**CSE467 - Parallel and Distributed Computing**

**Assignment 10**

**Hiba Mallick - 24015**

**Objective: Parallel Matrix (RxC) multiplication program using CUDA C/C++**

1. **By using shared variable for final result.**

%%cuda

#include <stdio.h>

#include <stdlib.h>

#include <cuda\_runtime.h>

\_\_global\_\_

void GPUmatmul(int N, double \*x, double \*y, double \*ans)

{

    int t = (blockDim.x \* blockDim.y) \* threadIdx.z

           + (threadIdx.y \* blockDim.x) + threadIdx.x;

    int b = (gridDim.x \* gridDim.y) \* blockIdx.z

           + (blockIdx.y \* gridDim.x) + blockIdx.x;

    int T = blockDim.x \* blockDim.y \* blockDim.z;

    int B = gridDim.x \* gridDim.y \* gridDim.z;

    for (int i = b; i < N; i += B){

        for(int j = t; j < N; j += T){

            for(int k = 0; k < N; k++){

                ans[i \* N + j] += (x[i \* N + k] \* y[k \* N + j]);

            }

        }

    }

}

int main() {

    int N = 4;

    double \*x, \*y, \*ans;

    x = (double \*)malloc(N \* N \* sizeof(double));

    y = (double \*)malloc(N \* N \* sizeof(double));

    ans = (double \*)calloc(N \* N, sizeof(double));

    //matrix x is all 2s

    for (int i = 0; i < N \* N; i++) {

        x[i] = 2.0;

    }

     //matrix y is all 3s

    for (int i = 0; i < N \* N; i++) {

        y[i] = 3.0;

    }

    double \*d\_x, \*d\_y, \*d\_ans;

    cudaMalloc((void \*\*)&d\_x, N \* N \* sizeof(double));

    cudaMalloc((void \*\*)&d\_y, N \* N \* sizeof(double));

    cudaMalloc((void \*\*)&d\_ans, N \* N \* sizeof(double));

    cudaMemcpy(d\_x, x, N \* N \* sizeof(double), cudaMemcpyHostToDevice);

    cudaMemcpy(d\_y, y, N \* N \* sizeof(double), cudaMemcpyHostToDevice);

    cudaEvent\_t start, stop;

    cudaEventCreate(&start);

    cudaEventCreate(&stop);

    dim3 blockSize(16, 16);

    dim3 gridSize(1, 1);

    cudaEventRecord(start);

    GPUmatmul<<<gridSize, blockSize>>>(N, d\_x, d\_y, d\_ans);

    cudaEventRecord(stop);

    cudaEventSynchronize(stop);

    float milliseconds = 0;

    cudaEventElapsedTime(&milliseconds, start, stop);

    printf("Time taken: %f milliseconds\n", milliseconds);

    cudaMemcpy(ans, d\_ans, N \* N \* sizeof(double), cudaMemcpyDeviceToHost);

    printf("Resultant Matrix (Shared Var):\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%f ", ans[i \* N + j]);

        }

        printf("\n");

    }

    cudaFree(d\_x);

    cudaFree(d\_y);

    cudaFree(d\_ans);

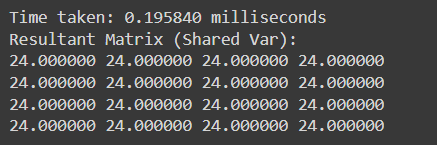
    free(x);

    free(y);

    free(ans);

    return 0;

}



1. **By using local variable for final result**

%%cuda

#include <stdio.h>

#include <stdlib.h>

#include <cuda\_runtime.h>

#define BLOCK\_SIZE 16

\_\_global\_\_

void GPUmatmul\_local(int N, double \*x, double \*y, double \*ans) {

    int tx = threadIdx.x;

    int ty = threadIdx.y;

    int bx = blockIdx.x;

    int by = blockIdx.y;

    int row = by \* blockDim.y + ty;

    int col = bx \* blockDim.x + tx;

    double temp = 0.0;

    for (int i = 0; i < N; i++) {

        temp += x[row \* N + i] \* y[i \* N + col];

    }

    ans[row \* N + col] = temp;

}

int main() {

    int N = 4;

    double \*x, \*y, \*ans;

    x = (double \*)malloc(N \* N \* sizeof(double));

    y = (double \*)malloc(N \* N \* sizeof(double));

    ans = (double \*)calloc(N \* N, sizeof(double));

    //matrix x has all 2s

    for (int i = 0; i < N \* N; i++) {

        x[i] = 2.0;

    }

    // matrix y with all 3s

    for (int i = 0; i < N \* N; i++) {

        y[i] = 3.0;

    }

    double \*d\_x, \*d\_y, \*d\_ans;

    cudaMalloc((void \*\*)&d\_x, N \* N \* sizeof(double));

    cudaMalloc((void \*\*)&d\_y, N \* N \* sizeof(double));

    cudaMalloc((void \*\*)&d\_ans, N \* N \* sizeof(double));

    cudaMemcpy(d\_x, x, N \* N \* sizeof(double), cudaMemcpyHostToDevice);

    cudaMemcpy(d\_y, y, N \* N \* sizeof(double), cudaMemcpyHostToDevice);

    dim3 blockSize(BLOCK\_SIZE, BLOCK\_SIZE); // Block size

    dim3 gridSize((N + BLOCK\_SIZE - 1) / BLOCK\_SIZE, (N + BLOCK\_SIZE - 1) / BLOCK\_SIZE);

    cudaEvent\_t start, stop;

    cudaEventCreate(&start);

    cudaEventCreate(&stop);

    cudaEventRecord(start);

    GPUmatmul\_local<<<gridSize, blockSize>>>(N, d\_x, d\_y, d\_ans);

    cudaDeviceSynchronize();

    cudaEventRecord(stop);

    cudaEventSynchronize(stop);

    float milliseconds = 0;

    cudaEventElapsedTime(&milliseconds, start, stop);

    printf("Time taken: %f milliseconds\n", milliseconds);

    cudaMemcpy(ans, d\_ans, N \* N \* sizeof(double), cudaMemcpyDeviceToHost);

    printf("Resultant Matrix (Local Variable):\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            printf("%f ", ans[i \* N + j]);

        }

        printf("\n");

    }

    cudaFree(d\_x);

    cudaFree(d\_y);

    cudaFree(d\_ans);

    free(x);

    free(y);

    free(ans);

    return 0;

}

A screen shot of a computer code

Description automatically generated