GPU- accelerated systems

סמאר אבא, תשע”ד
19.10.2014

מרצה: מרק זילברשטיין

말ון סופי

מותר כל חומר כתוב או ממוחשב.

אורך המבחן: 2.30
בדוקו במחברת 10 דפים

כל התשובות יכתבו במחברת ההבחנה

แทובות לשאלות סגורות יписанו במילים המדויקות לכל ליתן סאלה
แทובות לשאלות פתוחות יסופו ב’es’ ב’es’ התשובה על ידי עיגול

1
Question #1 (30 points)

You will implement a task of computing histogram of a given vector A of N integer elements in the range of 0-255.

1. Write a GPU-accelerated program (CPU and GPU code) that implements this task using CUDA.

Assumptions:
1. Array A is in GPU memory.
2. N is divisible by 1024.
3. The output histogram should be in GPU memory.
4. You may use a library function `vector_add<<<K, D>>>(float* vec, int D, int K)` which takes the vector vec in GPU memory, treats it as K vectors of length D and sums these vectors component-wise. The function operates in-place and return it’s result in the first D elements of vec.
5. Since atomic operations on global memory are very expensive, you should not use them.

```c
#include <cuda.h>

__device__ void vector_add(float* vec, int D, int K);

__global__ void histogram(int* A, float* hist){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < N)
        hist[i] = vector_add<<<1, D>>>(A + i * D, D, K);
}

#define N /* Array size */
void main(){
    float* gpu_A;
    ..... initialize gpu_A ..... 
    cudaDeviceSynchronize();
}
```
2. Does the code **YOU** wrote have warp level divergence? If yes, mark that part of the code.

**Answer:**

3. Does the code **YOU** wrote perform uncoalesced memory accesses? If yes, mark that part of the code.

**Answer:**

4. Does the code **YOU** wrote have bank conflicts? If yes, mark that part of the code.

**Answer:**

5. How much shared memory does each thread block uses in your implementation?

**Answer:**

6. Your code will be invoked on a GPU with the following properties:
   Each SM has 32K Registers, 4K shared memory, can run 64 warps, up to 8 thread blocks.

   Assuming each thread uses 32 registers and the amount of shared memory specified in the previous question, what is the GPU occupancy for this kernel.

**Answer:**
**Question #2 (7 points)**

Assume you have a GPU with 10 MPs, each MP may concurrently run up to 64 warps, and has 16K registers. Assume also you have a single kernel with 20 threadblocks 512 threads each. Assume that the kernel does not use shared memory.

Provide the upper bound on the number of registers per thread so that the use of a global barrier across all thread blocks of this kernel (using spinlock) would not result in a deadlock.

(1) 2  
(2) 4  
(3) 8  
(4) 16  
(5) 32  
(6) None of the above

**Question #3 (6 points)**

Given the following code and assuming all variables are initialized to 0

```
Thread 1                                      Thread 2
flag1=1;                                      if( flag2 == 2 )
flag2=2;                                      print flag1;
acquire;
flag3=3;
flag4=4;
release;
flag5=5;
```

Which of the following outputs is possible?

(1) 0, 3, 0  
(2) 1, 0, 1  
(3) 1, 3, 0  
(4) 0, 0, 1  
(5) 0, 3, 1
Question #4 (7 points)

Given the following kernel invoked with at least two GPU threads T1 and T2.

```c
__shared__ int a;
__device__ volatile int b;

__global__ void foo() {
    a=5;
    if( threadId == T1 ) {
        atomic_inc( &a, 1 );
    }

    if( threadId == T2 ) {
        atomic_inc( &a, a );
    }

    __syncthreads();
    b=a;
}
```

Choose one or more correct answers, which describe the state of the variables at the end of the execution:

1. **b definitely equals 12 if T1 and T2 are in the same thread block**
2. **b definitely equals 12 if T1 and T2 are in the same warp and there is only one warp**
3. **b definitely equals 12 if T1 and T2 are in the same thread block and T1 < T2**
4. **b can be any of 5, 6, 10, 11, 12 even if there is only one thread block**
5. **b cannot be 6 or 11 if T1 and T2 are in the same thread block**
Question #5 (5 points)

Consider an application that performs random data access to the disk, reading 4K per access, and performs floating point operations on that data with arithmetic intensity of 1 operation per float. This application can be implemented on the CPU or on the GPU using GPUfs.

Given the following system specifications: PCIe bandwidth: 16GB/s, HD bandwidth: 16GB/s, CPU performance: 1 Gflops, GPU performance: 8 Gflops, GPU memory b/w: 16GB/s, CPU memory b/w: 16 GB/s

Which parameter(s) of the system is (are) missing in order to determine whether using GPUfs is more efficient than using CPU? Explain.

Answer:

Question #6 (8 points)

Recall that MAPD uses prime encoding to perform quick string matching in tweets. List the main reasons why prime encoding is faster than a standard string matching algorithm on GPUs.

Answer:

Question #7 (7 points)

Assume that next generation GPUs will have 120 hardware queues (versus up to 32 queues today) to serve kernel invocation requests. Among SSLShader, string matching using GPUfs, gdev and Ptask, which system will most likely work faster on the new hardware? Explain.

Answer:
Question #8 (5 points)

Consider two GPU kernels 1000 threadblocks each. Kernel A runs 30 minutes, kernel B runs 1 minute. In every experiment, we run a CPU process which enqueues one kernel A, then one kernel B, waits until both complete and then terminates. Then we reset the system and run the experiment 1000 times again.

Select the order in which the kernels will be executed by a system managed by PTask?

- (1) AB(30 times)AB(30 times)...
- (2) Any order – no restrictions
- (3) ABABABABAB...
- (4) They will be executed concurrently
- (5) Like (1) if the system is not reset

Question #9 (5 points)

GDEV added some new functionality to GPUs that TimeGraph project did not address. Which feature was it?

Answer:

---

---
Question #10 (10 points)

Consider a queue of size one which allows to pass a single integer between producer and consumer. Specifically,

```c
struct queue{
    volatile int full;
    volatile int data;
}
```

```c
void init(struct queue* q){ q->full=false; }
```

You will implement two versions of function which adds new data into the queue.

Version 1 – assume that the queue is stored in **GPU memory**

```c
void push( struct queue* q, int new_data) {
```

Version 2 – assume that the queue is stored in memory which is **sequentially consistent**

```c
void push( struct queue* q, int new_data) {
```
Question 11 (10 points)

Consider a string matching program which reads from standard input, and writes the results to standard output. Assume that the string matching algorithm is implemented in discrete GPU.

Assume that
CPU->GPU and GPU->CPU memory bandwidth is 10 GB/s
CPU->CPU memory bandwidth is 10 GB/s,
string matching algorithm runs at 10 GB/s,
GPU->GPU memory bandwidth is 1000 GB/s

We invoke the string matching program on a text file of size 10GB located entirely in CPU memory as follows:

cat text | gpu_match string1 | gpu_match string 2

1. Calculate the expected execution time of this task on a standard NVIDIA CUDA system-wide

2. Recalculate the expected execution time assuming that the operating system may optimize the above using GDEV shared memory functionality.