Assignment 2

Group 7

Gaia Forghieri, Fjona Minga
Antonio Pelusi, Alberto Stefani

High Performance Computing a.y. 2022/2023

CUDA parallelization

The LU solver requires matrix operations which consist of many simple operations (i.e. sum + products between array elements)

Thus, this problem is easily parallelizable with the GPU threads, once we assign an element of the matrix to each thread.

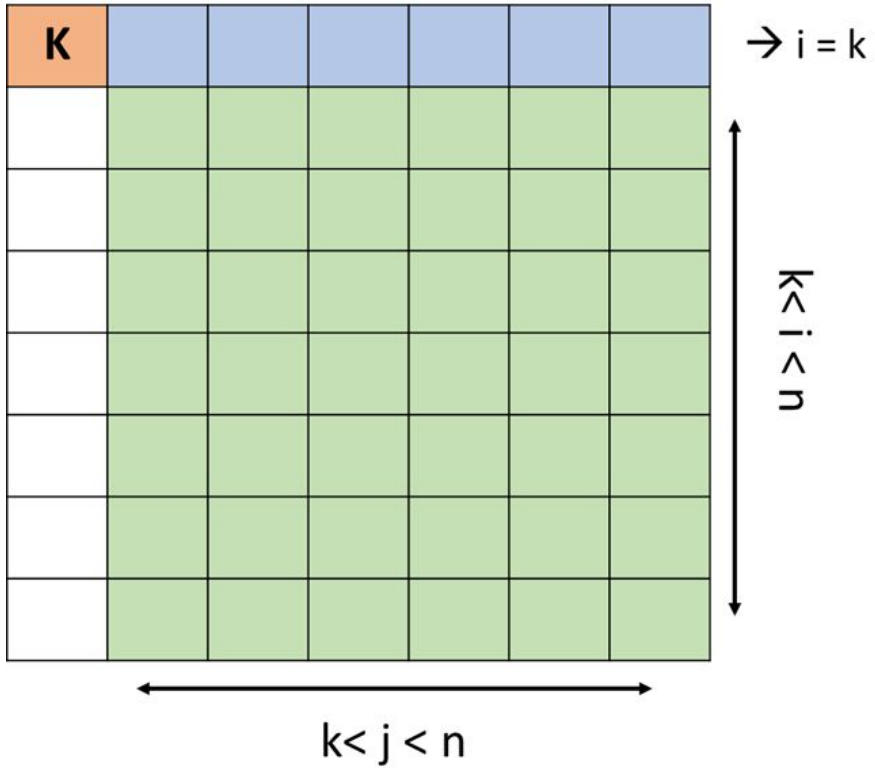
The main difficulty of the problem is having large datasets, which is the optimal situation for the use of GPUs with CUDA parallelization.

Algorithm

LU solver

For each k:

- Evaluate the line of index k
- Evaluate the submatrix with indices i>k, j>k



Implementation of the code

```
__global__ void gpu_kernel_lu(float *A,int k,int n)
{
    int i = threadIdx.y + blockIdx.y * blockDim.y;
    int j = threadIdx.x + blockIdx.x * blockDim.x;

    if (i<n && j<n && i>k && j>k)
    {
        if(i==n-1)
        {
            A[k*n+j] = A[k*n+j] / A[k*n+k];
        }
        __syncthreads();


        A[i*n+j] = A[i*n+j] + A[i*n+k] * A[k*n+j];
    }
}
```

Indices run on the submatrix

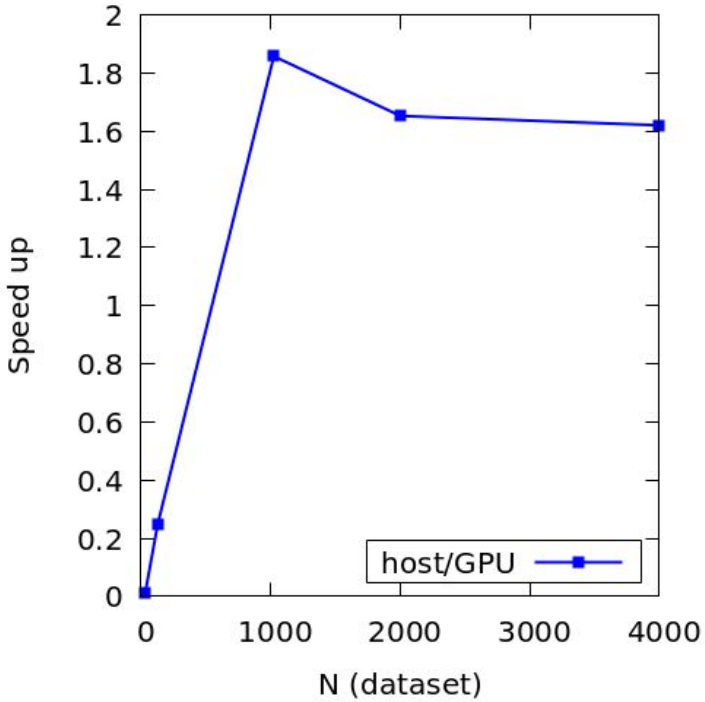
Fix i=n-1 for the first cycle (to run only once):

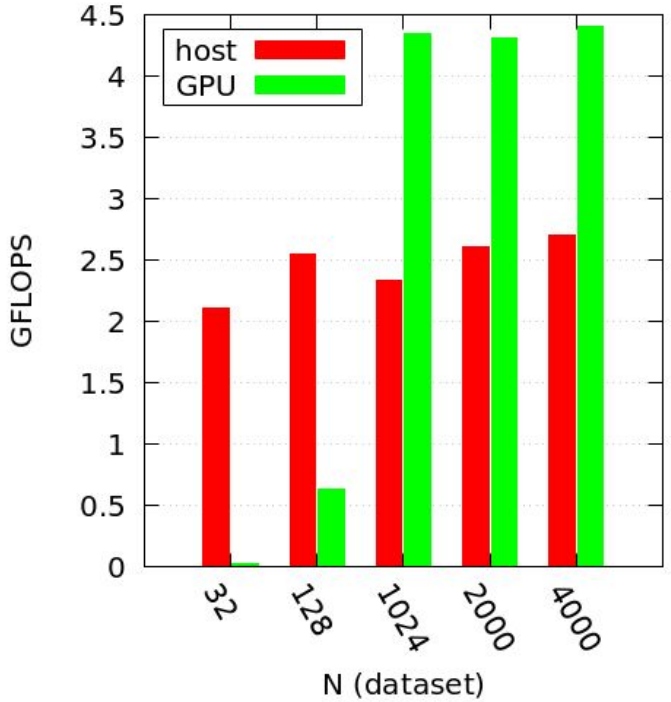
$$A_{k,j} = A[k * n + j]$$

Both indices used for second cycle:

$$A_{i,j} = A[i * n + j]$$

Speed-up with GPU





UVM

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	98.24%	847.78ms	1024	827.91us	401.41us	10.760ms	gpu_kernel_lu(float*, int, int)
	1.21%	10.447ms	1	10.447ms	10.447ms	10.447ms	[CUDA memcpy DtoH]
	0.55%	4.7505ms	1	4.7505ms	4.7505ms	4.7505ms	[CUDA memcpy HtoD]
API calls:	97.02%	31.8952s	1	31.8952s	31.8952s	31.8952s	cudaMalloc
	2.50%	820.52ms	2	410.26ms	4.8794ms	815.64ms	cudaMemcpy
	0.34%	111.21ms	1	111.21ms	111.21ms	111.21ms	cudaDeviceReset
	0.14%	45.957ms	1024	44.880us	31.615us	250.32us	cudaLaunchKernel

N=1024
1.8%

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	99.71%	38.5796s	4000	9.6449ms	6.1111ms	138.91ms	gpu_kernel_lu(float*, int, int)
	0.18%	68.212ms	1	68.212ms	68.212ms	68.212ms	[CUDA memcpy HtoD]
	0.12%	44.691ms	1	44.691ms	44.691ms	44.691ms	[CUDA memcpy DtoH]
API calls:	61.97%	32.0721s	4000	8.0180ms	33.177us	109.51ms	cudaLaunchKernel
	25.02%	12.9501s	1	12.9501s	12.9501s	12.9501s	cudaMalloc
	12.78%	6.61668s	2	3.30834s	67.785ms	6.54890s	cudaMemcpy
	0.22%	114.81ms	1	114.81ms	114.81ms	114.81ms	cudaDeviceReset

N=4000
0.29%

The time necessary to copy the matrix from the host to the device is negligible with respect to the overall computation time on the GPU, so we can avoid the optimization with the UVM.

Profiling

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	98.24%	847.78ms	1024	827.91us	401.41us	10.760ms	gpu_kernel_lu(float*, int, int)
	1.21%	10.447ms	1	10.447ms	10.447ms	10.447ms	[CUDA memcpy DtoH]
	0.55%	4.7505ms	1	4.7505ms	4.7505ms	4.7505ms	[CUDA memcpy HtoD]
API calls:	97.02%	31.8952s	1	31.8952s	31.8952s	31.8952s	cudaMalloc
	2.50%	820.52ms	2	410.26ms	4.8794ms	815.64ms	cudaMemcpy
	0.34%	111.21ms	1	111.21ms	111.21ms	111.21ms	cudaDeviceReset
	0.14%	45.957ms	1024	44.880us	31.615us	250.32us	cudaLaunchKernel

N=1024
5.4%

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	99.71%	38.5796s	4000	9.6449ms	6.1111ms	138.91ms	gpu_kernel_lu(float*, int, int)
	0.18%	68.212ms	1	68.212ms	68.212ms	68.212ms	[CUDA memcpy HtoD]
	0.12%	44.691ms	1	44.691ms	44.691ms	44.691ms	[CUDA memcpy DtoH]
API calls:	61.97%	32.0721s	4000	8.0180ms	33.177us	109.51ms	cudaLaunchKernel
	25.02%	12.9501s	1	12.9501s	12.9501s	12.9501s	cudaMalloc
	12.78%	6.61668s	2	3.30834s	67.785ms	6.54890s	cudaMemcpy
	0.22%	114.81ms	1	114.81ms	114.81ms	114.81ms	cudaDeviceReset

N=4000
83%

The main overhead comes instead from calling N times the same GPU kernel. Particularly for the largest dataset the average execution times of the CUDA kernel launcher is too high. However, decreasing such time does not change the overall execution time, meaning the results from profiling include some overlap in the time measurements.

Comparison with OpenMP

- Overall the GPU goes faster than the best parallelization with OpenMP performed on the host

- When the dataset is small, the idling time is not negligible with respect to the operation time, thus the total execution time increases