multiplication

May 8, 2025

[1]:
```
!pip install git+https://github.com/afnan47/cuda.git
%load_ext nvcc_plugin
```

```
Collecting git+https://github.com/afnan47/cuda.git
  Cloning https://github.com/afnan47/cuda.git to /tmp/pip-req-build-5sm3djy9
  Running command git clone --filter=blob:none --quiet
https://github.com/afnan47/cuda.git /tmp/pip-req-build-5sm3djy9
  Resolved https://github.com/afnan47/cuda.git to commit
aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) … done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) … done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl
size=4290
sha256=341c4cd6e71d3a45791e7c64b0b40d5b065ec65271f2cfc3713c0dbdf82a2ca7
  Stored in directory: /tmp/pip-ephem-wheel-cache-1vnmi6ec/wheels/bc/4e/e0/2d86b
d15f671dbeb32144013f1159dba09757fde36dc51a963
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
created output directory at /content/src
Out bin /content/result.out
```

[7]:
```
%%writefile matrix_multiply.cu
#include <iostream>
using namespace std;
// CUDA kernel to multiply matrices
__global__ void multiply(int* A, int* B, int* C, int size) {
  int row = blockIdx.y * blockDim.y + threadIdx.y;
  int col = blockIdx.x * blockDim.x + threadIdx.x;
  if (row < size && col < size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
      sum += A[row * size + i] * B[i * size + col];
      }
      C[row * size + col] = sum;
  }
}
```

```
void initialize(int* matrix, int size) {
for (int i = 0; i < size * size; i++) {
matrix[i] = rand() % 10;
}
}

void print(int* matrix, int size) {
for (int row = 0; row < size; row++) {
for (int col = 0; col < size; col++) {
cout << matrix[row * size + col] << " ";
}
cout << '\n';
}
cout << '\n';
}
int main() {
int N = 2;
size_t matrixBytes = N * N * sizeof(int);
int* A = new int[N * N];
int* B = new int[N * N];
int* C = new int[N * N];
initialize(A, N);
initialize(B, N);
cout << "Matrix A:\n";
print(A, N);
cout << "Matrix B:\n";
print(B, N);
int *d_A, *d_B, *d_C;
cudaError_t err;
// Allocate memory on the device
err = cudaMalloc(&d_A, matrixBytes);
if (err != cudaSuccess) {
cout << "CUDA malloc failed for A: " << cudaGetErrorString(err) << endl;
return -1;
}

err = cudaMalloc(&d_B, matrixBytes);
if (err != cudaSuccess) {
cout << "CUDA malloc failed for B: " << cudaGetErrorString(err) << endl;
return -1;
}
err = cudaMalloc(&d_C, matrixBytes);
if (err != cudaSuccess) {
cout << "CUDA malloc failed for C: " << cudaGetErrorString(err) << endl;
return -1;
}
```

```
// Copy data from host to device
err = cudaMemcpy(d_A, A, matrixBytes, cudaMemcpyHostToDevice);
if (err != cudaSuccess) {
cout << "CUDA memcpy failed for A: " << cudaGetErrorString(err) << endl;
return -1;
}
err = cudaMemcpy(d_B, B, matrixBytes, cudaMemcpyHostToDevice);
if (err != cudaSuccess) {
cout << "CUDA memcpy failed for B: " << cudaGetErrorString(err) << endl;
return -1;
}
// Thread and block dimensions
dim3 threads(2, 2);
dim3 blocks((N + threads.x - 1) / threads.x, (N + threads.y - 1) / threads.y);
// Launch kernel
multiply<<<blocks, threads>>>(d_A, d_B, d_C, N);
// Synchronize to make sure the kernel finishes
cudaDeviceSynchronize();
// Check for kernel launch errors
err = cudaGetLastError();
if (err != cudaSuccess) {
cout << "CUDA kernel launch failed: " << cudaGetErrorString(err) << endl;
return -1;
}
// Copy result back to host
err = cudaMemcpy(C, d_C, matrixBytes, cudaMemcpyDeviceToHost);
if (err != cudaSuccess) {
cout << "CUDA memcpy failed for C: " << cudaGetErrorString(err) << endl;
return -1;
}
// Output the result
cout << "Multiplication of Matrix A and B:\n";
print(C, N);
// Clean up
delete[] A;
delete[] B;
delete[] C;
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
return 0;
}
```

```
Overwriting matrix_multiply.cu
```

[8]:
```
!nvcc -arch=sm_75 -o matrix_multiply matrix_multiply.cu
```

[9]:
```
!./matrix_multiply
```

```
Matrix A:
3 6
7 5

Matrix B:
3 5
6 2

Multiplication of Matrix A and B:
45 27
51 45
```

[ ]:
```

```