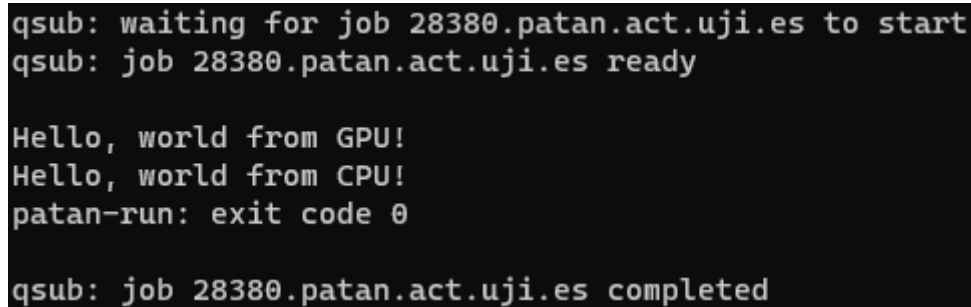


CUDA Exercises

1. Complete the file "hello_world.cu", so that a single thread was created, with no task, whereas the main prints the message.



```
qsub: waiting for job 28380.patan.act.uji.es to start
qsub: job 28380.patan.act.uji.es ready

Hello, world from GPU!
Hello, world from CPU!
patan-run: exit code 0

qsub: job 28380.patan.act.uji.es completed
```

Figure 1. Output in patan server for the hello_world.cu script.

2. Complete the file "pi_par_loop.cu", so that each thread accumulates all its calculations on a local variable and the result is obtained by accumulating all these values.

First compute the reduction in each warp by means of a single loop, and use atomic operations to accumulate the local result on final pi.

Execute the code in patan changing the number of threads, from 256 to 1024 using powers of 2, also changing the number of blocks, from 1 to the maximum number of blocks of the problems, and the number of rectangles {1e7, 5e7, 1e8, 5e8}.

Show the speed-ups obtained and your conclusions.

The results (see attached spreadsheet named results.xls) showcases the impact of varying thread and block configurations on the speedup achieved for different problem sizes in the pi_par_loop.cu code. Observations reveal that increasing the number of threads and blocks generally results in higher speedups, indicative of improved GPU resource utilization and parallelism. However, this scaling demonstrates diminishing returns, suggesting potential constraints within the GPU architecture or algorithmic efficiency.

Furthermore, it is noted that enlarging the problem size tends to marginally reduce speedup, possibly attributable to increased overhead or resource contention. Nonetheless, the execution time per rectangle remains relatively stable across different problem sizes, indicating consistent algorithmic efficiency.

The importance of striking a balance between threads and blocks for optimal performance is emphasized. Configurations with a greater number of blocks tend to exhibit superior speedups, underscoring efficient resource employment. However, achieving the optimal

configuration necessitates weighing considerations of parallelism, overhead, and resource allocation.

3. Complete the file “pi_par_unroll.cu”, so that each thread accumulates all its calculations on a local variable and the result is obtained by accumulating all these values.

First compute the reduction in each warp unrolling the loop, and use atomic operations to accumulate the local result on final pi.

Execute the code in patan changing the number of threads, from 256 to 1024 using powers of 2, also changing the number of blocks, from 1 to the maximum number of blocks of the problems, and the number of rectangles {1e7, 5e7, 1e8, 5e8}.

Show the speed-ups obtained and your conclusions.

Analyzing the results (see attached spreadsheet named results.xls) across all configurations, the speedup consistently demonstrates improvement with increasing thread counts, suggesting a direct relationship between parallelization and computational efficiency. Notably, the speedup tends to plateau or slightly decline as the number of threads surpasses a certain threshold, indicating diminishing returns on parallelization beyond a certain point.

When considering the impact of block count variation on speedup, a clear trend emerges: higher block counts generally correspond to improved performance. This trend underscores the importance of workload distribution and resource allocation within GPU architectures. However, the magnitude of speedup increase diminishes as block counts escalate, indicative of potential resource contention and diminishing returns on parallelism at larger scales.

Examining the influence of step sizes on speedup reveals a compelling pattern: larger step sizes consistently yield higher speedup values. This observation aligns with expectations, as larger step sizes allow for more substantial computational tasks to be distributed across parallel threads, thereby leveraging the GPU's processing capabilities more effectively. However, as both thread and block counts increase, the rate of speedup improvement diminishes, suggesting the presence of overhead and resource contention within the parallel execution environment.