

Lecture 11

Floating Point Arithmetic Particle Methods

Announcements

- A3 deadline delayed 24 hours (Thu)

Today's lecture

- Floating Point
- Particle Methods

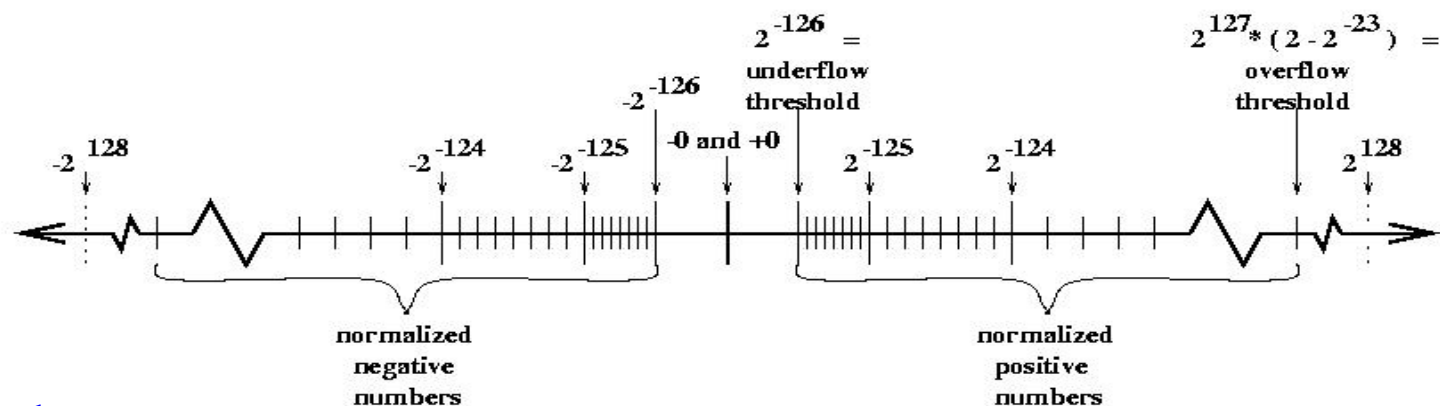
What is floating point?

- A representation
 - ♦ $\pm 2.5732... \times 10^{22}$
 - ♦ NaN ∞
 - ♦ Single, double, extended precision
- A set of operations
 - ♦ $+ = * / \sqrt{\text{rem}}$
 - ♦ Comparison $< \leq = \neq \geq >$
 - ♦ Conversions between different formats, binary to decimal
 - ♦ Exception handling
- Language and library support
- IEEE Floating point standard P754
 - ♦ Universally accepted
 - ♦ W. Kahan received the Turing Award in 1989 for design of IEEE Floating Point Standard
 - ♦ Revision in 2008

IEEE Floating point standard P754

- Normalized representation $\pm 1.d\dots d \times 2^{\text{exp}}$
 - Macheps** = Machine epsilon = $\epsilon = 2^{-\#\text{significant bits}}$
relative error in each operation
 - OV** = overflow threshold = largest number
 - UN** = underflow threshold = smallest number
- $\pm\text{Zero}$: $\pm\text{significant}$ and $\text{exponent} = 0$

Format	# bits	#significant bits	macheps	#exponent bits	exponent range
Single	32	23+1	2^{-24} ($\sim 10^{-7}$)	8	$2^{-126} - 2^{127}$ ($\sim 10^{+-38}$)
Double	64	52+1	2^{-53} ($\sim 10^{-16}$)	11	$2^{-1022} - 2^{1023}$ ($\sim 10^{+-308}$)
Double	≥ 80	≥ 64	$\leq 2^{-64}$ ($\sim 10^{-19}$)	≥ 15	$2^{-16382} - 2^{16383}$ ($\sim 10^{+-4932}$)



What happens in a floating point operation?

- Round to the nearest representable floating point number that corresponds to the exact value(correct rounding)
- When there are ties, round to the nearest value with the lowest order bit =0 (rounding toward nearest even)
- Others are possible
- We don't need the exact value to work this out!
- Applies to $+$ $=$ $*$ $/$ $\sqrt{}$ rem and to format conversion
- Error formula: $\text{fl}(a \text{ op } b) = (a \text{ op } b)(1 + \delta)$ where
 - op one of $+$, $-$, $*$, $/$
 - $|\delta| \leq \epsilon = \text{machine epsilon}$
 - assumes no overflow, underflow, or divide by zero
- Addition example
 - ♦ $\text{fl}(\sum x_i) = \sum_{i=1:n} x_i(1+e_i)$
 - ♦ $|e_i| \sim (n-1)\epsilon$

Addition example

$$\begin{aligned}\text{fl}(x_1+x_2+x_3+x_4) &= \{[(x_1+x_2)*(1+\delta_1) + x_3]*(1+\delta_2) + x_4\}*(1+\delta_3) \\ &= x_1*(1+\delta_1)*(1+\delta_2)*(1+\delta_3) + \\ &\quad x_2*(1+\delta_1)*(1+\delta_2)*(1+\delta_3) + \\ &\quad x_3*(1+\delta_2)*(1+\delta_3) + x_4*(1+\delta_3) \\ &\equiv x_1*(1+e_1) + x_2*(1+e_2) + x_3*(1+e_3) + x_4*(1+e_4) \\ &\quad \text{where } |e_i| \lesssim 3*\text{macheps}\end{aligned}$$

Exceptions

- An exception occurs when the result of a floating point operation is not a real number, or too extreme to represent accurately? We get an *exception*
 $1/0, \sqrt{-1}, 1.0e-30 + 1.0e+30$
- P754 standardizes how we handle exceptions
 - ♦ **Overflow**: - exact result $> OV$, too large to represent, returns $\pm\infty$
 - ♦ **Underflow**: exact result nonzero and $< UN$, too small to represent
 - ♦ **Divide-by-zero**: nonzero/0, returns $\pm\infty = \pm 0$ (Signed zeroes)
 - ♦ **Invalid**: $0/0, \sqrt{-1}, \log(0)$, etc.
 - ♦ **Inexact**: there was a rounding error (common)
- Two possible responses
 - ♦ Stop the program, given an error message
 - ♦ Continue; tolerate the exception, possibly repairing the error

An example

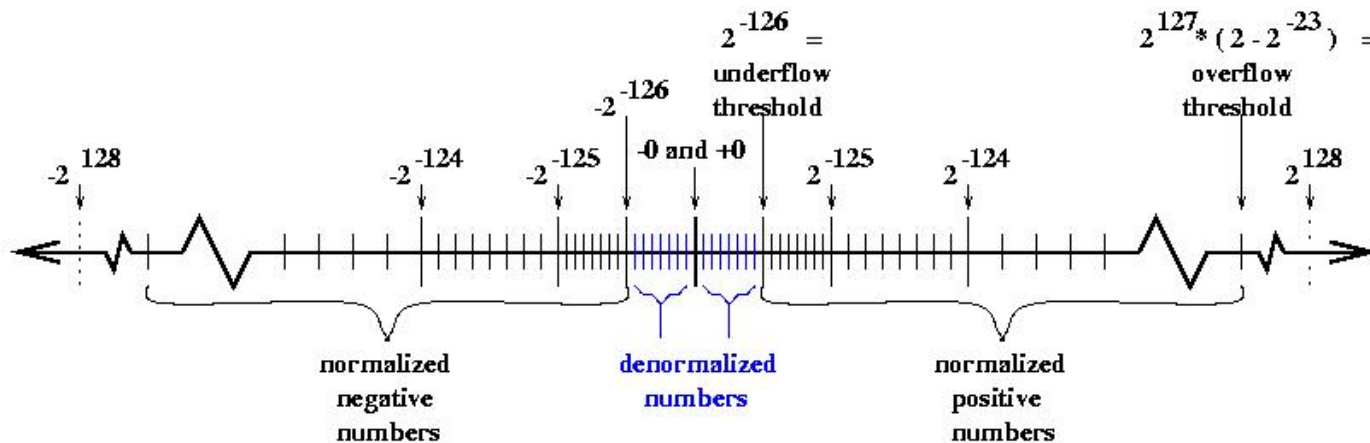
- Graph the function

$$f(x) = \sin(x) / x$$

- $f(0) = 1$
- But we get a singularity @ $x=0$: $1/x = \infty$
- This is an “accident” in how we represent the function (W. Kahan)
- We *catch* the exception (divide by 0)
- Substitute the value $f(0) = 1$

Denormalized numbers

- We compute if $(a \neq b)$ then $x = a/(a-b)$
- We should never divide by 0, even if $a-b$ is tiny
- *Underflow* exception occurs when exact result $a-b < \text{underflow threshold UN}$
- We return a *denormalized number* for $a-b$
 - ♦ Relax restriction that leading digit is 1: $\pm 0.d\dots d \times 2^{\text{min_exp}}$
 - ♦ Fills in the gap between 0 and UN uniform distribution of values
 - ♦ Ensures that we never divide by 0



Jim

Why is it important to handle exceptions properly?

- Crash of Air France flight #447 in the mid-atlantic
<http://www.cs.berkeley.edu/~wkahan/15June12.pdf>
- Flight #447 encountered a violent thunderstorm at 35000 feet and super-cooled moisture clogged the probes measuring airspeed
- The autopilot couldn't handle the situation and relinquished control to the pilots
- It displayed the message “Invalid Data” without explaining why
- Without knowing what was going wrong, the pilots were unable to correct the situation in time
- The aircraft stalled, crashing into the ocean 3 minutes later
- At 20,000 feet, the ice melted on the probes, but the pilots didn't know this so couldn't know which instruments to trust or distrust.

Infinites

- $\pm\infty$
- **Overflow exception**
 - ♦ Exact finite result is so large, cannot represent accurately
 - ♦ Example: $2*OV$
- **Divide by zero exception**
 - ♦ Return $\pm\infty = 1/\pm 0$
- Infinites extend the range of mathematical operators
 - ♦ $5 + \infty, 10*\infty \quad \infty*\infty$
 - ♦ No exception: the result is exact

NaN (Not a Number)

- Invalid exception

- ♦ Exact result is not a well-defined real number

$0/0$ $\sqrt{-1}$ $\infty - \infty$ NaN-10 NaN<2?

- We can have a quiet NaN or a signalling Nan

- ♦ Quiet –does not raise an exception, but propagates a distinguished value
 - E.g. missing data: $\max(3, \text{NaN}) = 3$
- ♦ Signaling - generate an exception when accessed
 - Detect uninitialized data

Exception handling - interface

- Each of the 5 exceptions manipulates 2 flags
- Sticky flag set by an exception
 - ♦ Remains set until explicitly cleared by the user
 - ♦ May be tested by the programmer
- Exception flag: should a trap occur?
 - ♦ By default we don't trap, continue computing
NaN ∞ denorm
 - ♦ If we do, enter a trap handler. accesses arguments in operation that caused the exception
 - ♦ Requires precise interrupts, causes problems on a parallel computer, usually not implemented
- We can use exception handling to build faster algorithms
 - ♦ Try the faster but “riskier” algorithm (but denorm can be slow)
 - ♦ Rapidly test for accuracy (use exception handling)
 - ♦ Substitute slower more stable algorithm as needed
 - ♦ See Demmel&Li: crd.lbl.gov/~xiaoye

Summary of representable numbers

- ± 0

$\pm$	0...0	0.....0
-------	-------	---------

- $\pm \infty$

$\pm$	1....1	0.....0
-------	--------	---------

- Normalized nonzeros

$\pm$	Not 0 or all 1s	anything
-------	--------------------	----------

- Denormalized numbers

$\pm$	0...0	nonzero
-------	-------	---------

- NaNs

$\pm$	0...0	nonzero
-------	-------	---------

- ♦ Signaling and quiet
 - ♦ Often supported as quiet NaNs only

Mixed precision

- Single precision is usually faster than double
 - ♦ Sometimes extreme: 1.3 device capability NVIDIA GPUs (x8), STI Cell (x10)
 - ♦ Shorter logic delays, less data motion
- Idea: use single precision to get an approximate answer, then finish off with double
 - ♦ We can get the same result at the speed of single
 - ♦ Or we can get a more accurate answer in the same time
- Example: solving linear systems of equations
 $Ax=b$
 - ♦ Baboulin, Marc, et al. “Accelerating scientific computations with mixed precision algorithms.”
Computer Physics Communications 180(12): 2526-2533, 2009

Anomalous behavior

- Floating point arithmetic is not associative

$$(x + y) + z \neq x + (y + z)$$

- When we change the number of processors, we can get a different answer, say in reducing an array down to a scalar $\text{sumall}(x_i)$
 - ♦ Race conditions, or reordering of arithmetic
 - ♦ Can even depend on the compile

- Distributive law doesn't always hold
- These expressions have different values when $y \approx z$

$$x * y - x * z \neq x * (y - z)$$

- Optimizers can't reason about floating point
- If we compute a quantity in extended precision (80 bits) we lose digits when we store to memory $y \neq x$

```
float x, y=..., z=...;  
x = y + z;  
y=x;
```

When compiler optimizations alter precision

- Let's say we support 79+ bit extended format in registers
- When we store values into memory, values truncated to the lower precision format, e.g. 64 bits
- Compilers can keep things in registers and we may lose *referential transparency*

- An example

```
float x, y, z;  
int j;  
....  
  x = y + z;  
  if (x >= j) replace x by something smaller than j  
  y=x;
```

- With optimization turned on, x is computed to extra precision; it is not a float
- If $x < j$ and held in a register....
no guarantee the condition $x < j$ will be preserved ...
when x is stored in y , i.e. $y \geq j$

P754 on the GPU

- Cuda Programming Guide (4.1)
“All compute devices follow the IEEE 754-2008 standard for binary floating-point arithmetic with the following deviations”
 - ♦ There is no mechanism for detecting that a floating-point exception has occurred and all operations behave as if the exceptions are always masked... SNaN ... are handled as quiet
- Cap. 2.x: FFMA ... is an IEEE-754-2008 compliant fused multiply-add instruction ... the full-width product ... used in the addition & a single rounding occurs during generation of the final result
 - ♦ $\text{rnd}(A \times A + B)$ with FFMA (2.x) vs $\text{rnd}(\text{rnd}(A \times A) + B)$ FMAD for 1.x
- FFMA can avoid loss of precision during subtractive cancellation when adding quantities of similar magnitude but opposite signs
- Also see *Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs,* by N. Whitehead and A. Fit-Florea

To read further

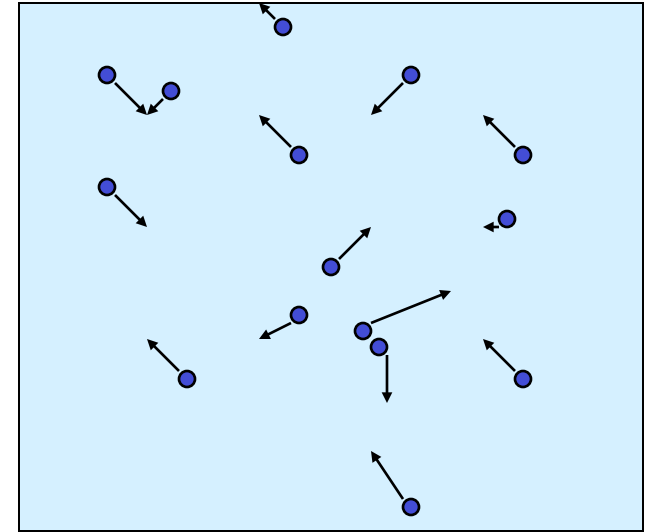
- W. Kahan' s lecture from 1996
www.cs.berkeley.edu/~wkahan/ieee754status/cs267fp.ps
- W. Kahan' s “Lecture Notes on IEEE 754”
www.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps
- W. Kahan' s “The Baleful Effects of Computer Benchmarks on Applied Math, Physics and Chemistry”
www.cs.berkeley.edu/~wkahan/ieee754status/baleful.ps

Today's lecture

- Floating Point
- Particle Methods

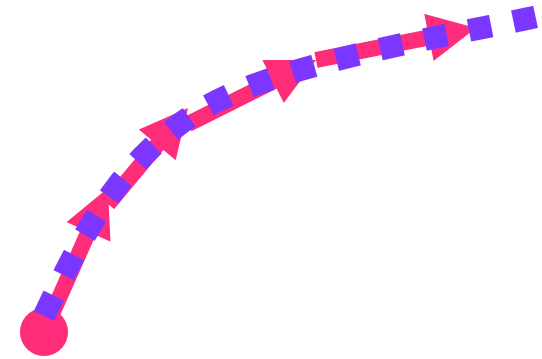
The N-body problem

- Compute trajectories of a system of N bodies often called *particles*, moving under mutual influence
 - ♦ The Greek word for particle: *somati'dion* = “little body”
 - ♦ No general analytic (exact) solution when $N > 2$
 - ♦ Numerical simulations required
 - ♦ N can range from thousands to millions
- A **force law** governs the way the particles interact
 - ♦ We may not need to perform all $O(N^2)$ force computations
 - ♦ Introduces non-uniformity due to uneven distributions



Discretization

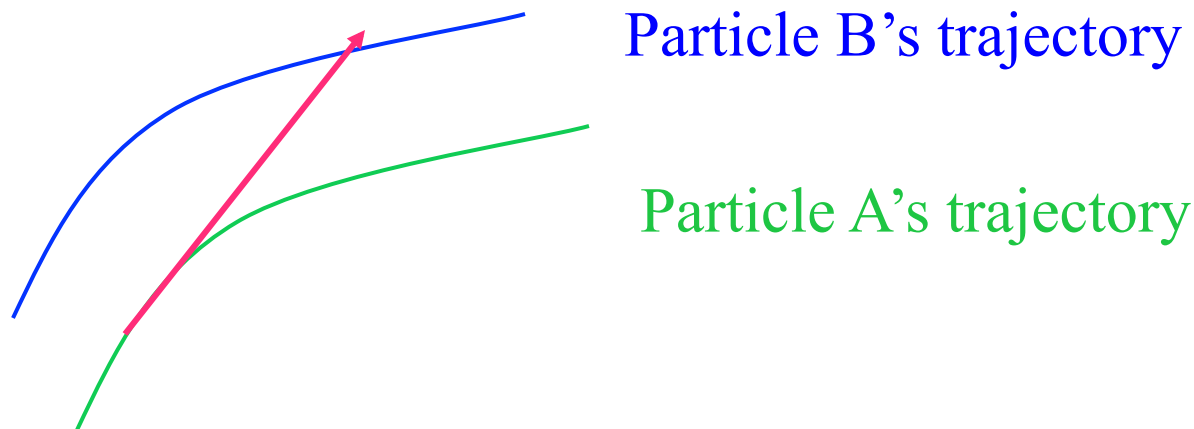
- Particles move continuously through space and time according to a force, a continuous function of position and time: $F(\mathbf{x}, t)$
- We approximate continuous values using a discrete representation
- Evaluate force field at discrete points in time, called *timesteps* Δt , $2\Delta t$, $3\Delta t$
 $\Delta t = \text{discrete time step}$ (a parameter)
- “Push” the bodies according to Newton’s third law: $\mathbf{F} = m\mathbf{a} = m \, d\mathbf{u}/dt$
- There is no self induced force



```
while (current time < end time)
  forall bodies  $i \in 1:N$ 
    compute force  $F_i$  induced by all
      bodies  $j \in 1:N$ 
  //  $F = \text{mass} * \text{Acceleration}$ 
  forall bodies  $i \in 1:N$ 
    update velocity  $v_i$  by  $a_i \Delta t$ 
    update position  $x_i$  by  $v_i \Delta t$ 
  current time +=  $\Delta t$ 
end
```

Timestep selection

- We approximate the velocity of a particle by the tangent to the particle's trajectory
- Since we compute velocities at discrete points in space and time, we approximate the true trajectory by a straight line
- So long as Δt is small enough, the resultant error is reasonable
- If not then we might “jump” to another trajectory: this is an error



Some details

- There is no self induced force
- How should we choose the timestep Δt ?
- We apply numerical criteria to select Δt , based on an analysis of the evolving solution
- Selection may be ongoing, i.e. dynamic

Computing the force

- The running time of the computation is dominated by the force computation, so we ignore the push phase
- The simplest approach is to use the direct method, with a running time of $O(N^2)$

$$\text{Force on particle } i = \sum_{j=0:N-1} F(x_i, x_j)$$

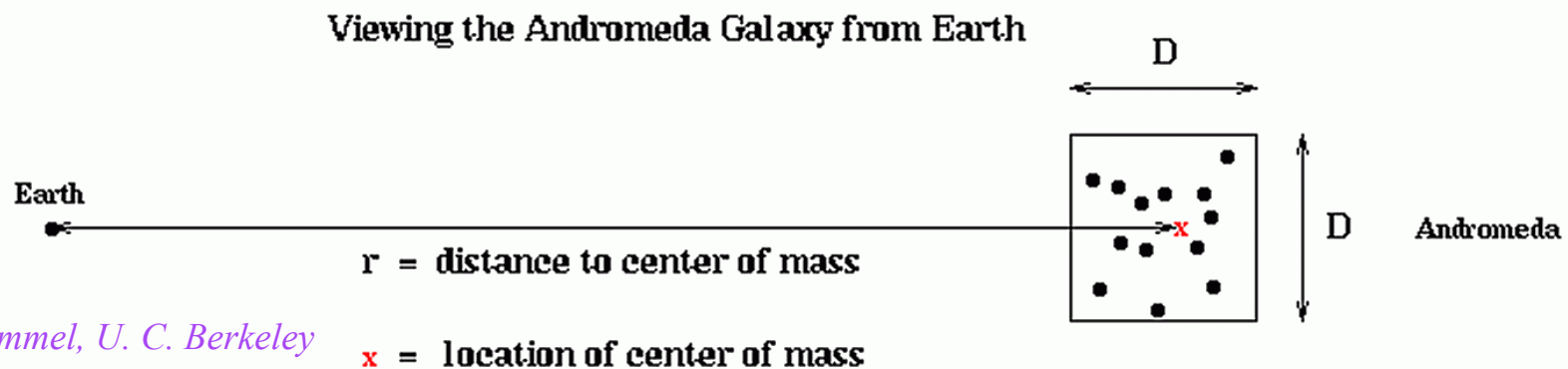
- $F()$ is the **force law**
- One example is the **gravitational force law**

$$G m_i m_j / r_{ij}^2 \text{ where } r_{ij} = \text{dist}(x_i, x_j)$$

G is the **gravitational constant**

Significance of locality

- Many scientific problems exhibit both **spatial** and **temporal locality**, due to the underlying physics
 - ♦ The values of the solution at nearby points in space and time are closer than for values at distant points
 - ♦ Gravitational interactions decay as $1/r^2$
 - ♦ Waves live in a localized region of space at any one time and will appear in a nearby region of space in a nearby point in time
- Numerically speaking, timesteps are “small:” the solution changes gradually
- Let’s consider a more highly localized force



Jim Demmel, U. C. Berkeley

A localized force law

- As simple repulsive force
- No induced force beyond a *cutoff distance* δ

if $\text{dist}(\mathbf{x}_i, \mathbf{x}_j) > \delta \Rightarrow F(\mathbf{x}, \mathbf{y}) = 0$

else

$$F(\mathbf{x}, \mathbf{y}) = C * (\mathbf{dx}, \mathbf{dy})$$

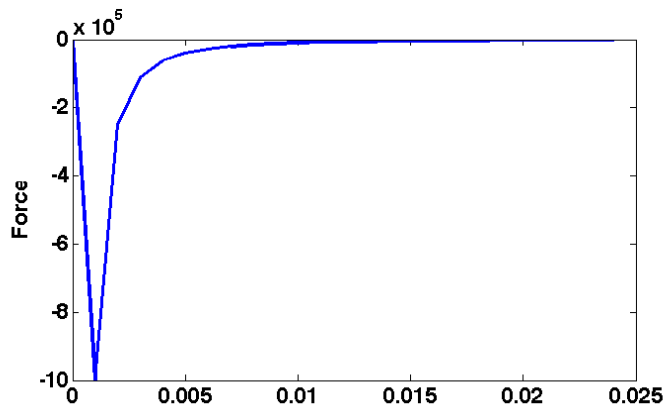
Where

$$C = (0.01/r^2 - 1/r^3)$$

$$r^2 = \max(\mathbf{dx}^2 + \mathbf{dy}^2, 10^{-6})$$

$$(\mathbf{dx}, \mathbf{dy}) = ((\mathbf{x}_j - \mathbf{x}_i), (\mathbf{y}_j - \mathbf{y}_i))$$

$$\delta = 0.01$$



Overall simulation

```
void SimulateParticles() {  
    for( int step = 0; step < nsteps; step++ ) {  
        apply_forces(particles,n);  
  
        move_particles(particles,n);  
  
        VelNorms(particles,n,uMax,vMax,uL2,vL2);  
    }  
}
```

Reducing the cost

- We have to compute the distance check between each pair of particles, even if we don't compute the force
- But many particles lay beyond the cutoff

if $\text{dist}(\mathbf{x}_i, \mathbf{x}_j) > \delta \Rightarrow F(\mathbf{x}, \mathbf{y}) = 0$

else

$$F(\mathbf{x}, \mathbf{y}) = C * (\mathbf{dx}, \mathbf{dy})$$

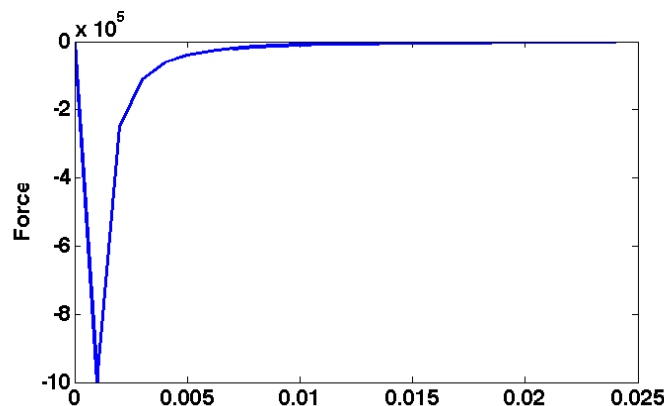
Where

$$C = (0.01/r^2 - 1/r^3)$$

$$r^2 = \max(\mathbf{dx}^2 + \mathbf{dy}^2, 10^{-6})$$

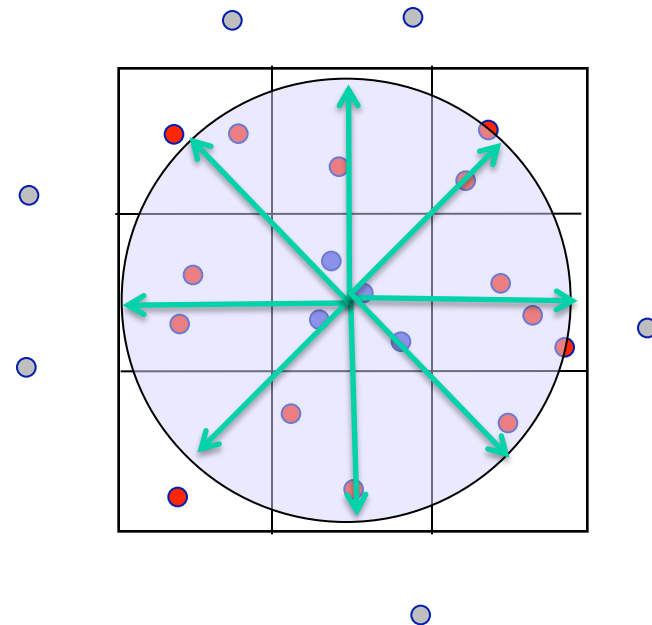
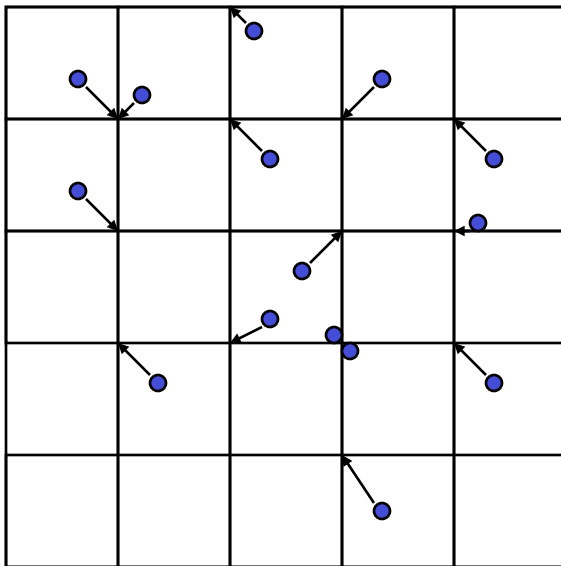
$$(\mathbf{dx}, \mathbf{dy}) = ((\mathbf{x}_j - \mathbf{x}_i), (\mathbf{y}_j - \mathbf{y}_i))$$

$$\delta = 0.01$$



Locality optimization

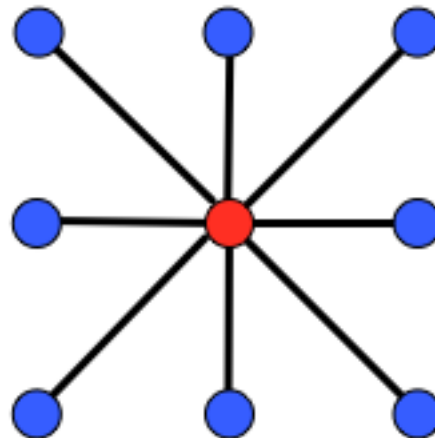
- We don't need to compute all the distance tests
- To speed up the search for nearby particles, sort into a *chaining mesh* (Hockney & Eastwood, 1981)
- Compute forces one box at a time, 8 surrounding cells only
- Still runs in time $O(N^2)$, but have reduced the constant



Jim Demmel, U. C. Berkeley

Reducing the cost of the force computation

```
for( int i = 0; i < nx; i++ )  
  for( int j = 0; j < ny; j++ )  
    for (int i0 = -1; i0 < 2; i0++ )  
      for (int j0 = -1; j0 < 2; j0++ )  
        Update forces on Box[i,j] from particles in Box [i,j]
```



Your assignment

- Implement the particle method with MPI
- Next time: MPI

