پروژه ۴

حانیه ناصری – ثمین مهدی زاده

سوال اول)

در ابتدا با srand ، آرایه ای از اعداد float با سایز گفته شده را مقدار دهی می کنیم.

ابتدا پیاده سازی **موازی** با OpenMp را توضیح می دهیم.

در تابع omp_parallel_max، متغیر های max_val، maxs، indexes و max_index را به ترتیب برای نگه داری عدد ماکسیمم کل، عدد ماکسیمم به دست آمده توسط هر نخ، اندیس ماکسیمم به دست آمده توسط هر نخ و اندیس ماکسیمم کل تعریف می کنیم. متغیر i را هم تعریف می کنیم تا در ادامه شمارنده حلقه مربوط به یافتن ماکسیمم باشد (همه متغیر ها از جمله ورودی های تابع که آرایه و انتها و ابتدای آن هستند به صورت shared و فقط متغیر شمارنده حلقه private خواهد بود)

سپس بدنه ی ناحیه omp parallel را با دستور#pragma omp parallel شروع می کنیم و default(shared) را به عنوان clause در کنار آن به کار می بریم (تعداد thread ها همان مقدار پیش فرض، یعنی ۴ است). در اینجا نخ اصلی به ۳ نخ جدید می پیوندد و ۴ نخ به موازات هم وارد بدنه parallel می شوند.

پس از آن عملیات یافتن ماکسیمم در آرایه را شروع می کنیم. برای یافتن ماکسیمم، به یک حلقه که شمارنده آن از عنصر اول آرایه تا آخر آن حرکت می کند نیاز داریم ولی می خواهیم که این حلقه به نخ ها سپرده شود تا هر یک با برداشتن سهمی از iteration های آن، ماکسیمم مربوط به سهم خود را پیدا کنند (در اینجا چون متغیر شمارنده یعنی i برای هر نخ مقادیر خاصی می پذیرد، پس private تعریف می شود). پس حلقه را با دستور

#pragma omp for private(i) nowait شروع می کنیم. همچنین clause nowait را هم تعریف کردیم، چرا که با خاتمه این حلقه، بدنه parallel هم خاتمه میابد و نیازی به دو barrier پشت سر هم نیست.

در حلقه for و برای هر ترد، به ازای هر عنصر آرایه چک می کنیم که اگر از مقدار کنونی ماکسیمم یافته شده توسط آن نخ بیشتر بود، ماکسیمم آن نخ را آپدیت می کنیم و indexes مربوط به آن نخ را نیز با مقدار ایندکس عنصر یافته شده آپدیت می کنیم (توجه شود که ماکسیمم هر نخ و ایندکس مربوط به آن به ترتیب در maxs[threadNum] و indexes[threadNum] نگه داشته می شوند که threadNum همان شماره نخ و برابر با omp_get_thread_num() می باشد)

در نهایت بعد از حلقه، بدنه parallel را نیز می بندیم تا همه نخ ها در barrier بعد آن join شده و نخ اصلی مسیر را ادامه دهد.

در نهایت، نخ اصلی از بین ماکسیمم های نخ ها که در maxs ذخیره شده اند، بیشترین را انتخاب می کند و اندیس آن را که در indexes متناظر می باشد، به عنوان دو جواب نهایی بر میگرداند.

در نسخه **سریال**، کل حلقه ماکسیمم توسط همان نخ اصلی اجرا کننده برنامه بررسی و اجرا می شود و طبق روال عادی ماکسیمم و ایندکس آن محاسبه می شوند.

تصویر اجرای برنامه را در زیر مشاهده می کنید:

```
mac@macs-MacBook-Pro parallel_4 %  g++-10 -fopenmp Q1.cpp -o Q1
mac@macs-MacBook-Pro parallel_4 % ./Q1
810196573-810196623

99.999954
472768
OMP PARALLEL Execution time is 0 s and 2924 micros

99.999954
472768
SERIAL Execution time is 0 s and 5835 micros

improvement : 1.995554
mac@macs-MacBook-Pro parallel_4 % ./Q1
810196573-810196623

99.999870
997128
OMP PARALLEL Execution time is 0 s and 1672 micros

99.999870
997128
SERIAL Execution time is 0 s and 3828 micros

improvement : 2.289474
mac@macs-MacBook-Pro parallel_4 % ./Q1
810196573-810196623

99.999924
260695
OMP PARALLEL Execution time is 0 s and 1437 micros

99.999924
260695
SERIAL Execution time is 0 s and 2973 micros

improvement : 2.068893
mac@macs-MacBook-Pro parallel_4 % ./Q1
810196573-810196623

99.999771
593186
OMP PARALLEL Execution time is 0 s and 1920 micros

99.999771
593186
SERIAL Execution time is 0 s and 3501 micros

improvement : 1.823437
mac@macs-MacBook-Pro parallel_4 % ./Q1
810196573-810196623

99.999878
754135
OMP PARALLEL Execution time is 0 s and 2065 micros

99.999878
754135
SERIAL Execution time is 0 s and 3278 micros

improvement : 1.587409
mac@macs-MacBook-Pro parallel_4 %
```

سوال دوم)

ابتدا الگوریتم Quick Sort را مختصرا توضیح می دهیم. در مرتب سازی quick sort، از بین اعضای آرایه، یک عضو را به عنوان pivot در نظر میگیریم. بدین صورت که از میان اعضای آرایه، یک عضو را در نظر گرفته و عضو های کوچکتر از آن را سمت چپ آن و عضو های بزرگتر را در سمت راست آن قرار می دهیم (در اینجا pivot را برابر با عنصر آخر ارایه (اندیس آخر) در نظر می گیریم). حال برای هر کدام از دو تکه ایجاد شده (تکه آرایه قبل pivot و تکه آرایه بعد آن) عمل فوق را به صورت بازگشتی انجام می دهیم. در نهایت اگر آرایه اولیه را به عنوان ریشه درخت در نظر بگیریم، در هر مرحله بر حسب pivot دو بچه تولید می شوند و... که ترتیب چپ به راست برگ ها، آرایه مرتب شده صعودی می باشد.

ابتدا نسخه **سریال** برنامه را مورد بررسی قرار می دهیم

یک تابع به نام quicksort تعریف می کنیم که آرایه و اندیس شروع و پایان قسمت مورد بررسی آن را دریافت می کند. سپس بررسی می کند و در صورتی که اندیس شروع از اندیس پایان کوچکتر بود، تابع partition را صدا می زند. در داخل تابع partition، عضو آخر به عنوان pivot انتخاب شده و بعد در یک حلقه for، روی همه عناصر حر کت کرده و هر کدام از pivot کوچک تر مساوی بود را به اول آرایه می بریم (برای اینکه بدانیم آن را کجا قرار دهیم، یک شمارنده داریم که از اندیس خانه اول منهای ۱ شروع می شود و هر گاه عضوی کوچک تر از pivot انتخاب شد، آن را یکی زیاده کرده و عضو یافته شده را با عضو آرایه با اندیس آن شماره، swap می کنیم)

در نهایت آرایه به گونه ای مرتب شده که عناصر کوچک تر از pivot حتما در قبل آن هستند و عناصر بزرگتر آن نیز در بعد آن. به عنوان خروجی تابع، اندیس نهایی pivot بعد از این تغییر ساختار را بر میگردانیم.

حال در ادامه تابع quicksort، قسمت قبل اندیس pivot را یک تکه جدا گرفته و برای آن quicksort را بازگشتی صدا می زنیم، همین طور برای تکه بعد اندیس pivot.

چون شرط اجرای بدنه تابع کمتر بودن اندیس شروع از اندیس پایان می باشد، در نهایت که به یک آرایه تک عنصری برسیم، به معنای خاتمه می باشد (ارایه تک عنصری نیاز به partition ندارد).

بعد از اتمام تابع quicksort ، آرایه را چاپ می کنیم (توجه شود که در تمام مراحل آدرس شروع آرایه به توابع پاس داده شده و بنابر این تغییرات ایجاد شده در تابع partition و ... خود محتوای آرایه در حافظه را تغییر می دهند و کپی از آن ساخته نمی شود)

حال نسخه **موازی** را بررسی می کنیم. در اینجا ابتدا به جای آنکه از ابتدا تابع quicksort را بر روی کل آرایه صدا بزنیم، ابتدا ۷ بار partition روی آن صدا میزنیم تا در هر بار یک pivot انتخاب شود و عناصر نسبت به آن در قبل یا بعد آن قرار بگیرند. پس آرایه ۸ قسمت می شود که در هر قسمت، عناصر از pivot قبل خود بزرگتر و از pivot بعد خود کوچکتر هستند (فقط توجه کنید که ممکن است در اجرای یک partition اندیس pivot بعد از تغییر ساختار آرایه جایی قرار بگیرد که تکه ای بعد آن یا قبل آن نباشد و بنابر این ۴ partition بعد آن تحت تاثیر قرار می گیرند که برای هندل کردن این موضوع، ابتدا اندیس تمام ۷ pivot را ۱- در نظر گرفتیم و بعد اگر برای partition. مقدار اندیس پایین کمتر از اندیس بالای تکه بود و pivot قبل یا بعد ۱- نبودند، partition را بر روی آن تکه صدا زده و اندیسpivot را آپدیت می کنیم. حال هر یک از ۸ قسمت را به ۸ نخ می دهیم تا به موازات هم quicksort را روی آن اجرا کنند (با دستورات (۸)#pragma omp parallel sections num_threads و #pragma omp section)

در نهایت همه نخ ها در barrier بعد ناحیه parallel جوین شده و آرایه مرتب شده را چاپ می کنیم.

تصویر اجرای برنامه را در زیر مشاهده می کنید:

```
mac@macs-MacBook-Pro parallel_4 %  g++-10 -fopenmp Q2.cpp -o Q2
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 122381 micros

SERIAL Execution time is 0 s and 230121 micros

improvement : 1.880365
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 1 s and 122856 micros

SERIAL Execution time is 0 s and 224457 micros

improvement : 1.812242
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 132682 micros

SERIAL Execution time is 1 s and 244161 micros

improvement : 1.847734
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 171224 micros

SERIAL Execution time is 0 s and 232397 micros

improvement : 1.357269
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 138820 micros

SERIAL Execution time is 0 s and 231090 micros

improvement : 1.664674
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 123713 micros

SERIAL Execution time is 0 s and 225582 micros

improvement : 1.823430
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 154400 micros

SERIAL Execution time is 0 s and 224559 micros

improvement : 1.454398
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 1 s and 182151 micros

SERIAL Execution time is 0 s and 241896 micros

improvement : 1.320746
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 139517 micros

SERIAL Execution time is 1 s and 228843 micros

improvement : 1.647419
mac@macs-MacBook-Pro parallel_4 % ./Q2
OMP PARALLEL Execution time is 0 s and 136297 micros

SERIAL Execution time is 0 s and 227031 micros

improvement : 1.665708
mac@macs-MacBook-Pro parallel_4 %
```

سوال ۳)

در این سوال سعی شده تا اجرای سریال یک برنامه با اجرای موازی آن با سه زمان بندی static,dynamic_100,dynamic_2000 مقایسه شود. به این منظور دو کد سریال و موازی داده شده که در کد موازی با وجود آوردن تغییراتی آن را در سه زمان بندی بالا با کد سریال مقایسه کردیم.

تغییرات:

۱)محاسبه میانگین زمان اجرا:

برای محاسبه ی میانگین زمان اجرا متغیر mean_executiom_time تعریف شده که در پایان بدنه موازی و پس از هر اجرا مدت زمان اجرای برنامه به آن اضافه می شود در انتها نیز پس از تمام شدن هر ۶ اجرا برای به دست آوردن میانگین مقدار گفته شده را بر ۶ تقسیم کرده و به عنوان مقدار میانگین گزارش می دهیم.

۲)محاسبه ی زمان اجرای هر ریسمان:

از آن جایی که تعداد ریسمان های ایجاد شده برابر ۴ است دو آرایه ۴ تایی thread_start و thread_time برای نگه داری زمان شروع هر ریسمان در هر بار اجرا و مدت زمان اجرای ریسمان تعریف شده اند. در ابتدای ساختار موازی زمان شروع هر ریسمان در عنصر آرایه باتوجه به شماره ریسمان ذخیره می شود(برای مثال اگر شماره ریسمان ۲ باشد زمان شروع آن در thread_start[2] ریخته می شود)و در انتهای ساختار for مدت زمان اجرای آن ریسمان محاسبه می شود باید توجه داشت که در انتهای ساختار for یک barrier وجود دارد و همین موضوع باعث می شود تا در صورت اتمام کار یک ریسمان در همان جا منتظر بماند تا کار بقیه به پایان برسد در این صورت زمان اجرای ریسمان به درستی گزارش نمی شود جهت حل این مشکل از nowaitاستفاده شده است تا یک ریسمان در صورت اتمام کار زمان خود را ثبت کند و در انتهای بدنه ی موازی در انتظار سایر ریسمان ها بماند.همچنین جهت ثبت زمان اولیه لازم بود که ساختار for به صورت جداگانه نوشته شود.

اجرای کد:

نتایج حاصل از اجرای کد به صورت زیر است:

-سریال:

```
[mac@macs-MacBook-Pro p4 %  g++-10 -fopenmp question3_1.cpp -o 3_1_serial
[mac@macs-MacBook-Pro p4 % ./3_1_serial
Serial timing for 100000 iterations

Time Elapsed      27700 mSecs Total=32.617277 Check Sum = 100000
mac@macs-MacBook-Pro p4 %
```

- موازی(static):

```
mac@macs-MacBook-Pro p4 %  g++-10 -fopenmp question3_2_static.cpp -o 3_2_static
mac@macs-MacBook-Pro p4 % ./3_2_static
OpenMP Parallel Timings for 100000 iterations

810196623 - 810196573

Time Elapsed      13966 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       2260 mSec
T1 excution time:       6858 mSec
T2 excution time:      10467 mSec
T3 excution time:      13965 mSec
Time Elapsed      13858 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       2197 mSec
T1 excution time:       6767 mSec
T2 excution time:      10395 mSec
T3 excution time:      13857 mSec
Time Elapsed      14046 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       2211 mSec
T1 excution time:       6679 mSec
T2 excution time:      10374 mSec
T3 excution time:      14046 mSec
Time Elapsed      14337 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       2433 mSec
T1 excution time:       7110 mSec
T2 excution time:      10858 mSec
T3 excution time:      14337 mSec
Time Elapsed      14106 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       2225 mSec
T1 excution time:       6779 mSec
T2 excution time:      10537 mSec
T3 excution time:      14106 mSec
Time Elapsed      14415 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       2386 mSec
T1 excution time:       7114 mSec
T2 excution time:      10844 mSec
T3 excution time:      14415 mSec
mean execution time:      14121 mSecs
mac@macs-MacBook-Pro p4 %
```

Speed up = $\frac{27700}{14121}$ = 1.54

- موازی(dynamic_1000):

```
mac@macs-MacBook-Pro p4 %  g++-10 -fopenmp question3_2_dynamic_1000.cpp -o 3_2_dynamic_1000
mac@macs-MacBook-Pro p4 % ./3_2_dynamic_1000
OpenMP Parallel Timings for 100000 iterations

810196623 - 810196573

Time Elapsed       9292 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9094 mSec
T1 excution time:       9124 mSec
T2 excution time:       9291 mSec
T3 excution time:       8945 mSec
Time Elapsed       9289 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       8841 mSec
T1 excution time:       8977 mSec
T2 excution time:       9289 mSec
T3 excution time:       9147 mSec
Time Elapsed       9329 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       8757 mSec
T1 excution time:       8962 mSec
T2 excution time:       9329 mSec
T3 excution time:       9111 mSec
Time Elapsed       9308 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       8733 mSec
T1 excution time:       9139 mSec
T2 excution time:       8992 mSec
T3 excution time:       9308 mSec
Time Elapsed       9294 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9294 mSec
T1 excution time:       8858 mSec
T2 excution time:       9225 mSec
T3 excution time:       9003 mSec
Time Elapsed       9440 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9440 mSec
T1 excution time:       9196 mSec
T2 excution time:       8955 mSec
T3 excution time:       8863 mSec
mean execution time:       9325 mSecs
mac@macs-MacBook-Pro p4 %
```

Speed up = $\frac{27700}{9325}$ = 2.97

- موازی(dynamic_2000):

```
mac@macs-MacBook-Pro p4 %  g++-10 -fopenmp question3_2_dynamic_2000.cpp -o 3_2_dynamic_2000
mac@macs-MacBook-Pro p4 % ./3_2_dynamic_2000
OpenMP Parallel Timings for 100000 iterations

810196623 - 810196573

Time Elapsed       9534 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9533 mSec
T1 excution time:       8532 mSec
T2 excution time:       9251 mSec
T3 excution time:       8891 mSec
Time Elapsed       9455 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9162 mSec
T1 excution time:       9455 mSec
T2 excution time:       8476 mSec
T3 excution time:       8922 mSec
Time Elapsed       9392 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       8834 mSec
T1 excution time:       8608 mSec
T2 excution time:       9392 mSec
T3 excution time:       9176 mSec
Time Elapsed       9433 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9175 mSec
T1 excution time:       9433 mSec
T2 excution time:       8530 mSec
T3 excution time:       8902 mSec
Time Elapsed       9622 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9082 mSec
T1 excution time:       9204 mSec
T2 excution time:       9622 mSec
T3 excution time:       8820 mSec
Time Elapsed       9355 mSecs Total=32.617277 Check Sum = 100000
T0 excution time:       9106 mSec
T1 excution time:       9036 mSec
T2 excution time:       8530 mSec
T3 excution time:       9355 mSec
mean execution time:       9465 mSecs
mac@macs-MacBook-Pro p4 %
```

Speed up = $\frac{27700}{9465}$ = 2.92

تحلیل نتایج:

در کد، حلقه ای به شکل زیر وجود دارد:

```
for( int j=0; j<VERYBIG; j++ )
      {
        // increment check sum
        sum += 1;

        // Calculate first arithmetic series
        sumx = 0.0;
        for( k=0; k<j; k++ )
         sumx = sumx + (double)k;

        // Calculate second arithmetic series
        sumy = 0.0;
        for( k=j; k>0; k-- )
         sumy = sumy + (double)k;

        if( sumx > 0.0 )total = total + 1.0 / sqrt( sumx );
        if( sumy > 0.0 )total = total + 1.0 / sqrt( sumy );
      }
```

در این حلقه دو حلقه دیگر وجود دارد که به اندازه j تکرار می شوند بنابراین هر چه j بزرگتر می شود تعداد تکرار این دو حلقه نیز بیشتر است.اگر این کد را با زمان بندی static اجرا کنیم مقدار j بر۴ تقسیم می شود و هر کدام از ریسمان ها بخش مربوط به خود را انجام می دهند اما ریسمان هایی که در j های بالاتر باید کار را انجام دهند چون j بزرگتری دارند تعداد تکرار حلقه های درونی آن ها بیشتر می شود و در نتیجه به زمان بیشتری احتیاج دارند در حالی که ممکن است ریسمان هایی با j کوچک تر بی کار منتظر باشند بنابراین زمان اجرای کل در این زمان بندی از بقیه بیشتر است.در زمان بندی dynamic,1000 هر ریسمان در ابتدا ۱۰۰۰ تکرار از حلقه بیرونی(j) را انجام می دهد و در صورت تمام کردن کار دوباره ۱۰۰۰تکرار دیگر به آن تعلق می گیرد پس بنابراین در زمان بندی dynamic مشکل بیکار ماندن یک ریسمان به وجود نمی آید و توازن بار توزیع شده بین ریسمان ها بیشتر است بنابراین زمان اجرای کل از static بهتر است این زمان بندی از dynamic,2000 نیز بهتر عمل می کند چرا که تعداد تکرار حلقه های درونی نسبت به حالت ۲۰۰۰ اختلاف کمتری دارند برای مثال در اختصاص اول و در آخرین مرحله در حالت ۲۰۰۰ حلقه ی درونی از ۰ تا ۱۹۹۹ کار انجام می دهد درحالی که در حالت ۱۰۰۰ حلقه ی درونی از ۰ تا ۹۹۹ کار انجام می دهد.