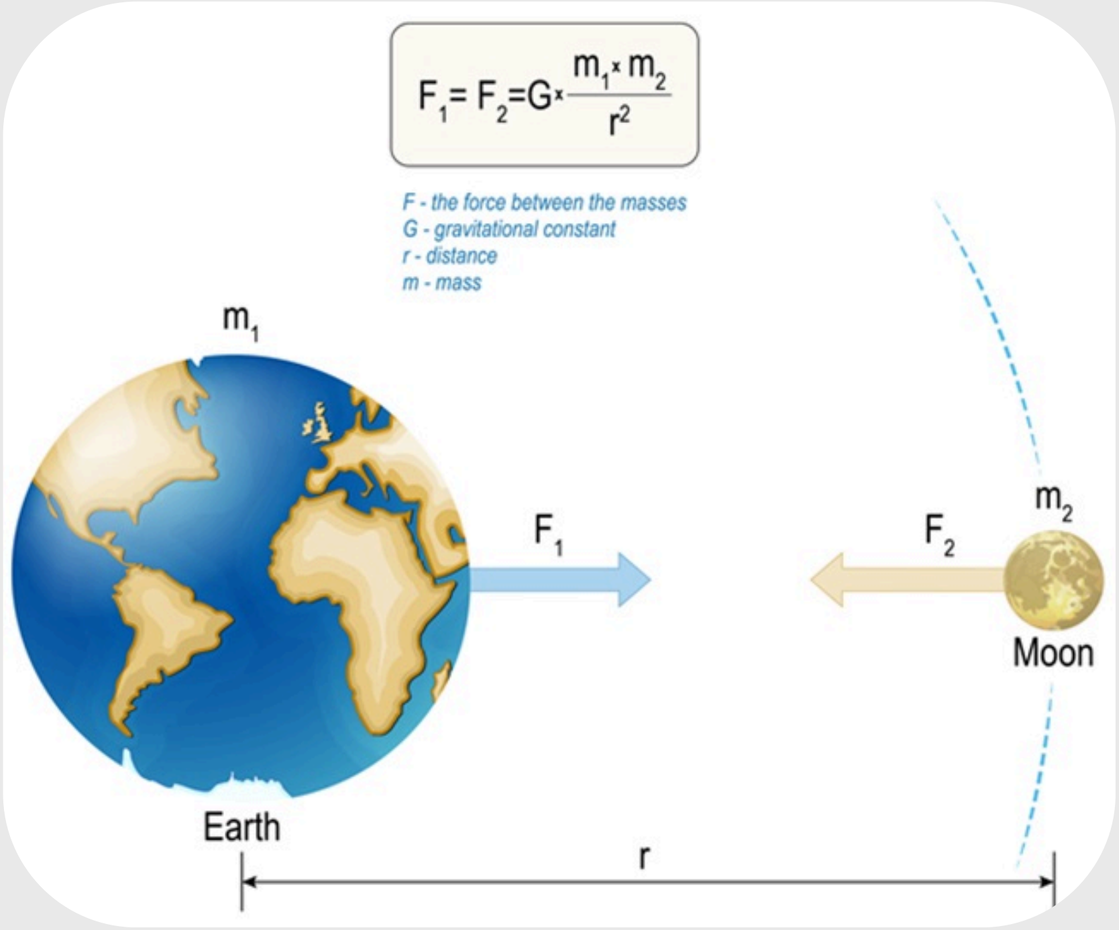
N-BODY SIMULATIONS WITH CUDA

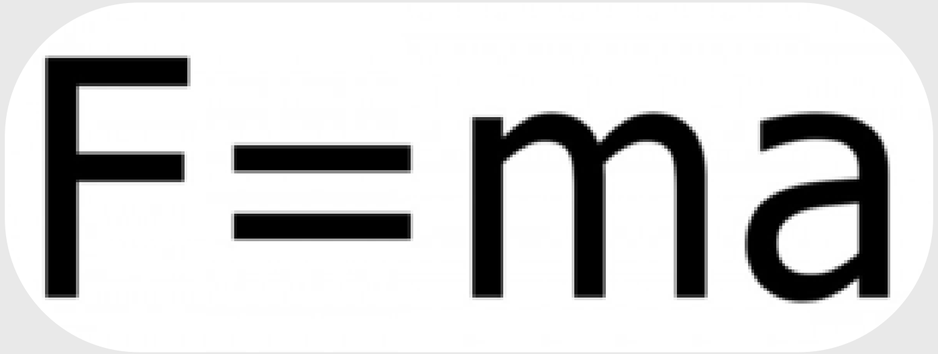
6610450871 ชนพัฒน์ โชติกุลรัตน์

CS343-65 (68-1) Parallel Computing with CUDA



https://www.boloji.com/articles/54461/newtons-law-of-universal-gravitation

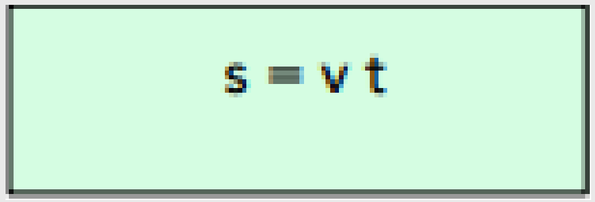
N-body problem คือ ปัญหาคลาสสิกในทางฟิสิกส์และดาราศาสตร์ ที่เกี่ยวกับการทำนายการเคลื่อนที่ของวัตถุจำนวน N ชิ้น ที่มีปฏิสัมพันธ์กันผ่านแรงโน้มถ่วง



https://www.pathwayz.org/Tree/Plain/F%3Dma

$$V = u + at$$

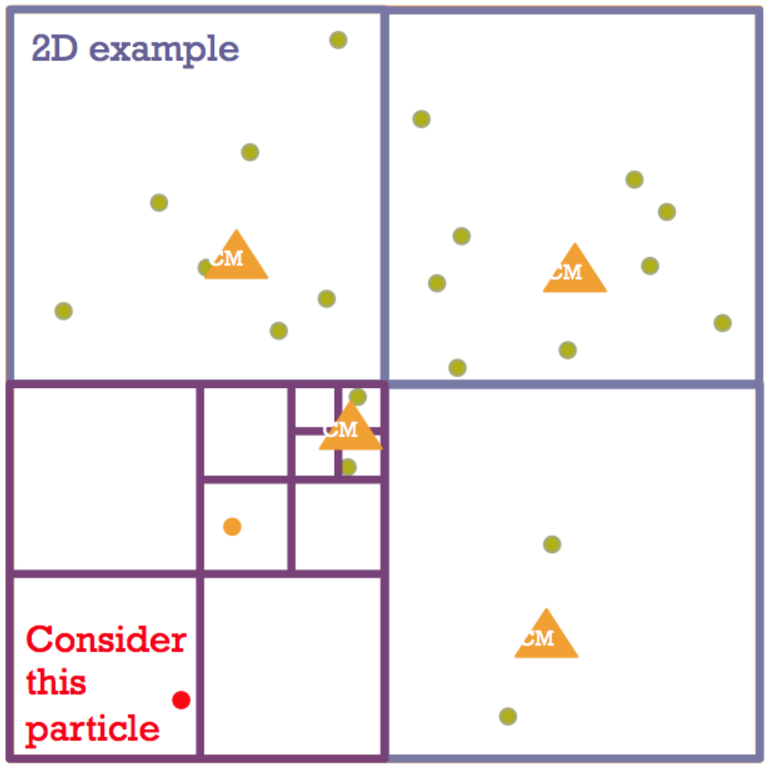
https://www.youtube.com/watch?v=dlT-3PFTFCo

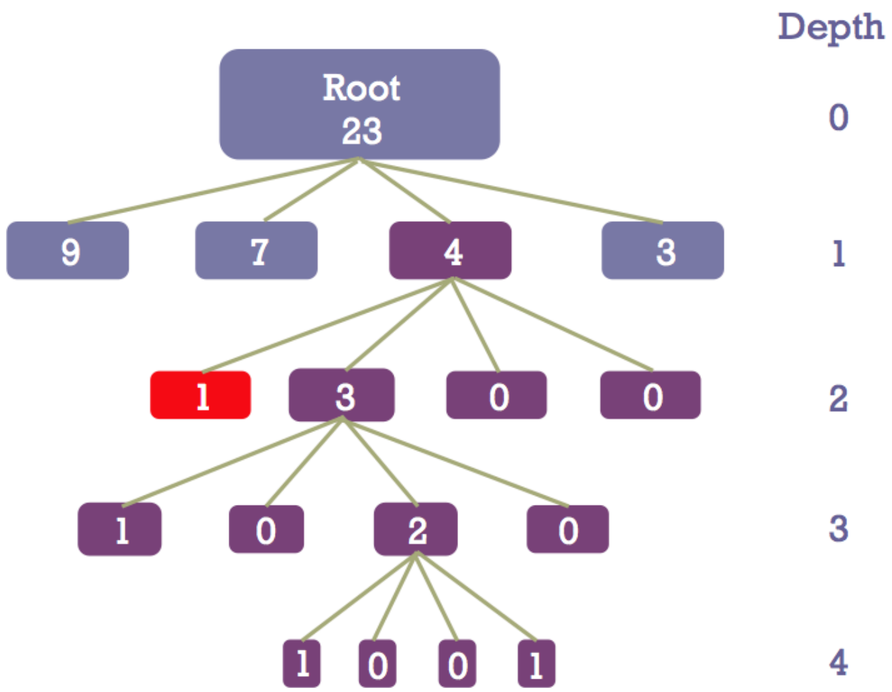


https://studymind.co.uk/notes/calculating-speed/

BEST SEQUENTIAL SOLUTION

BARNES HUT ALGORITHM



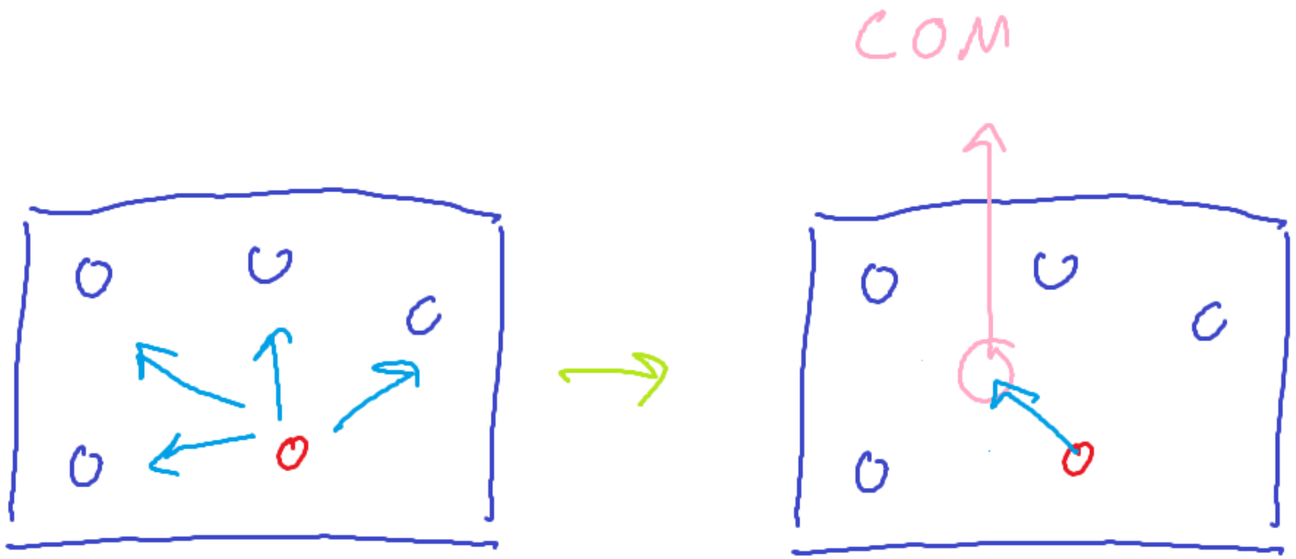


https://studymind.co.uk/notes/calculating-speed/

Theta

ค่าเกณฑ์ความละเอียด

box length / distance < Theta




```
((tree->size / d) < theta || tree->is_leaf == true)

  double dx = tree->COM_x - p->x;
  double dy = tree->COM_y - p->y;
  double dz = tree->COM_z - p->z;
  double dist_sqr = dx*dx + dy*dy + dz*dz + softening*softening;

  double invDist = 1.0/ sqrtf(dist_sqr);
  double invDistCube = invDist * invDist * invDist;

  double magnitude = G * tree->mass * invDistCube;

  ax += dx * magnitude;
  ay += dy * magnitude;
  az += dz * magnitude;
```

box length / distance >= Theta

```
else
{
    for(int i =0 ;i<8;i++)
    {
        if(tree->child_particles[i] != nullptr)
        {
            calculate_acceleration(p, tree->child_particles[i],ax,ay,az);
        }
    }
}
```

TIME COMPLEXITY

WORK:O(NLOGN)

SPAN:O(NLOGN)

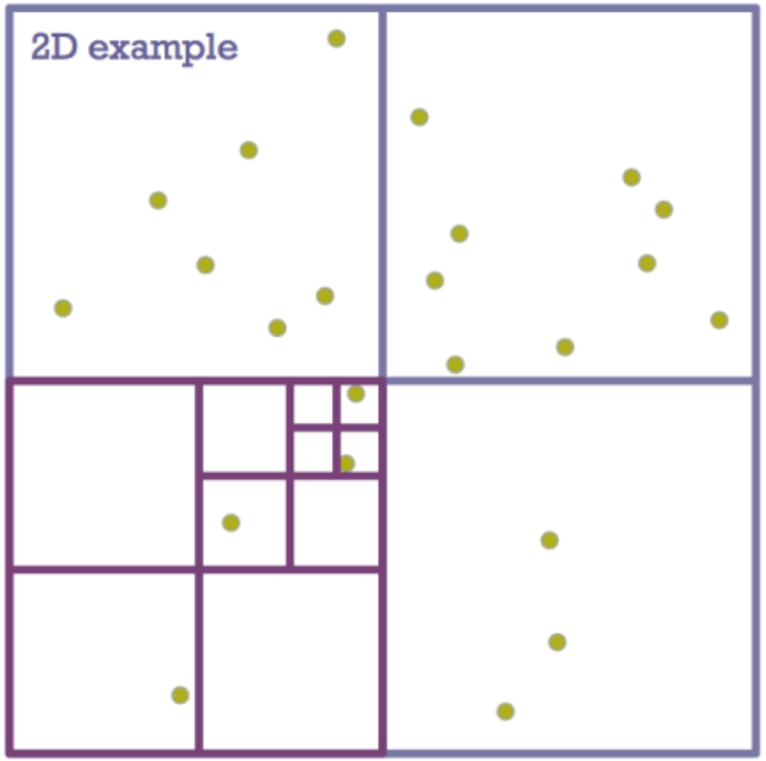
PARALLEL SOLUTION

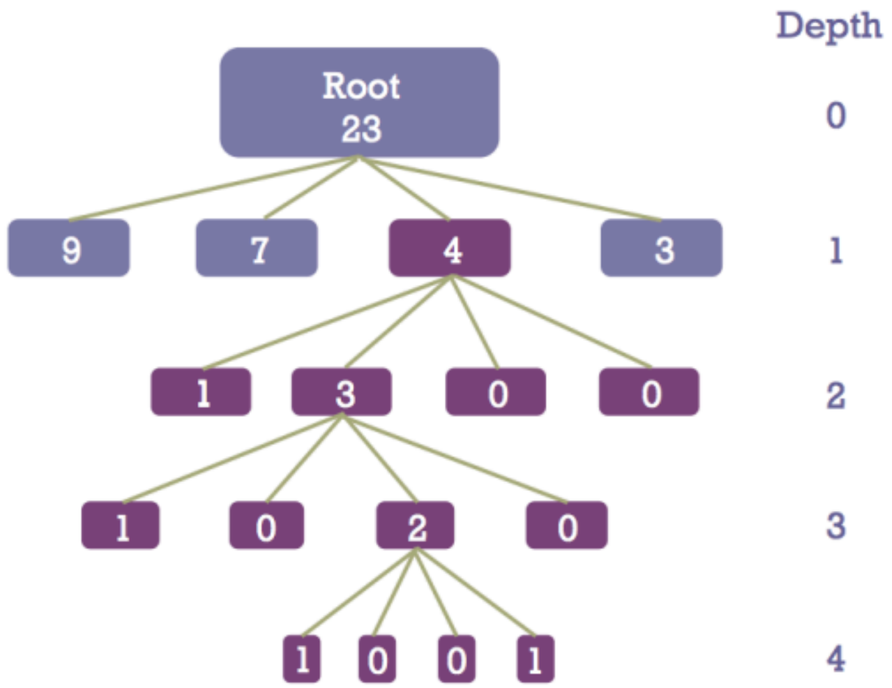
PARALLEL BARNES HUT

```cpp
__device__ void calculate_force_device(Particle *p,Tree* root)
{
    double ax=0,ay=0,az=0;

    calculate_acceleration(p, root, ax, ay, az);

    p->vx += ax * dt;
    p->vy += ay * dt;
    p->vz += az * dt;

    double force = sqrtf(ax*ax + ay*ay + az*az) * p->mass;
    p->force = force;
}

__global__ void calculate_force_kernel(Particle* particles, Tree* root)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if(idx<n)
    {
        calculate_force_device(&particles[idx], root);
    }
```

```cpp
__global__ void update_positions_kernel(Particle* particles)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(i<n)
    {
        particles[i].update_position();
    }
}

void update_positions(Particle* particles)
{
    int blockSize = 256;
    cudaOccupancyMaxPotentialBlockSize(NULL, &blockSize,update_positions_kernel,0,n);

    int numBlocks = (n + blockSize - 1) / blockSize;
    update_positions_kernel<<<numBlocks, blockSize>>>(particles);
    cudaDeviceSynchronize();
```

BOTTLENECK



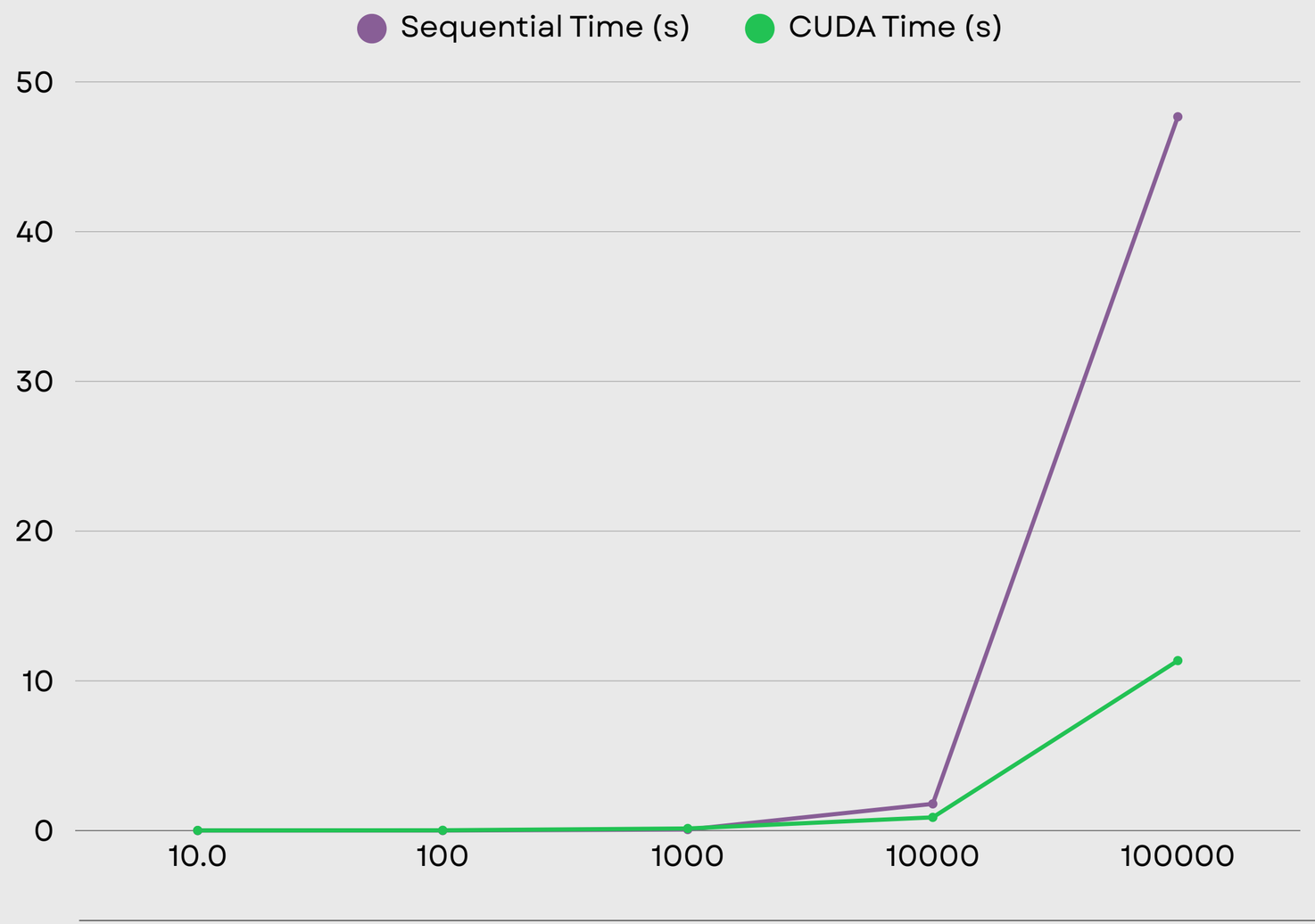


TIME COMPLEXITY

WORK:O(NLOGN)

SPAN:O(LOGN)

TIME USAGE


Sequential Time (s)
CUDA Time (s)
50
40
30
20
10
0
10.0
100
1000
10000
100000

THANK YOU