/\*

\* Rectangular matrix multiplication

\* A[M][K] \* B[k][N] = C[M][N]

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <sys/timeb.h>

#include <string.h>

/\* read timer in second \*/

double read\_timer()

{

struct timeb tm;

ftime(&tm);

return (double)tm.time + (double)tm.millitm/1000.0;

}

/\* read timer in ms \*/

double read\_timer\_ms()

{

struct timeb tm;

ftime(&tm);

return (double)tm.time \* 1000.0 + (double)tm.millitm;

}

#define REAL float

void init(int M, int N, REAL A[][N])

{

int i, j;

for (i = 0; i < M; i++) {

for (j = 0; j < N; j++) {

A[i][j] = (REAL)drand48();

}

}

}

double maxerror(int M, int N, REAL A[][N], REAL B[][N])

{

int i, j;

double error = 0.0;

for (i = 0; i < M; i++)

{

for (j = 0; j < N; j++)

{

double diff = (A[i][j] - B[i][j]) / A[i][j];

if (diff < 0)

diff = -diff;

if (diff > error)

error = diff;

}

}

return error;

}

void matmul\_base(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N]);

void matmul\_base\_1(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N]);

void matmul\_base\_sub(int i\_start, int j\_start, int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N]);

void matmul\_row1D(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N], int num\_tasks);

void matmul\_col1D(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N], int num\_tasks);

void matmul\_rowcol2D(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N], int num\_tasks);

int main(int argc,char \*argv[])

{

int N, i, j;

int num\_tasks = 5; /\* 5 is default number of tasks \*/

double elapsed\_base, elapsed\_base\_1, elapsed\_row1d, elapsed\_col1d, elapsed\_rowcol2d; /\* for timing \*/

if (argc < 2) {

fprintf(stderr,"correct format: N for (NxN) and # of tasks as: matmul <n> [<#tasks(%d)>]\n", num\_tasks);

exit(1);

}

N = atoi(argv[1]);

if (argc > 2) num\_tasks = atoi(argv[2]);

REAL A[N][N];

REAL B[N][N];

REAL C1[N][N]; // matmul\_base

REAL C2[N][N]; // matmul\_base\_1

REAL C3[N][N]; // matmul\_row1D

REAL C4[N][N]; // matmul\_column1D

REAL C5[N][N]; // matmul\_rowcol2D

// generating random numbers for the A and B matrix

srand48((1 << 12));

init(N, N, A);

init(N, N, B);

/\* output run \*/

elapsed\_base = read\_timer();

matmul\_base(N,N,N,A,B,C1);

elapsed\_base = (read\_timer() - elapsed\_base);

elapsed\_base\_1 = read\_timer();

matmul\_base\_1(N,N,N,A,B,C2);

elapsed\_base\_1 = (read\_timer() - elapsed\_base\_1);

elapsed\_row1d = read\_timer();

matmul\_row1D(N,N,N,A,B,C3,num\_tasks);

elapsed\_row1d = (read\_timer() - elapsed\_row1d);

elapsed\_col1d = read\_timer();

matmul\_col1D(N,N,N,A,B,C4,num\_tasks);

elapsed\_col1d = (read\_timer() - elapsed\_col1d);

elapsed\_rowcol2d = read\_timer();

matmul\_rowcol2D(N,N,N,A,B,C5,num\_tasks);

elapsed\_rowcol2d = (read\_timer() - elapsed\_rowcol2d);

/\*displaying output\*/

printf ("\n\n");

printf("-==============================================================================-\n");

printf("\t \t \t \tmatmul(%dx%d) \t \t \t \t\n", N, N);

printf("===============================================================================\n");

printf("-------------------------------------------------------------------------------\n");

printf("\t\tA[M][K] \* B[K][N] = C[M][N], M=N=K=%d\t\t\n",N);

printf("-------------------------------------------------------------------------------\n");

printf("\t \t Performance:\t\tRuntime (ms)\t MFLOPS\t \tERROR (Compared to Base)\t\t\n");

printf("\t \t matmul\_base:\t\t%4f\t%4f\t%g\n",elapsed\_base\*1.0e3,((((2.0 \* N) \* N) \* N) / (1.0e6 \* elapsed\_base)),maxerror(N,N,C1,C1));

printf("\t \t matmul\_base\_1:\t\t%4f\t%4f\t%g\t\n",elapsed\_base\_1\*1.0e3,((((2.0 \* N) \* N) \* N) / (1.0e6 \* elapsed\_base\_1)),maxerror(N,N,C1,C2));

printf("\t \t matmul\_row1D:\t\t%4f\t%4f\t%g\t\n",elapsed\_row1d\*1.0e3,((((2.0 \* N) \* N) \* N) / (1.0e6 \* elapsed\_row1d)),maxerror(N,N,C1,C3));

printf("\t \t matmul\_col1D:\t\t%4f\t%4f\t%g\t\n",elapsed\_col1d\*1.0e3,((((2.0 \* N) \* N) \* N) / (1.0e6 \* elapsed\_col1d)),maxerror(N,N,C1,C4));

printf("\t \t matmul\_rowcol2D:\t%4f\t%4f\t%g\t\n",elapsed\_rowcol2d\*1.0e3,((((2.0 \* N) \* N) \* N)/(1.0e6 \* elapsed\_rowcol2d)),maxerror(N,N,C1,C5));

printf("---------------------------------------------------------------------------------\n\n");

return 0;

}

void matmul\_base(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N])

{

// This is to compute the row-wise elements in the matrix C

int i,j,k;

for(i = 0; i < M; i++)

{

for(j = 0; j < N; j++)

{

C[i][j] = 0;

for(k = 0; k < K; k++)

C[i][j] += A[i][k]\*B[k][j];

}

}

}

void matmul\_base\_1(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N])

{

// This is to compute the column-wise elements in the matrix C

int i,j,k;

for(j=0;j<M;j++)

{

for(i=0;i<N;i++)

{

C[i][j] = 0;

for(k=0;k<K;k++)

C[i][j]+=A[i][k]\*B[k][j]

}

}

}

void matmul\_base\_sub(int i\_init, int j\_init,int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N])

{

/\*

Assuming i\_init, j\_init for C matrix

M, K, N general value from A and B

\*/

int i=i\_init,j=j\_init,k;

C[i][j] = 0;

for(k = 0; k < N; k++)

C[i][j] += A[i][k]\*B[k][j];

}

void matmul\_row1D (int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N], int num\_tasks)

{

// consider 1 col from B and A matrix to generate 1 row in C matrix

int i,j,k,z;

int initiali =0, initialj=0;

int num\_of\_work\_in\_one\_task = N / num\_tasks;

int num\_of\_matdata\_left = N % num\_tasks;

// calculated for N not being divisible by #tasks

int residue=0,residuer;

int not\_in\_cycle = N-num\_of\_matdata\_left;

if (num\_of\_matdata\_left == 0)//for N divisible by num\_tasks

{

for (i=0 ; i<num\_tasks; i++)

{

for (j=0; j<num\_of\_work\_in\_one\_task; ++j)

{

for (initiali=0;initiali<N;initiali++)

{

int i\_init = initiali;

int j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A,B,C);

}

initialj = initialj + 1;

}

}

}

else //for N non divisible by num\_tasks

{

for (i=0 ; i<num\_tasks; i++)

{

for (j=0; j<num\_of\_work\_in\_one\_task; ++j)

{

for (initialj=0; initialj<N;++initialj)

{

int i\_init = initiali;

int j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

}

initiali++;

if (initiali == not\_in\_cycle-2)

residue = not\_in\_cycle;

if (residue == not\_in\_cycle)

{

for (z=0;z<num\_of\_matdata\_left;z++)

{

for (initialj=0; initialj<N;++initialj)

{

int i\_init = initiali;

int j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

}

initiali++;

}

}

}

}

}

}

void matmul\_col1D (int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N], int num\_tasks)

{

// consider 1 col from B and A matrix to generate 1 col in C matrix

int i,j,k,z;

int initiali =0, initialj=0;

int num\_of\_work\_in\_one\_task = N / num\_tasks;

int num\_of\_matdata\_left = N % num\_tasks;

// calculated for N not being divisible by #tasks

int residue=N,residuer;

int not\_in\_cycle = N\*num\_of\_work\_in\_one\_task;

if (num\_of\_matdata\_left == 0)//for N divisible by num\_tasks

{

for (i=0 ; i<num\_tasks; i++)

{

for (j=0; j<num\_of\_work\_in\_one\_task; ++j)

{

for (initialj=0;initialj<N;initialj++)

{

int i\_init = initiali;

int j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A,B,C);

}

initiali = initiali + 1;

}

}

}

else//for N non divisible by num\_tasks

{

for (i=0 ; i<num\_tasks; i++)

{

for (j=0; j<num\_of\_work\_in\_one\_task; ++j)

{

for (initiali=0; initiali<N;++initiali)

{

int i\_init = initiali;

int j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

}

initialj++;

if (initialj == not\_in\_cycle)

residue = not\_in\_cycle;

if (initialj+1 == N-num\_of\_matdata\_left)

{

for (z=0;z<num\_of\_matdata\_left;z++)

{

for (initiali=0; initiali<N;++initiali)

{

int i\_init = initiali;

int j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

}

initialj++;

}

}

}

}

}

}

void matmul\_rowcol2D(int M, int K, int N, REAL A[][K], REAL B[][N], REAL C[][N], int num\_tasks)

{

int i,j,k;

int initiali=0,initialj=0;

int i\_init, j\_init;

double b = N/sqrt(num\_tasks);

int no\_of\_rows\_in\_block = b;

int no\_of\_blocks = N/no\_of\_rows\_in\_block;

int completed\_rows = no\_of\_rows\_in\_block\*no\_of\_blocks;

int leftdata = N-completed\_rows;

int residue = 0;

int rownum\_completed = 0;

/\*

For the computation of C matrix for N being non-divisible by num\_tasks

\*/

if (N%no\_of\_rows\_in\_block !=0)

{

for (residue=0;residue<N;)//forout

{

/\*

the first "if" for the completed row condition, is to check when we need to move to the lower half of the C matrix computation. Completed\_rows variable gives the N value. Once the N elements are completed, we need to move to the next block computation, which will follow the row below the already computed one.

\*/

if (residue != completed\_rows )//ifmain

{

for (k=0;k<=no\_of\_blocks;)//formain

{

/\*

The "if" here for the "no\_of\_blocks" is used for the computation of the left-over block data to be considered,

eg., for ./matmul 32 5

no\_of\_rows\_in\_block = 14

so, two 14X14 block will be computed and 4 elements will be left,

hence,there will be one 14X4 (left-most) and one 4X14(down) blocks

will be left unattended,which is computed below

\*/

if (k == no\_of\_blocks)//if

{

for(i=0;i<no\_of\_rows\_in\_block;i++)//for1

{

for(j=0; j < leftdata ; j++)

{

i\_init = initiali;

j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

initialj++;

}

initialj = initialj-leftdata;

initiali++;

}//for1

residue = residue + no\_of\_rows\_in\_block;

initialj =0;

k=k+1;

}//if

else

{

for(i=0; i<no\_of\_rows\_in\_block;i++)//for2

{

for(j=0; j<no\_of\_rows\_in\_block; j++)//for3

{

i\_init = initiali;

j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

initialj++;

}//for3

initialj = initialj-no\_of\_rows\_in\_block;

initiali++;

}//for2

k=k+1;

initiali = initiali-no\_of\_rows\_in\_block;

initialj = initialj+no\_of\_rows\_in\_block;

}//else

}//formain part

}//ifmain

else

{

rownum\_completed = rownum\_completed + no\_of\_rows\_in\_block;

for(i=0;i<leftdata;i++)//for1

{

for(j=0; j<no\_of\_rows\_in\_block; j++)

{

i\_init = initiali;

j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

initialj++;

}

initialj = initialj-no\_of\_rows\_in\_block;

initiali++;

}//for1

initiali = initiali - leftdata;

initialj = initialj+no\_of\_rows\_in\_block;

if (rownum\_completed == completed\_rows )

{

residue = residue + no\_of\_rows\_in\_block;

}

}

}//forout

/\*

This is to compute the left-over part of C matrix

after the block decomposition is done,

say, in our eg. of ./matmul 32

there will be 4X4 matrix to be computed, done here.

\*/

for (i=0 ; i<leftdata;i++ )

{

for (j=0 ; j<leftdata; j++)

{

i\_init = initiali;

j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

initialj++;

}

initialj = initialj-leftdata;

initiali++;

}

}

/\*

For the computation of C matrix for N being non-divisible by num\_tasks

This part of the program will compute the C matrix block-wise,

for N being exactly divisible by squareroot of num\_tasks.

the code computes the C matrix block by block using the counter values

initiali and initialj values.

\*/

else

{

for (completed\_rows =0; completed\_rows<2;completed\_rows++)

{

for (k=0;k<no\_of\_blocks;)

{

for(i=0; i<no\_of\_rows\_in\_block;i++)//for2

{

for(j=0; j<no\_of\_rows\_in\_block; j++)//for3

{

i\_init = initiali;

j\_init = initialj;

matmul\_base\_sub(i\_init,j\_init,M,K,N,A, B, C);

initialj++;

}//for3

initialj = initialj-no\_of\_rows\_in\_block;

initiali++;

}//for2

k=k+1;

initiali = initiali-no\_of\_rows\_in\_block;

initialj = initialj+no\_of\_rows\_in\_block;

}

initiali = initiali+no\_of\_rows\_in\_block;

initialj = 0;

}

}

}