

### 1. Parallel Programming

In class, we discussed a code which used a one-dimensional integral to estimate  $\pi$ . In fact, there are many methods which can be used estimate  $\pi$  using random numbers.

Perhaps the simplest (and least efficient) generates uniformly random  $x$  and  $y$  positions inside a unit square. By comparing the number of points inside the unit circle to the total number of points, we can determine the relative area between the quadrant of a unit circle and a unit square.

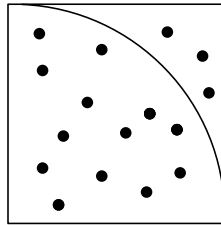


Figure 1: The ratio of the total number of random points inside the unit circle compared to the total number of random points is related to the ratio of the areas.

- (a) (5 pts) What are the relative areas between the quadrant of the unit circle and the unit square?

*The ratio of areas is given by :*

$$\frac{A_{\text{circle}}}{A_{\text{square}}} = \pi r^2/4/r^2 = \pi/4$$

- (b) (10 pts) Outline the PDL for a SHORT subroutine which calculates  $\pi$  using this method for some number of points  $n$ .

```
zero a counter
for each particle
  pick x and y positions which are
    uniformly distributed between zero and one
```

```
    calculate the distance from the point to the
        origin
    if the distance is less than one, add one to the counter
continue the particle loop
```

```
pi is equal to four times the counter / n
```

- (c) (10 pts) Outline the PDL needed to implement this routine inside a parallel computer using MPI. You may assume that there is a “good” random number generator available on each processors which creates numbers that are independent of the other processors. The exact MPI calls are NOT needed, but the sequence they will be called in and their general function is.

```
initialize MPI
```

```
find my local processor ID
find the total number of processors
```

```
on processor zero, input the number of points
to be generated
```

```
broadcast the total number of points to all processors
from processor zero
```

```
on each processor, call the random point subroutine with
n/ nproc points
```

```
do a summation global reduce on the results of the call
to processors zero
```

```
on processors zero, divide the results by the number of
processors thus averaging the results
```

```
one processor zero, print out the results
```

```
finalize MPI
```

- (d) (10 pts) Based on your understanding of parallel computing, un-

der what circumstances will this code run efficiently on a multi-processors machine? When will a single node computer most likely out-perform a parallel version of this code?

*If the number of points is comparable to the number of processors, the communication time will dominate in the problem and the parallel code will run slowly. If there are millions of points for each processor, most of the time will be spent calculating rather than communicating, and the code will run efficiently.*

## 2. Finite Difference Methods

A simple form of the diffusion equation is given by

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

When we discretize the problem to a regular grid, we can write

$$\begin{aligned} t^j &= j(\Delta t) \\ x_i &= i(\Delta x) \end{aligned}$$

so that

$$u(x_i, t^j) = u_i^j$$

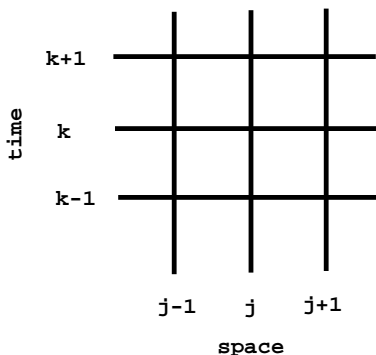


Figure 2: A finite difference grid for the one dimensional diffusion equation.

- (a) (5 pts) Using the notation defined above, write an implicit finite difference equation for solving the diffusion equation.

Assuming that  $k = \Delta t$  and  $h = \Delta x$ ,

$$\frac{1}{k}[u_i^{j+1} - u_i^j] = \frac{1}{h^2}[u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1}]$$

- (b) (10 pts) Using this equation, sketch the matrix form for this problem assuming that the grid is divide into  $n$  spatial points. Explain how large the final matrix will be and how the system can be solved.

The basic form of these equations is:

$$(1 + 2\frac{k}{h^2})u_i^{j+1} - \frac{k}{h^2}u_{i+1}^{j+1} - \frac{k}{h^2}u_{i-1}^{j+1} = u_i^j$$

For the  $i$ th row, the only non-zero coefficients are  $a_i = (1 + 2\frac{k}{h^2})$ ,  $a_{i-1} = -\frac{k}{h^2}$  and  $a_{i+1} = -\frac{k}{h^2}$ . Which means we have a tridiagonal system.

$$\begin{bmatrix} d & d_1 & 0 & 0 & 0 & \cdots & 0 \\ d_1 & d & d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_1 & d & d_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \cdots & \vdots & \vdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & d_1 & d \end{bmatrix} \begin{bmatrix} u_1^{j+1} \\ u_2^{j+1} \\ u_3^{j+1} \\ \vdots \\ u_n^{j+1} \end{bmatrix} = \begin{bmatrix} u_1^j \\ u_2^j \\ u_3^j \\ \vdots \\ u_n^j \end{bmatrix}$$

where  $d = (1 + 2\frac{k}{h^2})$  and  $d_1 = -\frac{k}{h^2}$

The system is a sparse diagonal system with  $n$  unknown coefficients.

Although any type of linear solver can be used to solve this problem, tridiagonal solvers are particularly well suited for this purpose.

- (c) (10 pts) List three types of boundary conditions that are possible for this problem, and explain briefly how they would be implemented.

- *Dirichlet - boundary conditions which have defined value. In this case, we set the value  $u_i^j$  at the boundary to the constant value at all time steps. This row can then be dropped from the matrix since it is time invariant.*

*Assuming the Dirichlet condition is on the left boundary, the finite diff equation becomes*

$$\frac{1}{k}[u_i^{j+1} - u_i^j] = \frac{1}{h^2}[u_{i+1}^{j+1} - 2u_i^{j+1} + u_D^{j+1}]$$

*we have assumed that  $i$  is the value just to the right of the left hand boundary.*

- *Neumann - boundary conditions which have a defined derivative. We must set the derivative equal to the correct value. Then*

$$\frac{1}{k}[u_i^{j+1} - u_i^j] = \frac{1}{h^2}[u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1}]$$

*becomes*

$$\frac{1}{k}[u_i^{j+1} - u_i^j] = \frac{1}{h^2}[(u_{i+1}^{j+1} - u_i^{j+1}) - (u_i^{j+1} - u_G^{j+1})]$$

*Where  $u_G$  is an unknown ghost point outside the boundary. We use a center difference technique to define this so that*

$$B = \frac{1}{2h}(u_{i+1}^{j+1} - u_G^{j+1})$$

*We have assumed that  $u_i$  is AT the boundary.*

*Where  $B$  is the derivative at the boundary.*

- (d) *Periodic BC - we connect the opposite sides of the domain. Conceptually, this is the simplest type of boundary. We just replace*

$$\frac{1}{k}[u_i^{j+1} - u_i^j] = \frac{1}{h^2}[(u_{i+1}^{j+1} - u_i^{j+1}) - (u_i^{j+1} - u_{i-1}^{j+1})]$$

*with*

$$\frac{1}{k}[u_i^{j+1} - u_i^j] = \frac{1}{h^2}[(u_{i+1}^{j+1} - u_i^{j+1}) - (u_i^{j+1} - u_R^{j+1})]$$

*where  $u_R$  is on a value on the right hand boundary.*

*We have assumed that  $u_i$  is AT the boundary.*

### 3. Ordinary Differential Equations

One of the principle reasons that the Eniac computer was built was to tabulate solutions to the artillery equation-

$$\begin{aligned}\frac{d^2 y}{dt^2} &= -\eta \frac{dy}{dt} - g \\ \frac{d^2 x}{dt^2} &= -\eta \frac{dx}{dt}\end{aligned}$$

where  $g$  is the gravitational acceleration and  $\eta$  is a the drag coefficient. The  $x$  and  $y$  are the positions of the shell which changes as a function of time. Initially artillery shells start with

$$\begin{aligned}x(0) &= 0 \\ y(0) &= 0 \\ \frac{dx}{dt}(0) &= v_x \\ \frac{dy}{dt}(0) &= v_y\end{aligned}$$

- (a) (5 pts) Break the two equations into four first-order equations.

$$\begin{aligned}\frac{dv_y}{dt} &= -\eta v_y - g \\ \frac{dv_x}{dt} &= -\eta v_x \\ \frac{dy}{dt} &= v_y \\ \frac{dx}{dt} &= v_x\end{aligned}$$

- (b) (5 pts) Outline the first-order Forward Euler method. Be brief, but specific.

*The basic equation is*

$$y_{n+1} = y_n + y'(n) \Delta t$$

*Each time we update  $y_n$ , we need to evaluate the derivative at the current timestep.*

- (c) (5 pts) Explain how you can monitor and control the size of the numerical error in this problem using only the First order Euler method. Explain the basic principle of why this works.

*Despite the fact that forward Euler is a simple method, we it still has a truncation error which is  $O(h^2)$*

$$y_{n+1} = y_n + y'(n) \Delta t + O((\Delta t)^2)$$

*This means if we decrease the step size, the error decrease. Comparing the differences between steps of size  $h$  and two steps of size  $h/2$  can give us an estimate about the accuracy of the method.*

*We can also keep the error low by looking at a finite difference approximation of the second derivative*

$$y'' = \frac{y'_n - y'_{n-1}}{t_n - t_{n-1}}$$

*then the error is approximately  $y''/2h^2$*

- (d) (5 pts) Write a short PDL routine for integrating the equations using the forward Euler method and your method of error control.

```
take a trial step
```

```
take two trial steps using half the trial step size
```

```
find the residual between the results
```

```
if the residual is less than the tolerance,
```

```
the new step size should be the current step
```

```
times the sqrt of the measured tolerance/ required tolerance
```

```
else
```

```
repeat the trial steps with a smaller step size
```

#### 4. Iterative Linear Systems

- (a) (10 pts) Most iterative linear methods use the same basic analytic framework. Describe how we move a linear system ( $Ax = b$ ) into a method which can be solved iteratively using this general framework.

*Given a linear system*

$$Ax = b \tag{1}$$

*We can always split the matrix  $A$  into two parts-*

$$A = B + R \tag{2}$$

*where  $B$  is an easily invertible matrix and  $R$  is a remainder. We can then write*

$$Bx = -Rx + b \tag{3}$$

*giving us*

$$x = -B^{-1}(A - B)x + B^{-1}b \tag{4}$$

*We can write this as an iterative equation*

$$x^{(t+1)} = -B^{-1}(A - B)x^{(t)} + B^{-1}b \tag{5}$$

*The key thing elements to making this work are discussed below.*

- (b) (5 pts) Under what conditions will the iterative method converge?  
*The key thing for convergence is the behavior of  $B^{-1}(A - B)$ . If the maximum eigenvalue is less than 1, the method will converge.*
- (c) (5 pts) Even though iterative linear systems use the same concepts, their rates of convergence depend on three key things. What are they?

*Three main things determine the characteristics of mesh relaxation methods:*

- *What invertible matrix is used for the iteration?*
- *How is the old and predicted value combined for the iteration?*
- *What order are points updated during the iteration?*