GPU- accelerated systems

סמסטר אביב, תשע"ו
22.07.2016

מרצה: מרק וילברשטיין

מבחני סופי - מצל א - פיתור

מותר לכל חומר חובה או מעניקה

אודרי המבחנים: שלוש שנות
בדקו INTERVAL 12 דפים

כל התשובות י⚄מו במכתבת המهذه

השובה לשאלות פונותה י الجهاز בפתקה המוזכר לעבר יידי לכל שאלה

השובה לשאלות אמבריקאיות סופות בורשים התשובות על יידי צו

046274 - מועד א, תשע"ו
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.

```c
// 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) {
    int tid = threadIdx.x;
    int gtid = threadIdx.x + blockIdx.x * blockDim.x;
    __shared__ bool p[1024];
    __shared__ int s_out[1024];
    __shared__ int s_out_size;
    __shared__ int start_idx;

    if (gtid >= in_size) return;
    p[tid] = predicate(in[gtid]);
    __syncthreads();
    prefix_sum(p);
    __syncthreads();

    if (tid == 0) {
        s_out_size = p[1023] + 1;
        start_idx = atomicAdd(out_size, s_out_size);
    }

    if (predicate(in[gtid])) {
        s_out[p[tid]] = in[gtid];
    }
    __syncthreads();

    if (tid < s_out_size) {
        out[start_idx + tid] = s_out[tid];
    }
}

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

**Answer:** yes.

*Thread zero updates the size and start index.*

*Only threads whose predicate is true will update the output array.*

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

**Answer:** no.

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

**Answer:** no.

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:**

*Registers needed to run 32 warps: 20K < 32K*

*Shared memory needed to run 32 warps: 8K < 24K*

*Blocks needed to run 32 warps: 1TB < 8 TB*

⇒ *we can run 32 warps per SM (which is the maximum) ⇒ Occupancy is 100%*
Question #2 (10 points)
CUDA Streams

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

\textbf{cudaEventRecord}(cudaEvent_t \textit{event}, cudaStream_t \textit{stream1}): Record the \textit{event} in \textit{stream1}. \textit{event} will be triggered once the GPU completes executing all the operations that preceded \textit{event} in \textit{stream1}.

\textbf{cudaStreamWaitEvent}(cudaStream_t \textit{stream2}, cudaEvent_t \textit{event}): Makes all the commands added to \textit{stream2} after the call to \textbf{cudaStreamWaitEvent()} delay their execution until \textit{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:

![Dependency Graph](image)

A valid implementation of the dependency graph is below:

cudaEvent_t \textit{e};
<initialize events>
cudaStream_t \textit{s1}, \textit{s2};
<initialize streams>
cudaMemcpyAsync(d_input, h_input, ..., cudaMemcpyHostToDevice, \textit{s1});
kernelf1<<<\textit{s1}>>>();
cudaStreamWaitEvent(\textit{s1}, \textit{e});
kernelf3<<<\textit{s1}>>>();
kernelf2<<<\textit{s2}>>>();
cudaEventRecord(\textit{e}, \textit{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.
cudaMemcpyAsync(d_input, h_input, ..., cudaMemcpyHostToDevice, s1);
cudaEventRecord(ev_memcpy_done, s1);
f1<<<s1>>>();
cudaEventRecord(ev_f1_done, s1);
g1<<<s1>>>();
cudaEventRecord(ev_g1_done, s1);
cudaMemcpyAsync(h_output1, d_output1, ..., cudaMemcpyDeviceToHost, s1);
cudaStreamWaitEvent(s2, ev_f1_done);
g2<<<s2>>>();
cudaStreamWaitEvent(s2, ev_g1_done);
cudaStreamWaitEvent(s2, ev_g3_done);
h1<<<s2>>>();
cudaMemcpyAsync(h_output2, d_output2, ..., cudaMemcpyDeviceToHost, s2);
cudaStreamWaitEvent(s3, ev_memcpy_done);
f2<<<s3>>>();
g3<<<s3>>>();
cudaEventRecord(ev_g3_done, s3);
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:**
  No need.

- **Increment:**
  Needs access to the page reference count.

- **Dereferencing:**
  Needs page table access if the translation is invalid.

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.

Higher, because there is no latency hiding in hardware.

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: no. there is no communication between threads.

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: uncoalesced memory access. (threads in a warp access distant locations)

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

Answer: bank conflicts (all threads in a warp access the same bank)

4. If __syncthreads() at line 3 is removed, will it change the optimization in (2), optimization in (3), both, or none? Explain.

Answer: none. The problem is at the warp level, which is always executed in lockstep regardless of __syncthreads();
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.

   GPU runs in user privileges, and disk is a shared system resource. (e.g. OS will not be able to enforce file permissions)

2. GPUfs implements a 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.

   Producer consumer between CPU and GPU via a file system. In regular case, write to a file and read from the file does not require close and open. In GPUfs it does.

3. Suggest and explain which property(s) is (are) missing in today’s hardware in order to enable efficient implementation of standard POSIX file system consistency in GPUfs.

   Low latency high throughput interconnect between CPU and GPU.

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.

Sentence encoding is the product of the prime numbers encoding its words. If frequent words were given big prime numbers, most of the sentences would require large numbers to encode them, which means we will need to allocate more memory for the encoding (and the search will be more expensive since integer comparison will not be enough).