



دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

برنامه نویسی چندهسته‌ای

تمرین چهارم: آشنایی با برنامه نویسی 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					زمان اجرا (ثانیه)
1024×1024	512×512	256×256	128×128	64×64	
6.277222	0.645355	0.119641	0.016559	0.001755	

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

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

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

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



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

از آنجا که برای تمام N ها 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، بن بست 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، بن بست 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:

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

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

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: $A_{1,1} \rightarrow L_{1,1}U_{1,1}$	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: $A_{3,3} \rightarrow L_{3,3}U_{3,3}$
5: $U_{1,3} = L_{1,1}^{-1}A_{1,3}$	10: $A_{2,2} \rightarrow L_{2,2}U_{2,2}$	



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 و 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 از `#pragma omp taskwait` اجرای نخ سازنده وظایف را متوقف می‌کنیم تا تمامی وظایف ساخته شده اجرا شوند. در نهایت ماتریس‌های L و U و مقدار دترمینان را برای یک ماتریس 9×9 با بلوک‌های 3×3 در 3 بررسی می‌کنیم. همانطور که در شکل زیر می‌بینیم موازی‌سازی به شکل صحیح صورت گرفته است.

```
radin@Radin-PC: /mnt/c/Users/ \ Administrator: Windows PowerShell
[~] Invalid No. of arguments.
[~] Try -> <n>
>>> 9
[~] dim size is: 9, and dataset size is: 324 bytes

12.00  3.00  8.00  2.00  6.00  24.00  20.00  3.00  23.00
18.00 16.00  6.00 13.00  5.00  29.00  1.00  30.00  14.00
28.00  2.00 14.00  8.00 30.00  2.00 13.00  21.00  4.00
17.00 23.00 29.00 29.00  3.00  3.00  20.00  12.00  4.00
19.00  2.00 19.00  4.00 24.00 16.00 18.00  23.00  8.00
20.00  4.00 29.00  2.00 22.00 19.00 16.00  15.00  6.00
16.00 19.00  6.00  8.00  7.00 13.00 14.00  4.00  20.00
29.00 16.00 19.00 16.00 14.00  1.00  28.00  15.00  1.00
 8.00 30.00 29.00 12.00 11.00  5.00 10.00  1.00  29.00

1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
1.50  1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
2.33 -0.43  1.00  0.00  0.00  0.00  0.00  0.00  0.00
1.42  1.63 -3.77  1.00  0.00  0.00  0.00  0.00  0.00
1.58 -0.24 -0.67  0.22  1.00  0.00  0.00  0.00  0.00
1.67 -0.09 -2.08  0.40  1.72  1.00  0.00  0.00  0.00
1.33  1.30 -0.43 -0.11  1.47  1.65  1.00  0.00  0.00
2.42  0.76 -0.58  0.21 -0.04  1.36  0.55  1.00  0.00
 0.67  2.43 -5.26  0.69  4.78  2.90  1.51 -0.69  1.00

12.00  3.00  8.00  2.00  6.00  24.00  20.00  3.00  23.00
 0.00 11.50 -6.00 10.00 -4.00 -7.00 -29.00 25.50 -20.50
 0.00  0.00 -7.28  7.68 14.26 -57.04 -46.28 25.09 -59.58
 0.00  0.00  0.00 30.04 54.03 -234.81 -135.64 60.02 -216.18
 0.00  0.00  0.00  0.00 11.29 -11.32 -22.44 28.09 -26.03
 0.00  0.00  0.00  0.00  0.00 -27.02 -23.38 -8.18 -24.08
 0.00  0.00  0.00  0.00  0.00  0.00 61.20 -43.30 45.05
 0.00  0.00  0.00  0.00  0.00  0.00  0.00 26.61 -20.66
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 18.02

determinant: 349297573888.000000
[~] Time Elapsed: 0.000155 Secs

[~] The average running time was: 0.000155 secs.
radin@Radin-PC: /mnt/c/Users/Radin/Documents/MCP/HW4$ |
```

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

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