



2-D Excitable Media Simulations On an MPI Cluster

Joe DeMaio, Beckie Russell, Mike Poko

CMPT 585

12/04/03

Dr. R. Zaritski, MSU



Summary of objectives

- Modify Initial conditions from one to many spiral pairs
- Modify border conditions from no-flux to torus
- Implement a beat to beat frequency counter for a variety of conditions



Randomly Placed Spiral Pairs

Generating Triplets From
Randomly Placed Refractoriness
and Excitation Patch-Pairs

ClassicIC()

- Placement of excitation and refractoriness.

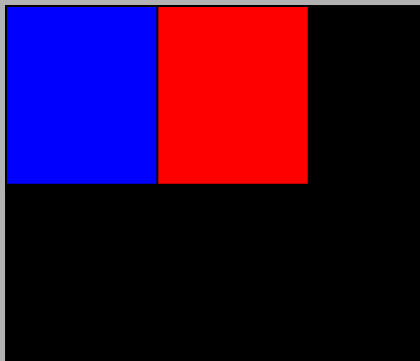


Figure 2.1 Classic Initial Conditions

- Development into a single spiral.



PatchesIC()

- Placement of excitation and refractoriness.
- Overlapping squares of equal size.
- Varying degrees of overlap.
- Varying sizes of patches.

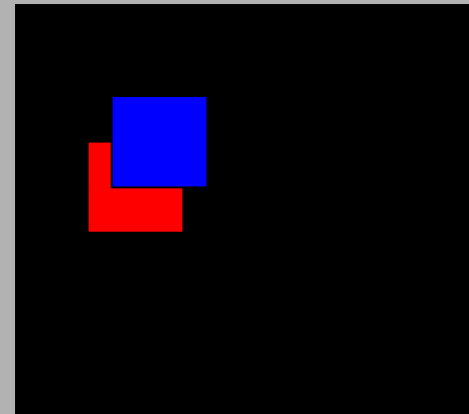


Figure 2.3 Single randomly placed patch pair.



Randomizations

- Randomized Elements
 - Side length
 - between 10 and 30
 - interference with sizes larger than 30 and more than 2 or 3 patch pairs
 - Offset
 - x and y offsets determined independently
 - at least $1/3$ of but no more than $2/3$ of the side length
 - guarantees spiral development

Randomizations

- Randomized Elements (continued)
 - placement
 - x, y coordinates of upper left corner of excitation
 - no guarantees that patch pairs won't overlap each other
 - direction of offset
 - independently determines x and y offset directions
 - ensures that spirals will rotate in different directions

Source Code

```
//select size of patch, position of patch and offset of
//refractoriness patch
    int side      =      10 + rand()%21;
    int xshift    = side/3 + rand()%(2*side/3 + 1);
    int yshift    = side/3 + rand()%(2*side/3 + 1);
    int x         =      rand()%(N - side - xshift);
    int y         =      rand()%(M - side - yshift);

//direction of shift of refractoriness patch from excitation patch.
    int xdir      =      rand() % 2;
    int ydir      =      rand() % 2;
    if(xdir == 0)
        xshift *= -1;
    if(ydir == 0)
        yshift *= -1;
```

Snapshots



Figure 2.4 Spiral Triplet, joined

Snapshots



Figure 2.5 Spiral Triplet, open

Conclusion

- Larger numbers of smaller patches led to spiral triplets much faster than fewer numbers of larger patches.
- A single triplet can exhibit the joined behavior or the open behavior seen in the two snapshots.



No-Flux to Torus Conditions

- Using No-flux conditions, waves were terminated at the window borders

Using Torus conditions,

- crossing the right border would continue at the left border,
- crossing the left border would continue at the right border,
- crossing the top border would continue at the bottom border, and
- crossing the bottom border would continue at the top border.



Torus Conditions

Two cases needed to be handled when implementing the torus conditions:

- More than one slave CPU
- One slave CPU

Torus Conditions

> 1 slave CPU

For wavefronts crossing the right border when $csize > 1$:

Send the data in column L_min_2 to the CPU with $lrank = 1$ (left most part of the Window)

```
// G5: RIGHTMOST CPU OF MULTI-CPU CASE SEND TO CPU 1  
// G5: lrank = size, then pass L-min_2 to lrank = 1
```

```
MPI_Send((void*)uu->GetColumnPtr(L_min_2), N_plus_1,  
         MPIFPTYPE, 1, 55+DiscrTime, MPI::COMM_WORLD);
```

Torus Conditions

> 1 slave CPU

CPU with `lrank = 1` receives data from the right most CPU and stores it in column 0.

```
// G5: Receive from CPU 1.  
MPI_Recv((void*)uu->GetColumnPtr(0), N_plus_1, MPI_FLOAT,  
csize, 55+DiscrTime, MPI::COMM_WORLD, status);
```

Torus Conditions

> 1 slave CPU

Similarly crossing the left border when $csize > 1$,
Send the data in column 1 to the CPU with $lrank$ equal to
 $csize$ (right most part of the window).

```
// G5: lrank = size, then pass column 1 lrank = csize  
MPI_Send((void*)uu->GetColumnPtr(1), N_plus_1, MPIFPTYPE,  
csize, 66+DiscrTime, MPI::COMM_WORLD);
```

Torus Conditions

> 1 slave CPU

CPU with `lrank = 1` receives data from the left most CPU and stores it in column `L_min_1`.

```
// G5: Receive from CPU 1.  
MPI_Recv((void*)uu->GetColumnPtr(L_min_1), N_plus_1,  
MPIFPTYPE, 1, 66+DiscrTime, MPI::COMM_WORLD, status);
```

Torus Conditions

1 slave CPU

Case when $csize = 1$

MPI Send and MPI Receive calls not needed. Torus conditions were implementing by appropriately manipulating values of the array holding the data.

```
FPTYPE tmpub;  
FPTYPE tmpub2;  
for(i=0;i<=N;i++)  
{  
    tmpub = (*uu)(i,1);  
    tmpub2 = (*uu)(i,L_min_2);  
    (*uu)(i,0) = tmpub2;  
    (*uu)(i,L_min_1) = tmpub;  
}
```

Torus Conditions

All Cases – Top & Bottom

Wavefronts crossing the top and bottom borders:
No MPI commands are use. The array values for the top and bottom of the window are manipulated.

```
FPTYPE tmpub;  
FPTYPE tmpub2;  
for (j=1; j<=L_min_1; j++)  
{  
    tmpub = (*uu) (1, j);  
    tmpub2 = (*uu) (N-1, j);  
    (*uu) (0, j) = tmpub2;  
    (*uu) (N, j) = tmpub;  
}
```

Torus Conditions

Illustration of torus conditions using multiple slave CPUs.



Torus Conditions

Illustration of torus conditions using a single slave CPUs.



Torus Conditions

Illustration of waves using torus conditions and 5 random spiral pairs as initial conditions.



torus1.jpg

Torus Conditions

Illustration of waves using torus conditions and 5 random spiral pairs as initial conditions.



torus2.jpg

Torus Conditions

Wave form development



Torus Conditions

Wave form development



Torus Conditions

Wave form development





Quick Overview

- In this section, we'll discuss the following:
 - The coding behind the Probe
 - Speed Up Analysis
 - Beat to Beat Intervals
 - 2 Interesting problems that arose
 - Internal CPU MPI passing
 - Outputting to Files on Clusters

The Probe - Setup

- Setup

- Set the probe on a specific computer
- Set a point to monitor – Center of Segment
- Create an output file

- Obstacles

- The only major obstacle to overcome was not creating files on many all CPU's
- Interesting problem arose – delete existing file or append to existing

Probe Setup

```
//start of probe set up
int Probe_Rank = 1;           //designates the rank in which the probe lies

int Probe_X = L/2; //give these values
int Probe_Y = N/2; //give these values
int Counter = 0; //counts the cycles of excitation
bool isExcited = false;
ofstream output;

if(countWaves == true)
{
    if(Probe_Rank == lrank){
        if>DeleteFile)
            system("rm -f Probe.dat");

        output.open("Probe.dat", ios::app);
    }
}
//end probe set up
}
```

The Probe - Coding

- The code is inserted just after the computations
- Code will only execute on a specific CPU
- Alternative solution – Instead of an If Statement, we could have used conditional compiling
- Simple but effective
 - The probe is simply 2 nested if statements
 - Checks Rank and excitation
- Outputs to a file
- Interesting observation: Output `<< (*uu)(i,j)` writes to a file on the computer with Rank 0

Probe Code

```
//MAIN PART:
if(countWaves == true){
    //calculations performed here
    //begin probe
    if(Probe_Rank == lrank){
        if( Probe_X == j && Probe_Y == i){
            if((*uu)(i,j) >= .6) {
                if( !isExcited ){
                    output << (*uu)(i,j) << " " << Counter << " Begin Excitation " << endl;
                    isExcited = true;
                    Counter = 0;
                }
                else{ Counter++; }
            }
            else{ //U < .6
                if(isExcited){ isExcited = false; } else{ Counter++; }
            }
        }
    }
}
//end probe
}
```

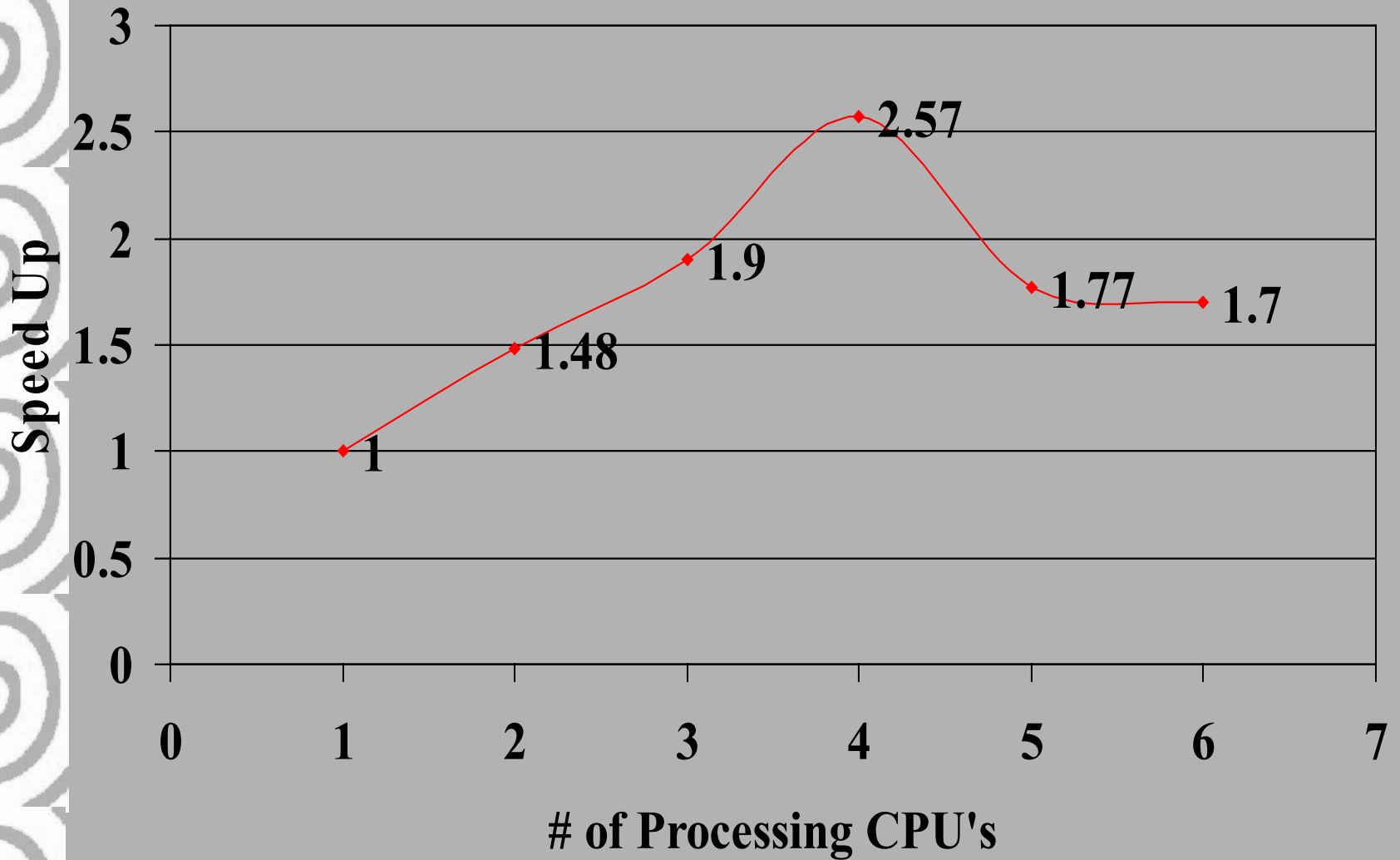
Speed Up Observations

- Adding additional CPU's into processing did decrease execution time
- Speed Up maxed out at 2.57
- Additional CPU's added beyond max number of available had an adverse affect on the speed up

Speed Up & Torus – A Problem

- Interesting results:
 - The speed up analysis was originally performed with a Torus system
 - With 5 Processing CPU's, CPU 1 talks to CPU 5
 - CPU 1 is CPU 5
 - When passing information from Left to Right or vice versa, severe slow down occurred
 - In one of the few tests we allowed to complete, 1000 cycles took 6 min 45 sec to execute
 - To show how drastic a slow down this is, a 20,000 cycle no torus execution on 5 processing CPU's only required 53 seconds to complete

Speed Up Analysis





Graphs of Beat to Beat

- 4 graphs
 - 1 Displays Torus results
 - 3 Display No Torus Results

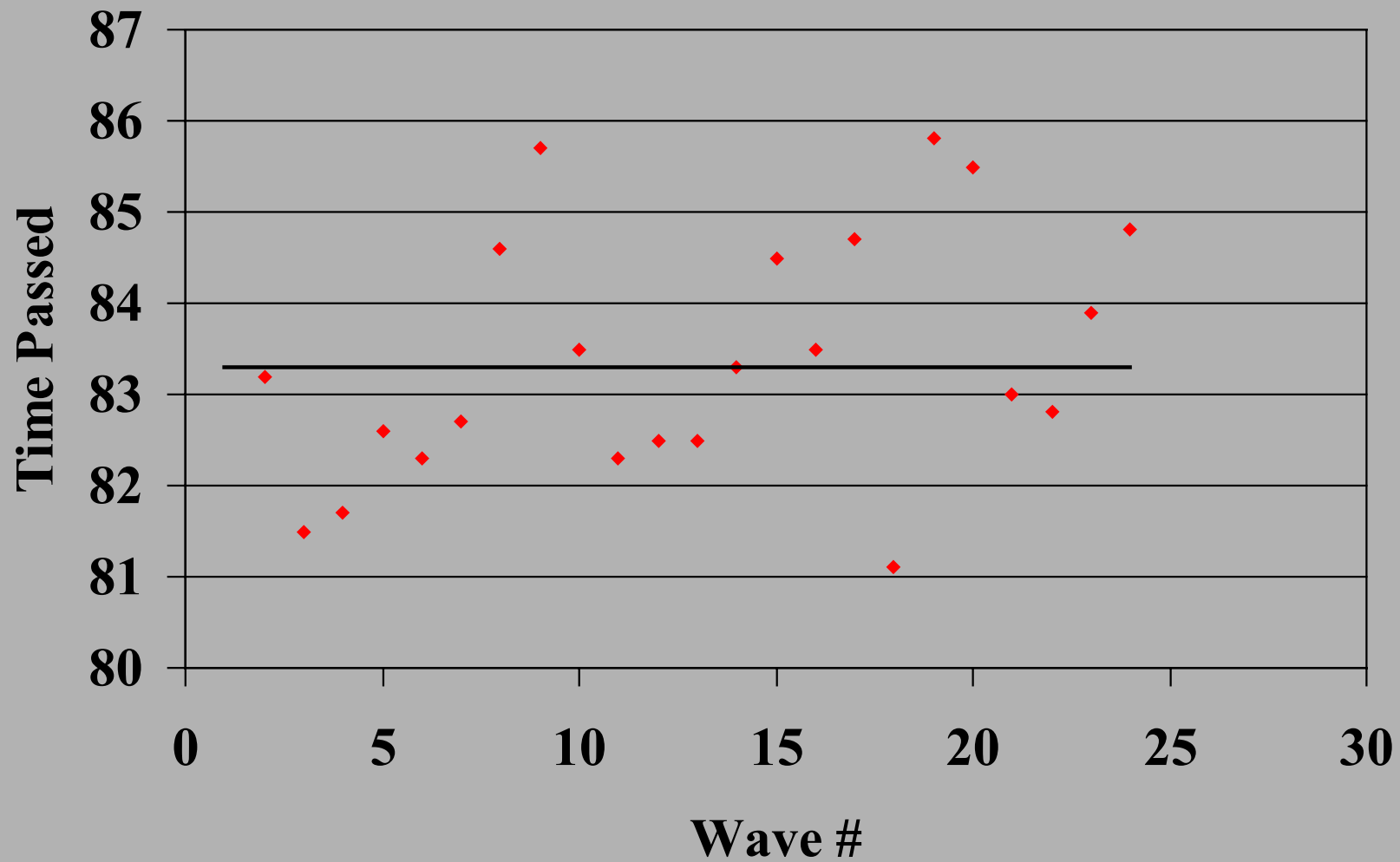
No Torus Charts

- In No Torus examples, an almost sinuous trend curve appears
- In each example the time between wave fronts is close to an average
- The time seldom deviates by an extreme amount
- Extreme deviations typically happened during the first several wave fronts

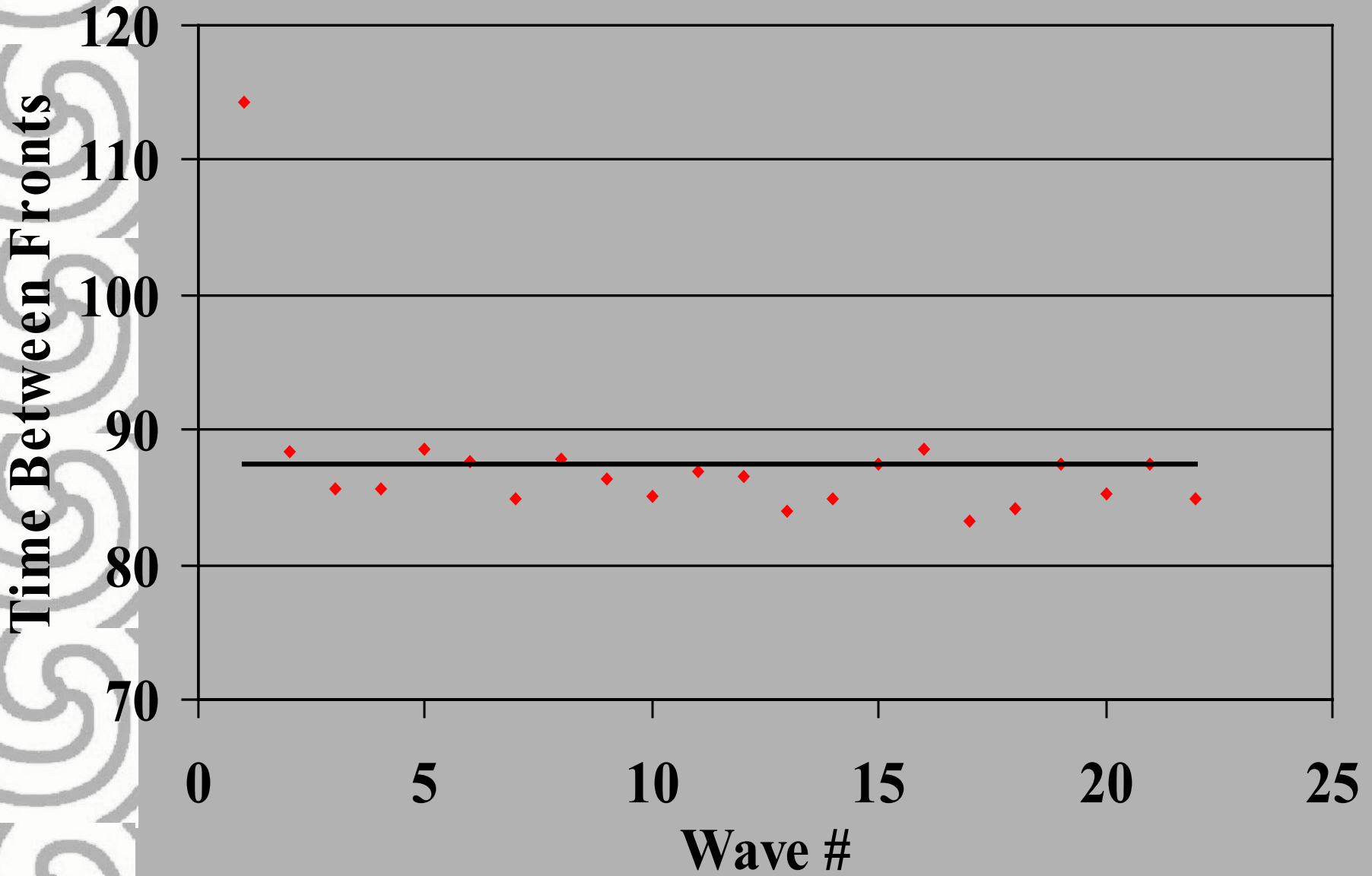
Torus Chart

- When torus was active, the wave fronts passed at more dispersed intervals
- The length of time between fronts was high at first, but as time passed they became much more frequent
- An average line was included, but it does not help to describe what is happening in the system.

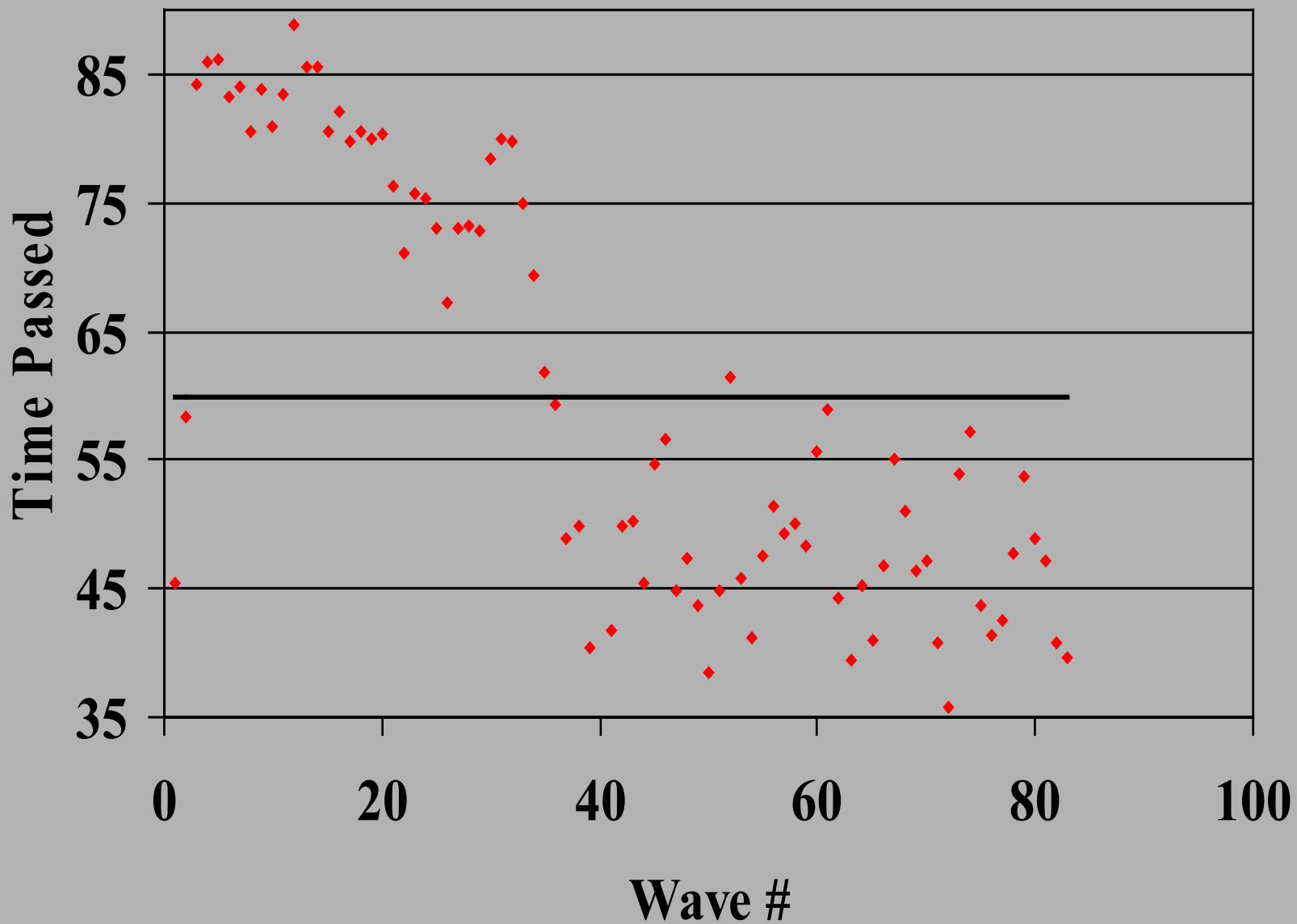
Wave Beats #1 - No Torus, 1 Patch, 2000 Cycles



Wave Beats #2 - No Torus, 2000 Cycles



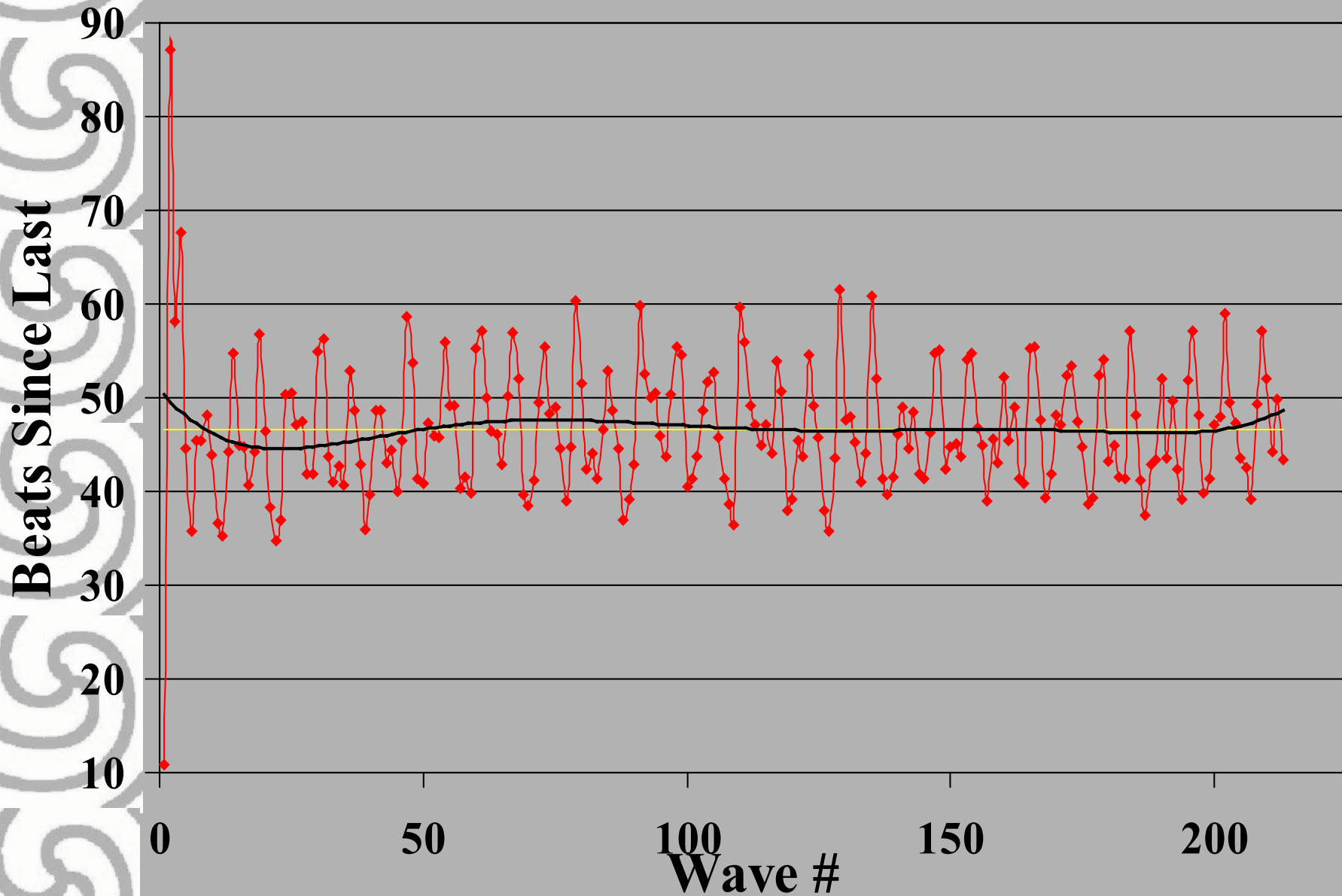
Beat to Beat – Torus



A more interesting Chart

- Red line traces the Time passed from wave to wave
- The Yellow line represents the Average Line
- The Black line is a Trend Line
- The Trend Line is a Polynomial of degree 6
- The line displays a sinuous curve
- It is interesting to see the points rebound about the average point.

Beat between Waves - No Torus; 20 Initial Points





Interesting Problems

- 2 Interesting problems arose:
 - MPI Processing from a CPU to itself
 - Writing to file on an Cluster

Message Passing Error

- MPI Processing

- When Size + 1 Processors were used, CPU 1 needed to communicate with CPU Size
- These are the same physical CPU
- When trying to pass information between these 2 CPU's, we experienced severe slow down
- Interestingly enough, if we add another processing CPU, thus breaking the internal communication, the execution time returned to normal
- This indicates that the problem was with the Message Passing between a CPU and itself
- We were unable to identify the cause of this problem

Writing to File

- The second interesting problem involved outputting to file
- The behavior of the code execution was not as expected
- It was expected that the file would be created on the disk space of the CPU creating the file, written to, and then we would have to transfer it to the master node
- What happened was:
 - File was created on Master
 - File on Master was overwritten or Appended
 - No activity appeared to happen on the slave
 - Outputting to a file was executed similarly to a cout statement

Responsibilities Matrix

Each individual in this group accomplished parts 1, 2, 3, and 4 independently. This required that we learn and understand the code as presented before making alterations and modifications. The remaining tasks were split up as detailed below.

Task	DeMaio	Poko	Russell
Part 5: Random Patches			X
Part 6: Torus Conditions	X		
Part 7: Beat to Beat Probe		X	
Snapshots			X
Probe Data: Graphs		X	
Speedup and Scalability Graphs		X	
Report: Formatting and Assembly	X		
Presentation: Formatting and Assembly	X		
Responsibilities Grid			X