Lecture 9

Irregular applications
Particle Methods
In class exercises

Announcements

- Testing time – final choice
  - 7pm to 9pm

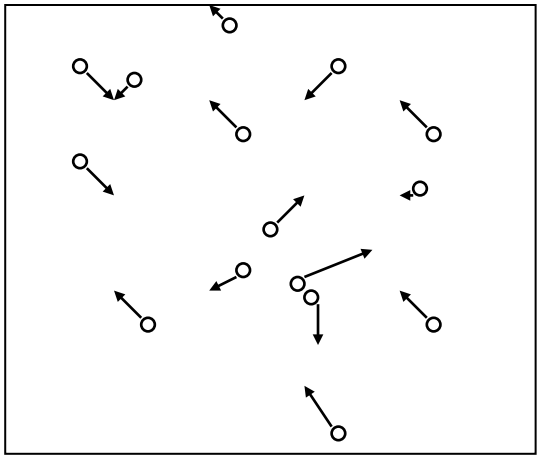
Today's lecture

- Advanced OpenMP Scheduling
- Load Balancing
- Code design

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



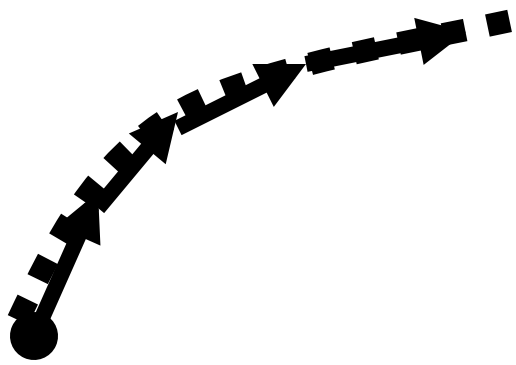
Discretization

- 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

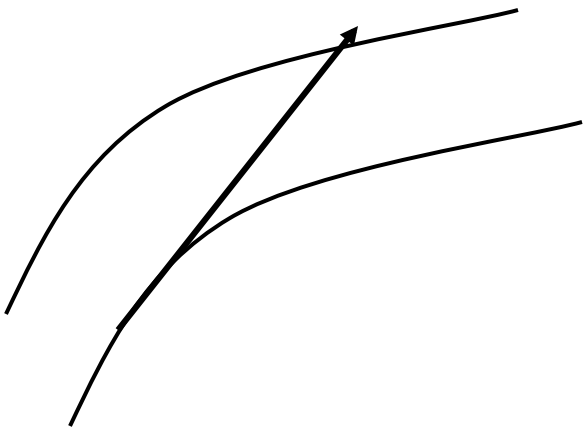
$$\mathbf{F} = m\mathbf{a} = m\, d\mathbf{u}/dt$$



```
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 Δt  forall i
    current time += Δt
end
```

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

Computing the force

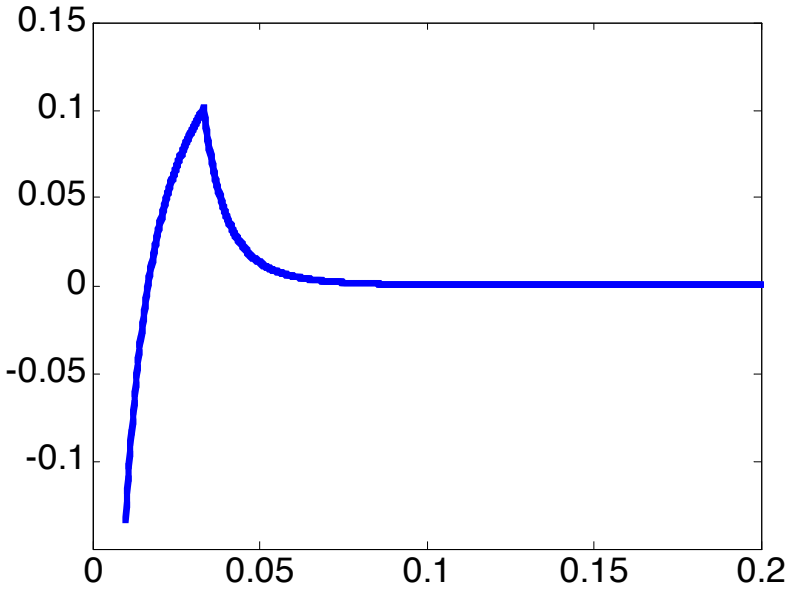
- The running time of the computation is dominated by the force computation, so we ignore the push phase
- The simplest approach is to use the direct method, with a running time of $O(N^2)$

  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} = distance(x_i, x_j)

  G is the **gravitational constant**
- Let's consider a more highly localized force

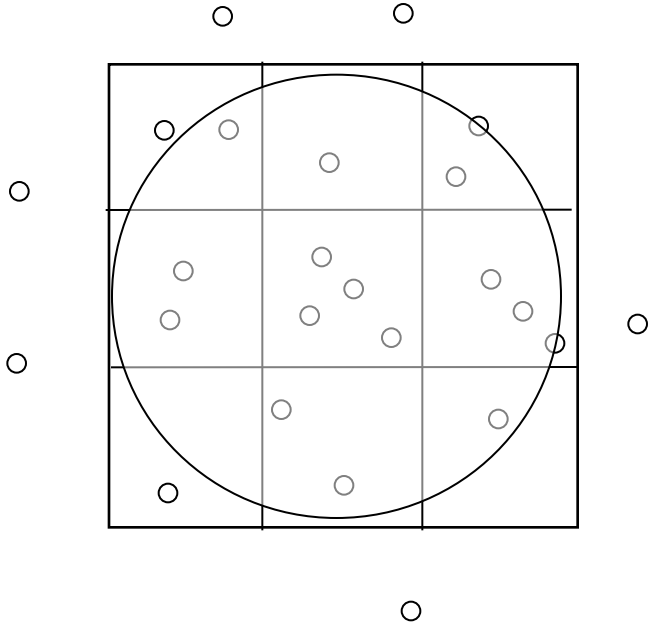
Localized Van Der Waals Force



$$F(r) = C\,(2 - 1 / (30r)) \qquad r < \delta$$
$$C / (30r)^5 \qquad r \geq \delta,$$
$$C = 0.1$$

Taking advantage of locality

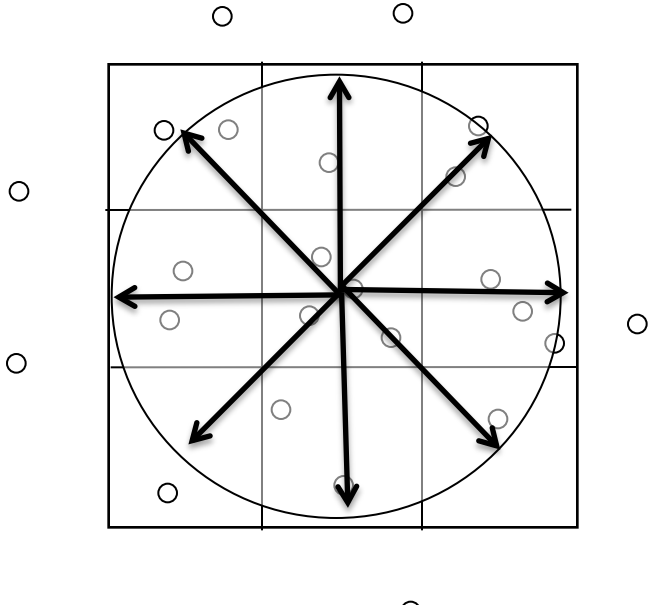
- We don't compute all $O(N^2)$ interactions
- To speed up the search for nearby particles, store in a mesh (Hockney & Eastwood, 1981)
- Spatial sort
  $(x,y) \rightarrow \text{Bin } (x-x_0)*m,(y-y_0)*n)$
- Nearest neighbor interactions



Jim Demmel, U. C. Berkeley

Implementation

```
include <list>;
typedef struct{
  double x, y, s;
  double u,v, u_o, v_o;
} particle;
list<particle> **World;
```



```
#pragma omp parallel for schedule(dynamic,chunk)
for i = 0 : N-1
  for j = 0 : N-1
     for x= -1 : +1
        for y = -1 : +1
            CalcForce(World[i][j],World[i+x][j+y]);
```

Today's lecture

- Advanced OpenMP Scheduling
- Load Balancing
- Code design

Why do we look at numerically intensive applications?

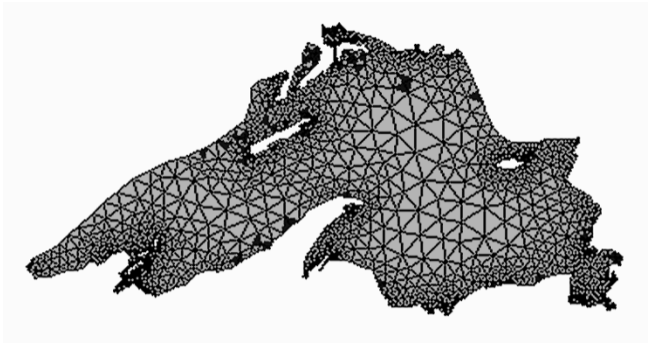
- Highly repetitive computations are prime candidates for parallel implementation
- Tight loop nests are the "low hanging fruit"
- Many real world applications
  - Data Mining
  - Image processing
  - Simulations – financial modeling, weather, biomedical
- We can classify applications according to Patterns of *communication* and computation that persist over time and across implementations

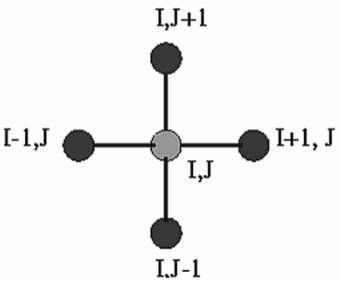
  Phillip Colella's 7 Dwarfs

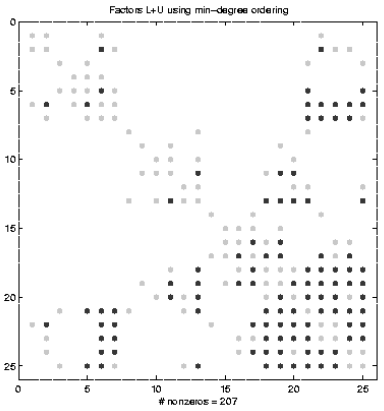
Courtesy of Randy Bank

Classifying the application domains

- Structured grids
  - Image processing, simulations
- Dense linear algebra:
  - Matrix multiply, Gaussian elimination, Data mining
- N-body methods
- Sparse linear algebra
  - In a sparse matrix, we take advantage of knowledge about the locations of non-zeros, improving some aspect of performance
  - Used widely
- Unstructured Grids
- Spectral methods (FFT)
- Monte Carlo







Courtesy of Randy Bank

Application-specific knowledge is important

- There currently exists no tool that can turn a serial program into an efficient parallel program

  … for all applications … all of the time… on all hardware

- The more we know about the application…

  … specific problem … math/physics ... initial data …

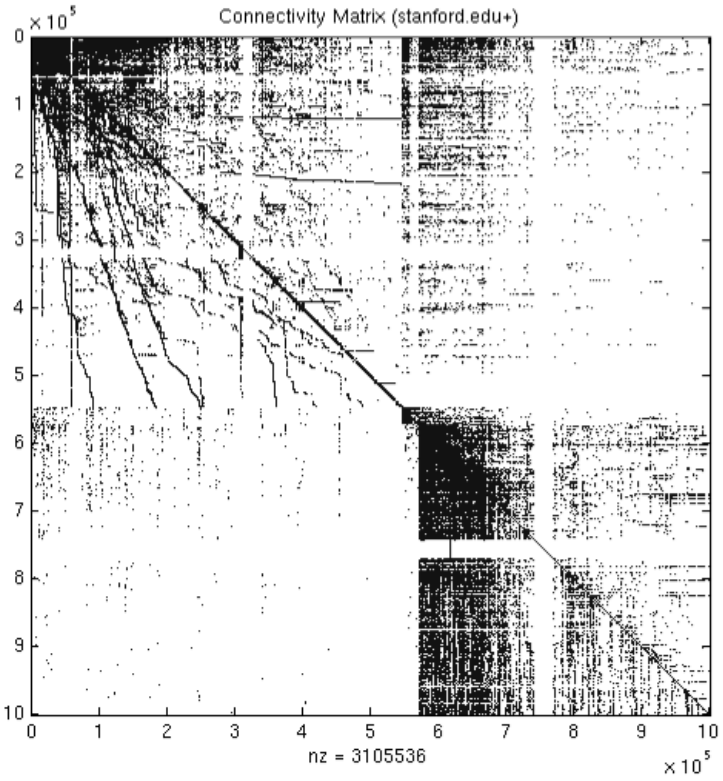
  … context for analyzing the output…

  … the more we can improve productivity

- Issues
  - Data motion and locality
  - Load balancing
  - Serial sections

Example sparse matrix

- 1M x 1M submatrix of a web connectivity graph constructed from an archive at Stanford Web Base
- 3 non-zeroes/row
  - Dense: $2^{20} \times 2^{20} = 2^{40}$ $= 1024$ Gwords
  - Sparse: $(3/2^{20}) \times 2^{40}$ $= 3$ Mwords
  - Sparse representation saves a factor of 1 million in storage



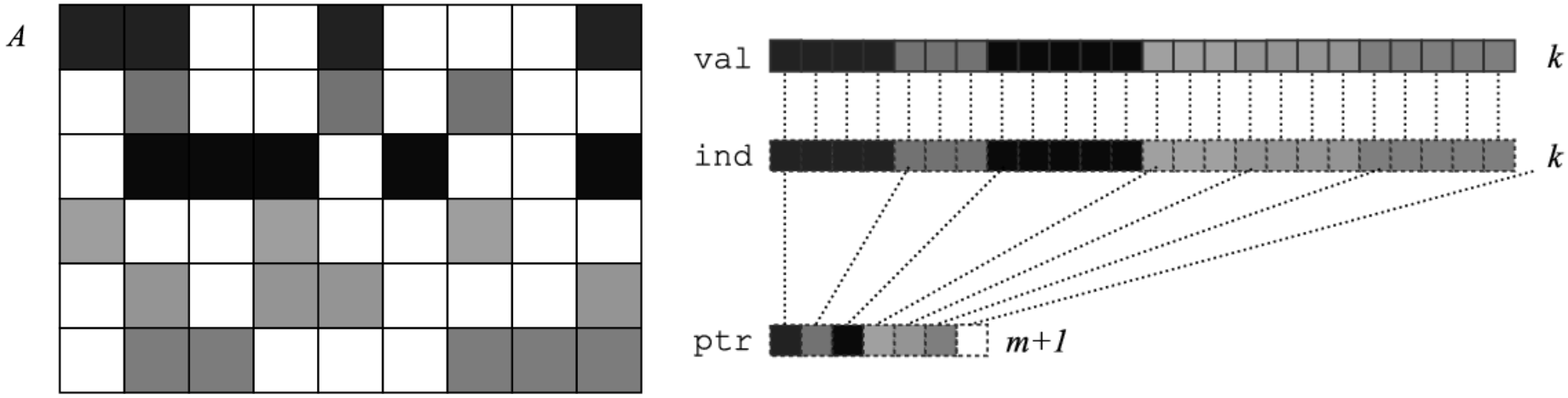
Courtesy of Jim Demmel

Sparse Matrix Linear Algebra

- Sparse matrix-vector multiply

  y[i] += A[i,j] × x[j]

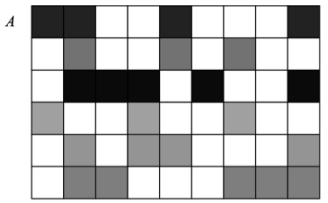
- Many formats, common one is Compressed Sparse Row (CSR)

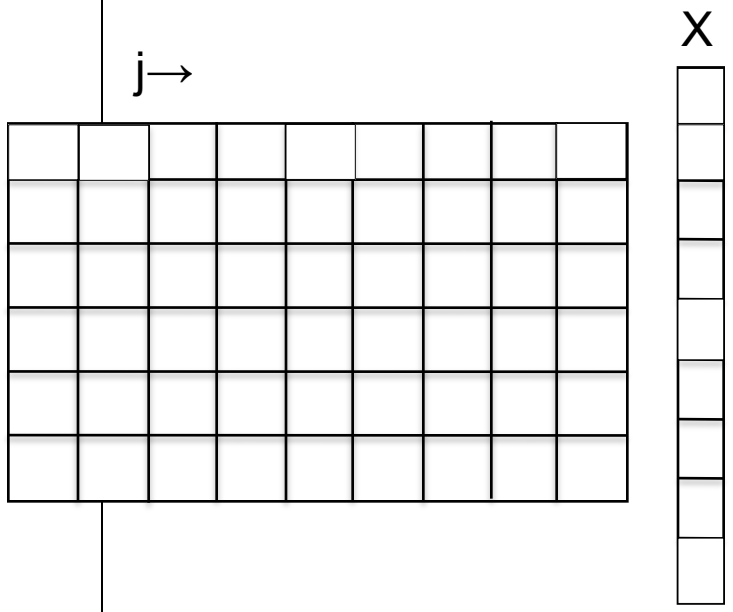


Jim Demmel

Sparse matrix vector multiply

```
// y[i] += A[i,j] × x[j]
#pragma parallel for schedule
            (dynamic,chunk)
for i = 0 : N-1        // rows
  i0= ptr[i]
  i1 = ptr[i+1] – 1
  for j = i0 : i1      // cols
    y[ ind[j]] +=
        val[ j ] *x[ ind[j] ]
  end j
end i
```





In class exercises

Questions

1. Time constrained scaling
2. Tree Summation
3. Synchronization
4. Iteration to thread mapping
5. Printing the letters of the alphabet
6. Removing data dependencies
7. Dependence analysis
8. Performance

1. Time constrained scaling

- Sum N numbers on P processors
- Let N >> P
- Determine the largest problem that can be solved in time $T=10^4$ time units on 512 processors
- Let time to perform one addition = 1 time unit
- Let β = time to add a value inside a critical section

Performance model

- Local additions: N/P - 1
- Reduction: β (lg P)
- Since N >> P

  $T(N,P) \sim (N/P) + \beta (\lg P)$
- Determine the largest problem that can be solved in time T= 10^4 time units on P=512 processors, β = 1000 time units
- Constraint: $T(512,N) \leq 10^4$

  $\Rightarrow (N/512) + 1000 (\lg 512)$

  $= (N/512) + 1000*(9) \leq 10^4$

  $\Rightarrow \quad N \leq 5 \times 10^5$ (approximately)

2. Tree Summation

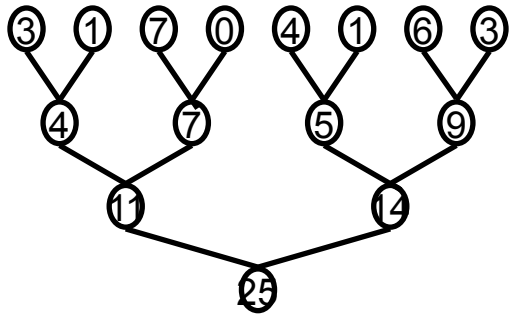
- Input: an array x[], length N >> P
- Output: Sum of the elements of x[]
- Goal: Compute the sum in lg P time

```
sum = 0;
for i=0 to N-1
    sum += x[i]
```

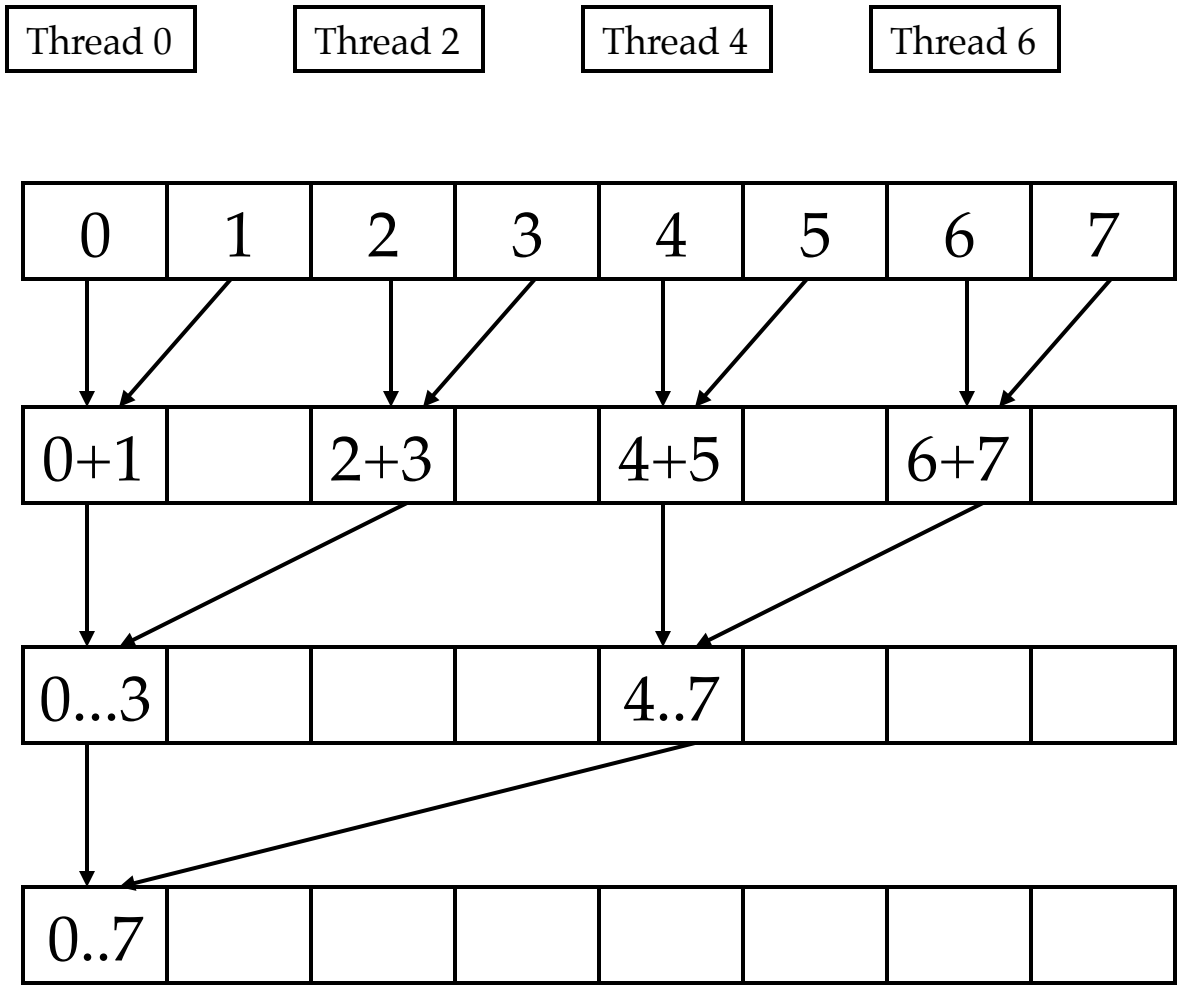
- Assume P is a power of 2, K = lg P
- Starter code

```
for m = 0 to K-1 {

}
```



Visualizing the Summation



3. Synchronization

List all possible outputs that result from running the following

```
#pragma omp parallel for shared(j,k)
for ( int i=0, j=0, k=0; i< 5; i++ )
    j = j + 10;
    k = j + 10;
}
cout << “k = “ << k << endl;
```

4. Iteration to thread mapping

```
#pragma omp parallel shared(N,iters) private(i)
#pragma omp for
for (i = 0; i < N; i++)
        iters[i] = omp_get_thread_num();
```

$HW/Examples/Assign

N = 9, # of openMP threads = 3

0 0 0 1 1 1 2 2 2

N = 16, # of openMP threads = 4, schedule(static,2)

0 0 1 1 2 2 3 3 0 0 1 1 2 2 3 3

N=9: 0 0 1 1 2 2 0 0 1

N = 16, # of openMP threads = 4, schedule(dyanmic,2)

3 3 0 0 1 1 2 2 3 3 3 3 3 3 3 3

2 2 3 3 0 0 1 1 2 2 2 2 2 2 2 2

5. Printing the letters of the alphabet

```
#pragma omp parallel private(i){
    int LettersPerThread = 26 / omp_get_num_threads();
    int ThisThread = omp_get_thread_num();
    int Start= 'a'+ThisThread*LettersPerThread;
    int End = 'a'+(ThisThreadNum+1)*LettersPerThread;
     for (i=Start; i<End; i++)
        printf ("%c", i);
}
```

6. Removing data dependencies

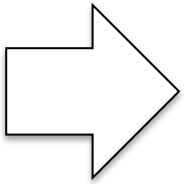
- B initially: 0 1 2 3 4 5 6 7
- B on 1 thread: 0 9 10 11 12 21 22 23
- B on 2 threads : 0 17 18 19 12 13 14 15
- How can we split into 2 loops so that each loop parallelizes, the result it correct?

```
for i  = 1 to N
    B[i] += B[N-i];
```

Splitting a loop

- For iterations i=N/2+1 to N, B[200-i] reference newly computed data
- All others reference "old" data
- B initially: 0 1 2 3 4 5 6 7
- Correct result: 0 9 10 11 12 21 22 23

```
for i = 1 to N
    B[i] += B[N-i];
```



```
for i = 1 to N/2
    B[i] += B[N-i];
for i = N/2+1 to N
    B[i] += B[N-i];
```

7. Loop Dependence Analysis

- Can we run parallelize these with openmp?

1.
```
for i = 1 to N
    A[i] = A[i] + B[i-1];
```

2.
```
for i = 1 to N
    A[i+1] = A[i] + 1;
```

3.
```
A[0] = 0;
for i = 1 to N-1
    A[i] = A[i-1] + i;
```

4.
```
for i = 1 to N{
    A[i] = B[i];
    C[i] = A[i] + B[i];
    E[i] = C[i+1];
}
```

Nested loop dependence analysis

- Can we parallelize the inner loops as shown?

LOOP #1

```
for j = 0 to n-1
    for i = 0 to n-1
        A[i, j+1] = A[i, j];
```

LOOP #2

```
for j = 0 to n-1
    for i = 0 to n-1
        A[i, j+1] = A[i, j];
```

8. Performance

- You observe the following running times for a parallel program running a fixed workload N
- Assume that the only losses are due to serial sections
- What is the speedup and efficiency on 8 processors?
- What will the running time be on 4 processors?
- What is the maximum possible speedup on an infinite number of processors?
- What fraction of the total running time on 1 processor corresponds to the serial section?
- What fraction of the total running time on 2 processors corresponds to the serial section?

NT	Time
1	10000
2	6000
8	3000