# HPC LAB 7

**PD_10_Kushagra Suryawanshi**

---

**Title: Write a CUDA program to find the sum/avg of elements in the N-element vector. (Where N is large number)**

**Aim:** Design and implement a CUDA program to find the sum/avg of elements element in the N-element vector. (Where N is large number)

**Objective:**

1. Write a CUDA program for finding sum/avg of elements in N element vector.
2. To understand and implement the CUDA-Device memory management functionalities
3. To understand and implement parallelism using CUDA.

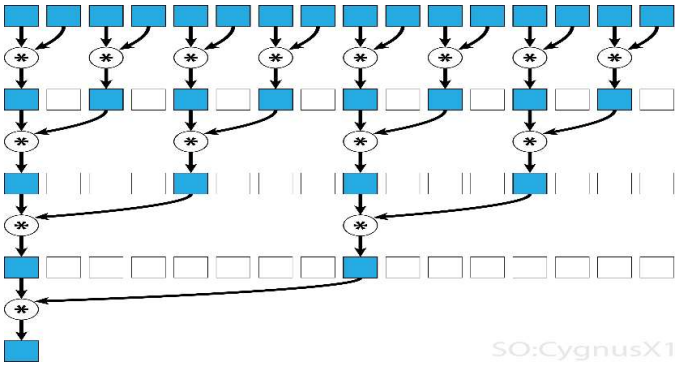
**Theory:**

**1) Explain the logic of reduction in addition/summation**

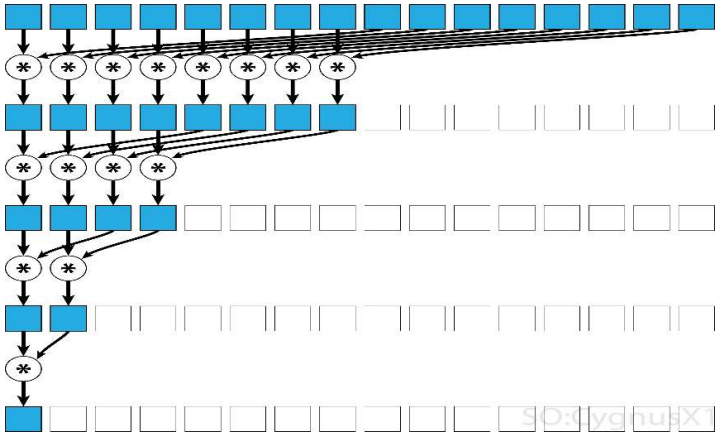
Parallel reduction algorithm typically refers to an algorithm which combines an array of elements, producing a single result. Typical problems that fall into this category are:

- summing up all elements in an array
- finding a maximum in an array

In general, the parallel reduction can be applied for any binary associative operator, i.e. (A*B)*C = A*(B*C) . With such operator *, the parallel reduction algorithm repeatedly groups the array arguments in pairs. Each pair is computed in parallel with others, halving the overall array size in one step. The process is repeated until only a single element exists.

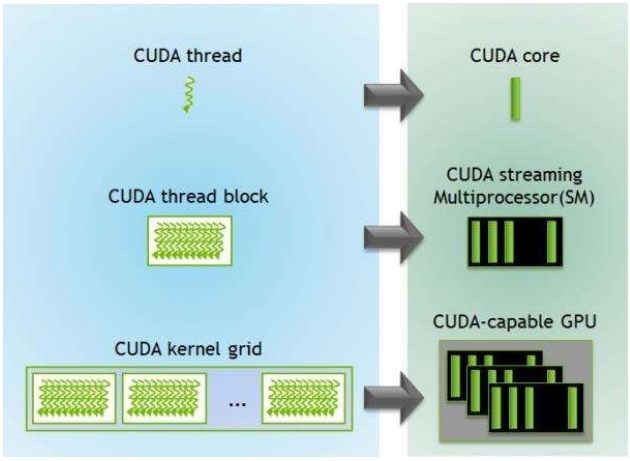


If the operator is commutative (i.e. A*B = B*A ) in addition to being associative, the algorithm can pair in a different pattern. From theoretical standpoint it makes no difference, but in practice it gives a better memory access pattern:



## 2) What role do threads play in addition of numbers using reduction

Every CUDA kernel starts with a __global_____declaration specifier. Programmers provide a unique global ID to each thread by using built-in variables. A group of threads is called a CUDA block. CUDA blocks are grouped into a grid. A kernel is executed as a grid of blocks of threads.




The idea is to use the multiple thread blocks in a GPU to reduce a small portion of the array. A tree based reduction is used inside each thread block and shared memory is used to achieve communication among threads of the same block and synchronization among them.

**CODE:**

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <iostream>
#include <numeric>
#include <stdio.h>
#include <stdlib.h> using
namespace std;

__global___void sum(int* input)
{
	const int tid = threadIdx.x; //similar to omp_get_thread_num()
	//we have spawned only one block see line no 46 <<< block, threads per block>>>
//blockIdx.x gives block id starting from 0
	int iteration=1;
	auto int step_size = 1;
	int number_of_threads = blockDim.x;

	while (number_of_threads > 0 )
	{
		printf("\n\titeration %d",iteration++);
		if (tid < number_of_threads) // still alive?
		{
			const auto int fst = tid * step_size * 2;
			const auto int snd = fst + step_size;
			input[fst] += input[snd];
			printf("\nComparing and adding elements at arr[%d]= %d and arr[%d]
= %d by thread id %d",fst,input[fst],snd,input[snd],tid);
		}

		step_size *= 2;
		number_of_threads /= 2;
```

```
        }
}

int main()
{
        const auto int count = 8;
        const int size = count * sizeof(int); int
        h[] = {1, 2, 3, 4, 5, 6, 7, 8};//10000

        int* d;

        cudaMalloc(&d, size);
        cudaMemcpy(d, h, size, cudaMemcpyHostToDevice);

        sum <<<1, count / 2 >>>(d);

        int result;
        cudaMemcpy(&result, d, sizeof(int), cudaMemcpyDeviceToHost);

        cout << "Sum is " << result << endl;

        cudaFree(d);

        return 0;
}
```

**Input:** N element vector

**Output:** Finding out the Maximum/minimum element in the N-element vector

**Platform:** Ubuntu (give latest version) or Windows [ installation of CUDA toolkit and presence of CUDA enabled platform ]

**Conclusion:** Thus, successfully studied, analysed and implemented CUDA program.

**FAQs:**

**1. Name the top 5 super computers of the world**

Ans:

**1. Fugaku, Japan**

Built by Fujitsu, Fugaku is installed at the RIKEN Center for Computational Science (R-CCS) in Kobe, Japan. With its additional hardware, the system achieved a new world record of 442 petaflops result on HPL, making it three times ahead of the number two system in the list.

**2. Summit, U.S.**

Based at the Oak Ridge National Laboratory (ORNL) in Tennessee, Summit was built by IBM and is the fastest system in the US.. Launched in 2018, it has a performance of 148.8 petaflops and has 4,356 nodes, each one housing two 22-core Power9 CPUs and six NVIDIA Tesla V100 GPUs.

**3. Sierra, U.S.**

A system at the Lawrence Livermore National Laboratory (LLNL) in California, Sierra has an HPL mark of 94.6 petaflops. With each of its 4,320 nodes equipped with two Power9 CPUs and four NVIDIA Tesla V100 GPUs, it has an architecture similar to that of Summit.

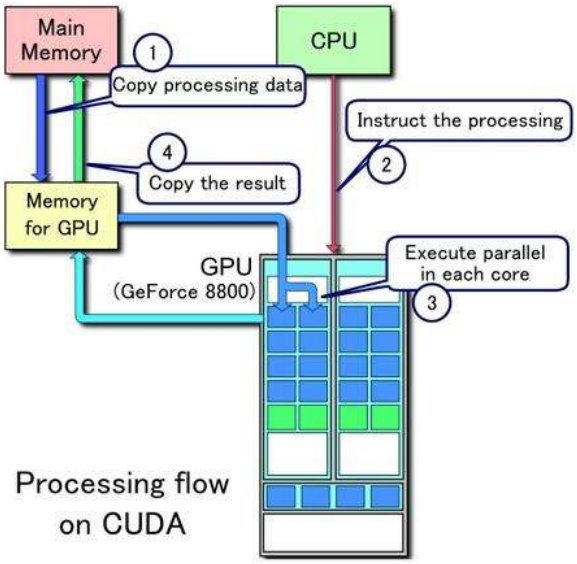
**4. Sunway TaihuLight, China**

Installed at China's National Supercomputing Center in Wuxi, Sunway TaihuLight previously held the Number 1 spot for two years .

**5. Selene, U.S.**

Installed in-house at NVIDIA Corp, Selene jumped to fifth position from seventh position in the June rankings. After a recent upgrade, Selene achieved 63.4 petaflops on HPL, nearly doubling its previous score of 27.6 petaflops.

**2. Name the components of GPU responsible for parallel processing**



A GPU has multiple **streaming multiprocessors (SM)** that contain

- memory registers for threads to use
- several memory caches
    - shared memory
    - constant cache
    - texture memory
    - L1 cache
- thread schedulers
- Several CUDA cores (analagous to streaming processsor in AMD cards) - number depends on microarchitecture generation
    - Each core consists of an Arithmetic logic unit (ALU) that handles integer and single precision calculations and a Floating point unit (FPU) that handles double precsion calculations
- Special function units (SFU) for transcendental functions (e.g. log, exp, sin, cos, sqrt)