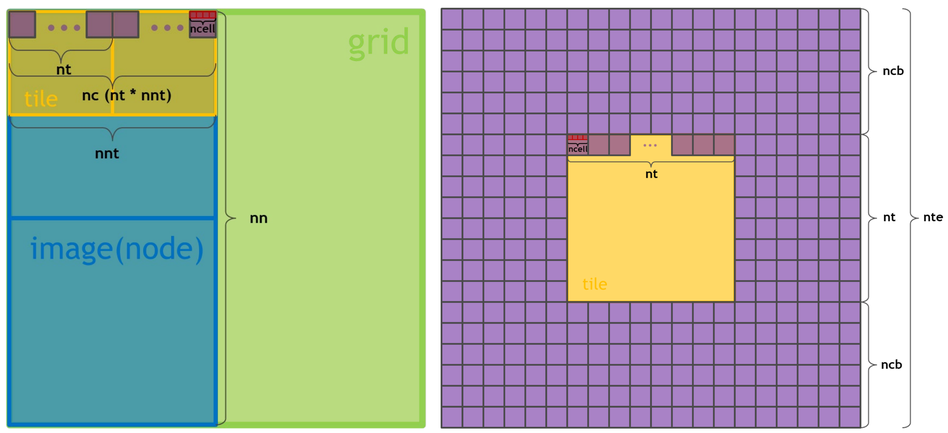

Figure 1: Structure of CUBE

Optimization

1. Change data addressing method. The origin data is addressed by the coarse cell sequence, and the particle's data in array is out of order. We use bucket algorithm to restore the particle's data by the fine cell sequence to achieve sequential load/store.
2. Use Struct of Array (SoA) to replace Array of Struct (AoS). Put all particles' x-axis coordinates together to cooperate with the synchronous load/store operation in the same warp. Shown as Figure 2.
3. Use shared memory to store the temporary array for calculation. The PP algorithm use two particles' positions and gravity law to calculate the force between them. Because this calculation is $O(N^2)$, when load positions from the global memory, the load operation takes a long time. So we use shared memory to store the particle positions in one coarse cell, which is calculated by a GPU block.
4. Use dynamic parallelism when particles in one fine cell is very huge. The particles will gather together as the simulation progress, some fine cells may have many particles while other neighbor fine cells have few. There will be severe load unbalanced between threads in the same warp. We use dynamic parallelism in the fine cell which have many particles and launch a new kernel in this thread to do further parallelization, shown as Figure 3.

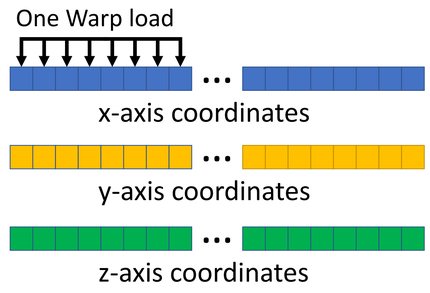

Figure 2: SoA of particles' coordinates

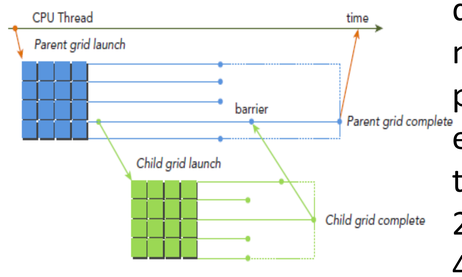

Figure 3: Dynamic parallelism

Experiment Setup

We implement our optimized version on NVIDIA V100 and compared with the CPU version on Intel Xeon Gold 6248 to evaluate the effectiveness of optimization. The results shows in Figure 4.

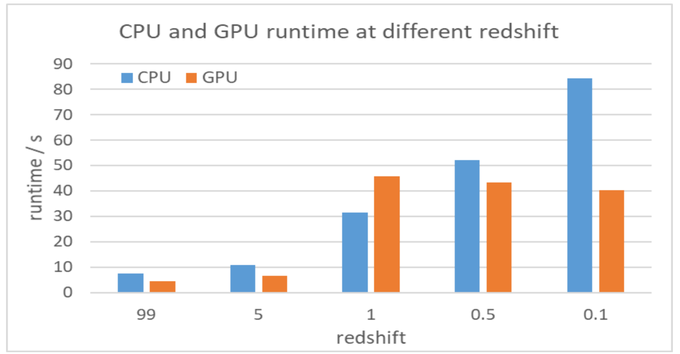

Figure 4: The runtime of GPU version and CPU version

Details and System Configuration:

1) This case has about 16.77 billion particles.
2) The CPU platform has 40 cores per node, 192 GB memory.
3) The main time-consuming part is at low redshift (< 0.5).

What's Next

In this poster, we presented an optimization case on NVIDIA V100. We proposed a series of optimization methods including changing data addressing method, use SoA, shared memory to improve the memory access performance and use dynamic parallelism to eliminate the load unbalance between threads. Finally, we got an acceleration up to 2 times compared with Intel Xeon Gold 6248 40 cores. In future, more optimization will be done to simulate larger scale.