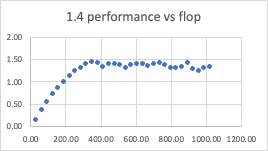
P1.1-1.3

Graphical user interface, text

Description automatically generated

P1.4

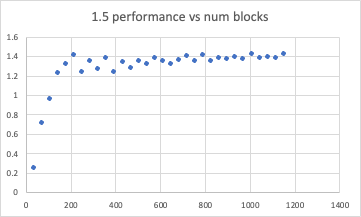
A picture containing table

Description automatically generated

We can see the performance start to peak out at ~350 threads. We will fully utilize all resources then and additional threads will not improve performance.

A picture containing timeline

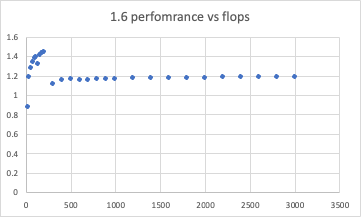
Description automatically generated



Using hint, we have 8 blocks/SM(8\*128=1024). And 72 SMs, so we have 72\*8=576 blocks. We can see performance peaking out around 550(less variance there compared to 200). This make sense as we have used all our resources. Doing more blocks will fight for resource with existing blocks

A picture containing table

Description automatically generated

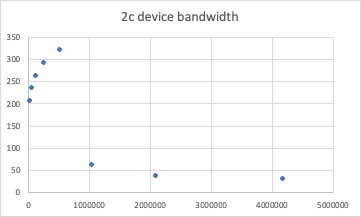


On the performance vs iterations plot, we can see performance peaking out at around 250 iterations, and there’s a break after that. I have no idea why, my best guess is resources are fully utilize up to 250, and break down after that.

Question 2

Table

Description automatically generated



2.4. The plot is increasing initially, but decreasing after that. It’s increasing in the first part as there were idle resources not being fully make use of. However, as number of nodes increases too much. We are not able to fully store the constants properly in a SM. This result in more reading from further caches/memory.

3

Text

Description automatically generated with medium confidence



First of all, data is stored contiguously. And we might not be able to store all in a single SM. We have to go there further places to fetch data. When stride length is small, most of the data can be access “nearby”, nearest cache or shared memory. We can see as stride length increases, performance decreases.