Graphics Processing Units - Fall 2018
Homework 1

Daniel Rivera Ruiz
Department of Computer Science
New York University
drr342@nyu.edu

1. To design the next generation GPU, a company has several choices to make:
   1) Increasing number of SM.
   2) Increasing number of SPs (or cuda cores) per SM.
   3) Increasing memory bandwidth.
   4) Increasing shared memory per SM.
   5) Increasing L2 cache size.

   Discuss the pros and cons for each one of the following scenarios:

Scenario	**Pros**	**Cons**
a) Doing 1	If the number of SPs is fixed, increasing the number of SMs would mean less SPs per SM, and therefore the hardware interconnections in the SM would be simplified.	With less SPs per SM, the number of threads that can communicate with one another, synchronize and share memory is reduced.
b) Doing 2	More SPs per SM would mean faster running applications since there would be more threads communicating, synchronizing and sharing memory.	With more SPs, the hardware required in an SM to support all the interactions among threads would need to be more robust and therefore more expensive.
c) Doing 3	Increasing the memory bandwidth would mean that more data can get to the cores simultaneously, accelerating the overall time of execution of an application.	To increase the bandwidth of the memory a channel of communication with more capacity and less noise is required, which translates to more expensive hardware.
d) Doing 4	More shared memory per SM would mean that the access to GDRAM could be substantially reduced, therefore accelerating the overall time of execution of an application.	More shared memory per SM would also mean less space in the SM for actual SPs. With less processing units, the amount of parallelization would be reduced.
e) Doing 5	Similarly to the previous condition, increasing the L2 cache would reduce access to GDRAM and improve data transfer latency.	A bigger L2 cache would take up more space on the GPU chip, therefore compromising other components (e.g. SPs, shared memory) and possibly overall performance.
f) Doing 1 and 2	Increasing both the number of SPs and SMs would mean more computation and parallelization capabilities.	It would be necessary to compromise other components of the system, use smaller technology (unlikely) or make a larger device.

Scenario	Pros	Cons
g) Doing 2 and 3	Applications could run much faster because there would be more cores to perform computations, plus the memory infrastructure to provide them with data fast enough.	The device would definitely need to be larger and more expensive to manufacture, due to the additional/more sophisticated components required.
h) Doing 1 and 4	More SMs (with fix total SPs) would mean less complex hardware interconnections, plus the extra shared memory could reduce the number of data transfers to/from GDRAM.	With less SPs per SM, the number of threads that can actually benefit from the larger shared memory is smaller.
i) Doing 2 and 4	With more SPs, the number of threads that benefit from the larger shared memory would increase and therefore the application would run faster.	Other components of the device would have to be compromised or the device made larger and therefore more expensive.
j) Doing 4 and 5	With more shared memory and L2 cache, the access to GDRAM could be substantially reduced and potentially make the application faster.	The computation and parallelization capabilities would be compromised (less SPs to make space for more memory) and potentially make the application slower.
k) Doing 3 and 5	The bigger L2 cache would make the access to GDRAM less often, plus the increased bandwidth would allow for more data to be fetched every time.	The hardware would be more expensive, and the device bigger (or other components would have to be compromised). Besides, the cores might not take full advantage of the improved memory capabilities.

2. Let's assume an application has a lot of independent and similar operations to be performed on data. Does the amount of data has anything to do with the expected performance of the GPU? Justify.

   The amount of data has a lot to do with the expected performance of the GPU. The advantages that the GPU provides rely strongly on its SIMD (single instruction, multiple data) architecture. This means that the GPU will schedule the same set of instructions to be executed in parallel over different portions of data. Therefore, if there is not enough data to feed the GPU, the execution of the application might even be slower than in a CPU. The number of cores in a GPU is much larger than that of a CPU, but the cores are also simpler and so they have to be utilized simultaneously (with different data) to profit from them. Also, the overhead generated by transferring data from the CPU to the GPU, and from the GPU's memory to the execution cores will not be worth it if there is not enough data to work with, i.e. it will take more time to transfer the data than to actually process it.

3. For each of the following applications state whether it is beneficial to implement them on a GPU and justify your answer.

   a) Finding whether a number exist in an array of 10M integers.
   It is beneficial. The process can be easily parallelized by comparing the number to several items in the array at once. It might even have additional functionality by returning to the CPU all the indices in the array where the number was found instead of just the index of the first occurrence.

   b) Calculating the first 1M Fibonacci numbers.
   It can be beneficial. Depending on the algorithm used, it can be beneficial to calculate the first 1M Fibonacci numbers in a GPU. If a closed-form formula (like this one) is used, then it is easy to calculate several numbers independently and therefore in parallel. However, if the regular recursive definition is used, it is less likely that a GPU can help because the calculations must be done in order and are dependant from one another.

   c) Multiplying two 100x100 matrices.
   It can be beneficial. This one will depend largely on the CPU and GPU available, as well as the

connections between them. The actual multiplication will definitely be faster in the GPU, since several elements of the resulting matrix can be calculated in parallel. However, depending on the overhead generated by data transfer latency: CPU Memory $\rightarrow$ GPU Memory $\rightarrow$ GPU cores, it might take longer for the overall process to complete in the GPU than in the CPU alone.

d) Adding 1Mx1M matrices.
It is beneficial. In this case the amount of data is big enough to overcome the latency of the system, and since all the addition operations are independent from one another, they can easily be performed in parallel in the GPU.

4. We said in class that communication between system memory and CPU memory is very expensive in terms of latency, which negatively affects performance. Why not having one main memory for both the CPU and GPU? State twp reasons at least.

   - Memory in the CPU and the GPU are optimized for different things, and therefore it would be difficult to have one unified memory to serve them both. While in the CPU low latency is crucial for sequential operations over one piece of data, in the GPU the main focus is high bandwidth, to be able to supply all the cores with different portions of the data on which to perform operations.
   - Having only one memory for both CPU and GPU would make it much more difficult to ensure data integrity. If we had all the cores from the CPU plus all the cores from the GPU trying to read and write data concurrently from the same memory chip, the efforts required in both hardware and software would be much larger in order to satisfy all the memory access petitions while maintaining data integrity.