Problem set 6

Theory questions

EVEN FLØRENÆS

TDT 4200

PARALLEL COMPUTING

FALL 2016

Problem 1

Describe a situation/application/ memory access pattern where it can be beneficial to use it and explain why it is beneficial to use the given memory type over others

---

a) Texture memory

Texture memory can be beneficial to use in graphics applications. Graphics application has a a memory access pattern which exhibits a great deal of spatial locality. Texture memories are in this application beneficial as it only costs one device memory read only on a cache miss; otherwise, it just cost one read from the texture cache. Texture memory is also designed for streaming fetches with a constant latency. This can be advantageous in graphics application where visual quality require stability and constant latency with opportunity to stream new frames.

---

b) Shared memory

Usage of shared memory may be beneficial in high-performance cooperative parallel algorithms like a parallelized implementation of matrix multiplication. Shared memory latency is roughly 100x lower than uncached global memory latency. Shared memory is also allocated per thread block, so all threads in the block have access to the same shared memory. Threads can access data in shared memory loaded from global memory by other threads within the same thread block. Using shared memory between many threads can give efficient implementation of high-performance cooperative parallel algorithms.

---

c) Constant memory

Constant memory is beneficial to use when you know that your input data will not change during the execution and when you know that all threads will access data from the same part of memory. In this case constant memory will be beneficial over other types of memory blocks as constant memory can save bandwidth over standard reads of global memory. A single read from constant memory can be broadcast to other “nearby” threads, effectively saving up to 15 reads. Constant memory is cached , so consecutive reads of the same address will not incur any additional memory traffic.

Problem 2

Which of the following code snippets will cause branch divergence?

---

a)

This code snippet will choose the same branch for every thread in a block, and no branch divergence will occur. The branch is determined by blockIdx which is the same for every thread within a block.

---

b)

This code snippet contains branching dependent on thread id. This will lead to branch divergence as some threads will execute the if statement and the rest will execute the else statement. This will lead to thread execution being serialized to first perform the first statement and then the second.

---

c)

Within a warp the threads need to follow the same execution trajectory. For this code snippet the thread execution is dependent on the thread id, leading to different behavior for different threads in the same block. This will lead to branch divergence.

1 Problem 3

1.1 Introduction to problem

For a parallel application with input and output of size n. The computation can be divided between CPU and GPU. Where the ratio of the size computed using the GPU is b. Which is a value between 0 and 1.The time necessary to compute the different parts of the input for CPU and GPU will amount to:

$$t_{GPU} = n_{GPU} R_{GPU} \tag{1}$$

$$t_{CPU} = n_{CPU} R_{CPU} \tag{2}$$

As we divide the input into two parts for CPU and GPU computation, n_{GPU} and n_{CPU} can be set as part of the total input size n:

$$n_{GPU} = bn \tag{3}$$

$$n_{CPU} = (1-b)n \tag{4}$$

1.2 Finding expression for t_{CPU} and $t_{total_{GPU}}$

Applying (3) and (4) in equations (1) and (2), the amount of time to perform the GPU and CPU computation will then be:

$$t_{GPU} = bnR_{GPU} \tag{5}$$

$$t_{CPU} = (1-b)nR_{CPU} \tag{6}$$

When using GPU we are required to transfer the data input from the CPU to the GPU. The transfer cost in time is dependent on the bandwidth B and will for a size m amount to:

$$t_{transfer} = mB \tag{7}$$

The total execution time on GPU will be the sum of computation cost and transfer cost:

$$\begin{aligned} t_{total_{GPU}} &= t_{GPU} + t_{transfer} = bnR_{GPU} + bnB \\ &= bn(R_{GPU} + B) \end{aligned} \tag{8}$$

1.3 Finding expression for b to divide work evenly

The GPU are generally faster than the CPU. A evenly distributed system using both CPU and GPU should divide the work between them so they finish simultaneously. This can be achieved by finding an expression for b which makes time to perfrom CPU and GPU calculation equal:

$$t_{CPU} = t_{total_{GPU}} \tag{9}$$

$$(1-b)nR_{CPU} = bn(R_{GPU} + B)$$

$$b = \frac{1}{1 + \frac{R_{GPU}+B}{R_{CPU}}} \tag{10}$$

1.4 Will it ever be beneficial to not use the GPU at all?

Equation (10) describes the fraction of the input to be computed using the GPU. In this section we will determine if it will ever be meaningful to set $b = 0$, and effectively not using the GPU. Looking at equation (10) we se that there are three parameters which affects the ratio b. Setting $b = 0$ would suggest that the denominator would approach infinity. The only part of the denominator of eq. (10) that can change is the relation between R_{GPU},B and R_{CPU}:

$$\frac{R_{GPU} + B}{R_{CPU}} \Rightarrow \infty$$

$$\frac{R_{GPU}}{R_{CPU}} + \frac{B}{R_{CPU}} \Rightarrow \infty \tag{11}$$

We have already established that the GPU is faster than CPU for all general and relevant cases for this calculations. Which gives:

$$\frac{R_{GPU}}{R_{CPU}} < 1 \tag{12}$$

Using equation (12) in (11) we get:

$$\frac{B}{R_{CPU}} \Rightarrow \infty \tag{13}$$

It will be benefical to not use the GPU and limit our computations to only use the CPU if the bandwidth between the GPU and CPU is much larger than the computation time on the CPU. Meaning if transferring data to the GPU costs much more than computing the same data on the CPU, the GPU should not be used.

Problem 4

a) What are coalesced memory accesses?

Coalesced memory access or memory coalescing refers to combining multiple memory accesses into a single transaction. A coalesced memory transaction is one in which all of the threads in a half-warp access global memory at the same time. This over simple, but the correct way to do it is just have consecutive threads access consecutive memory addresses. So if threads 0, 1, 2 and 3 read global memory 0x0, 0x4, 0x8 and 0xc, it should be a coalesced read.

b) What are shared memory bank conflicts?

If multiple addresses of a memory request map to the same memory bank, the access are serialized. The hardware splits a memory request that has bank conflicts into as many separate conflict-free requests as necessary, decreasing the effective bandwidth by a factor equal to the number of separate memory requests.