pc-practical-matrix-multiplication

April 9, 2024

[1]:
```
!python --version
!nvcc --version
!pip install nvcc4jupyter
%load_ext nvcc4jupyter
```

```
Python 3.10.12
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
Collecting nvcc4jupyter
  Downloading nvcc4jupyter-1.2.1-py3-none-any.whl (10 kB)
Installing collected packages: nvcc4jupyter
Successfully installed nvcc4jupyter-1.2.1
Detected platform "Colab". Running its setup…
Source files will be saved in "/tmp/tmp529a5t29".
```

[2]:
```
%%cuda

#include <cuda_runtime.h>
#include <iostream>

__global__ void matmul(int* A, int* B, int* C, int N) {
    int Row = blockIdx.y*blockDim.y+threadIdx.y;
    int Col = blockIdx.x*blockDim.x+threadIdx.x;
    if (Row < N && Col < N) {
        int Pvalue = 0;
        for (int k = 0; k < N; k++) {
            Pvalue += A[Row*N+k] * B[k*N+Col];
        }
        C[Row*N+Col] = Pvalue;
    }
}


int main() {
    std::cout<<"This is Atharva Pingale's code.\n";
```

```
    std::cout<<"Practical 4\n";
    int N = 16;
    int size = N * N * sizeof(int);
    int* A, * B, * C;
    int* dev_A, * dev_B, * dev_C;
    cudaMallocHost(&A, size);
    cudaMallocHost(&B, size);
    cudaMallocHost(&C, size);
    cudaMalloc(&dev_A, size);
    cudaMalloc(&dev_B, size);
    cudaMalloc(&dev_C, size);
    // C.NotebookApp.iopub_data_rate_limit = 10000000


    // Initialize matrices A and B
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i*N+j] = i*N*j+1;
            B[i*N+j] = j*N*i+1;
        }
    }


    std::cout<<"\nPrinting A: ";
    std::cout<<"\n";
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
          std::cout<<A[i*N+j]<<" ";
        }
        std::cout<<"\n";
    }
    std::cout<<"\n";

    std::cout<<"\nPrinting B : ";
    std::cout<<"\n";
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            std::cout<<B[i*N+j]<<" ";
        }
        std::cout<<"\n";

    }
    std::cout<<"\n";


    cudaMemcpy(dev_A, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_B, B, size, cudaMemcpyHostToDevice);
```

```
    dim3 dimBlock(16, 16);
    dim3 dimGrid(N/dimBlock.x, N/dimBlock.y);

    matmul<<<dimGrid, dimBlock>>>(dev_A, dev_B, dev_C, N);

    cudaMemcpy(C, dev_C, size, cudaMemcpyDeviceToHost);

    // Print the result
    std::cout<<"\nPrinting C ( Result ) : ";
    std::cout<<"\n";
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            std::cout << C[i*N+j] << " ";
        }
        std::cout << std::endl;
    }

    // Free memory
    cudaFree(dev_A);
    cudaFree(dev_B);
    cudaFree(dev_C);
    cudaFreeHost(A);
    cudaFreeHost(B);
    cudaFreeHost(C);

    return 0;
}
```

```
This is Atharva Pingale's code.
Practical 4

Printing A:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241
1 33 65 97 129 161 193 225 257 289 321 353 385 417 449 481
1 49 97 145 193 241 289 337 385 433 481 529 577 625 673 721
1 65 129 193 257 321 385 449 513 577 641 705 769 833 897 961
1 81 161 241 321 401 481 561 641 721 801 881 961 1041 1121 1201
1 97 193 289 385 481 577 673 769 865 961 1057 1153 1249 1345 1441
1 113 225 337 449 561 673 785 897 1009 1121 1233 1345 1457 1569 1681
1 129 257 385 513 641 769 897 1025 1153 1281 1409 1537 1665 1793 1921
1 145 289 433 577 721 865 1009 1153 1297 1441 1585 1729 1873 2017 2161
1 161 321 481 641 801 961 1121 1281 1441 1601 1761 1921 2081 2241 2401
1 177 353 529 705 881 1057 1233 1409 1585 1761 1937 2113 2289 2465 2641
1 193 385 577 769 961 1153 1345 1537 1729 1921 2113 2305 2497 2689 2881
1 209 417 625 833 1041 1249 1457 1665 1873 2081 2289 2497 2705 2913 3121
```

```
1 225 449 673 897 1121 1345 1569 1793 2017 2241 2465 2689 2913 3137 3361
1 241 481 721 961 1201 1441 1681 1921 2161 2401 2641 2881 3121 3361 3601


Printing B :
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241
1 33 65 97 129 161 193 225 257 289 321 353 385 417 449 481
1 49 97 145 193 241 289 337 385 433 481 529 577 625 673 721
1 65 129 193 257 321 385 449 513 577 641 705 769 833 897 961
1 81 161 241 321 401 481 561 641 721 801 881 961 1041 1121 1201
1 97 193 289 385 481 577 673 769 865 961 1057 1153 1249 1345 1441
1 113 225 337 449 561 673 785 897 1009 1121 1233 1345 1457 1569 1681
1 129 257 385 513 641 769 897 1025 1153 1281 1409 1537 1665 1793 1921
1 145 289 433 577 721 865 1009 1153 1297 1441 1585 1729 1873 2017 2161
1 161 321 481 641 801 961 1121 1281 1441 1601 1761 1921 2081 2241 2401
1 177 353 529 705 881 1057 1233 1409 1585 1761 1937 2113 2289 2465 2641
1 193 385 577 769 961 1153 1345 1537 1729 1921 2113 2305 2497 2689 2881
1 209 417 625 833 1041 1249 1457 1665 1873 2081 2289 2497 2705 2913 3121
1 225 449 673 897 1121 1345 1569 1793 2017 2241 2465 2689 2913 3137 3361
1 241 481 721 961 1201 1441 1681 1921 2161 2401 2641 2881 3121 3361 3601


Printing C ( Result ) :
16 1936 3856 5776 7696 9616 11536 13456 15376 17296 19216 21136 23056 24976
26896 28816
1936 321296 640656 960016 1279376 1598736 1918096 2237456 2556816 2876176
3195536 3514896 3834256 4153616 4472976 4792336
3856 640656 1277456 1914256 2551056 3187856 3824656 4461456 5098256 5735056
6371856 7008656 7645456 8282256 8919056 9555856
5776 960016 1914256 2868496 3822736 4776976 5731216 6685456 7639696 8593936
9548176 10502416 11456656 12410896 13365136 14319376
7696 1279376 2551056 3822736 5094416 6366096 7637776 8909456 10181136 11452816
12724496 13996176 15267856 16539536 17811216 19082896
9616 1598736 3187856 4776976 6366096 7955216 9544336 11133456 12722576 14311696
15900816 17489936 19079056 20668176 22257296 23846416
11536 1918096 3824656 5731216 7637776 9544336 11450896 13357456 15264016
17170576 19077136 20983696 22890256 24796816 26703376 28609936
13456 2237456 4461456 6685456 8909456 11133456 13357456 15581456 17805456
20029456 22253456 24477456 26701456 28925456 31149456 33373456
15376 2556816 5098256 7639696 10181136 12722576 15264016 17805456 20346896
22888336 25429776 27971216 30512656 33054096 35595536 38136976
17296 2876176 5735056 8593936 11452816 14311696 17170576 20029456 22888336
25747216 28606096 31464976 34323856 37182736 40041616 42900496
19216 3195536 6371856 9548176 12724496 15900816 19077136 22253456 25429776
28606096 31782416 34958736 38135056 41311376 44487696 47664016
21136 3514896 7008656 10502416 13996176 17489936 20983696 24477456 27971216
31464976 34958736 38452496 41946256 45440016 48933776 52427536
```

```
23056 3834256 7645456 11456656 15267856 19079056 22890256 26701456 30512656
34323856 38135056 41946256 45757456 49568656 53379856 57191056
24976 4153616 8282256 12410896 16539536 20668176 24796816 28925456 33054096
37182736 41311376 45440016 49568656 53697296 57825936 61954576
26896 4472976 8919056 13365136 17811216 22257296 26703376 31149456 35595536
40041616 44487696 48933776 53379856 57825936 62272016 66718096
28816 4792336 9555856 14319376 19082896 23846416 28609936 33373456 38136976
42900496 47664016 52427536 57191056 61954576 66718096 71481616
```

[2]: