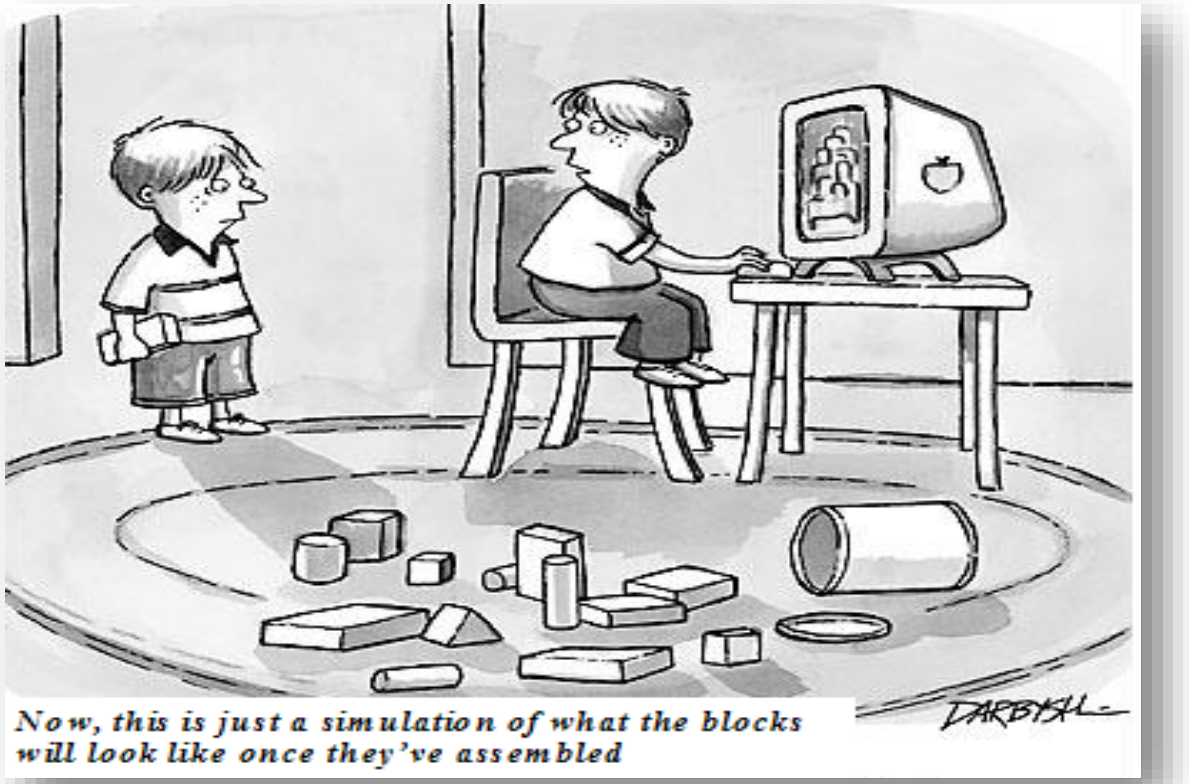
ECE569
Module 17



- CUDA Memory Review

Shared Memory

```
__shared__ int x_dim;
__shared__ float x[128];
__shared__float x[width][height]
```

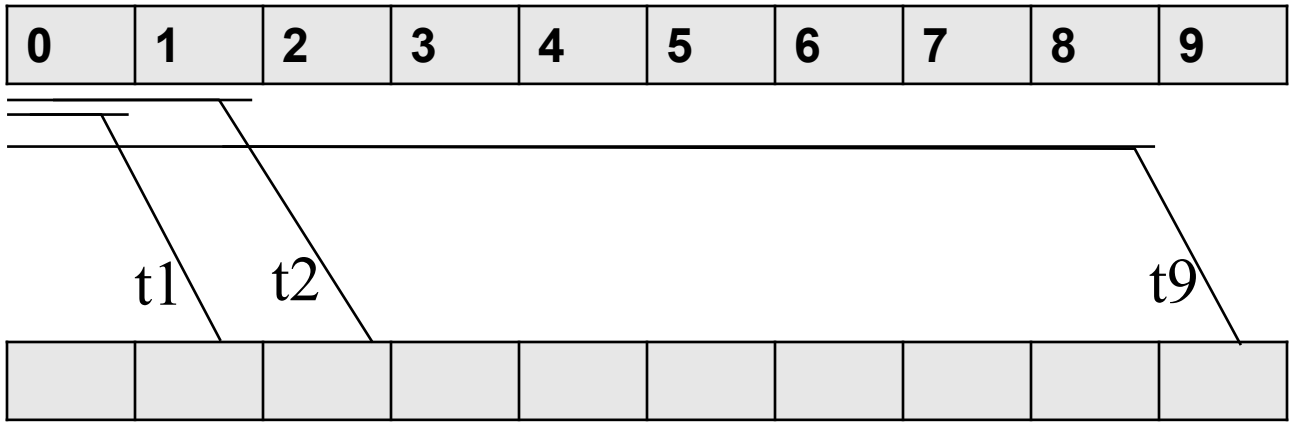
- data shared between all the threads in a thread block – any thread can set its value, or read it.
- contents will disappear after the corresponding thread finishes terminates execution
- There can be several benefits:
  - essential for operations requiring communication between threads
  - useful for data re-use and alternative to local arrays in device memory

Shared Memory

- If a thread block has more than one warp, it's **not pre-determined** when each warp will execute its instructions
  - warp 1 could be many instructions ahead of warp 2, or well behind.
- Consequently, almost always **need thread synchronization** to ensure correct use of shared memory.

Code Review (Refer to Demo=>4.Memory)

- **Function:**
  - Input: `input_array` of 128 elements
  - Output: `output_array` of 128 elements
  - For each position in the `input_array`, find average of all previous elements and write into the `output_array`



Global Memory Version

```
__global__ void global_memory_average(int
*array, float* average, int size){
int i, index = threadIdx.x;
float avg = 0.0f;
int sum = 0;

for (i=0; i<index; i++)
  sum += array[i];
if (index>0)
  avg = sum/(index+0.0f);

average[index] = avg;
}
```

Time:0.0408 ms

Using Shared Memory

Code Review

```
__global__ void shared_memory_average(int*array,float*average,int size){
    // local variables, private to each thread
    int i, index = threadIdx.x;
    float avg =0.0f;
    int sum = 0;

    // __shared__ variables are visible to all threads in the block
    // and have the same lifetime as the thread block
    __shared__ float sh_arr[128];
    // all blocks create their single copy of sh_arr, size known

    // copy data from "array" in global memory to sh_arr in shared mem.
    // here, each thread is responsible for copying a single element.
    // sharing workload and collectively brining each others data
    sh_arr[index] = array[index];

    __syncthreads();// ensure all writes to shared memory have completed

    // sh_arr is fully populated.

    //find average of all previous elements
    :
    :
```

Using Shared Memory

```
__global__ void use_shared_memory_GPU(float *array)
{   //cntd. from previous slide
    //find average of all previous elements
    for (i=0; i<index; i++)
    {
       sum += sh_arr[i];
    }
    if (index>0)
    avg = sum / (index + 0.0f);

    average[index] = avg;
    // since array[] is in global memory, this change will be seen
    // by the host (and potentially other thread blocks, if any)

}
int main(int argc, char **argv)
{
    /*  First, call a kernel that shows using shared memory */
    /*  d_d input array, d_average: output
    shared_memory_average<<<1, 128>>>(d_d,d_average,n);
}
```

Time: 0.0408 to 0.0129ms

Memory Model

- **True/False**
  - ❑ All threads from a block can access the same variable in that block's shared memory
  - ❑ Threads from two different blocks can access the same variable in global memory
  - ❑ Threads from different blocks have their own copy of local variables in local memory
  - ❑ Threads from the same block have their own copy of local variable in local memory.

Rank from fastest (ranked 1st) to slowest (ranked 4th). Which one is ranked 3rd?

```
__global__void foo(float* x,float* y,float* z)
{
  __shared__ float a,b,c;
  float s,t,u;


  s=*x

  t=s;

  a=b;

  *y=*z;


}
```

Resources

- CUDA Programming Guide
  - **Appendix B.1-B.4 – essential**
  - Chapter 3, sections 3.2.1-3.2.3
- Other reading:
  - Wikipedia article on caches:
    - en.wikipedia.org/wiki/CPU cache
  - web article on caches:
    - lwn.net/Articles/252125/

Next

- **Thread Synchronization**