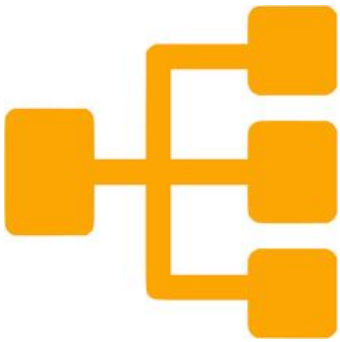

CA4 - OpenMP

Parallel Programming

امید بداقی 810196423

مبینا شاه بنده 810196488

پاییز 99

دکتر صفری

مقدمه

در این پروژه سه بخش با سه هدف متفاوت داریم که حالت موازی آنها با استفاده از **OpenMP** پیاده سازی شده است. در قسمت اول هدف بدست آوردن بزرگ‌ترین عنصر یک آرایه به همراه اندیس آن است. در قسمت دوم هدف مرتب سازی آرایه به صورت صعودی و در قسمت سوم هدف بررسی و مقایسه ی دو پیاده سازی موازی و سری یک کد است.

بدست آوردن بزرگ ترین عنصر آرایه

ابتدا آرایه ای با 2^{20} عدد ممیز شناور رندوم می‌سازیم. با استفاده از `omp_get_num_procs` تعداد هسته‌های پردازنده را بدست می‌آوریم. به اندازه نصف آن(تعداد هسته‌های فیزیکی)، ترد برای اجرای موازی برنامه در نظر می‌گیریم. حال برنامه را یکبار به صورت سریال و یکبار به صورت موازی اجرا می‌کنیم و نتایج را ثبت می‌کنیم. برای محاسبه‌ی زمان اجرا در هر دو حالت، از `omp_get_wtime()` در ابتدا و انتهای اجرا، استفاده می‌کنیم.

پیاده سازی سری

یک متغیر برای مقدار بیشینه و یک متغیر برای اندیس آن در نظر می‌گیریم. تمام اعضای آرایه را بررسی می‌کنیم و هربار، مقدار آن از مقدار ذخیره‌شده بیشتر بود، مقدار ذخیره‌شده و اندیس را بروزرسانی می‌کنیم.

پیاده سازی موازی

به اندازه مشخص‌شده ترد می‌سازیم و آرایه مقادیر را به صورت **shared** بین آنها قرار می‌دهیم. با توجه به **omp for**، هر ترد قسمتی(قسمت‌های مساوی) از اجرای حلقه را به عهده می‌گیرد و در قسمت اختصاص داده‌شده، مقدار بیشینه و اندیس آن را پیدا می‌کند. سپس در خطوط ۶۶ الی ۶۹، هر ترد مقدار بیشینه اصلی را در صورتی که مقدار بیشینه خود آن ترد بیشتر باشد، تغییر می‌دهد. همچنین با توجه به عدم وابستگی تردها به یکدیگر در حلقه **for**، در انتهای آن از **nowait** استفاده شده‌است تا هر ترد به محض اتمام کار خود، مقدار بیشینه اصلی و اندیس را در صورت امکان تغییر دهد و منتظر اتمام حلقه سایر تردها نماند و اجرا کمی سریعتر شود.

میزان تسریع

همانطور که مشاهده می‌شود، **speedup** ای حدود ۳ تا ۴ به دست می‌آید و مقادیر در روش سریال و موازی، با

```
[Omids-MacBook-Pro:final omid$ clang++ -Xpreprocessor -fopenmp q1.cpp -o q1 -lomp
[Omids-MacBook-Pro:final omid$ ./q1
 Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

 Serial Result
 maxVal=517246  index=591027
 Parallel Result
 maxVal=517246  index=591027

 Serial Exec Time=0.00247002
 Parallel Exec Time=0.000678062
 Speedup=3.64276

[Omids-MacBook-Pro:final omid$ ./q1
 Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

 Serial Result
 maxVal=730775  index=752220
 Parallel Result
 maxVal=730775  index=752220

 Serial Exec Time=0.00248504
 Parallel Exec Time=0.000825167
 Speedup=3.01156

[Omids-MacBook-Pro:final omid$ ./q1
 Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

 Serial Result
 maxVal=308253  index=718727
 Parallel Result
 maxVal=308253  index=718727

 Serial Exec Time=0.00248098
 Parallel Exec Time=0.000736952
 Speedup=3.36655

[Omids-MacBook-Pro:final omid$ ./q1
 Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

 Serial Result
 maxVal=3.71284e+06  index=642910
 Parallel Result
 maxVal=3.71284e+06  index=642910

 Serial Exec Time=0.00244808
 Parallel Exec Time=0.000725985
 Speedup=3.37209

[Omids-MacBook-Pro:final omid$ ./q1
 Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

 Serial Result
 maxVal=384907  index=754489
 Parallel Result
 maxVal=384907  index=754489

 Serial Exec Time=0.00247598
 Parallel Exec Time=0.000675917
 Speedup=3.66314

[Omids-MacBook-Pro:final omid$ ./q1
 Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

 Serial Result
 maxVal=357142  index=644797
 Parallel Result
 maxVal=357142  index=644797

 Serial Exec Time=0.00269079
 Parallel Exec Time=0.00084877
 Speedup=3.17022

 Omids-MacBook-Pro:final omid$
```

یکدیگر برابر می‌باشند.

نتایج

برای چک کردن درستی نتایج، از تابع **checkResults** استفاده شده‌است که برابری مقادیر سریال و موازی، درست بودن مقادیر در اندیس‌های متناظر و بیشینه بودن عدد در آرایه را بررسی می‌کند و در صورت نقض هریک از موارد گفته‌شده، در ترمینال، ارور چاپ می‌کند.

مرتب سازی آرایه در حالت صعودی

ابتدا آرایه ای با 2^{20} عدد ممیز شناور رندوم می‌سازیم. با استفاده از `omp_get_num_procs`، تعداد هسته‌های پردازنده را بدست می‌آوریم. به اندازه نصف آن(تعداد هسته‌های فیزیکی)، ترد برای اجرای موازی برنامه در نظر می‌گیریم. حال برنامه را یکبار به صورت سریال و یکبار به صورت موازی اجرا می‌کنیم و نتایج را ثبت می‌کنیم. برای محاسبه‌ی زمان اجرا در هر دو حالت، از `omp_get_wtime()` در ابتدا و انتهای اجرا، استفاده می‌کنیم.

پیاده سازی سری

هربار یک عنصر(در روش پیاده‌سازی‌شده عنصر آخر آرایه) را به عنوان محور در نظر می‌گیریم. اعداد بزرگتر از آن را سمت راست و اعداد کوچکتر را سمت چپ آن قرار می‌دهیم. حال سمت راست و سمت چپ را مستقلا سورت می‌کنیم و نتیجه، یک آرایه‌ی مرتب‌شده خواهد بود.

پیاده سازی موازی

به طور کلی، در موازی سازی برنامه‌های بازگشتی، بهتر است به جای **section**، از **task** استفاده‌شود. به طور کلی در **sections**، هر ترد مسئول اجرای کامل بخش خود می‌باشد و ممکن است بخش‌های مختلف، لود کاری متفاوتی داشته‌باشند که این ایده‌ال نمی‌باشد. بنابراین برای بهبود عملکرد، می‌توان از **task** استفاده‌کرد. در اینجا، **task** ها در یک صف قرار می‌گیرند و هربار، یک ترد یکی از آنها را برداشته و آن را اجرا می‌کند. اینگونه، مشکل قسمت قبل رخ نخواهد داد. ابتدا در تابع **sortParallel**، تعدادی ترد ساخته می‌شود و یکی از آن تردها، تابع **sortParallelUtil** را اجرا می‌کند. این تابع، ابتدا **partitioning** را انجام می‌دهد و سپس، برای سورت‌کردن دو سمت چپ و راست، دو **task** جدید تعریف می‌کند. آرایه، در تمام مراحل به صورت **shared** می‌باشد. عبارت `#pragma omp taskwait` بسیار شبیه **barrier** اما برای **task** ها می‌باشد. در واقع اطمینان حاصل می‌کند که روند کنونی اجرا، تا زمان اجرای تمام **task** های در نوبت، متوقف می‌شود. در

واقع، هربار اجرای **sortParallelUtil** دو **task** مختلف می‌سازد و سپس در پایان صبر می‌کنیم تا اجرای آنها به اتمام برسد. دلیل استفاده از **single** در **sortParallel**، این است که **task** ها توسط یک ترد ساخته‌شوند. در صورتی که از این عبارت استفاده نمی‌کردیم، هر **task** به اندازه‌ی تعداد تردها ساخته می‌شد که این مناسب نمی‌باشد. استفاده از `final(cond)` به این معنا می‌باشد که زمانی که **cond=True**، دیگر ساخت **task** های جدید از دل آن **task** متوقف شود. در خطوط ۷۲ و ۷۴، از این عبارت استفاده شده‌است و زمانی که طول آرایه برای **sort**، از ۱۰۰ کمتر باشد، دیگر **task** های جدید ساخته نمی‌شوند و آن آرایه به صورت سریال مرتب می‌شود. دلیل آن این است که هزینه شکستن یک آرایه‌ی کوچک به چندین **task** و مرتب‌سازی آن به صورت موازی، می‌تواند سربار بیشتری نسبت به اجرای سریال آن داشته‌باشد.

نتایج

```
Omids-MacBook-Pro:final omid$ clang++ -Xpreprocessor -fopenmp q2.cpp -o q2 -lomp
Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.161881
Parallel Exec Time=0.0567911
Speedup=2.85047

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.164727
Parallel Exec Time=0.0531819
Speedup=3.09743

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.168242
Parallel Exec Time=0.0586672
Speedup=2.86774

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.159177
Parallel Exec Time=0.0562391
Speedup=2.83036

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.16008
Parallel Exec Time=0.0511529
Speedup=3.12944

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.161019
Parallel Exec Time=0.0525339
Speedup=3.06505

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.169501
Parallel Exec Time=0.0585289
Speedup=2.89602

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.164425
Parallel Exec Time=0.0560601
Speedup=2.93301

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.162804
Parallel Exec Time=0.0517261
Speedup=3.14742

Omids-MacBook-Pro:final omid$ ./q2
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Exec Time=0.169535
Parallel Exec Time=0.0532789
Speedup=3.18203

Omids-MacBook-Pro:final omid$
```

همانطور که مشاهده می‌شود، میزان تسریعی حدود ۳ بدست می‌آید. برای چک کردن درستی نتایج، از تابع **checkResults** استفاده شده‌است که برابری مقادیر سریال و موازی، برابر بودن عناصر دو آرایه‌ی مرتب‌شده

توسط برنامه موازی و سریال و صعودی بودن آن توسط **assert** چک می‌شود و در صورت نقض هریک از موارد گفته‌شده، در ترمینال، ارور چاپ می‌کند.

مقایسه و بررسی دو پیاده سازی سری و موازی

در این قسمت دو پیاده سازی سری و موازی از یک کد که حاصل یک سری را محاسبه می کند با یکدیگر مقایسه شده اند. در این کد یک حلقه ی بیرونی داریم:

```
for( int j=0; j<VERYBIG; j++ )
```

درون این حلقه، دو حلقه ی داخلی داریم که وابسته به مقدار **j** حلقه ی بیرونی هستند:

```
for( k=0; k<j; k++ )
for( k=j; k>0; k-- )
```

در ادامه به بررسی دو پیاده سازی می پردازیم.

پیاده سازی سری

در این پیاده سازی تمام **iteration** های حلقه ها توسط یک ترد اجرا می شوند و از **directive** های **OpenMP** استفاده نشده است. برنامه یک بار اجرا می شود و زمان اول و آخر حلقه را می گیریم تا زمان اجرای آن بدست آید.

پیاده سازی موازی

در این پیاده سازی از **4** ترد استفاده شده است. ساختار **parallel** به صورت زیر تعریف شده است:

```
#pragma omp parallel \
    num_threads (4) \
    private( sumx, sumy, k, j, tstarttime, telapsedtime )
```

متغیرهایی که از نوع **private** تعریف شده اند در خط آخر این **directive** آمده اند. دو متغیر **sumx** و **sumy** در هر **iteration** حلقه مقداردهی می شوند، دو متغیر **k, j** پیمایش کننده های حلقه ها هستند و متغیرهای **tstarttime** و **telapsedtime** برای بدست آوردن زمان اجرای هر ترد استفاده می شوند. پس هیچکدام از این متغیرها بین تردها مشترک نیستند و در نتیجه از نوع **private** تعریف می شوند. حال به ساختار **for** توجه کنید که درون این ساختار **parallel** قرار گرفته است:

```
#pragma omp for \
        reduction( +: sum, total ) \
        schedule( static ) \
        nowait
```

علت استفاده از **reduction** این است که می خواهیم حاصل جمع چندین **iteration** ای که به هر ترد اختصاص می یابد با همین حاصل جمع برای تردهای دیگر باهم جمع شوند. سپس نوع **schedule** را تعیین کرده ایم که حالت های استاتیک، داینامیک با **chunck size** برابر با **1000** و داینامیک با **chunck size** برابر با **2000** استفاده شده اند. علت استفاده از **nowait** این است که می خواهیم هر ترد پس از اجرای **iteration** های حلقه به پایان آن برسد و زمان پایان خود را ثبت کند و منتظر تردهای دیگر نماند. یک بار در آغاز و بار دیگر در پایان حلقه زمان را ثبت کرده ایم تا زمان اجرای هر ترد بدست آید؛ علاوه بر آن در آغاز و پایان ساختار **parallel** هم زمان را ثبت کرده ایم تا زمان کل هر بار اجرای کد بدست آید.

نتایج

تصویر خروجی کد:

سری:

```
asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
$ ./q3_1
Omid Bodaghi -- 810196423       Mobina Shahbandeh -- 810196488

Serial timing for 100000 iterations

Time Elapsed       32092 mSecs Total=32.617277 Check Sum = 100000

asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
$ ./q3_1
Omid Bodaghi -- 810196423       Mobina Shahbandeh -- 810196488

Serial timing for 100000 iterations

Time Elapsed       31871 mSecs Total=32.617277 Check Sum = 100000

asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
$ ./q3_1
Omid Bodaghi -- 810196423       Mobina Shahbandeh -- 810196488

Serial timing for 100000 iterations

Time Elapsed       32652 mSecs Total=32.617277 Check Sum = 100000

asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
```

موازی:

static

```
asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
$ ./q3_2
Omid Bodaghi -- 810196423      Mobina Shahbandeh -- 810196488

OpenMP Parallel Timings for 100000 iterations

Time Elapsed for T0:      1972 mSecs
Time Elapsed for T1:      6138 mSecs
Time Elapsed for T2:     10308 mSecs
Time Elapsed for T3:     14268 mSecs

Time Elapsed for T0:      1987 mSecs
Time Elapsed for T1:      6136 mSecs
Time Elapsed for T2:     10179 mSecs
Time Elapsed for T3:     14086 mSecs

Time Elapsed for T0:      1972 mSecs
Time Elapsed for T1:      6017 mSecs
Time Elapsed for T2:     10049 mSecs
Time Elapsed for T3:     13898 mSecs

Time Elapsed for T0:      1975 mSecs
Time Elapsed for T1:      6043 mSecs
Time Elapsed for T2:      9980 mSecs
Time Elapsed for T3:     13917 mSecs

Time Elapsed for T0:      1975 mSecs
Time Elapsed for T1:      6071 mSecs
Time Elapsed for T2:     10007 mSecs
Time Elapsed for T3:     13941 mSecs

Time Elapsed for T0:      1952 mSecs
Time Elapsed for T1:      6084 mSecs
Time Elapsed for T2:     10006 mSecs
Time Elapsed for T3:     14062 mSecs

Average Time Elapsed      14030 mSecs Total=32.617277 Check Sum = 100000

asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
```

dynamic, 1000

```
asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
$ ./q3_2
Omid Bodaghi -- 810196423       Mobina Shahbandeh -- 810196488

OpenMP Parallel Timings for 100000 iterations

Time Elapsed for T1:      8170 mSecs
Time Elapsed for T0:      8413 mSecs
Time Elapsed for T3:      8506 mSecs
Time Elapsed for T2:      8582 mSecs

Time Elapsed for T0:      8214 mSecs
Time Elapsed for T2:      8247 mSecs
Time Elapsed for T3:      8451 mSecs
Time Elapsed for T1:      8773 mSecs

Time Elapsed for T2:      8007 mSecs
Time Elapsed for T0:      8063 mSecs
Time Elapsed for T3:      8263 mSecs
Time Elapsed for T1:      8531 mSecs

Time Elapsed for T3:      8048 mSecs
Time Elapsed for T0:      8055 mSecs
Time Elapsed for T1:      8169 mSecs
Time Elapsed for T2:      8482 mSecs

Time Elapsed for T2:      8022 mSecs
Time Elapsed for T0:      8055 mSecs
Time Elapsed for T1:      8159 mSecs
Time Elapsed for T3:      8439 mSecs

Time Elapsed for T1:      7983 mSecs
Time Elapsed for T0:      8100 mSecs
Time Elapsed for T3:      8222 mSecs
Time Elapsed for T2:      8415 mSecs

Average Time Elapsed       8538 mSecs Total=32.617277 Check Sum = 100000

asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
```

dynamic, 2000

```
asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
$ ./q3_2
Omid Bodaghi -- 810196423       Mobina Shahbandeh -- 810196488

OpenMP Parallel Timings for 100000 iterations

Time Elapsed for T1:      7686 mSecs
Time Elapsed for T0:      8023 mSecs
Time Elapsed for T2:      8288 mSecs
Time Elapsed for T3:      8815 mSecs

Time Elapsed for T3:      7647 mSecs
Time Elapsed for T0:      7698 mSecs
Time Elapsed for T2:      8390 mSecs
Time Elapsed for T1:      8731 mSecs

Time Elapsed for T0:      7763 mSecs
Time Elapsed for T3:      7989 mSecs
Time Elapsed for T1:      8553 mSecs
Time Elapsed for T2:      8774 mSecs

Time Elapsed for T3:      7734 mSecs
Time Elapsed for T0:      7750 mSecs
Time Elapsed for T1:      8309 mSecs
Time Elapsed for T2:      8573 mSecs

Time Elapsed for T0:      7625 mSecs
Time Elapsed for T3:      7800 mSecs
Time Elapsed for T1:      8337 mSecs
Time Elapsed for T2:      8460 mSecs

Time Elapsed for T0:      7623 mSecs
Time Elapsed for T1:      8031 mSecs
Time Elapsed for T2:      8262 mSecs
Time Elapsed for T3:      8799 mSecs

Average Time Elapsed       8693 mSecs Total=32.617277 Check Sum = 100000

asus@Mobina MINGW64 ~/Desktop/UT/UT7/Parallel Programming/CA/CA#04
```

همانطور که مشاهده می شود در حالت استاتیک اختلاف فاحشی بین زمان اجرای تردها وجود دارد. علت آنست که در حالت استاتیک **iteration** هایی با تعداد مساوی به ترتیب به تردها اختصاص داده می شود؛ بنابراین **iteration** های انتهایی به تردهای **T2** و **T3** اختصاص داده می شود. حال اگر به حلقه های داخلی توجه کنید مشاهده می کنید که تعداد **iteration** های آنها وابسته به **j** حلقه ی بیرونی است. پس اگر حلقه ی بیرونی **j** بزرگتری داشته باشد (**iteration** های انتهایی) حلقه های بیرونی نیز **iteration** های بیشتری خواهند داشت و درنتیجه زمان اجرا بیشتر خواهد شد. در حالت های داینامیک این مسئله وجود ندارد زیرا هرگاه تردی **idle** شود درخواست می کند که **chunk** های جدیدی به او اختصاص یابند و **iteration** های انتهایی همگی به یک ترد اختصاص نمی یابند و در نتیجه اختلاف فاحشی بین زمان اجرای تردها ایجاد نمی شود. در حالتی

که chunk size برابر با 2000 است بخش بیشتری از iteration های انتهایی به یک ترد اختصاص می یابند و در نتیجه اختلاف میان زمان اجرای تردها بیشتر از حالتی می شود که chunk size برابر با 1000 است.

میزان تسریع

همانطور که در تصاویر مشاهده شد، زمان اجرای برنامه ی سری در حدود 32000 میلی ثانیه است. زمان اجرای برنامه ی موازی در سه حالت به شرح زیر است:

static: 14030

dynamic, 1000: 8538

dynamic, 2000: 8693

پس میزان تسریع به صورت زیر خواهد شد:

static: $\frac{32000}{14030} = 2.28$

dynamic,1000: $\frac{32000}{8538} = 3.74$

dynamic,2000: $\frac{32000}{8693} = 3.68$