Silviu Gruber, [gruber13@purdue.edu](mailto:gruber13@purdue.edu)

*Note:   
 I wasn’t exactly sure about the format or the form of the analysis.*

*Please give me feedback so I can adjust for the future assignments even if I lose points here.*

*CUDA was a bit of a learning curve but I am very hungry for that ALEXNET project I hope I can measure up to it!!!!*

For my two models SAXPY and MCPi I utilized the following default values:

#define VECTOR\_SIZE (1 << 20)

#define MC\_SAMPLE\_SIZE      1e6

    #define MC\_ITER\_COUNT       32

    #define WARP\_SIZE           32

    #define SAMPLE\_SIZE         MC\_SAMPLE\_SIZE

    #define GENERATE\_BLOCKS     1024

    #define REDUCE\_SIZE         32

    #define REDUCE\_BLOCKS       (GENERATE\_BLOCKS / REDUCE\_SIZE)

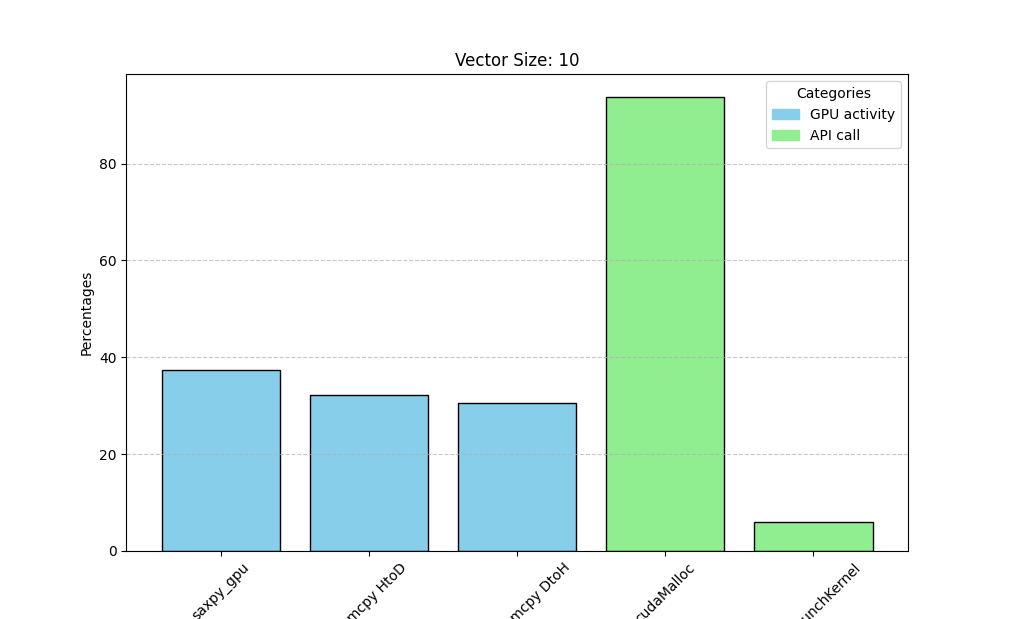
When testing different inputs I kept all other values to their default setting.

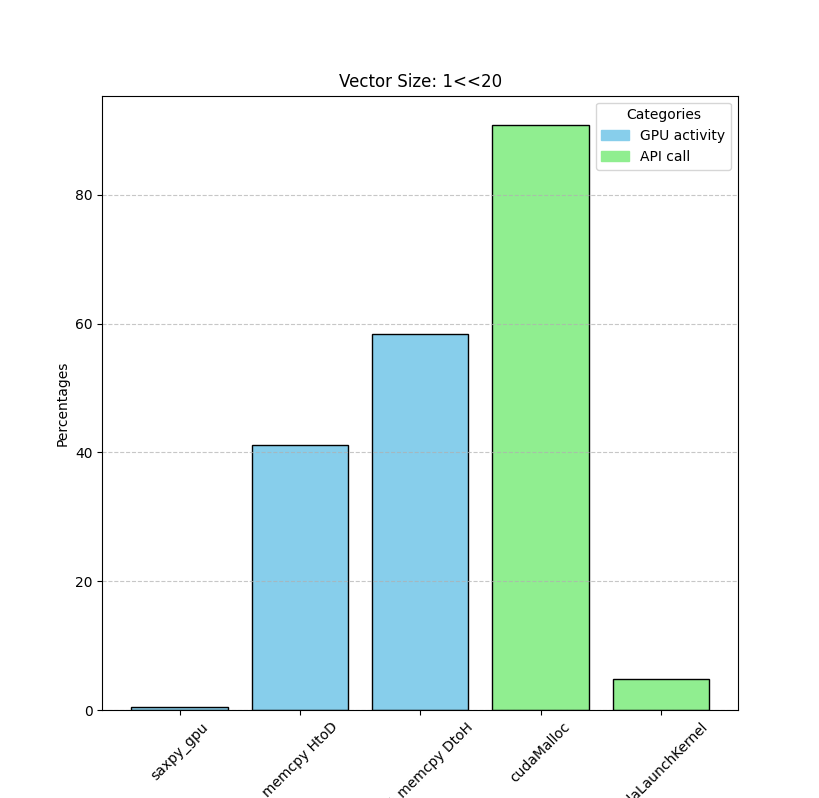
For my SAXPY model I varied the input VECTORSIZE from (10, 1<<20, 1<<30)

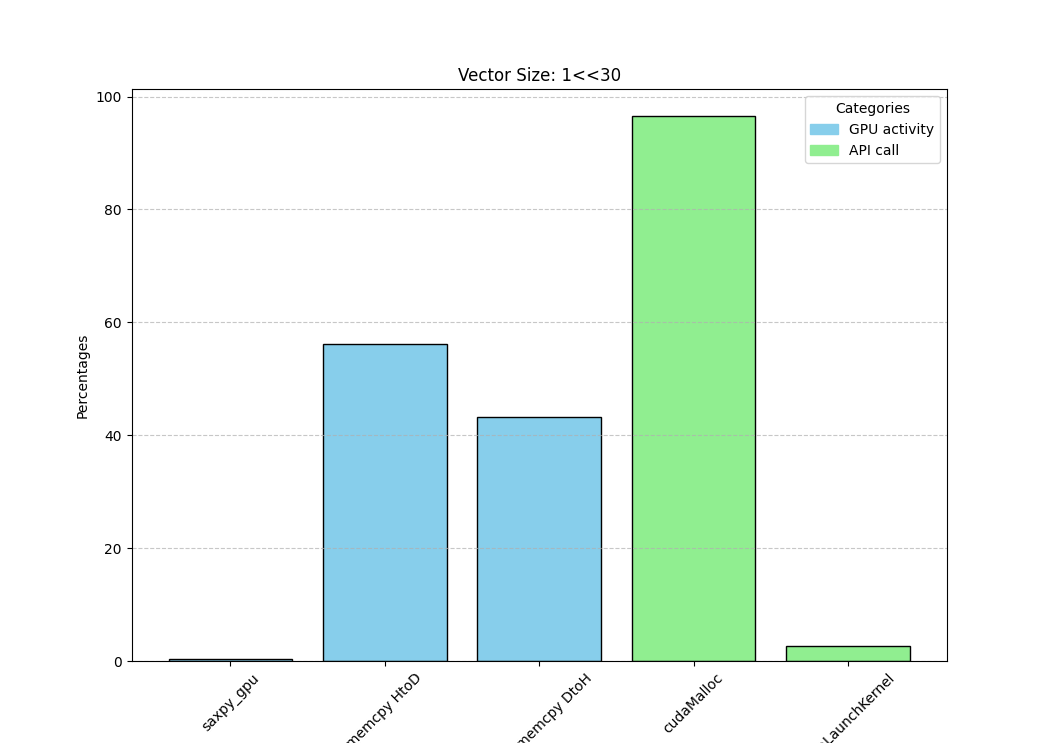
The execution times were roughly

ON GPU (2.3 us, 4ms, 4s)

ON CPU( 105ms, 110ms , 100s )







For Monte-Carlo-PI

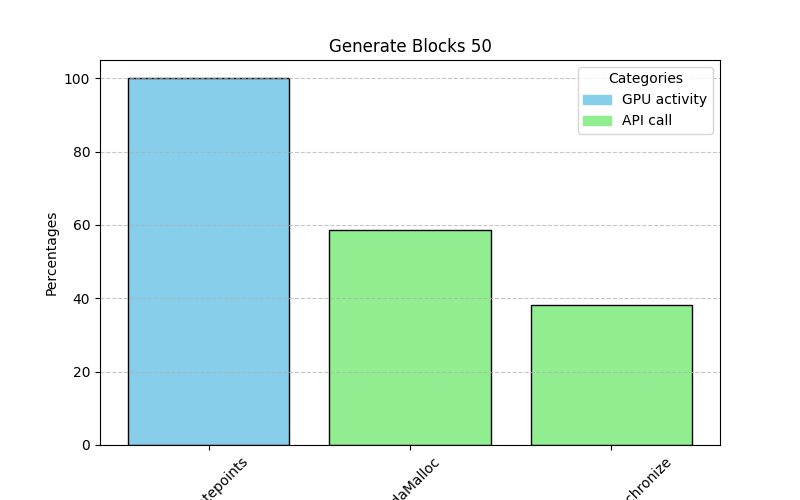
I varied the inputs GENERATE\_BLOCKS, SAMPLE\_SIZE, REDUCE\_BLOCKS, and REDUCE\_SIZE

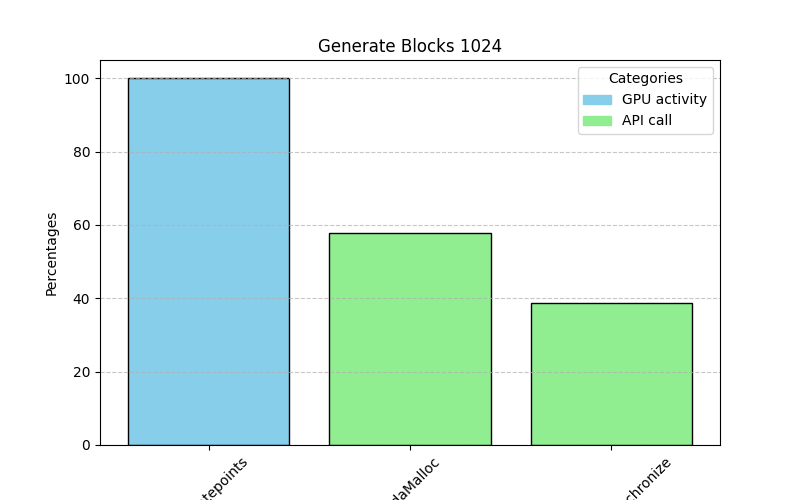
GENERATE\_BLOCKS between (50, 1024, 60005536)

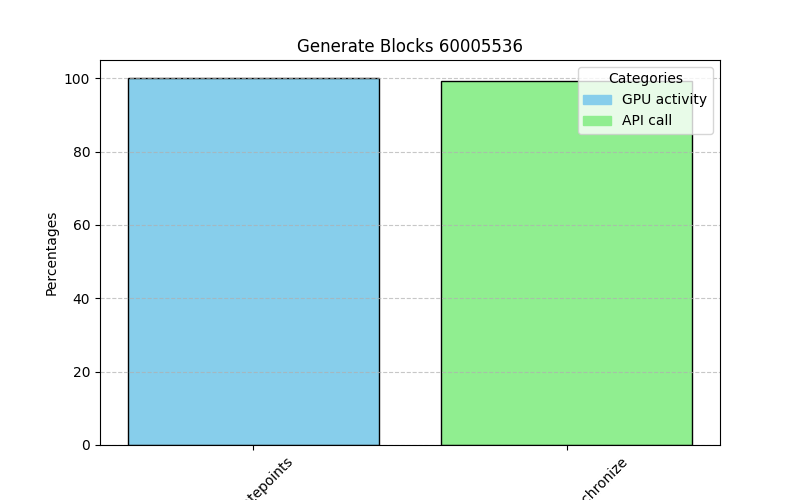
The times were roughly

On GPU (63 ms, 63.126 ms, 137 s)

On CPU (166.37 ms, 163.0377 ms, 137 s)





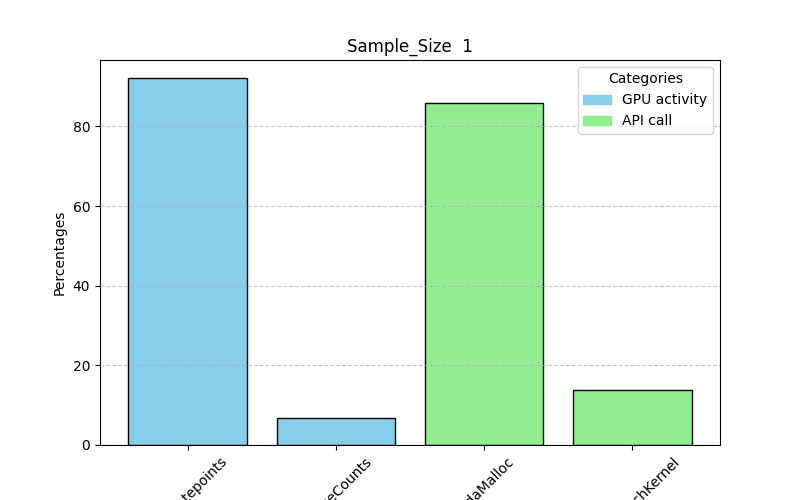


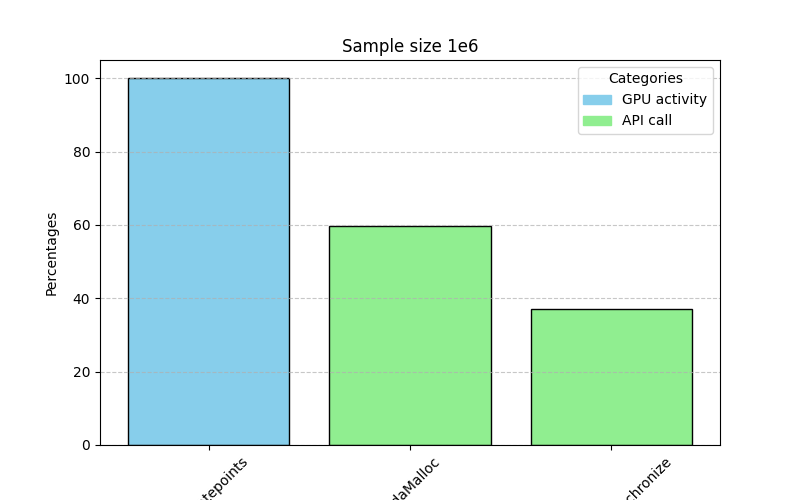
Sample\_Size between (1, 1e6, 1e9)

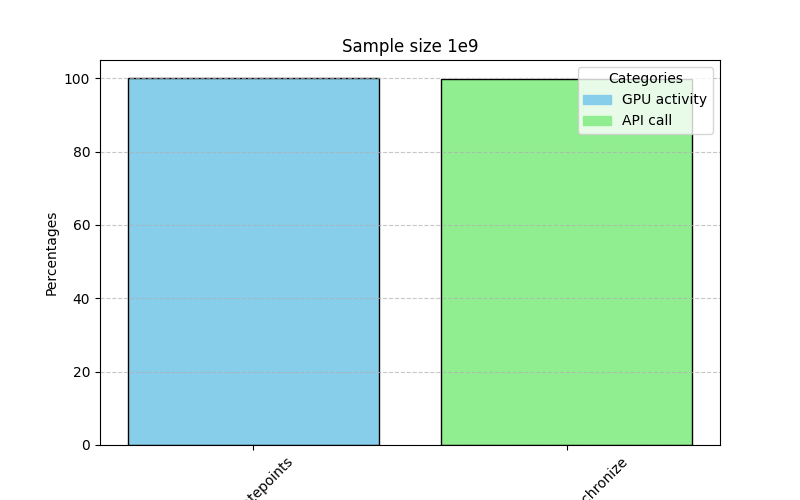
The times roughly

ON GPU ( 157 us, 67 ms, 56 s )

ON CPU ( 118 ms, 181 ms, 56 s)





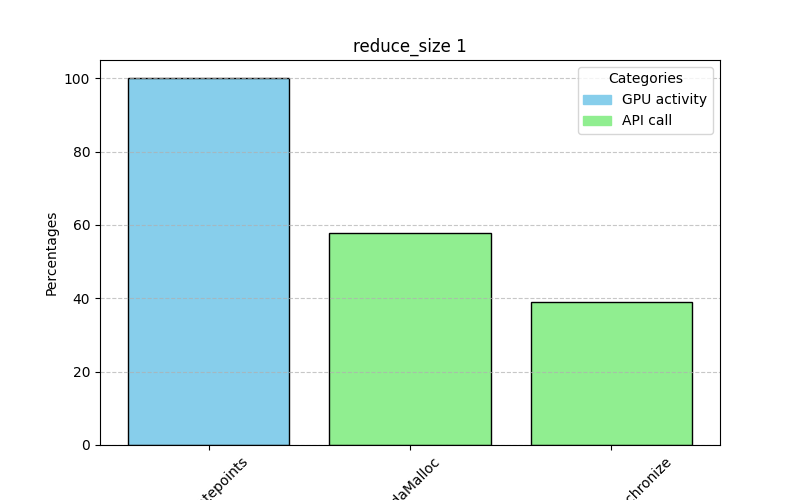


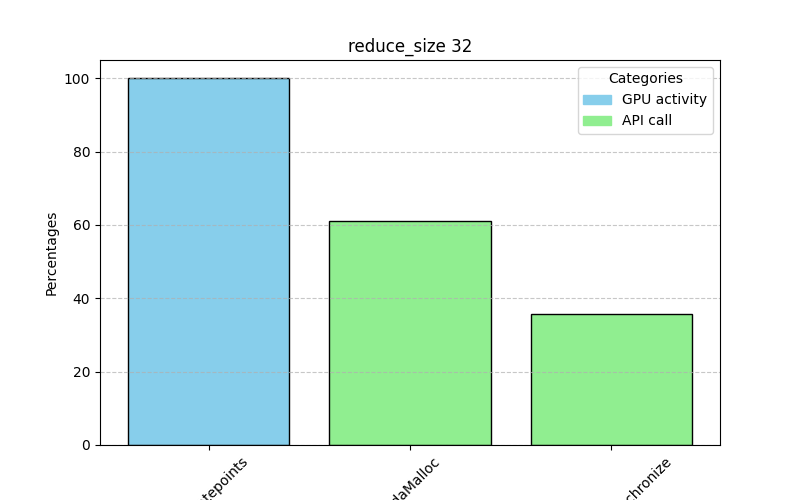
REDUCE\_SIZE from (1, 32, 1024)

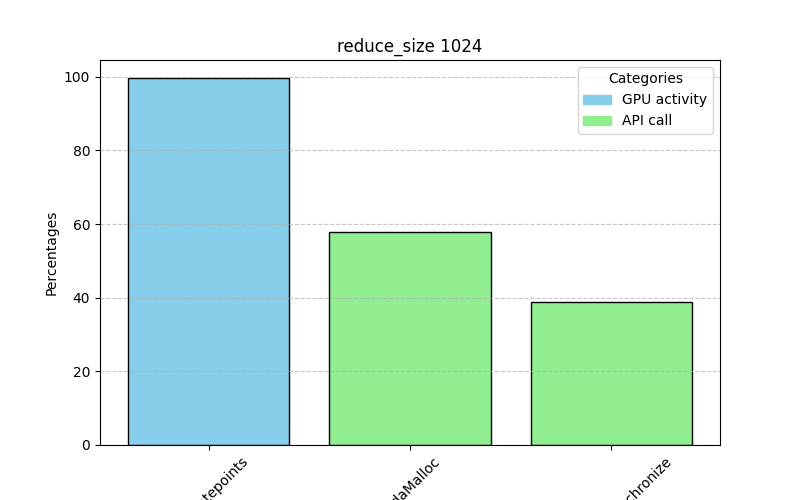
Execution time roughly

On GPU (67 ms, 63 ms, 63 ms)

On CPU ( 172 ms, 176 ms, 164 ms)







Summary for SAXPY:

As one would expect, increasing the input size of the SAXPY vector increases the processing time for each processor (CPU and GPU). The CPU processing time is consistently higher than the GPU processing time which make sense in terms of the effort allocating memory. As the input increased the amount of time processing the data (in the SAXPY Kernel) decreases relative to sending data back and forth between the Host and Device.

An interesting thing to note is that at 1<<20 sized input the time ratio transmitting data from the host to the device is 40% and the time receiving data back from the device to the host 60%, which is the exact reverse for the 1<<30 size input. It would make sense for all DtoH transmissions to take less time than HtoD because half of the data is being transmitted back, (in the size 10 vector input the transfer time is roughly the same either way).

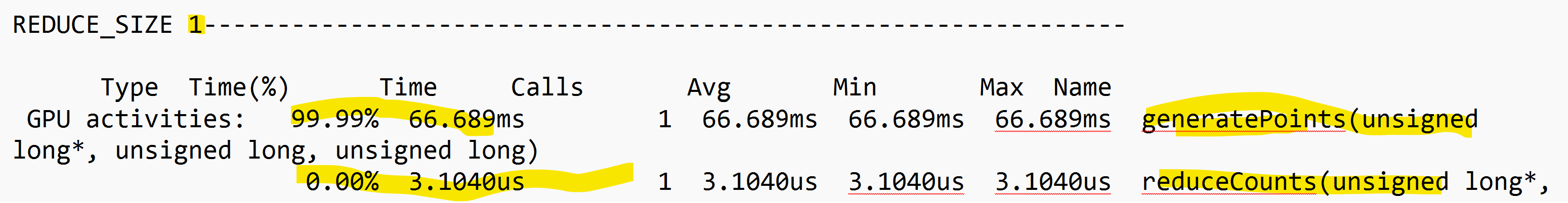
I believe that for lower data transfer sizes some kind of transmit overhead between the device and the host is causing the DtoH memcpy to take longer while the data size is still trivial but that for non trivial sizes the fact that the device is sending back half the amount of memory that the host originally transmitted causes the DtoH to be noticeably less than HtoD.

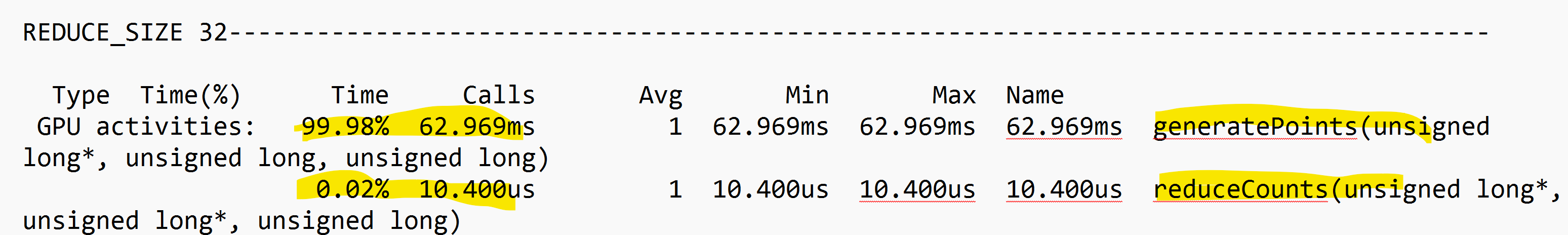
*Note: Both 1<<20\*4Bytes (less than a GB) and 1<<30\*4bytes (4GB) data transfer seem to fall comfortably into the Tesla V100-PCIE-32GB 16GB/S PCIe bandwidth even if we consider that transfer to be mitigated (practically 8 or 4GB/s) I would almost expect there to be no difference even with the larger inputs.*

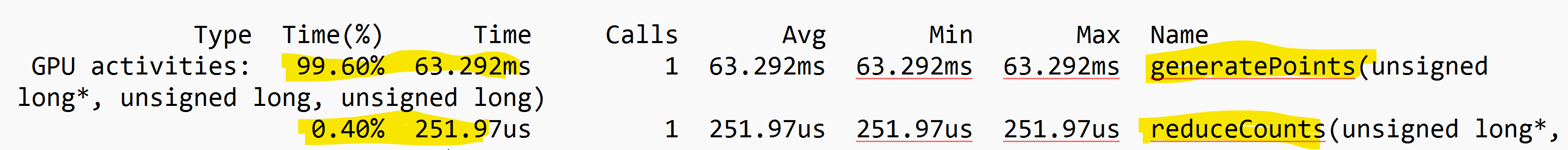
Summary for MCPi:

Varying Generate\_Blocks and SampleSize both had similar effects in that they increased the number of points being created for the MC simulation and therefore both increased the use of the generatePoints kernel and eventually dominated the \_\_synchronize API call.

For Reduce\_Size I never varied it over 1024 because then no REDUCE\_BLOCKS would be generated (REDUCE\_BLOCKS = GENERATE\_BLOCKS/REDUCE\_SIZE) and GENERATE\_BLOCKS default is 1024. The proportions did not change much with the variation of Reduce\_Size, the generatePoints kernel took the majority of GPU time, and the cudaMalloc and synchronize took the majority of the API. I would have expected increasing Reduce\_Size to steal time away from the generatePoints kernel to the reduceCounts kernel and looking at my raw data it did seem to :







The percentage of time in the generatePoints kernel vs the reduceCounts kernel shifted but not dramatically enough to be included in the bar-chart.