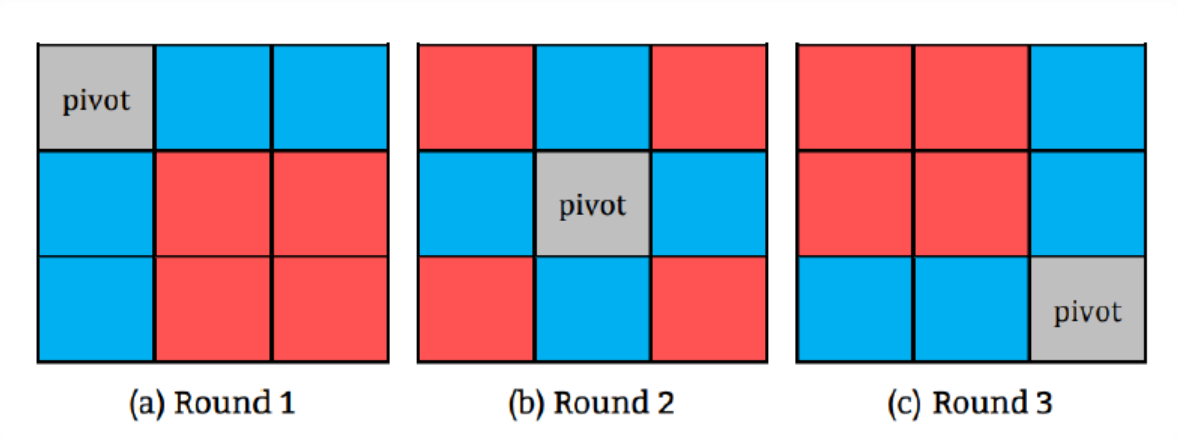
2024 Parallel Programming HW3 [112062610 劉得崙]

> Blocked Floyd-Warshall illustration
>
> 
>

1. Implementation

a. Which algorithm do you choose in hw3-1?

> I've tried these with full testcases:
>
> - Nornal FW (+omp) -> 73.25 s
> - Dijkstra's algo. (priority queue) -> 40.96 s
> - Dijkstra's algo. (binary heap) -> 33.99 s
> - Dijkstra's algo. (Fibonacci heap) -> 38.96 s
> - Blocked FW (+omp) -> 54.93 s
> - Blocked FW (+omp +SSE2 +unroll) -> 18.04 s (chosen)

b. How do you divide your data in hw3-2, hw3-3?

> `B (FW_BZ) = 78`
>
> `BlockDim.x/y (CUDA_BZ) = 26`

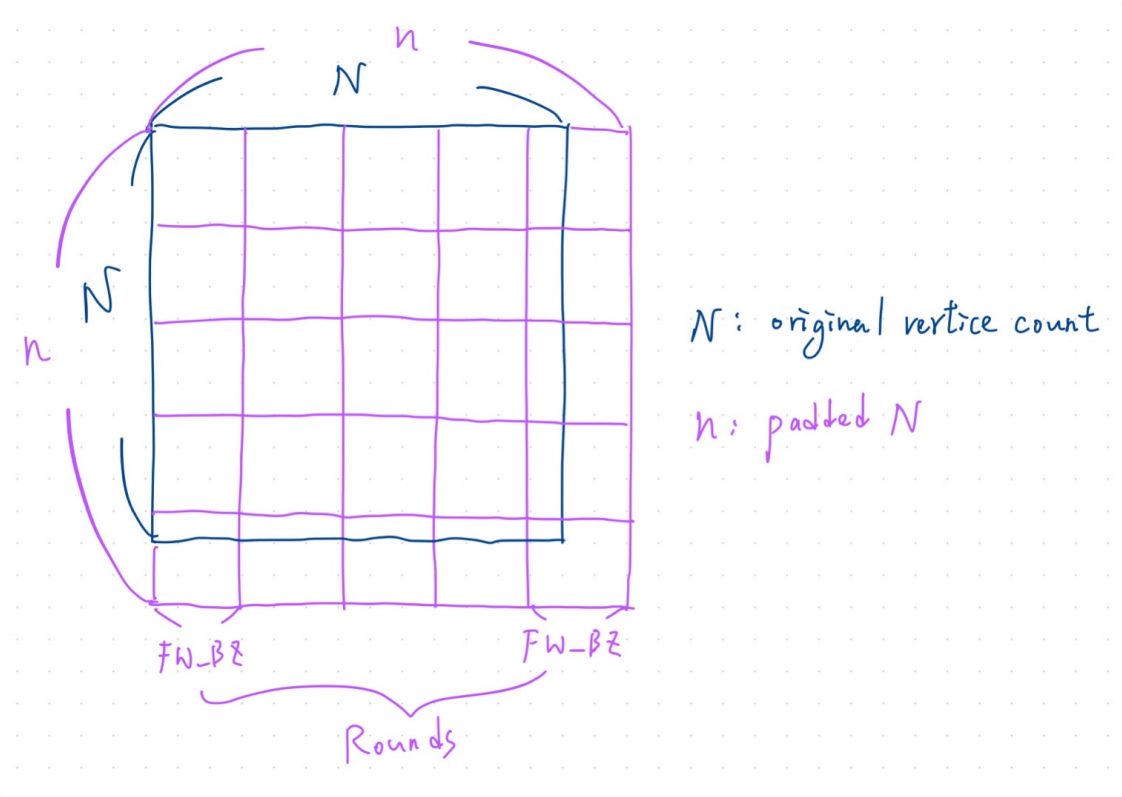
c. What's your configuration in hw3-2, hw3-3? And why?

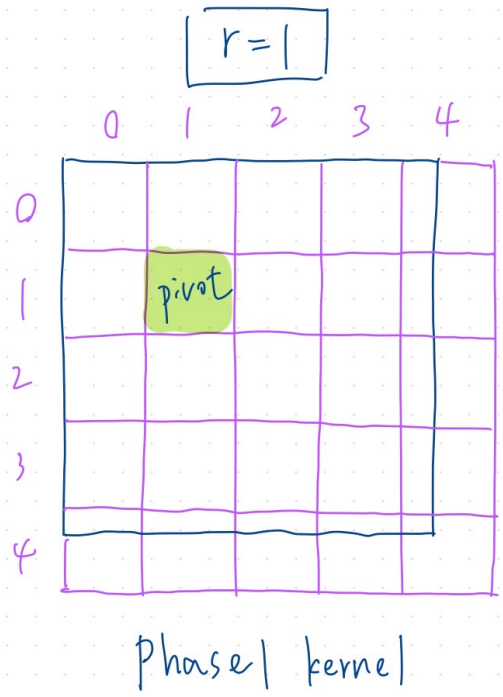
> `Round = N / FW_BZ` (N has been padded to FW_BZ's multiple)
>
> `dim3 BlockDim(26, 26)` -> 676 threads per block
>
> `dim3 GridDim(Round, Round)` -> for Phase 3 kernel launch
>
> Meaning one thread calculates (78/26) * (78/26) = 9 data, which utilizes the shared memory the best.

d. How do you implement the communication in hw3-3?

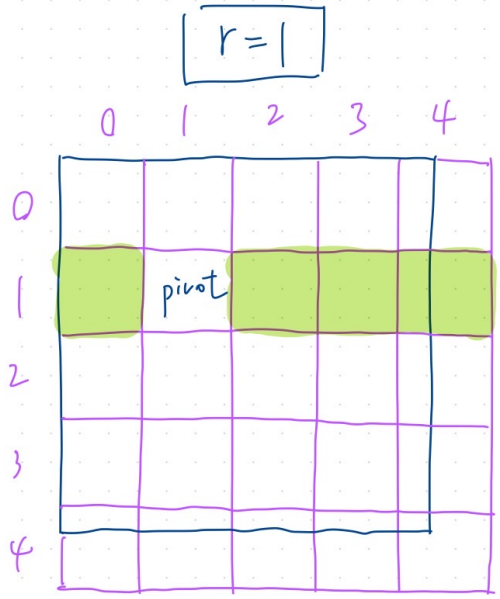
> With `cudaMemcpy(..., cudaMemcpyDeviceToDevice)`, it serves as both synchronization and communication.

e. Briefly describe your implementations in diagrams, figures or sentences.

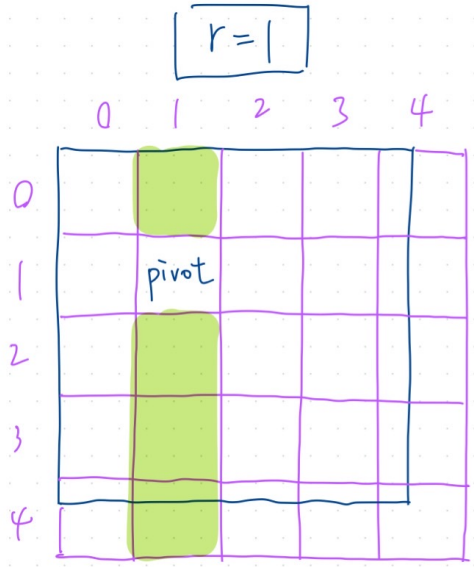
> - Step 1: Padding
>
> 
>
>
> - Step 2: Rounds of kernel launches
>
> ```
> #pragma unroll
> for (int r = 0; r < round; ++r) {
>     phase1_kernel<<<1, block>>>(d_Dist, n, r);
>     phase2_kernel_row<<<dim3(round, 1), block>>>(d_Dist, n, r);
>     phase2_kernel_col<<<dim3(round, 1), block>>>(d_Dist, n, r);
>     phase3_kernel<<<dim3(round, round), block>>>(d_Dist, n, r);
> }
> ```



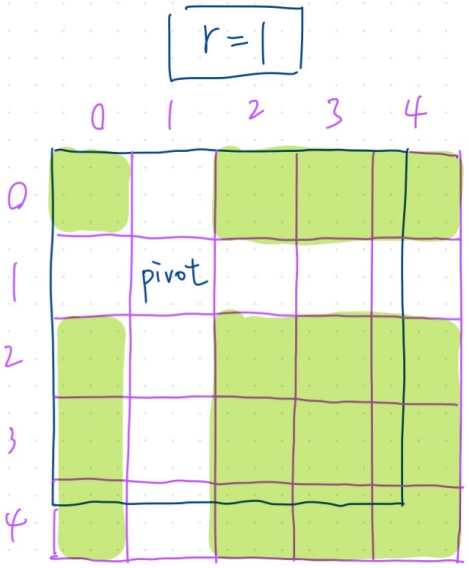
Phase1 kernel



Phase2 kernel (row)

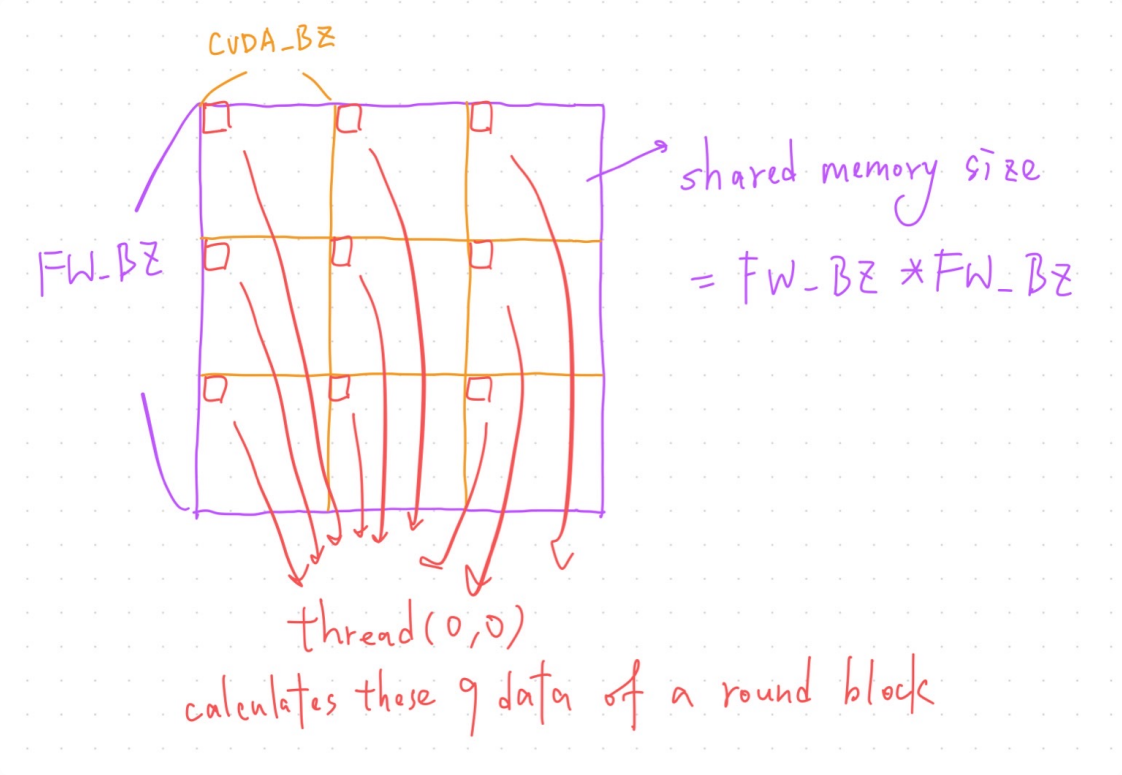


Phase2 kernel (column)



Phase3 kernel

- Step 3: Processes with shared memory



2. Profiling Results (hw3-2)

- Command

```
make hw3-2
srun -pnvidia -N1 -n1 -c2 --gres=gpu:1 nvprof \
--metrics achieved_occupancy,sm_efficiency,\
gld_throughput,gst_throughput,gld_efficiency,gst_efficiency \
./hw3-2 /share/testcases/hw3/p12k1 p12k1.out
```

- Output (phase3_kernel)

Metric	Min	Max	Average
Achieved Occupancy	0.646763	0.649202	0.647960
Multiprocessor Activity	99.88%	99.94%	99.93%
Global Load Throughput	270.97GB/s	277.55GB/s	274.80GB/s
Global Store Throughput	116.25GB/s	119.07GB/s	117.89GB/s
Global Memory Load Efficiency	71.01%	71.01%	71.01%
Global Memory Store Efficiency	55.17%	55.17%	55.17%

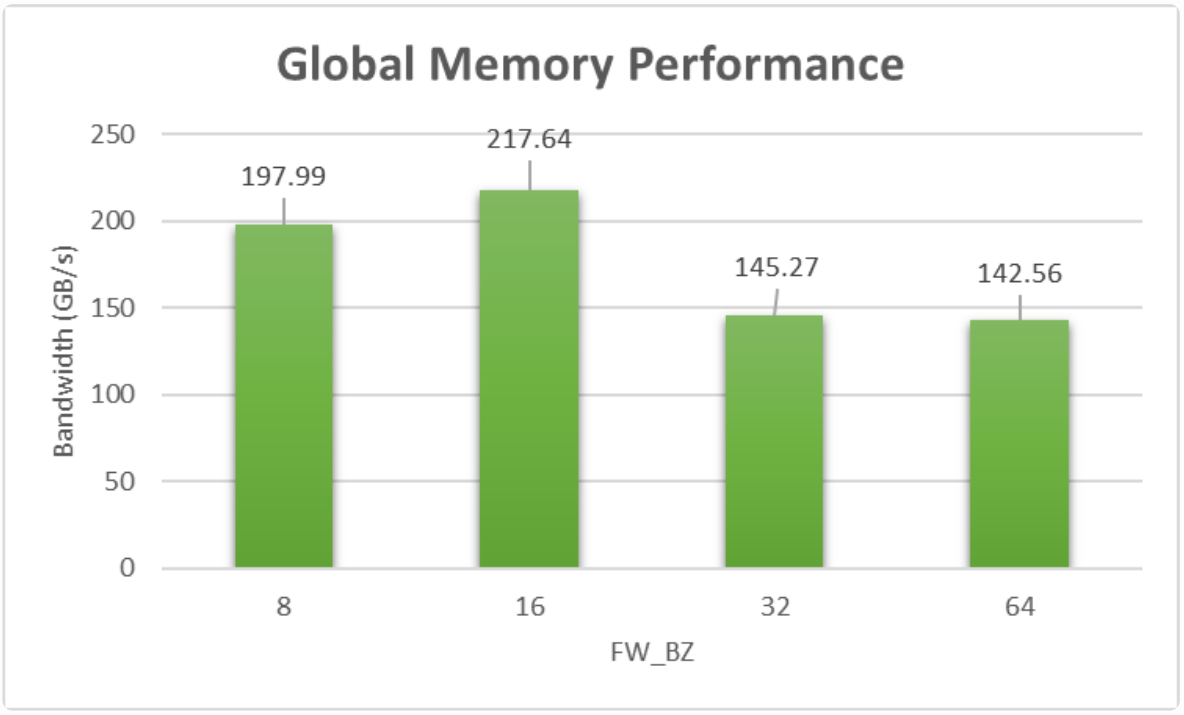
3. Experiment & Analysis

a. System Spec

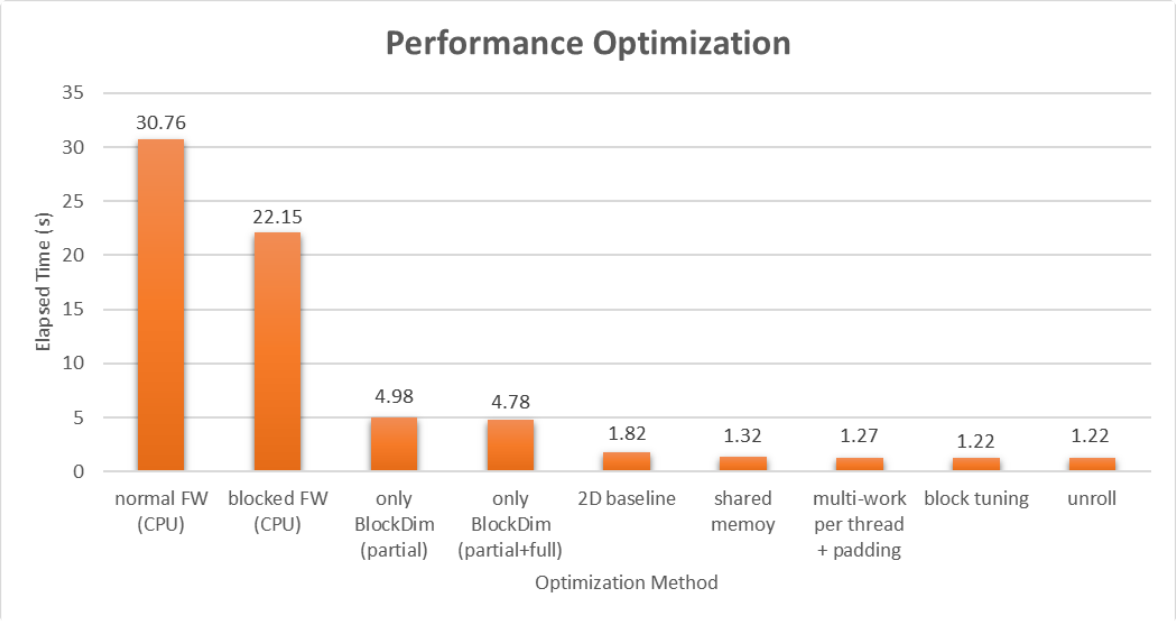
> Apollo-gpu workstation

b. Blocking Factor (hw3-2)

> - Command
>
> ```
> // nvprof
> srun -pnvidia -N1 -n1 -c2 --gres=gpu:1 \
> nvprof --metrics inst_integer,gld_throughput \
> ./hw3-2 /share/testcases/hw3/c08.1 c08.1.out\
>
> // nsys
> srun -pnvidia -N1 -n1 -c2 --gres=gpu:1 \
> nsys profile --trace=cuda \
> ./hw3-2 /share/testcases/hw3/c08.1 c08.1.out
> ```

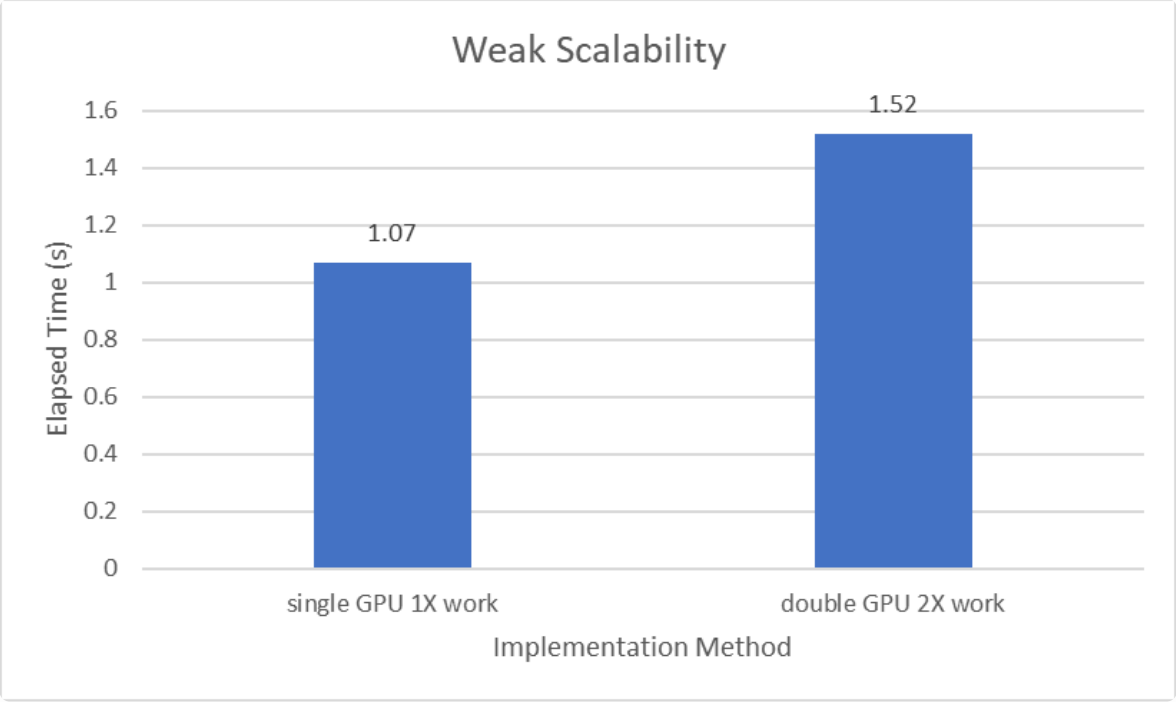


c. Optimization (hw3-2)

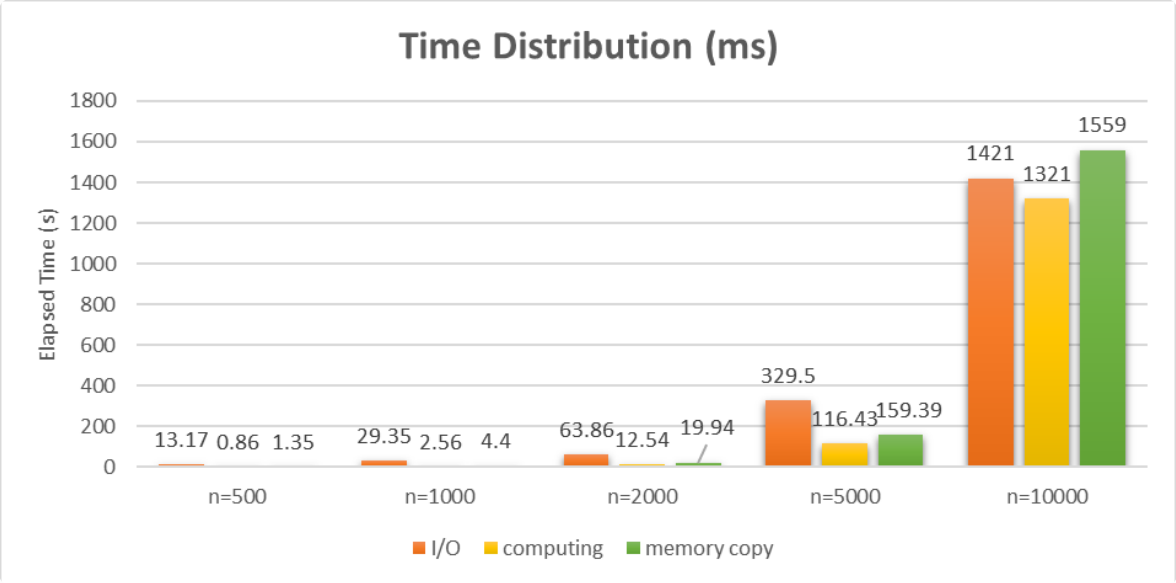


d. Weak Scalability (hw3-3)

	single GPU (c21.1, n=5000)	double GPU (p12k1, n=10000)
Time	1.07 s (1X work)	3.04 s (4X work) -> 1.52 s (2X work)



e. Time Distribution (hw3-2)



4. Experiment on AMD GPU

- Difference

nvidia

```
cudaMalloc(&d_Dist, n*n*sizeof(int));
cudaMemcpy(d_Dist, h_Dist, n*n*sizeof(int), cudaMemcpyHostToDevice);
```

```
cudaMemcpy(h_Dist, d_Dist, n*n*sizeof(int), cudaMemcpyDeviceToHost);
cudaFree(d_Dist);
```

amd

```
hipMalloc(&d_Dist, n*n*sizeof(int));
hipMemcpy(d_Dist, h_Dist, n*n*sizeof(int), hipMemcpyHostToDevice);
```

```
hipMemcpy(h_Dist, d_Dist, n*n*sizeof(int), hipMemcpyDeviceToHost);
hipFree(d_Dist);
```

- Execution with AMD GPU

```
// Single GPU
hipify-clang hw3-2.cu
make hw3-2-amd
srun -p amd -N1 -n1 -c2 --gres=gpu:1 \
nsys profile --trace=cuda,nvtx \
./hw3-2-amd /share/testcases/hw3/c21.1 c21.1.out
```

```
pp24s069@apollo-nv-test:~/hw3$ srun -p amd -N1 -n1 -c2 --gres=gpu:1 nsys profile --trace=cuda,nvtx ./hw3-2-amd /share/testcases/hw3/c21.1 c21.1.out
WARNING: CPU IP/backtrace sampling not supported, disabling.
Try the 'nsys status --environment' command to learn more.

WARNING: CPU context switch tracing not supported, disabling.
Try the 'nsys status --environment' command to learn more.

Collecting data...
Generating '/tmp/nsys-report-e426.qdstrm'
[1/1] [========================100%] report1.nsys-rep
Generated:
    /home/pp24/pp24s069/hw3/report1.nsys-rep
```

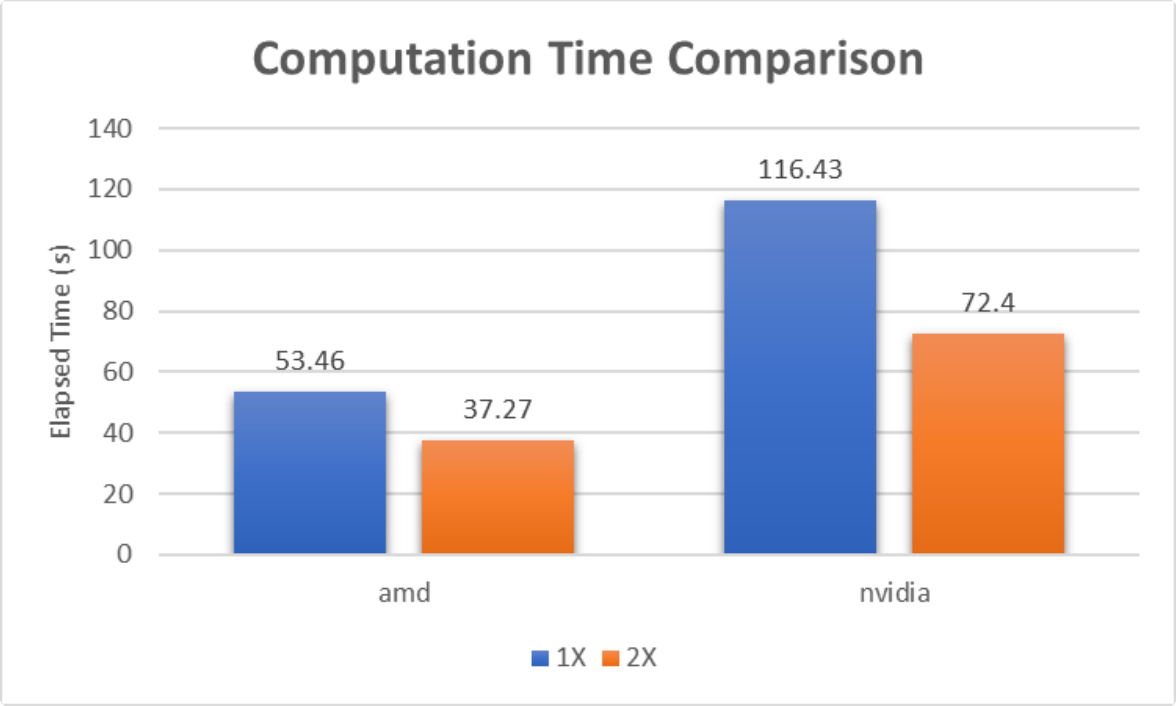
```
// Multi GPU
hipify-clang hw3-3.cu
make hw3-3-amd
srun -p amd -N1 -n1 -c2 --gres=gpu:2 \
nsys profile --trace=cuda,nvtx \
./hw3-3-amd /share/testcases/hw3/c21.1 c21.1.out
```

```
pp24s069@apollo-nv-test:~/hw3$ srun -p amd -N1 -n1 -c2 --gres=gpu:2 nsys profile --trace=cuda,nvtx ./hw3-3-amd /share/testcases/hw3/c21.1 c21.1.out
WARNING: CPU IP/backtrace sampling not supported, disabling.
Try the 'nsys status --environment' command to learn more.

WARNING: CPU context switch tracing not supported, disabling.
Try the 'nsys status --environment' command to learn more.

Collecting data...
Generating '/tmp/nsys-report-0bce.qdstrm'
[1/1] [========================100%] report2.nsys-rep
Generated:
    /home/pp24/pp24s069/hw3/report2.nsys-rep
```

- Comparison



5. Experience & Conclusion

Through this homework, I've gained valuable insights into high-performance parallel computing and GPU programming. Working with the All-Pairs Shortest Path problem provided hands-on experience with different parallel architectures, from CPU threading to single and multi-GPU implementations.

The blocked Floyd-Warshall algorithm implementation taught me the importance of data locality and efficient memory access patterns. Performance optimization was particularly enlightening, as I learned to use profiling tools and understand how different factors like block size, thread configuration, and memory access patterns impact GPU performance.

The multi-GPU implementation highlighted the complexities of device synchronization and workload distribution. Additionally, working with both NVIDIA and AMD GPUs exposed me to the nuances of different GPU architectures and the challenges of cross-platform GPU computing.

This practical experience has deepened my understanding of parallel algorithm design and the critical role of hardware-aware optimization in high-performance computing.