#include<iostream>

using namespace std;

#include <omp.h>

#include<windows.h>

#include<pmmintrin.h> //SSE3

#include <immintrin.h>//AVX

int NUM\_THREADS = 7;

void display(float \*\*A, int n){

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

for(int j=0; j<n; j++)

{cout<<A[i][j]<<' ';}

cout<<endl;}

}

int main(){

//矩阵初始化

for(int n=4;n<=1024;n=n\*2)

{

cout<<n<<' ';

float\*\* A = new float\* [n];

for(int i=0;i<n;i++)

A[i]= new float[n];

for(int i=0;i<n;i++)

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

A[i][j]=0;

A[i][i]=1.0;

for(int j=i+1;j<n;j++)

A[i][j]=rand();

}

for(int k=0;k<n;k++)

for(int i=k+1;i<n;i++)

for(int j=0;j<n;j++)

A[i][j]+=A[k][j];

//计时

long long head, tail , freq ;

QueryPerformanceFrequency((LARGE\_INTEGER \*)&freq );

QueryPerformanceCounter((LARGE\_INTEGER \*)&head);

for(int l=1;l<=10;l++){

// 在外循环之外创建线程，避免线程反复创建销毁，注意共享变量和私有变量的设置

int i,j,k;

float tmp;

#pragma omp parallel num\_threads(NUM\_THREADS) private(i, j, k, tmp)

for(k = 1; k < n; ++k){

\_\_m256 t1, t2, t3, t4;

// 除法部分 采取串行

#pragma omp single

{

/\*tmp = A[k][k];

for(int j = k + 1; j < n; ++j){

A[k][j] = A[k][j] / tmp;

}

A[k][k] = 1.0;\*/

float tmp[8] = {A[k][k], A[k][k], A[k][k], A[k][k], A[k][k], A[k][k], A[k][k], A[k][k]};

t1 = \_mm256\_loadu\_ps(tmp);

int j = n-8;

for(j=n-8;j>=k;j = j-8)

{

t2 = \_mm256\_loadu\_ps(A[k]+j);

t3 = \_mm256\_div\_ps(t2, t1);

\_mm256\_storeu\_ps(A[k]+j, t3);

}

if((j+8)!=k){

for(int i=k; (i<n)&&(i<(j+8)); i++){

A[k][i] = A[k][i]/A[k][k];

}

}

}

// 并行部分，使用行划分

#pragma omp for

for(int i=k+1;i<n;i++){

float tmp[8] = {A[i][k], A[i][k], A[i][k], A[i][k],A[i][k], A[i][k], A[i][k], A[i][k]};

t1 = \_mm256\_loadu\_ps(tmp);

int j = n-8;

for(j=n-8;j>=k;j=j-8){

//A[i][j] = A[i][j]-A[i][k]\*A[k][j];

t2 = \_mm256\_loadu\_ps(A[i] + j);

t3 = \_mm256\_loadu\_ps(A[k] + j);

t4 = \_mm256\_sub\_ps(t2, \_mm256\_mul\_ps(t1, t3));

\_mm256\_storeu\_ps(A[i]+j, t4);}

if((j+8)!=k)

{

for(int s=k; s<(j+8);s++)

A[i][s] = A[i][s]-A[i][k]\*A[k][s];

}

}

}

/\*for(i = k + 1; i < n; ++i)

{

tmp = A[i][k];

for(j = k + 1; j < n; ++j)

{

A[i][j] = A[i][j] - tmp\*A[k][j];

}

A[i][k] = 0.0;

}\*/

// 离开for循环时，各个线程默认同步，进入下一行的处理

}

QueryPerformanceCounter((LARGE\_INTEGER \*)&tail );

cout<<(tail - head)\*1000.0/freq<<endl;//时间单位：毫秒

}

return 0;

}