Hardware/Software Co-Design

Performance Considerations – III

Miaoqing Huang
University of Arkansas
Fall 2011
Outline

1. Data Prefetching
2. Loop Unrolling
3. Kernel Launch Overhead
Outline

1. Data Prefetching
2. Loop Unrolling
3. Kernel Launch Overhead
Data Prefetching

- Global memory access is the biggest bottleneck in current GPU
  
  Solution: move data from global memory to shared memory and then proceed from there
Data Prefetching

- Global memory access is the biggest bottleneck in current GPU
  - Solution: move data from global memory to shared memory and then proceed from there
  - Question: is this mechanism sufficient for all cases?

```c
Loop {
    Load current tile to shared memory;
    __syncthreads();
    Compute current tile;
    __syncthreads();
}
```
Data Prefetching

- Global memory access is the biggest bottleneck in current GPU
  - Solution: move data from global memory to shared memory and then proceed from there
  - Question: is this mechanism sufficient for all cases?

```c
Loop {
    Load current tile to shared memory;
    __syncthreads();
    Compute current tile;
    __syncthreads();
}
```

Diagram showing the execution of multiple warps in parallel, with different stages of computation (Compute), synchronization (Idle, Stall).
Data Prefetching – using double buffers

- One could double buffer the computation, getting better instruction mix within each thread

Load the first tile from global memory into registers;
__syncthreads();
Loop {
    Deposit tile from registers to shared memory;
    __syncthreads();
    Load next tile from global memory into registers
    Compute current tile on shared memory;
    __syncthreads();
}
Matrix Multiplication Using Multiple Blocks with Tile

- Deposit Red tile from register into shared memory
- Syncthreads
- Load Pink tile into register
- Compute Red tile
- Syncthreads
- Deposit Pink tile into shared memory
- ....
Outline

1. Data Prefetching
2. Loop Unrolling
3. Kernel Launch Overhead
Limited Processing Bandwidth of An SM

```c
for (int k = 0; k < BLOCK_SIZE; ++k) {
    Pvalue += Ms[ty][k] * Ns[k][tx];
}
```

- How many instructions are required to be carried out in each iteration?
Limited Processing Bandwidth of An SM

How many instructions are required to be carried out in each iteration?

- Two floating-point arithmetic instructions
- One loop branch instruction
- Two address arithmetic instruction
- One loop counter increment instruction

```c
for (int k = 0; k < BLOCK_SIZE; ++k) {
    Pvalue += Ms[ty][k] * Ns[k][tx];
}
```
Limited Processing Bandwidth of An SM

```c
for (int k = 0; k < BLOCK_SIZE; ++k) {
    Pvalue += Ms[ty][k] * Ns[k][tx];
}
```

- How many instructions are required to be carried out in each iteration?
  - Two floating-point arithmetic instructions
  - One loop branch instruction
  - Two address arithmetic instruction
  - One loop counter increment instruction
  - Only $\frac{1}{3}$ of the instructions executed are for real computation!!!
Loop Unrolling

// Assume BLOCK_SIZE = 16
Pvalue = Ms[ty][0] * Ns[0][tx] + Ms[ty][1] * Ns[1][tx] + ...
+ Ms[ty][15] * Ns[15][tx];

- Loop branch instructions → gone
- Loop counter increment instructions → gone
- Address arithmetic instructions → gone
  - Indices are constants
  - Compiler is able to eliminate address arithmetic instructions
- Only floating-point arithmetic instructions are still there
Loop Unrolling

// Assume BLOCK_SIZE = 16
Pvalue = Ms[ty][0] * Ns[0][tx] + Ms[ty][1] * Ns[1][tx] + ...
+ Ms[ty][15] * Ns[15][tx];

- Loop branch instructions → gone
- Loop counter increment instructions → gone
- Address arithmetic instructions → gone
  - Indices are constants
  - Compiler is able to eliminate address arithmetic instructions
- Only floating-point arithmetic instructions are still there
  - Close to peak performance!!!
Kernel Launch Overhead

- Kernel launches are not free
  - A null kernel launch will take non-trivial time
  - Actual number changes with HW generations and driver software
  - If you are launching lots of small grids you will lose substantial performance due to this effect

- Independent kernel launches are cheaper than dependent kernel launches
  - Dependent launch: Some readback to the cpu

- If you are reading back data to the cpu for control decisions, consider doing it on the GPU
  - Even though the GPU is slow at serial tasks, can do surprising amounts of work before you used up kernel launch overhead