دستور کار آزمایشگاه
درس برنامه نویسی چندهسته‌ای – دکتر ممتازپور
دانشکده مهندسی کامپیوتر
دانشگاه صنعتی امیرکبیر

به نام خدا

آزمایش ۱

هدف‌ها:	نیازمندی‌ها:
• آشنایی با محیط ویژوال استودیو[1]	• ویژوال استودیو/ ویژوال C++
• آشنایی با OpenMP	• آشنایی با زبان C یا C++
• بررسی یک برنامه سریال و موازی‌سازی آن.	

مقدمه:

در این آزمایش شما باید یک برنامه سریال را با تنظیمات مناسب در محیط Visual Studio کامپایل و اجرا کنید. سپس به کمک دستورات راهنمای[2] OpenMP یک برنامه سریال نمونه را موازی و از صحت عملکرد آن اطمینان حاصل کنید. در نهایت تسریع به‌دست‌آمده را محاسبه کنید.

شرح آزمایش:

❖ **مرحله اول: ایجاد پروژه در ویژوال استودیو، تنظیم پروژه، کامپایل و اجرای یک کد سریال نمونه**

۱. ابتدا محیط Visual Studio را اجرا کرده و از منوی File گزینه New Project را انتخاب کنید.
۲. در سمت چپ صفحه باز شده، از لیست Template، گزینه‌ی C++ را انتخاب کرده و Win32 را انتخاب کنید.
۳. از وسط صفحه گزینه Win32 Console Application را انتخاب کنید. مسیر و نام پروژه را مشخص کنید و کلید Next را کلیک کنید.
۴. بر روی Next کلیک کنید. Application type باید Console Application باشد. Empty Project را انتخاب و Finish را کلیک کنید.
۵. در پنجره Solution Explorer بر روی پوشه‌ی Source Files کلیک کنید. سپس کلید ترکیبی Shift + Alt + A را فشار دهید و فایل cpp این آزمایش را به پروژه اضافه کنید.
۶. کد داده شده را بخوانید و سپس در فایل قرار دهید.
۷. از نوارابزار بالای برنامه در قسمت Configuration، Debug را به Release تغییر دهید.
۸. برای اندازه‌گیری زمان اجرا، از دستور omp_get_wtime استفاده شده که در کتابخانه OpenMP فراهم شده است. این تابع زمان اجرا را به ثانیه بر می‌گرداند و خروجی آن double است. سعی کنید همیشه از همین تابع استفاده کنید و هنگام پرینت کردن آن، حواستان به نوع داده (double) باشد. برای استفاده از این تابع و البته بقیه توابع OpenMP، کتابخانه omp.h باید در فایل کد include شده باشد.
۹. برای اجرای برنامه از کلید ترکیبی Ctrl + F5 استفاده کنید.

[1] Visual Studio
[2] Directive

به سوالات زیر پاسخ دهید:

۱. کدام قسمت برنامه بیشترین سهم در زمان اجرا را دارد؟ بررسی کنید کدام قسمت نیاز به موازی‌سازی دارد.
۲. فلسفه تکرار محاسبات در یک حلقه ۱۰ تایی و چاپ نتایج در هر مرحله چیست؟ کدام بخش نتایج بین تکرارها متفاوت است؟ چرا؟
۳. تفاوت دو حالت Debug و Release در چیست؟ چرا کد کامپایل شده در حالت Release سریعتر است؟
۴. برای موازی‌سازی این کد از چه **روش تجزیه** و **الگوی موازی‌سازی** می‌توان استفاده کرد؟

❖ **مرحله دوم: موازی‌سازی کد**

۱. برای موازی‌سازی کد اطمینان حاصل کنید OpenMP در تنظیمات پروژه شما فعال شده باشد. اینکار را باید به ازای هر پروژه‌ای که می‌سازید تکرار کنید. قبل از آن، با اضافه کردن قطعه کد زیر به ابتدای تابع main، می‌توان بررسی کرد که OpenMP فعال شده است یا خیر. با اجرای برنامه باید پیغام عدم پشتیبانی را ببینید.

```
#ifndef _OPENMP
	printf("OpenMP is not supported, sorry!\n");
	getchar();
	return 0;
#endif
```

۲. برای فعالسازی OpenMP، بر روی نام پروژه در Solution Explorer کلیک کرده و Properties را انتخاب کنید. سپس به قسمت C/C++ و Language رفته و گزینه OpenMP Support را بر روی Yes قرار دهید. دقت کنید در بالای همین پنجره این کار را برای هر دو Configuration یعنی Debug و Release تکرار کنید و هر بار گزینه Apply را بزنید تا اگر لازم شد کد را دیباگ کنید فراموش نکنید OpenMP را فعال کنید.
۳. برای موازی‌سازی کد، قبل از حلقه اصلی VERYBIG، راهنمای زیر را اضافه کنید.کلمه parallel در این راهنما باعث می‌شود چند نخ ساخته شده و کلمه for نیز باعث می‌شود تکرارهای حلقه بین چند نخ تقسیم شود و نخها تکرارها را به صورت موازی اجرا کنند.

```
#pragma omp parallel for
```

۴. برنامه را اجرا کنید و خروجی را با خروجی برنامه سریال مقایسه کنید. چرا مقادیر sum و total با برنامه سریال و همچنین از یک اجرا به اجرای دیگر متفاوت است؟
۵. همانطور که حدس زدید این برنامه شرایط مسابقه دارد و مشکلات آن باید حل شود. دقت کنید در وضعیت فعلی، متغیرهای sumx، sumy، total و همچنین اندیس k به صورت shared است. تنها اندیس j که اندیس حلقه موازی شده است (حلقه‌ای که بلافاصله بعد از راهنمای for می‌آید) به صورت اتوماتیک اختصاصی (private) می‌شود. بنابراین باید با استفاده از عبارت private()، متغیرهایی که باید اختصاصی باشند را اختصاصی کنیم. از کد نوشته شده استنباط می‌شود که متغیرهای sumx و sumy باید اختصاصی باشند چون مقدار آنها در ابتدای هر تکرار حلقه VERYBIG صفر می‌شود. ولی متغیر sum و total اشتراکی هستند چون نتیجه محاسبه شده در همه تکرارها را تجمیع می‌کنند (متغیر sum یک شمارنده است که تعداد تکرارها را می‌شمرد و متغیر total نتیجه محاسبات sumx و sumy همه تکرارها را ادغام می‌کند). بنابراین متغیرهای k، sumx و sumy باید اختصاصی شوند و متغیرهای sum و total باید اشتراکی باقی بمانند (پیش‌فرض متغیرها اشتراکی هستند و نیازی به گذاشتن عبارت shared() نیست مگر پیش‌فرض را بر روی default(none) قرار دهید).

```
#pragma omp parallel for private(k, sumx, sumy)
```

۶. برنامه را پس از اعمال تغییر بالا یکبار دیگر اجرا کنید. همانطور که می‌بینید مشکل پابرجاست. دلیل آن این است که هنوز مشکل شرایط رقابت برطرف نشده و دسترسی نخها به صورت همزمان به متغیرهای sum و total هماهنگ نشده است. یک راه برای حل این مشکل استفاده از بلوک critical است. این بلوک باعث می‌شود در هر لحظه فقط یک نخ بتواند مقدار sum یا total را به روز کند.

۷. قسمت‌هایی از کد را که به روز رسانی sum یا total را انجام می‌دهند با استفاده از بلوک critical محافظت کنید. مثلا:

```
#pragma omp critical
            sum += 1;
```

سپس کد را دوباره اجرا کنید و نتیجه را ببینید.

۸. یک راه دیگر برای حل مشکل رقابت در این برنامه نمونه، استفاده از متغیرهای اختصاصی sum و total برای نخهاست، به طوری که هر نخ متغیرهای sum و total خودش را جداگانه محاسبه کند و در انتها مقادیر با یکدیگر جمع شود. می‌توان این دو کار را (یعنی اختصاصی کردن متغیرها و تجمیع نتایج آنها) با استفاده از عبارت reduction()‌ به خود کامپایلر سپرد. در واقع با استفاده از دستور reduction، کامپایلر ابتدا یک کپی اختصاصی از متغیرهایی که در این عبارت نام برده شده به ازای هر نخ ایجاد می‌کند و محاسبات درون حلقه موازی را بر روی آنها انجام می‌دهد، سپس کد مورد نیاز برای ادغام این متغیرهای اختصاصی و ذخیره نتایج بر روی متغیر اشتراکی متناظر را نیز به کد اضافه می‌کند.

```
#pragma omp parallel for private(k, sumx, sumy) reduction(+:sum, total)
```

۹. کد را یکبار دیگر و پس از اعمال تغییر بالا اجرا کنید و زمان اجرا را مقایسه کنید.

۱۰. کد را یکبار دیگر بدون استفاده از critical، atomic و reduction اصلاح کنید و نتیجه و زمان اجرا را گزارش کنید.

به سوالات زیر پاسخ دهید:

۱. بررسی کنید تعداد نخهای ساخته شده در راهنمای parallel در صورتی که num_threads را قید نمی‌کنید چندتاست و چه رابطه‌ای با تعداد هسته‌های کامپیوتر شما دارد؟
۲. آیا در این برنامه به جای critical می‌توان از atomic استفاده کرد؟
۳. با توجه به این که معمولا استفاده از reduction نسبت به critical توصیه می‌شود، آیا زمان اجرای بندهای ۷ و ۹ تفاوت دارند؟ به ازای مقادیر بزرگتر VERYBIG یا تعداد بیشتر یا کمتر نخ چطور؟

موارد تحویلی:

کدهای نوشته شده، اسکرین شات نتایج هر مرحله، تحلیل‌های انجام شده و پاسخ به سوالات را در قالب یک فایل pdf به نام و شماره دانشجویی خودتان در سامانه دروس بارگزاری کنید. studenName_stdNumber_Lab1.pdf