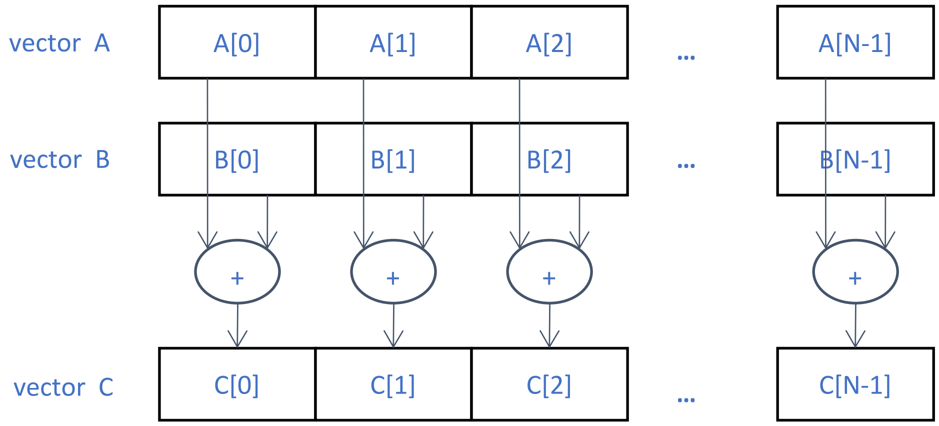
Programming Practice 1: Vector Addition

Objective: To understand basic programming model of GPU-accelerated heterogenous computing.

Vector addition:

- Given two arrays on *n* integers A, B.
- An empty array C of size *n*.
- Add elements of A and B in corresponding positions (i.e., compute C[i] = A[i] + B[i]).



(NVIDIA and UIUC, 2017)

Simple flow in GPU-accelerated heterogenous computing:

- Part 1: Copy input data from CPU memory to GPU memory.
- Part 2: Load GPU program and execute, caching data on GPU chip for full performance.
- Part 3: Copy results from GPU memory to CPU memory.

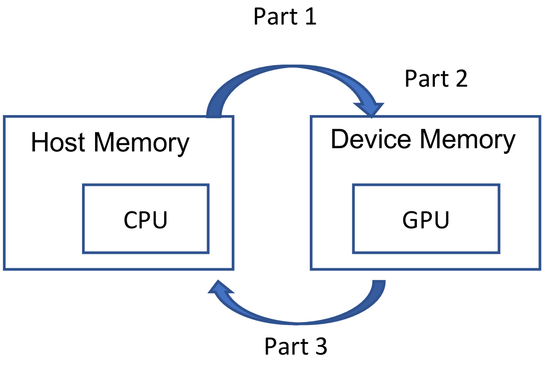
Host code template:

```
#include<cuda.h>
int main()
{
   // allocate and initialize host memory
   int n = 512; int *h_a = …, *h_b = …; *h_c = …(empty);

   // Part 1
   // allocate device memory for a, b, and c
   // copy a and b to device memory

   // Part 2
   // kernel launch code which let the device performs the actual vector addition

   // Part 3
   // copy c to host memory
   // free device memory
}
```



(NVIDIA and UIUC, 2017)

Basic CUDA Device Memory Management API functions:

- *cudaMalloc*(): allocates an object in the device global memory with the following two parameters in order:
    - Address of a pointer to the allocated object.
    - Size of allocated object in terms of bytes.

  Example:
  ```
  int size = n * sizeof(float); float *d_A;
  cudaMalloc((void **) &d_A, size);
  ```

- *cudaFree*(): frees object from device global memory with the following parameter.
    - pointer to freed object.

  Example:
  ```
  cudaFree(d_A);
  ```

- *cudaMemcpy*(): memory data transfer with the following four parameters in order:
    - pointer to destination
    - pointer to source
    - number of bytes copied
    - Type/direction of transfer

  Example:
  ```
  cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
  ```

Practice1.1: Implement host code for parallel addition of 512-element vectors.

Practice 1.2: Implement device code for parallel addition of 512-element vectors.

Practice 1.3: Combine your codes in 1.1 and 1.2 to compute parallel addition of 512-element vectors.

Practice 1.4: Adjust your codes in 1.3 to compute parallel addition of *n*-element vectors.