Dis1a Week7

Karl

May 13, 2022

Correctness

Lab 3

» Do you have a misunderstanding of the CUDA programming model?

- Parallelism is achieved through your thread count * block count
- These can replace for loops, but threads can still do for loops
- Shared mem. is tied to the block level
- Threads in a block aren't synchronized until you do so explicitly
- Threads should roughly evenly collaboratively load into shared
- There are limits on everything
  - Memory, total threads per blocks, total blocks per SM

Correctness

- No misunderstanding of CUDA, but mistaken application logic
  - Add bias then RELU, or RELU then bias?
  - Does final output tile match in size to an input size?

Correctness

Lab 3

- Correct understanding of CUDA and application logic
- Reason through your implementation
  - Likely have a simple bug
    - Length, dim or indexing
    - Take advantage of x, y, z dims to make things easier
- If above fails visualize the outputs so you can detect patterns which will lead you to where the bug is

Correctness


```
template <typename T>
void print_image_3D(const T *inimg1d, const GridCoord lengths,
                    const T bkg_thresh = 0) {
  auto total_count = coord_prod_accum(lengths);
  cout << "Print image 3D:\n";
  for (int zi = 0; zi < lengths[2]; zi++) {
    cout << "y | Z=" << zi << '\n';
    for (int xi = 0; xi < 2 * lengths[0] + 4; xi++) {
      cout << "-";
    }
    cout << '\n';
    for (int yi = 0; yi < lengths[1]; yi++) {
      cout << yi << " | ";
      for (int xi = 0; xi < lengths[0]; xi++) {
        auto index = coord_to_id(GridCoord(xi, yi, zi), lengths);
        assertm(index < total_count, "requested index is out of bounds");
        auto val = inimg1d[index];
        if (val > bkg_thresh) {
          cout << val << " ";
        } else {
          cout << "- ";
        }
      }
      cout << '\n';
    }
    cout << '\n';
  }
}
```

Correctness

- Other ways to simplify multiple ops of the CNN kernel?

Performance

- If an instruction has 2 loads and 1 store which ones should have the fastest access?
- What structure has more reuse?

Performance

- If an instruction has 2 loads and 1 store which ones should have the fastest access?
  - All should be available at the same latency
  - Therefore you need to tile
- What structure has more reuse?
  - Each 2D kernel is small, but it gets convolved across

Memory Hierarchy

Dis1a Week7

Karl

Lab 3

- Thread has local memory
- Block has larger shared memory

Type	Descriptive name	Closest old term outside of GPUs	Official CUDA/NVIDIA GPU term	Short explanation
Program abstractions	Vectorizable Loop	Vectorizable Loop	Grid	A vectorizable loop, executed on the GPU, made up of one or more Thread Blocks (bodies of vectorized loop) that can execute in parallel
	Body of Vectorized Loop	Body of a (Strip-Mined) Vectorized Loop	Thread Block	A vectorized loop executed on a multithreaded SIMD Processor, made up of one or more threads of SIMD instructions. They can communicate via local memory
	Sequence of SIMD Lane Operations	One iteration of a Scalar Loop	CUDA Thread	A vertical cut of a thread of SIMD instructions corresponding to one element executed by one SIMD Lane. Result is stored depending on mask and predicate register
Machine object	A Thread of SIMD Instructions	Thread of Vector Instructions	Warp	A traditional thread, but it only contains SIMD instructions that are executed on a multithreaded SIMD Processor. Results stored depending on a per-element mask
	SIMD Instruction	Vector Instruction	PTX Instruction	A single SIMD instruction executed across SIMD Lanes
Processing hardware	Multithreaded SIMD Processor	(Multithreaded) Vector Processor	Streaming Multiprocessor	A multithreaded SIMD Processor executes threads of SIMD instructions, independent of other SIMD Processors
	Thread Block Scheduler	Scalar Processor	Giga Thread Engine	Assigns multiple Thread Blocks (bodies of vectorized loop) to multithreaded SIMD Processors
	SIMD Thread Scheduler	Thread Scheduler in a Multithreaded CPU	Warp Scheduler	Hardware unit that schedules and issues threads of SIMD instructions when they are ready to execute; includes a scoreboard to track SIMD Thread execution
	SIMD Lane	Vector Lane	Thread Processor	A SIMD Lane executes the operations in a thread of SIMD instructions on a single element. Results stored depending on mask
Memory hardware	GPU Memory	Main Memory	Global Memory	DRAM memory accessible by all multithreaded SIMD Processors in a GPU
	Private Memory	Stack or Thread Local Storage (OS)	Local Memory	Portion of DRAM memory private to each SIMD Lane
	Local Memory	Local Memory	Shared Memory	Fast local SRAM for one multithreaded SIMD Processor, unavailable to other SIMD Processors
	SIMD Lane Registers	Vector Lane Registers	Thread Processor Registers	Registers in a single SIMD Lane allocated across a full Thread Block (body of vectorized loop)

Figure 4.12 Quick guide to GPU terms used in this chapter. We use the first column for hardware terms. Four groups cluster these 11 terms. From top to bottom: program abstractions, machine objects, processing hardware, and memory hardware. Figure 4.21 on page 312 associates vector terms with the closest terms here, and Figure 4.24 on page 317 and Figure 4.25 on page 318 reveal the official CUDA/NVIDIA and AMD terms and definitions along with the terms used by OpenCL.

Memory Hierarchy

Dis1a Week7

Karl

Lab 3

- Thread has local memory
  - Subtile at the thread
- Block has larger shared memory
  - Subtile at the block

Type	Descriptive name	Closest old term outside of GPUs	Official CUDA/NVIDIA GPU term	Short explanation
Program abstractions	Vectorizable Loop	Vectorizable Loop	Grid	A vectorizable loop, executed on the GPU, made up of one or more Thread Blocks (bodies of vectorized loop) that can execute in parallel
	Body of Vectorized Loop	Body of a (Strip-Mined) Vectorized Loop	Thread Block	A vectorized loop executed on a multithreaded SIMD Processor, made up of one or more threads of SIMD instructions. They can communicate via local memory
	Sequence of SIMD Lane Operations	One iteration of a Scalar Loop	CUDA Thread	A vertical cut of a thread of SIMD instructions corresponding to one element executed by one SIMD Lane. Result is stored depending on mask and predicate register
Machine object	A Thread of SIMD Instructions	Thread of Vector Instructions	Warp	A traditional thread, but it only contains SIMD instructions that are executed on a multithreaded SIMD Processor. Results stored depending on a per-element mask
	SIMD Instruction	Vector Instruction	PTX Instruction	A single SIMD instruction executed across SIMD Lanes
Processing hardware	Multithreaded SIMD Processor	(Multithreaded) Vector Processor	Streaming Multiprocessor	A multithreaded SIMD Processor executes threads of SIMD instructions, independent of other SIMD Processors
	Thread Block Scheduler	Scalar Processor	Giga Thread Engine	Assigns multiple Thread Blocks (bodies of vectorized loop) to multithreaded SIMD Processors
	SIMD Thread Scheduler	Thread Scheduler in a Multithreaded CPU	Warp Scheduler	Hardware unit that schedules and issues threads of SIMD instructions when they are ready to execute; includes a scoreboard to track SIMD Thread execution
	SIMD Lane	Vector Lane	Thread Processor	A SIMD Lane executes the operations in a thread of SIMD instructions on a single element. Results stored depending on mask
Memory hardware	GPU Memory	Main Memory	Global Memory	DRAM memory accessible by all multithreaded SIMD Processors in a GPU
	Private Memory	Stack or Thread Local Storage (OS)	Local Memory	Portion of DRAM memory private to each SIMD Lane
	Local Memory	Local Memory	Shared Memory	Fast local SRAM for one multithreaded SIMD Processor, unavailable to other SIMD Processors
	SIMD Lane Registers	Vector Lane Registers	Thread Processor Registers	Registers in a single SIMD Lane allocated across a full Thread Block (body of vectorized loop)

Figure 4.12 Quick guide to GPU terms used in this chapter. We use the first column for hardware terms. Four groups cluster these 11 terms. From top to bottom: program abstractions, machine objects, processing hardware, and memory hardware. Figure 4.21 on page 312 associates vector terms with the closest terms here, and Figure 4.24 on page 317 and Figure 4.25 on page 318 reveal the official CUDA/NVIDIA and AMD terms and definitions along with the terms used by OpenCL.

Performance

Lab 3

- Rely on local and shared memory
  - L1 and L2 are opaque
- Does your shared memory strategy reduce the number of repeated accesses to global memory?
- Does your strategy introduce a lot of branching control paths where the behavior of threads differ highly from one another? '#pragma unroll'
- Do you go over a limit of memory or concurrency?
- The total threads in a thread block should ideally be a multiple of the warp size
- The total number of blocks you should decide based off factors we've discussed like SM count, and SM concurrency and memory limitations

Performance

- Example:
  - M60 has 96 KB total shared memory per SM
  - Limit of 48 KB shared memory per thread block.
  - If 48 KB of shared memory per thread block
    - Only 2 blocks can run per SM
    - However, on M60, 32 threads blocks can execute concurrently per SM
    - You would loose significant parallelism if you used 48 KB of shared/thread block

Resources

Dis1a Week7

Karl

Lab 3

- https://docs.nvidia.com/cuda/maxwell-tuning-guide/index.html
- https://www.microway.com/knowledge-center-articles/in-depth-comparison-of-nvidia-tesla-maxwell-gpu-accel
- https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
- https://www.microway.com/download/whitepaper/NVIDIA_Maxwell_GM204_Architecture_Whitepaper.pdf