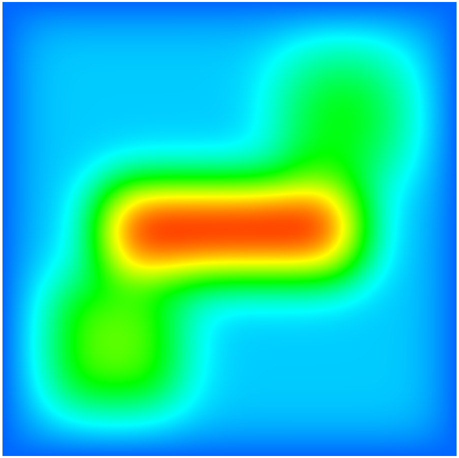
Recitation 3

Announcements

- Assignment 6 is out:
  - Theory, pass/fail, deadline 9.11
  - Code, graded (10%) deadline 16.11
- Assignment 7 is also out, it is optional. Deadline 23.11.

Heat equation



Once more, this time using CUDA.

Parallelization

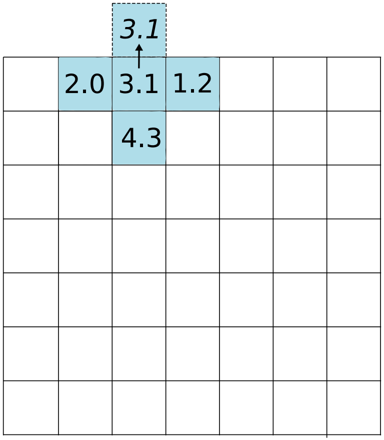
- One kernel launch for each time step.
- One GPU thread for each grid cell.
- We also need one kernel launch for each time step for the external heat function.

Boundary condition

- We switch from constant(Dirichelt) to "mirrored" (Neuman) boundary condition.
- For this reason there are no halos for any of the arrays.

Boundary condition

- When you access youside the array, you should use the value at the boundary.



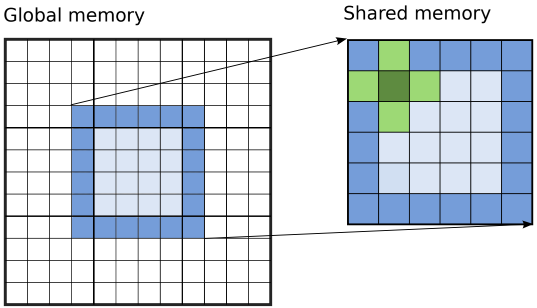
Your task

- You should write three different versions:
- One using just global memory.
- One using shared memory.
- One using texture memory.
- Time the different versions with different block sizes.

Shared memory

- Shared memory is fast, on-chip memory, which can be used as a programmer managed cache.
- For the shared memory version, you should store the (input) temperature array in shared memory.
- In particular, each thread block needs to load a the part of the temperature array they will be using into shared memory.
- The threads of the block will then perform the computation, using the values in shared memory.

Shared memory



Light blue: area of one thread block. Dark blue: additional halo. (There is no global halo)

Shared memory

To allocate shared memory:

Size known at compile time

```
__shared__ float sharedArray[1024];
```

Size not known at compile time

```
my_kernel<<<nBlocks, nThreads,
  sharedArraySize>>>(...);
...
extern __shared__ float sharedArray[];
```

Shared memory is shared between the threads of a thread block.

Synchronization

- `__syncthreads()` works as a barrier for the threads of one thread block.
- (Global synchronization of all the threads requires returning controll to the CPU)
- After loading from global to shared memory, the threads should synchronize, so that all the threads know all the data is available.

Texture memory

- Texture memory is physically a part of the global memory, but it is read thorough a special texture cache, which is optimized for spatial locality.
- You also have hardware for address translation, interpolation etc.
- For the assignment, you should read the (input) temperature array, using texture memory.

Texture memory

- To use texture memory, you create a texture reference, and bind it to an array in global memory. Later, you use the texture reference to access the array.
- Texture references can be bound to:
  - linear arrays, allocated with `cudaMalloc`
  - picth allocated linear arrays, allocated with `cudaMallocPitch`
  - CUDA arrays
- For the assignment, you are only required to use the first option. (I.e. this will give you full score)
- The other two can give better performance, but requires that you also change the allocation, indexing calculations etc.

Texture memory

- There is a simple code example on it’s Learning.

Timing

- You should time the different versions, using different block sizes, and write a simple report.
- The report should contain a table/graph with the timings, and some brief comments (1-2 paragraphs). In particular, you should try to explain any unexpected results.
- The kernel launch functions all return floats, this should be the kernel execution time. There is already code for finding the min/max/avg of all the kernel launches.
- Info on how to time kernels can be found in Nvidias best practices guide: `http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html`

Block sizes

- You should use a two dimensional thread grid. You should support arbitrary thread block sizes (i.e not just powers of two, even numbers).
- (You can assume that the size of the temperature/material arrays are allways powers of two)

Practicalities

- All your code will go into a single .cu file, a command line argument can be used to specify which version to use.
- The thread block size can also be read from the command line.
- A serial reference implementation has also been included. (Due to the different boundary condition, and a larger grid, the output is not identical to that of previous assignments.)