Roll No: 41453

High Performance Computing: Assignment 1

Title:
Parallel Computing Using CUDA

Problem Statement:
a) Implement Parallel Reduction using Min, Max, Sum and Average operations.
b) Write a CUDA program that, given an N-element vector, find-
The maximum element in the vector
The minimum element in the vector
The arithmetic mean of the vector
The standard deviation of the values in the vector
Test for input N and generate a randomized vector V of length N (N should be large). The program should generate output as the two computed maximum values as well as the time taken to find each
value.

Learning Objectives:
- Learn parallel decomposition of problem.
- Learn parallel computing using CUDA.

Learning Outcome:
Be able to:
- Decompose problem into sub problems, to learn how to use GPUs, to learn to solve sub problem using threads on GPU cores.

Software and Hardware Requirements:
- 64-bit CPU
- 4 GB RAM
- Ubuntu 18 OS
- CUDA toolkit
- Nvidia GPU
- NVCC compiler

Steps to run:
- Install CUDA for your GPU
- Check NVCC version
- Use NVCC to compile .cu program (nvcc filename.cu -o filename)
- Run .cu program using generated output file (./filename)
- Check runtimes using command nvprof (nvprof ./filename)

Theory:

Dividing a computation into smaller computations and assigning them to different processors for parallel execution are the two key steps in the design of parallel algorithms. The process of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel, is called decomposition. Tasks are programmer-defined units of computation into which the main computation is subdivided by means of decomposition. Simultaneous execution of multiple tasks is the key to reducing the time required to solve the entire problem. Tasks can be of arbitrary size, but once defined, they are regarded as indivisible units of computation. The tasks into which a problem is decomposed may not all be of the same size.

CUDA Programming:
Terminology:

- Host The CPU and its memory (host memory)
- Device The GPU and its memory (device memory

Simple Processing Flow:

- Copy input data from CPU memory to GPU memory
- Load GPU program and execute, caching data on chip for performance
- Copy results from GPU memory to CPU memory

CUDA C/C++ keyword__global__indicates a function that:

- Runs on the device
- Is called from host code

Triple angle brackets mark a call from host code to device code Host and device memory are separate entities

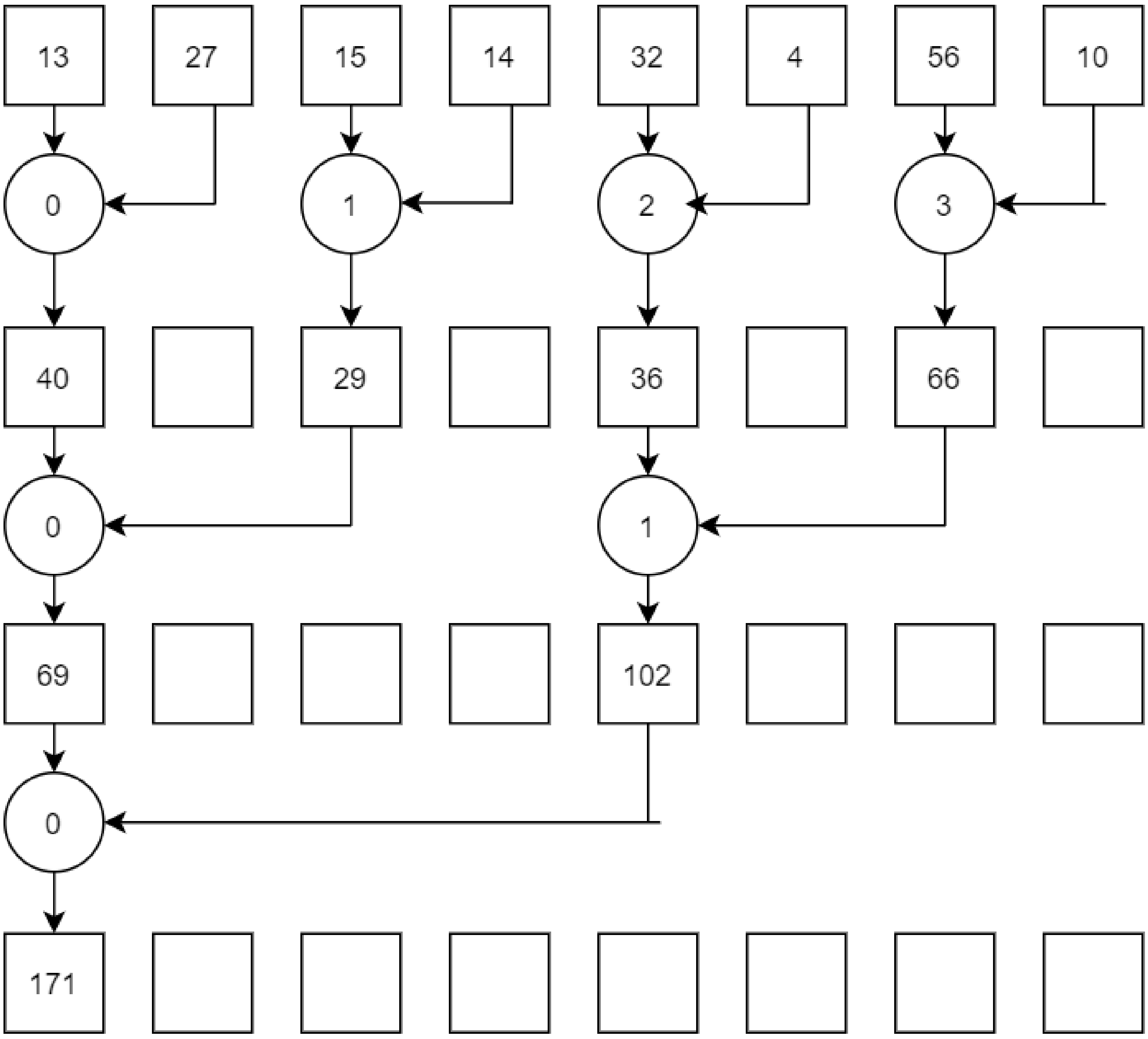
- Device pointers point to GPU memory
  - May be passed to/from host code
  - May not be dereferenced in host code
- Host pointers point to CPU memory
  - May be passed to/from device code
  - May not be dereferenced in device code

Simple CUDA API for handling device memory

- cudaMalloc(), cudaFree(), cudaMemcpy()
- Similar to the C equivalents malloc(), free(), memcpy()

```
__global__void add(int *a, int *b, int *c)
{
        c[blockIdx.x] = a[blockIdx.x] + b[blockIdx.x];
}
add<<< N, 1 >>>();
```

By using blockIdx.x to index into the array, each block handles a different element of the array.



Assuming N as the number of the elements in an array, we start N/2 threads, one thread for every two elements

- Each thread computes the sum of the corresponding two elements, storing the result at the position of the first one.
- Iteratively, each step:
  - the number of threads halved (for example, starting with 4, then 2, then 1)
  - doubles the step size between the corresponding two elements (starting with 1, then 2, then 4)
- after some iterations, the reduction result will be stored in the first element of the array.

Results:

For an array
62 20 72 59 79 88 6 92 1 58 51 96 23 83 2 21 44 71 8 10 26 73 1 65 36 36 14 95 90 37
63 53 57 36 64 88 76 70 81 30 80 84 26 4 68 80 25 64 51 85

```
CPU:
Vector Sum using CPU :2604
Vector Average using CPU :52.08
Vector Standard Deviation using CPU :29.15
Vector Min using CPU :1
Vector Max using CPU :96

GPU:
Vector Sum using GPU :2604
Vector Average using GPU :52.08
Vector Standard Deviation using GPU :29.14
Vector Min using GPU :1
Vector Max using GPU :96
```

Conclusion:

Successfully implemented Parallel Reduction for Min, Max, Sum and Average operations using CUDA on GPU.