به نام خدا

تمرین کامپیوتری چهارم درس برنامه نویسی موازی

رستا تدین طهماسبی ۸۱۰۱۹۶۴۳۶

دیار محمدی ۸۱۰۱۹۶۵۵۳

سوال اول

در این سوال قصد داریم بزرگ‌ترین عنصر آرایه و اندیس آن را به دو روش سریال و موازی به دست آوریم و سرعت این دو روش را مقایسه کنیم.

از ساختاری مشابه ساختارهای استفاده شده در تمرین‌های قبلی برای محاسبه‌ی زمان اجرا استفاده شده است که به دلیل سادگی از توضیح خودداری می‌کنیم.

ابتدا آرایه‌ای با اندازه‌ی گفته شده از اعداد رندوم می‌سازیم.

```
float* array = new float[ARRAY_SIZE];
for (int i=0; i<ARRAY_SIZE; i++)
    array[i] = rand();
```

در روش سریال با یک حلقه‌ی ساده روی آرایه هر بار چک می‌کنیم که آیا باید مقدار بزرگ‌ترین عنصر و اندیس آن آپدیت شود یا نه.

با توجه به اینکه مقادیر رندوم غیر منفی هستند پس مقدار اولیه‌ی متغیر بزرگ‌ترین عنصر را (۱-) قرار می‌دهیم.

```
int max_id_S = -1;
float max_value_S = -1;

gettimeofday(&start1, NULL);
for (int i=0; i<ARRAY_SIZE; i++)
    if (max_value_S < array[i])
    {
        max_value_S = array[i];
        max_id_S = i;
    }
gettimeofday(&end1, NULL);
```

در روش موازی، آرایه را بین تردهایی که داریم به صورت مساوی تقسیم میکنیم که هرکدام در زیرآرایه‌ی خود مقدار بزرگترین عنصر را پیدا کنند و در نهایت با مقایسه‌ی نتایج هر ترد، مقدار بزرگ‌ترین عنصر و اندیس آن را به دست می‌آوریم.

```
   float max_value_P = -1;
   int max_id_P;
   int block_size = ARRAY_SIZE / THREADS_NUM;
   gettimeofday(&start2, NULL);
   #pragma omp parallel
   {
       int tid = omp_get_thread_num();
       int start = tid*block_size;
       int stop = min( (tid+1)*block_size, ARRAY_SIZE);
       for (int i=start; i<stop; i++)
           if (max_values_P[tid] < array[i])
           {
               max_values_P[tid] = array[i];
               max_ids_P[tid] = i;
           }
   }
   for (int i=0; i<THREADS_NUM; i++)
       if (max_value_P < max_values_P[i])
       {
           max_value_P = max_values_P[i];
           max_id_P = max_ids_P[i];
       }
   gettimeofday(&end2, NULL);
```

نتایج دو روش را مقایسه کره و زمان‌های اجرا و میزان تسریع را حساب میکنیم.

```
long seconds1 = (end1.tv_sec - start1.tv_sec);
   long micros1 = ((seconds1 * 1000000) + end1.tv_usec) -
(start1.tv_usec);

   long seconds2 = (end2.tv_sec - start2.tv_sec);
   long micros2 = ((seconds2 * 1000000) + end2.tv_usec) -
(start2.tv_usec);

   printf("Serial result: Max value = %f, Max value id = %d\n",
max_value_S, max_id_S);
   printf("Parallel result: Max value = %f, Max value id = %d\n",
max_value_P, max_id_P);
```

```
  printf ("Serial Run time = %ld \n", micros1);
  printf ("Parallel Run time = %ld \n", micros2);
  printf ("Speedup = %4.2f\n", (float) (micros1)/(float) micros2);
```

نتیجه‌ی اجرا:

```
➜ 1 make && ./main
make: Nothing to be done for 'all'.
Rasta Tadayon    810196436
Diyar Mohammadi          810196553
Serial result: Max value = 2147480064.000000, Max value id = 245298
Parallel result: Max value = 2147480064.000000, Max value id = 245298
Serial Run time = 2698
Parallel Run time = 972
Speedup = 2.78
```

سوال دوم.

در این سوال قصد داریم یک quicksort را روی یک آرایه با دو روش سریال و موازی پیاده‌سازی و اجرا کنیم.

دو آرایه با سایز داده‌شده را با مقادیر رندوم و یکسان برای هر دو آرایه می‌سازیم.

```
   struct timeval start1, end1, time1;
   struct timeval start2, end2, time2;

   float* arrayS = new float[ARRAY_SIZE];
   float* arrayP = new float[ARRAY_SIZE];
   for (int i=0; i<ARRAY_SIZE; i++)
       arrayS[i] = arrayP[i] = rand();
```

الگوریتم quicksort سریال را اجرا میکنیم. در این روش ابتدا pivot را تعیین کرده و عناصر بزرگ‌تر را سمت راست و عناصر کوچک‌تر را در سمت چپ آن قرار می‌دهیم. حال خود quicksort را روی زیر آرایه چپ و راست اجرا می‌کنیم.

```
void partition(float* array, int &i, int &j)
{
   float tmp;
   float pivot = array[(i + j) / 2];

   while (i <= j) {
       while (array[i] < pivot)
           i++;
       while (array[j] > pivot)
           j--;
       if (i <= j) {
           tmp = array[i];
           array[i] = array[j];
           array[j] = tmp;
           i++;
           j--;
       }
   }
}
```

```
void quickSortS(float* array, int left, int right)
{
   int i = left, j = right;

   partition(array, i, j);

   if (left < j)
       quickSortS(array, left, j);

   if (i< right)
       quickSortS(array, i, right);
}

   //Serial
   gettimeofday(&start1, NULL);
       quickSortS(arrayS, 0, ARRAY_SIZE-1);
   gettimeofday(&end1, NULL);
```

برای روش موازی از همان تابع partition بخش قبل استفاده میکنیم ولی پس از انجام partitioning برای اجرای quicksort روی زیرآرایه‌های راست و چپ directive مناسب قرار می‌دهیم تا به عنوان یک task برای اجرا به پردازنده سپرده شود و پردازنده در صورتی که بتواند این task ها را به صورت موازی انجام دهد.

اگر طول کل آرایه از مقدار SUB_ARRAY_LEN_CUTOFF کمتر باشد دیگر نیازی به ایجاد تسک جدید و تحمل سربارهای آن نیست و کد به صورت عادی اجرا می‌شود.

```
void quickSortP(float* array, int left, int right)
{
   int i = left, j = right;
   partition(array, i, j);
   if ( ((right-left)<SUB_ARRAY_LEN_CUTOFF) )
   {
       if (left < j){ quickSortP(array, left, j); }
       if (i < right){ quickSortP(array, i, right); }
   }
   else
```

```
    {
        #pragma omp task
        {
            quickSortP(array, left, j);
        }
        #pragma omp task
        {
            quickSortP(array, i, right);
        }
    }
}

    //Parallel
    gettimeofday(&start2, NULL);
    #pragma omp parallel num_threads(THREADS_NUM)
    {
        #pragma omp single nowait
        {
            quickSortP(arrayP, 0, ARRAY_SIZE-1);
        }
    }
    gettimeofday(&end2, NULL);
```

برای اجرای quickSortP در main هم ابتدا مشخص میکنیم که این قسمت از کد با تعداد مشخصی ترد به صورت موازی انجام شود، سپس برای خود quickSortP هم تعیین می‌کنیم که فقط توسط یکی از تردها اجرا شود (single) و به مابقی تردها اجازه می‌دهیم در حالیی که هنوز اجرای quickSortP تمام نشده، پردازش دیگر task ها را انجام دهند.(nowait)

سپس نتایج دو روش را از نظر صحت و برابری با همدیگر مقایسه می‌کنیم.

```
    for (int i=0; i<ARRAY_SIZE; i++)
    {
        if ((i+1 != ARRAY_SIZE) && (arrayS[i] > arrayS[i+1]))
        {
            valid_result = 0;
```

```
            printf("Results are not valid. arrayS[%d] = %f > arrayS[%d] =
%f\n", i, arrayS[i], i+1, arrayS[i+1]);
            break;
        }
        if (arrayS[i] != arrayP[i])
        {
            printf("arrayS[%d] = %f, arrayP[%d] = %f\n", i, arrayS[i], i,
arrayP[i]);
            same_results = 0;
        }
    }
    if (valid_result && same_results)
        printf("Results are valid and the same.\n");
```

زمان‌های اجرا و مقدار سپیدآپ را محاسبه می‌کنیم.

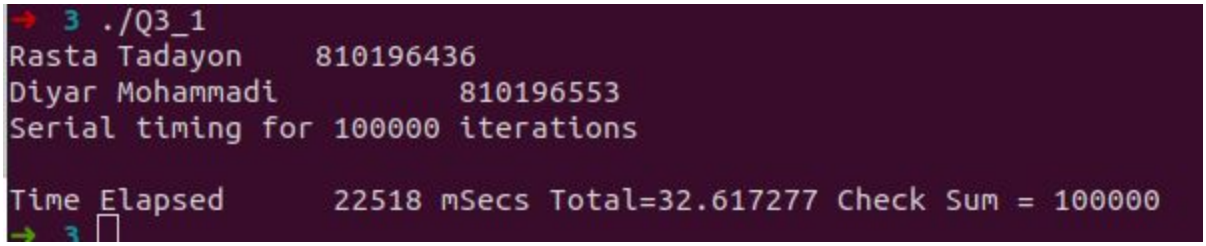
```
    long seconds1 = (end1.tv_sec - start1.tv_sec);
    long micros1 = ((seconds1 * 1000000) + end1.tv_usec) -
(start1.tv_usec);

    long seconds2 = (end2.tv_sec - start2.tv_sec);
    long micros2 = ((seconds2 * 1000000) + end2.tv_usec) -
(start2.tv_usec);

    int valid_result = 1;
    int same_results = 1;

    printf ("Serial Run time = %ld \n", micros1);
    printf ("Parallel Run time = %ld \n", micros2);
    printf ("Speedup = %4.2f\n", (float) (micros1)/(float) micros2);
```

نتیجه‌ی اجرا:

```
→ 2 nice --20 ./main
nice: cannot set niceness: Permission denied
Rasta Tadayon    810196436
Diyar Mohammadi          810196553
Results are valid and the same.
Serial Run time = 194689
Parallel Run time = 43461
Speedup = 4.48
```

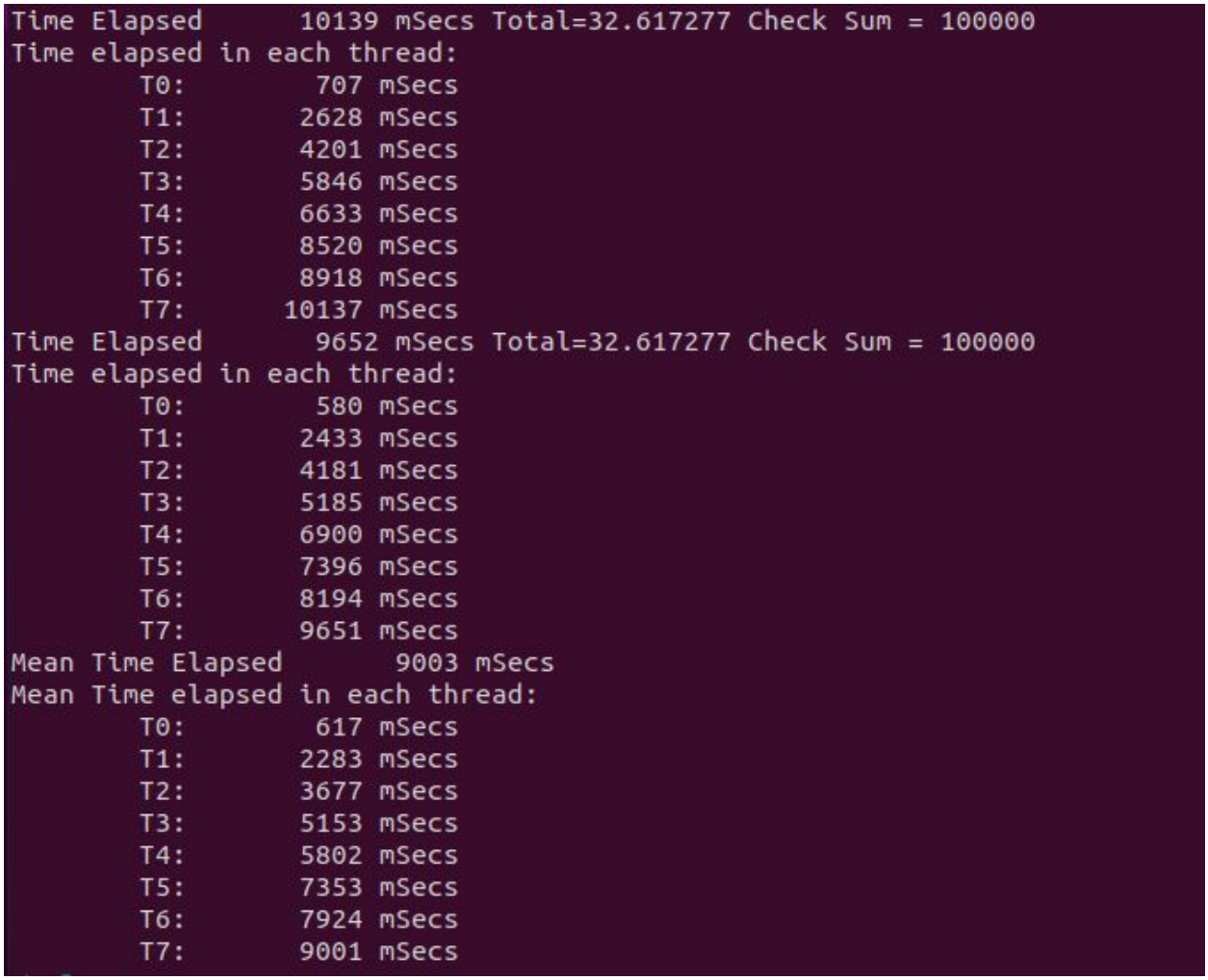
سوال سوم.

نتیجه‌ی اجرای کد سریال:



نتیجه‌ی اجرای کد موازی (static):

```
→ 3 ./Q3_2
Rasta Tadayon    810196436
Diyar Mohammadi          810196553
OpenMP Parallel Timings for 100000 iterations

Time Elapsed       6587 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:         497 mSecs
        T1:        1662 mSecs
        T2:        2612 mSecs
        T3:        3513 mSecs
        T4:        3711 mSecs
        T5:        5098 mSecs
        T6:        5728 mSecs
        T7:        6584 mSecs
Time Elapsed       7890 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:         527 mSecs
        T1:        1687 mSecs
        T2:        2831 mSecs
        T3:        3676 mSecs
        T4:        3919 mSecs
        T5:        5981 mSecs
        T6:        6406 mSecs
        T7:        7889 mSecs
Time Elapsed      10075 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:         724 mSecs
        T1:        2751 mSecs
        T2:        4391 mSecs
        T3:        6510 mSecs
        T4:        6837 mSecs
        T5:        8695 mSecs
        T6:        9465 mSecs
        T7:       10073 mSecs
Time Elapsed       9671 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:         669 mSecs
        T1:        2538 mSecs
        T2:        3846 mSecs
        T3:        6187 mSecs
        T4:        6814 mSecs
        T5:        8428 mSecs
        T6:        8831 mSecs
        T7:        9670 mSecs
```



زمان اجرای میانگین: 9003 mSec میزان سپیدآپ: 2.50

توزیع بار روی ترد ها به خوبی انجام نشده و زمان اجرای تردها بسیار اختلاف دارد و همین باعث افزایش زمان اجرا شده است.

نتیجه‌ی اجرای کد موازی (dynamic, 1000);

```
→ 3 ./Q3_2
Rasta Tadayon    810196436
Diyar Mohammadi          810196553
OpenMP Parallel Timings for 100000 iterations

Time Elapsed       4997 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       4943 mSecs
        T1:       4794 mSecs
        T2:       4995 mSecs
        T3:       4657 mSecs
        T4:       4801 mSecs
        T5:       4526 mSecs
        T6:       4669 mSecs
        T7:       4742 mSecs
Time Elapsed       5987 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       5402 mSecs
        T1:       5109 mSecs
        T2:       5097 mSecs
        T3:       5985 mSecs
        T4:       5526 mSecs
        T5:       5789 mSecs
        T6:       5582 mSecs
        T7:       5741 mSecs
Time Elapsed       8765 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       7879 mSecs
        T1:       8349 mSecs
        T2:       8476 mSecs
        T3:       7940 mSecs
        T4:       8656 mSecs
        T5:       8447 mSecs
        T6:       8763 mSecs
        T7:       7966 mSecs
Time Elapsed       8372 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       7486 mSecs
        T1:       7650 mSecs
        T2:       8234 mSecs
        T3:       8254 mSecs
        T4:       8243 mSecs
        T5:       8365 mSecs
        T6:       8332 mSecs
        T7:       7757 mSecs
```

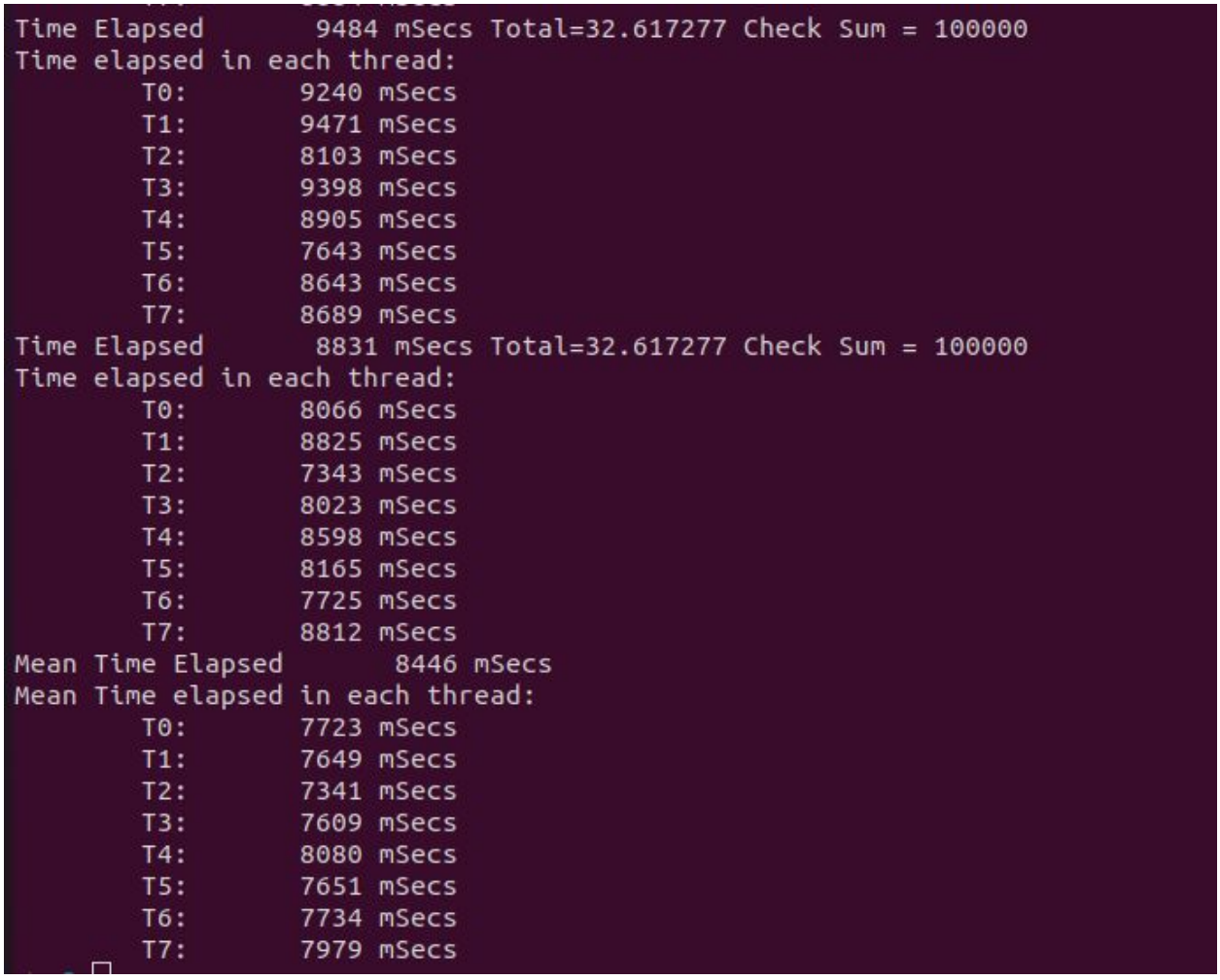
```
Time Elapsed       8357 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       7833 mSecs
        T1:       8344 mSecs
        T2:       8355 mSecs
        T3:       8302 mSecs
        T4:       8211 mSecs
        T5:       7495 mSecs
        T6:       7980 mSecs
        T7:       7737 mSecs
Time Elapsed       8413 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       8037 mSecs
        T1:       7850 mSecs
        T2:       8189 mSecs
        T3:       7537 mSecs
        T4:       7762 mSecs
        T5:       7915 mSecs
        T6:       8411 mSecs
        T7:       7832 mSecs
Mean Time Elapsed       7482 mSecs
Mean Time elapsed in each thread:
        T0:       6930 mSecs
        T1:       7016 mSecs
        T2:       7224 mSecs
        T3:       7113 mSecs
        T4:       7200 mSecs
        T5:       7089 mSecs
        T6:       7290 mSecs
        T7:       6963 mSecs
```

زمان اجرای میانگین: 7482 mSec میزان سپیدآپ: 3

زمان اجرای تردها تقریبا مساوی است که یعنی توزیع بار نسبت به حالت قبل بهتر صورت گرفته است.

نتیجه‌ی اجرای کد موازی (dynamic, 2000);

```
➜ 3 ./Q3_2
Rasta Tadayon    810196436
Diyar Mohammadi          810196553
OpenMP Parallel Timings for 100000 iterations

Time Elapsed       5355 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       5153 mSecs
        T1:       4637 mSecs
        T2:       4896 mSecs
        T3:       4228 mSecs
        T4:       5346 mSecs
        T5:       4258 mSecs
        T6:       4487 mSecs
        T7:       5116 mSecs
Time Elapsed       7451 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       6414 mSecs
        T1:       5895 mSecs
        T2:       6687 mSecs
        T3:       6629 mSecs
        T4:       7449 mSecs
        T5:       7122 mSecs
        T6:       7342 mSecs
        T7:       5766 mSecs
Time Elapsed      10607 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       9407 mSecs
        T1:       9232 mSecs
        T2:       8825 mSecs
        T3:      10190 mSecs
        T4:      10604 mSecs
        T5:       9772 mSecs
        T6:       9646 mSecs
        T7:      10604 mSecs
Time Elapsed       8948 mSecs Total=32.617277 Check Sum = 100000
Time elapsed in each thread:
        T0:       8057 mSecs
        T1:       7836 mSecs
        T2:       8192 mSecs
        T3:       7188 mSecs
        T4:       7579 mSecs
        T5:       8946 mSecs
        T6:       8559 mSecs
        T7:       8884 mSecs
```



زمان اجرای میانگین: 8446 mSec میزان سپیدآپ: 2.66

زمان اجرای تردها تقریبا مساوی است که یعنی توزیع بار به خوبی صورت گرفته است.