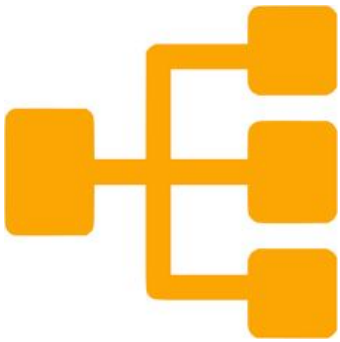


CA5 - Pthread

Parallel Programming

امید بداقی 810196423

مبینا شاه بنده 810196488

پاییز 99

دکتر صفری

مقدمه

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

بدست آوردن بزرگ ترین عنصر آرایه

ابتدا آرایه ای با 2^{20} عدد ممیز شناور رندوم می‌سازیم. برای محاسبه‌ی زمان اجرا، از تابع تعریف‌شده `getTime()` و از تابع `getTimeOfDay` کتابخانه **time.h** استفاده می‌کنیم. زمان ابتدا و انتهای اجرا را محاسبه می‌کنیم تا زمان اجرای الگوریتم مدنظر بدست بیاید. برای چاپ نتایج، از تابع `printResult()` استفاده می‌کنیم که مقدارهای بدست آمده توسط دو روش موازی و سریال و همچنین اسپیدآپ در آن گزارش می‌شوند.

پیاده سازی سری

به صورت کاملا عادی، آرایه را پیمایش می‌کنیم و هربار مقداری بزرگتر از آخرین مقدار بیشینه دیدیم، مقدار بیشینه و اندیس را بروزرسانی می‌کنیم.

پیاده سازی موازی

به اندازه تعداد هسته‌های فیزیکی(۶)، ترد برای اجرای موازی برنامه در نظر می‌گیریم. در تابع `findMaxParallel()`، ابتدا ورودی هر ترد را مشخص می‌کنیم. برای داشتن یک لیست از ورودی‌ها، از استراکت **parFunInput** استفاده می‌کنیم که ابتدا و انتهای بازه‌ای را که ترد در آن جست‌وجو می‌کند دارد. این لیست از ورودی‌ها اندازه یکسانی دارند تا لود کاری تمام تردها برابر باشد.

حال به تعداد مشخص‌شده، ترد می‌سازیم. هر ترد، با توجه به ورودی خود(که بازه را مشخص می‌کند) بر روی `findLocalMax()` صدا زده می‌شود. در انتها صبر می‌کنیم تا تمام تردها کار خود را انجام بدهند(با استفاده از **join** به تعداد تردها، هیچ تردی به خطوط بعدی نمی‌رود تا زمانی که تمام تردها کارشان تمام شود) و سپس

زمان اجرا محاسبه می‌شود. همچنین، در ابتدا یک **mutex_lock**، مقدار دهی اولیه می‌شود و در انتهای این تابع، آن را **destroy** می‌کنیم.

در تابع `findLocalMax()`، ابتدا ورودی را که از جنس **void*** می‌باشد را به همان جنس **parFunInput** تبدیل می‌کنیم. همانند قسمت سریال، در بازه‌ی مشخص‌شده توسط ورودی، مقدار بیشینه و اندیس آن را پیدا می‌کنیم. در پایان، مقدار بیشینه‌ی نهایی و اندیس نهایی را باید بروز رسانی کنیم. با توجه به اینکه چند ترد در حال انجام این تابع می‌باشند و بیشینه و اندیس نهایی، مقادیر به اشتراک گذاشته‌شده ای هستند، باید قبل از بروزرسانی آن‌ها، یک قفل قرار بدهیم و پس از انجام تغییرات(در صورت امکان) قفل را برداریم تا سایر تردها بتوانند کار خود را انجام بدهند.

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.

میزان تسریع

همانطور که مشاهده می‌شود، **speedup** ای حدود ۴.۲ تا ۴.۹ به دست می‌آید و مقادیر در روش سریال و موازی، با یکدیگر برابر می‌باشند.

نتایج

```
[172-11-14-155:CA5 omid$ g++ q1.cpp -o q1 -lpthread
[172-11-14-155:CA5 omid$ ./q1
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Result
maxVal=782577  index=344786
Parallel Result
maxVal=782577  index=344786

Serial Exec Time=0.00261807
Parallel Exec Time=0.000543118
Speedup=4.82046

[172-11-14-155:CA5 omid$ ./q1
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Result
maxVal=680427  index=79816
Parallel Result
maxVal=680427  index=79816

Serial Exec Time=0.00267816
Parallel Exec Time=0.000566006
Speedup=4.73168

[172-11-14-155:CA5 omid$ ./q1
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Result
maxVal=571289  index=729621
Parallel Result
maxVal=571289  index=729621

Serial Exec Time=0.00271988
Parallel Exec Time=0.000557899
Speedup=4.87521

[172-11-14-155:CA5 omid$ ./q1
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Result
maxVal=501198  index=307412
Parallel Result
maxVal=501198  index=307412

Serial Exec Time=0.00271797
Parallel Exec Time=0.000577927
Speedup=4.70297

[172-11-14-155:CA5 omid$ ./q1
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Result
maxVal=704955  index=611163
Parallel Result
maxVal=704955  index=611163

Serial Exec Time=0.00283098
Parallel Exec Time=0.000573874
Speedup=4.93311

[172-11-14-155:CA5 omid$ ./q1
Mobina Shahbandeh  --  810196488        Omid Bodaghi  --  810196423

Serial Result
maxVal=627819  index=1036560
Parallel Result
maxVal=627819  index=1036560

Serial Exec Time=0.00279498
Parallel Exec Time=0.000617981
Speedup=4.52276

172-11-14-155:CA5 omid$
```

مرتب سازی آرایه در حالت صعودی

ابتدا آرایه ای با 2^{20} عدد ممیز شناور رندوم می‌سازیم. حال برنامه را یکبار به صورت سریال و یکبار به صورت موازی اجرا می‌کنیم و نتایج را ثبت می‌کنیم. برای محاسبه‌ی زمان اجرا از `timeGetTime()` استفاده کرده ایم.

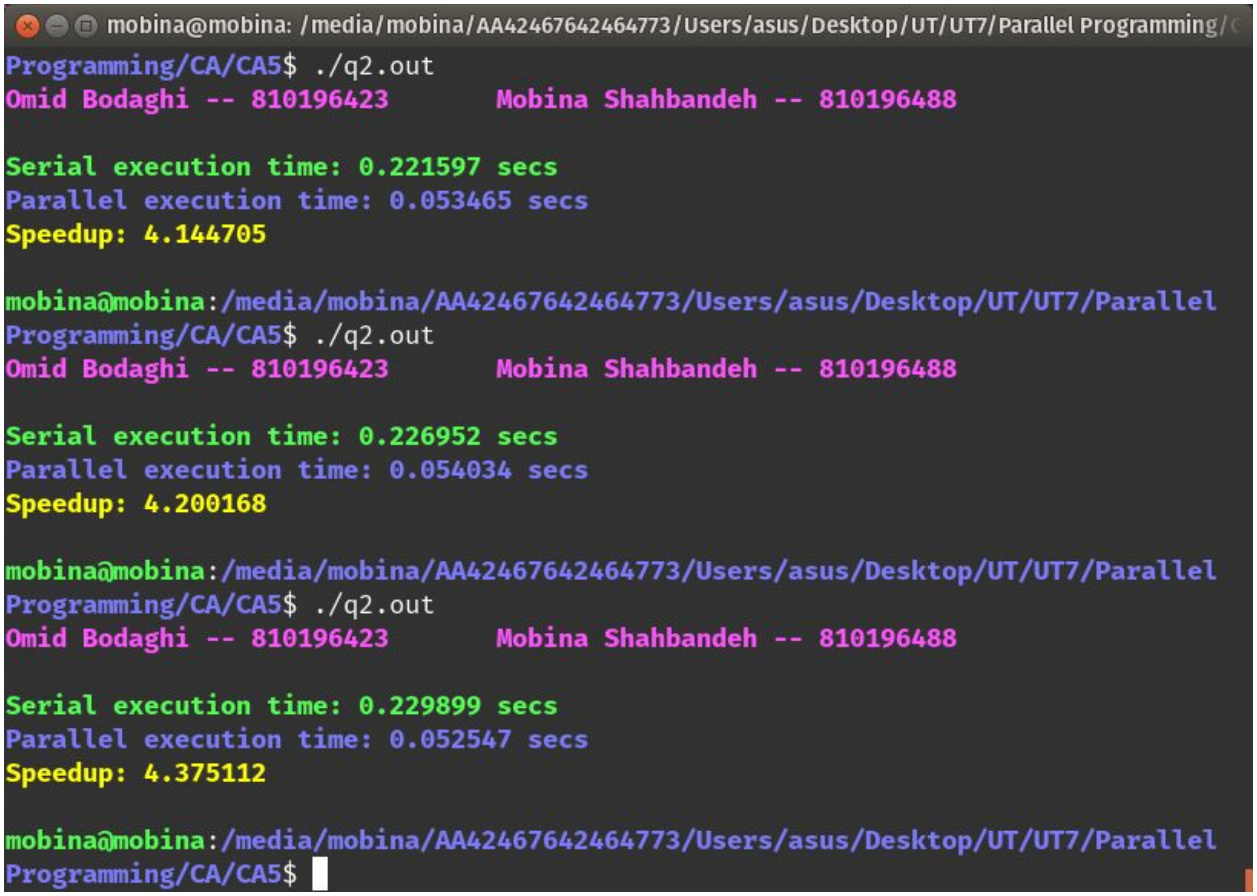
پیاده سازی سری

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

ابتدا در تابع **main**، ترد اصلی تابع `parallelQuickSort` را اجرا می کند. این تابع، ابتدا **partitioning** را انجام می‌دهد و سپس، برای سورت‌کردن سمت چپ یک ترد جدید تعریف می‌کند و برای سورت کردن سمت راست همان ترد کنونی مجددا این تابع را اجرا می کند. آرایه، در تمام مراحل به صورت **shared** می‌باشد. زمانی که طول آرایه برای **sort**، از آستانه ی تعیین شده **(50000)** کمتر باشد، دیگر ترد جدید ساخته نمی شود و آن آرایه به صورت سریال مرتب می‌شود. دلیل آن این است که هزینه ساختن ترد برای آرایه های کوچک و مرتب‌سازی آن به صورت موازی، می‌تواند سربار بیشتری نسبت به اجرای سریال آن داشته‌باشد.

نتایج



همانطور که مشاهده می‌شود، میزان تسریعی حدود ۴ 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.