Multicore and GPU Programming - Sum Reduction

Prof. Lee Jinho

Due - ~~05/18 Wed 23:59~~ **5/20 Fri 23:59**

0 Introduction

In this project, you will implement parallel sum reduction with GPU Programming. In summary, you need to do the following:

- Run all seven versions of reduction kernels provided in lecture notes by writing correct kernel invocation (the invocation will slightly vary according to the kernel version)
- Write a **good** report.
- Your final code submission should execute the fastest version you have for 16777216 (2^ 24) item reduction.
- However, the submission should still include **all 7 versions of code, including the kernel invocation** for 7 different versions.
    - They do not get executed, but leave them as evidence.

The specification has the following contents. You just need to use CUDA on it.

!!!Caution: Plagiarism!!!

You need to be careful to comply with plagiarism regulations. There is an ambiguity between discussion, idea, and solution. Therefore, be careful not to take away the opportunity of your colleague. **This assignment gives you all the kernel code, but it does not mean that you can search or look at others' code.**

- No web code-searching
- No idea sharing
- Ok discussion about the project
- Searching for (parallel) reduction is okay, but don't look for cuda codes.

1 Environment Setup

The same manner as in previous homeworks.

You don't need to set up the environment. We prepared everything you need to do an experiment about GPU programming. Check GNU GCC/G++ 9.2.1, GNU make, CUDA 11.3 and HTCondor documentation for more information.

2 Framework

2-1 Sum Reduction

Parallel reduction refers to algorithms which combine an array of elements producing a single value as a result. This homework is aimed at implementing a reduction.

2-1-0 Structure of template code

```
/HW6$ tree .
.
├── bin                     // make file generate run file to here
├── cuobj                   // make file generate CUDA objective files to here
├── obj                     // main objective file in here
│   └── driver.o            // includes main(). Only the objective file provided!
├── Makefile                // Compile, Clean, Format, Submit
├── reduce64.cmd        // Command for condor
├── reduce4194304.cmd       // Command for condor
├── reduce8388608.cmd       // Command for condor
├── reduce16777216.cmd      // Command for condor
├── result                  // Result from condor
└── src                     // Source Code
    ├── reduction_kernel.cu
    └── reduction.h

5 directories, 7 files
```

2-1-1 Implement your optimized solution

```
/HW6$ cat src/reduction_kernel.cu

#include <cuda_runtime.h>
#include <device_launch_parameters.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
```

```
#include <math.h>

#include "reduction.h"

void allocateDeviceMemory(void** M, int size)
{
  cudaError_t err = cudaMalloc(M, size);
  assert(err==cudaSuccess);
}

void deallocateDeviceMemory(void* M)
{
  cudaError_t err = cudaFree(M);
  assert(err==cudaSuccess);
}

void cudaMemcpyToDevice(void* dst, void* src, int size) {
  cudaError_t err = cudaMemcpy((void*)dst, (void*)src, size,
cudaMemcpyHostToDevice);
  assert(err==cudaSuccess);
}


void cudaMemcpyToHost(void* dst, void* src, int size) {
  cudaError_t err = cudaMemcpy((void*)dst, (void*)src, size,
cudaMemcpyDeviceToHost);
  assert(err==cudaSuccess);
}

void reduce_ref(const int* const g_idata, int* const g_odata, const int n) {
  for (int i = 0; i < n; i++)
    g_odata[0] += g_idata[i];
}

void reduce_optimized(const int* const g_idata,  int* const g_odata,
                  const int* const d_idata, int* const d_odata, const int n) {

  // TODO: Implement your CUDA code.
  // Reduction result must be stored in d_odata[0]
}
```

As you see, the code given is almost the same as your HW5. However, you're supposed to use GPU to get the performance even lower. The target is 0.5 msec.

The following are variable descriptions. Please read carefully and fit your code for correct grading.

g_idata
- 1D host integer input sequence. The total summation value is not over 2^31-1 or not under -2^31

g_odata
- 1D host integer output sequence. This is for copying the cuda result to host. This process will work in the main function so you don't need to care about it.

d_idata
- 1D device integer input sequence. This is for copying the sequence to device. This process will work in the main function so you can assume that the sequence is already stored in d_idata.

d_odata
- 1D device integer output sequence. Your result must be stored in d_odata[0]

n
- the size of input sequence.

You don't include the time for memcpy and allocating the device memories. In case you want to do something about the allocation or copy (You probably don't) we have put separate alloc/dealloc/memcpy functions. In the driver.cc that is provided as pre-compiled binaries, it would look something like this:

```
int main()
{
…
g_idata = new int[N];
g_odata = new int[N];
allocateDeviceMemory(&d_idata);
allocateDeviceMemory(&d_odata);
cudaMemcpyToDevice(d_idata, g_idata, N*sizeof(int));
time_start();
reduction_optimized(g_idata, g_odata, d_idata, d_odata, N);
time_end();
cudaMemcpyToHost(g_odata, d_odata, sizeof(int));
deallocateDeviceMemory(&d_idata);
deallocateDeviceMemory(&d_odata);
free g_idata, g_odata;
…
}
```

If you want to allocate anything else, you should do it within **reduction_optimized()**. Of course, you're not allowed to change or re-write the main(). If you think you need to, please contact us.

2-1-2 Compile and run matmul

The 'reduce' program gets two arguments

- inputPath: path to input file
- outputPath: path to output file

```
# Compile
/HW6$ make
mkdir -p cuobj
nvcc -g --ptxas-options=-v -std=c++11 -O3 -Iinclude -c src/reduction_kernel.cu -o
cuobj/reduction_kernel.o
ptxas info    : 0 bytes gmem
CUDA Compiled src/reduction_kernel.cu successfully!
mkdir -p bin
OBJ: obj/driver.o
CUOBJ: cuobj/reduction_kernel.o
g++  obj/driver.o cuobj/reduction_kernel.o -o bin/reduce -g -std=c++11 -Wall
-Wno-sign-compare -O3 -L/usr/local/cuda/lib64 -lcudart -Iinclude
-I/usr/local/cuda/include
Compiled obj/driver.o successfully!

# Local run
/HW6$ ./bin/reduce /nfs/home/mgp2022_data/hw6/input_64.txt
/nfs/home/mgp2022_data/hw6/output_64.txt

====================================
          Sum Reduction
====================================
The size of Sequence: 64
====================================

Read input file(/nfs/home/mgp2022_data/hw6/input_64.txt)...
Read output file(/nfs/home/mgp2022_data/hw6/output_64.txt)...

Run your solution...

reduce_optimal took 1.4e-04 msec
Correct
```

3 Remote Execution

There are four submit function, **submit_64, submit_4194304, submit_8388608** and **submit_16777216**. Each number means the size of sequence.

Step 0. Submit a job to condor.
Step 1. Wait until finished.(Check **result/reduce.log** for Job terminated)
Step 2. Check the output(**result/reduce.out**). There will be **TIME** and **CORRECTNESS** of your solution.

```
/HW6$ cat reduce64.cmd
#####################################################
##
## Sum Reduction Condor command file
##
#####################################################

executable      = bin/reduce
output          = result/reduce.out
error           = result/reduce.err
log             = result/reduce.log
environment = "LD_LIBRARY_PATH=/usr/local/cuda/lib64"
request_cpus = 16
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT
arguments               = /nfs/home/mgp2022_data/hw6/input_64.txt
/nfs/home/mgp2022_data/hw6/output_64.txt

/HW6$ make submit_64
mkdir -p result
condor_submit reduce64.cmd
Submitting job(s).
1 job(s) submitted to cluster 11801.

/HW6$ cat result/.out
==================================
          Sum Reduction
==================================
The size of Sequence: 64
==================================

Read input file(/nfs/home/mgp2022_data/hw6/input_64.txt)...
Read output file(/nfs/home/mgp2022_data/hw6/output_64.txt)...

Run your solution...

reduce_optimal took TIME msec
CORRECTNESS
```

We check only **the runtime of submit_16777216 (2^24).** Other submit cmds will be a faster indicators of whether you are doing well in our server. So, do not hesitate to use these.

4 Criteria

4-1 Requirements

4-1-0 General

- You should implement sum reduction with CUDA.
- No external libraries. You can only use STL.
    - This of course means: No cuBLAS
- Do not override driver.o

4-1-1 Sum Reduction

- Problem size: 16777216 (2^24)
- Performance requirement for the final version: 0.5 msec (it has a huge margin)
- Correctness: Should work correctly for sequence input between size of 16 (2^4) ~ 268435456 (2^28)
    - You can assume the size is a power of 2 (=2^N)

4-2 Measurements

We are measuring performance from a real machine, and it involves a certain amount of luck. To reduce the luck effect as much as possible, we do the following:

- Turn off DVFS of the machine
- Warm up the server for a few minutes before measuring. This means that the **program will run slower than your own tests** for all of you during our grading.
- We will run your program at least five times and **use the minimum exec.time**.

5 Report

Your report should include at least

- What each version of kernel is doing. Yes, it will have some overlap with the lecture slides, but assume that you're teaching a friend. Use your own words to describe them.
- How the kernel invocations change according to each kernel. This is the only part that you get to code on your own. So explain it.
- Perform a lot of experiments. You can do it for the 7 different versions, problem sizes, and so many hyperparameters (threadblock size, how many elements each thread handles, etc).
- Try more things to get further speedup.

- Explain! why did some technique work? why did it not? it can be wrong, but please at least try to explain the theory and the results.

- **PDF only**
- **No page limit this time** to encourage high-quality reports. But I promise that reports full of meaningless pages will be penalized.

6 Grading

- Submitted code: 50pts
  - You get 50 pts if your program runs correctly within 0.5 ms for 16777216 (2^ 24) elements .
    - To get the full score, your program must not only run correctly within 0.5 ms for input size 16777216 (2^ 24), but also run correctly for any 2^N-sized input between size of 16 (2^4) ~ 268435456 (2^28)
  - 25pts for being a correct CUDA program
    - Compile error or incorrect results: 0pts
    - If the program can be fixed with only with Makefile: 5pts penalty (20pts)
  - I will not disclose the performance-related point this time, because I believe the techniques are already given from the seven versions of kernels. But I will give extra credits for those who get significant improvement over others, or deduct points from poorly performing implementations.
- Report: 50pts
  - Make sure you have a high-quality report

7 More materials

- GNU GCC/G++ 9.2.1
- GNU make
- CUDA
- HTCondor
- Our Lecture Slides