به‌نام خدا

۱. در این بخش برای موازی‌سازی، به تعداد NUM_THREADS ترد در نظر می‌گیریم و آرایه را نیز به بخش‌های مساوی تقسیم می‌کنیم که تعداد این بخش‌ها نیز برابر با NUM_THREADS است تا بتوانیم عملیات را به‌طور موازی انجام دهیم و هر ترد بخش مربوط به خودش را پردازش کند. ورودی هر ترد با ساختار inParam مشخص می‌شود و خروجی آن هم با ساختار maxElement.

```
typedef struct
{
  long startIdx;
  long endIdx;
} inParam;
```

```
typedef struct
{
  float value;
  long idx;
} maxElement;
```

سپس از دو متغیر محلی localMax و localMaxIdx برای هر ترد استفاده می‌کنیم تا هر ترد با داده‌های مربوط به خودش ماکسیمم را حساب کند و نهایتاً هر ترد مقدار ماکسیمم خودش را از طریق ساختار maxElement و با pthread_exit خروجی می‌دهد. برای یافتن مقدار ماکسیمم کلی نیز ترد اصلی یا همان پدر، همه‌ی این مقادیر را از تردها پس از pthread_join دریافت کرده و آن را با ماکسیمم فعلی مقایسه می‌کند و اگر بیشتر بود آن را به‌روزرسانی می‌کند.

پس از اجرا مشاهده می‌کنیم که پاسخ‌ها با هم برابر هستند و همچنین تسریعی که به دست آمده ۱٫۷۹ برابر است. همچنین لازم‌به‌ذکر است که مقادیر مختلفی برای NUM_THREADS تست شد که در حالت ۸ ترد بیش‌ترین تسریع به دست آمد.

```
~/Courses/9-Fall99/PP/CAs/CA5/1 ./main
Hossein Soltanloo
Student Number: 810195407
The serial result is  Value = 100.000000    Index = 1310
The parallel result is  Value = 100.000000    Index = 1310
Serial Run time = 4872292
Parallel Run time = 2710189
Speedup = 1.797768
```

۲. در این قسمت ابتدا به مشاهده‌ی نتایج می‌پردازیم:

```
~/Courses/9-Fall99/PP/CAs/CA5/2 ./main
Hossein Soltanloo
Student Number: 810195407

Serial Run time = 434233891
Parallel Run time = 96382481
Parallel Speedup = 4.505320
Test if arrays are sorted:
Serial:
Array sorted.
Parallel:
Array sorted.
```

اولین موردی که به آن باید اشاره شود این است که در quicksort آرایه با pivot به دو بخش بزرگ‌تر و کوچک‌تر از pivot تقسیم می‌شود که به‌طور بازگشتی این بخش‌ها خودشان شکسته می‌شوند تا به تک عنصر برسیم و این‌گونه عمل مرتب‌سازی انجام می‌گیرد. حال برای انجام این کار با pthread، اولین چیزی که به ذهن می‌رسد این است که این دو بخش‌ها را بین دو ترد تقسیم کنیم که به همین ترتیب به‌صورت بازگشتی، تردهای دیگر درون این‌ها ایجاد می‌شوند و در نتیجه تعداد زیادی ترد ایجاد می‌شود. برای جلوگیری از تولید بی‌شمار تردها، یک شرط تعیین می‌کنیم که اگر اندازه‌ی آرایه‌ای که به هر ترد محول می‌شود کوچک‌تر از یک مقدار خاص باشد، آن مرتب‌سازی به‌صورت سریال انجام شود و فقط وقتی این اندازه بزرگ‌تر است، ترد جدیدی ایجاد می‌شود تا بدین شکل بتوانیم سربار تردها را کاهش دهیم و تسریع زیادی به‌دست آوریم. بدین ترتیب از ساختن بی‌رویه‌ی تردها جلوگیری می‌شود و سربار بسیار کاهش پیدا می‌کند تا جایی که در تصویر مشاهده می‌شود میزان تسریع ۴٫۵ برابر است. نهایتاً مشاهده می‌شود که در همه‌ی حالت‌ها آرایه به‌درستی مرتب شده است.

هم‌چنین برای مشخص‌کردن محدوده‌ای که هر ترد روی آن پردازش انجام می‌دهد از ساختار data استفاده می‌کنیم تا ابتدا و انتهای محدوده مشخص شود. این ساختار نیز از آن‌جا که malloc شده است باید free شود.