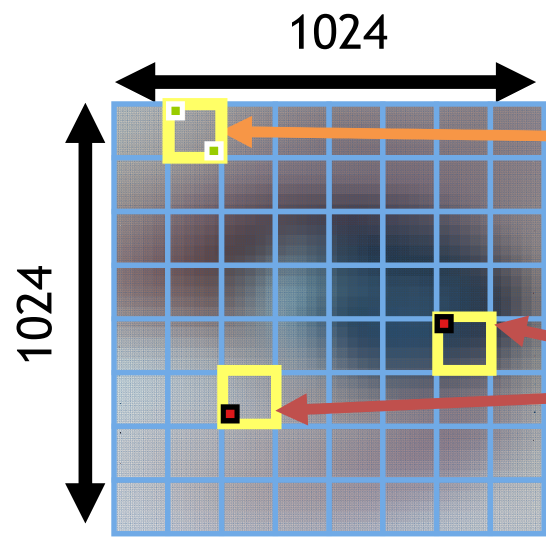
Lecture 7

SYNCHRONIZATION IN OPENCL

Consider N-dimensional domain of work-items

- Global Dimensions:
  - 1024x1024 (whole problem space)
- Local Dimensions:
  - 64x64 (**work-group**, executes together)



Synchronization between work-items possible only within work-groups: barriers and memory fences

Cannot synchronize between work-groups within a kernel

Synchronization: when multiple units of execution (e.g. work-items) are brought to a known point in their execution. Most common example is a barrier … i.e. all units of execution "in scope" arrive at the barrier before any proceed.

Work-Item Synchronization

Ensure correct order of memory operations to local or global memory (with flushes or queuing a memory fence)

- Within a work-group
  `void barrier()`
  - Takes optional flags
    CLK_LOCAL_MEM_FENCE and/or CLK_GLOBAL_MEM_FENCE
  - A work-item that encounters a barrier() will wait until ALL work-items in its work-group reach the barrier()
  - Corollary: If a barrier() is inside a branch, then the branch must be taken by either:
    - ALL work-items in the work-group, OR
    - NO work-item in the work-group

- Across work-groups
  - No guarantees as to where and when a particular work-group will be executed relative to another work-group
  - Cannot exchange data, or have barrier-like synchronization between two different work-groups! (Critical issue!)
  - Only solution: finish the kernel and start another

Where might we need synchronization?

- Consider a reduction … reduce a set of numbers to a single value
  - E.g. find sum of all elements in an array
- Sequential code

```
int reduce(int Ndim, int *A)
{
  int sum = 0;
  for (int i = 0; i < Ndim; i++)
    sum += A[i];
  return sum;
}
```

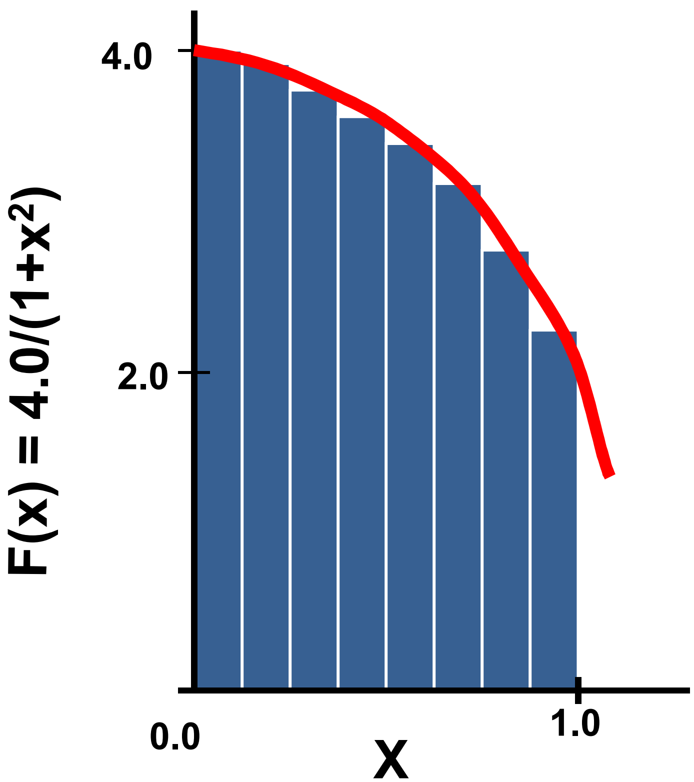
Simple parallel reduction

- A reduction can be carried out in three steps:
  1. Each work-item sums its private values into a local array indexed by the work-item's local id
  2. When all the work-items have finished, one work-item sums the local array into an element of a global array (indexed by work-group id).
  3. When all work-groups have finished the kernel execution, the global array is summed on the host.

- Note: this is a simple reduction that is straightforward to implement. More efficient reductions do the work-group sums in parallel on the device rather than on the host. These more scalable reductions are considerably more complicated to implement.

A simple program that uses a reduction

Numerical Integration



Mathematically, we know that we can approximate the integral as a sum of rectangles.

Each rectangle has width and height at the middle of interval.

Numerical integration source code

The serial Pi program

```
static long num_steps = 100000;
double step;
void main()
{
  int i; double x, pi, sum = 0.0;

  step = 1.0/(double) num_steps;

  for (i = 0; i < num_steps; i++) {
    x = (i+0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
  }
  pi = step * sum;
}
```

Exercise 9: The Pi program

- Goal:
  - To understand synchronization between work-items in the OpenCL C kernel programming language
- Procedure:
  - Start with the provided serial program to estimate Pi through numerical integration
  - Write a kernel and host program to compute the numerical integral using OpenCL
  - Note: You will need to implement a reduction
- Expected output:
  - Output result plus an estimate of the error in the result
  - Report the runtime

Hint: you will want each work-item to do many iterations of the loop, i.e. don't create one work-item per loop iteration. To do so would make the reduction so costly that performance would be terrible.