1.程式的內容

可以分成三個部份

.程式自行生成 data.txt(!!!!!我有先把兩隻程式自行產生的隨機data.txt先註解掉，讓助教測試時會用到同一筆資料-->我繳交的data.txt)

//自行產生隨機變數-->data.txt

FILE \*fp = fopen("data.txt","w");

srand(time(NULL));

double j;

for(int i=0;i<400;i++){

for(int k=0;k<400;k++){

j = rand() % (N+1) / (double) (N+1);

fprintf(fp,"%.5lf ",j);

}

fprintf(fp,"\n");

}

fclose(fp);

.計算過程

(1)Non-SIMD version

-想法：

用三層的for迴圈一個一個慢慢計算，相乘過後再相加（超偷懶的方法...）

for(int i=0;i<200;i++){

for(int j=0;j<200;j++){

for(int k=0;k<198;k++){

ans[i]=ans[i]+matrix\_A[i][k]\*matrix\_B[j][k]; //一個一個相加

}

}

}

(2)SIMD version

-想法：

先把B\_matrix的每一行用SIMD的方式相加起，接著一樣用SIMD方式把它跟A\_matrix相乘最後在相加

//先把B裡每一行加起來

for(int i=0;i<50;i++){

for(int j=0;j<200;j++){

b = (\_\_m128\*) B[j];

c = (\_\_m128\*) C;

c=c+i;

b=b+i;

\*c = \_mm\_add\_ps(\*b,\*c);

}

}

//把剛剛算好的B所有(其實就是C)乘上A

for(int i=0;i<50;i++){

for(int j=0;j<200;j++){

a = (\_\_m128\*) A[j];

c = (\_\_m128\*) C;

d = (\_\_m128\*) D[j];

c=c+i;

a=a+i;

d=d+i;

\*d = \_mm\_mul\_ps(\*a,\*c);

}

}

//把每一列的D相加

float ans[200];

for(int i=0;i<200;i++){

ans[i]=0;

for(int j=0;j<198;j++){

ans[i]=ans[i]+D[i][j];

}

}

.把結果放到output.txt

FILE \*fp2 = fopen("output.txt","w");

for(int i=0;i<200;i++){

fprintf(fp2,"%.5lf\n",ans[i]);

}

fclose(fp2);

補充：

-clock\_gettime的用法：

#include <time.h> -->函式

struct timespec tt1, tt2; -->兩個時間點，要計算經過時間就把tt2-tt1

clock\_gettime(CLOCK\_REALTIME, &tt1); -->開始時間

clock\_gettime(CLOCK\_REALTIME, &tt2); -->結束時間

減出來的數字時間單位是[奈秒]

-讀檔時Non-SIMD version和SIMD version的不同：

因為SIMD有用對齊

例如：float A[200][200]\_\_attribute\_\_ ((aligned(16))) --->align \*記的後面的16是指分配16個[byte]

所以我用200格，因為如果沒用200只用198會造成我執行時它會跳出（核心傾倒？），所以我開200，但是！！！讀檔時他是一個一個讀，如果讓它199 200都讀的話會造成輸入文字根本不同所以要記得跳過

-兩支程式測出來的值小數點後面幾位會因為浮點數關係而有一點差異，但是整數部份和小數點前幾位幾乎沒有太大差別

2.執行環境

CPU型號:(--> 指令：cat /proc/cpuinfo)

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant\_tsc arch\_perfmon nopl xtopology tsc\_reliable nonstop\_tsc cpuid pni pclmulqdq ssse3 fma cx16 pcid sse4\_1 sse4\_2 x2apic movbe popcnt tsc\_deadline\_timer aes xsave avx f16c rdrand hypervisor lahf\_lm abm 3dnowprefetch cpuid\_fault invpcid\_single ssbd ibrs ibpb stibp ibrs\_enhanced fsgsbase tsc\_adjust bmi1 avx2 smep bmi2 invpcid rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xsaves arat md\_clear flush\_l1d arch\_capabilities

記憶體：(--> 指令：free -m)

total used free shared buff/cache available

Mem: 3896 1668 386 5 1841 1975

作業系統：Ubuntu linux

3.Non-SIMD version vs SIMD version

read data\_time computation\_time write\_data\_time

Non-SIMD version: 49022033 64424212 687700

SIMD version: 25000274 449071 601837

因為我只有在computation那邊用SIMD，所以可以很明顯看出使用SIMD的話時間大幅降低...(read和write沒有時間想如何用SIMD了QQ)

4.編譯程式下的參數

Non-SIMD version: gcc -msse4 -o hw7simd hw7simd.c

SIMD version: gcc -o hw7 hw7.c

5.使用了哪些指令: SSE

6.如何執行你的程式

all:

gcc -msse4 -o hw7\_simd hw7simd.c

gcc -o hw7 hw7.c

clean:

rm hw7 hw7\_simd

7.心得：

SIMD太強大了！！時間可以少很多