```cpp
///========================================================================
/// CSCI 176 Program 3
/// Kenneth Willeford
///
///     This program performs parallel matrix multiplication on matrices
///     of the form specified in prog3.pdf.
///     Once compiled the program is ran as so...
///         <exe_name> <L_value> <m_value> <n_value> <number_of_thread>
///         ie our required test cases...
///         g++ mmultiply.cpp && ./a.out 1003 2000 3000 1 > output1.txt
///         g++ mmultiply.cpp && ./a.out 1003 2000 3000 2 > output2.txt
///         g++ mmultiply.cpp && ./a.out 1003 2000 3000 4 > output4.txt
///         g++ mmultiply.cpp && ./a.out 1003 2000 3000 8 > output8.txt
///         The relevant output for these test cases will be provided separately.
///     Additional Notes-
///         I thought I would also turn this program into a practice in
///         cache optimization. I create a 'matrix' class that can be built
///         'rotated', which allows for row-major access on column elements.
///         This is useful for optimizing the layout of data for the B operand.
///========================================================================
#include<iostream>      // cout
#include<cstdlib>       // atoi
#include<vector>        // Dynamic Arrays (I prefer not to use the manual approach if I
don't have to.)
#include<ctime>         // For Benchmarking
#include<pthread.h>     // pthread_t, mthread_mutex_t, _lock, _unlock, _create, _join
#include <sstream>      // for capturing output to send to the guarded cout
using namespace std;
//========================================================================
// Object Definitions / Implementation
//========================================================================
class matrix{
    // This bool is set when the matrix is created. It marks whether or not the matrix
    was created while 'rotated.'
    // A rotated matrix's its column vectors are in row major.
    bool rotated;
    // The actual matrix as a 2D Dynamic Array
    vector<vector<long long int> > m;
    // The Number of Rows/Columns
    long long int r,c;
    public:
    // Empty Constructor
    matrix(){}
    // long long intiializes Matrix with its Dimensions.
    matrix(long long int rows, long long int columns, bool rot = false){
        rotated = rot;
        if(rotated)
            m = vector<vector<long long int> >(columns,vector<long long int>(rows,0));
        else
            m = vector<vector<long long int> >(rows,vector<long long int>(columns,0));
        r = rows;
        c = columns;
    }
    // Provides the number of rows within the matrix.
    long long int rows(){ return r; }
    // Provides the number of columns within the matrix.
    long long int columns(){ return c; }
    // Retrieves an individual value in the matrix by row and column.
    long long int get(long long int row, long long int column){
        if(rotated)
            return m[column][row];
        else
            return m[row][column];
    }
    // Assigns an inidivdual value to the matrix by row and column.
    void assign(long long int row, long long int column, long long int val){
        if(rotated)
            m[column][row] = val;
        else
            m[row][column] = val;
    }
};
//=======================================================================
// Global Values
//=======================================================================
namespace GLOBAL {
    // Corresponds to the L,m,n values in the prog3.pdf specifications
    long long int L,m,n;
    // remainder rows that will need to be added in during cyclical assignment.
    long long int remainderRows = 0;
    // The number of threads
    long long int num_threads = 1;
    // The the input matrices and the output matrix.
    matrix A,B,C;
    // The execution time of a benchmark in seconds.
    double executionTime;
};
//=======================================================================
// Function Definitions
//=======================================================================
// Performs a cout command on a string guarded by a mutex.
void cout_semaphore(string);
// Master function for the matrix multiplication. Statically generates the needed
threads and executes the matrix multiplication.
void*matrixMultiplicationMaster(void*);
// Gets four command line arguments. L,m,n,num_threads
void getCommandLineArguments(char* argv[]);
// Utilizes L,m,n and num_threads to initialize the matrices and any needed metadata.
void initializeMatrices();
// Performs the dot product between a row in A and a column in B.
long long int dotProduct(long long int rowA, long long int colB);
// Slave function for matrix multiplication. Performs part of the multiplication.
void* matrixMultiplication(void*);
// Takes in a generic function and performs benchmarking on it, stores the result
globally.
void benchmark(void*(*)(void*),void*arg);
// Prints the final output.
void printResults();
//=======================================================================
// Main
//=======================================================================
int main(int argc, char* argv[]){
    getCommandLineArguments(argv);
    initializeMatrices();
    benchmark(matrixMultiplicationMaster,NULL);
    printResults();
}
//=======================================================================
// Function Implementations
//=======================================================================
void* matrixMultiplication(void* arg){
    // Get Current Thread ID
    long long int my_rank = (long long int)arg;
    // Output Data to screen
    ostringstream ss;
    ss << "Thread_" << my_rank << ": " << my_rank << " ~ " << GLOBAL::C.rows() << ",
    step " << GLOBAL::num_threads << endl;
    cout_semaphore(ss.str());
    // If there are 'remainderRows' remaining...
    if(my_rank < GLOBAL::remainderRows){
        // Then go ahead and account for extra rows...
        for(long long int i = my_rank; i < GLOBAL::C.rows(); i+=GLOBAL::num_threads)
            for(long long int j = 0; j < GLOBAL::C.columns(); j++)
                GLOBAL::C.assign(i,j,dotProduct(i,j));
    // Otherwise...
    } else {
        // Don't accout for extra rows. (don't hit the same rows twice.)
        for(long long int i = my_rank; i < GLOBAL::C.rows() - GLOBAL::remainderRows;
        i+=GLOBAL::num_threads)
            for(long long int j = 0; j < GLOBAL::C.columns(); j++)
                GLOBAL::C.assign(i,j,dotProduct(i,j));
    }
    // So the compiler doesn't return a warning.
    return NULL;
}

long long int dotProduct(long long int rowA, long long int colB){
    long long int sum = 0;
    long long int range = GLOBAL::m;
    for(long long int i = 0; i < range; i++)
        // Perform Dot Product to Build up sum
        sum += GLOBAL::A.get(rowA,i) * GLOBAL::B.get(i,colB);
    return sum;
}

void getCommandLineArguments(char* argv[]){
    GLOBAL::L = atoi(argv[1]);
    GLOBAL::m = atoi(argv[2]);
    GLOBAL::n = atoi(argv[3]);
    // If the number of threads specified are less than 1(or not specified at all)
    default to 1.
    GLOBAL::num_threads = atoi(argv[4]) < 1 ? 1:atoi(argv[4]);
    // Print to Screen
    cout << "L=" << GLOBAL::L << ",m=" << GLOBAL::m << ",n=" << GLOBAL::n << endl;
}

void initializeMatrices(){
    // Initialize Matrix A according to prog3.pdf Specifications
    GLOBAL::A = matrix(GLOBAL::L,GLOBAL::m);
    for(long long int i = 0; i < GLOBAL::A.rows(); i++)
        for(long long int j = 0; j < GLOBAL::A.columns(); j++)
            GLOBAL::A.assign(i,j,i+j+1);
    // Initialize Matrix B according to prog3.pdf Specifications
    GLOBAL::B = matrix(GLOBAL::m,GLOBAL::n,true);
    for(long long int i = 0; i < GLOBAL::B.rows(); i++)
        for(long long int j = 0; j < GLOBAL::B.columns(); j++)
            GLOBAL::B.assign(i,j,i+j);
    // Initialize Matrix C  dimensions according to prog3.pdf Specifications
    GLOBAL::C = matrix(GLOBAL::L,GLOBAL::n);
    // Determine the remainder rows. Guarding this statement is proabbly unnecessary.
    if(GLOBAL::num_threads > 0)
        GLOBAL::remainderRows = GLOBAL::C.rows() % GLOBAL::num_threads;
}

void benchmark(void*(*f)(void*),void*arg){
    // Begin Timing
    clock_t t = clock();
    // Run Function
    f(arg);
    // Get Timing
    GLOBAL::executionTime = ((float)(clock()-t))/CLOCKS_PER_SEC;
}
```

```cpp

void*matrixMultiplicationMaster(void*){
    // Build up the requested number of threads and launch them with their IDs.
    vector<pthread_t> threads = vector<pthread_t>(GLOBAL::num_threads);
    for(long long int i = 0; i < GLOBAL::num_threads; i++)
        pthread_create(&threads[i], NULL, matrixMultiplication, (void*)i);
    // Wait for each thread to finish before continuing.
    for(long long int i = 0; i < GLOBAL::num_threads; i++)
        pthread_join(threads[i],NULL);
}

void printResults(){
    // Print the related sub-matrix
    cout << "===C:first_20*first_10===" << endl;
    for(long long int i = 0; i < 20 && i < GLOBAL::C.rows(); i++){
        for(long long int j = 0; j < 10 && j < GLOBAL::C.columns(); j++){
            cout << GLOBAL::C.get(i,j) << " ";
        }
        cout << endl;
    }
    // Print the related sub-matrix
    cout << "===C:last_20*last_10===" << endl;
    for(long long int i = GLOBAL::C.rows() - 20; i > 0 && i < GLOBAL::C.rows(); i++){
        for(long long int j = GLOBAL::C.columns() - 10; j > 0 && j <
        GLOBAL::C.columns(); j++){
            cout << GLOBAL::C.get(i,j) << " ";
        }
        cout << endl;
    }
    // Print benchmark information
    cout << "Time taken (sec) = " << GLOBAL::executionTime << endl;
}

void cout_semaphore(string s){
    // Shared lock between function calls. Protects cout.
    static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
    pthread_mutex_lock(&lock);
    cout << s << endl;
    pthread_mutex_unlock(&lock);
}
```

```
L=1003,m=2000,n=3000
Thread_0: 0 ~ 1003, step 1

===C:first_20*first_10===
2666666000 2668667000 2670668000 2672669000 2674670000 2676671000 2678672000
2680673000 2682674000 2684675000
2668665000 2670668000 2672671000 2674674000 2676677000 2678680000 2680683000
2682686000 2684689000 2686692000
2670664000 2672669000 2674674000 2676679000 2678684000 2680689000 2682694000
2684699000 2686704000 2688709000
2672663000 2674670000 2676677000 2678684000 2680691000 2682698000 2684705000
2686712000 2688719000 2690726000
2674662000 2676671000 2678680000 2680689000 2682698000 2684707000 2686716000
2688725000 2690734000 2692743000
2676661000 2678672000 2680683000 2682694000 2684705000 2686716000 2688727000
2690738000 2692749000 2694760000
2678660000 2680673000 2682686000 2684699000 2686712000 2688725000 2690738000
2692751000 2694764000 2696777000
2680659000 2682674000 2684689000 2686704000 2688719000 2690734000 2692749000
2694764000 2696779000 2698794000
2682658000 2684675000 2686692000 2688709000 2690726000 2692743000 2694760000
2696777000 2698794000 2700811000
2684657000 2686676000 2688695000 2690714000 2692733000 2694752000 2696771000
2698790000 2700809000 2702828000
2686656000 2688677000 2690698000 2692719000 2694740000 2696761000 2698782000
2700803000 2702824000 2704845000
2688655000 2690678000 2692701000 2694724000 2696747000 2698770000 2700793000
2702816000 2704839000 2706862000
2690654000 2692679000 2694704000 2696729000 2698754000 2700779000 2702804000
2704829000 2706854000 2708879000
2692653000 2694680000 2696707000 2698734000 2700761000 2702788000 2704815000
2706842000 2708869000 2710896000
2694652000 2696681000 2698710000 2700739000 2702768000 2704797000 2706826000
2708855000 2710884000 2712913000
2696651000 2698682000 2700713000 2702744000 2704775000 2706806000 2708837000
2710868000 2712899000 2714930000
2698650000 2700683000 2702716000 2704749000 2706782000 2708815000 2710848000
2712881000 2714914000 2716947000
2700649000 2702684000 2704719000 2706754000 2708789000 2710824000 2712859000
2714894000 2716929000 2718964000
2702648000 2704685000 2706722000 2708759000 2710796000 2712833000 2714870000
2716907000 2718944000 2720981000
2704647000 2706686000 2708725000 2710764000 2712803000 2714842000 2716881000
2718920000 2720959000 2722998000
===C:last_20*last_10===
16493013000 16496980000 16500947000 16504914000 16508881000 16512848000 16516815000
16520782000 16524749000 16528716000
16500992000 16504961000 16508930000 16512899000 16516868000 16520837000 16524806000
16528775000 16532744000 16536713000
16508971000 16512942000 16516913000 16520884000 16524855000 16528826000 16532797000
16536768000 16540739000 16544710000
16516950000 16520923000 16524896000 16528869000 16532842000 16536815000 16540788000
16544761000 16548734000 16552707000
16524929000 16528904000 16532879000 16536854000 16540829000 16544804000 16548779000
16552754000 16556729000 16560704000
16532908000 16536885000 16540862000 16544839000 16548816000 16552793000 16556770000
16560747000 16564724000 16568701000
16540887000 16544866000 16548845000 16552824000 16556803000 16560782000 16564761000
16568740000 16572719000 16576698000
16548866000 16552847000 16556828000 16560809000 16564790000 16568771000 16572752000
16576733000 16580714000 16584695000
16556845000 16560828000 16564811000 16568794000 16572777000 16576760000 16580743000
16584726000 16588709000 16592692000
```

```
16564824000 16568809000 16572794000 16576779000 16580764000 16584749000 16588734000
16592719000 16596704000 16600689000
16572803000 16576790000 16580777000 16584764000 16588751000 16592738000 16596725000
16600712000 16604699000 16608686000
16580782000 16584771000 16588760000 16592749000 16596738000 16600727000 16604716000
16608705000 16612694000 16616683000
16588761000 16592752000 16596743000 16600734000 16604725000 16608716000 16612707000
16616698000 16620689000 16624680000
16596740000 16600733000 16604726000 16608719000 16612712000 16616705000 16620698000
16624691000 16628684000 16632677000
16604719000 16608714000 16612709000 16616704000 16620699000 16624694000 16628689000
16632684000 16636679000 16640674000
16612698000 16616695000 16620692000 16624689000 16628686000 16632683000 16636680000
16640677000 16644674000 16648671000
16620677000 16624676000 16628675000 16632674000 16636673000 16640672000 16644671000
16648670000 16652669000 16656668000
16628656000 16632657000 16636658000 16640659000 16644660000 16648661000 16652662000
16656663000 16660664000 16664665000
16636635000 16640638000 16644641000 16648644000 16652647000 16656650000 16660653000
16664656000 16668659000 16672662000
16644614000 16648619000 16652624000 16656629000 16660634000 16664639000 16668644000
16672649000 16676654000 16680659000
Time taken (sec) = 196.517
```

```
L=1003,m=2000,n=3000
Thread_0: 0 ~ 1003, step 2

Thread_1: 1 ~ 1003, step 2

===C:first_20*first_10===
2666666000 2668667000 2670668000 2672669000 2674670000 2676671000 2678672000
2680673000 2682674000 2684675000
2668665000 2670668000 2672671000 2674674000 2676677000 2678680000 2680683000
2682686000 2684689000 2686692000
2670664000 2672669000 2674674000 2676679000 2678684000 2680689000 2682694000
2684699000 2686704000 2688709000
2672663000 2674670000 2676677000 2678684000 2680691000 2682698000 2684705000
2686712000 2688719000 2690726000
2674662000 2676671000 2678680000 2680689000 2682698000 2684707000 2686716000
2688725000 2690734000 2692743000
2676661000 2678672000 2680683000 2682694000 2684705000 2686716000 2688727000
2690738000 2692749000 2694760000
2678660000 2680673000 2682686000 2684699000 2686712000 2688725000 2690738000
2692751000 2694764000 2696777000
2680659000 2682674000 2684689000 2686704000 2688719000 2690734000 2692749000
2694764000 2696779000 2698794000
2682658000 2684675000 2686692000 2688709000 2690726000 2692743000 2694760000
2696777000 2698794000 2700811000
2684657000 2686676000 2688695000 2690714000 2692733000 2694752000 2696771000
2698790000 2700809000 2702828000
2686656000 2688677000 2690698000 2692719000 2694740000 2696761000 2698782000
2700803000 2702824000 2704845000
2688655000 2690678000 2692701000 2694724000 2696747000 2698770000 2700793000
2702816000 2704839000 2706862000
2690654000 2692679000 2694704000 2696729000 2698754000 2700779000 2702804000
2704829000 2706854000 2708879000
2692653000 2694680000 2696707000 2698734000 2700761000 2702788000 2704815000
2706842000 2708869000 2710896000
2694652000 2696681000 2698710000 2700739000 2702768000 2704797000 2706826000
2708855000 2710884000 2712913000
2696651000 2698682000 2700713000 2702744000 2704775000 2706806000 2708837000
2710868000 2712899000 2714930000
2698650000 2700683000 2702716000 2704749000 2706782000 2708815000 2710848000
2712881000 2714914000 2716947000
2700649000 2702684000 2704719000 2706754000 2708789000 2710824000 2712859000
2714894000 2716929000 2718964000
2702648000 2704685000 2706722000 2708759000 2710796000 2712833000 2714870000
2716907000 2718944000 2720981000
2704647000 2706686000 2708725000 2710764000 2712803000 2714842000 2716881000
2718920000 2720959000 2722998000
===C:last_20*last_10===
16493013000 16496980000 16500947000 16504914000 16508881000 16512848000 16516815000
16520782000 16524749000 16528716000
16500992000 16504961000 16508930000 16512899000 16516868000 16520837000 16524806000
16528775000 16532744000 16536713000
16508971000 16512942000 16516913000 16520884000 16524855000 16528826000 16532797000
16536768000 16540739000 16544710000
16516950000 16520923000 16524896000 16528869000 16532842000 16536815000 16540788000
16544761000 16548734000 16552707000
16524929000 16528904000 16532879000 16536854000 16540829000 16544804000 16548779000
16552754000 16556729000 16560704000
16532908000 16536885000 16540862000 16544839000 16548816000 16552793000 16556770000
16560747000 16564724000 16568701000
16540887000 16544866000 16548845000 16552824000 16556803000 16560782000 16564761000
16568740000 16572719000 16576698000
16548866000 16552847000 16556828000 16560809000 16564790000 16568771000 16572752000
16576733000 16580714000 16584695000
```

```
16556845000 16560828000 16564811000 16568794000 16572777000 16576760000 16580743000
16584726000 16588709000 16592692000
16564824000 16568809000 16572794000 16576779000 16580764000 16584749000 16588734000
16592719000 16596704000 16600689000
16572803000 16576790000 16580777000 16584764000 16588751000 16592738000 16596725000
16600712000 16604699000 16608686000
16580782000 16584771000 16588760000 16592749000 16596738000 16600727000 16604716000
16608705000 16612694000 16616683000
16588761000 16592752000 16596743000 16600734000 16604725000 16608716000 16612707000
16616698000 16620689000 16624680000
16596740000 16600733000 16604726000 16608719000 16612712000 16616705000 16620698000
16624691000 16628684000 16632677000
16604719000 16608714000 16612709000 16616704000 16620699000 16624694000 16628689000
16632684000 16636679000 16640674000
16612698000 16616695000 16620692000 16624689000 16628686000 16632683000 16636680000
16640677000 16644674000 16648671000
16620677000 16624676000 16628675000 16632674000 16636673000 16640672000 16644671000
16648670000 16652669000 16656668000
16628656000 16632657000 16636658000 16640659000 16644660000 16648661000 16652662000
16656663000 16660664000 16664665000
16636635000 16640638000 16644641000 16648644000 16652647000 16656650000 16660653000
16664656000 16668659000 16672662000
16644614000 16648619000 16652624000 16656629000 16660634000 16664639000 16668644000
16672649000 16676654000 16680659000
Time taken (sec) = 114.446
```

```
L=1003,m=2000,n=3000
Thread_0: 0 ~ 1003, step 4

Thread_1: 1 ~ 1003, step 4

Thread_2: 2 ~ 1003, step 4

Thread_3: 3 ~ 1003, step 4

===C:first_20*first_10===
2666666000 2668667000 2670668000 2672669000 2674670000 2676671000 2678672000
2680673000 2682674000 2684675000
2668665000 2670668000 2672671000 2674674000 2676677000 2678680000 2680683000
2682686000 2684689000 2686692000
2670664000 2672669000 2674674000 2676679000 2678684000 2680689000 2682694000
2684699000 2686704000 2688709000
2672663000 2674670000 2676677000 2678684000 2680691000 2682698000 2684705000
2686712000 2688719000 2690726000
2674662000 2676671000 2678680000 2680689000 2682698000 2684707000 2686716000
2688725000 2690734000 2692743000
2676661000 2678672000 2680683000 2682694000 2684705000 2686716000 2688727000
2690738000 2692749000 2694760000
2678660000 2680673000 2682686000 2684699000 2686712000 2688725000 2690738000
2692751000 2694764000 2696777000
2680659000 2682674000 2684689000 2686704000 2688719000 2690734000 2692749000
2694764000 2696779000 2698794000
2682658000 2684675000 2686692000 2688709000 2690726000 2692743000 2694760000
2696777000 2698794000 2700811000
2684657000 2686676000 2688695000 2690714000 2692733000 2694752000 2696771000
2698790000 2700809000 2702828000
2686656000 2688677000 2690698000 2692719000 2694740000 2696761000 2698782000
2700803000 2702824000 2704845000
2688655000 2690678000 2692701000 2694724000 2696747000 2698770000 2700793000
2702816000 2704839000 2706862000
2690654000 2692679000 2694704000 2696729000 2698754000 2700779000 2702804000
2704829000 2706854000 2708879000
2692653000 2694680000 2696707000 2698734000 2700761000 2702788000 2704815000
2706842000 2708869000 2710896000
2694652000 2696681000 2698710000 2700739000 2702768000 2704797000 2706826000
2708855000 2710884000 2712913000
2696651000 2698682000 2700713000 2702744000 2704775000 2706806000 2708837000
2710868000 2712899000 2714930000
2698650000 2700683000 2702716000 2704749000 2706782000 2708815000 2710848000
2712881000 2714914000 2716947000
2700649000 2702684000 2704719000 2706754000 2708789000 2710824000 2712859000
2714894000 2716929000 2718964000
2702648000 2704685000 2706722000 2708759000 2710796000 2712833000 2714870000
2716907000 2718944000 2720981000
2704647000 2706686000 2708725000 2710764000 2712803000 2714842000 2716881000
2718920000 2720959000 2722998000
===C:last_20*last_10===
16493013000 16496980000 16500947000 16504914000 16508881000 16512848000 16516815000
16520782000 16524749000 16528716000
16500992000 16504961000 16508930000 16512899000 16516868000 16520837000 16524806000
16528775000 16532744000 16536713000
16508971000 16512942000 16516913000 16520884000 16524855000 16528826000 16532797000
16536768000 16540739000 16544710000
16516950000 16520923000 16524896000 16528869000 16532842000 16536815000 16540788000
16544761000 16548734000 16552707000
16524929000 16528904000 16532879000 16536854000 16540829000 16544804000 16548779000
16552754000 16556729000 16560704000
16532908000 16536885000 16540862000 16544839000 16548816000 16552793000 16556770000
16560747000 16564724000 16568701000
```

```
16540887000 16544866000 16548845000 16552824000 16556803000 16560782000 16564761000
16568740000 16572719000 16576698000
16548866000 16552847000 16556828000 16560809000 16564790000 16568771000 16572752000
16576733000 16580714000 16584695000
16556845000 16560828000 16564811000 16568794000 16572777000 16576760000 16580743000
16584726000 16588709000 16592692000
16564824000 16568809000 16572794000 16576779000 16580764000 16584749000 16588734000
16592719000 16596704000 16600689000
16572803000 16576790000 16580777000 16584764000 16588751000 16592738000 16596725000
16600712000 16604699000 16608686000
16580782000 16584771000 16588760000 16592749000 16596738000 16600727000 16604716000
16608705000 16612694000 16616683000
16588761000 16592752000 16596743000 16600734000 16604725000 16608716000 16612707000
16616698000 16620689000 16624680000
16596740000 16600733000 16604726000 16608719000 16612712000 16616705000 16620698000
16624691000 16628684000 16632677000
16604719000 16608714000 16612709000 16616704000 16620699000 16624694000 16628689000
16632684000 16636679000 16640674000
16612698000 16616695000 16620692000 16624689000 16628686000 16632683000 16636680000
16640677000 16644674000 16648671000
16620677000 16624676000 16628675000 16632674000 16636673000 16640672000 16644671000
16648670000 16652669000 16656668000
16628656000 16632657000 16636658000 16640659000 16644660000 16648661000 16652662000
16656663000 16660664000 16664665000
16636635000 16640638000 16644641000 16648644000 16652647000 16656650000 16660653000
16664656000 16668659000 16672662000
16644614000 16648619000 16652624000 16656629000 16660634000 16664639000 16668644000
16672649000 16676654000 16680659000
Time taken (sec) = 92.135
```

```
L=1003,m=2000,n=3000
Thread_0: 0 ~ 1003, step 8

Thread_1: 1 ~ 1003, step 8

Thread_2: 2 ~ 1003, step 8

Thread_3: 3 ~ 1003, step 8

Thread_4: 4 ~ 1003, step 8

Thread_5: 5 ~ 1003, step 8

Thread_6: 6 ~ 1003, step 8

Thread_7: 7 ~ 1003, step 8

===C:first_20*first_10===
2666666000 2668667000 2670668000 2672669000 2674670000 2676671000 2678672000
2680673000 2682674000 2684675000
2668665000 2670668000 2672671000 2674674000 2676677000 2678680000 2680683000
2682686000 2684689000 2686692000
2670664000 2672669000 2674674000 2676679000 2678684000 2680689000 2682694000
2684699000 2686704000 2688709000
2672663000 2674670000 2676677000 2678684000 2680691000 2682698000 2684705000
2686712000 2688719000 2690726000
2674662000 2676671000 2678680000 2680689000 2682698000 2684707000 2686716000
2688725000 2690734000 2692743000
2676661000 2678672000 2680683000 2682694000 2684705000 2686716000 2688727000
2690738000 2692749000 2694760000
2678660000 2680673000 2682686000 2684699000 2686712000 2688725000 2690738000
2692751000 2694764000 2696777000
2680659000 2682674000 2684689000 2686704000 2688719000 2690734000 2692749000
2694764000 2696779000 2698794000
2682658000 2684675000 2686692000 2688709000 2690726000 2692743000 2694760000
2696777000 2698794000 2700811000
2684657000 2686676000 2688695000 2690714000 2692733000 2694752000 2696771000
2698790000 2700809000 2702828000
2686656000 2688677000 2690698000 2692719000 2694740000 2696761000 2698782000
2700803000 2702824000 2704845000
2688655000 2690678000 2692701000 2694724000 2696747000 2698770000 2700793000
2702816000 2704839000 2706862000
2690654000 2692679000 2694704000 2696729000 2698754000 2700779000 2702804000
2704829000 2706854000 2708879000
2692653000 2694680000 2696707000 2698734000 2700761000 2702788000 2704815000
2706842000 2708869000 2710896000
2694652000 2696681000 2698710000 2700739000 2702768000 2704797000 2706826000
2708855000 2710884000 2712913000
2696651000 2698682000 2700713000 2702744000 2704775000 2706806000 2708837000
2710868000 2712899000 2714930000
2698650000 2700683000 2702716000 2704749000 2706782000 2708815000 2710848000
2712881000 2714914000 2716947000
2700649000 2702684000 2704719000 2706754000 2708789000 2710824000 2712859000
2714894000 2716929000 2718964000
2702648000 2704685000 2706722000 2708759000 2710796000 2712833000 2714870000
2716907000 2718944000 2720981000
2704647000 2706686000 2708725000 2710764000 2712803000 2714842000 2716881000
2718920000 2720959000 2722998000
===C:last_20*last_10===
16493013000 16496980000 16500947000 16504914000 16508881000 16512848000 16516815000
16520782000 16524749000 16528716000
16500992000 16504961000 16508930000 16512899000 16516868000 16520837000 16524806000
16528775000 16532744000 16536713000
```

```
16508971000 16512942000 16516913000 16520884000 16524855000 16528826000 16532797000
16536768000 16540739000 16544710000
16516950000 16520923000 16524896000 16528869000 16532842000 16536815000 16540788000
16544761000 16548734000 16552707000
16524929000 16528904000 16532879000 16536854000 16540829000 16544804000 16548779000
16552754000 16556729000 16560704000
16532908000 16536885000 16540862000 16544839000 16548816000 16552793000 16556770000
16560747000 16564724000 16568701000
16540887000 16544866000 16548845000 16552824000 16556803000 16560782000 16564761000
16568740000 16572719000 16576698000
16548866000 16552847000 16556828000 16560809000 16564790000 16568771000 16572752000
16576733000 16580714000 16584695000
16556845000 16560828000 16564811000 16568794000 16572777000 16576760000 16580743000
16584726000 16588709000 16592692000
16564824000 16568809000 16572794000 16576779000 16580764000 16584749000 16588734000
16592719000 16596704000 16600689000
16572803000 16576790000 16580777000 16584764000 16588751000 16592738000 16596725000
16600712000 16604699000 16608686000
16580782000 16584771000 16588760000 16592749000 16596738000 16600727000 16604716000
16608705000 16612694000 16616683000
16588761000 16592752000 16596743000 16600734000 16604725000 16608716000 16612707000
16616698000 16620689000 16624680000
16596740000 16600733000 16604726000 16608719000 16612712000 16616705000 16620698000
16624691000 16628684000 16632677000
16604719000 16608714000 16612709000 16616704000 16620699000 16624694000 16628689000
16632684000 16636679000 16640674000
16612698000 16616695000 16620692000 16624689000 16628686000 16632683000 16636680000
16640677000 16644674000 16648671000
16620677000 16624676000 16628675000 16632674000 16636673000 16640672000 16644671000
16648670000 16652669000 16656668000
16628656000 16632657000 16636658000 16640659000 16644660000 16648661000 16652662000
16656663000 16660664000 16664665000
16636635000 16640638000 16644641000 16648644000 16652647000 16656650000 16660653000
16664656000 16668659000 16672662000
16644614000 16648619000 16652624000 16656629000 16660634000 16664639000 16668644000
16672649000 16676654000 16680659000
Time taken (sec) = 99.401
```