Parallel Programming

Assignment V

Omid Ahmadi
Mohammad Mahdi Islami

Part 1

در ابتدا شماره دانشجویی اعضای گروه را پرینت می کنیم.سپس متغیر های اولیه مانند تعداد تردها را تعریف می کنیم.سپس ساختار ThreadParameters را تعریف می‌کنیم که در آن پارامترهای مورد نیاز برای پاس دادن به تابع ترد وجود دارد.سپس در main 2 به توان 20 عدد ممیز شناور تصادفی را تعریف می کنیم.

Serial

ابتدا متغیر max را تعریف می کنیم که درون آن عنصر اول ارایه است. سپس درون حلقه for بررسی می کنیم که آیا عنصر بعدی ارایه از عنصر فعلی بزرگتر است یا نه. اگر بزرگتر بود آن را جایگزین می کنیم. همچنین اندیس آن را درون متغیر index نگه می داریم. بدین ترتیب بزرگترین عنصر ارایه به همراه اندیس آن پیدا می شود.

Parallel

ابتدا متغیرهای مورد نیاز مانند pthread_t و ThreadParameters که وابسته به تعداد تردهاست را تعریف می‌کنیم. بعد در یک حلقه پارمترهای مختص به هر ترد مانند شروع و پایان آن را مشخص می‌کنیم. سپس در همین حلقه ترد ها درست می‌شوند و تابع find_max را به همراه ارگومان ان یعنی ThreadParameters به آن پاس می‌دهیم. در تابع find_max ابتدا پارامترهای ورودی را تجزیه می کنیم. سپس در یک حلقه که شروع و پایان آن برای هر ترد متفاوت است اقدام به محاسبه بزرگترین عدد و اندیس آن می‌کنیم. در انتها این دو مقدار را درون دو پارامتر که در ThreadParameters مشخص شده است می‌ریزیم. حال به main برمی‌گردیم و در حلقه ای با استفاده از pthread_join منتظر خاتمه ی تردهای ایجاد شده می‌شویم.در انتها مقدار به دست آمده در تردها را با یک دیگر مقایسه می‌کنیم تا در نهایت بزرگترین عنصر و اندیس آن پیدا شود.
حال به مقایسه جواب های حاصل، مدت زمان اجرا و نسبت آن در این دوحالت می پردازیم:

Result

همانطور که مشاهده می شود جواب های یکسان است و speedup حدود دو و نیم است.

```
mahdi@MacBook-Pro-2 Part1 % ./main
Serial Result: 100
index: 1310
POSIX Result: 100
index: 1310
Serial Runtime: 5981038
Parallel Runtime: 2965908
Speed-up: 2.0166
mahdi@MacBook-Pro-2 Part1 %
```

Part 2

در این قسمت میخواهیم مرتب‌سازی را در برنامه ی سریال و موازی مقایسه کنیم.

Serial

هسته ی الگوریتم Quick Sort تابع Partition است. این تابع عنصر اول بازه ای را که قرار است رویش عمل کند را در نظر می‌گیرد و بعد این تابع عنصرها را به۲ دسته تقسیم می‌کند آن هایی که بزرگ تر از عنصر اول هستند آن هایی که کوچک تر از عنصر اول هستند و عنصر اول را بین این دو دسته قرار می دهد و بعد مکانی را که عنصر ما بین این دو دسته قرار دارد را بر می گرداند .

```
int partition (int low, int high, vector<float>& vec){
  int i, j, temp, key;
    key = vec[low];
    i= low + 1;
    j= high;
    while(1){
      while(i < high && key >= vec[i])
          i++;
      while(key < vec[j])
            j--;
      if(i < j){
          temp = vec[i];
          vec[i] = vec[j];
          vec[j] = temp;
      } else {
          temp= vec[low];
          vec[low] = vec[j];
          vec[j]= temp;
          return(j);
      }
    }
}
```

Quick sortیک روش مرتب سازی بازگشتی است، بنابراین هر بازه را به بازه های کوچکتر تقسیم میکند و دوباره خودش را فراخوانی میکند :

Serial Quick Sort Implementation:

```
void serial_quick_sort (int p, int r, vector<float>& vec){
    if (p < r) {
        int q = partition (p , r, vec);
        serial_quick_sort (p, q - 1, vec);
        serial_quick_sort (q + 1, r, vec);
    }
}
```

Parallel

از میان الگوریتم‌های مرتب‌سازی که قابل موازی‌سازی هستند، الگوریتم Quick Sort را انتخاب می‌کنیم که از نظر هزینه به نسبت مطلوب است :

O(nlgn)

در قسمت موازی از Posix و کتابخانه pthread استفاده کردیم. روند اجرا به این صورت است که در ابتدا یک ترد اصلی ایجاد می‌کنیم. هر ترد آرگومان های نقطه ی شروع و پایان خود و تعداد ترد ها را دارد. هر بار که ترد ایجاد می‌شود این مقدار دو برابر می‌شود تا زمانی که با مقدار ماکزیمم مطلوب که با سعی و خطا به دست آمده است برسد و در آن صورت هر ترد شروع به فعالیت مرتب سازی سریع می‌کند و سپس join می‌شود با ترد پدر خود و همین‌طور الی آخر. با استفاده از دستور زیر تابع ترد را اجرا می‌کنیم و آرگومان‌های آن را پاس می‌دهیم:

```
pthread_create(&my_threads[i], NULL, parallelQuicksort, (void*)&arg[i]);
```

```
void* parallelQuicksort(void* args) {
    threadArguments* arg = (threadArguments*) args;
    int low = arg->low;
    int High = arg->High;
    int threads = arg->threadNum;
    if (threads >= MAX)
        smallQuickSort(low, High);
    else if (low <  High)
    {
        pthread_t my_threads[2];
        int pivot = threadPartition(low, High);
        threadArguments arg[2];
        arg[0].low = low;
        arg[0].High = pivot - 1;
        arg[0].threadNum = threads * 2;
        arg[1].low = pivot + 1;
        arg[1].High = High;
        arg[1].threadNum = threads * 2;
        for (int i = 0; i < 2; ++i)
            pthread_create(&my_threads[i], NULL, parallelQuicksort, (void*)&arg[i]);
        for (int i = 0; i < 2; ++i)
            pthread_join(my_threads[i], NULL);
    }
    pthread_exit(0);
}
```

Result

با ۲۰ ترد :

```
Part2 — -zsh — 80×24
mahdi@MacBook-Pro-2 Part2 % make
g++-10 -c -fopenmp -I/opt/intel/ipp/include main.cpp
g++-10 -fopenmp  -L/opt/intel/ipp/lib /opt/intel/ipp/lib/libippi.a  /opt/intel/i
pp/lib/libipps.a /opt/intel/ipp/lib/libippvm.a /opt/intel/ipp/lib/libippcore.a m
ain.o -o main
mahdi@MacBook-Pro-2 Part2 % ./main
SIDs : 810195548 , 810195346
----------------------------
Serial Runtime: 698890528
Parallel Runtime: 388387602
Speed-up: 1.79947
mahdi@MacBook-Pro-2 Part2 %
```