CUDA Programming

By Matthew Zeiler

Big Data, Large Scale Machine Learning

John Langford and Yann LeCun
New York University

GPU Programming

- OpenCL
  - Works on CPU and GPU
  - Supported by AMD, Intel, Nvidia, and others
- CUDA
  - Works on Nvidia GPUs only
  - Wide developer adoption

Why CUDA

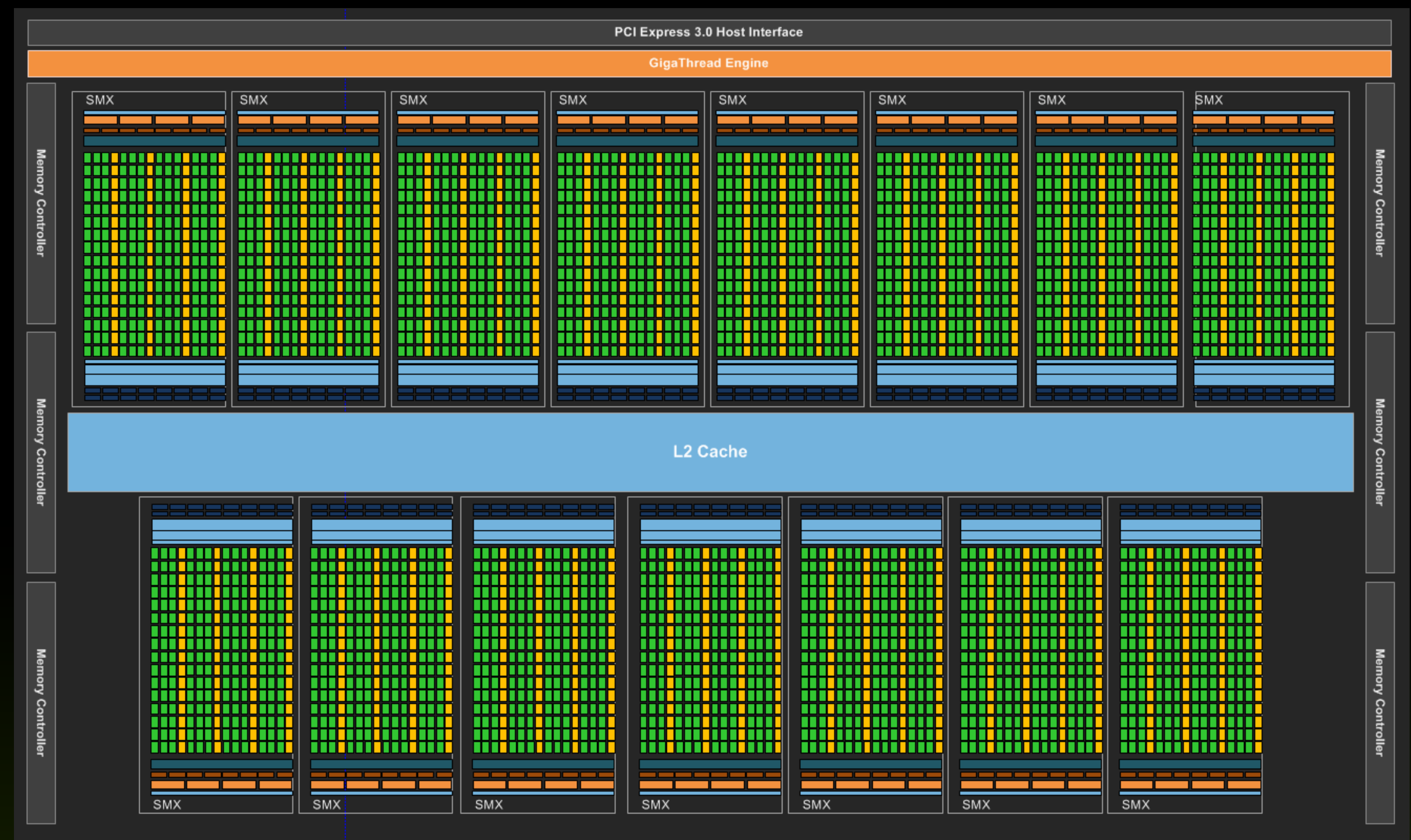
- Pros:
  - Massively parallel architecture
  - Immense speedups over CPUs
- Cons:
  - Different programming style
  - Very difficult to get optimal performance

Latest Architecture

Kepler GK110 Block Diagram

Architecture

- 7.1B Transistors
- 15 SMX units
- > 1 TFLOP FP64
- 1.5 MB L2 Cache
- 384-bit GDDR5
- PCI Express Gen3

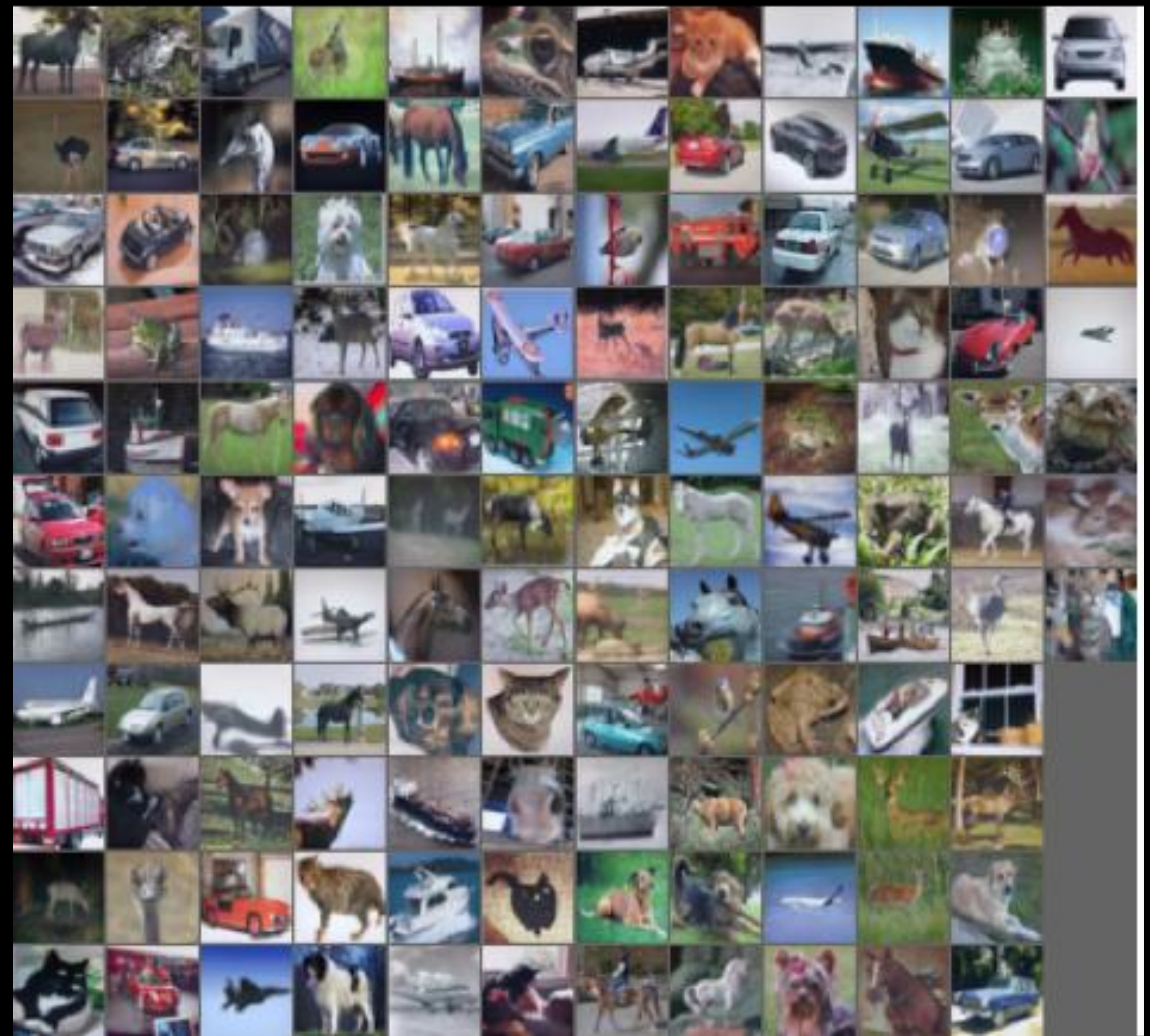


SM(X)

DEMO

- CIFAR-10 Image Classification

- 10 Classes:
  - airplane
  - automobile
  - bird
  - cat
  - deer
  - dog
  - frog
  - horse
  - ship
  - truck



Intro to CUDA

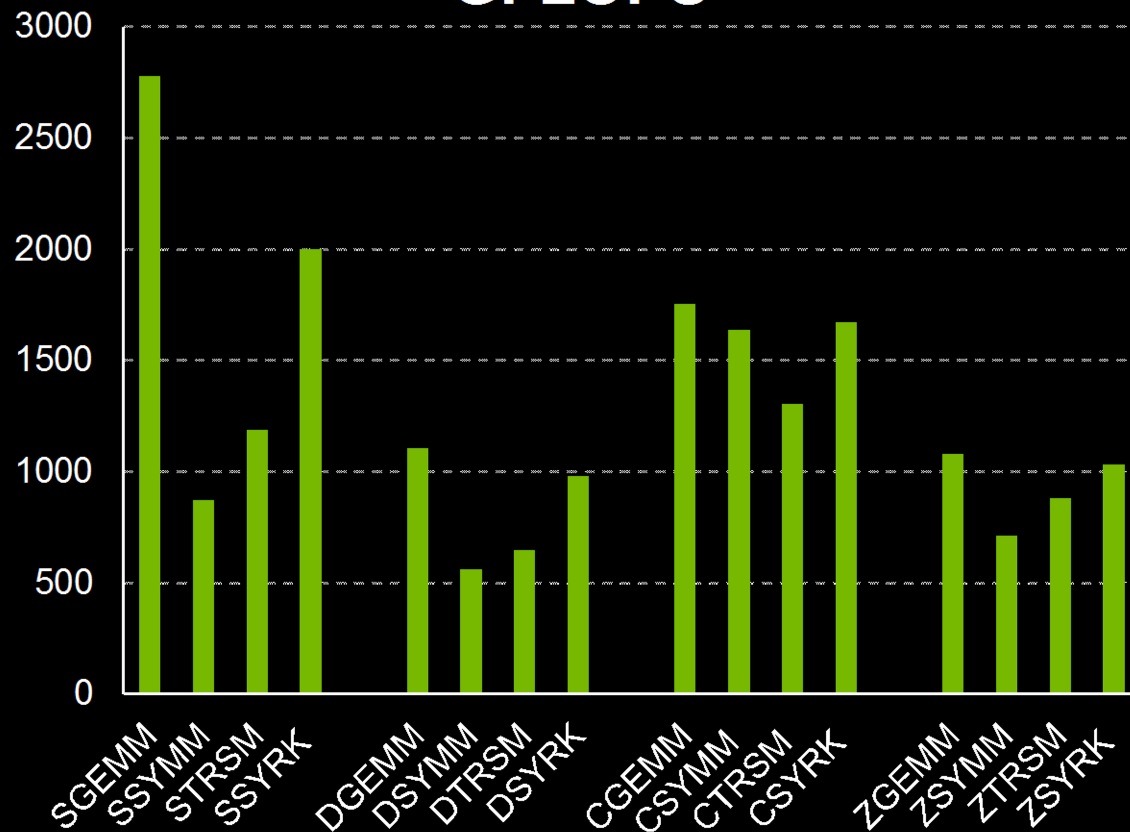
- See Slides

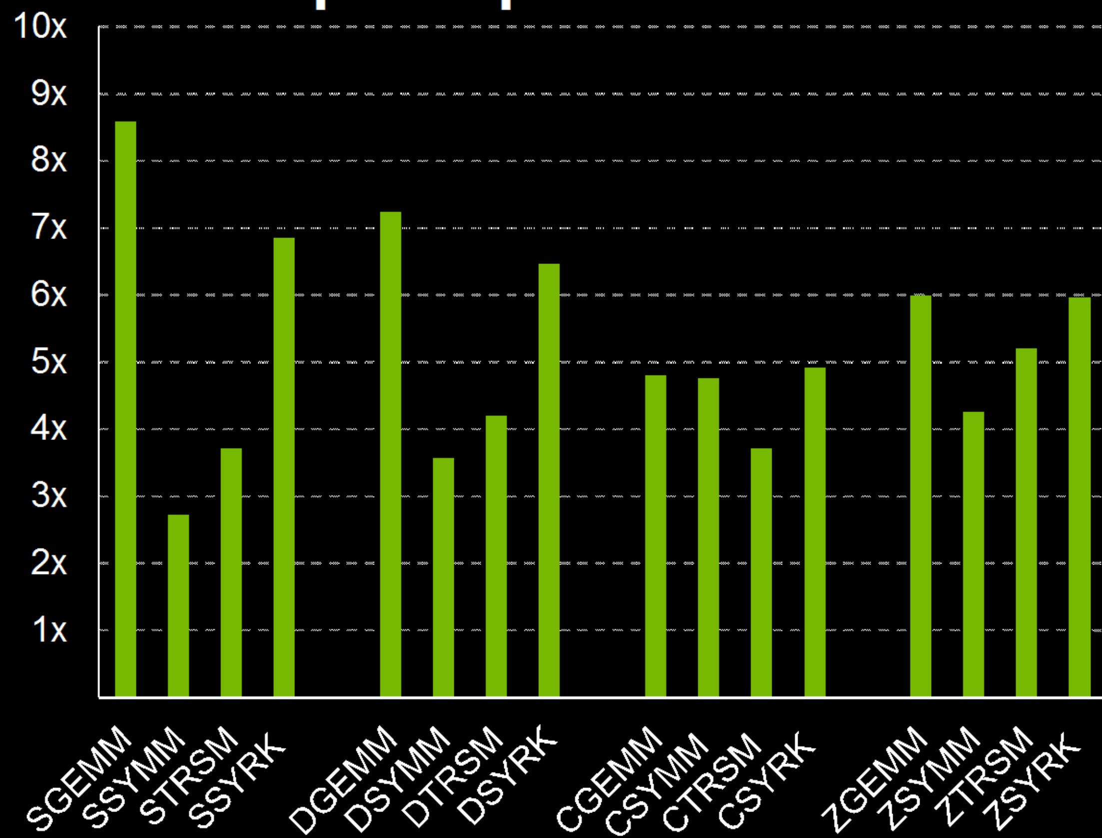
Fast Math

- Use special function units in hardware
- Replace exp(), cos(), etc. with:
  - __expf(), __cosf(), etc.
- OR, compile with “–use_fast_math” flag

cuBLAS

cuBLAS: >1 TFLOPS double-precision





- cuBLAS 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz

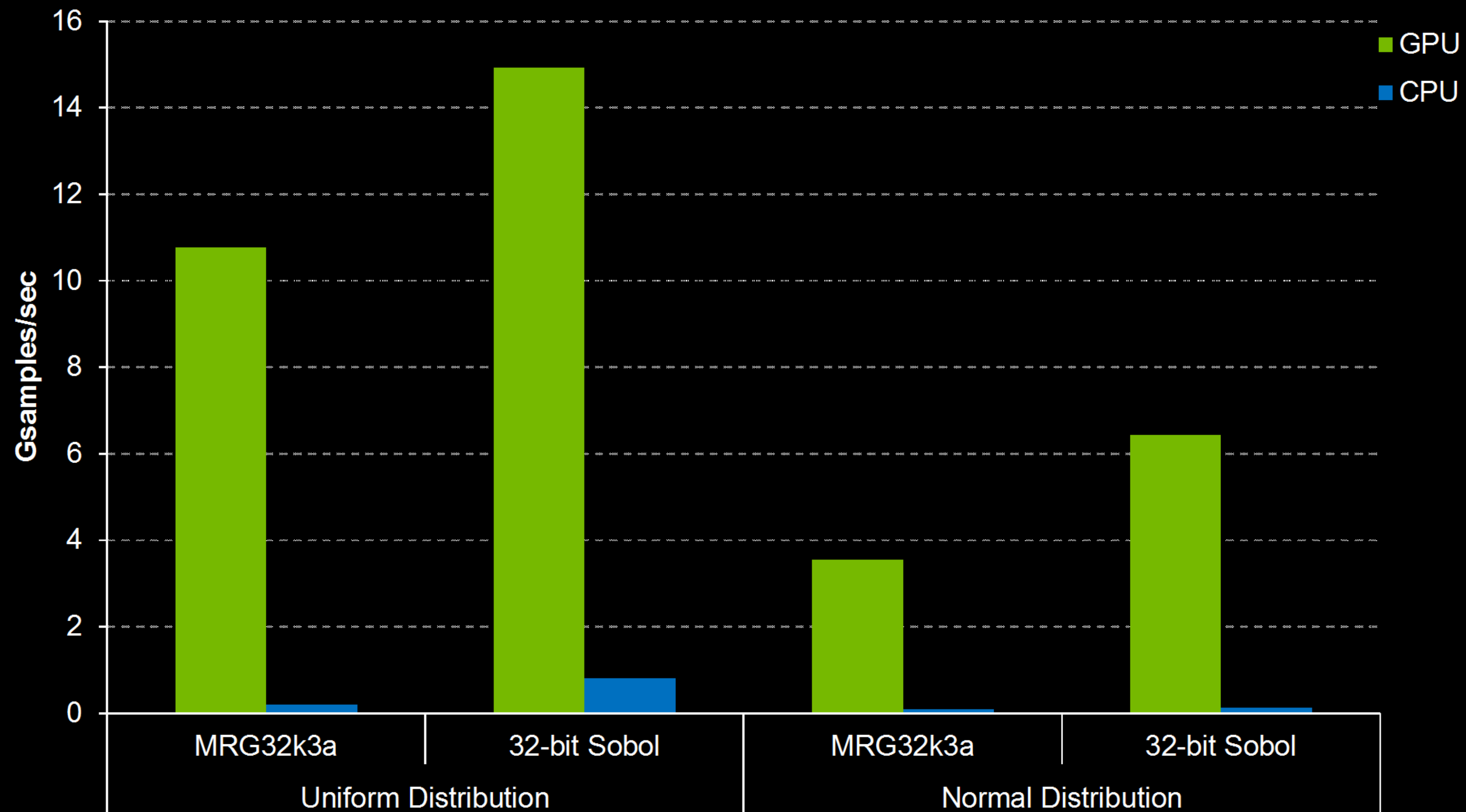
Performance may vary based on OS version and motherboard configuration

Cuda Performance Report, Jan25, 2013

cuRAND

cuRAND Performance

Double Precision RNGs



- cuRAND 5.0 on K20X, input and output data on device
- MKL 10.2.3 on Intel SandyBridge E5-2687W @ 3.10GHz

Performance may vary based on OS version and motherboard configuration

Cuda Performance Report, Jan25, 2013

Max Pooling CPU

```
void MaxUpdates(float * unpooled, float * pooled, float * inds,
                float * null1, float * null2, float * null3, float * null4, float * null5,
                int n, int num_threads, int numCases, int IM_FIRST,
                int zSizex,      int zSizey,    int zSizek,
                int pSizex,      int pSizey,    int pSizek,
                int poolx,       int pooly,     int poolk,
                int shiftx,      int shifty,     int shiftk,
                const float scaleTargets, const float scaleOutputs,
                const float* add_args)
{
  // Get the linear index into pooled.
  for (int pooledIndex = 0; pooledIndex < n; pooledIndex++) {

    // Make sure the index to the last elemnt of the pool region
    if ((pooledIndex) < n){
      // Get the location in the pooled region and the unpooled region top left elemtn
      unpooledRegion pReg = pooledMapIndex2unpooledRegion(pooledIndex, numCases, IM_FIRST,
                                                         zSizex, zSizey, pSizex, pSizey,
                                                         poolx, pooly, shiftx, shifty);
      // Also need an index to the current element of the inptu pool region.
      unsigned int inIndex = 0;
      float mx = 0;
      float mind = 0;
      float temp = 0;
      // Loop over the pooling region in the image.
      for(int y=0;y<pReg.valpooly;y++){
        // Check we are in boundary in y dimension.
        for(int x=0;x<pReg.valpoolx;x++){
          // Get current image element's linear index in the input.
          inIndex = pReg.topLeftIndex+(y*zSizex+x)*pReg.multSkip;
          temp = (unpooled[inIndex]);
          if(fabsf(temp)>fabsf(mx)){
            mx = temp;
            mind = x+y*poolx;
          }
        }
      }
      if (scaleTargets == 0) {
        pooled[pooledIndex] = scaleOutputs * mx;
      } else {
        pooled[pooledIndex] = scaleOutputs * mx + scaleTargets * pooled[pooledIndex];
      }
      inds[pooledIndex] = mind;
    }
  }
}
```

Max Pooling GPU

```
__global__ void MaxUpdates(float * unpooled, float * pooled,
                           float * inds,
                           int n, int offset,  int numCases, int IM_FIRST,
                           int zSizex,      int zSizey,
                           int pSizex,      int pSizey,
                           int poolx,       int pooly,
                           int shiftx,      int shifty,
                           const float scaleTargets, const float scaleOutputs)
{
  // Get the linear intext into odata (do not touch this).
  unsigned int pooledIndex = blockIdx.x * blockDim.x + threadIdx.x+offset;

  // Make sure the index to the last elemnt of the pool region
  if ((pooledIndex-offset) < n){
    // Get the location in the pooled region and the unpooled region top left elemtn
    unpooledRegion pReg = pooledMapIndex2unpooledRegion(pooledIndex, numCases, IM_FIRST,
                                                        zSizex, zSizey, pSizex, pSizey,
                                                        poolx, pooly, shiftx, shifty);
    // Also need an index to the current element of the inptu pool region.
    unsigned int inIndex = 0;
    float mx = 0;
    float mind = 0;
    float temp = 0;
    // Loop over the pooling region in the image.
    for(int y=0;y<pReg.valpooly;y++){
      // Check we are in boundary in y dimension.
      for(int x=0;x<pReg.valpoolx;x++){
        // Get current image element's linear index in the input.
        inIndex = pReg.topLeftIndex+(y*zSizex+x)*pReg.multSkip;
        temp = (unpooled[inIndex]);
        if(fabsf(temp)>fabsf(mx)){
          mx = temp;
          mind = x+y*poolx;
        }
      }
    }
    if (scaleTargets == 0) {
      pooled[pooledIndex] = scaleOutputs * mx;
    } else {
      pooled[pooledIndex] = scaleOutputs * mx + scaleTargets * pooled[pooledIndex];
    }
    inds[pooledIndex] = mind;
  }
}
```