Lecture 8

Advanced Scheduling & Applications

Announcements

- Use ieng6-204 for performance tuning and benchmarking
- Use ieng6-203 for development
- We will set aside blocks of "development time" where there will be no long running jobs on the machine
  - Take the surveymonkey

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Today's lecture

- A few words on code design
- Advanced Scheduling
- Load Balancing

Parallel control flow

- High overhead associated with spawning and joining threads

```
for (it= 0; it<nIters; it++) {
#pragma omp parallel shared( … ) private(i,j)
#pragma omp for
  {
    for (int i=1; i<=nx; i++)
      for (int j=1; j<=ny; j++)
        Un[i,j] = (U[i-1, j] + U[i+1,j] + … ) /4;

    U ←→Un;
  }
}
```

Parallel control flow

- Lower overhead if we put the parallel section outside the iteration loop

```
#pragma omp parallel shared( … ) private(i,j)
for (it= 0; it<nIters; it++) {
#pragma omp for
  {
    for (int i=1; i<=nx; i++)
      for (int j=1; j<=ny; j++)
        Un[i,j] = (U[i-1, j] + U[i+1,j] + … ) /4;

    U ↔Un;
  }
}
```

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Advanced Scheduling in OpenMP

Workload Decomposition

- How to split up the work over processors?
  - Partitioning
  - Processor mapping
  - Dynamic or static?
  - Uniform or non-uniform
- Data parallelism
- Task parallelism

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Data parallelism: iteration mapping

- By default, OpenMP maps iterations to threads with a static uniform mapping
- Split N iterations into NT pieces as evenly as possible
  Thread K gets iterations K*N/NT .. (K+1)*N/NT-1
- What if the iterations have different running times?

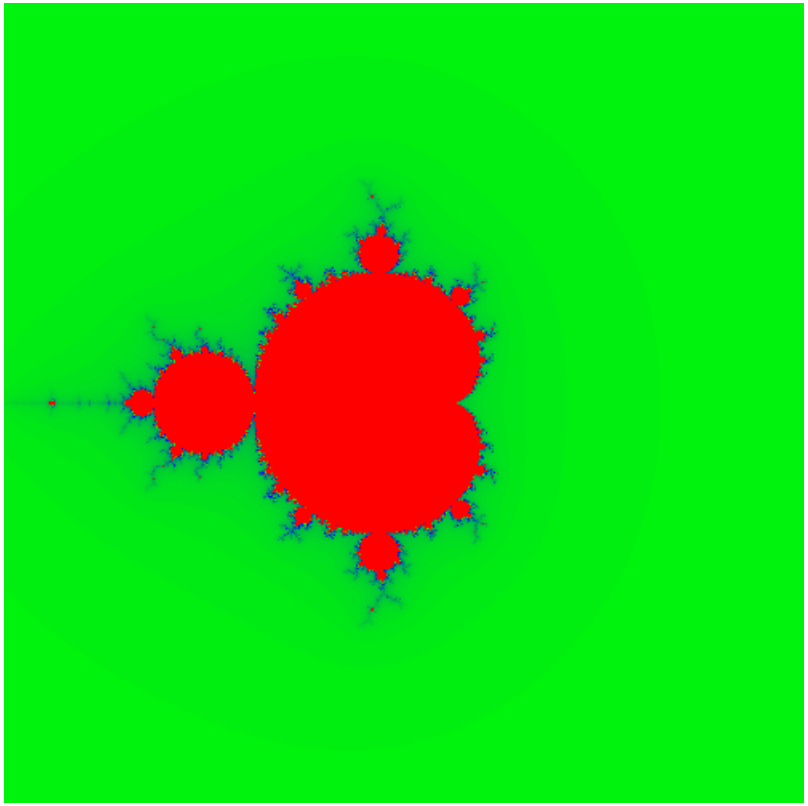
```
for (int i=0; i<n)
    work(i);
```

- Consider the Mandelbrot set computation

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Motivating application

- Mandelbrot set computation
- Named after B. Mandelbrot



© 2010 Scott B. Baden / CSE 160 / Winter 2011

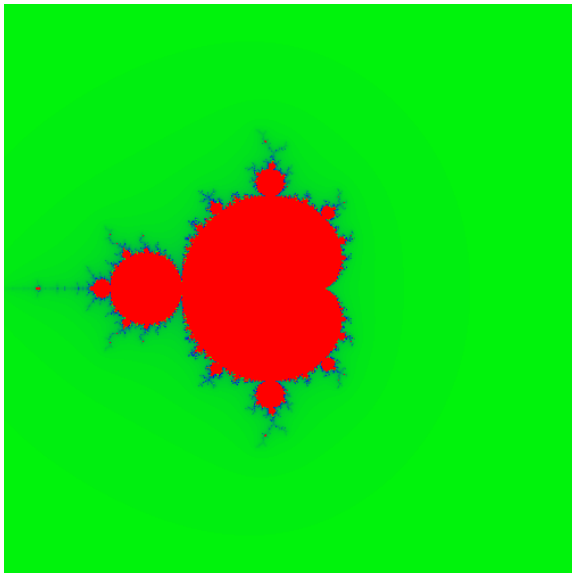
The Mandelbrot set

- For which points *c* in the complex plane does the following iteration remain bounded?

$$z_{k+1} = z_k^2 + c$$

c is a complex number, $z_0 = 0$

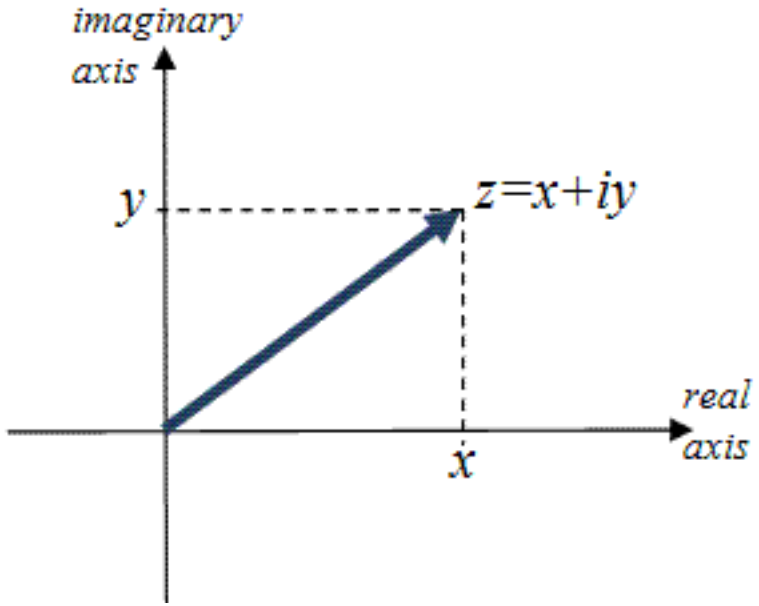
- When c=0, all points lay within a unit disk : $|z| \leq 1$
- If $|z| \geq 2$, the iteration is guaranteed to diverge to ∞
- Plot the rate at which points in a given region diverge
- Stop the iterations when $z_{k+1} \geq 2$ or *k* reaches some limit
- Plot *k* at each position



© 2010 Scott B. Baden / CSE 160 / Winter 2011

A quick review of complex numbers

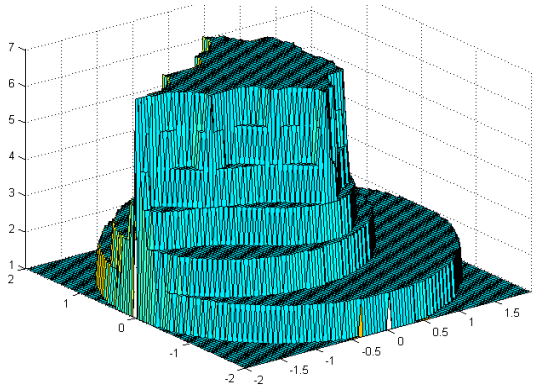
- Define **i** = √ -1
- *Complex number* z = x + **i** y
  - x is called the *real part*
  - y is called the *imaginary part*
- We associate each complex number with a point in the *x-y* plane
- Magnitude of a complex number same as vector length: $|z| = \sqrt{(x^2 + y^2)}$

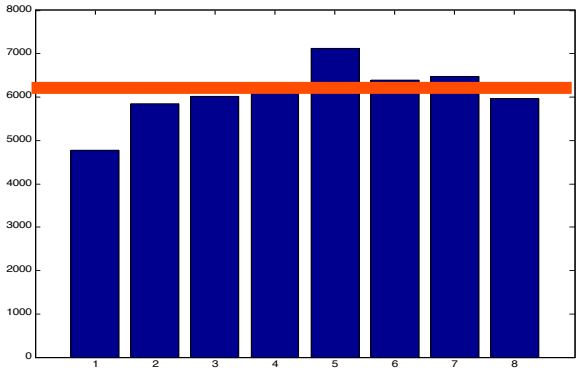


© 2010 Scott B. Baden / CSE 160 / Winter 2011

A load balancing problem

- Some points iterate longer than others
- Uniform decomposition: uneven workload distribution
- We have a ***load imbalance***





```
#pragma omp parallel for
for i = 0 to n-1
    for j = 0 to n-1
    z0 =Complex (x[i],y[i])
    while (| zk|< 2 or k >MAXITER)
            zk++  += zk2 + c
    Ouput(i,j) = k
```

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Load balancing efficiency

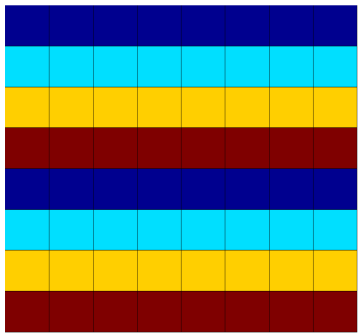
- If we ignore serial sections and other overheads, then we may express load imbalance in terms of a
  load balancing efficiency metric
- Let each processor *i* complete its assigned work in time T_i
- Thus, the running time $T_P = \text{MAX}(T_i)$
- Define $\overline{T} = \sum_i T_i$
- We define the **load balancing efficiency** $\eta = \frac{\overline{T}}{PT_P}$
- Ideally $\eta = 1.0$

© 2010 Scott B. Baden / CSE 160 / Winter 2011

First approach: chunking

- Divide into rows in bundles of CHUNK consecutive rows
- Processor k gets chunks 0, NT, 2*NT, …
- Also called *round robin* or *block cyclic*

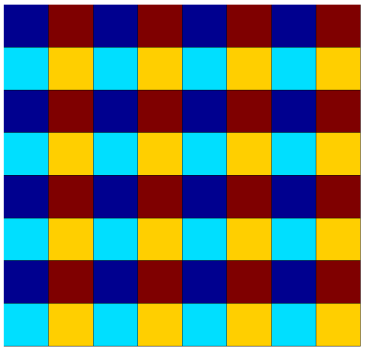
```
#pragma omp parallel for schedule(static, CHUNK)
for i = 0 to n-1
    for j = 0 to n-1
    z0 =Complex (x[i],y[i])
    while (| zk|< 2 or k >MAXITER)
            zk++  += zk2 + c
    Ouput(i,j) = k
```

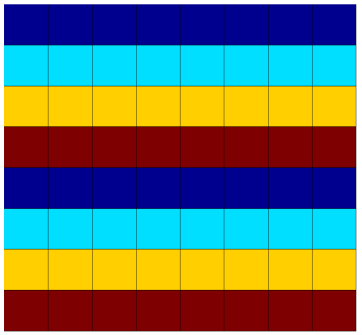


© 2010 Scott B. Baden / CSE 160 / Winter 2011

Data Distribution

- OpenMP implementations may not support nested parallelism: 2D decompositions





```
#pragma omp parallel for schedule(…)
for i = 0 to n-1
#pragma for schedule( … )
    for j = 0 to n-1
    z0 =Complex (x[i],y[i])
    while (| zk|< 2 or k >MAXITER)
            zk++  += zk2 + c
    Ouput(i,j) = kç
```

Dynamic scheduling with OpenMP

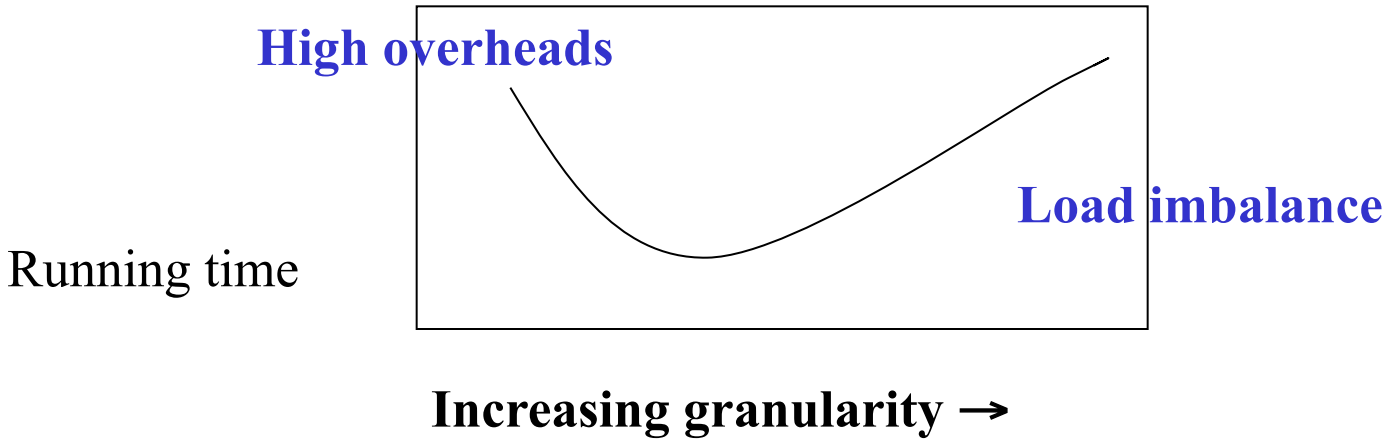
- Iterations mapped at run time
- Also called processor *self-scheduling*
- *What are the tradeoffs?*

```
#pragma omp parallel private(i) shared(n)
{
#pragma omp for schedule(dynamic, CHUNK)
for(i=0; i < n; i++)
    work(i);
}
```

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Processor self-scheduling

- Each thread samples a unique set of indices, will change from run to run
- Sample a shared counter or work queue to obtain work
- User tunes work granularity (chunk size) to trade off the overhead of sampling the queue against increased load imbalance



© 2010 Scott B. Baden / CSE 160 / Winter 2011

How does self scheduling work?

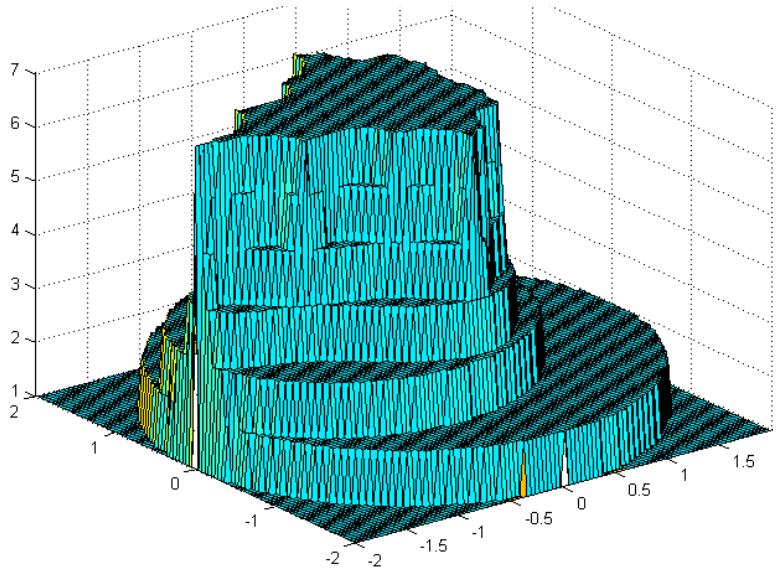
```
while (getNextChunk(&mymin,&mymax ))
    for i = mymin to mymax
       work(i);
    end for
end while
```

```
SelfScheduler S(n,P,CSize);
Boolean getNextChunk(int * mymin, int * mymax ){

   #pragma omp parallel

   #pragma omp critical
   {
       k = S.counter += S.ChunkSize;
   }
   mymin = k;
   mymax = k + S.chunkSize;
}
```

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Granularity tradeoff

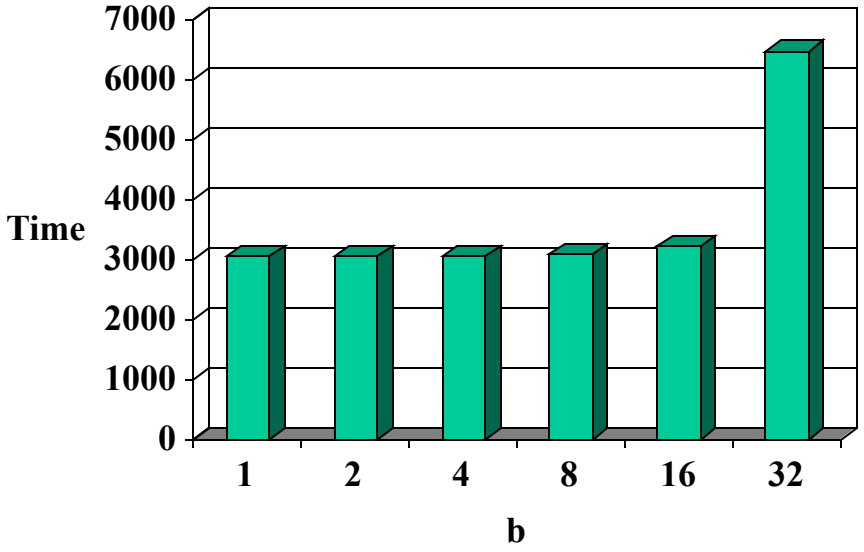
- The cost of load balancing goes up as we improve the load imbalance
- The finest granularity: each point a separate task
- Coarsest granularity: one block per processor
- What is the tradeoff ?



© 2010 Scott B. Baden / CSE 160 / Winter 2011

Simulations

- Let's simulate the workload distribution
- Assumptions:
  - Work distribution is instantaneous (zero cost)
  - Running time time is proportional to # iterations *k*
- Vary the chunk size *b*
- Optimal running time on 16 cores: 3044



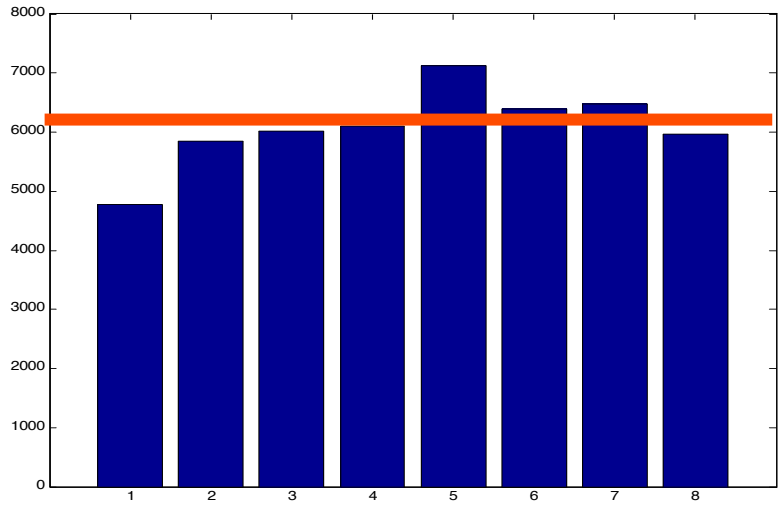
© 2010 Scott B. Baden / CSE 160 / Winter 2011

Limits to performance

- What happened when the chunk size was too large?
- Consider P=8
- The running times on the processors:

4771	5855	6018	6101
7129	6390	6470	5964

- The optimal running time is 6088



© 2010 Scott B. Baden / CSE 160 / Winter 2011

Guided self-scheduling

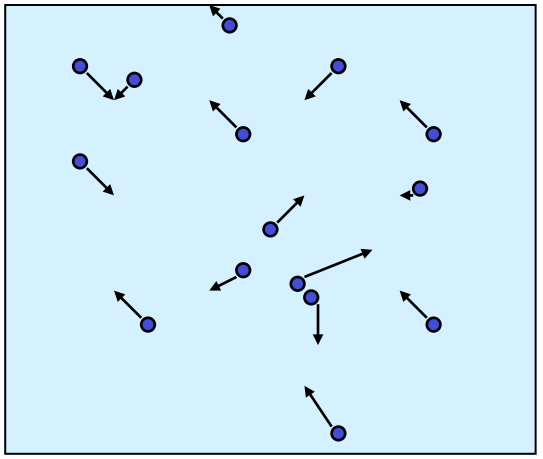
- We need many more pieces of work than processors (say 10 times) to a do a good job of load balancing
- But the cost of load balancing goes up as we improve the load imbalance
- Adjust or "guide" task granularity in proportion to the amount of remaining work
- For example, g = NRemain/10P
- When there's lots of work to do, we hand out large chunks. Toward the end, we hand out smaller chunks
- See Polychronopoulos, C.D., and Kuck, D.J. "Guided self-scheduling: a practical scheduling scheme for parallel supercomputers." *IEEE Trans, on Computers*, Vol. C-36, No. 12, Dec. 1987, pp. 1425-1439.

© 2010 Scott B. Baden / CSE 160 / Winter 2011

The N-body problem

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"
  - N can range from thousands to millions
  - No general analytic (exact) solution when N > 2
  - Numerical simulations required
- 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



© 2010 Scott B. Baden / CSE 160 / Winter 2011

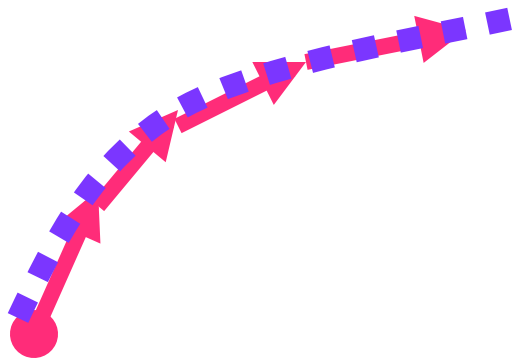
Discretization

- Because we cannot solve the problem analytically we must solve it numerically
- Particles move continuously through space and time
- On a computer we represent continuous values using a discrete approximation

The calculation

- Evaluate forces at discrete points in time, called timesteps Δt, $2\Delta t$, $3\Delta t$, …
  - Δt is called the *time discretization* or *discrete time step* (a parameter)
- "Push" the bodies according to Newton's third law

$$\mathbf{F} = m\mathbf{a} = m\, d\mathbf{u}/dt$$

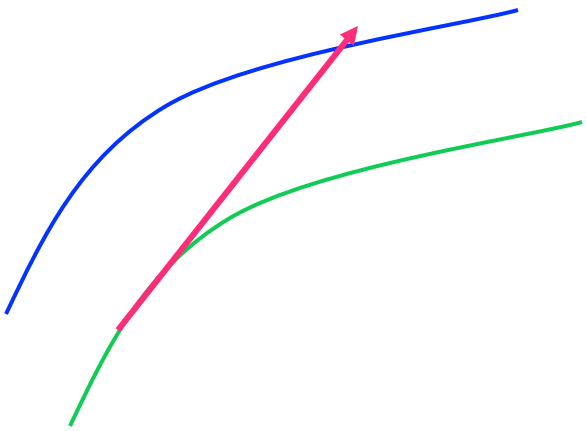


while (current time < end time)
  forall bodies i ∈ 1:N
    compute force F_i induced by all bodies j ∈ 1:N
  update position x_i by $F_i\, \Delta t$ forall i
  current time += Δt
end

© 2010 Scott B. Baden / CSE 160 / Winter 2011

Selecting Δt

- 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



Particle B's trajectory

Particle A's trajectory

© 2010 Scott B. Baden / CSE 160 / Winter 2011

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 $O(N^2)$

  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**

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

  G is the **gravitational constant**

© 2010 Scott B. Baden / CSE 160 / Winter 2011