Measurement and performance analysis of General Purpose GPU Computing

Sugeerth Murugesan

1 February 2014

Main characteristics of the GPU . We discuss the essential parts of the GPU hardware that makes it unique. The programs are written in PyCuda. Python is simply used to set up the computation and the kernels which are in C. Characteristics of the GPU:

1 Maximising Floating point operations per second (FLOPS)

By maximising FLOPS we mean to either increase the math per thread or the reduce the memory per thread.This is given by the equation (FLOPS=Computations per thread/Memory used by the thread).By refining it, we could tell that the program that minimizes the time spent on the memory per thread and the program that maximizes compute operations per thread achieves maximum FLOPS. One of the strategy is to move frequently accessed data to fast memory. Essentially, we have an amount of memory which is made global by passing the pointer to the GPU to do operations and return them to the host (CPU) processor. The number of math operations that we do per thread is 12 and the total time taken to do the math is 0.000823s. We do more number of arithmetic operations per thread to maximise the FLOPS.

2 Maximising the Graphics memory Bandwidth

The program characterizes by utilizing many memory references per thread. This is possible as the operations would be able to read adjacent memory locations using the parallel data cache. All the threads in a half-warp access global memory at the same time. In other words, consecutive threads access consecutive memory addresses. For example if threads 0,1,2, and 3 read global memory 0x0, 0x4, and 0x8, it will be a coalesced read. Each thread makes many accesses. Hence, we achieve the maximum memory bandwidth on our hardware.

3 Thread vs Performance

The program that we use is an arithmetic calculator where many comptations are initiated simultaneously. One has to find the size of the blocks/grid to match the data, at the same time he also has to think about maximizing occupancy. The thread block size is always a mltiple of 32(Warps). If one runs 50 threads, the GPU will still issue commands to 64 threads. In this figure, we try to describe how thread blocks affects performance. The maximum shared memory per block is found to be 48k. Hence declaring a shared memory of partition that is more than 24k would ensure that only one block is running at a time on all streaming multiprocessor. Large thread blocks in general have less instrction cache and stall at sync threads. From the graph it is visible that the SM in the processor is found to be 2. The time(ms) is levels off once there are 2 thread blocks.This indicates that that it would be highly inefficient to leave an SM idle by initiating 1 thread block per grid.

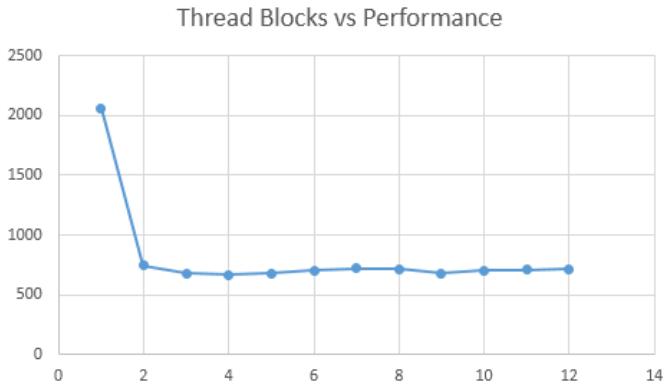

Figure 1: The Thread vs Time(ms) graph

4 Warps vs Performance

Our program of interest calculates the Manhattan Distance from a particular point. The Manhattan Distance or city block distance is found out when the "Kernels" are launched and are operating. We configured the number of blocks such that there is maximum performance. As the Cuda Kernel in Python implementation only computes on numpy arrays all the data are stored in a numpy array and sent to the kernel for computation.

Each Streaming Multiprocessor(executes thread blocks) has 1-4 warp.The more active a warp is ,the larger its ability to be selected in the next cycle. In most of the cases, we can see that the optimal performance is achieved when there are sufficient number of active warps per SM(atleast 1).

Increasing the occupancy beyond the 1 warp elligible condition may not necessarily increase the performance. In the Figure 2, the computation time is more or less a constant when the number of warps increases.

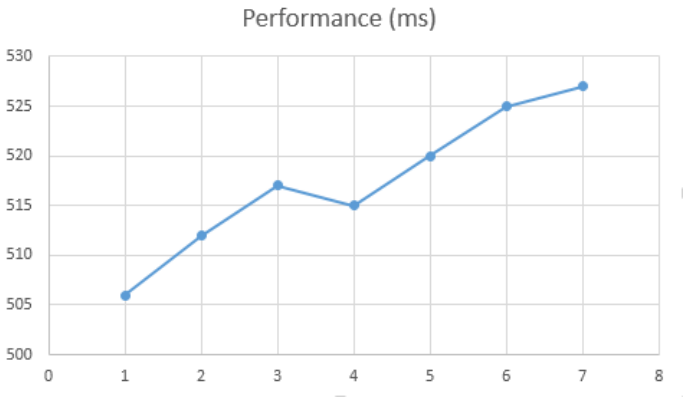

Figure 2: Warp vs Time graph

5 Branch granlarity

Here we are trying to find out the effect of branching with the performance. Ideally, if branching occurs in a warp, the threads would have to wait for each other to execute smultaneously. The use of sync threads ensures a barrier act among the threads. The compiler uses predication to eliminate branch or to change the warp. The process of bookkeeping for diverging branches may also add instructions. We utilized the 'mod' function in the kernnel to determine the vary the probability of the branches in the instruction queue. For example, a mod 2 would indicate that branches are often taken alternatively. Which may not necessarily be two independent divergent paths as we could rearrange our data so that the branches occurs in the pixel boundary. The figure suggests that performance loss is most seen in the branch statements within warps. As the complexity of the branch increases, the time taken is also increased. Hence, care should be taken to avoid divergence within warps. It is the second-most source of performance loss after non-coalesced global memory access.

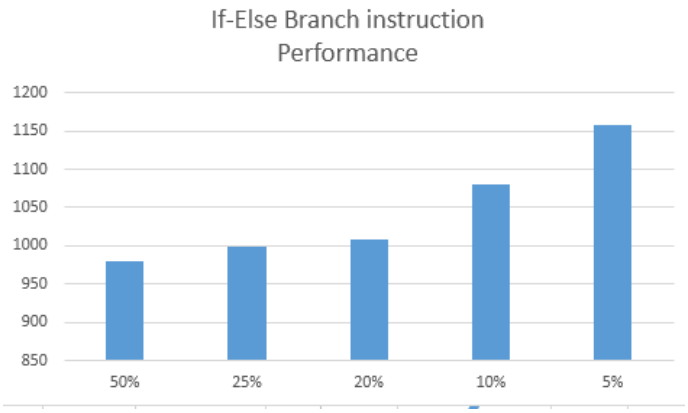

Figure 3: X-axis: Probabilty that the branch is taken, Y-axis: Time in (ms)