Comparison in Reduction Algorithm

Hunter Adomtis

Project Overview

The goal of this project was to analyze the performance of the newly added reduction algorithm in C++17. This was accomplished by taking a simple reduction algorithm, both optimized and optimized for the hardware, and comparing the speed of the program to reduce function provided by the standard library. Finally, in order to fully test the performance of the reduction function extra tests were performed to further benchmark the program. These tests include using sequential execution rather than parallel and creating a serialized algorithm in C++ and comparing the results.

GPU Usage

The gpu role in this project was to run a simple parallel reduction algorithm. This was implemented by having each thread copy two pieces of data into shared memory, and then looping until the stride was larger than the remaining elements in the array. Both an optimized version and unoptimized version were implemented for further data analysis. The difference between the two algorithms is that the optimized version utilizes the hardware better through minimizing warp divergence through keeping the partial result towards the beginning of the array.

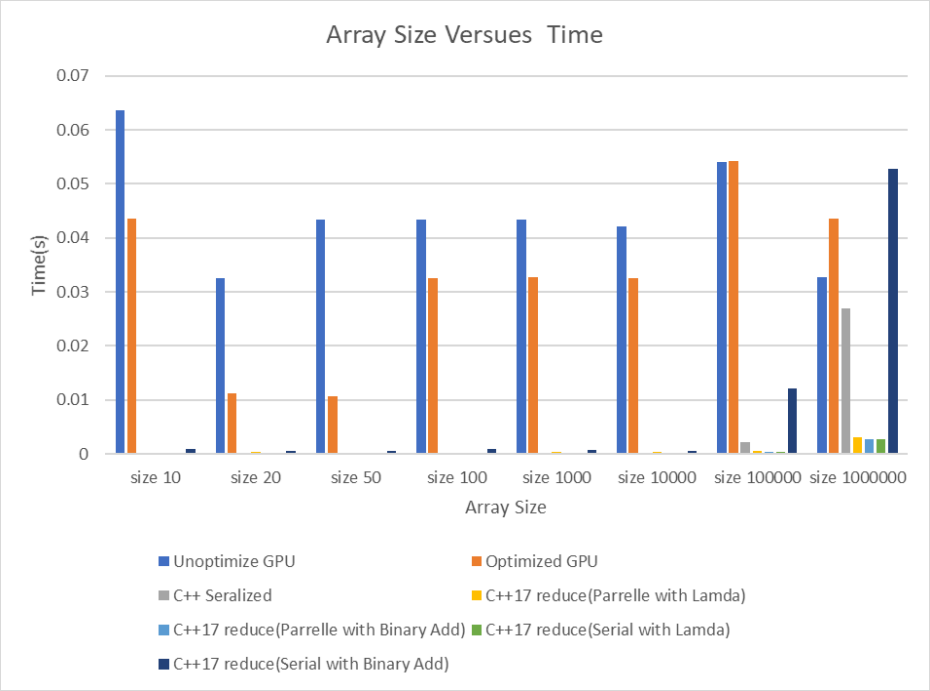
The threads were partitioned in one dimensional blocks of 512 threads. The reason 512 threads was chosen was simply because it was the block size utilized for assignment two. Finally, 1-D blocks were chosen because the reduction algorithm works on a 1-D array therefore it is the most logical to format the blocks.

For the parallel algorithm utilized by the reduce function in C++17 a gpu is not being utilized. This is because the parallelism is achieved via multi-threading on the cpu.

Implementation

I implemented two extremely simple reduction algorithms in cuda, one not optimized for hardware and the other one that is. I then implemented a serialized reduction in C++ and used the reduce function in C++17. For the reduce function I used both serial and parallel execution mode and the standard binary add operation and a lambda function. I then measured compilation time for all instances of the program and compared the results.

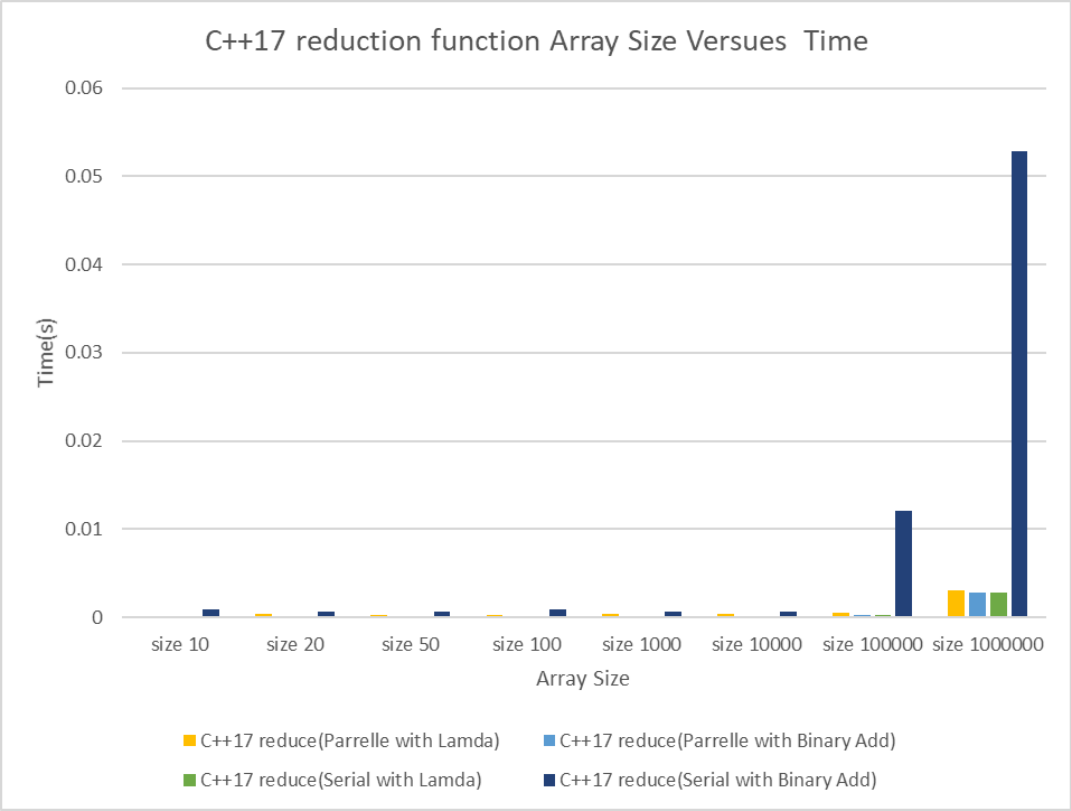
Results



This graph shows the different competition times for the different reduction methods. Clearly relying on the gpu is the worst choice for small data sets, given the transfer delay, but this cost is mitigated with larger data sets. This is explained by extra parallelization able to outpace the data processing of serialization. This phenomenon is demonstrated in the data from the reduce function using the serial data processing using the binary add operation. The combination of

serial operations and the function call of the binary operation delayed the program long enough outpaced the transfer delay of going from the cpu.

An important fact demonstrated by this graph is the usefulness of including the C++17 reduce function for improving performance of programs working with medium amounts of data. This is clear when function is able to maintain a high performance when simple serialized addition operation begin to overwhelm the cpu and it being faster than the transfer delay between the cpu and gpu.



The purpose of the graph is to show the effectiveness of using the reduce function in C++17 with parallel versus serialized execution and using a lambda function versus a binary operation. From the graph, it is clear that using parallel execution has very little effect on the overall performance. This is due to the fact that the reduction algorithm is using cpu parallelization therefore only a small portion of the data is being processed at a time in comparison to gpu parallelization. An interesting implication of this graph is that using lambda function is better than utility binary operation. This could be due to the fact that the lambda function has better locality in comparison to calling a function whose definition might need to be retrieved from memory.

How to Run Code

In bender use make to compile the cuda code. Two executable files will be made, reduction and optimized reduction.

In bender to compile the serialized C++ array reduction use

g++ -std=c++11 -o redcutionSeralizedCpp reduction.cpp

An executable redcutionSeralizedCpp will be made.

For the C++ reduce function code go to Coliru (stacked-crooked.com) and copy code into editor

```
#if PARALLEL
#include <execution>
#define SEQ std::execution::seq,
```

```
#define PAR std::execution::par,
#else
#define SEQ
#define PAR
#endif


#include <chrono>
#include <iomanip>
#include <iostream>
#include <numeric>
#include <utility>
#include <vector>


int main()
{
    std::cout.imbue(std::locale("en_US.UTF-8"));
        const std::vector<double> v(10000000, rand());
        std::cout<<"Reduce Parrelle with lamda function"<<std::endl;
        const auto t1 = std::chrono::high_resolution_clock::now();
         std::cout << std::reduce(PAR v.begin()+1, v.end(),*v.begin(),
```

```
[](double a, double b){ return a+b;}) <<'\n';
 const auto t2 = std::chrono::high_resolution_clock::now();
const std::chrono::duration<double, std::milli> ms = t2 - t1;
std::cout<<ms.count()<<"ms"<<std::endl;

std::cout<<"Reduce Parrelle with standard binary addation opperation"<<std::endl;
const auto t3 = std::chrono::high_resolution_clock::now();
 std::cout << std::reduce(PAR v.begin()+1, v.end(),*v.begin(),
std::plus<>()) <<'\n';
const auto t4 = std::chrono::high_resolution_clock::now();
 const std::chrono::duration<double, std::milli> ms2 = t4 - t3;
std::cout<<ms2.count()<<"ms"<<std::endl;

std::cout<<"Reduce sequental with lamda function"<<std::endl;
const auto t5 = std::chrono::high_resolution_clock::now();
 std::cout << std::reduce(SEQ v.begin()+1, v.end(),*v.begin(),
[](double a, double b){ return a+b;}) <<'\n';
 const auto t6 = std::chrono::high_resolution_clock::now();
const std::chrono::duration<double, std::milli> ms3 = t6 - t5;
```

```
    std::cout<<ms3.count()<<"ms"<<std::endl;


    std::cout<<"Reduce Sequental with standard binary addation opperation"<<std::endl;
   const auto t7 = std::chrono::high_resolution_clock::now();
    std::cout << std::reduce(SEQ v.begin()+1, v.end(),*v.begin(),
    std::plus<>()) <<'\n';
    const auto t8 = std::chrono::high_resolution_clock::now();
     const std::chrono::duration<double, std::milli> ms4 = t8 - t7;
    std::cout<<ms4.count()<<"ms"<<std::endl;
}
```

Next change the command line at the bottom of the editor to

g++ -std=c++23 -O2 -Wall -pedantic -pthread main.cpp && ./a.out

Finally, click the compile code button.

Problems faced

The one major problem I encountered was the bender server does not have the capability to compile C++17 code. This comes from the fact the gcc compiler installed on the server is outdated and only has the ability to compile code of C++11 or older. To solve this issue, I found a code editor on the website Coliru.Stacked-Crooked that can compile code. In order to account for the variation of running on hardware versus the website I ran simple programs, such as hello

world and a simple sequential addition of vector, average the difference in compilation. I then subtracted value from the result of running the reduce function in the browser.

Contributions

Hunter Adomitis = 100%