CSC8014 Topics: GPU Programming
Student: Lina Mi

Homework #4

1. What's the purpose of texture memory? In what circumstance does texture memory increase the performance of GPU computing?

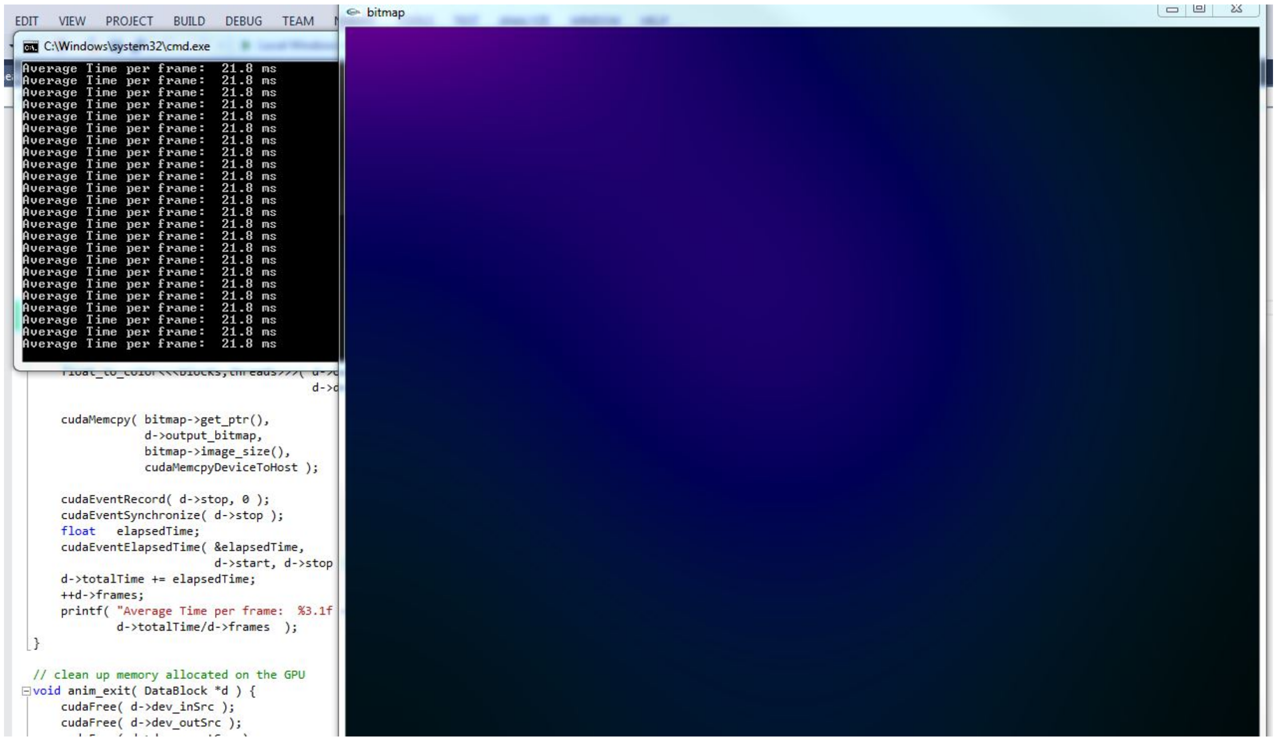
   Ans: the purpose of texture memory is to improve performance and reduce memory traffic when read have certain access patterns. In the graphic applications and the computation applications where the memory access exhibit a great deal of 'spatial locality', namely a thread is likely to read from the address 'near' the address that nearby threads read, texture memory could increase the performance of GPU computing

2. Read the attached code `heat.cu` and answer the following questions:
   a. What's the purpose of calling `copy_const_kernel()` in the for loop in the method `anim_cpu()`? What expect to happen if we comment this method call out?

      Ans: the purpose of `copy_const_kernel()` is to copy the temperature of cells with heater to the grid as input temperature of heating cells in every time of temperature update computation, that means the heater in the grid keeps heating the neighbor cells.

      If this method is called out by comments, the grid will lose the heating source after being heated by the heater at the very beginning of the program, then the whole grid will cool down and return to original temperature.

      The resulting image of the program with method `copy_const_kernel()` being commented is shown as following:

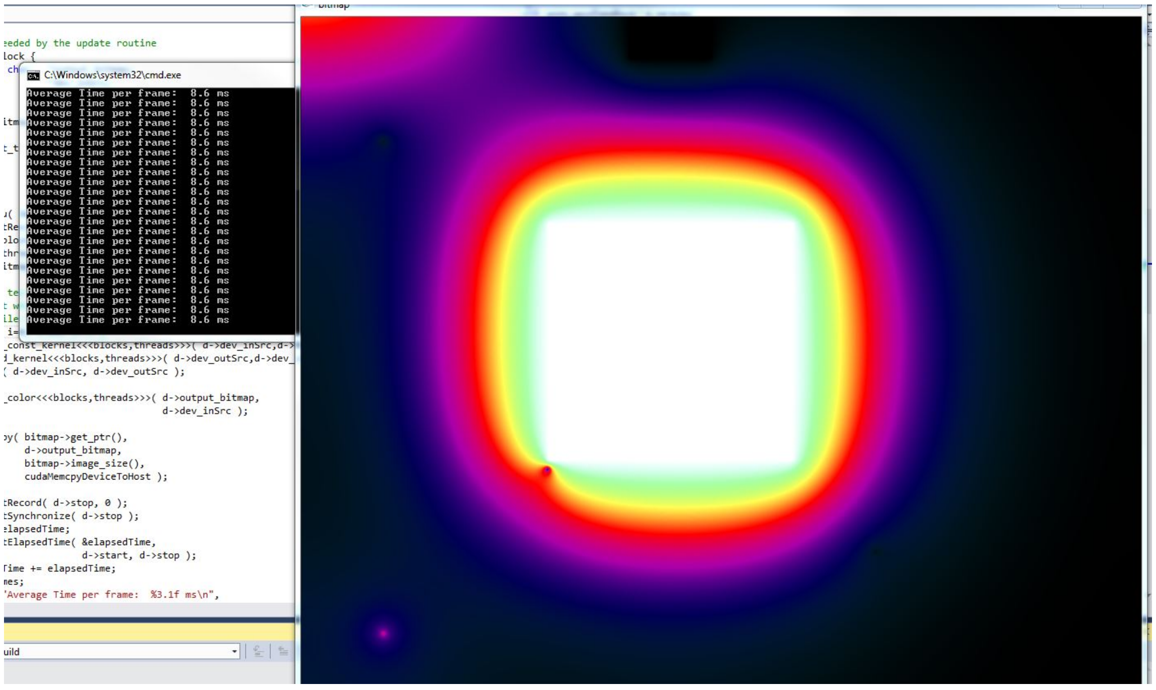

EDIT VIEW PROJECT BUILD DEBUG TEAM
C:\Windows\system32\cmd.exe
bitmap

```
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
Average Time per frame:  21.8 ms
```

```
                                        d->c

    cudaMemcpy( bitmap->get_ptr(),
                d->output_bitmap,
                bitmap->image_size(),
                cudaMemcpyDeviceToHost );

    cudaEventRecord( d->stop, 0 );
    cudaEventSynchronize( d->stop );
    float   elapsedTime;
    cudaEventElapsedTime( &elapsedTime,
                          d->start, d->stop
    d->totalTime += elapsedTime;
    ++d->frames;
    printf( "Average Time per frame:  %3.1f
            d->totalTime/d->frames  );
}

// clean up memory allocated on the GPU
void anim_exit( DataBlock *d ) {
    cudaFree( d->dev_inSrc );
    cudaFree( d->dev_outSrc );
```

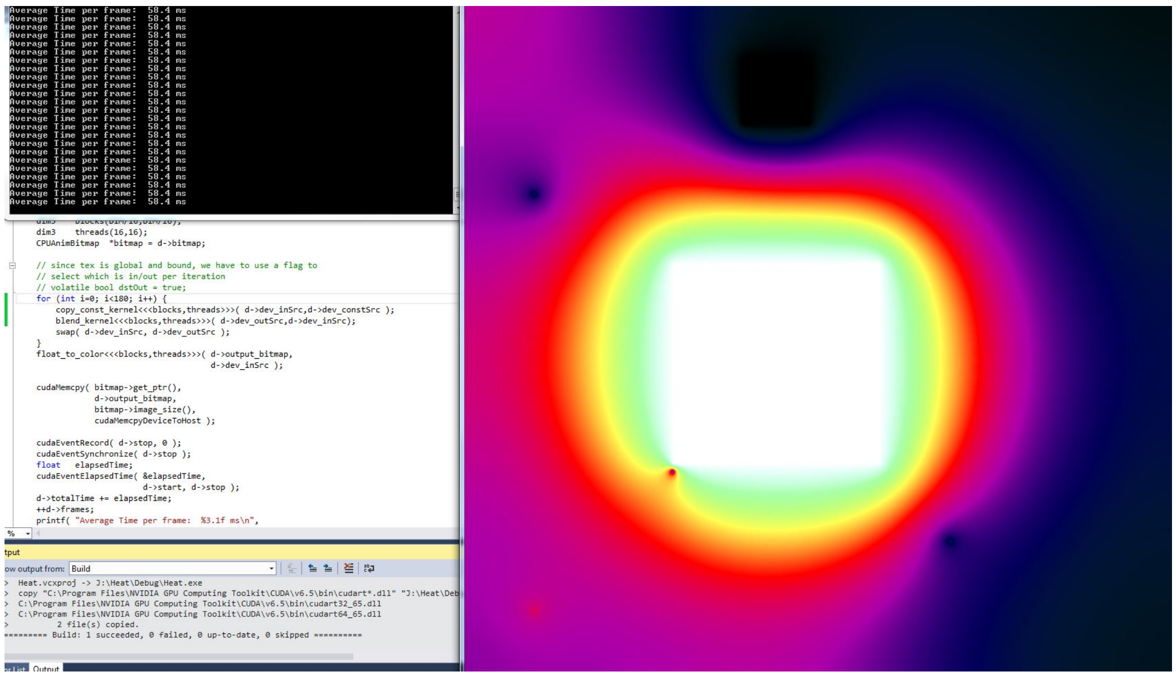
b. In method, the for loop iterates 90 times. Can we change the number? What difference does it make if we decrease the number? What about if we increase the number to 180?

   Ans: we can change that number, if the number of iteration in `for` loop is decreased, such as 20, the average time per frame is decreased, and the temperatures (presented as color of screen) in the animate image of heat simulation will change slower than that with 90 iteration number. The resulting image is shown as following:



```
eeded by the update routine
lock {
  ch
C:\Windows\system32\cmd.exe
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
Average Time per frame:  8.6 ms
  i=
_const_kernel<<<blocks,threads>>>( d->dev_inSrc,d->
d_kernel<<<blocks,threads>>>( d->dev_outSrc,d->dev_
( d->dev_inSrc, d->dev_outSrc );

_color<<<blocks,threads>>>( d->output_bitmap,
                              d->dev_inSrc );

py( bitmap->get_ptr(),
    d->output_bitmap,
    bitmap->image_size(),
    cudaMemcpyDeviceToHost );

tRecord( d->stop, 0 );
tSynchronize( d->stop );
elapsedTime;
tElapsedTime( &elapsedTime,
              d->start, d->stop );
Time += elapsedTime;
mes;
'Average Time per frame:  %3.1f ms\n",
```

   If we increase the number to 180, the the ge time per frame increased. temperature of cells in the grid will change much faster than before, presented as faster color changing of the animate image. The resulting image is shown as following:

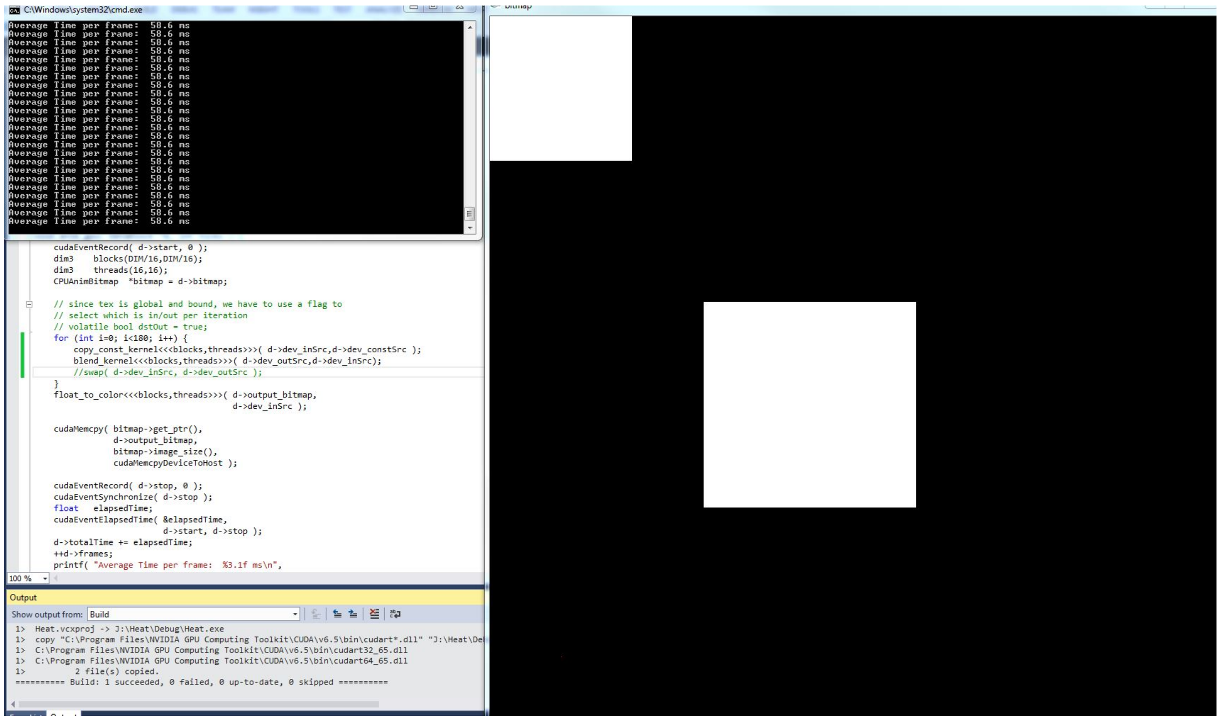


```
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
Average Time per frame:  58.4 ms
    dim3    blocks(DIM/16,DIM/16);
    dim3    threads(16,16);
    CPUAnimBitmap  *bitmap = d->bitmap;

    // since tex is global and bound, we have to use a flag to
    // select which is in/out per iteration
    // volatile bool dstOut = true;
    for (int i=0; i<180; i++) {
        copy_const_kernel<<<blocks,threads>>>( d->dev_inSrc,d->dev_constSrc );
        blend_kernel<<<blocks,threads>>>( d->dev_outSrc,d->dev_inSrc);
        swap( d->dev_inSrc, d->dev_outSrc );
    }
    float_to_color<<<blocks,threads>>>( d->output_bitmap,
                                        d->dev_inSrc );

    cudaMemcpy( bitmap->get_ptr(),
                d->output_bitmap,
                bitmap->image_size(),
                cudaMemcpyDeviceToHost );

    cudaEventRecord( d->stop, 0 );
    cudaEventSynchronize( d->stop );
    float   elapsedTime;
    cudaEventElapsedTime( &elapsedTime,
                          d->start, d->stop );
    d->totalTime += elapsedTime;
    ++d->frames;
    printf( "Average Time per frame:  %3.1f ms\n",
```

```
> Heat.vcxproj -> J:\Heat\Debug\Heat.exe
> copy "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\bin\cudart*.dll" "J:\Heat\Deb
> C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\bin\cudart32_65.dll
> C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\bin\cudart64_65.dll
>         2 file(s) copied.
========= Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

c. What's the purpose of calling `swap()` in the for loop in the method `anim_cpu()`?

`Ans:swap()` has the function of exchanging the output buffer with input buffer to make preparation for next iteration of temperatures update computation, namely, it will use the output temperatures of grid in current iteration as the input in next iteration of temperature computation and the input buffer in current iteration computation is used to hold the output of next iteration of temperature update computation, since in each iteration of temperature update computation, the new temperature vales of cells are computed based on the old state of cell temperatures, namely the output temperature values of cells in previous iteration. Without the `swap()`, the temperature of grid will not change after first iteration of temperature computation since all computation will end with same temperature values with same input temperature of grid, it means that there is no heat transfer from 'heater' cells to their neighbor cells. The resulted image is shown in following:



```
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
Average Time per frame:  58.6 ms
```

```
    cudaEventRecord( d->start, 0 );
    dim3    blocks(DIM/16,DIM/16);
    dim3    threads(16,16);
    CPUAnimBitmap  *bitmap = d->bitmap;

    // since tex is global and bound, we have to use a flag to
    // select which is in/out per iteration
    // volatile bool dstOut = true;
    for (int i=0; i<180; i++) {
        copy_const_kernel<<<blocks,threads>>>( d->dev_inSrc,d->dev_constSrc );
        blend_kernel<<<blocks,threads>>>( d->dev_outSrc,d->dev_inSrc);
        //swap( d->dev_inSrc, d->dev_outSrc );
    }
    float_to_color<<<blocks,threads>>>( d->output_bitmap,
                                        d->dev_inSrc );

    cudaMemcpy( bitmap->get_ptr(),
                d->output_bitmap,
                bitmap->image_size(),
                cudaMemcpyDeviceToHost );

    cudaEventRecord( d->stop, 0 );
    cudaEventSynchronize( d->stop );
    float   elapsedTime;
    cudaEventElapsedTime( &elapsedTime,
                          d->start, d->stop );
    d->totalTime += elapsedTime;
    ++d->frames;
    printf( "Average Time per frame:  %3.1f ms\n",
```

```
1>  Heat.vcxproj -> J:\Heat\Debug\Heat.exe
1>  copy "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\bin\cudart*.dll" "J:\Heat\De
1>  C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\bin\cudart32_65.dll
1>  C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\bin\cudart64_65.dll
1>          2 file(s) copied.
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

3. Read the attached code `heat_2d.cu` where texture memory is used. Please answer the following questions:

   a. The implementation of `anim_gpu()` is different from the one in `heat.cu`, why is the method `swap()` no longer called?

   The `swap()` function is implemented by the code section in the `for` loop in `anim_gpu()`:

```
    if (dstOut) {
                in  = d->dev_inSrc;
                out = d->dev_outSrc;
            } else {
                out = d->dev_inSrc;
                in  = d->dev_outSrc;
         }
    dstOut=!dstOut;
```

and the code section in `blend_kernel()`:

```
if (dstOut) {
        t = tex2D(texIn,x,y-1);
        l = tex2D(texIn,x-1,y);
        c = tex2D(texIn,x,y);
        r = tex2D(texIn,x+1,y);
        b = tex2D(texIn,x,y+1);
    } else {
        t = tex2D(texOut,x,y-1);
        l = tex2D(texOut,x-1,y);
        c = tex2D(texOut,x,y);
        r = tex2D(texOut,x+1,y);
        b = tex2D(texOut,x,y+1);
    }
```

b. How do we bind the GPU memory constant buffer of heaters, input, and output temperature to texture memory.

   Ans: use the function `cudaBindTexture2D()` to bind GPU memory to texture memory:

   Binde the constant buffer of heaters to texture memory:

```
cudaBindTexture2D( NULL, texConstSrc,  data.dev_constSrc, desc, DIM, DIM,
sizeof(float) * DIM );
```

   Binde the input buffer of heaters to texture memory:

```
cudaBindTexture2D( NULL, texIn, data.dev_inSrc, desc, DIM, DIM,
sizeof(float) * DIM );
```

   Binde the output buffer of heaters to texture memory:

```
cudaBindTexture2D( NULL, texOut,   data.dev_outSrc, desc, DIM, DIM,
                        sizeof(float) * DIM );
```