GPU- accelerated systems

םסרת אביב, תש''ו
22.07.2016

מרצה: מרק וייברשטין

מבחן סופי - מועד א'

מותר כל חומר כתוב או ממוחשב
combo: 110 מיקודים, לא יותר פקודות.

אודיט המבחנים: שלוש שעות

בדקו כי המשבצות 12 דפים

כל התשושות ייכתבו במשבצות המ biçimנה

השעיה לשאלות מתוכחה יינתו בנקודות המורכבים לקד ליד כל שאלה

השעיה לשאלות אפסים кварט מלומנות ברשימת השעיות על יד יFileNotFoundException
Question #1 (30 points)

You will implement the task of array compaction using a certain predicate as a function.

Example:

Input:
[1,2,3,4,5,6,1,2,3,4,5]

Compaction by using a predicate “find all odd elements” (x%2!=0)

Output
[1,3,5,1,3,5]

1. Write a GPU kernel which implements the task by (optionally) using the helper prefix_sum() function defined below. Assume that input and output arrays are pre-allocated in GPU memory. Assume that the kernel is invoked with 1024 threads/TB, 1024 TBs. The goal is to achieve maximum parallelism.

// returns false if the value is not included in the output, true ///
// otherwise
// can be called by each thread separately
__device__ bool predicate(int val);

// in-place exclusive prefix sum of the input array.
// assumes inout to be in shared memory. Uses 4096 bytes of shared //
// memory internally. To be called by all the threads together
__device__ prefix_sum(int* inout);
// *out_size is initialized to zero before the kernel launch
__global__ void
compact(int *in, int* out, int in_size, int *out_size) {
}
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. Your code will be invoked on a GPU with the following properties:
   Each SM has 32K Registers, 24K shared memory, can run 32 warps, up to 8 thread blocks.

   Assuming each thread uses 20 registers and the amount of shared memory as determined in your code, what is the GPU occupancy for this kernel.

   Answer:___________________________
**Question #2 (10 points)**

**CUDA Streams**

In addition to kernels and memory movement operations, the following operations can be used with a CUDA stream:

- `cudaEventRecord(cudaEvent_t event, cudaStream_t stream1)`: Record the `event` in `stream stream1`. `event` will be triggered once the GPU completes executing all the operations that preceded `event` in `stream1`.

- `cudaStreamWaitEvent(cudaStream_t stream2, cudaEvent_t event)`: Makes all the commands added to `stream2` after the call to `cudaStreamWaitEvent()` delay their execution until `event` has completed. The command is asynchronous.

This mechanism allows to introduce dependencies between the streams and implement *dependency graphs*.

Many computations can be represented as a dependency graphs with nodes as tasks and arrows as dependencies. Specifically, an arrow from node A to node B in the graph means that B must not be invoked until A is finished. Note that the graph shows which tasks can be invoked *in parallel*, thereby potentially improving the system performance.

In this exercise you will implement such a graph with the help of events and streams.

---

**Example dependency graph:**

```
Data transfer
H_data -> D_data

Kernel1

Kernel2

Kernel3
```

A valid implementation of the dependency graph is below:

```c
cudaEvent_t e;
<initialize events>
cudaStream_t s1, s2;
<initialize streams>
cudaMemcpyAsync(d_input, h_input, ..., cudaMemcpyHostToDevice, s1);
kernel1<<<s1>>>();
cudaStreamWaitEvent(s1, e);
kernel3<<<s1>>>();
kernel2<<<s2>>>();
cudaEventRecord(e, s2);
```
You will have to create the program that includes kernel launches, memory transfers and event-based synchronization. You are allowed to use **up to 3 streams** and as many events as you wish. Assume all the events and the streams are initialized and ready to use. Your goal is to provide the implementation with the maximum parallelism, i.e., without introducing any false dependencies.

As in the example, you need not specify the kernel arguments and launch parameters, except for the CUDA stream in which each kernel is launched on.
Question #3 (10 points)

Consistency

Consider the following code executed by three threads in the same threadblock

```c
__global__  int X;
__shared__  int x,y,z,w; // initialized to zero

if (tid==T3){z=7;}
__syncthreads();
if (tid==T2){while(z!=7); y=4; w=6;}
if (tid==T1){while(y!=4); x=5; if (w==6) x=2;}
__syncthreads();
X=x;
```

Choose all the correct answers

Answer:

a) The kernel will deadlock
b) X=2
c) if T1-T3 are in the same warp, X=2
d) if T1-T3 are all in different warps, X=2 or X=5
e) T1-T2 are in the same warp there will be deadlock
f) X is undefined because it is written by all the threads concurrently
Question #4 (10 points)
ActivePointers

Explain when an operation performed on an active pointer (assignment, increment, dereferencing) requires access to the page table and the reason for this access:

assignment:

increment:

dereferencing:

A programmer implemented ActivePointers for CPUs. She discovered that the overhead of software address translation on CPUs with exactly the same implementation is _________ (higher/lower - choose the right one) than the overhead of such translation on GPUs. Explain why.

Question #5 (10 points)
Producer consumer

Given 2 different systems:
System A does not support atomics over PCIe.
System B does support atomics over PCIe.

Choose among the following producer-consumer queue designs all the designs that CANNOT be efficiently implemented in system A with a single queue, but can be implemented in system B?

1) Single producer (CPU) & single consumer (GPU)
2) Two producers (CPU threads) & two consumers (GPU threads)
3) Two producers (one CPU thread, one GPU thread) & single consumer (CPU thread).
4) Single producer (CPU thread) & two consumers (threads on two different GPUs)
Question #6 (10 points)
Optimizations

Given the following code snippet executed by all threads of a threadblock:

1. 
   for (int i = 0; i < 127; i++) {
2.   
   arr[threadIdx.x * 128 + i] = threadIdx.x + i;
3.   
   __syncthreads();
4. 
}

1. Is __syncthreads() at line 3 required for the code to be correct? Why?
   Answer: ______________________________________________________________________

2. What performance issue does this code have if arr is in global memory? Explain. Assume the
gpu memory architecture studied in the class.
   Answer: ______________________________________________________________________

3. What performance issue does this code have if arr is in shared memory. Explain. Assume the
gpu architecture studied in the class.
   Answer: ______________________________________________________________________

4. If __syncthreads() at line 3 is removed, will it change the optimization in (2), optimization in
(3), both, or none? Explain.
   Answer: ______________________________________________________________________
Question #7 (10 points)

GPUfs

1. GPUfs employs CPU in order to perform file operations. Consider an imaginary extension of the GPUfs system in which GPU is performing operations on the disk directly, without using the CPU. Explain briefly why such an enhanced system represents a significant threat to the security of the system if deployed on the existing hardware.

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

2. GPUfs implements weak data consistency semantics for its buffer cache. Give an example of an application in which such consistency semantics requires special explicit care from the developer. In other words, the implementation under LINUX would require different code.

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

3. Suggest and explain which hardware property(s) is(are) required in order to provide efficient way of implementing standard POSIX file system consistency in GPUfs.

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

Question #8 (10 points)

GPUnet

Assume GPUnet system deployed on hardware that supports GPUDirect RDMA technology (as studied in the class). Select the task(s) that are executed by the CPU:

1) Upon send(), copying the data from GPU to CPU memory.
2) Upon recv(), copying the data from the bounce buffer to the user buffer.
3) Upon send(), triggering the network card to send it.
4) Polling the completion queues for notifications about completed network operations.
Question #9 (10 points)
MapD

1. Which of these match operations are **NOT** supported by MapD’s prime encoding (select all the correct options):

1) Check if a sentence in the database contains both words: foo and bar
2) Check if a sentence in the database contains either word: foo or bar but not both
3) Check if a sentence in the database contains the exact phrase: “foo bar”
4) Check if a sentence in the database contains the **substring**: foo (e.g. will match against the sentence “bigfoot was here”)

2. MapD’s prime encoding assigns small prime numbers to the most frequent words. Explain why.

__________________________________________________________________________
__________________________________________________________________________