

CS601: Software Development for Scientific Computing

Autumn 2023


Week15 : Particle Methods (N-Body
Problems), Misc Topics

Matrix Algebra and Efficient Computation

- Pic source: the Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)

<i>Motif</i>	Embed	Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser	<i>Motif</i>					Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser
1 Finite State Mach.	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	9 N-Body	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot
2 Combinational	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	10 MapReduce	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot
3 Graph Traversal	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	11 Backtrack/B&B	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot
4 Structured Grid	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	12 Graphical Models	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot
5 Dense Matrix	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	13 Unstructured Grid	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot
6 Sparse Matrix	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	<i>Temperature Chart of Need</i>					DB = database									
7 Spectral (FFT)	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Warm	Med	Cool	Hot	ML = machine learning									
8 Dynamic Prog	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	HPC = High Perf. Comp.									

Figure 4. Temperature Chart of the 13 Motifs. It shows their importance to each of the original six application areas and then how important each one is to the five compelling applications of Section 3.1. More details on the motifs can be found in (Asanovic, Bodik et al. 2006).

⇒ Seen earlier
 Next..

Particle (Simulation) Methods

- N-Body Simulation – Problem

System of N-bodies (e.g. galaxies, stars, atoms, light rays etc.) interacting with each other continuously


- Problem:

- Compute force acting on a body due to all other bodies in the system
 - Determine position, velocity, at various times for each body

- Objective:

- Determine the (approximate) evolution of a system of bodies interacting with each other simultaneously

Particle (Simulation) Methods

- N-Body Simulation - Examples
 - Astrophysical simulation: E.g. each body is a star/galaxy
https://commons.wikimedia.org/w/index.php?title=File%3AGalaxy_collision.ogv
 - Graphics: E.g. each body is a ray of light emanating from the light source.
<https://www.fxguide.com/fxfeatured/brave-new-hair/>

 - Here each body is a point on a strand of hair

N-Body Simulation

- All-pairs Method
 - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
 - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
 - Examples:
 - Barnes-Hut
 - Fast Multipole Method

N-Body Simulation


- Three fundamental simulation approaches
 - Particle-Particle (PP)
 - Particle-Mesh (PM)
 - Particle-Particle-Particle-Mesh (P3M)
- Hybrid approaches
 - Nested Grid Particle Scheme
 - Tree Codes
 - Tree Code Particle Mesh (TPM)
- Self Consistent Field (SCF), Smoothed-Particle Hydrodynamics (SPH), Symplectic etc.

Particle-Particle method

- Simplest. Adopts an all-pairs approach.
- State of the system at time t given by particle positions $x_i(t)$ and velocity $v_i(t)$ for $i=1$ to N

$$\{x_i(t), v_i(t); i = 1, N\}$$

– Steps:

- 
1. Compute forces
 2. Integrate equations of motion
 3. Update time counter
- Each iteration updates $x_i(t)$ and $v_i(t)$ to compute $x_i(t + \Delta t)$ and $v_i(t + \Delta t)$

Particle-Particle Method

1. Compute forces

```
//initialize forces
```

```
for i=1 to N
```

```
   $F_i = 0$ 
```

```
//Accumulate forces
```

```
for i=1 to N-1
```

```
  for j=i+1 to N
```

```
     $F_i = F_i + F_{ij}$  ←  $F_{ij}$  is the force on particle i due to particle j
```

```
     $F_j = F_j - F_{ij}$ 
```

Typically: $F_i = F_{\text{external}} + F_{\text{nearest_neighbor}} + F_{\text{N-Body}}$

Particle-Particle Method

2. Integrate equations of motion

for $i=1$ to N

$$v_i^{new} = v_i^{old} + \frac{F_i}{m_i} \Delta t \quad // \text{using } a=F/m \text{ and } v=u+at$$

$$x_i^{new} = x_i^{old} + v_i \Delta t$$

3. Update time counter

$$t^{new} = t^{old} + \Delta t$$

Particle-Particle Method

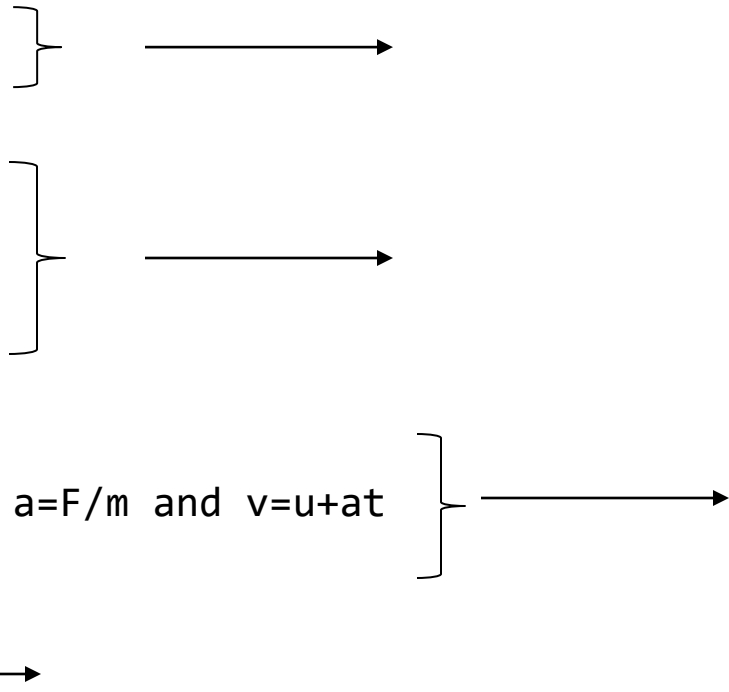
```
t=0
while(t<tfinal) {
  //initialize forces
  for i=1 to N
    Fi = 0
  //Accumulate forces
  for i=1 to N-1
    for j=i+1 to N
      F[i] = F[i] + Fij
      F[j] = F[j] - Fij
  //Integrate equations of motion
  for i=1 to N
    
$$v_i^{new} = v_i^{old} + \frac{F_i}{m_i} \Delta t$$
 //using a=F/m and v=u+at
    
$$x_i^{new} = x_i^{old} + v_i \Delta t$$

  // Update time counter
  t = t + Δt
}
```

Particle-Particle Method

- Costs (CPU operations)?

```
t=0
while(t<tfinal) {
  //initialize forces
  for i=1 to N
    Fi = 0
  //Accumulate forces
  for i=1 to N-1
    for j=i+1 to N
      F[i] = F[i] + Fij
      F[j] = F[j] - Fij
  //Integrate equations of motion
  for i=1 to N
    vinew = viold +  $\frac{F_i}{m_i} \Delta t$  //using a=F/m and v=u+at
    xinew = xiold + vi  $\Delta t$ 
  // Update time counter
  t = t +  $\Delta t$ 
}
```



The diagram illustrates the sequence of operations in the Particle-Particle Method. It consists of three main blocks, each marked with a curly brace on the right side and an arrow pointing to the right, indicating a flow of operations:

- Block 1:** Initialization of forces. The code is `for i=1 to N` followed by `Fi = 0`.
- Block 2:** Accumulation of forces. The code is a nested loop: `for i=1 to N-1` followed by `for j=i+1 to N`, with the calculations `F[i] = F[i] + Fij` and `F[j] = F[j] - Fij` inside.
- Block 3:** Integration of equations of motion. The code is `for i=1 to N` followed by the calculations `vinew = viold + $\frac{F_i}{m_i} \Delta t$ and xinew = xiold + vi Δt . A comment //using a=F/m and v=u+at is present.`

Finally, the time counter is updated: `// Update time counter` followed by `t = t + Δt` .

Particle-Particle Method

- Experimental results (then):
 - Intel Delta = 1992 supercomputer, 512 Intel i860s
 - 17 million particles, 600 time steps, 24 hours elapsed time

M. Warren and J. Salmon

Gordon Bell Prize at Supercomputing 1992

 - Sustained 5.2 GigaFlops = 44K Flops/particle/time step
 - 1% accuracy
 - *Direct method (17 Flops/particle/time step) at 5.2 Gflops would have taken 18 years, 6570 times longer*

Particle-Particle Method

- Experimental results (now):

Vortex particle simulation of turbulence

- Cluster of 256 NVIDIA GeForce 8800 GPUs
- 16.8 million particles
 - T. Hamada, R. Yokota, K. Nitadori, T. Narumi, K. Yasuki et al
 - *Gordon Bell Prize for Price/Performance at Supercomputing 2009*
- Sustained 20 Teraflops, or \$8/Gigaflop

Particle-Particle (PP) Method

- Discussion
 - Simple/trivial to program
 - High computational cost
 - Useful when number of particles are small (few thousands) and
 - We are interested in close-range dynamics when the particles in the range contribute *significantly* to forces
 - Constant time step must be replaced with variable time steps and numerical integration schemes for close-range interactions

N-Body Simulation

- All-pairs Method
 - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
 - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
 - Examples:
 - Barnes-Hut
 - Fast Multipole Method

Tree Codes

$$F_i = F_{\text{external}} + F_{\text{nearest_neighbor}} + F_{\text{N-Body}}$$

- F_{external} can be computed for each body independently. $O(N)$
- $F_{\text{nearest_neighbor}}$ involve computations corresponding to few nearest neighbors. $O(N)$
- $F_{\text{N-Body}}$ require all-to-all computations. Most expensive. $O(N^2)$ if computed using all-pairs approach.

for(i = 1 to N)

$$F_i = \sum_{j \neq i} F_{ij} \quad F_{ij} = \text{force on } i \text{ from } j$$

$$F_{ij} = c \cdot v / \|v\|^3 \text{ in 3D, } F_{ij} = c \cdot v / \|v\|^2 \text{ in 2D}$$

v = vector from particle i to particle j , $\|v\|$ = length of v , c = product of masses or charges