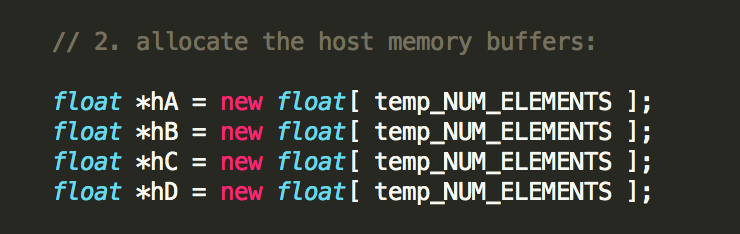
Parallel Programming

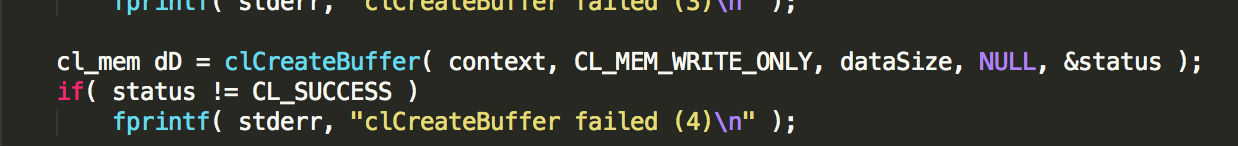
CS575

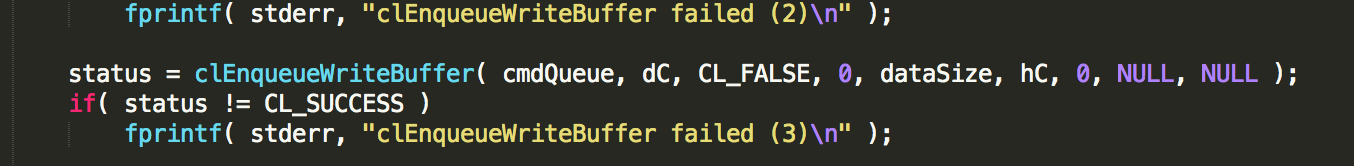
Chao Zhang

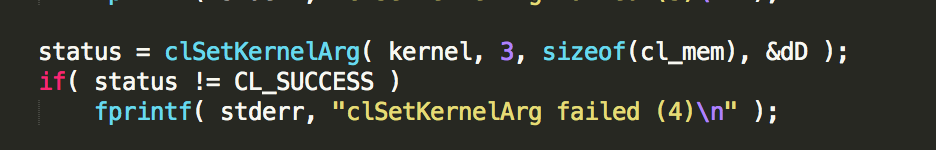
Project #6

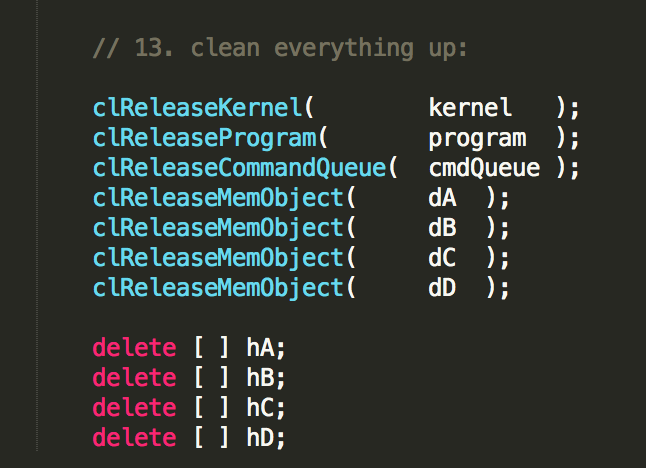
1. Source listing



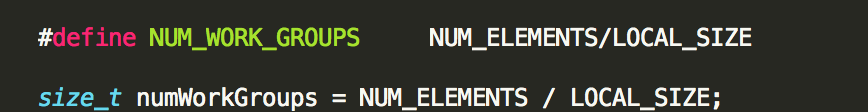


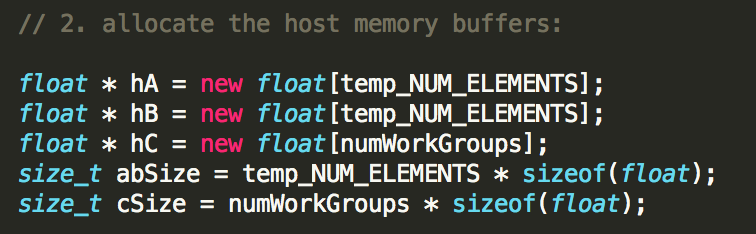


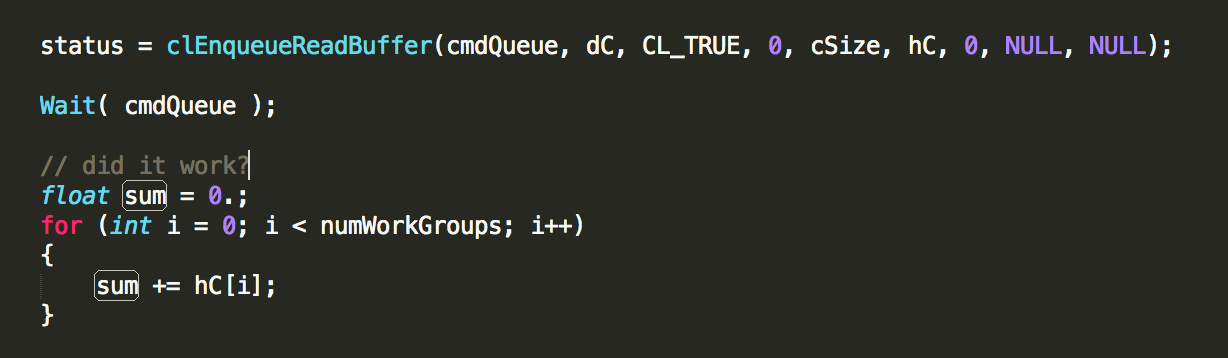




Those are the code added to the first.cpp to make the Multipy-add work. All of those code are used to add the third array.

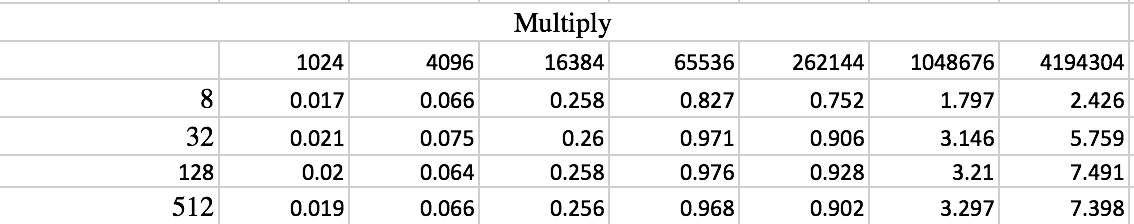




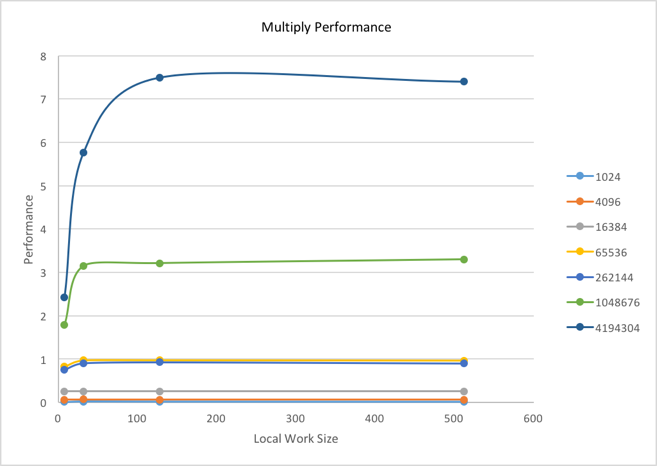
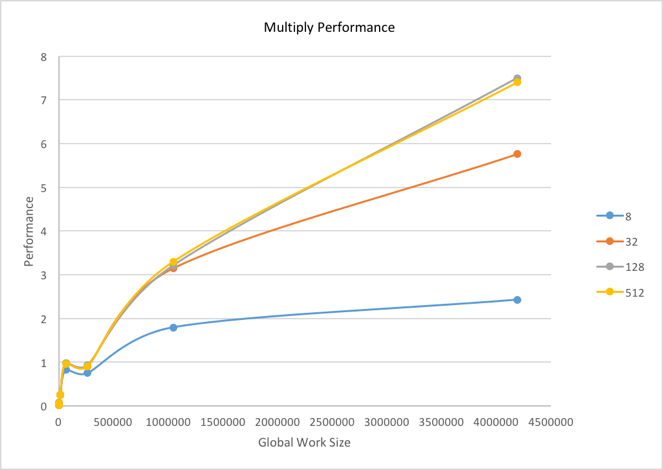


Those are for the reduction, it make the size\_t numWorkGroups = NUM\_ELEMENTS / LOCAL\_SIZE; and use the sum to do the reduction.

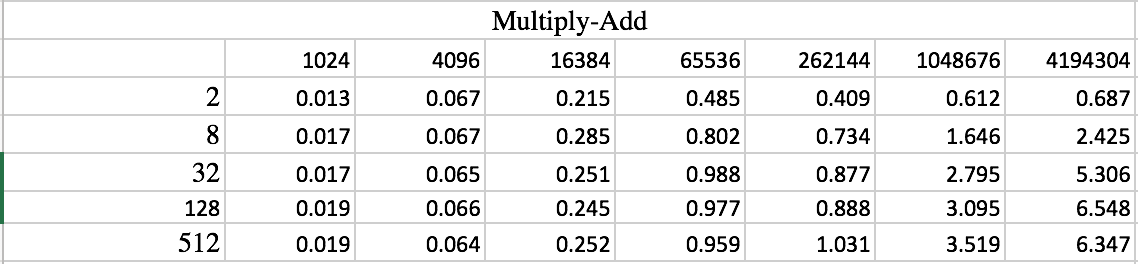
1. Result and analysis

I run this project on the rabbit. 

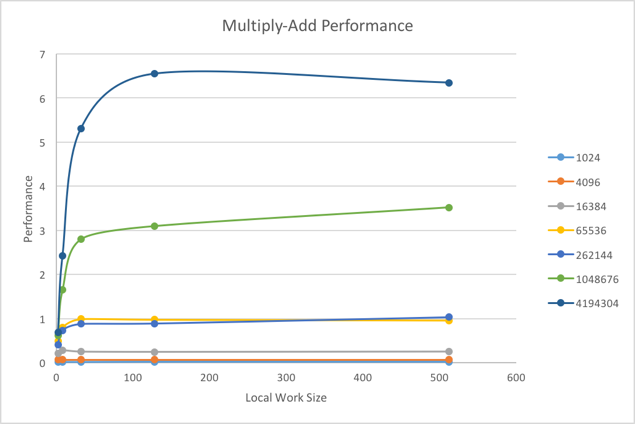
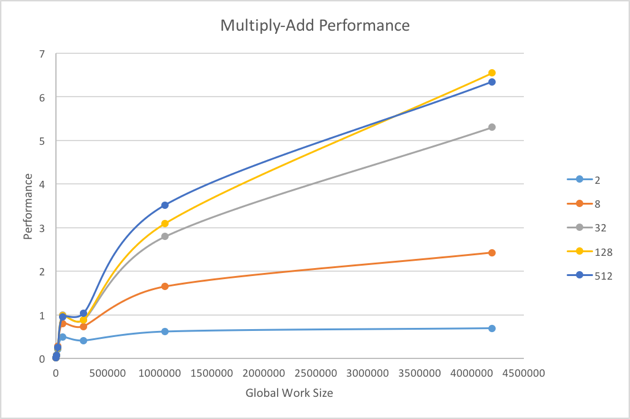
Multiply Results Table



Multiply Results Graphs

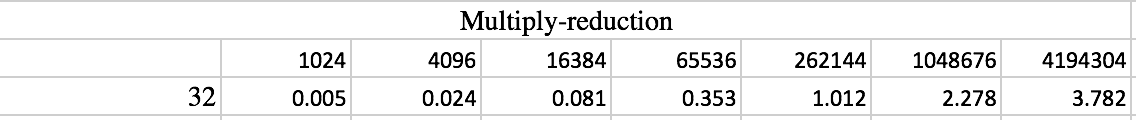


Multiply-Add Results Table

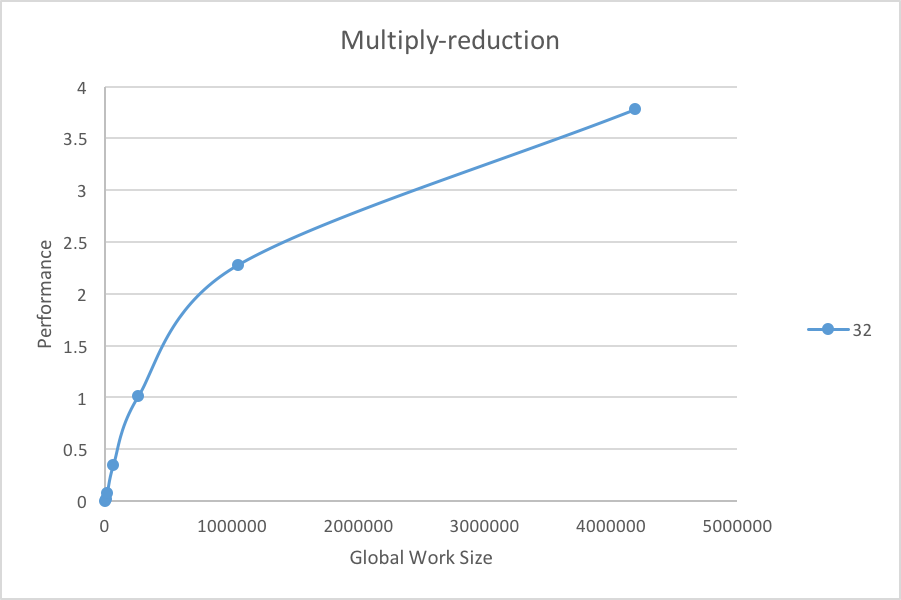


Multiply-Add Results Graphs

We can easily get the conclusion from those four graphs that the performance increased with the increase of the work size. The performance gets better when the work size gets larger. For the size of 128 and 512, I think the performance are almost get the limit so the performance looks the same. The reason why the Multiply and the Multiply-Add has the similar graph because the data is too small to shown the difference. But the Multiply should have a better performance. With the use of GPU, the computing performance can be increased by the control or it can make a good usage of the power of the GPU.



Multiply-Reduction Table



Multiply-Reduction Graph

Compare the Multiply-Reduction and the other two, I can say that the Multiply-Reduction has a better performance. As we know that the reduction will half the size each time, so the bigger the size is, the better performance we got. So I think the reduction is a good way to deal the multiply calculation in the GPU.