#include<iostream>

using namespace std;

#include<pthread.h>

#include <sys/time.h>

#include<time.h>

#include <immintrin.h>//AVX

#include <xmmintrin.h>

//动态线程版本

//定义线程数据结构

typedef struct{

int k; //消去的轮次

int t\_id; //线程id

int act\_row;

int n;

float\*\* A;

}threadParam\_t;

//定义线程函数

void \*threadFunc(void \*param){

threadParam\_t \*p = (threadParam\_t\*)param;

int n = p->n;

float\*\* A = p->A;

int k = p->k;

int t\_id = p->t\_id;

int i = p->act\_row;

//int i = k+t\_id+1; 获取自己的计算任务

//AVX-消去部分并行化

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

/\*

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

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

}

A[i][k] = 0;\*/

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;

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(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;

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];

}

}

\*/

pthread\_exit(NULL);

}

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

int n ;

for(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];

//动态线程版本

struct timeval start;

struct timeval end;

gettimeofday(&start,NULL);

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

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

//主线程做除法操作

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

A[k][k]=1.0;

//创建工作线程，进行消去操作

int worker\_count = n-1-k;

if(worker\_count>7){worker\_count = 7;} //工作的线程数不超过8个

int cowork\_cycle;

if(worker\_count>0){

cowork\_cycle = (n-1-k)/worker\_count;

}

int remain = n-1-k - cowork\_cycle\*worker\_count;

for(int c=0;c<cowork\_cycle;c++){

pthread\_t\* handles = new pthread\_t[worker\_count];//创建对应的handle参数

threadParam\_t \*param = new threadParam\_t[worker\_count]; //创建对应的线程数据结构

//分配任务

for(int t\_id = 0;t\_id<worker\_count; t\_id++){

param[t\_id].k = k;

param[t\_id].t\_id = t\_id;

param[t\_id].A = A;

param[t\_id].n = n;

param[t\_id].act\_row = k + c\*worker\_count + t\_id + 1;

}

//创建线程(核心函数)

for(int t\_id=0;t\_id<worker\_count;t\_id++)

pthread\_create(&handles[t\_id],NULL,threadFunc,&param[t\_id]);

for(int t\_id = 0; t\_id<worker\_count;t\_id++)

//主线程挂起等待所有工作线程完成

pthread\_join(handles[t\_id], NULL);

}

//处理剩余的remain行

pthread\_t\* handles = new pthread\_t[remain];//创建对应的handle参数

threadParam\_t \*param = new threadParam\_t[remain]; //创建对应的线程数据结构

for(int t\_id = 0;t\_id<remain; t\_id++){

param[t\_id].k = k ;

param[t\_id].t\_id = t\_id;

param[t\_id].A = A;

param[t\_id].n = n;

param[t\_id].act\_row = cowork\_cycle\*worker\_count + k + 1 + t\_id;

}

//创建线程(核心函数)

for(int t\_id=0;t\_id<remain;t\_id++)

pthread\_create(&handles[t\_id],NULL,threadFunc,&param[t\_id]);

for(int t\_id = 0; t\_id<remain;t\_id++)

//主线程挂起等待所有工作线程完成

pthread\_join(handles[t\_id], NULL);

}

gettimeofday(&end,NULL);

cout<<((long long)end.tv\_sec-(long long)start.tv\_sec)\*1000000+((long long)end.tv\_usec-(long long)start.tv\_usec)<<endl;//微秒

}

return 0;}

#include<iostream>

using namespace std;

#include<pthread.h>

#include <sys/time.h>

#include<time.h>

#include <immintrin.h>//AVX

#include <xmmintrin.h>

//动态线程版本

//定义线程数据结构

typedef struct{

int k; //消去的轮次

int t\_id; //线程id

int act\_row;

int n;

float\*\* A;

}threadParam\_t;

//定义线程函数

void \*threadFunc(void \*param){

threadParam\_t \*p = (threadParam\_t\*)param;

int n = p->n;

float\*\* A = p->A;

int k = p->k;

int t\_id = p->t\_id;

int i = p->act\_row;

//int i = k+t\_id+1; 获取自己的计算任务

//AVX-消去部分并行化

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

/\*

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

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

}

A[i][k] = 0;\*/

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;

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(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;

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];

}

}

\*/

pthread\_exit(NULL);

}

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

int n =4;

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];

//动态线程版本

struct timeval start;

struct timeval end;

gettimeofday(&start,NULL);

for(int i=0;i<100;i++){

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

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

//主线程做除法操作

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

A[k][k]=1.0;

//创建工作线程，进行消去操作

int worker\_count = n-1-k;

if(worker\_count>7){worker\_count = 7;} //工作的线程数不超过8个

int cowork\_cycle;

if(worker\_count>0){

cowork\_cycle = (n-1-k)/worker\_count;

}

int remain = n-1-k - cowork\_cycle\*worker\_count;

for(int c=0;c<cowork\_cycle;c++){

pthread\_t\* handles = new pthread\_t[worker\_count];//创建对应的handle参数

threadParam\_t \*param = new threadParam\_t[worker\_count]; //创建对应的线程数据结构

//分配任务

for(int t\_id = 0;t\_id<worker\_count; t\_id++){

param[t\_id].k = k;

param[t\_id].t\_id = t\_id;

param[t\_id].A = A;

param[t\_id].n = n;

param[t\_id].act\_row = k + c\*worker\_count + t\_id + 1;

}

//创建线程(核心函数)

for(int t\_id=0;t\_id<worker\_count;t\_id++)

pthread\_create(&handles[t\_id],NULL,threadFunc,&param[t\_id]);

for(int t\_id = 0; t\_id<worker\_count;t\_id++)

//主线程挂起等待所有工作线程完成

pthread\_join(handles[t\_id], NULL);

}

//处理剩余的remain行

pthread\_t\* handles = new pthread\_t[remain];//创建对应的handle参数

threadParam\_t \*param = new threadParam\_t[remain]; //创建对应的线程数据结构

for(int t\_id = 0;t\_id<remain; t\_id++){

param[t\_id].k = k ;

param[t\_id].t\_id = t\_id;

param[t\_id].A = A;

param[t\_id].n = n;

param[t\_id].act\_row = cowork\_cycle\*worker\_count + k + 1 + t\_id;

}

//创建线程(核心函数)

for(int t\_id=0;t\_id<remain;t\_id++)

pthread\_create(&handles[t\_id],NULL,threadFunc,&param[t\_id]);

for(int t\_id = 0; t\_id<remain;t\_id++)

//主线程挂起等待所有工作线程完成

pthread\_join(handles[t\_id], NULL);

}

}

gettimeofday(&end,NULL);

cout<<((long long)end.tv\_sec-(long long)start.tv\_sec)\*1000000+((long long)end.tv\_usec-(long long)start.tv\_usec)<<endl;//微秒

return 0;}