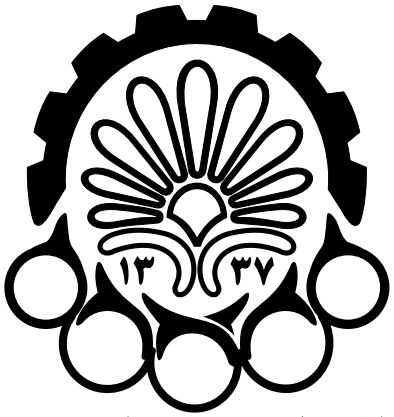


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

برنامه‌نویسی چندهسته‌ای

تمرین چهارم: آشنایی با برنامه‌نویسی OpenMP

رادین شایانفر

شماره دانشجویی: ۹۷۳۱۰۳۲

بهار ۱۴۰۰

۱. کد برنامه در فایل Q1.cpp موجود است.

ابتدا توابع مربوط به جمع ضرب ماتریس‌ها را به صورت سریال می‌نویسیم. همچنین تابع ترانهاده را به دو شکل عادی و in place (برای صرفه‌جویی در حافظه اگر ممکن باشد) می‌نویسیم. سپس با استفاده از `#pragma omp parallel for` توابع نوشته شده را موازی می‌کنیم. باید توجه داشت که تقسیم کار تابع ترانهاده در حالت in place متوازن نیست. به همین دلیل از `schedule(static, n / 32)` برای آن استفاده کردیم.

پس از نوشتن کد سریال و موازی، آن‌ها را با اندازه ورودی‌های مختلف آزمایش می‌کنیم. نتایج این آزمایش‌ها در جدول‌های ۱، ۲ و ۳ به ترتیب برای اجرای سریال، اجرا با ۴ نخ و اجرا با ۸ نخ آمده است.

جدول ۱: زمان اجرای **سریال** با اندازه ورودی‌های مختلف

	اندازه ورودی				
	64×64	128×128	256×256	512×512	1024×1024
زمان اجرا (ثانیه)	0.001755	0.016559	0.119641	0.645355	6.277222

جدول ۲: زمان اجرا با **۴ نخ** و اندازه ورودی‌های مختلف

	اندازه ورودی				
	64×64	128×128	256×256	512×512	1024×1024
زمان اجرا (ثانیه)	0.000911	0.003059	0.024683	0.212814	2.463785
تسریع	1.92	5.41	4.84	3.03	2.54

جدول ۳: زمان اجرا با **۸ نخ** و اندازه ورودی‌های مختلف

	اندازه ورودی				
	64×64	128×128	256×256	512×512	1024×1024
زمان اجرا (ثانیه)	0.000327	0.001978	0.016777	0.143360	2.719983
تسریع	5.36	8.37	7.13	4.50	2.30

همانطور که می‌بینیم، برای Nهای ۱۲۸ و ۲۵۶ بیشترین تسریع را داریم. دلیل آن احتمالا به علت جا شدن ماتریس‌ها در کش و همچنین به صرفه بودن سربار موازی‌سازی برای آن است.

از آنجا که برای تمام Nها به تسریع بیشتر از یک دست پیدا کردیم، بنابراین حالت سریال در هیچ‌کدام از موارد بهتر عمل نمی‌کند.

۲. کد برنامه در فایل Q2.cpp موجود است.

در این کد با استفاده از دو قفل `lock1` و `lock2` و داشتن دو `section` که تنها دو نخ آن‌ها را اجرا می‌کنند، بن‌بست به وجود می‌آوریم. به این شکل که اگر پیش از آن که یک نخ بتواند هر دو قفل را در اختیار بگیرد، نخ دوم قفل دیگر را در اختیار بگیرد، بن‌بست رخ می‌دهد و هر دو نخ منتظر آزاد شدن قفلی هستند که در اختیار نخ دیگر است.

۳. کد برنامه در فایل Q3.cpp موجود است.

برای محاسبه دترمینان، از تجزیه LU استفاده می‌کنیم. این تجزیه را برای موازی‌سازی به صورت بلوکی محاسبه می‌کنیم. به عنوان مثال، اگر ماتریس را به بلوک‌های ۳ در ۳ بشکنیم خواهیم داشت:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{pmatrix}$$

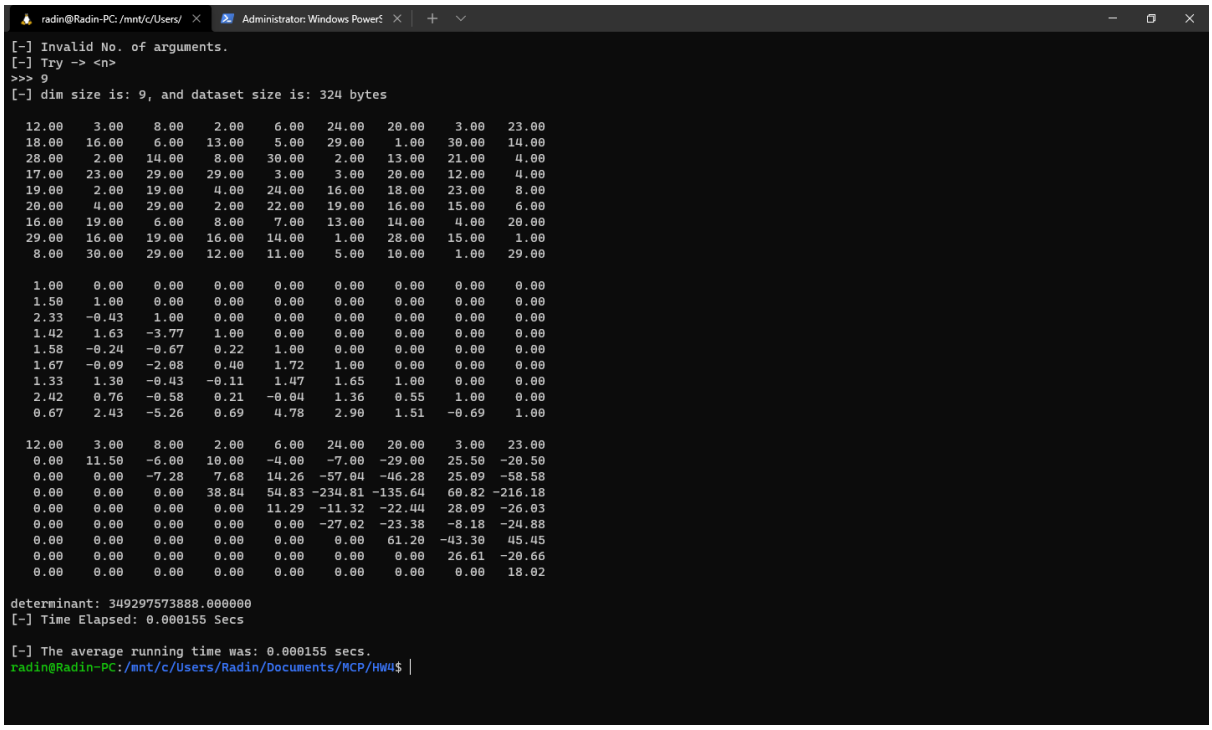
$$\begin{aligned} A_{11} &= L_{11}U_{11}, & A_{12} &= L_{11}U_{12}, & A_{13} &= L_{11}U_{13}, \\ A_{12} &= L_{21}U_{11}, & A_{22} &= L_{21}U_{12} + L_{22}U_{22}, & A_{23} &= L_{21}U_{13} + L_{22}U_{23}, \\ A_{31} &= L_{31}U_{11}, & A_{32} &= L_{31}U_{12} + L_{32}U_{22}, & A_{33} &= L_{31}U_{13} + L_{32}U_{23} + L_{33}U_{33}. \end{aligned}$$

با کمی اعمال جبری، می‌توانیم محاسبه بلوک‌های بالا را به وظایف زیر بشکنیم. وظایفی که در یک مستطیل قرار دارند، می‌توانند به صورت مستقل از هم اجرا شوند. اما پیش از رفتن به مستطیل بعدی، باید کار مستطیل قبلی تمام شده باشد.

1: $A_{1,1} \to L_{1,1}U_{1,1}$	6: $A_{2,2} = A_{2,2} - L_{2,1}U_{1,2}$	11: $L_{3,2} = A_{3,2}U_{2,2}^{-1}$
2: $L_{2,1} = A_{2,1}U_{1,1}^{-1}$	7: $A_{3,2} = A_{3,2} - L_{3,1}U_{1,2}$	12: $U_{2,3} = L_{2,2}^{-1}A_{2,3}$
3: $L_{3,1} = A_{3,1}U_{1,1}^{-1}$	8: $A_{2,3} = A_{2,3} - L_{2,1}U_{1,3}$	13: $A_{3,3} = A_{3,3} - L_{3,2}U_{2,3}$
4: $U_{1,2} = L_{1,1}^{-1}A_{1,2}$	9: $A_{3,3} = A_{3,3} - L_{3,1}U_{1,3}$	14: $A_{3,3} \to L_{3,3}U_{3,3}$
5: $U_{1,3} = L_{1,1}^{-1}A_{1,3}$	10: $A_{2,2} \to L_{2,2}U_{2,2}$	

حال به پیاده‌سازی الگوریتم گفته شده می‌پردازیم. ابتدا کد آن را در حالت سریال نوشته و صحت عملکرد آن را با بررسی درستی ماتریس‌های L، U و مقدار دترمینان (ضرب مقادیر روی قطر اصلی U) بررسی می‌کنیم.

پس از نوشتن کد سریال، با استفاده از `#pragma omp task` وظایفی که از هم مستقل هستند را توسط نخ‌های مختلف موازی می‌کنیم. همچنین پس از ساخت وظایف موجود در هر مستطیل شکل بالا، با استفاده `#pragma omp taskwait` اجرای نخ سازنده وظایف را متوقف می‌کنیم تا تمامی وظایف ساخته شده اجرا شوند. در نهایت ماتریس‌های L، U و مقدار دترمینان را برای یک ماتریس ۹ در ۹ با بلوک‌های ۳ در ۳ بررسی می‌کنیم. همانطور که در شکل زیر می‌بینیم موازی‌سازی به شکل صحیح صورت گرفته است.



در انتها برنامه را برای تعداد نخ‌های مختلف و اندازه ورودی‌های بزرگ‌تر می‌آزماییم که نتایج آن در جدول ۴ آمده است. این نتایج روی یک پردازنده ۸ هسته‌ای به دست آمده است. بنابراین استفاده از ۱۰ و ۱۲ نخ نتیجه‌ای متفاوت از حالت ۸ نخ به ما نمی‌دهد.

توجه شود که در اینجا هر سطر و ستون ماتریس به ۱۶ قسمت تقسیم شده است. به همین دلیل موازی‌سازی آن در ماتریس‌هایی که اندازه آن‌ها کوچک‌تر است منجر به تسریع مناسبی نمی‌شود. برای تسریع گرفتن در اندازه‌های کوچک‌تر، می‌توان بلوک‌ها را بزرگ‌تر کرد. همچنین ستون تسریع نیز با میانگین گرفتن تسریع در ۲ ستون سمت چپ جدول محاسبه شده است.

جدول ۴: نتایج اجرا با ۱۶ بلوک در هر سطر و ستون

تعداد نخ‌ها	زمان اجرا (ثانیه)					تسریع
	128×128	256×256	512×512	1024×1024	4096×4096	
سریال	0.000821	0.005034	0.041647	0.338921	23.258996	-
۲	0.001806	0.005035	0.020869	0.168551	12.561260	1.93
۵	0.001290	0.002762	0.010622	0.082139	5.782565	4.07
۸	0.001988	0.002709	0.009184	0.065322	4.416457	5.22
۱۰	0.021728	0.022061	0.022059	0.069593	4.248432	5.17
۱۲	0.021897	0.022579	0.023398	0.068674	4.278032	5.18