**Lập trình song song trên GPU**

**Hoàng Minh Thanh (18424062)**



**BT3 : Tính tổng tích luỹ**

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Bộ môn Công nghệ phần mềm

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

Contents

[1 Quá trình cài đặt Error! Bookmark not defined.](#_Toc44532818)

[a) Chương trình CUDA nhân hai ma trận : 3](#_Toc44532819)

[b) Chương trình đo thời gian chạy trên device ở với nhiều kích thước khác nhau : **Error! Bookmark not defined.**](#_Toc44532820)

[c) Chương trình đo thông tin Device trong CUDA **Error! Bookmark not defined.**](#_Toc44532821)

[2 Báo cáo và rút ra nhận xét, kết luận 10](#_Toc44532822)

[a) Đề cài đặt chương trình chạy trên GPU thì ta cần : **Error! Bookmark not defined.**](#_Toc44532823)

[b) Nhân xét và gridSize và blockSize : **Error! Bookmark not defined.**](#_Toc44532824)

[Rút ra so sánh và kết luận (**Đối với GPU trên Google Colab**) : **Error! Bookmark not defined.**](#_Toc44532825)

[c) Kết quả thông tin của một device của Google Colab Pro **Error! Bookmark not defined.**](#_Toc44532826)

# Cài đặt chương trình

Vì máy tính cả nhân của em không có GPU nên bắt buộc em phải sử dụng Google Colab :

<https://colab.research.google.com/drive/1-5Rznm3w2rAGpjHAzhTOqPP9ohnH-9S4#scrollTo=jQaNUnaMTQEJ>

(Thầy có thể vào link Online để xem luồng chạy dễ hơn báo cáo )

## a) Chương trình CUDA tính tổng tích lũy :

Cài đặt ở file **sumVector.cu**

Đây là kết quả tính tổng vector

%%cu

#include <stdio.h>

#include <stdlib.h>

#include <cuda\_runtime.h>

#include <sys/time.h>

using namespace std;

#define CHECK(call)                                                        \

{                                                                          \

    const cudaError\_t error = call;                                        \

    if (error != cudaSuccess)                                              \

    {                                                                      \

        printf("Error: %s:%d, ", \_\_FILE\_\_, \_\_LINE\_\_);                      \

        printf("code:%d, reason: %s\n", error, cudaGetErrorString(error)); \

        exit(1);                                                           \

    }                                                                      \

}

double seconds(){

    struct timeval tp;

    gettimeofday(&tp, NULL);

    return ((double)tp.tv\_sec + (double)tp.tv\_usec \* 1.e-6);

}

void initialData(float \*data, int size)

{

    srand(0);

    for (int i = 0; i < size; i++)

    {

        data[i] = (float)(rand()) / RAND\_MAX;

    }

}

void printFirst10(float \*data, int size){

    for (int i = 0; i < size && i < 10; i++){

        printf("%f\n", data[i]);

    }

}

float recursiveReduce(float \*data, int const size)

{

    if (size == 1)

        return data[0];

    if (size % 2 == 1) {

        data[0] += data[size];

    }

    int const stride = size / 2;

    for (int i = 0; i < stride; i++){

        data[i] += data[i + stride];

    }

    return recursiveReduce(data, stride);

}

\_\_global\_\_ void reduceNeighbored(float \*g\_idata, float \*g\_odata, unsigned int n)

{

    // set thread ID

    unsigned int tid = threadIdx.x;

    unsigned int idx = threadIdx.x + blockIdx.x \* blockDim.x;

    // convert global data pointer to the local pointer of this block

    float \*idata = g\_idata + blockIdx.x \* blockDim.x;

    // boundary check

    if (idx >= n) return;

    // in-place reduction in gloabl memory

    for (int stride = 1; stride < blockDim.x; stride \*= 2){

        if ((tid % (2 \* stride)) == 0){

            idata[tid] += idata[tid + stride];

        }

        // synchronize within threadblock

        \_\_syncthreads();

    }

    // write result for this block to global mem

    if (tid == 0)

        g\_odata[blockIdx.x] = idata[0];

}

\_\_global\_\_ void reduceNeighboredLess(float \*g\_idata, float \*g\_odata, unsigned int n){

    // set thread ID

    unsigned int tid = threadIdx.x;

    unsigned int idx = blockIdx.x \* blockDim.x + threadIdx.x;

    // convert global data pointer to the local pointer of this block

    float \*idata = g\_idata + blockIdx.x \* blockDim.x;

    // boundary check

    if (idx >= n) return;

    // in-place reduction in gloabl memory

    for (int stride = 1; stride < blockDim.x; stride \*= 2){

        // convert tid into local array index

        int index = 2 \* stride \* tid;

        if (index < blockDim.x){

            idata[index] += idata[index + stride];

        }

        // synchronize within threadblock

        \_\_syncthreads();

    }

    // write result for this block to global mem

    if (tid == 0)

        g\_odata[blockIdx.x] = idata[0];

}

int main()

{

    // set up device

    int dev = 0;

    cudaSetDevice(dev);

    int const BLOCK\_SIZE = 32;

    // set up vector

    int size = pow(2, 2);

    float \*data, host\_total, device\_total = 0; // vector

    int nBytes = size \* sizeof(float);

    data = (float \*)malloc(nBytes);

    // initialize data at host side

    initialData(data, size);

    printFirst10(data, size);

    // malloc device global memory

    float \*g\_idata, \*g\_odata;

    CHECK(cudaMalloc((float \*\*)&g\_idata, nBytes));

    CHECK(cudaMalloc((float \*\*)&g\_odata, nBytes));

    CHECK(cudaMemcpy(g\_idata, data, nBytes, cudaMemcpyHostToDevice));

    double host\_start = seconds();

    host\_total = recursiveReduce(data, size);

    double host\_elaps = seconds() - host\_start;

    printf("Total host : %f, Time host : %f sec\n", host\_total, host\_elaps);

    // Launch add() kernel on GPU

    dim3 blockSize(BLOCK\_SIZE, 1, 1);

    unsigned int gridWith = (size + blockSize.x - 1) / blockSize.x;

    printf("gridWith %d\n", gridWith);

    dim3 gridSize(gridWith, 1, 1);

    // output each block in gpu

    float \*odata;

    odata = (float \*)malloc(gridWith \* sizeof(float));

    double device\_start = seconds();

    reduceNeighboredLess<<<gridSize, blockSize>>>(g\_idata, g\_odata, size);

    CHECK(cudaDeviceSynchronize());

    // Copy result back to host

    CHECK(cudaMemcpy(odata, g\_odata, gridWith \* sizeof(float), cudaMemcpyDeviceToHost));

    printFirst10(odata, gridWith);

    for (int i = 0; i < gridWith; i++){

        device\_total += odata[i];

    }

    double device\_elaps = seconds() - device\_start;

    printf("Total device %f, Time device : %f sec\n", device\_total, device\_elaps);

    // Cleanup

    cudaFree(g\_idata);

    cudaFree(g\_odata);

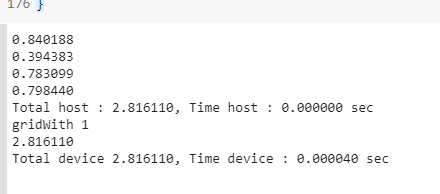
    free(data);

    free(odata);

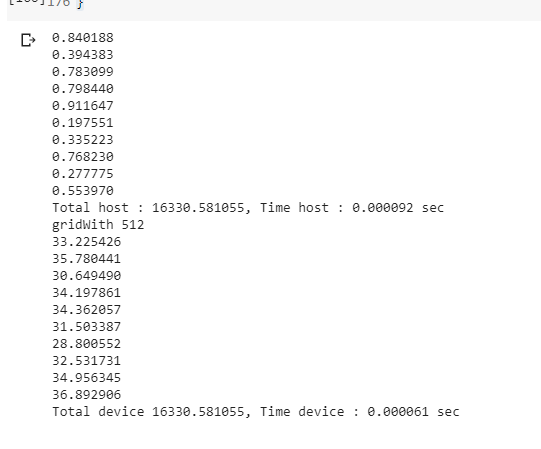
    return 0;

}

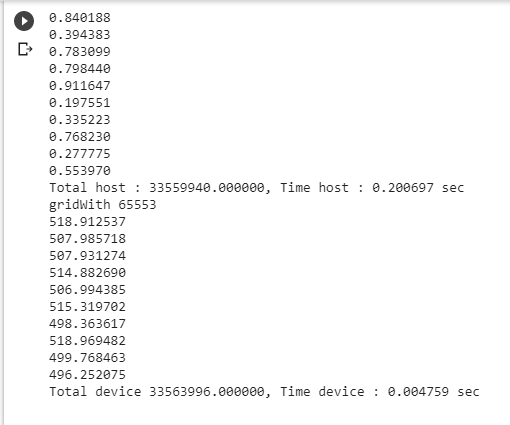
Kết quả :



Với BLOCK SIZE = 64



Với BLOCK SIZE = 1024



# Báo cáo và rút ra nhận xét, kết luận

\* Vì xử lý trên số thực nên khi số lượng tính toán càng lớn có thể sẽ dấn đến sai số

\* Có thể thấy với khối lượng tính toán càng lớn thì GPU xử lý nhanh hơn rất nhiều so với host (CPU)

\* Khi tăng số block size, và thay đổi tương ứng với grid size thì tốc độ tính toán của GPU tăng nhanh hơn