Assignment 2

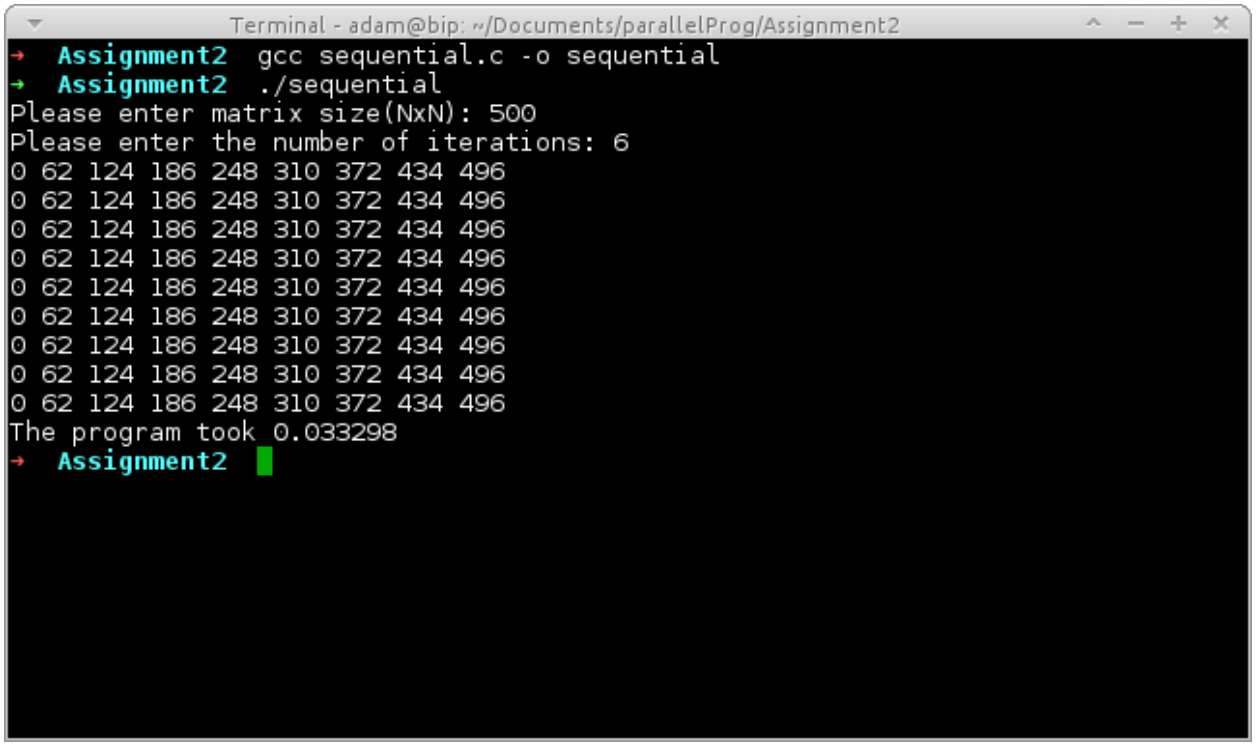
by Adam Van Hine

Task 1

```
#include <stdio.h>
#include <time.h>

void main(int arc, char* argv) {
    clock_t begin,end;
    double time_spent;
    int iteration, i, j, current, next, N, T;

    // Get the size of the matrix
    int valid = 0;
    while(!valid) {
        printf("Please enter matrix size(NxN): ");
        scanf("%d", &N);
        if(N) {
            valid = 1;
        }
    }

    // Get the number of iterations
    valid = 0;
    while(!valid) {
        printf("Please enter the number of iterations: ");
        scanf("%d", &T);
        if(N) {
            valid = 1;
        }
    }

    //initaialize the array
    double h[2][N][N];

    // Initialize both arrays to all 0s
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            h[0][i][j] = 0;
            h[1][i][j] = 0;
        }
```

```
    }


    // Set all the walls to 20C degrees
    for(i = 0; i < N; i++) {
        h[0][0][i] = 20.0;
        h[0][i][0] = 20.0;
        h[0][N-1][i] = 20.0;
        h[0][i][N-1] = 20.0;
    }

    // Set the starting point and the ending point of the fireplace
    double fp_start, fp_end;
    fp_start = 0.3 * N;
    fp_end = (0.7 * N);

    // Initialize the values of the location of the fireplacee to 100C
    for(i = fp_start; i < fp_end; i++) {
        h[0][0][i] = 100.0;
    }

    // Get the start time of the program
    begin = clock();

    // The actual calculation of the temperature of the room.
    current = 0;
    next = 1;
    for (iteration = 0; iteration < T; iteration++) {
        for( i = 1; i < N-1; i++) {
            for( j = 1; j < N-1; j++) {
                h[next][i][j] = 0.25 * (h[current][i-1][j] + h[current][i+1][j]+ h[current][i][j-1] +
h[current][i][j+1]);
            }
        }
        current = next;
        next = (current + 1) % 2;
    }

    // Get the end time
    end = clock();
    // Calculate the time spent
    time_spent = (double)(end-begin) /CLOCKS_PER_SEC;
```

```
    // Print an 8x8 array the N/8 x N/8 positions of the room.
    for(i = 0; i < N; i+= N/8) {
        for(j = 0; j < N; j+= N/8) {
            printf("%.2f ", h[0][i][j]);
        }
        printf("\n");
    }
    printf("\n");

    // Print the time in seconds that the program took to run
    printf("The program took %f seconds to run.\n", time_spent);
}
```



Task 2

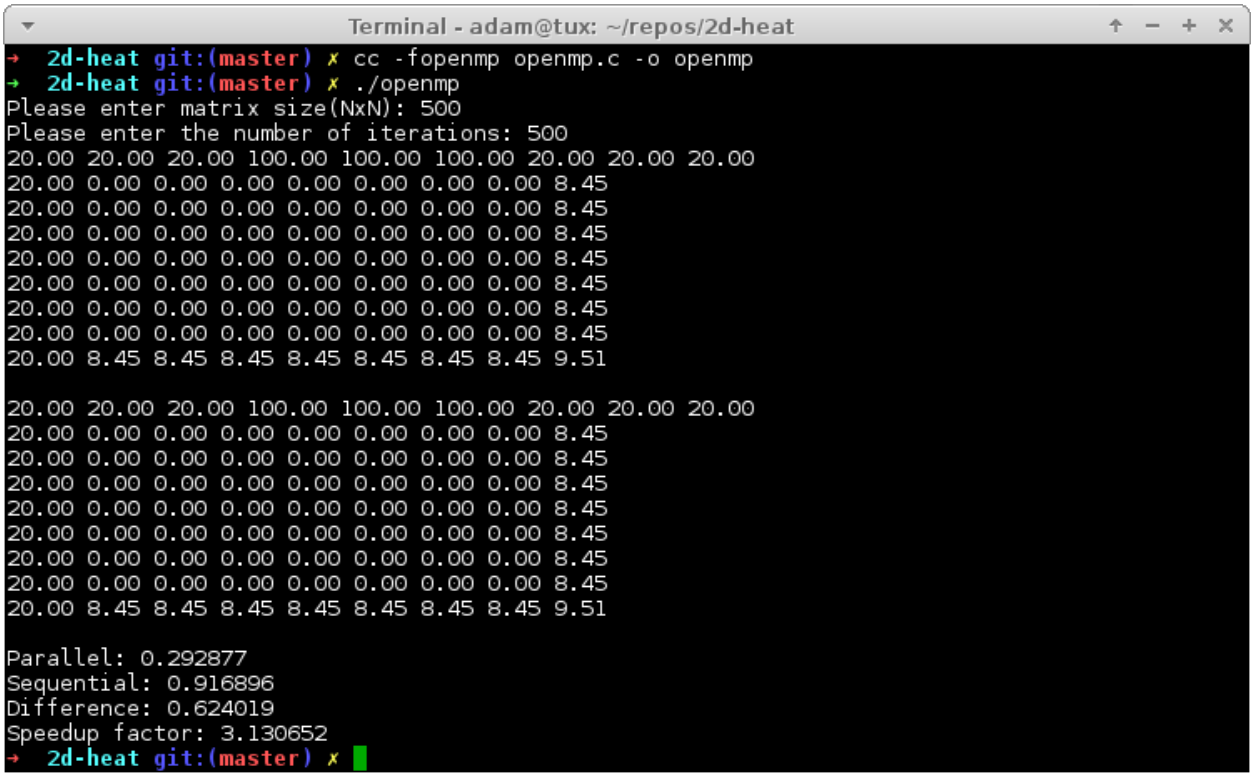
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void getInput(int*, int*);
```

```
void main(int arc, char* argv) {
    omp_set_num_threads(4);
    double begin,end;
    double sequential_time, parallel_time;
    int iteration, i, j, current, next, N, T;

    getInput(&N, &T);

    //initaialize the array
    double h[2][N][N];
    double g[2][N][N];

    // Initialize both arrays to all 0s
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            h[0][i][j] = 0;
            h[1][i][j] = 0;
            g[0][i][j] = 0;
            g[1][i][j] = 0;
        }
    }

    // Set all the walls to 20C degrees
    for(i = 0; i < N; i++) {
        h[0][0][i] = 20.0;
        h[0][i][0] = 20.0;
        h[0][N-1][i] = 20.0;
        h[0][i][N-1] = 20.0;
        g[0][0][i] = 20.0;
        g[0][i][0] = 20.0;
        g[0][N-1][i] = 20.0;
        g[0][i][N-1] = 20.0;
    }

    // Set the starting point and the ending point of the fireplace
    double fp_start, fp_end;
    fp_start = 0.3 * N;
    fp_end = (0.7 * N);

    // Initialize the values of the location of the fireplacee to 100C
    for(i = fp_start; i < fp_end; i++) {
        h[0][0][i] = 100.0;
        g[0][0][i] = 100.0;
```

```
    }

    // Get the start time of the program
    begin = omp_get_wtime();

    /**
     * Begin the sequential calculation section
     */
    current = 0;
    next = 1;
    for (iteration = 0; iteration < T; iteration++) {
        for( i = 1; i < N-1; i++) {
            for( j = 1; j < N-1; j++) {
                h[next][i][j] = 0.25 * (h[current][i-1][j] + h[current][i+1][j] + h[current][i][j-1] +
h[current][i][j+1]);
            }
        }
        current = next;
        next = (current + 1) % 2;
    }
    // Calculate time spent
    end = omp_get_wtime();
    sequential_time = end-begin;

    // Print an 8x8 array the N/8 x N/8 positions of the room.
    for(i = 0; i < N; i+= N/8) {
        for(j = 0; j < N; j+= N/8) {
            printf("%.2f ", h[current][i][j]);
        }
        printf("\n");
    }
    printf("\n");


    /**
     * Begin Parallel section
     */
    begin = omp_get_wtime();
    for (iteration = 0; iteration < T; iteration++) {
        #pragma omp parallel for private(i, j)
        for( i = 1; i < N-1; i++) {
            for( j = 1; j < N-1; j++) {
```

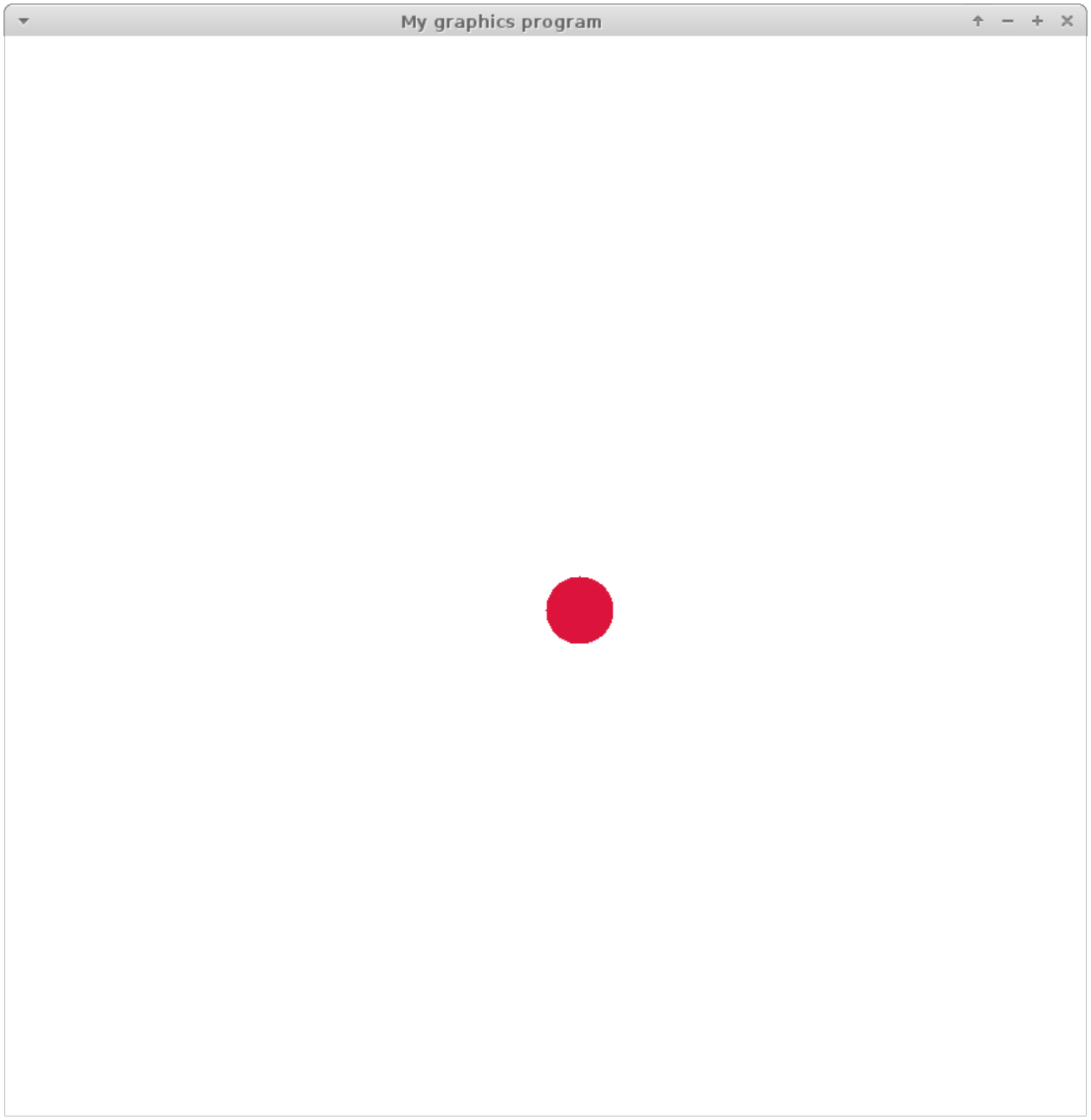
```
                g[next][i][j] = 0.25 * (g[current][i-1][j] + g[current][i+1][j]+ g[current][i][j-1] +
g[current][i][j+1]);
            }
        }
        current = next;
        next = (current + 1) % 2;
    }
    // Calculate the time spent
    end = omp_get_wtime();
    parallel_time = end - begin;

    // Print an 8x8 array the N/8 x N/8 positions of the room.
    for(i = 0; i < N; i+= N/8) {
        for(j = 0; j < N; j+= N/8) {
            printf("%.2f ", g[0][i][j]);
        }
        printf("\n");
    }
    printf("\n");

    // Check to see if the two arrays match
    for(i = 0; i < N; i++) {
        for(j = 0; j < N; j++) {
            if(g[0][i][j] != h[current][i][j]) {
                printf("2 arrays do not match\n");
                exit(1);
            }
        }
    }

    // Print the time in seconds that the program took to run
    printf("Parallel: %f\nSequential: %f\nDifference: %f\n", parallel_time, sequential_time,
sequential_time - parallel_time);
    printf("Speedup factor: %f\n", sequential_time/parallel_time);

}

/**
 * Function to get the input from the user
 * Params: int* N - A pointer to the number 'N' of the NxN matrix
 *         int* T - A pointer to the number of iterations
 */
void getInput(int* N, int* T) {
```

```
    // Get the size of the matrix
    int valid = 0;
    while(!valid) {
        printf("Please enter matrix size(NxN): ");
        scanf("%d", N);
        if(N) {
            valid = 1;
        }
    }

    // Get the number of iterations
    valid = 0;
    while(!valid) {
        printf("Please enter the number of iterations: ");
        scanf("%d", T);
        if(N) {
            valid = 1;
        }
    }
}
```



Task 3

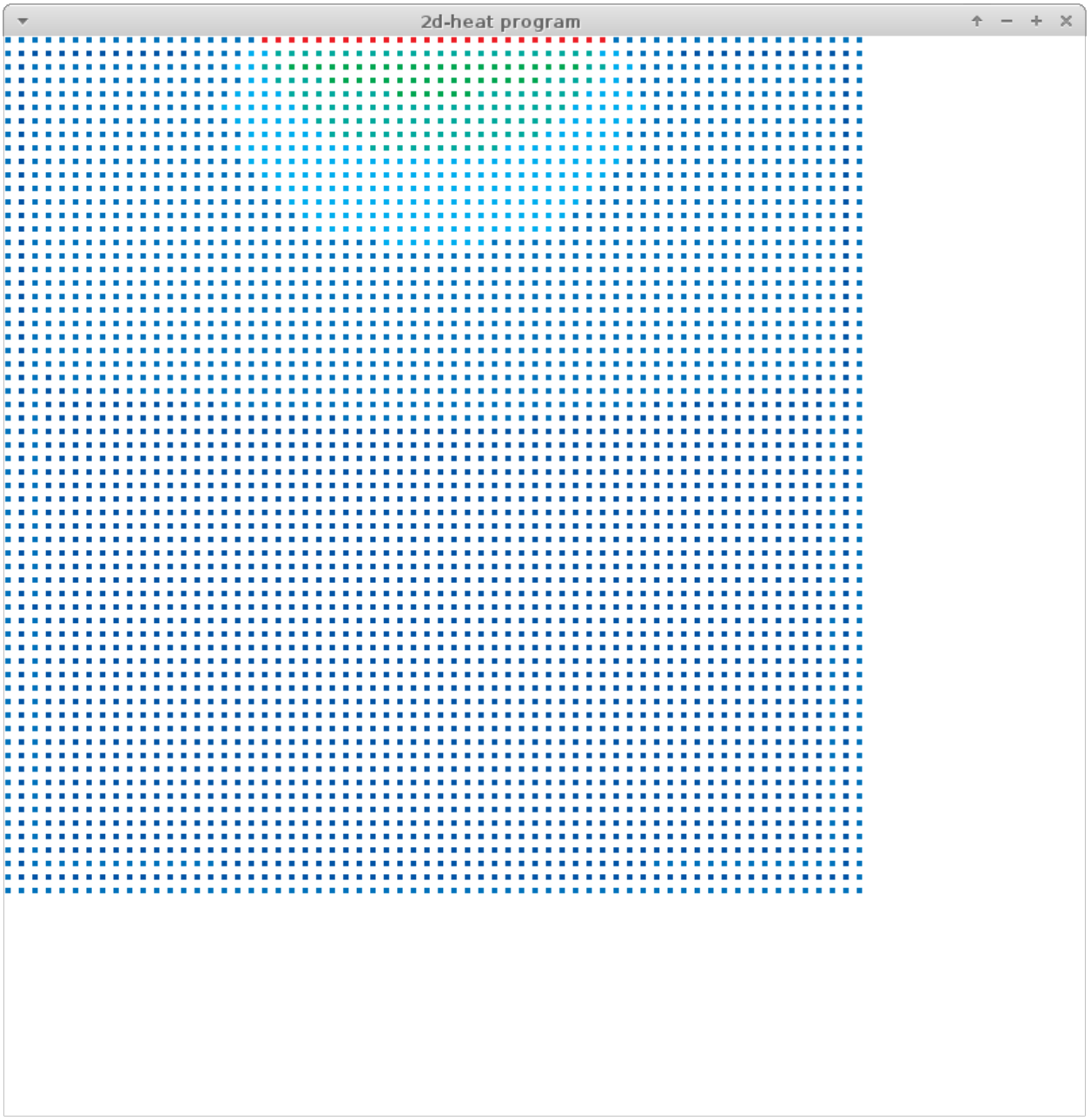
a)



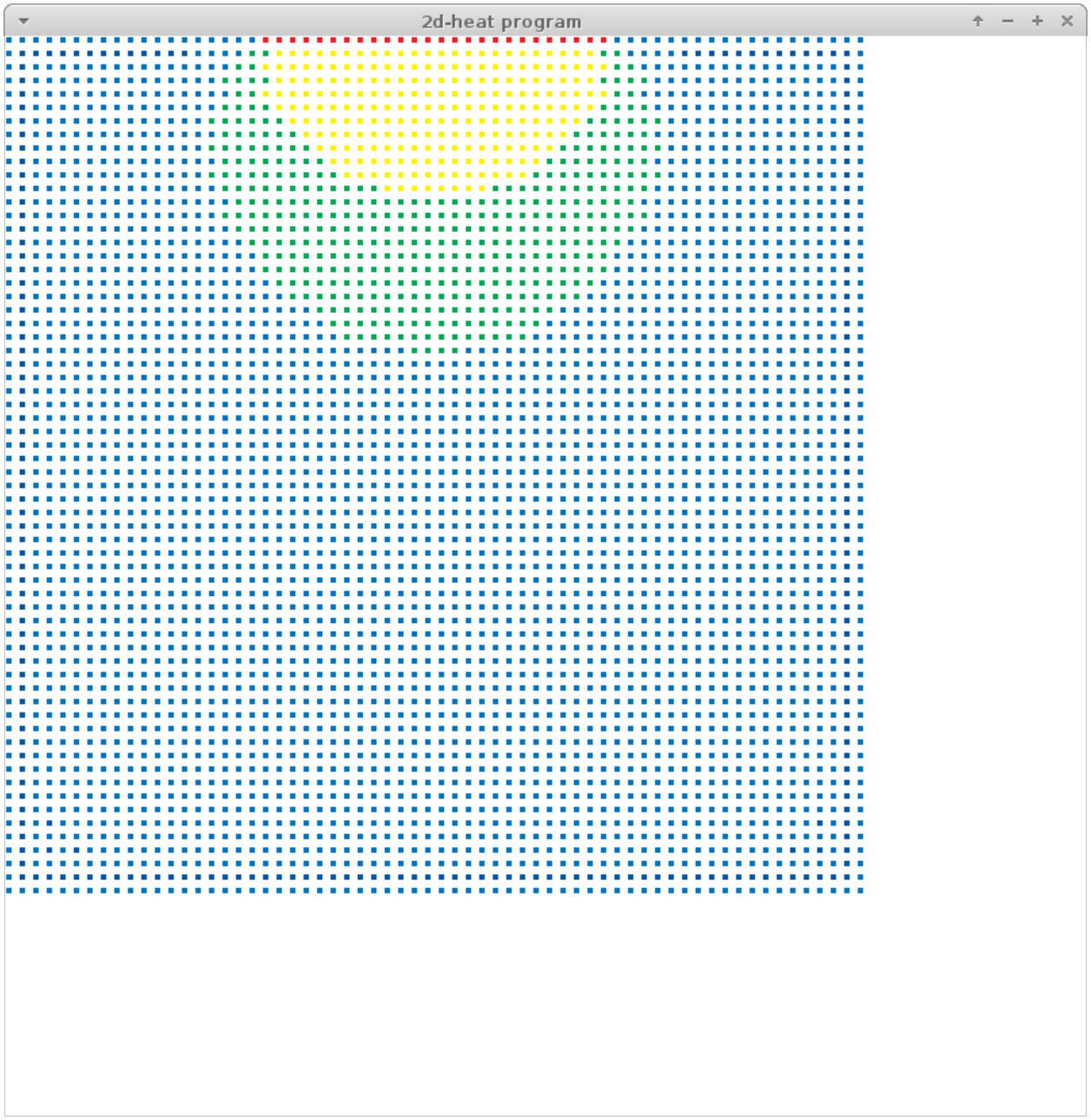
In this sample program we see that most of the code is used for the setup of the window. We see functions that set the height and width of the screen, what color the background and foreground should be, as well as setting the name for the window seen in this screen capture. The final pieces of code are the parts that actually render the red circle on the canvas. We see

that given the command XFillArc() we can draw a circle at the specified positions. This will come in handy when mapping the heat dissipation.

b)



Here is an image of the output created when I run the sequential heat distribution program with X11 code included. This is as requested with the color changing per 10C degrees. For this code I used the XFillArc to draw a 5x5 circle of each point in the room. I was not very happy with the looks of this image so I modified the colors to be a bit more reflective of the heat distribution. This is a second image of the same program with different color assignments.



I believe it is much easier to see what is actually happening this way.

Here is the code I used to render this image

Sequential X11 code

```
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

```
#include <X11/Xos.h>
#define X_RESN 800
#define Y_RESN 800

void getInput(int*, int*);

int main(int arc, char* argv) {
    clock_t begin,end;
    double time_spent;
    int iteration, i, j, current, next, N, T;

    getInput(&N, &T);

    //initaialize the array
    double h[2][N][N];

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            h[0][i][j] = 0;
            h[1][i][j] = 0;
        }
    }
    // Set all the walls to 20C degrees
    for(i = 0; i < N; i++) {
        h[0][0][i] = 20.0;
        h[0][i][0] = 20.0;
        h[0][N-1][i] = 20.0;
        h[0][i][N-1] = 20.0;
    }

    // Set the starting point and the ending point of the fireplace
    double fp_start, fp_end;
    fp_start = 0.3 * N;
    fp_end = (0.7 * N);

    // Initialize the values of the location of the fireplacee to 100C
    for(i = fp_start; i < fp_end; i++) {
        h[0][0][i] = 100.0;
    }

    // Get the start time of the program
    begin = clock();
```

```
    // The actual calculation of the temperature of the room.
    current = 0;
    next = 1;
    for (iteration = 0; iteration < T; iteration++) {
        for( i = 1; i < N-1; i++) {
            for( j = 1; j < N-1; j++) {
                h[next][i][j] = 0.25 * (h[current][i-1][j] + h[current][i+1][j]+ h[current][i][j-1] +
h[current][i][j+1]);
            }
        }
        current = next;
        next = (current + 1) % 2;
    }

    // Get the end time
    end = clock();
    // Calculate the time spent
    time_spent = (double)(end-begin) /CLOCKS_PER_SEC;

    // Print an 8x8 array the N/8 x N/8 positions of the room.
    for(i = 0; i < N; i+= N/8) {
        for(j = 0; j < N; j+= N/8) {
            printf("%.2f ", h[0][i][j]);
        }
        printf("\n");
    }
    printf("\n");

    // Print the time in seconds that the program took to run
    printf("The program took %f seconds to run.\n", time_spent);

    Window win;
    unsigned int width, height,
    win_x, win_y,
    border_width,
    display_width, display_height,
    screen;
    char *window_name = "2d-heat program", *display_name = NULL;
    GC gc;
    unsigned long valuemask = 0;
    XGCValues values;
    Display *display;
    Pixmap bitmap;
```

```
XPoint points[800];
FILE *fp, *fopen();
char str[100];

XSetWindowAttributes attr[1];

if( (display = XOpenDisplay(display_name)) == NULL) { // connect to Xserver
   fprintf(stderr, "Cannot connect to X server %s\n", XDisplayName(display_name));
   exit(-1);
}

screen = DefaultScreen(display); // get screen sized
display_width = DisplayWidth(display, screen);
display_height = DisplayHeight(display, screen);

width = X_RESN;
height = Y_RESN;
win_x = 0;
win_y = 0;

border_width = 4;
win = XCreateSimpleWindow (display, RootWindow (display, screen),
                win_x, win_y, width, height, border_width,
                BlackPixel (display, screen), WhitePixel (display, screen));

XStoreName(display, win, window_name);

gc = XCreateGC (display, win, valuemask, &values); // create graphics context
XSetBackground (display, gc, WhitePixel (display, screen));
XSetForeground (display, gc, BlackPixel (display, screen));
XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);
XMapWindow (display, win);
XSync(display, 0);

int counter = 0;
int blue = 0x0054A6;
int cyan_blue = 0x0072BC;
int cyan = 0x00AEEF;
int green_cyan = 0x00A99D;
int green = 0x00A651;
int yellow_green = 0x39B54A;
int yellow = 0xFFF200;
int yellow_orange = 0xF7941D;
```

```
int orange = 0xF26522;
int red = 0xED1C24;

usleep(100000);
XClearWindow(display,win);
XSetForeground(display,gc, (long) 0xDC143C);
int x_counter, y_counter = 0;
for(i = 0; i < N; i++) {
   for(j = 0; j < N; j++){
      if(h[0][i][j] >= 0.0 && h[0][i][j] <= 5.0){
         XSetForeground(display,gc, (long) blue);
      }
      if(h[0][i][j] > 5.0 && h[0][i][j] <= 10.0){
         XSetForeground(display,gc, (long) cyan_blue);
      }
      else if(h[0][i][j] > 10.0 && h[0][i][j] <= 20.0){
         XSetForeground(display,gc, (long) cyan);
      }
      else if(h[0][i][j] > 20.0 && h[0][i][j] <= 30.0){
         XSetForeground(display,gc, (long) green_cyan);
      }
      else if(h[0][i][j] > 30.0 && h[0][i][j] <= 40.0){
         XSetForeground(display,gc, (long) green);
      }
      else if(h[0][i][j] > 40.0 && h[0][i][j] <= 50.0){
         XSetForeground(display,gc, (long) yellow_green);
      }
      else if(h[0][i][j] > 50.0 && h[0][i][j] <= 60.0){
         XSetForeground(display,gc, (long) yellow);
      }
      else if(h[0][i][j] > 60.0 && h[0][i][j] <= 70.0){
         XSetForeground(display,gc, (long) yellow_orange);
      }
      else if(h[0][i][j] > 70.0 && h[0][i][j] <= 80.0){
         XSetForeground(display,gc, (long) orange);
      }
      else if(h[0][i][j] > 80.0 && h[0][i][j] <= 90.0){
         XSetForeground(display,gc, (long) orange);
      }
      else if(h[0][i][j] > 90.0 && h[0][i][j] <= 100.0){
         XSetForeground(display,gc, (long) red);
      }
      XFillArc(display,win,gc,x_counter, y_counter,5,5,0,23040);
```

```
            x_counter += 10;
        }
        x_counter = 0;
        y_counter += 10;
    }


    XFlush(display);
    usleep(10000000);
    return 0;

}

void getInput(int* N, int* T) {
    // Get the size of the matrix
    int valid = 0;
    while(!valid) {
        printf("Please enter matrix size(NxN): ");
        scanf("%d", N);
        if(N) {
            valid = 1;
        }
    }

    // Get the number of iterations
    valid = 0;
    while(!valid) {
        printf("Please enter the number of iterations: ");
        scanf("%d", T);
        if(N) {
            valid = 1;
        }
    }
}
```

OpenMP Version of X11 code

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

```
#include <X11/Xos.h>
#define X_RESN 800
#define Y_RESN 800

void getInput(int*, int*);

void main(int arc, char* argv) {
    omp_set_num_threads(4);
    double begin,end;
    double time_spent;
    int iteration, i, j, current, next, N, T;

    getInput(&N, &T);

    //initaialize the array
    double h[2][N][N];

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            h[0][i][j] = 0;
            h[1][i][j] = 0;
        }
    }
    // Set all the walls to 20C degrees
    for(i = 0; i < N; i++) {
        h[0][0][i] = 20.0;
        h[0][i][0] = 20.0;
        h[0][N-1][i] = 20.0;
        h[0][i][N-1] = 20.0;
    }

    // Set the starting point and the ending point of the fireplace
    double fp_start, fp_end;
    fp_start = 0.3 * N;
    fp_end = (0.7 * N);

    // Initialize the values of the location of the fireplacee to 100C
    for(i = fp_start; i < fp_end; i++) {
        h[0][0][i] = 100.0;
    }

    // Get the start time of the program
    begin = omp_get_wtime();
```

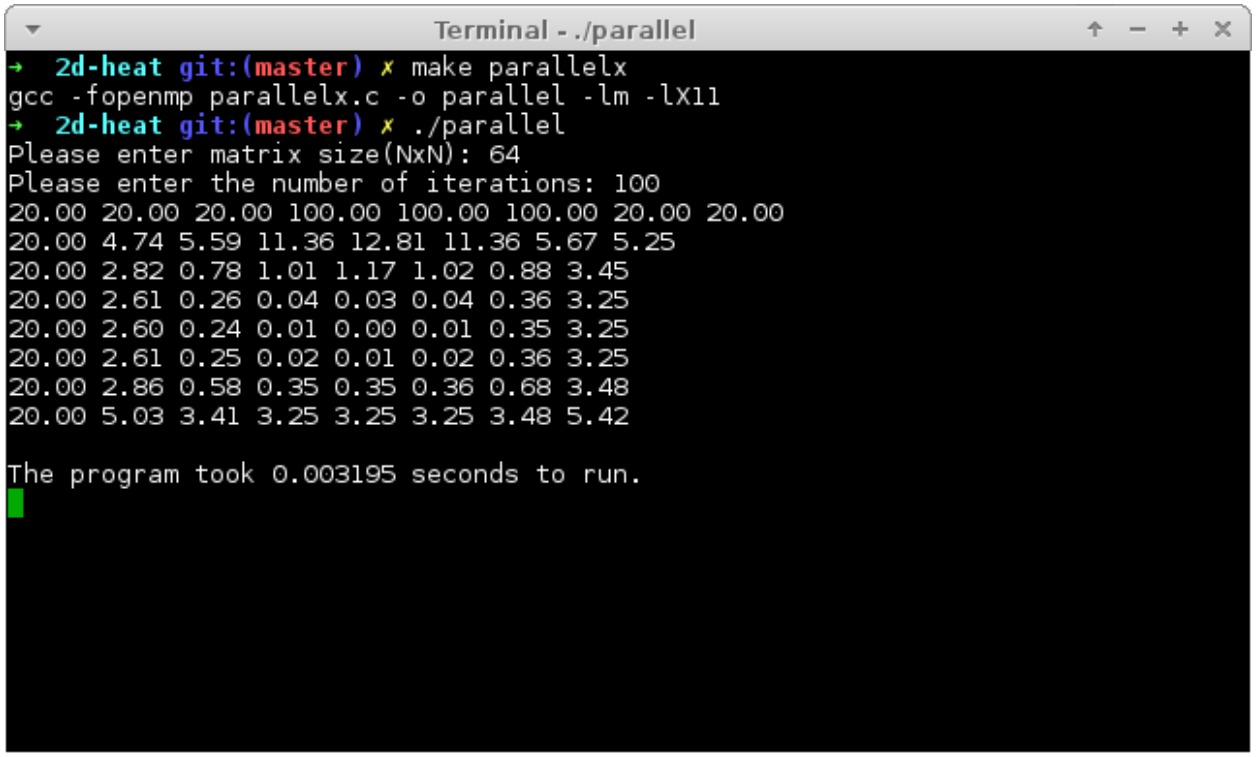
```
    // The actual calculation of the temperature of the room.
    current = 0;
    next = 1;
    for (iteration = 0; iteration < T; iteration++) {
        #pragma omp parallel for private(i, j)
        for( i = 1; i < N-1; i++) {
            for( j = 1; j < N-1; j++) {
                h[next][i][j] = 0.25 * (h[current][i-1][j] + h[current][i+1][j]+ h[current][i][j-1] +
h[current][i][j+1]);
            }
        }
        current = next;
        next = (current + 1) % 2;
    }

    // Get the end time
    end = omp_get_wtime();
    // Calculate the time spent
    time_spent = end - begin;

    // Print an 8x8 array the N/8 x N/8 positions of the room.
    for(i = 0; i < N; i+= N/8) {
        for(j = 0; j < N; j+= N/8) {
            printf("%.2f ", h[0][i][j]);
        }
        printf("\n");
    }
    printf("\n");

    // Print the time in seconds that the program took to run
    printf("The program took %f seconds to run.\n", time_spent);

    /* Draw X stuff
    *
    */
    Window win;
    unsigned int width, height,
    win_x, win_y,
    border_width,
    display_width, display_height,
    screen;
    char *window_name = "2d-heat program", *display_name = NULL;
```

```
GC gc;
unsigned long valuemask = 0;
XGCValues values;
Display *display;
Pixmap bitmap;
XPoint points[800];
FILE *fp, *fopen();
char str[100];

XSetWindowAttributes attr[1];

if( (display = XOpenDisplay(display_name)) == NULL) { // connect to Xserver
   fprintf(stderr, "Cannot connect to X server %s\n", XDisplayName(display_name));
   exit(-1);
}

screen = DefaultScreen(display); // get screen sized
display_width = DisplayWidth(display, screen);
display_height = DisplayHeight(display, screen);

width = X_RESN;
height = Y_RESN;
win_x = 0;
win_y = 0;

border_width = 4;
win = XCreateSimpleWindow (display, RootWindow (display, screen),
                win_x, win_y, width, height, border_width,
                BlackPixel (display, screen), WhitePixel (display, screen));

XStoreName(display, win, window_name);

gc = XCreateGC (display, win, valuemask, &values); // create graphics context
XSetBackground (display, gc, WhitePixel (display, screen));
XSetForeground (display, gc, BlackPixel (display, screen));
XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);
XMapWindow (display, win);
XSync(display, 0);

int counter = 0;
int blue = 0x0054A6;
int cyan_blue = 0x0072BC;
int cyan = 0x00AEEF;
```

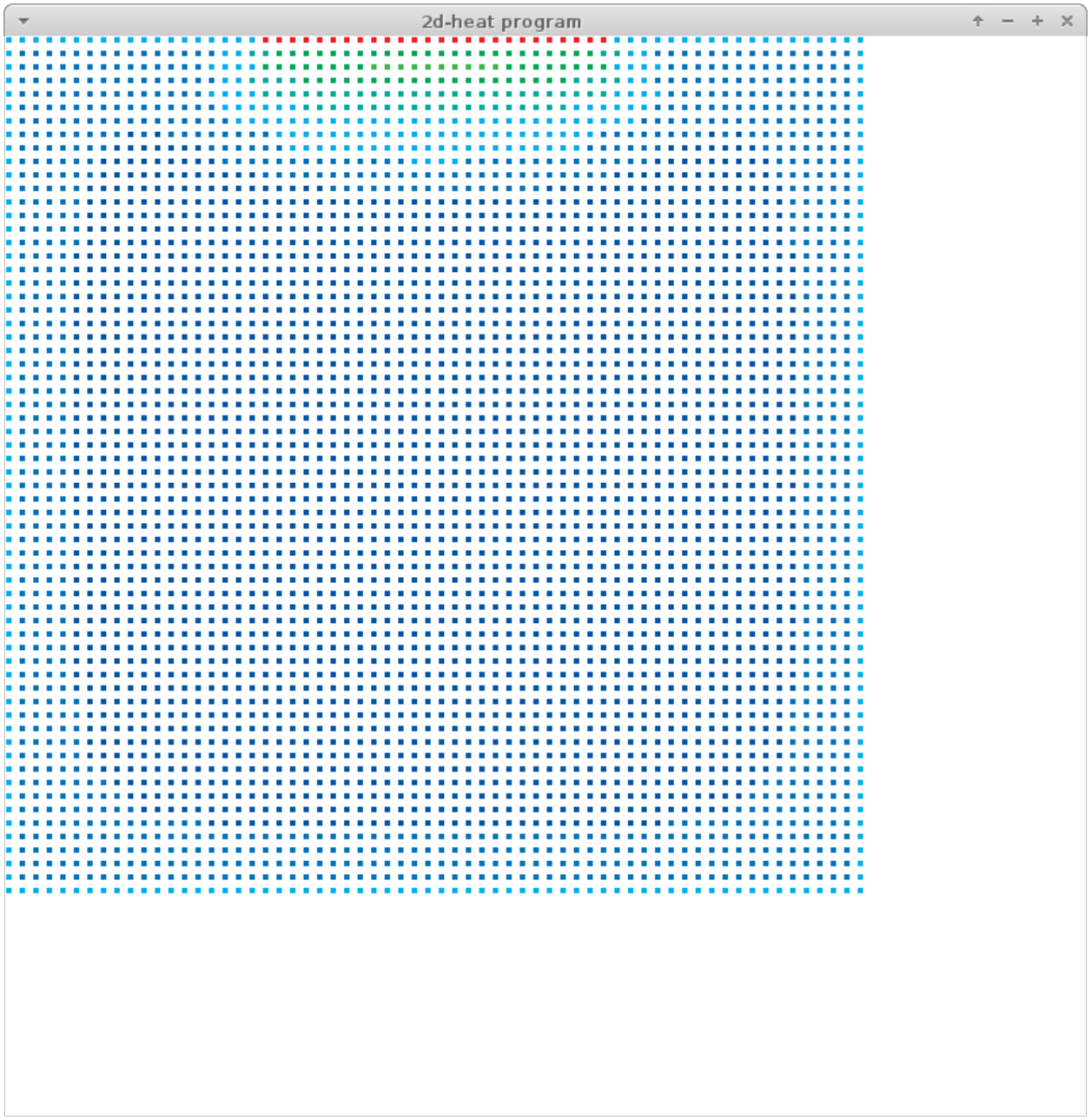
```
int green_cyan = 0x00A99D;
int green = 0x00A651;
int yellow_green = 0x39B54A;
int yellow = 0xFFF200;
int yellow_orange = 0xF7941D;
int orange = 0xF26522;
int red = 0xED1C24;

usleep(100000);
XClearWindow(display,win);
XSetForeground(display,gc, (long) 0xDC143C);
int x_counter, y_counter = 0;
#pragma omp for private(i, j)
for(i = 0; i < N; i++) {
   for(j = 0; j < N; j++){
      if(h[0][i][j] >= 0.0 && h[0][i][j] <= 5.0){
         XSetForeground(display,gc, (long) blue);
      }
      if(h[0][i][j] > 5.0 && h[0][i][j] <= 10.0){
         XSetForeground(display,gc, (long) cyan_blue);
      }
      else if(h[0][i][j] > 10.0 && h[0][i][j] <= 20.0){
         XSetForeground(display,gc, (long) cyan);
      }
      else if(h[0][i][j] > 20.0 && h[0][i][j] <= 30.0){
         XSetForeground(display,gc, (long) green_cyan);
      }
      else if(h[0][i][j] > 30.0 && h[0][i][j] <= 40.0){
         XSetForeground(display,gc, (long) green);
      }
      else if(h[0][i][j] > 40.0 && h[0][i][j] <= 50.0){
         XSetForeground(display,gc, (long) yellow_green);
      }
      else if(h[0][i][j] > 50.0 && h[0][i][j] <= 60.0){
         XSetForeground(display,gc, (long) yellow);
      }
      else if(h[0][i][j] > 60.0 && h[0][i][j] <= 70.0){
         XSetForeground(display,gc, (long) yellow_orange);
      }
      else if(h[0][i][j] > 70.0 && h[0][i][j] <= 80.0){
         XSetForeground(display,gc, (long) orange);
      }
      else if(h[0][i][j] > 80.0 && h[0][i][j] <= 90.0){
```

```
            XSetForeground(display,gc, (long) orange);
        }
        else if(h[0][i][j] > 90.0 && h[0][i][j] <= 100.0){
            XSetForeground(display,gc, (long) red);
        }
        XFillArc(display,win,gc,x_counter, y_counter,5,5,0,23040);
        x_counter += 10;
      }
      x_counter = 0;
      y_counter += 10;
   }


   XFlush(display);
   usleep(10000000);
}

void getInput(int* N, int* T) {
   // Get the size of the matrix
   int valid = 0;
   while(!valid) {
      printf("Please enter matrix size(NxN): ");
      scanf("%d", N);
      if(N) {
         valid = 1;
      }
   }

   // Get the number of iterations
   valid = 0;
   while(!valid) {
      printf("Please enter the number of iterations: ");
      scanf("%d", T);
      if(N) {
         valid = 1;
      }
   }
}
```

Screenshot of openmp code running on my machine:



And the output of the X11 window

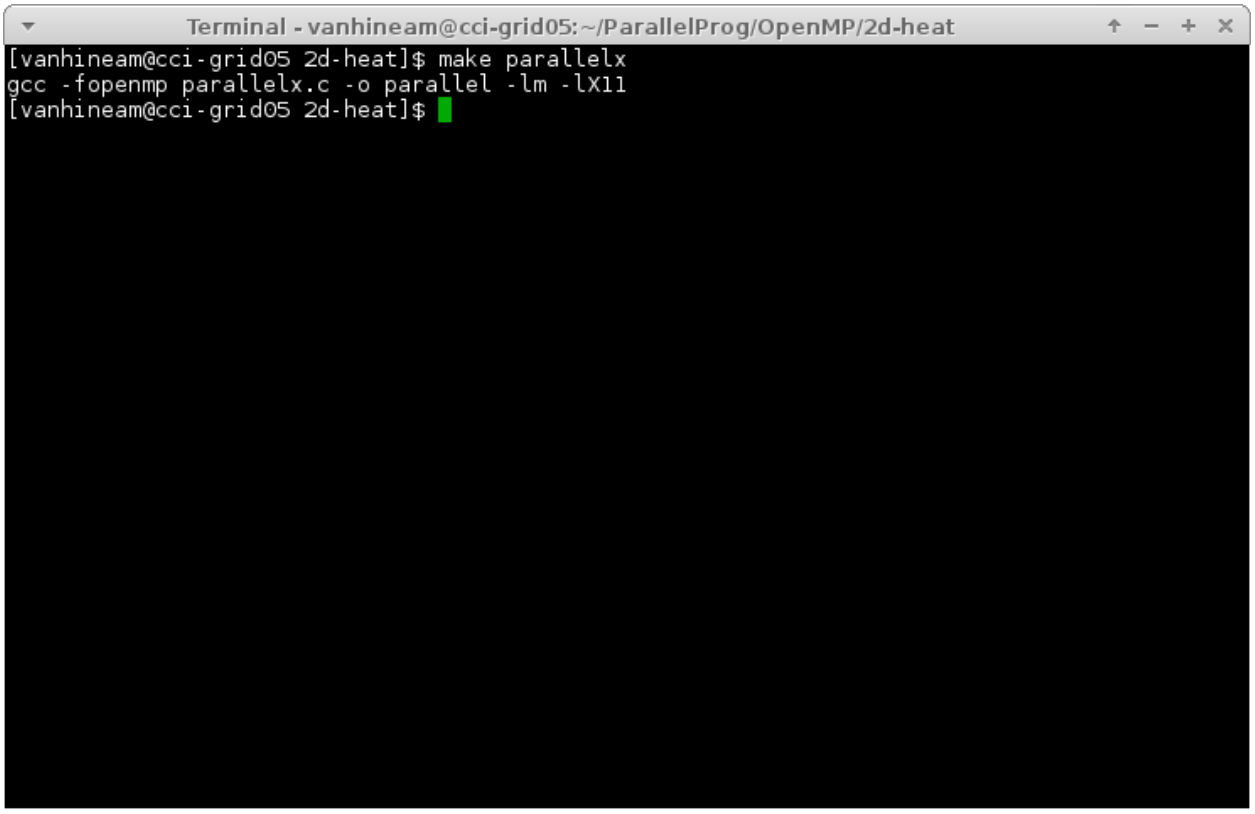


I did not render this image with the amplified colors as I did before. I did as the assignment stated and used colors every 10C degrees apart.

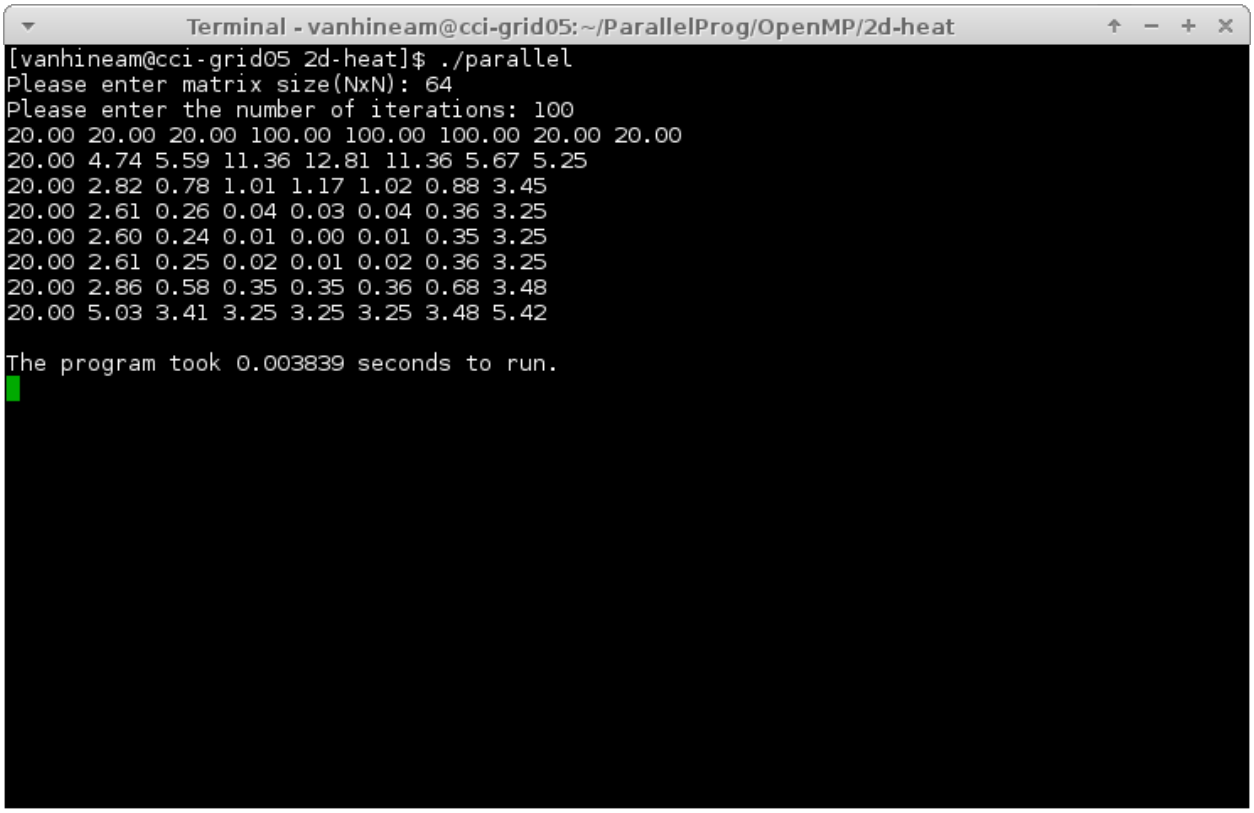
Conclusion:
I really enjoyed learning about X11 graphics. I have used X11 for a long time with linux, but never programatically rendered anything with it. Being able to have something cool like heat distribution to render was a huge plus as well. This was a really fun assignment and I learned a lot.

Task 4

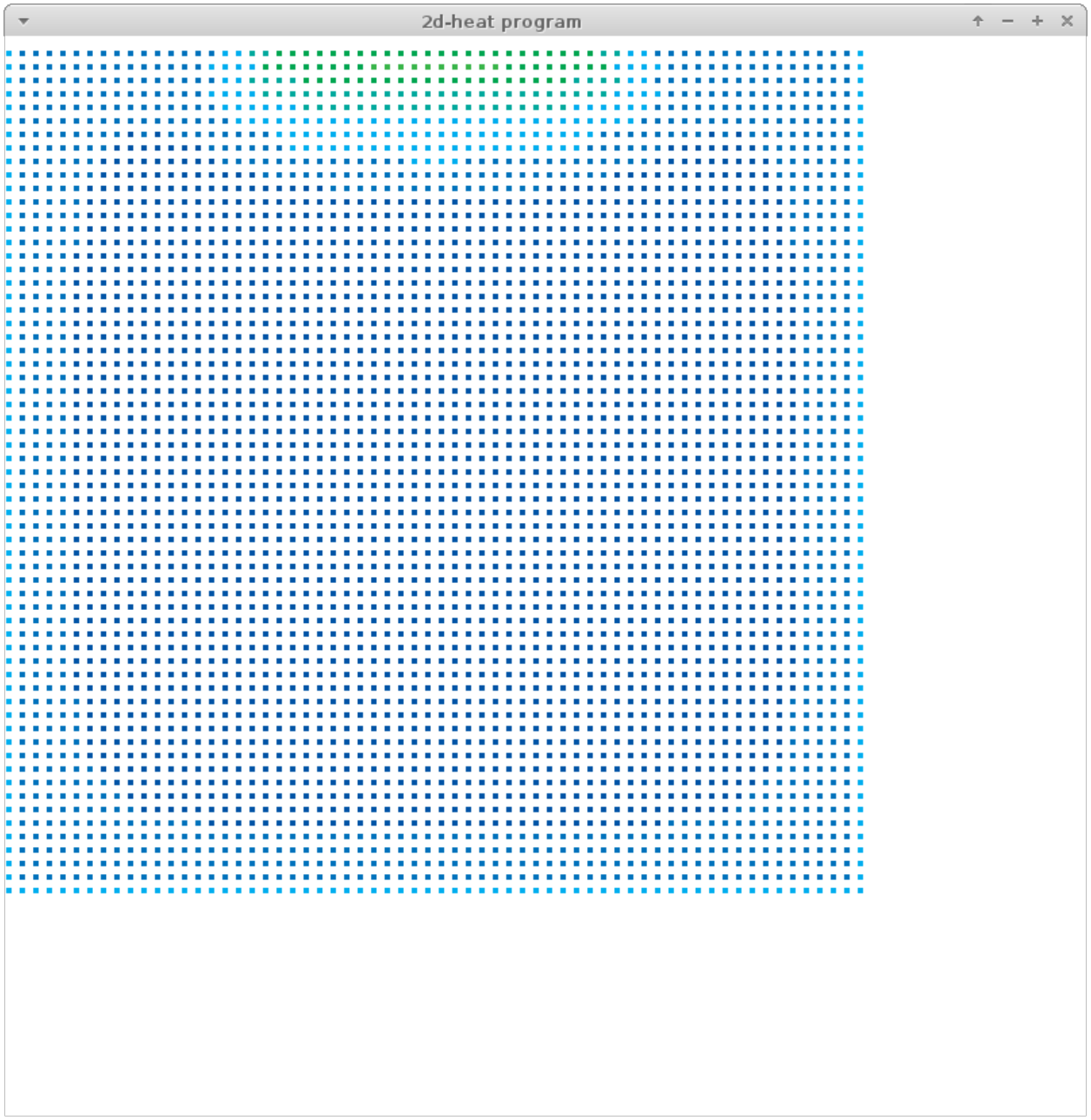
Screenshot of compiling on cci-grid05



Running on cci-grid05



Graphics on cci-grid05



Conclusion:
This output I was not expecting. As we can see on the output of the program there are multiple 100.0 along the top of the grid. These points are not being rendered as red for some reason. I have looked into the issue with no avail but will continue to search and ask in class if need be. Running things on clusters is awesome!