SA-MIRI 2025

Practice Pe: Distributed GPU Programming

Jakub Seliga (jakub.seliga@estudiantat.upc.edu)
Thomas Aubertier (thomas.aubertier@estudiantat.upc.edu)

Task 6.1 Reflecting on CUDA's Execution Model

Why the same CUDA program runs efficiently on GPUs with different numbers of SMs ?

- CUDA divides work into blocks and threads, not fixed hardware units
- The CUDA runtime automatically assigns blocks to whatever number of SMs (Streaming Multiprocessors) the GPU has
- If a GPU has more SMs, more blocks run in parallel → faster execution; if fewer, they run in sequence but still correctly
- This makes CUDA programs scalable across different GPUs, from small devices to large H100 accelerators

Task 6.1 Reflecting on CUDA's Execution Model

How does the CUDA execution model simplify programming ?

- Developers only specify the grid and block structure, not how the hardware schedules threads
- The runtime and hardware handle scheduling, memory access, and synchronization across SMs automatically
- Programmers don’t need to manage warps, instruction queues, or low-level scheduling
- This abstraction hides hardware complexity, making it easier to write parallel code that runs efficiently everywhere

Task 6.2 Precision Trade-offs: True or False?

- Using FP16 or BF16 can help a program run faster on GPUs that support Tensor Cores.

  True. FP16/BF16 are supported by Tensor Cores, which execute matrix ops much faster than normal cores, so using them can speed up workloads.

- Switching from FP32 to FP16 always produces identical numerical results.

  False. FP16 has less precision and range than FP32, so rounding/truncation can change numerical results.

- Reducing precision can lower the amount of memory required for computations.

  True. Fewer bits per value (FP16 vs FP32) means smaller memory footprint and less bandwidth used.

Task 6.2 Precision Trade-offs: True or False?

- All calculations in a GPU program must use the same precision format.

  False. Programs commonly use mixed precision (different parts in different formats); GPUs and libraries support mixing formats.

- Developers can benefit from Tensor Cores without writing any specialized GPU code.

  True. High-level libraries and frameworks (cuBLAS, cuDNN, PyTorch, etc.) will automatically use Tensor Cores when the data types permit, so you don’t have to write special GPU code.

Task 6.3 Submit and Validate the First Performance Run

Build the binary and submit a job:

`$ make clean` (inside slurm file)

`$ make default` (inside slurm file)

`$ sbatch jacobi-4gpus.slurm`

Job ran successfully, no execution errors occurred:

```
[nct01042@alogin1 Chapter.06]$ cat jacobi-4gpus_30734761.err
[nct01042@alogin1 Chapter.06]$
```

Task 6.3 Submit and Validate the First Performance Run

```
Single GPU jacobi relaxation: 100000 iterations on 16384 x 16384 mesh with norm check every 100 iterations
    0, 31.999014
10000, 0.028435
20000, 0.016827
30000, 0.012391
40000, 0.009989
50000, 0.008403
60000, 0.007348
70000, 0.006515
80000, 0.005878
90000, 0.005409
Jacobi relaxation: 100000 iterations on 16384 x 16384 mesh with norm check every 100 iterations
    0, 31.999008
10000, 0.028542
20000, 0.016955
30000, 0.012502
40000, 0.010074
50000, 0.008512
60000, 0.007427
70000, 0.006608
80000, 0.005973
90000, 0.005471
Num GPUs: 4.
16384x16384: 1 GPU: 648.5866 s, 4 GPUs: 165.9215 s, speedup:     3.91, efficiency:    97.72
```

The computed speedup of 3.91 is close to the theoretical maximum (4), and the efficiency of 97.72% indicates great load distribution and efficient utilization of available resources

Task 6.4 Understand the Metrics Collection

- Program runs 2 times in each execution: it first runs the computation on 1 GPU, then repeats it using n GPUs (4 in provided SLURM file).
- The L2 norm is following the same evolution between both tests : they output the same results and are thus comparable.
- Time of execution is measured using MPI_Wtime().
- After both runs finish, the program automatically computes speedup and efficiency.
- `efficiency = 97.72%` => close to 100% => good parallelization.

Task 6.5 Explore the Effect of Optimized Compilation Flags

Now, with a change of `make default` to `make optimized` :

```
Single GPU jacobi relaxation: 100000 iterations on 16384 x 16384 mesh with norm check every 100 iterations
    0, 31.999010
10000, 0.028568
20000, 0.016980
30000, 0.012523
40000, 0.010089
50000, 0.008531
60000, 0.007439
70000, 0.006624
80000, 0.005991
90000, 0.005483
Jacobi relaxation: 100000 iterations on 16384 x 16384 mesh with norm check every 100 iterations
    0, 31.999008
10000, 0.028570
20000, 0.016982
30000, 0.012525
40000, 0.010091
50000, 0.008533
60000, 0.007440
70000, 0.006626
80000, 0.005993
90000, 0.005484
Num GPUs: 4.
16384x16384: 1 GPU: 191.9530 s, 4 GPUs:  50.6825 s, speedup:     3.79, efficiency:    94.68
```

Task 6.5 Explore the Effect of Optimized Compilation Flags

- Speedup and efficiency are a bit worse than those achieved using `make default` (≈ -3%), but the overall execution time is far better (≈ x3.5).
- `make optimized` > `make default` , at least for this grid size.

Task 6.6 Evaluate the Impact of the CUB Library

Now, with a change of `make default` to `make cub` :

```
Single GPU jacobi relaxation: 100000 iterations on 16384 x 16384 mesh with norm check every 100 iterations
    0, 31.999022
10000, 0.028571
20000, 0.016983
30000, 0.012525
40000, 0.010092
50000, 0.008534
60000, 0.007441
70000, 0.006626
80000, 0.005993
90000, 0.005485
Jacobi relaxation: 100000 iterations on 16384 x 16384 mesh with norm check every 100 iterations
    0, 31.999023
10000, 0.028571
20000, 0.016983
30000, 0.012525
40000, 0.010092
50000, 0.008534
60000, 0.007441
70000, 0.006626
80000, 0.005993
90000, 0.005485
Num GPUs: 4.
16384x16384: 1 GPU: 178.0728 s, 4 GPUs:  47.2049 s, speedup:     3.77, efficiency:    94.31
```

Task 6.6 Evaluate the Impact of the CUB Library

- Speedup and efficiency are tadly the same than those achieved using `make optimized` (-0.37%), but the computation time is slightly better (≈ 7%).
- Both options are close in performances, but `make cub` works better for low numbers of GPUs : preferable for Task 6.7.

Task 6.7 Benchmarking the Impact of GPU Count and Problem Size

- Efficiency decreases with the number of GPUs (Amdahl's law), but do better with big instances (Gustavson's law).