

# Parallelization of GIS / Computational Geometry Problems

James Jefferson Jarvis  
Computer Engineering 525  
3 May 2004

## Problem

- ◆ I deal with many geographic points (latitude, longitude) and have written programs for finding closest geographic point, nearest road, and polygon that contains the point
- ◆ These programs take a long time to run with large number of points and/or large geographic datasets
- ◆ Would like to use parallel computing to speed up these operations

## Problem (cont.)

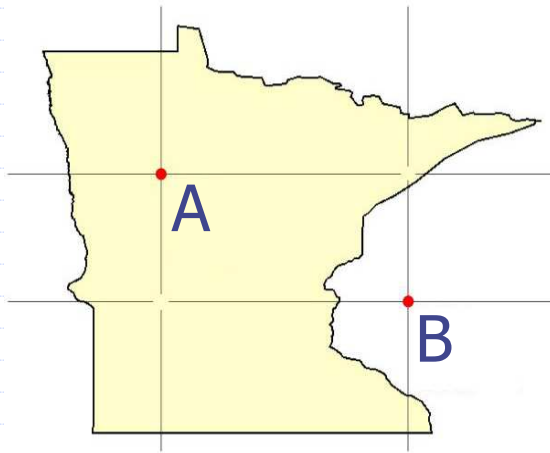
- ◆ Existing polygon program is CPU bound and is therefore a good candidate for parallel computing
- ◆ Existing nearest road program is I/O bound so with proper high performance computer architecture would be suitable for parallelization

## Geographic Information System Data

- ◆ Sources
  - US Census Bureau makes street level maps available for the United States
    - ◆ Approximately 8.8 gigabytes for US data
- ◆ Data Format
  - Shapefile is a standard vector format for GIS data
    - ◆ Free library called shapelib for reading Shapefiles using C



## Polygon Searching



Point A is inside polygon because it has an odd number of edge crossings

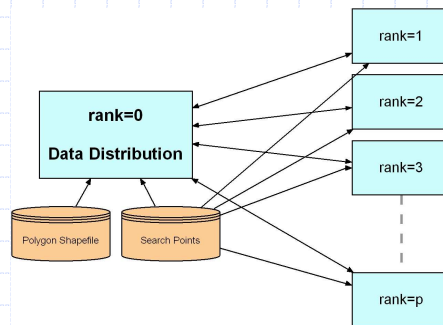
Point B is outside the polygon because there is an even number of edge crossings

## Polygon searching function

```
int pnpoly(int npol, int start, double *xp, double *yp, double x, double y) {  
    int i, j, c = 0;  
    for (i = start, j = npol-1; i < npol; j = i++) {  
        if ( ((yp[i]<=y) && (y<yp[j])) ||  
            ((yp[j]<=y) && (y<yp[i]))) &&  
            (x < (xp[j] - xp[i])*(y - yp[i]) / (yp[j] - yp[i]) + xp[i])) {  
                c = !c;  
            }  
        }  
    }  
    return c;  
}
```

## Decomposing problem to multiple nodes

- ◆ Embarrassingly parallel because there are many unrelated points to test



## Packaging of Data

- ◆ Shapefiles are read on each machine from Network File System (MPI not used)
  - /ptmp/jjj/gisdata in my case
- ◆ Search points are read on rank=0 node and distributed to rank>0 nodes using MPI

## Moving C structures with MPI

- ◆ C structures can be packaged using MPI derived types, however performance is bad
- ◆ I simply pass the structure as a sequence of binary bytes as specified by MPI\_BYTE
  - Would not work on a heterogeneous architecture cluster
  - sizeof(point) is word aligned – may be up to three extra bytes

```
typedef struct PT {  
    double x;  
    double y;  
    char source[11];  
} POINT;  
...  
MPI_Xsend( pt, sizeof(POINT), MPI_BYTE, ... )
```

## Blocking MPI Sending

- ◆ Use *for* loop to send search points to nodes one by one
  - mpi\_send(pt,sizeof(POINT),p,...)
- ◆ Wasteful because not all points take as long to find.
  - Each group of p-1 points takes as long as the longest point

## Buffered MPI sending

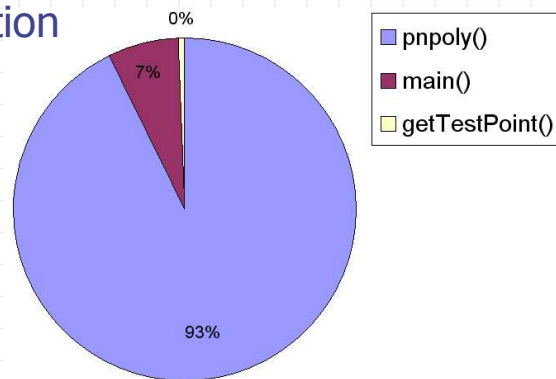
- ◆ Setup system buffer with `MPI_Buffer_attach`
  - Need to either make a buffer large enough to hold all data and overhead or keep track of buffer usage
- ◆ Now we can blast out  $n/(p-1)$  points to all nodes and be done
  - `MPI_Bsend(pt,sizeof(POINT),p,...)`

## Immediate Synchronous MPI sending

- ◆ We initially send points to  $p-1$  nodes and then send another point to each node as soon as it finishes the previous point
  - Now search time can be different and we will still keep all nodes utilized!
  - All nodes will finish at the same time but will have found a different number of points

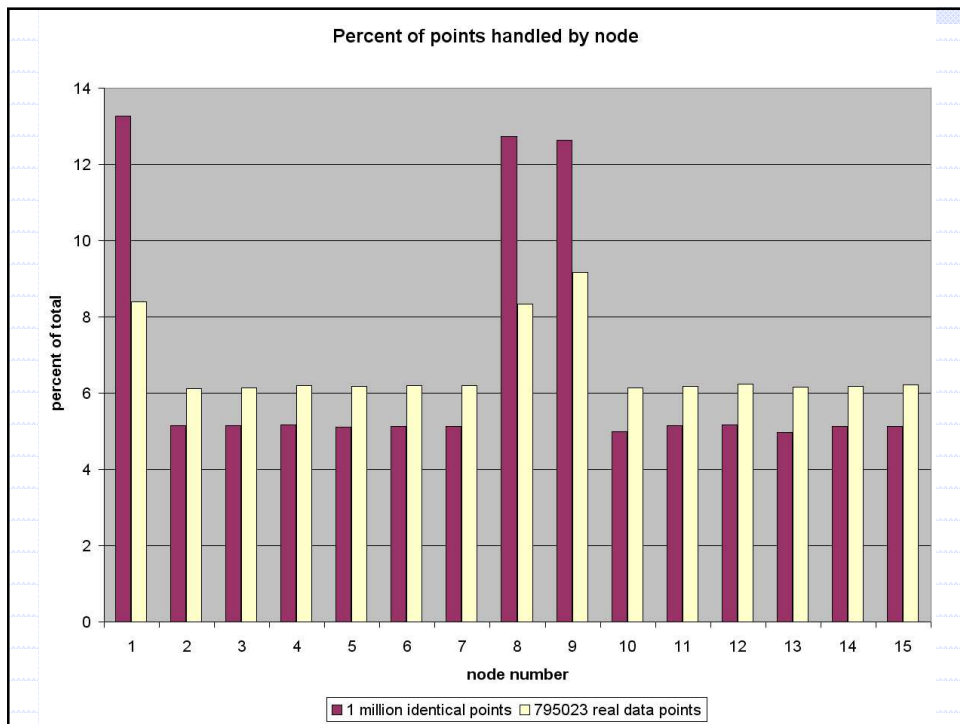
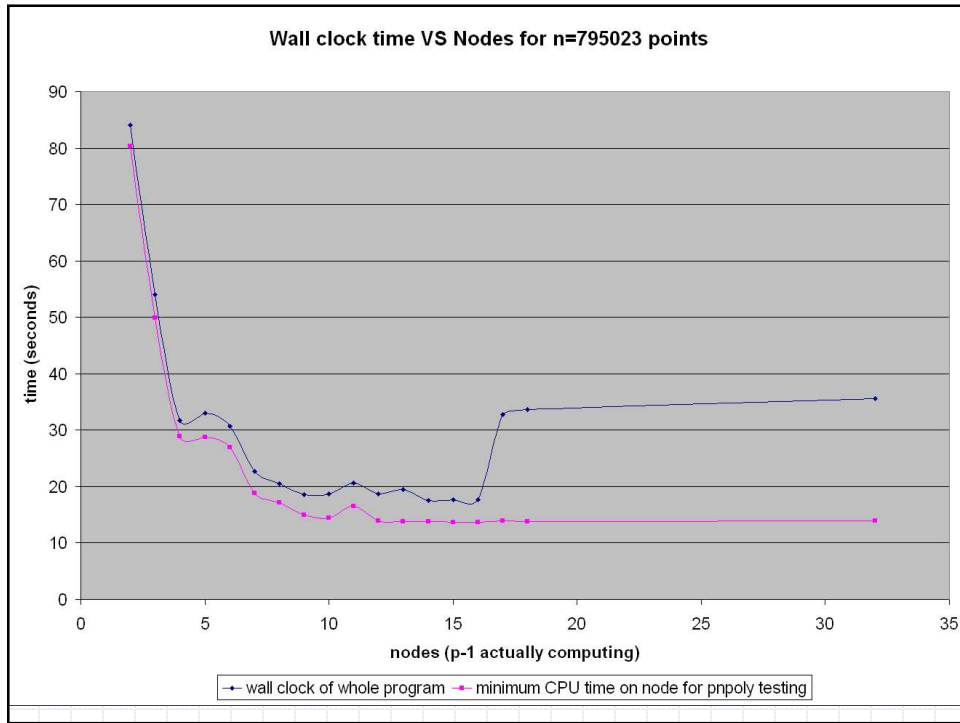
## Results

- ◆ Program was initially CPU bound and therefore should be a good candidate for parallelization

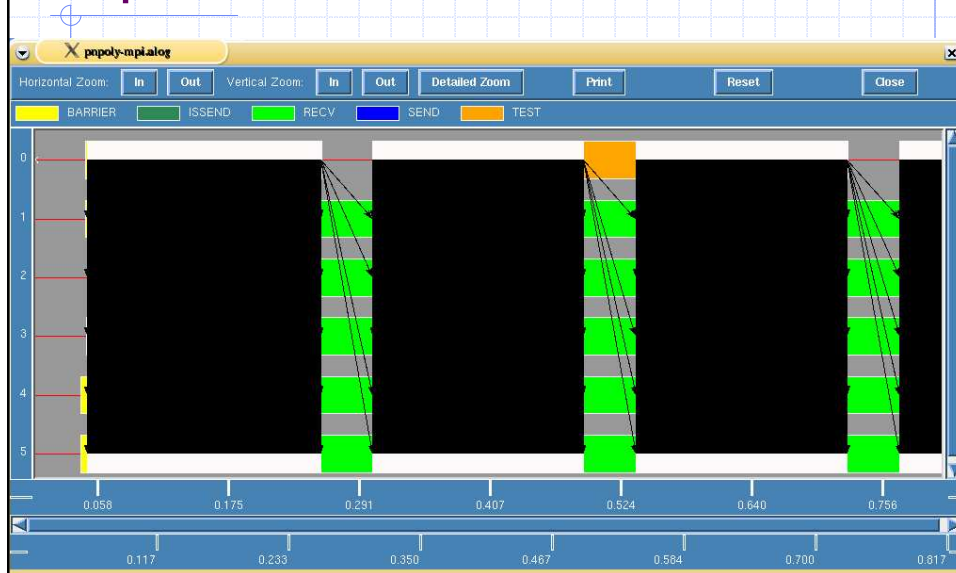


## hpc-class.iastate.edu performance

- ◆ Program was run on hpc-class using bmpi run.
- ◆ Time was measured both with a CPU clock timer on each node and a wall clock timer on rank=0 node
- ◆ No attempt was made flush cache or otherwise manipulate the system memory
  - This was so "real-world" performance could be seen



## Upshot Profile



## Cray T3E performance

- ◆ Program was not run on the Cray machine
  - Shapelib assumes 32-bit integers and would have require extensive work to make it support 64-bit integers.
    - ◆ typedef'ing int32 to be a short did not work
- ◆ Insufficient disk space to load GIS data

## Conclusions

- ◆ Overall success!
  - Program runs faster with more  $p$  until Amdahl's law catches up
- ◆ Peak performance with  $p=14$ , however the wall clock run time was only 16% better than with  $p=8$

## Future Plans

- ◆ Optimize serial portion of code (translating shape ID's into human readable labels)
- ◆ Write parallel / MPI version of nearest road searching program and test performance
  - Needs cluster with fast disk IO or needs an alternative form of distributing search point data

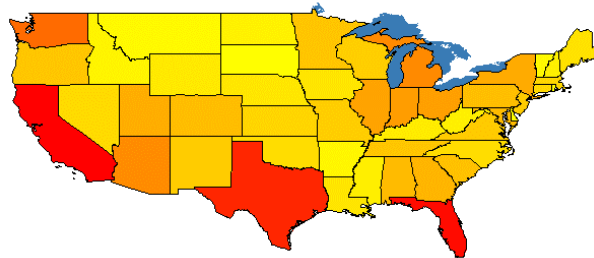
## Designing a machine for this application

- ◆ Store static GIS data on each node's local disk for fast access time
- ◆ Large main memory for operating system level caching of disk
- ◆ Fast floating point operations

## Applications

### ◆ Input

- "42.0142 -93.6513 KB0THN-7"



### ◆ Output (with world shapefile)

- "1|UNITED STATES|"



Any questions?

