# HPC Project: Shade Model Optimization

Troy Axthelm, Jared Baker, Matthew J. Brazell, Nels Frazier

May 10, 2015

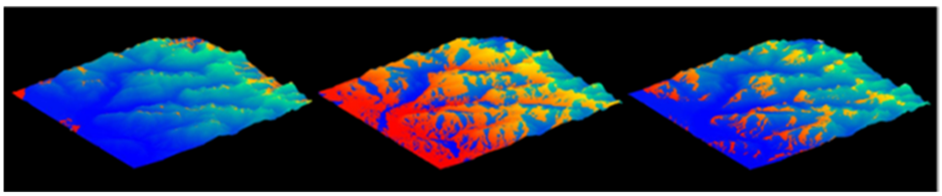

Figure 1:

## List of Figures



## 1 Introduction

This program is designed to model shade given elevation and location information for a selected land area. With this information a model of watershed can be created for primary use by CI-Water Project for predicting locations for forest fire remediation. The uploaded land area of interest can simply be downloaded off of GIS, the user can select a time interval over the course of a specific day of the year. The GPS coordinates of the land area as well as day of the year provide solar angle along with the topography of the land indicate if an area is shaded or not. Currently the program only indicates if an element of the land area is shaded or not shaded. No information of temperature or partial shading is provided and may be a future goal however currently this information alone is sufficient for CI-Water Project's needs.

The original program was created by Troy Axthelm and Jingyu Li funded by CI-Water Project. For more information and location of the open source code please see the link below:

https://sites.google.com/site/uwyoshademodel/

As a project for a graduate coarse: Designing and Building Applications for Extreme Scale Systems lead by instructor William Gropp and Professor Craig Douglas, our group intends make improvements in performance leading to faster simulations as well as making the program more flexible and easier to use for the user during runtime as well as post processing and visualization.

## 1.1 Initial Benchmarking

### 1.1.1 Parlib MPI

For performance just using MPI instead of Parlib removes a level of Libraries as well as making threading too abstract????.

### 1.1.2 MPI

An initial profiling of the code in parallel was used to determine trouble areas in the code where performance improvements could be made. The result of using the Allinea profiler showed that over 80% of the computing time was spent in a loop in main.c, the specific code for this problem area is shown below for convienence:

```
//Step along azimuth until off of grid or the observer point is determined to be shaded
while (tempX >= 0 && tempX <= (numCols-1) && tempY >= 0 && tempY <= (numRows - 1))
{

roundTempX = (int)round(tempX);
roundTempY = (int)round(tempY);

if(roundTempX != i ||  roundTempY != j)
{
tempSlope = (ourData[roundTempY][roundTempX].elevation - ourData[i][j].elevation)/sqrt(lenX*lenX + lenY*lenY);

if(solarAlt <= tempSlope)
{
ourData[i][j].shading[k] = 1;
break;
}
}

lenX += stepX;
lenY += stepY;

tempX += stepTempX;
tempY += stepTempY;

}
```

## 1.2 Coding Improvements

### 1.2.1 Algorithmic Improvements

introduce global constants reduce built in algebraic functions example code from landReader.c:

$$temp.latitude = latitude * 180.0/(4 * atan(1)); //convertthelatitudetodegrees$$

where $4 * atan(1) = 3.141592653589793 = \pi$

### 1.2.2 Linear Memory

To make improvements to the part of the code where the majority of the compute time is taken, by improving the way the array's "outData.elevation" is read and "ourData.shading" is written will make significant improvements to performance. To verify that this is an area that can be improved a simple performance model is created and compared with timing results of the actual code over this loop. This performance model is only to demonstrate an upper limit on performance.

```
//Step along azimuth until off of grid or the observer point is determined to be shaded
loop through numCols & numRows: n,m while (tempX >= 0 && tempX <= (numCols-1) && tempY >= 0 && tempY <= (numRows -
{

%n*m*(r+0.5*w) roundTempX = (int)round(tempX);
%n*m*(r+0.5*w) roundTempY = (int)round(tempY);

if(roundTempX != i ||  roundTempY != j)
{
n*m*((2+4*0.5)*r+w+2c) tempSlope = (ourData[roundTempY][roundTempX].elevation - ourData[i][j].elevation)/sqrt(lenX*

if(solarAlt <= tempSlope)
{
%n*m*(w) ourData[i][j].shading[k] = 1;
break;
}
}

%4*n*m*0.5*(r+w) lenX += stepX;
lenY += stepY;

tempX += stepTempX;
tempY += stepTempY;

}
```

total time $= n * m * (8r + 5w + 2c)$

### 1.2.3 Improvements for User

Add input file to modify time intervals and....

output file structure

## 1.3 Results

### 1.3.1 Final BenchMark

## 1.4 Conclusion

Table 1: Loop Timing Results

| - | Time for Loop(s) | Rate [MB/s] |
|---|---|---|
| min | 4E-06 | |
| max | 1.08E-04 | |

# A Original Code

Listing 1: ./orig_code/main.c

```
/*
/ Troy Axthelm and Jingyu Li
/ Land shade model program
/ 12 December 2013
/
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PAR_MPI 1
#include "parlib.h"

#include "landStruct.h"
#include "landReader.h"
#include "sunDeclination.h"
#include "timeDifference.h"
#include "localHourAngle.h"
#include "solarAltitude.h"
#include "azimuth.h"

int main(int argc, char** argv)
{
        //Set to the day of shading calculation
        //January 1  = 0; December 31 = 364
        int Day = 20;

        //Interval of time to have shading information for
        //Must change landStruct shade array size to 86400/timeInterval
        int timeInterval = 300;

        //Set to the file name to read elevation data from
        char* fileName = "DEM.asc";

        //Variables will be set at runtime
        LandData** ourData;
        int numRows=0;
        int numCols=0;
        double sunDeclin  = 0.0;
        double localHrAngle = 0.0;
        double azi = 0.0;
        double solarAlt = 0.0;
        double timeDif = 0.0;

        //pi
        const double PI = 4*atan(1);

        //index variables
        int i,j,k;

        double temp;

        //ourData is now populated with the necessary information
        //from the inpt file.
        ourData = extractData(fileName, &numCols, &numRows);

        //Calculate the sun declination for the given day.
        sunDeclination(&sunDeclin,Day);

        double stepX;
        double stepY;
        double lenX;
        double lenY;
```

```
        double tempX;
        double tempY;
        int roundTempX;
        int roundTempY;
        double stepTempX;
        double stepTempY;
        double tempSlope;
        double ourStep;
        double darkAngle;

        int rank;
        int psize;
        par_start(argc, argv, &psize, &rank);

        //Not including lower portions of triangular mesh
        for(k = rank; k < 288; k += psize)
        {
                //Set the name of the current file to write to
                char fileName[50];
                sprintf(fileName, "shadingPlot_%d.m", k);
                FILE *thisFile;
                thisFile = fopen(fileName, "w");
                fprintf(thisFile, "a = [ ");

                for( i = 0; i < numRows; i++)
                {
                        for (j = 0; j < numCols; j++)
                        {

                                double currentSizeX = ourData[i][j].sizeX;
                                double currentSizeY = ourData[i][j].sizeY;
                                ourStep = currentSizeX;


                                //Calculate the dark angle
                                darkAngle = acos(-tan(ourData[i][j].latitude) * tan(sunDeclin));

                                //For each thirty minute azimuth, calculate the horizon(in this day)

                                //-----
                                timeDifference(&timeDif, ourData[i][j].thetaS, ourData[i][j].thetaL);

                                localHourAngle(&localHrAngle,(double)(k * timeInterval), timeDif, 0.0, timeInterval);

                                if (((localHrAngle < (darkAngle + PI - (15.0*PI/180))) && k < (86400/timeInterval/2))
|| ((localHrAngle > (PI-darkAngle + (30.0*PI/180.0)) && k >= (86400/timeInterval/2))))
                                {
                                        ourData[i][j].shading[k] = 1.0;
                                }
                                else
                                {

                                        //LATITUDE NEEDS TO BE IN RADIANS!!!!!!!!
                                        solarAltitude(&solarAlt, sunDeclin, ((ourData[i][j].latitude)*PI/180), localHrAngl

                                        azimuth(&azi, solarAlt, ourData[i][j].latitude, sunDeclin, k * timeInterval);

                                        stepX = ourStep*-sin(azi);
                                        stepY = ourStep*cos(azi);

                                        lenX = 0.0;
                                        lenY = 0.0;

                                        stepTempX = stepX / currentSizeX;
                                        stepTempY = stepY / currentSizeY;

                                        lenX += stepX;
                                        lenY += stepY;

                                        tempX = lenX/currentSizeX + (double)j;
                                        tempY = lenY/currentSizeY + (double)i;

                                        ourData[i][j].shading[k] = 0;

                                        solarAlt = tan(solarAlt);

                                        //Step along azimuth until off of grid or the observer point is determind to be sh
                                        while (tempX >= 0 && tempX <= (numCols-1) && tempY >= 0 && tempY <= (numRows - 1))
                                        {
                                                roundTempX = (int)round(tempX);
                                                roundTempY = (int)round(tempY);
```

```
                                                if(roundTempX != i ||  roundTempY != j)
                                                {
                                                        tempSlope = (ourData[roundTempY][roundTempX].elevation - ourData[i

                                                        if(solarAlt <= tempSlope)
                                                        {
                                                                ourData[i][j].shading[k] = 1;
                                                                break;
                                                        }
                                                }

                                                lenX += stepX;
                                                lenY += stepY;

                                                tempX += stepTempX;
                                                tempY += stepTempY;

                                        }

                                }
                                //Print all of shade data to files in .m format
                                fprintf(thisFile, "%d␣", ourData[i][j].shading[k]);

                        }

                        fprintf(thisFile, ";␣");
                }
/*
                int timeHours = (int)floor(k*timeInterval/3600);
                int timeMinutes = (int)((k*timeInterval)/60);
                timeMinutes = timeMinutes%60;
                char time[5];

                sprintf(time,"%2d:%02d", timeHours, timeMinutes);
*/

                fprintf(thisFile, "];");

                fclose(thisFile);

                printf("%d␣remaining\n", 288-k);
        }

        //fprintf(thisFile,"\nh = surf(a)\n m = [0:9.17:2053*9.17];\nn = [1965*9.17:-9.17:0];\nh = surf(m, n, b, a);\nset(
Time: %s');\naxis equal\naxis vis3d\nsaveas(h, 'shadePlot_%02d.jpeg', 'jpeg' );", timeInterval/60, time, k);



        //Release all memory allocation
        for(i = 0; i < numRows; i++)
        {
                free(ourData[i]);
        }
        free(ourData);
        par_end( );
        return 0;
}
```

Listing 2: ./orig_code/azimuth.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "tilt.h"

void azimuth(double *azimuth, double solarAlt, double latitude, double declination, int time){

        *azimuth = ((sin(solarAlt) * sin((latitude/180.0)*(4*atan(1)))) - sin(declination))/(cos(solarAlt) * cos((latitude

        *azimuth = acos(*azimuth);

        if (time <= 43200)
                *azimuth = -*azimuth;
}
```

Listing 3: ./orig_code/hourAngleTest.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "localHourAngle.h"

int main()
{
        int i = 0;
        double t =0.0;

        for( i = 0; i < 86400; i += 3600)
        {
                localHourAngle(&t, (double)i, 0.0, 0.0, 0);
                printf("%d␣␣␣%f\n", i, (t*180)/(4*atan(1)));
        }
}
```

Listing 4: ./orig_code/landReader.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "landStruct.h"
#include "tilt.h"

LandData** extractData(char *inFileName, int* num_Cols, int* num_Rows)
{
        FILE *inFile;
        int numCols = 0;
        int numRows = 0;
        float xllcorner = 0.0;
        float yllcorner = 0.0;
        float cellsize = 0.0;
        float elevation = 0.0;


        //potential variables
        double thetaS = -106.0; //standard meridian longitude

        int i,j;

        //open the file to be read
        inFile = fopen(inFileName, "r");

        //make sure the file was opened
        if(inFile == NULL)
        {
                printf("Error,␣Cannot␣open␣file:␣%s\n", inFileName);
                exit(1);
        }

        //Read in variables from file
        fscanf(inFile, "%*s␣%d", &numCols);
        fscanf(inFile, "%*s␣%d", &numRows);
        fscanf(inFile, "%*s␣%f", &xllcorner);
        fscanf(inFile, "%*s␣%f", &yllcorner);
        fscanf(inFile, "%*s␣%f", &cellsize);

        LandData** structMat = (LandData**)malloc(sizeof(LandData*)*(numRows));

        for(i = 0; i < numRows; i++)
        {
                structMat[i] = (LandData*)malloc(sizeof(LandData)*(numCols));
                for(j = 0; j < numCols; j++)
                {
                        fscanf(inFile, "%f", &elevation);

                        LandData temp;
                        temp.sizeX = cellsize;
                        temp.sizeY = cellsize;
                        temp.elevation = elevation;

                        //calculation of latitude
                        double latitude = (yllcorner + (numRows-(i+1))*cellsize)/6378100.0;
                        //calculation of longitude
                        temp.latitude = latitude*180.0/(4*atan(1));     //convert the latitude to degrees
                        temp.thetaL = -106 + (xllcorner - ((j/2)-1))/(6378100.0 * cos(latitude));
                        temp.thetaS = thetaS;
```

```
                        structMat[i][j] = temp;
                }
        }

        //Fill the secondary cells of matrix, used for triangular mesh
        for(i = 0; i < (numRows-1); i++)
        {
                for(j = 0; j < (numCols-1); j++)
                {
                        //LandData temp;
                        LandData temp1;
                        LandData temp2;
                        LandData temp3;

                        //temp.elevation = structMat[i][j].elevation;
                        //temp.latitude = structMat[i][j].latitude;
                        //temp.thetaL = structMat[i][j].thetaL;
                        //temp.thetaS = thetaS;
                        //temp.sizeX = cellsize;
                        //temp.sizeY = cellsize;

                        temp1 = structMat[i][j+1];
                        temp2 = structMat[i+1][j];
                        temp3 = structMat[i][j];

                        //calculate the angles of each struct
                        //tilt(&temp, &temp1, &temp2, 0);
                        tilt(&temp3, &temp1, &temp2, 1);


                        //if(i == 0 && j-1 == 0)
                        structMat[i][j] = temp3;
                        //structMat[i][j] = temp;
                }
        }

        fclose(inFile);

        *num_Rows = numRows;
        *num_Cols = numCols;

        return structMat;
}
```

Listing 5: ./orig_code/localHourAngle.c

```
/*
REU HPC Summer 2013
Program Purpose: Linear algebra Library.
Created by: Noll Roberts
On: 2013.06.20
Modified by:
On:
*/

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"localHourAngle.h"


/************ FUNCTIONS ***************************/
// see project.h
void localHourAngle( double *tau, double Ts, double deltaT1, double deltaT2, int timeStep)
{
   if(Ts<43200.0)
        {
        *tau= ((Ts/3600)+12-deltaT1+deltaT2)*15*(4*atan(1)/180);
   }
        else
   {
                *tau= ((Ts/3600)-12-deltaT1+deltaT2)*15*(4*atan(1)/180);
        }

   //Ts+=timeStep;
   return;
}
```

Listing 6: ./orig_code/parlib_mpi.c

```
// ***************************************************************************
//
// Define common data for MPI
//
//      Data types
//      Reduce operations
//
// ***************************************************************************

#include <stdlib.h>
#include <mpi.h>

#include "parlib.h"
//#include "parlib_mpi.h"

MPI_Datatype
     par_datatypes[] =  { MPI_INT, MPI_FLOAT,
                          MPI_DOUBLE, MPI_CHAR };

MPI_Op
     par_reduce_ops[] = { MPI_OP_NULL, MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD,
                          MPI_LAND, MPI_BAND, MPI_LOR, MPI_BOR, MPI_LXOR,
                          MPI_BXOR, MPI_MINLOC };



// ***************************************************************************

void par_start( int argc, char **argv, int* psize, int* prank ) {

    MPI_Init( &argc, &argv );
    *psize = par_size( );
    *prank = par_rank( );

}


// ***************************************************************************

void par_end( ) {

    MPI_Finalize();
    exit(0);

}


// ***************************************************************************

int par_rank( ) {

    int my_rank;

    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    return my_rank;
}


// ***************************************************************************

int par_size( ) {

    int size;

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    return size;

}


// ***************************************************************************

double par_walltime( ) {

    return MPI_Wtime();

}
```

```
// ***************************************************************************

int par_send( void *buff, int size, int type, int dest, int tag ) {

    return MPI_Send( buff, size, par_datatypes[type], dest,
                     tag, MPI_COMM_WORLD );

}

// ***************************************************************************

int par_recv( void* buff, int size, int type, int source, int tag ) {

    MPI_Status status;

    return MPI_Recv( buff, size, par_datatypes[type], source,
                     tag, MPI_COMM_WORLD, &status);

}

// ***************************************************************************

int par_isend( void* buff, int size, int type, int dest, int tag,
               par_request *request ) {

    return MPI_Isend( buff, size, par_datatypes[type], dest,
                      tag, MPI_COMM_WORLD, request);

}

// ***************************************************************************

int par_irecv( void* buff, int size, int type, int source, int tag,
               par_request *request ) {

    return MPI_Irecv( buff, size, par_datatypes[type], source,
                      tag, MPI_COMM_WORLD, request);

}

// ***************************************************************************

int par_wait( par_request* request, par_status* status ) {

    return MPI_Wait( request, status );

}

// ***************************************************************************

int par_waitall( int count, par_request* request, par_status* status ) {


    return MPI_Waitall( count, request, status );

}

// ***************************************************************************

int par_waitany( int count, par_request* requests, int* index,
                 par_status* status ) {

    return MPI_Waitany( count, requests, index, status );

}

// ***************************************************************************

int par_bcast( void* buff, int size, int type, int source ) {

    return MPI_Bcast( buff, size, par_datatypes[type], source,
                      MPI_COMM_WORLD );

}
```

```
// ***************************************************************************

int par_reduce( void* inbuff, void* outbuff, int count, int datatype,
                int reduce_op, int root ) {

    return MPI_Reduce( inbuff, outbuff, count,
                       par_datatypes[datatype],
                       par_reduce_ops[reduce_op], root,
                       MPI_COMM_WORLD);

}


// ***************************************************************************

void par_barrier( ) {

    MPI_Barrier( MPI_COMM_WORLD );

}
```

Listing 7: ./orig_code/parlib.c

```
// ***************************************************************************
//
//
//
//
//
// ***************************************************************************

#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

#include "parlib.h"

// ***************************************************************************
//
// Include the implementation that is specific to the middleware, which
// currently is/are for the following:
//
//     MPI
//
// See Makefile for definitions, which can be overriden on the make command
// line.
//
// ***************************************************************************

// The MPI version

#if PAR_MPI == 1
#include "parlib_mpi.c"
#endif
```

Listing 8: ./orig_code/solarAltitude.c

```
#include <math.h>

#include "solarAltitude.h"

void solarAltitude(double * solarAlt, double sunDeclination, double latitude, double localHourAngle)
{
        //Equation for solar altitude as found on page 25 of reading (Planetary Motions and the Distribution of Radiation,
        *solarAlt = asin(sin(sunDeclination)*sin(latitude) + cos(sunDeclination)*cos(latitude)*cos(localHourAngle));

}
```

Listing 9: ./orig_code/solarAltTest.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```c
#include "solarAltitude.h"
#include "localHourAngle.h"
#include "sunDeclination.h"
#include "timeDifference.h"

int main()
{
        double lat = 40.100178;
        double longi = -105.99758;
        double stdMer = -106.0;
        int day = 172;
        double time = 43200;
        double sunDec = 0.0;
        double timeDiff = 0.0;
        double localHrAng = 0.0;
        double solarAlt = 0.0;

        sunDeclination(&sunDec, day);
        timeDifference(&timeDiff, stdMer, longi);
        localHourAngle(&localHrAng, time, timeDiff, 0.0, 0);

        printf("vals␣are:␣%f,␣%f\n", sunDec, localHrAng);

        solarAltitude(&solarAlt, sunDec, lat*(4*atan(1)/180.0), localHrAng);

        printf("solar␣alt␣is:␣%f\n", solarAlt*180/(4*atan(1)));

        return 0;
}
```

Listing 10: ./orig_code/sunDeclination.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "sunDeclination.h"



void sunDeclination(double *delta, int day){
        *delta = ((23.45*4*atan(1))/180)*cos(((2*4*atan(1))/365)*(172-day));

        return;

}
```

Listing 11: ./orig_code/test.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "timeDifference.h"

int main(int argc, char** argv)
{
        double delta;
        int day;

        printf("SunDeclination,\tDay\n");
        for( day = 0; day < 365; day++)
        {
                sunDeclination(&delta, day);
                delta = (delta*180)/(4*atan(1));
                printf("%f\t%d\n", delta, day);
        }

        return 0;
}
```

Listing 12: ./orig_code/tilt.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "landStruct.h"
#include "tilt.h"

void tilt(LandData *mainPlot, LandData *xNeighbor, LandData *yNeighbor, int upperLower){
	if (upperLower == 1){
		mainPlot->angleX = atan((xNeighbor->elevation - mainPlot->elevation)/mainPlot->sizeX);
		mainPlot->angleY = atan((mainPlot->elevation - yNeighbor->elevation)/mainPlot->sizeY);
	} else {
		mainPlot->angleX = atan((mainPlot->elevation - xNeighbor->elevation)/mainPlot->sizeX);
		mainPlot->angleY = atan((yNeighbor->elevation - mainPlot->elevation)/mainPlot->sizeY);
	}
}
```

Listing 13: ./orig_code/timeDifference.c

```
//Bill Matonte
//6.27.13
//REU SUMMER 2013
//timedifference.c
//This function finds the offset for the time of day for the solar cumilative
//radiation formula.
#include "timeDifference.h"
#include <stdio.h>
//The Variables ThetaS and ThetaL with the pointer to the deltaT
//see timedifference.h
//thetaS = standard time meridian longitde of time zone
//thetaL = local longitude

void timeDifference(double* deltaT1, double thetaS,double thetaL)
{
	*deltaT1=-( thetaS-thetaL)/15.0;
	return;
}
```