GPU- accelerated systems

סמסטר אביב, תשע"ה
3.07.2015

מרצה: מרק וולברשטיין

מבחון סופי

מותר כל חומר כתוב או ממוחשב

אורך המבחן: שלוש שעות
בדקו כי במחברת 9 דפים
כל התשובות יכתבו במחברת הבוחנה

תשובות לשאלות מתוכנות ינ螫ו במכונת המודעות ליך עד לכל שאלה
תשובות לשאלות מרקראות יסומנו ברשימת התשובות על יד הכותב
Question #1 (30 points)

You will implement the task of finding the nearest neighbor of a query in a data set. The output is the index of the nearest entry from the beginning of the dataset (only one such entry exists) and the Euclidean distance of that entry from the query.

The query and the dataset entries are represented as struct Vec.

```plaintext
struct Vec {
    float x, y, z, w;
};
```

Each dataset is an array of elements of type Vec, total of 256 elements per dataset.

The input comprises two arrays: an array of queries and an array of datasets, one dataset for each query. The output comprises two arrays: for each query in the input, an array stores the index of the nearest entry in the respective dataset. Another array stores the respective distances.

Assume that the number of queries in the input is Q.

Example (assuming 3 entries in each dataset, Q=2)

Input:
Vec* data: (0,0,0,1) (0,0,0,0) (1,2,3,4) (3,4,3,4) (1,1,1,1) (2,3,4,5)
Vec* query: (1,2,3,3) (3,4,5,4)

float* nearest: 1.0, 2.0
int* nearestIdx: 2, 0

Write a GPU kernel which implements the task. You are required to employ parallel reduction algorithm as a part of your implementation. Assume that all arrays are preallocated in GPU memory.
__global__ void
nearest_neighbor( Vec *query, Vec* data, float *nearest, int *nearestIdx) {


}

2. Does your code have divergence? If yes, mark that part of the code.

Answer: ____________________________

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

Answer: ____________________________

4. Does your code 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 64K Registers, 48K shared memory, can run 64 warps, up to 8 thread blocks.
   Assuming each thread uses 35 registers and the amount of shared memory specified in the previous question, what is the GPU occupancy for this kernel.

Answer: ____________________________
Question #2 (10 points)

Consider a GPU-accelerated application that performs accesses to a large data structure which is shared between the CPU and the GPU. Namely, the application writes the data on the CPU first, invokes the GPU kernel which writes the data from GPU, and then, after the kernel terminates, reads the data from the CPU again. Choose one or more of the following access patterns that will result in the smallest amount of unnecessary data transferred between the CPU and the GPU when using an Asymmetric Distributed Shared Memory (ADSM):

1. Access to 1 byte from the GPU and access to the entire data structure from the CPU
2. Access to 1 byte from the CPU and access to the entire data structure from the GPU
3. Access to the entire data structure from both the CPU and the GPU
4. Access to 1 byte from both the CPU and the GPU

Question #3 (10 points)

In the RSVM paper, the authors claim that the GPUfs can be implemented on top of RSVM. Explain how it can be done, and describe two of the main disadvantages of such implementation compared to the original GPUfs system.

Hint: Consider the case of random access to extremely large files (1TB)

Answer:
Question #4 (10 points)

Given a GPU with 32K registers/SM and 32K shared memory/SM, 64 warps/SM, 4 TBs/SM. A kernel contains the following line:

```c
__shared__ char[SIZE];
```

Which configuration allows the highest utilization of ALUs on an SM?

1. SIZE=1024; #Threads/TB=256; 10 reg/thread
2. SIZE=1024; #Threads/TB=32; 10 reg/thread
3. SIZE=20*1024; #Threads/TB=1024; 10 reg/thread
4. SIZE=10*1024; #Threads/TB=512; 10 reg/thread

Question #5 (10 points)

Consider an implementation of a “single produce” - “single consumer” queue to pass one integer between two discrete GPUs connected via a new enhanced version of the PCIexpress bus. The queue is accessed by a single thread in each GPU.

Choose which of the improvements below is (are) expected to improve the throughput of the queue.

1. 3x higher bus bandwidth
2. peer-to-peer memory accesses between GPUs
3. improved atomic operations performance in GPUs
4. 3x lower latency
5. atomic operations across PCIe, but only if also (3)
6. atomic operations across PCIe, but only if also (2)

Question #6 (10 points)

Given the following code snippet.

```
Thread T1
flag1=1;
threadfence_block();
flag2=2;
threadfence();
flag3=3;
threadfence_system();
```

```
Thread T2
if (flag3==3)
  print 50+flag1
else
  print 60+flag2
```

Assume all variables are volatile, readable to a CPU and a GPU, and initialized to 0.

Choose all correct answers regarding the output of thread 2

1. Output can be any of 50, 51, 60 if T2 runs on a CPU
2. Output can be any of 50, 51, 60 if T2 runs in a different threadblock than T1
3. Output can be any of 51, 60 if T2 runs in the same threadblock as T1
4. If T1 and T2 are in the same warp, the output is one of 50, 51, 60
5. Output can be any of 50, 51,60, 62 only if T2 runs on a CPU
Question #7 (10 points)

Given two GPU threads T1 and T2 which access global variables \texttt{a}, \texttt{b}, \texttt{ready1} and \texttt{ready2} and execute the following code.
Assumptions:
• T1, T2 runs on different threadblocks.
• Print is fully supported and write its output directly to the screen
• All variables are initialized to 0.

```c
__device__ volatile int a,b;
__device__ volatile int ready1;
__device__ int ready2;

__global__ void foo() {
  if( threadId == T1 ) {
    a = 3;
    threadfence();
    ready1 = 1;
    threadfence();
    while( true ) {
      if( ready2 == 1 ) {
        print b;
        break;
      }
    }
  }

  if( threadId == T2 ) {
    b = 4;
    ready2 = 1;
    threadfence();
    while( true ) {
      if( ready1 == 1 ) {
        print a;
        break;
      }
    }
  }
}
```

Which of the following outputs are possible? (read form left to right):

(1) \texttt{3, 4}
(2) \texttt{0, 4}
(3) \texttt{0, 3}
(4) \texttt{3 <no output produced>}
(5) \texttt{0, 0}
Question #8 (5 points)

While explaining the logic of reordering the packets in a TCP stream the GASPP paper mentions the following:
(a) “A memory barrier is used to guarantee that all threads have finished hashing their packets.”
(b) “Each packet is handled by one thread”

Which of the following statements are correct?

(1) Each batch must be processed by a single thread block, because no global synchronization across GPU thread blocks is possible.
(2) **Assuming 33 packets per batch, the GPU utilization can be only slightly higher than 50%**.
(3) Using shared memory for storing the hash table is possible only if a batch contains up to 1024 packets, e.g. the maximum number of threads in a single threadblock.
(4) **GASPP authors are wrong: memory barrier cannot be used as a barrier between threads to synchronize their progress.**

Question #9 (15 points)

Implement a consumer-producer queue using files. The queue is of size 1, and can be used to pass positive integers (>0) between CPU (producer) and GPU (consumer) via file. The implementation should use the GPUfs system to read or write a file “a.txt” as a way to share data.

Assume that the file is initialized with zeros. Assume that all system calls are always successful.

Function reference:
int open/gopen(char* filename, O_RDWR);
ssize_t pread/gread(int fd, void *buf, size_t count, off_t offset);
ssize_t pwrite/gwrite(int fd, const void *buf, size_t count, off_t offset);
int close/gclose(int fd);

<table>
<thead>
<tr>
<th>GPU:</th>
<th>CPU:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int get()</code></td>
<td><code>void put(int data)</code></td>
</tr>
<tr>
<td><code>{</code></td>
<td><code>{</code></td>
</tr>
<tr>
<td><code>int buf, f=-1;</code></td>
<td><code>int buf, f=-1;</code></td>
</tr>
<tr>
<td><code>while(1)</code></td>
<td><code>while(1)</code></td>
</tr>
<tr>
<td><code>if (f!=-1) close(f);</code></td>
<td><code>if (f!=-1) close(f);</code></td>
</tr>
<tr>
<td><code>f=open(“a.txt”);</code></td>
<td><code>f=open(“a.txt”);</code></td>
</tr>
<tr>
<td><code>read(f,&amp;buf, 1,0);</code></td>
<td><code>read(f,&amp;buf, 1,0);</code></td>
</tr>
<tr>
<td><code>if (buf==0) continue;</code></td>
<td><code>if (buf!=0) continue;</code></td>
</tr>
<tr>
<td><code>res=buf; buf=0;</code></td>
<td><code>buf=data;</code></td>
</tr>
<tr>
<td><code>write(f,&amp;buf,4,0);</code></td>
<td><code>write(f,&amp;buf,4,0);</code></td>
</tr>
<tr>
<td><code>close(f);</code></td>
<td><code>close(f);</code></td>
</tr>
<tr>
<td><code>return res;</code></td>
<td><code>return;</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>