مرحله‌ی اول:

۱. قسمت for (j = 0; j < VERYBIG; j++).
iterationهای حلقه‌ی `for (i = 0; i < 10; i++)` از یک‌دیگر مستقل هستند و می‌توانند موازی‌سازی شوند.

۲. محاسبه‌ی تاثیر عوامل خارجی (بقیه پروسه‌های موجود در سیستم و سیستم‌عامل) در زمان اجرای برنامه. قسمت `Time Elapsed` بین تکرارها متفاوت است. (به علت تاثیرهای ذکر شده)

۳. حالت `debug` برای استفاده در زمان `development` و `testing` توسط برنامه‌نویس است که `Visual Studio` در این حالت با ذخیره‌کردن `header`های `debug` در فایل در حال کامپایل، یافتن مشکلات برنامه (با استفاده از ابزارهایی همچون `breakpoint`) آسان‌تر می‌کند ولی در حالت `release` این امر انجام نمی‌شود و در عوض کامپایلر زبان `C++` تلاش می‌کند تا با ایجاد `optimization` تا جای ممکن سرعت برنامه را افزایش دهد. به همین علت کد کامپایل‌شده‌ی قسمت `release` سریع‌تر است.

۴. `loop decomposition` که `iteration`های مختلف یک حلقه را به صورت موازی اجرا می‌کند.

مرحله دوم:

مقایسه‌ی زمان‌های اجرا:

```
Time Elapsed: 1.404635 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.434006 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.407637 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.371026 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.411491 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.347880 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.342539 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.353172 Secs, Total = 30.656747, Check Sum = 50000
```

Using Reduction

```
Serial Timings for 50000 iterations

Time Elapsed: 1.529956 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.543466 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.441585 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.342374 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.328263 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.412324 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.462198 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.382420 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.461533 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.365399 Secs, Total = 30.656747, Check Sum = 50000
```

Using Custom Code with arrays on stack

```
Serial Timings for 50000 iterations

Time Elapsed: 1.443528 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.464844 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.446758 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.359486 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.368157 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.333837 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.349341 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.373533 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.352320 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.335911 Secs, Total = 30.656747, Check Sum = 50000
```

Using Critical

```
Serial Timings for 50000 iterations

Time Elapsed: 1.646439 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.410758 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.351108 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.404366 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.435217 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.368556 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.337432 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.335441 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.327631 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 1.321985 Secs, Total = 30.656747, Check Sum = 50000
```

Using custom code with vector

کد استفاده شده در قسمت Custom – Arrays on stack:

```cpp
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <algorithm>
#include <numeric>
using namespace std;

const long int VERYBIG = 50000;
// ************************************************************************
int main(void)
{
    int i;
    long int j, k, sum;
    double sumx, sumy, total;
    double starttime, elapsedtime;
    // ------------------------------------------------------------------------
    printf("Serial Timings for %d iterations\n\n", VERYBIG);
    for (i = 0; i < 10; i++)
    {
        starttime = omp_get_wtime();
        int sums[VERYBIG] = {};
        double totals[VERYBIG] = {};
        sum = 0;
        total = 0.0;

#pragma omp parallel for private(k, sumx, sumy)
        for (j = 0; j < VERYBIG; j++)
        {
            sums[j] += 1;
            sumx = 0.0;
            for (k = 0; k < j; k++)
                sumx = sumx + (double)k;
            sumy = 0.0;
            for (k = j; k > 0; k--)
                sumy = sumy + (double)k;
            if (sumx > 0.0) totals[j] = totals[j] + 1.0 / sqrt(sumx);
            if (sumy > 0.0) totals[j] = totals[j] + 1.0 / sqrt(sumy);
        }
        sum = accumulate(begin(sums), end(sums), 0);
        total = accumulate(begin(totals), end(totals), 0.0);
        elapsedtime = omp_get_wtime() - starttime;
        printf("Time Elapsed: %f Secs, Total = %lf, Check Sum = %ld\n",
            elapsedtime, total, sum);
    }
    getchar();

    return 0;
}
```

کد قسمت Custom – with Vector:

```cpp
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <algorithm>
#include <numeric>
#include <vector>

using namespace std;

const long int VERYBIG = 50000;
// ************************************************************************
int main(void)
{
    int i;
    long int j, k, sum;
    double sumx, sumy, total;
    double starttime, elapsedtime;
    // ------------------------------------------------------------------------
    printf("Serial Timings for %d iterations\n\n", VERYBIG);
    for (i = 0; i < 10; i++)
    {
        starttime = omp_get_wtime();
        vector<int> sums(VERYBIG);
        vector<double> totals(VERYBIG);
        sum = 0;
        total = 0.0;

#pragma omp parallel for private(k, sumx, sumy)
        for (j = 0; j < VERYBIG; j++)
        {
            sums[j] += 1;
            sumx = 0.0;
            for (k = 0; k < j; k++)
                sumx = sumx + (double)k;
            sumy = 0.0;
            for (k = j; k > 0; k--)
                sumy = sumy + (double)k;
            if (sumx > 0.0) totals[j] = totals[j] + 1.0 / sqrt(sumx);
            if (sumy > 0.0) totals[j] = totals[j] + 1.0 / sqrt(sumy);
        }
        sum = accumulate(begin(sums), end(sums), 0);
        total = accumulate(begin(totals), end(totals), 0.0);
        elapsedtime = omp_get_wtime() - starttime;
        printf("Time Elapsed: %f Secs, Total = %lf, Check Sum = %ld\n",
            elapsedtime, total, sum);
    }
    getchar();

    return 0;
}
```

۱. طبق الگوریتم زیر تعیین می‌شود:

let *ThreadsBusy* be the number of OpenMP threads currently executing in this contention group;

let *ActiveParRegions* be the number of enclosing active parallel regions;

if an `if` clause exists

then let *IfClauseValue* be the value of the `if` clause expression;

else let *IfClauseValue* = *true*;

if a `num_threads` clause exists

then let *ThreadsRequested* be the value of the `num_threads` clause expression;

else let *ThreadsRequested* = value of the first element of *nthreads-var*;

let *ThreadsAvailable* = (*thread-limit-var* - *ThreadsBusy* + 1);

if (*IfClauseValue* = *false*)

then number of threads = 1;

else if (*ActiveParRegions* = *max-active-levels-var*)

then number of threads = 1;

else if (*dyn-var* = *true*) and (*ThreadsRequested* ≤*ThreadsAvailable*)

then 1 ≤ number of threads ≤*ThreadsRequested*;

else if (*dyn-var* = *true*) and (*ThreadsRequested* > *ThreadsAvailable*)

then 1 ≤ number of threads ≤*ThreadsAvailable*;

else if (*dyn-var* = *false*) and (*ThreadsRequested* ≤*ThreadsAvailable*)

then number of threads = *ThreadsRequested*;

else if (*dyn-var* = *false*) and (*ThreadsRequested* > *ThreadsAvailable*)

then behavior is implementation defined;

که در آن nthreads-var و thread-limit-var به صورت زیر هستند:

- *nthreads-var* - controls the number of threads requested for encountered `parallel` regions. There is one copy of this ICV per data environment.
- *thread-limit-var* - controls the maximum number of threads participating in the contention group. There is one copy of this ICV per data environment.

ICV مخفف Internal Control Variable است که تعریف آن در زیر آمده‌است:

An OpenMP implementation must act as if there are internal control variables (ICVs) that control the behavior of an OpenMP program. These ICVs store information such as the number of threads to use for future `parallel` regions, the schedule to use for worksharing loops and whether nested parallelism is enabled or not. The ICVs are given values at various times (described below) during the execution of the program. They are initialized by the implementation itself and may be given values through OpenMP environment variables and through calls to OpenMP API routines. The program can retrieve the values of these ICVs only through OpenMP API routines.

مقدارهای اولیه‌ی nthreads-var و thread-limit-var از متغیرهای محیطی (Environment Variable)های زیر تعریف می‌شوند

ICV	Environment Variable	Initial value
nthreads-var	`OMP_NUM_THREADS`	Implementation defined
thread-limit-var	`OMP_THREAD_LIMIT`	Implementation defined

که OPM_NUM_THREADS به صورت زیر:

The `OMP_NUM_THREADS` environment variable sets the number of threads to use for `parallel` regions by setting the initial value of the *nthreads-var* ICV. See Section 2.5 on page 171 for a comprehensive set of rules about the interaction between the `OMP_NUM_THREADS` environment variable, the `num_threads` clause, the `omp_set_num_threads` library routine and dynamic adjustment of threads, and Section 2.6.1 on page 224 for a complete algorithm that describes how the number of threads for a `parallel` region is determined.

The value of this environment variable must be a list of positive integer values. The values of the list set the number of threads to use for `parallel` regions at the corresponding nested levels.

The behavior of the program is implementation defined if any value of the list specified in the `OMP_NUM_THREADS` environment variable leads to a number of threads that is greater than an implementation can support, or if any value is not a positive integer.

و OMP_THREAD_LIMIT به صورت زیر تعریف می‌شوند:

The `OMP_THREAD_LIMIT` environment variable sets the maximum number of OpenMP threads to use in a contention group by setting the *thread-limit-var* ICV.

The value of this environment variable must be a positive integer. The behavior of the program is implementation defined if the requested value of `OMP_THREAD_LIMIT` is greater than the number of threads an implementation can support, or if the value is not a positive integer.

۲. بله. اما تنها در صورت استفاده از intel c++ به جای Microsoft c++.

```
Serial Timings for 50000 iterations

Time Elapsed: 1.765975 Secs, Total = 30.646248, Check Sum = 50000
Time Elapsed: 1.694307 Secs, Total = 30.643539, Check Sum = 50000
Time Elapsed: 1.365507 Secs, Total = 30.619402, Check Sum = 50000
Time Elapsed: 1.326361 Secs, Total = 30.632740, Check Sum = 50000
Time Elapsed: 1.374004 Secs, Total = 30.630563, Check Sum = 50000
Time Elapsed: 1.365505 Secs, Total = 30.629100, Check Sum = 50000
Time Elapsed: 1.325508 Secs, Total = 30.628638, Check Sum = 50000
Time Elapsed: 1.355239 Secs, Total = 30.643223, Check Sum = 50000
Time Elapsed: 1.326223 Secs, Total = 30.620996, Check Sum = 50000
Time Elapsed: 1.331264 Secs, Total = 30.623719, Check Sum = 50000
```

کد این قسمت:

```c
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif // _OPENMP
#include <math.h>
#ifdef _WIN32
#include <Windows.h>
#include <intrin.h>
#else
#include <sys/time.h>
#endif
#define bool _Bool
#define true 1
#define false 0
#define VERYBIG  50000
// ********************************************************************

// Description:
// Return the current time value in seconds
// Microsoft Windows* uses QueryPerformanceFrequency/QueryPerformanceCounter for time
// Linux*/macOS* uses get_timeofday for time
double get_time() {
#ifdef _WIN32

    // Time
    unsigned __int64 m_frequency;
    // QueryPerformanceFrequency works with QueryPerformanceCounter to return a human-readable time, provided in Windows.h
    QueryPerformanceFrequency((LARGE_INTEGER*)&m_frequency);
    unsigned __int64 now;
    QueryPerformanceCounter((LARGE_INTEGER*)&now);
    // Divide the raw counter by m_frequency for time in seconds
    return ((double)(now) / m_frequency);
#else

    // Time
    struct timeval now;
    gettimeofday(&now, 0); //Returns the time of the day
    //tv_sec records time in seconds and tv_usec records time in micro seconds
    return ((double)now.tv_sec + (double)now.tv_usec / 1000000.0);
#endif
}


int main(void)
{
    int i;
    long int j, k, sum;
    double sumx, sumy, total;
    double starttime, elapsedtime;
    // ------------------------------------------------------------------------
    printf("Serial Timings for %d iterations\n\n", VERYBIG);
    for (i = 0; i < 10; i++)
    {
        starttime = get_time();
        sum = 0;
        total = 0.0;
```

ادامه در صفحه بعد:

```
#pragma omp parallel for private(k, sumx, sumy)
        for (j = 0; j < VERYBIG; j++)
        {
#pragma omp atomic write
            sum += 1;
            sumx = 0.0;
            for (k = 0; k < j; k++)
                sumx = sumx + (double)k;
            sumy = 0.0;
            for (k = j; k > 0; k--)
                sumy = sumy + (double)k;
            if (sumx > 0.0)
#pragma omp atomic write
                total = total + 1.0 / sqrt(sumx);
            if (sumy > 0.0)
#pragma omp atomic write
                total = total + 1.0 / sqrt(sumy);
        }
        elapsedtime = omp_get_wtime() - starttime;
        printf("Time Elapsed: %f Secs, Total = %lf, Check Sum = %ld\n",
            elapsedtime, total, sum);
    }
    getchar();

    return 0;
}
```

.۳

```
Time Elapsed: 14.307501 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.033001 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 13.975144 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 13.991133 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 13.989082 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.430083 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 15.171136 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.069446 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.024346 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 13.975249 Secs, Total = 32.617277, Check Sum = 100000
```

Critical default threads 10000 tries

```
Serial Timings for 100000 iterations

Time Elapsed: 14.717693 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.190356 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.194448 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.182789 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.217974 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.210109 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.200628 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.376642 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.273010 Secs, Total = 32.617277, Check Sum = 100000
Time Elapsed: 14.020645 Secs, Total = 32.617277, Check Sum = 100000
```

Reduction default threads 10000 tries

```
Serial Timings for 25000 iterations

Time Elapsed: 0.884825 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.876378 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.872861 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.876503 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.877649 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.871792 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.873223 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.873554 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.872971 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.873826 Secs, Total = 28.696202, Check Sum = 25000
```

Reduction default threads 2500 tries

```
Serial Timings for 25000 iterations

Time Elapsed: 1.040345 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.960728 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.883593 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.873892 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.877365 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.873360 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.876909 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.879549 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.872116 Secs, Total = 28.696202, Check Sum = 25000
Time Elapsed: 0.881537 Secs, Total = 28.696202, Check Sum = 25000
```

Critical default threads 2500 tries

```
Time Elapsed: 4.003345 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.504082 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.489854 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.496818 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.563282 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.605435 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.661987 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.651321 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.523239 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.573146 Secs, Total = 30.656747, Check Sum = 50000
```

Critical 4 threads 50000 tries

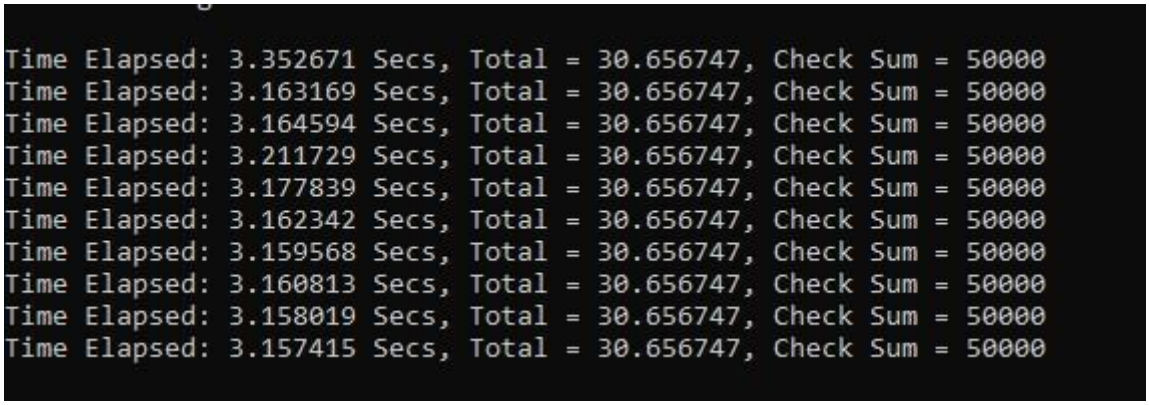
```
Serial Timings for 50000 iterations

Time Elapsed: 3.264900 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.166657 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.158110 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.158741 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.157210 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.165233 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.156358 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.159926 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.155376 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.171064 Secs, Total = 30.656747, Check Sum = 50000
```

Critical 16 threads 50000 tries

```
Time Elapsed: 3.341744 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.167833 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.156479 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.174560 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.161799 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.183875 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.192307 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.322736 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.170486 Secs, Total = 30.656747, Check Sum = 50000
Time Elapsed: 3.183351 Secs, Total = 30.656747, Check Sum = 50000
```

Reduction 16 threads 50000 tries



Reduction 4 threads 50000 tries

۲.

کد:

```
void run_serial(int* heights) {
    int* results = decompose_by_input(heights, 0, false);
    print_results(results);
}

int main()
{
    srand(time(0));
    int* heights = new int[SUPERBIG];
#pragma omp parallel for
    for (int i = 0; i < SUPERBIG; i++)
        heights[i] = rand() % limit;
#pragma omp parallel for num_threads(limitCount)
    for (int i = 0; i <= limitCount; i++)
        limits[i] = i * limitBound;

    std::cout << "Starting computation with " << threads << "threads... \n";
    double elapsedTime, startTime = omp_get_wtime();
    // Call function here:
    //run_serial(heights);
    run_input_decomposition(heights);
    //run_output_decomposition(heights);
    elapsedTime = omp_get_wtime() - startTime;
    std::cout << "Elapsed time is: " << elapsedTime << "s \n";
}
```

ادامه در ص بعد:

```cpp
#include <iostream>
#include <time.h>
#include <omp.h>

const int SUPERBIG = 800'000'000;
const int limit = 100;
const int threads = 16;
const int limitCount = 4;
const int indexBound = SUPERBIG / threads;
const int limitBound = limit / limitCount;

int limits[limitCount + 1];
int globalResults[threads] = {};

void print_results(int* results)
{
    std::cout << "The histogram is as follows: " << std::endl;
    for (int i = 0; i < limitCount; i++)
        std::cout << results[i] << std::endl;
}

int* decompose_by_input(int* heights, int firstIndex, bool isParallel=true) {
    static int results[limitCount] = {};
    int lastIndex = firstIndex + indexBound;
    if (!isParallel)
        lastIndex = SUPERBIG;

    for (int i = firstIndex; i < lastIndex; i++)
        for (int lowerBoundLimit = 0; lowerBoundLimit < limitCount; lowerBoundLimit++)
            if ((heights[i] >= limits[lowerBoundLimit]) && (heights[i] < limits[lowerBoundLimit + 1]))
                results[lowerBoundLimit]++;

    return results;
}

void decompose_by_output(int* heights, int upperBoundIndex) {
    for (int i = 0; i < SUPERBIG; i++)
        if ((heights[i] >= limits[upperBoundIndex]) && (heights[i] < limits[upperBoundIndex + 1]))
#pragma omp critical
            globalResults[upperBoundIndex]++;
}

void run_input_decomposition(int* heights) {
    int* localResults = new int[limitCount];
#pragma omp parallel for num_threads(threads) reduction(+:localResults[:limitCount])
    for (int i = 0; i < threads; i++)
        localResults = decompose_by_input(heights, i * indexBound);
    print_results(localResults);
}

void run_output_decomposition(int* heights) {
#pragma omp parallel for num_threads(limitCount)
    for (int i = 0; i < limitCount; i++)
        decompose_by_output(heights, i);
    print_results(globalResults);
}
```

نتایج به ازای تجزیه ورودی:

```
Starting computation with 2threads...
The histogram is as follows:
200191753
200194874
200016646
199596727
Elapsed time is: 6.31132s
```

```
Starting computation with 4threads...
The histogram is as follows:
200202299
200183948
200020275
199593478
Elapsed time is: 6.29672s
```

```
Starting computation with 8threads...
The histogram is as follows:
200205145
200188806
200015545
199590504
Elapsed time is: 6.32308s
```

```
Starting computation with 16threads...
The histogram is as follows:
200197460
200198125
200031215
199573200
Elapsed time is: 6.31271s
```

نتایج به ازای تجزیه خروجی؛

```
Starting computation with 2threads...
The histogram is as follows:
400392101
399607899
Elapsed time is: 6.81474s
```

```
Starting computation with 4threads...
The histogram is as follows:
200198186
200188734
200019980
199593100
Elapsed time is: 13.5536s
```

```
Starting computation with 8threads...
The histogram is as follows:
96092301
96100957
96097392
96084927
96105770
95991477
95799205
95800352
Elapsed time is: 25.4098s
```

```
Starting computation with 16threads...
The histogram is as follows:
48046125
48046295
48047469
48048256
48056323
48051773
48040244
48038209
48054883
48042932
48051102
47949677
47906084
47892725
47904472
47902754
Elapsed time is: 60.6503s
```

در موارد ورودی تعداد دسته‌ها برابر با ۴ و در خروجی‌ها برابر با میزان نخ‌ها است.

در مورد خروجی به علت کم‌بازده بودن الگوریتم با زیاد کردن تعداد نخ‌ها دچار افت کارکرد می‌شویم و در مورد خروجی از ۲ به ۴ نخ رفتن بازده را بالا می‌برد اما در ۸ نخ یک افت می‌بینیم که نشانه‌ی غالب‌شدن context switchها به کارکرد خود نخ‌ها هستیم، اما در ۱۶ نخ باز یک کاهش زمان را مشاهده می‌کنیم که می‌تواند به این علت باشد که به علت ازدیاد نخ‌های برنامه (و زیاد نشدن context switch به همان نسبت زیرا ویندوز در حالت پایه دارای ۲۰۰۰ نخ است) زمان بیش‌تری از cpu به این برنامه تسبت به دیگر برنامه‌ها تعلق می‌گیرد.

۳. دستورات به شرح زیر هستند:

۱. simd: اجرای دستورات iterationهای حلقه را به صورت موازی و با استفاده از دستورات سخت‌افزاری Single Instruction Multiple Data فراهم می‌کند. تفاوت این دستور با parallel for این است که این دستور سعی می‌کند دستورات درون یک نخ را موازی‌سازی کند و با استفاده از فرآیندهای موازی درون یکی از هسته‌های پردازنده کار خود را به صورت موازی پیش ببرد اما parallel for دستورات را به چند نخ متفاوت assign می‌کند.

۲. collapse برای افزایش تعداد iterationهایی‌ست که بین OMP Threadهای در دسترس پخش می‌شوند تا بتوان درشت‌دانگی کارهای انجام‌شده توسط نخ‌ها را کاهش داد. اگر میزان کار انجام‌شده توسط هر نخ از قبل غیر بدیهی باشد، این دستور توانایی افزایش scalability برنامه را دارد.

۳. final وقتی یک final clause بر روی ساخت task پیاده‌سازی شود و مقدار آن true باشد، task ساخته‌شده یک تسک نهایی خواهد بود. به این معنی که این task و تمامی subtaskهای آن به صورت سریال اجرا می‌شوند.

۴. taskwait یک وقفه روی اتمام child taskهای task فعلی اضافه می‌کند.

۵. omp_dynamic: یک متغیر محیطی است که تنظیم کردن تعداد نخ‌های مورداستفاده در قسمت‌های مشخص شده با کلمه‌ی parallel استفاده می‌شود.

۴. الگوریتم‌های هستند که به ازای مساله‌های «خیلی بزرگ» سریع‌تر از تمامی الگوریتم‌های دیگر اجرا می‌شوند، اما «خیلی بزرگ» به اندازه‌ای بزرگ است که در عمل هیچ‌گاه از این الگوریتم‌ها استفاده نمی‌شود.
مثال ها:

۱. سریع‌ترین روش شناخته‌شده برای ضرب ۲ عدد که مبتنی بر یک تبدیل فوریه‌ی ۱۷۲۹ بعدی است که تا وقتی عدد حداقل $2^{1729^{12}}$ بیتی نباشد به کارایی نهایی خود نمی‌رسد.

۲. بهترین حمله‌ی شناخته‌شده علیه الگوریتم رمزگذاری AES که 2^{126} operation نیاز دارد.

۳. الگوریتم Hutter که می‌تواند هر مساله‌ی درست-تعریف‌شده‌ای را در زمان اپتیمال حل کند اما چون ضرایب ثابت آن بسیار بزرگ هستند هیچ‌گاه در عمل استفاده نمی‌شود.