بسمه تعالی

## عنوان درس: مباحث پیشرفته در زبانهای برنامه نویسی موازی

**استاد: آقای دکتر سوادی**

**پروژه : پیاده سازی و موازی سازی الگوریتم Gradient descend linear regression بوسیله nvidia/cuda**

**اعضاء گروه : محمد دانش آموز – مهسا زاهدی**

توضیحات فاز موازی سازی:

در موازی سازی این الگوریتم با چالشهای مختلفی روبرو شدیم که مهمترین آنها بحث لزوم انجام عملیات تاحدودی سنگین برای هر هسته کودا بود، برنامه را مکررا و با حالات مختلف اجرا می کردیم ولی زمان اجرا در بهترین حالت مشابه زمان اجرا برروی ۲ هسته cpu بوسیله openmp میشد تا اینکه بعد از تحقیق و بررسی متوجه این مسئله شدیم و بار پردازشی هر ترد پردازنده گرافیک را بیشتر کردیم که نتیجه اجرای برنامه بصورت نمودارهایی در تصویر صفحه بعد قابل ملاحظه است. برای بهینه سازی اجرا، برنامه را در حالتهای مختلف با سایز بلاک های مختلف اجرا نمودیم که بهترین حالت سایز ۱۶ و۳۲ برای هر بلاک بود که در نمودار هر قابل مشاهده است.

لینک پروژه :

https://github.com/mohammaddan/gradient-descend-cuda.git

CPU : Intel 7700 4 core – 8 hyper thread - freq: 4Ghz – cache size: 16MB – cache line size : 64Byte – RAM 16GB DDR4
OS : Ubuntu 20.04.2

| Execute with CPU cores (openmp) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| thread count | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| execute time | 99.104 | 51.911 | 38.34 | 33.262 | 45.025 | 40.99 | 35.106 | 36.875 |

| Execute with GPU cores (Cuda) | | | | | | |
|---|---|---|---|---|---|---|
| Block size | 1 | 2 | 4 | 8 | 16 | 32 |
| execute time | 29.283 | 22.323 | 21.023 | 16.808 | 16.228 | 16.756 |

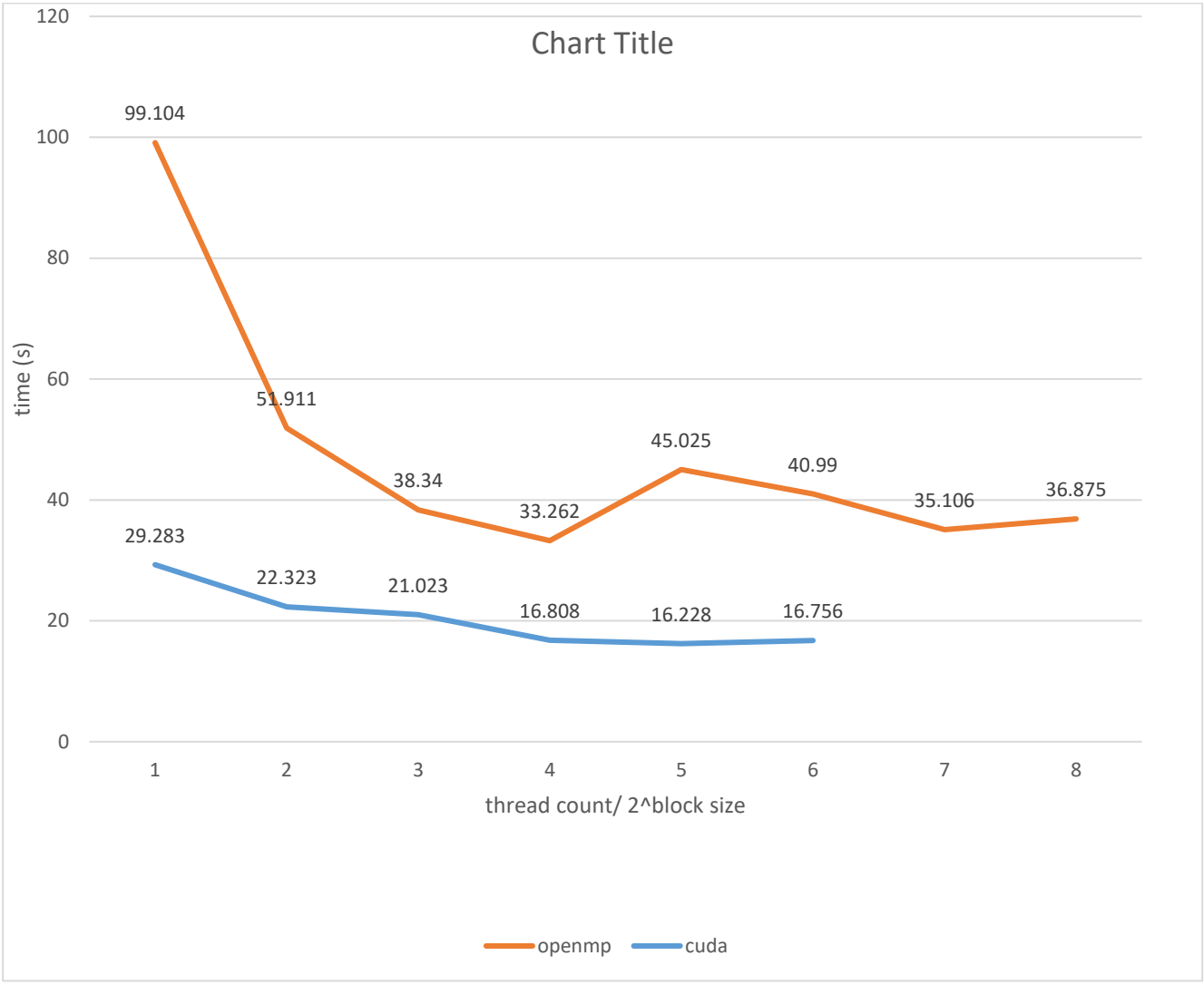
Chart Title
120
100
80
60
40
20
0
time (s)
99.104
51.911
38.34
33.262
45.025
40.99
35.106
36.875
29.283
22.323
21.023
16.808
16.228
16.756
1
2
3
4
5
6
7
8
thread count/ 2^block size
openmp
cuda

```cpp
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "kernel.h"


__global__ void mulKernel(double* c, double* a, double* b,int n,int p) {
    int row = threadIdx.x+blockDim.x*blockIdx.x;
    if (row >= n) return;
    double temp = 0;
    for (int i = 0; i < p; i++) {
        temp += a[row * p + i] * b[i];
    }
    c[row] = temp;
}
__global__ void mulTransposeKernel(double* c, double* a, double* b,int n,int p,double alpha=1.0) {
    int col = threadIdx.x + blockDim.x * blockIdx.x;
    if (col >= n) return;
    double temp = 0;
    for (int i = 0; i < n; i++) {
        temp += a[i * p + col] * b[i];
    }
    c[col] += temp*alpha;
}


__global__ void minusKernel(double* c, double* a, double* b,int n) {
    int index = threadIdx.x + blockDim.x * blockIdx.x;
    if (index >= n) return;
    c[index] = a[index] - b[index];
}

__global__ void arrayXarray(double* c, double* a, double* b,int n) {
    int index = threadIdx.x + blockDim.x * blockIdx.x;
    if (index >= n) return;
    c[index] = a[index] * b[index];
}

void tink4(double* theta,double* train_x,double* train_y,int n,int p,int blocksize) {
    double alpha = 0.0083;
    double *err_0,*y_pred,*dev_train_x,*dev_train_y,*dev_y_pred, *dev_theta,*dev_err,*dev_err2;
    for (int i = 0; i < p; i++)
        theta[i] = (rand() % 10) / 1000.0;

    cudaMalloc((void**)& dev_theta,p * sizeof(double));
    cudaMalloc((void**)& dev_train_x, n*p * sizeof(double));
    cudaMalloc((void**)& dev_train_y, n* sizeof(double));
    cudaMalloc((void**)& dev_y_pred, n* sizeof(double));
    cudaMalloc((void**)& dev_err, n* sizeof(double));
    cudaMalloc((void**)& dev_err2, n* sizeof(double));

    cudaMemcpy(dev_train_x, train_x, p * n * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_train_y, train_y, n * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_theta, theta, p * sizeof(double), cudaMemcpyHostToDevice);

    err_0 = (double*)malloc(n * sizeof(double));

    int cnt = 2000000, it = 0;
    double last_error, error = 100;
    do {
        last_error = error;
```

```
        mulKernel <<< ceil(n/blocksize),blocksize >> > (dev_y_pred, dev_train_x, dev_theta, n,p);

        cudaDeviceSynchronize();
        minusKernel << <ceil(n/blocksize),blocksize>> > (dev_err, dev_y_pred, dev_train_y,n);
        cudaDeviceSynchronize();
        arrayXarray << <ceil(n/blocksize),blocksize>> > (dev_err2, dev_err, dev_err,n);

        cudaDeviceSynchronize();
        mulTransposeKernel << <1,p >> > (dev_theta, dev_train_x, dev_err,n,p,-alpha/n);

        cudaDeviceSynchronize();
        cudaMemcpy(err_0, dev_err2, n * sizeof(double), cudaMemcpyDeviceToHost);
        for (int i = 0; i < n; i++) error2 += err_0[i];

        error = sqrt(error2);
        if (error > last_error)
            alpha /= 1.2;
        cudaDeviceSynchronize();

    } while (it++ < cnt && fabs(error - last_error) > epsilon);
    printf("in %d iterations \n", it);
    cudaMemcpy(theta, dev_theta, p * sizeof(double), cudaMemcpyDeviceToHost);
    cudaFree(dev_err);
    cudaFree(dev_train_x);
    cudaFree(dev_train_y);
    cudaFree(dev_y_pred);
    cudaFree(dev_theta);
}
```