CUDA: Performance Considerations

Mark Greenstreet

CpSc 418 – November 26, 2018

- Occupancy
- Thread Divergence
- Instruction Mix

Occupancy

- We want lots of threads running on each SM. How do we get this?
- To keep every SM busy, the number of blocks should be greater than the number of SMs.
  - An SM can have 2048 threads, but a block can have at most 1024 threads.
  - We want at least two blocks per SM. More is better.
- Limits on occupancy
  - At most 8 blocks per SM.
  - 2^{16} registers/SM. Use
    `nvcc -O3 -c --ptxas-options -v examples.cu`
  - 96Kbytes of shared memory per SM. At most 48Kbytes/block.
  - Check out the occupancy calculator:
    CUDA_Occupancy_calculator.xls

Predicated Execution

- What about `if` (or `for`)?
- When a warp executes the `if`, all threads execute it?
  - What if some threads take the then-branch and others take the else-branch?
- Predicated execution.
  - Each thread keeps a bit saying whether it's a then-thread or an else-thread.
  - The GPU fetches instructions from the then and else clauses, and marks them as then-instructions or else-instructions
  - The then-threads execute then-instructions and ignore else-instructions.
    - The else-instructions are still fetched and dispatched for the SPs that need them.
    - They are treated as NOPs by the SPs for then-threads, but they take up excution time.
  - Likewise, the else-threads execute else-instructoins and ignore then-instructions.
- Predication handles multiple levels of control flow nesting.

Thread Divergence and Performance

- If a warp consists of a mix of then-threads and else-threads
  - Then instructions are are fetched that are ignored by some pipelines.
  - This results in performance loss.
  - Simple estimate: if half of instructions then-instructions and half are else-instructions then 50% efficiency when divergent.
- It can be worse
  - Nested if-then-else statements: can divide threads into more than two groups.
  - For-loops with thread-dependent bounds: e.g. reduce.
  - While-loops.
- It can be better
  - If all threads in a warp are then-threads, then the else-instructions aren't fetched.
  - Else if all the threads are else-threads, then the then-instructions aren't fetched.

Thread Divergence and `__syncthreads()`

- `__syncthreads()` requires all threads in the block to reach the call along convergent paths.
- E.g. a `__syncthreads()` in a then-clause for one thread can't match a `__synchthreads()` in a else-clause for another thread.
- Likewise, if there is a `__syncthreads()` in the body of a for-loop, all threads must reach it on the same iteration.
- Getting this wrong leads to deadlock: the block hangs forever at the `__syncthreads()`

Instruction Mix

- We measure our program performance in terms of the critical, unavoidable operations
  - Typically “floating point operations” for matrix-multiplication or other scientific computing applications.
  - Often main memory accesses for sorting, or other data-intensive applications.
- But, the program does other operations as well
  - This is where you see me counting instructions on my fingers during lecture.
  - Optimizing performance can involve minimizing this overhead:
    - Good algorithm design.
    - Memory access optimization.
    - Loop unrolling

Bigger Kernels

```
__global__myKernel(...)  {
   do something
}
```

Unless *do something* is big, kernel launch takes most of the time.

- We can launch a big-grid
  - If we have a huge number of array elements than each need a small amount of work, this can be a good idea.
  - BUT we're likely to create a memory-bound problem.
- Or, we can make each thread do many somethings.

```
__global__myKernel(int m, ...)  {
   for(int i = 0; i < m; i++)
      do something
}
```

Loop Limitations

- It takes two or three instructions per loop iteration to manage the loop:
  - One to update the loop index
  - One or two to check the loop bounds and branch.
  - If *do something* is only three or four instructions, then 40-50% of the execution time is for loop management.
- If each iteration of *do something* depends on the previous one
  - Then the long latency of the SP pipelines can limit performance.
  - Even if we have 64 warps running.

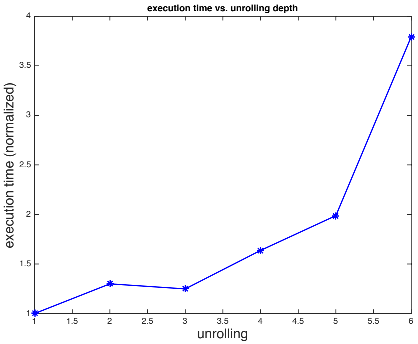
Loop Unrolling

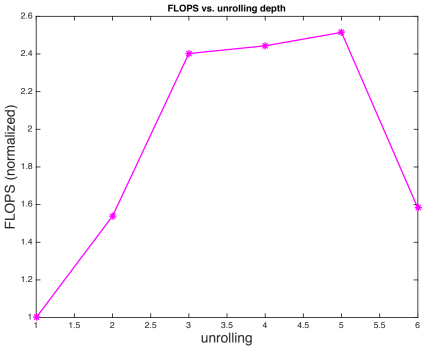
- Have each loop iteration perform multiple copies of the loop body

```
__global__myKernel(int m, ...)  {
  for(int i = 0; i < m; i += 4) {
    do something 1
    do something 2
    do something 3
    do something 4
  }
}
```

- More "real work" for each time the loop management code is executed.
- Need to make sure that `m` is a multiple of four, or handle end-cases separately.
- Often, we need more registers.

Unrolling – the plots





This example is from 2015W2 HW3, Q1.

Examples

You choose:

- Merge-sort vs. bitonic sort on a GPU.
- Reduce on a GPU.
- Homework 5.
- Homework 6
  - I'll post review questions.
  - But there won't be a graded HW6.

Preview

November 28: CUDA (and other) Examples
November 30: Review
Dec 3: Final Exam review session (piazza poll)
Dec 4: Final Exam – 8:38am (ouch!)