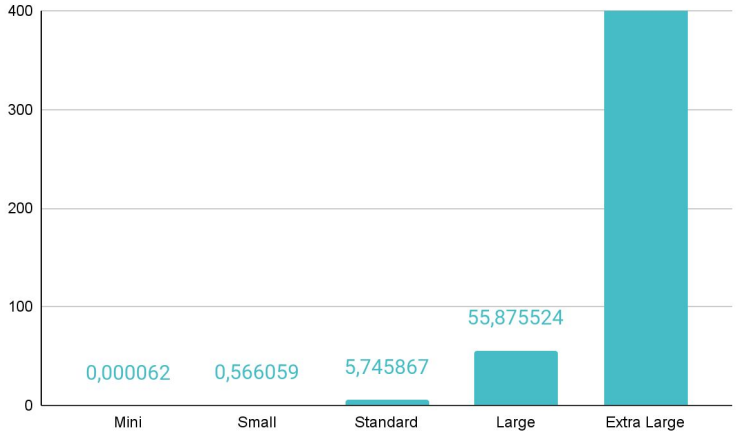
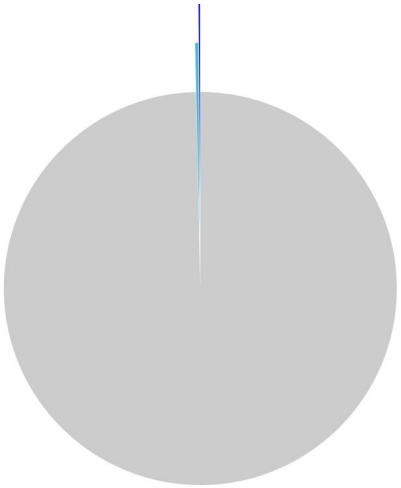
SECOND ASSIGNMENT HPC *COVARIANCE*

Team 3
Mattia Papari, Andrea Tolve, Andrea Grassi, Samuele Mariotti

Profiling original code

Execution time:





99,64%		KERNEL_COVARIANCE
0,24%		POLYBENCH_ALLOC_DATA
0,12%		INIT_ARRAY

Kernel Dimension

DATASETS	Matrix Dimension	Kernel Dimension M	Kernel Dimension N	Num Threads
Mini	1024	32	32	1024
Small	250.000	500	500	262.144
Standard	1.000.000	1000	1000	1.115.136
Large	4.000.000	2000	2000	4.064.256
Extra Large	16.000.000	4000	4000	16.257.024

MAIN

In the main section, we initialized memory using Unified Virtual Memory (**UVM**).

We set the **block** and **grid** size and launched the **kernels** .

```
cudaMallocManaged((void**)&h_flags,     M         * sizeof(int));
cudaMallocManaged((void**)&h_symmat, M * M    * sizeof(DATA_TYPE));
cudaMallocManaged((void**)&h_mean,    M * M    * sizeof(DATA_TYPE));
cudaMallocManaged((void**)&h_data,    M * N    * sizeof(DATA_TYPE));
cudaMemset(h_symmat, 0,         M * M    * sizeof(DATA_TYPE));
cudaMemset(h_mean,    0,         M        * sizeof(DATA_TYPE));
cudaMemset(h_flags,    0,         M        * sizeof(int));

dim3 dimBlock(BLOCK_DIM,  BLOCK_DIM);
dim3 dimGrid(((N+BLOCK_DIM-1)/BLOCK_DIM),
                ((N+BLOCK_DIM-1)/BLOCK_DIM));
kernel_calc_mean<<<dimGrid, dimBlock>>>(h_data, h_mean);
kernel_covariance<<<dimGrid, dimBlock>>>(h_data, h_mean, h_flags,
                                                    float_n,  h_symmat);
cudaDeviceSynchronize();
```

INDEX SELECTION

Here, we introduce the indices used in each of CUDA's global functions.

These indices are unique for each thread started during execution.

The functions:

→ ***__global__*** void **kernel_calc_mean**
→ ***__global__*** void **kernel_covariance**

```
int tidy = threadIdx.y;
int tidx = threadIdx.x;
int j = blockIdx.x * blockDim.x + tidx;
int i = blockIdx.y * blockDim.y + tidy;
```

INITIALIZATION

In this section we can obtain the correct initial data for each block inside the **device** , synchronized by *__syncthreads()*.

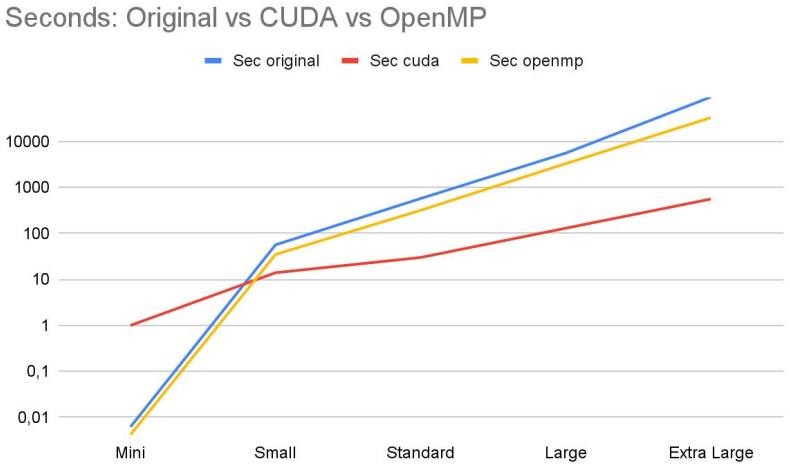
```
__global__ void kernel_calc_mean(  DATA_TYPE* data,
                                   DATA_TYPE* mean)
{
        //initialize index for each threads (i and j)
        if (i <  M && j <  N) {
                data[i *  M + j] = ((DATA_TYPE) i * j) /  M;
                __syncthreads();
                atomicAddDouble(&mean[j], data[i *  M + j]);
        }
}
```

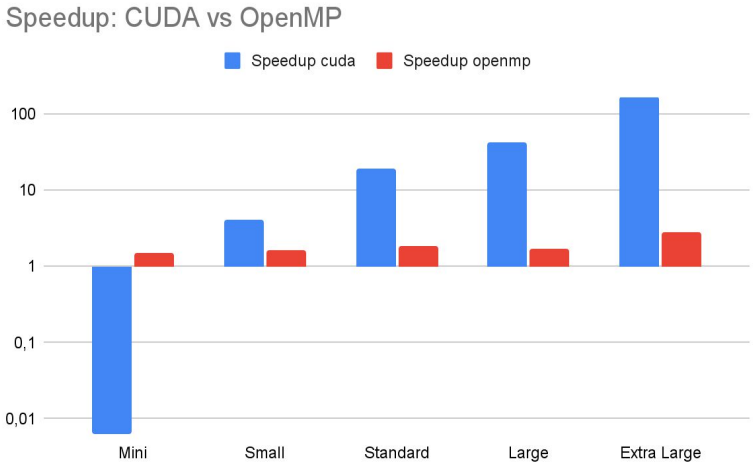
COMPUTATION

In this section of our code we want to show what we had done to calculate the covariance matrix.

```
__global__ void kernel_covariance( DATA_TYPE* data,
                                   DATA_TYPE* mean,
                                   int* flags,
                                   DATA_TYPE float_n,
                                   DATA_TYPE* symmat) {
  //initialize index for each threads (i and j)
  if (i <  M && j <  N) {
    if (atomicCAS(&flags[j],0,1) == 0)
        mean[j] /= float_n;
     __syncthreads();
    atomicAddDouble(&data[i *  M + j], -(mean[j]));
    if (j <= i){
                DATA_TYPE sum = 0.0;
                for (int k = 0; k <  N; k++)
                    sum += (data[k *  M + j] * data[k *  M + i]);
                symmat[j *  M + i] = sum;
                symmat[i *  M + j] = sum;
    }
  }
}
```

RESULT


Seconds: Original vs CUDA vs OpenMP
Sec original
Sec cuda
Sec openmp
10000
1000
100
10
1
0,1
0,01
Mini
Small
Standard
Large
Extra Large


Speedup: CUDA vs OpenMP
Speedup cuda
Speedup openmp
100
10
1
0,1
0,01
Mini
Small
Standard
Large
Extra Large

THANKS

Mattia Papari Mat. 201712

Andrea Tolve Mat. 169302

Samuele Mariotti Mat. 146470

Andrea Grassi Mat. 196608