Convolution

Mark Greenstreet and Ian M. Mitchell

CpSc 418 – March 19, 2018

Table of Contents

Outline

Convolution in One Dimension

- Assume input array $\{x_i\}_{i=0}^{i=n-1}$ and output array $\{y_i\}_{i=0}^{i=n-1}$.
- Each y_i is a weighted sum of x_i for $i \in [i-k, i+k]$.
- Mathematically

$$y_i = \sum_{\ell=-k}^{\ell=+k} w_\ell x_{i+\ell} \qquad \text{for all } i = 0, \ldots, n$$

  where $\{w_\ell\}_{\ell=-k}^{\ell=+k}$ and k are given.
  - Weights $\{w_\ell\}$ called the convolution “kernel”, “mask” or “stencil”.
  - Mask contains $2k+1$ elements with $k \ll n$.
  - Value k called “half-width” or (confusingly) “width”.
- Graphically, see K&H figures 7.1 and 7.2.
  - The input $\{x_i\}$ is stored in `N`.
  - The output $\{y_i\}$ is stored in `P`.
  - The mask $\{w_\ell\}$ is stored in `M`.
- Need to handle the cases when $i+\ell < 0$ or $i+\ell >= n$.
  - Typically substitute $x_i = 0$ for these values of i.
  - See K&H figure 7.3.

Convolution in Higher Dimensions

In two dimensions:

- Mathematically

$$y_{i,j} = \sum_{\ell_1=-k_1}^{\ell_1=+k_1} \sum_{\ell_2=-k_2}^{\ell_2=+k_2} w_{\ell_1,\ell_2} x_{i+\ell_1,j+\ell_2}$$

- See K&H figures 7.4 and 7.5.

Conceptually easy to extend to higher dimension.

- Number of weights grows quickly with dimension.

Implementation

Basic CPU implementation:

- Double iteration over output values and mask entries.

```
int stencil_half_width = stencil_width / 2;
for(int i = 0; i < n; i++) {
  float sum = 0.0;
  for(int k = 0; k < stencil_width; k++) {
    int index = i + k - stencil_half_width;
    if((index >= 0) && (index < n))
      sum += stencil[k] * data_in[index];
  }
  data_out[i] = sum;
}
```

Basic GPU implementation:

- Assign one thread to each output value.
- Iterate over mask entries.
- see K&H figure 7.6.

CGMA of Basic Convolution Implementation

Consider 1D convolution and assume data size *n* is much bigger than mask size *k* (to justify ignoring boundary behaviour).

- Examine innermost line in K&H figure 7.8:
  `Pvalue += N[N_start_point + j] * M[j];`
- How many elements are loaded from memory?
- How many operations are performed?
- What is the CGMA?

How can we do better?

- How many threads read each element of `N`?
- How many threads read each element of `M`?

Outline

1. Convolution: Basics
   - Implementation
   - Analysis
2. Convolution: Reducing Global Memory Demand
   - Tiling
   - Constant Memory
   - Analysis
3. Convolution: Example

Shared Memory

Remember from GPU Performance Idiosyncracies:

- Conceptually: A small (48KB in all production CCs) amount of memory shared between the threads in a block.
  - Slower than registers but much faster than global memory.
  - Must be explicitly allocated and loaded by kernels.
  - Some memory coherence guarantees; for example, after `__syncthreads()`.
  - Typical uses: Rapid access to selected array data and/or communication of data between threads.
- Practically: A small (96KB in CC 6.1) amount of memory with low latency and high bandwidth available to each SM.
  - Same latency, same (or higher) bandwidth as L1 cache.
  - Divided into 32 banks, each of which can serve a load or store every cycle.
  - For maximum efficiency: Threads in a warp either all access the same location in a bank or access locations in different banks.

Tiling

A common GPU programming pattern for data that is accessed by multiple threads:

1. Divide the full set of data into "tiles".
2. Load a small number of tiles into shared memory.
   - Number of tiles required at one time is determined by the algorithm.
   - Size of tiles is determined by the number required and size of shared memory.
   - Choosing smaller blocks allows more shared memory / thread but often requires more overhead.
3. Do as much work as possible on those tiles.
4. Save results.
5. Repeat from step 2 until all data is processed.

Examples

- Matrix multiply K&H section 4.5.
- Convolution K&H section 7.4.

Tiled Convolution

Convolution requires a very simple version of tiling.

- Each block loads a single tile of the input data into shared memory.
- Tile should contain "ghost" or "halo" entries: input data stretching k elements on each side of the desired output elements.
- See K&H figure 7.10 for visualization of tiles and halo elements.
- See K&H figure 7.11 for kernel implementation.
- Can be extended to higher dimensions.
  - Shared memory size may severely limit tile size in higher dimensions, and halo overhead becomes significant.
  - For example, in 2D tile size m yields m^2 interior elements and $\sim 4mk$ halo elements.
  - For $k = 10$ and $m = 32$ (so $m^2 = 1024$ is maximum number of threads in a block), that requires $\sim$ 9KB shared memory.

Constant Memory

What about the mask weights?

- We could explicitly load them into shared memory, but:
  - The amount of shared memory is very limited.
  - Each block would have to load weights into its own shared memory.
- Instead we will use GPU's "constant memory".
  - Another special category of memory (limited to 64 KB)
  - Explicitly loaded by the host.
  - Kernel code cannot modify the values.
  - Logically resides in global memory.
  - Specialized constant cache hardware (limited to 10 KB / SM) allows fast access if all threads request exactly the same memory location each cycle.

Constant Memory for Convolution Weights

Implementation:

- Host declares global array with `__constant__` keyword.
- Host loads weights using `cudaMemcpyToSymbol()` (instead of `cudaMemcpy()`).
- Kernel accesses weights as a global array.
- See K&H section 7.3 for example code.

Note: Constant memory cannot change during kernel execution.

- Tiling of mask weights could only be done by launching a new kernel.
- Fortunately, the mask is usually small enough to fit in constant memory.

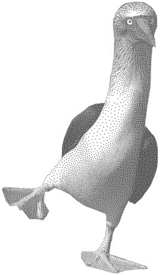
CGMA of Tiled Convolution with Mask in Constant Memory

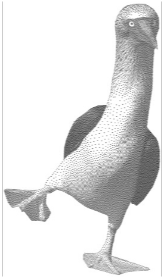
Consider 1D convolution and assume data size n is much bigger than mask size k (to justify ignoring boundary behaviour).

- Tile size $t > k$ but not $t \gg k$, so we cannot ignore halo cell overhead.
- Consider a single tile.
  - We complete t output elements.
  - How many input elements are loaded from global memory?
  - How many mask weights are loaded from global memory?
  - How many output elements are written to global memory?
  - How many floating point operations are performed?
  - What is the CGMA?

Outline

Example: Image Convolution





Taken from 2016W2 Homework 5.

- Work with greyscale images (stored as `.ppm`).
- Convolution mask generated by a Gaussian curve.
  - Implements a typical blurring effect.
- Original assignment included
  - 1D horizontal convolution, 1D vertical convolution, 2D box convolution (implemented by sequential horizontal and then vertical).
  - Basic and tiled versions of each.
- Added for this demo: Basic CPU version of each.