# Exercise 03 - Dot product

In this exercise we will consider doing a vector dot product in parallel and compare the performance of a sequential computation to find the break-even point in terms of vector size from which parallel execution pays off. The scalar value $c$ is determined as a vector dot product between two vectors $\mathbf{a} = (a_1, a_2, ..., a_N)$ and $\mathbf{b} = (b_1, b_2, ..., b_N)$ and can be expressed as

$$c = \mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{N} a_i b_i$$

where the size of the vectors $N$ can be of arbitrary size. The vector dot product (also referred to as an inner product) is a *reduction* or *gather* operation where each operation is not independent on all previous operations.

## Concepts covered

The following concepts are covered in this exercise

- How to time parts of code execution.
- How to allocate device memory.
- How to copy data from CPU to GPU and from GPU to CPU.
- How to compute dot product for arbitrary size $N$ utilizing maximum number of threads on device.
- How to combine computations on GPU with those of CPU to deliver final results.

## Files needed

The following files will be needed for the exercise

- `DotProd_gold.c` (supposed to be correct and will not need changes)
- `DotProd.cu` (not correct and will need to be changed)
- `DotProd_kernel.cu` (not correct and will need to be changed)

## Work steps

You will need to modify the C code supplied in the files. You will implement a version of the stencil code for parallel execution on the GPU device.

You will need to modify the C code supplied in the files.

The host code in `DotProd.cu` needs to be updated through the steps

1. Allocate arrays for the vectors `Vec1_d`, `Vec2_d` and `gpuResult` in device memory.
2. Transfer arrays `Vec1_h` and `Vec2_h` from host to device.
3. Define the number of threads per block and from this a sufficient number of blocks per grid to be used in the invocation of the kernel `DotProd_kernel.cu`.
4. Transfer vector `gpuResult` from device to the vector `cpuResult` in host memory.
5. Free allocated device memory for vectors `Vec1_d`, `Vec2_d` and `gpuResult`.

The device code in `DotProd_kernel.cu` needs to be updated through the steps

6. Modify the kernel such that per thread it reads values from `Vec1_d` and `Vec2_d` from global device memory and stores the element-wise products in the vector `gpuResult` in device memory which has a size equal to at most the maximum number of threads that can be allocated on the device. Final reduction to scalar value is done on host (see supplied code in `DotProd.cu`).

Small performance study

7. Do a small study where you analyze the performance of the kernel compared to the CPU version. For example, split the gpu timing into data transfer time and computation time and compare the relative speedup with and without memory transfer. Examine how the grid configuration into blocks and threads influence the performance.

## Compilation

Compile the final code using a `Makefile`. In case of compilation errors you will need to debug the code until it compiles successfully.

## Execution

Execute your compiled program. If you program executes successfully the output should look like this

```
Computation of the dot product between two vectors of size N.
  ./DotProd <N:default=1000> <THREADS_PR_BLOCK:default=MaxOnDevice>

Device 0: Maximum number of threads per block is ?.
N: ?
Threads per block = ?.
Blocks allocated = ?

  CPU time    : ? (ms)
  GPU time    : ? (ms) , speedup ?x

PASSED!
```