four-a

April 9, 2025

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
 ↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
 ↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
 ↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
 ↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
 ↪outside of the current session
```

[2]:
```
!nvidia-smi
```

```
Wed Apr  9 13:41:42 2025
+-----------------------------------------------------------------------------
----------+
| NVIDIA-SMI 560.35.03              Driver Version: 560.35.03      CUDA Version:
12.6     |
|-----------------------------------------+------------------------+-----------
----------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
Compute M. |
|                                         |                        |
MIG M. |
```

```
|=========================================+========================+============
==========|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
0 |
| N/A   45C    P8             10W /   70W |       1MiB /  15360MiB |      0%
Default |
|                                         |                        |
N/A |
+-----------------------------------------+------------------------+------------
----------+
|   1  Tesla T4                       Off |   00000000:00:05.0 Off |
0 |
| N/A   34C    P8              9W /   70W |       1MiB /  15360MiB |      0%
Default |
|                                         |                        |
N/A |
+-----------------------------------------+------------------------+------------
----------+

+---------------------------------------------------------------------------------
----------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|=================================================================================
==========|
|  No running processes found
|
+---------------------------------------------------------------------------------
----------+
```

[3]:
```
%%writefile vector_add.cu
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include <math.h> // For fabs in verification

// Simple CUDA Error Handling Macro
#define CHECK_CUDA_ERROR(err) \
    if (err != cudaSuccess) { \
        fprintf(stderr, "CUDA Error at %s:%d: %s\n", __FILE__, __LINE__,␣
 ↪cudaGetErrorString(err)); \
        exit(EXIT_FAILURE); \
    }
```

```
// CUDA Kernel for Vector Addition
__global__ void vectorAddKernel(const float *a, const float *b, float *c, int␣
 ↪n) {
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index < n) {
        c[index] = a[index] + b[index];
    }
}

int main() {
    int n = 1 << 24; // ~16.7 million elements (a large vector)
    size_t size = n * sizeof(float);
    printf("Vector Addition (CUDA)\nVector size: %d elements (%.2f MB)\n", n,␣
 ↪(float)size / (1024*1024));

    // Host memory
    float *h_a = (float*)malloc(size);
    float *h_b = (float*)malloc(size);
    float *h_c = (float*)malloc(size);
    if (!h_a || !h_b || !h_c) {
        fprintf(stderr, "Failed to allocate host vectors!\n"); return␣
 ↪EXIT_FAILURE;
    }

    // Initialize host vectors
    for (int i = 0; i < n; ++i) {
        h_a[i] = (float)i;
        h_b[i] = (float)i * 2.0f;
    }

    // Device memory
    float *d_a = NULL, *d_b = NULL, *d_c = NULL;
    printf("Allocating %.2f MB on device...\n", 3.0f * size / (1024*1024));
    CHECK_CUDA_ERROR(cudaMalloc(&d_a, size));
    CHECK_CUDA_ERROR(cudaMalloc(&d_b, size));
    CHECK_CUDA_ERROR(cudaMalloc(&d_c, size));

    // Copy data Host -> Device
    printf("Copying data to device...\n");
    CHECK_CUDA_ERROR(cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice));
    CHECK_CUDA_ERROR(cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice));

    // Kernel launch configuration
    int blockSize = 256;
    int gridSize = (n + blockSize - 1) / blockSize;
```

```
    printf("Launching kernel (Grid: %d blocks, Block: %d threads)...\n",␣
↪gridSize, blockSize);

    // Launch kernel
    vectorAddKernel<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);
    CHECK_CUDA_ERROR(cudaPeekAtLastError()); // Check for launch errors
    CHECK_CUDA_ERROR(cudaDeviceSynchronize()); // Wait for kernel completion &␣
↪check run errors
    printf("Kernel finished.\n");

    // Copy data Device -> Host
    printf("Copying result back to host...\n");
    CHECK_CUDA_ERROR(cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost));

    // Verification (simple check)
    printf("Verifying result...\n");
    bool success = true;
    float tolerance = 1e-5f;
    if (fabs(h_c[0] - (h_a[0] + h_b[0])) > tolerance ||
        fabs(h_c[n-1] - (h_a[n-1] + h_b[n-1])) > tolerance) {
        success = false;
        printf("Mismatch detected: h_c[0]=%.f vs %.f, h_c[n-1]=%.f vs %.f\n",
               h_c[0], h_a[0] + h_b[0], h_c[n-1], h_a[n-1] + h_b[n-1]);
    }

    printf("Verification: %s\n", success ? "Successful!" : "FAILED!");

    // Cleanup
    printf("Freeing memory...\n");
    CHECK_CUDA_ERROR(cudaFree(d_a));
    CHECK_CUDA_ERROR(cudaFree(d_b));
    CHECK_CUDA_ERROR(cudaFree(d_c));
    free(h_a);
    free(h_b);
    free(h_c);

    printf("Vector addition complete.\n");
    return EXIT_SUCCESS;
}
```

```
Writing vector_add.cu
```

[4]:
```
!nvcc vector_add.cu -o vector_add
```

[5]:
```
!./vector_add
```

```
Vector Addition (CUDA)
```

```
Vector size: 16777216 elements (64.00 MB)
Allocating 192.00 MB on device…
Copying data to device…
Launching kernel (Grid: 65536 blocks, Block: 256 threads)…
Kernel finished.
Copying result back to host…
Verifying result…
Verification: Successful!
Freeing memory…
Vector addition complete.
```