

به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش کار آزمایش دوم

برنامه نویسی چندهسته‌ای

زهره لطیفی 9923069

آیدا احمدی پارسا 9923003

موازی سازی سطری

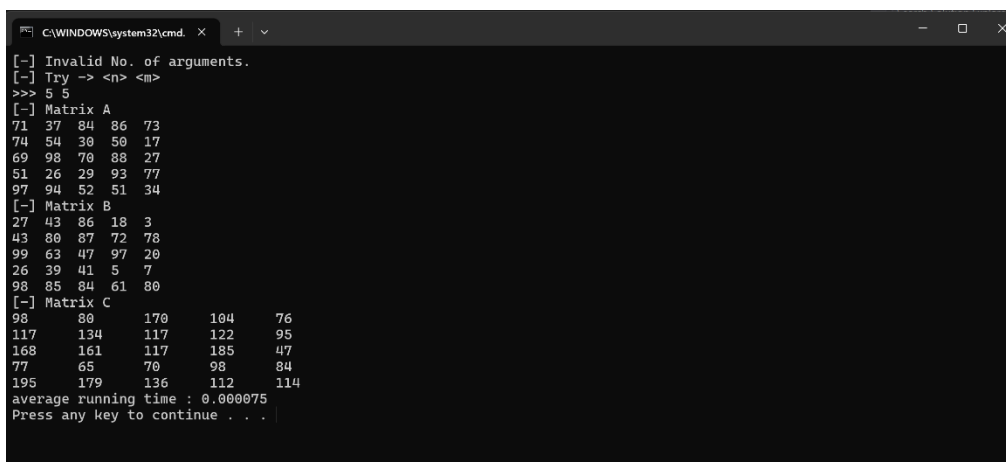
برای موازی سازی سطری می توان به این صورت عمل کرد که یک `pragma parallel` در تابع `add` پیش از حلقه اصلی به کار برد. با توجه به اینکه اولین متغیر حلقه اختصاصی در نظر گرفته می شود تنها نیاز است که متغیر `j` را `private` کنیم. درواقع این الگوریتم هر سطر از ماتریس ها را به یک نخ اختصاص می دهد و نهایتاً جمع هر دو سطر توسط یک نخ انجام می شود.

نکته دیگر برای محاسبه ابعاد ماتریس های داده شده است.

با توجه به اینکه `int` 4 بایت است، حجم های داده شده را بر 4 تقسیم می کنیم سپس از عدد بدست آمده جذر می گیریم تا ابعاد ماتریس ها بدست بیاید.

اما برای حجم 1 گیگابایت، ابعاد ماتریس $15000 * 15000$ خواهد بود که حافظه سیستم قدرت پردازش این حجم ماتریس را ندارد. در نتیجه با حداکثر ابعاد قابل پردازش ماتریس $11000 * 11000$ کار می کنیم.

```
void add1D(DataSet dataSet) {
    int i, j;
    #pragma omp parallel for private(j) num_threads(8)
    for (i = 0; i < dataSet.n; i++) {
        for (j = 0; j < dataSet.m; j++) {
            dataSet.C[i * dataSet.m + j] = dataSet.A[i * dataSet.m + j] +
            dataSet.B[i * dataSet.m + j];
        }
    }
}
```



```
C:\WINDOWS\system32\cmd. x + v
[-] Invalid No. of arguments.
[-] Try -> <n> <m>
>>> 5 5
[-] Matrix A
71 37 84 86 73
74 54 30 50 17
69 98 70 88 27
51 26 29 93 77
97 94 52 51 34
[-] Matrix B
27 43 86 18 3
43 80 87 72 78
99 63 47 97 20
26 39 41 5 7
98 85 84 61 80
[-] Matrix C
98 80 170 104 76
117 134 117 122 95
168 161 117 185 47
77 65 70 98 84
195 179 136 112 114
average running time : 0.000075
Press any key to continue . . .
```

برای اندازه‌گیری زمان اجرای برنامه از تابع `omp_get_wtime()` استفاده می‌کنیم.

به این صورت که در ابتدای اجرای تابع اصلی زمان شروع را با این تابع اندازه‌گیری کرده و در یک متغیر میانی به نام `start_time` قرار می‌دهیم. سپس در پایان اجرای تابع اصلی، مجدداً زمان را اندازه‌گیری می‌کنیم و اختلاف دو عدد بدست آمده را به عنوان زمان اجرای برنامه گزارش می‌کنیم.

با توجه به اینکه در هر بار اجرای برنامه و با ثابت نگه داشتن متغیرها زمان بدست آمده متفاوت بود، برای افزایش دقت اعداد گزارش شده، هر عملیات را 10 بار تکرار کرده و از زمان‌های بدست آمده میانگین می‌گیریم.

```
for (int i = 0; i < EXP_NUM; i++) {
    double starttime = omp_get_wtime();
    //addSerial(dataSet);
    add1D(dataSet);
    //add2D(dataSet);
    double elapsedtime = omp_get_wtime() - starttime;
    elapsed_time_sum += elapsedtime;
    //printDataSet(dataSet);
}
printDataSet(dataSet);
printf("average running time : %f\n", elapsed_time_sum / EXP_NUM);

closeDataSet(dataSet);
//system("PAUSE");
return EXIT_SUCCESS;
}
```

در نتیجه تابع `Add` را در یک حلقه 10 تایی قرار می‌دهیم. نهایتاً میانگین زمان‌های بدست آمده را پرینت می‌کنیم. همچنین برای تغییر دادن تعداد نخ‌ها از دستور `num_threads` استفاده می‌کنیم.

موازی سازی بلوکی

در این روش از دو تابع استفاده می‌کنیم.

تابع اول درواقع همان عملیات تابع سری را انجام می‌دهد با این تفاوت که محدوده حرکت کردن روی درایه‌های ماتریس را به یک عدد مشخص مبتنی بر سایز چانک‌ها محدود می‌کنیم. برخلاف عملیات سری که روی تمام کل ابعاد ماتریس انجام می‌شود.

```
void block_add(DataSet dataSet, int i, int j) {
    for (int k = i * BLOCK_SIZE; k < (i + 1) * BLOCK_SIZE && k < dataSet.n; k++) {
        for (int p = j * BLOCK_SIZE; p < (j + 1) * BLOCK_SIZE && p < dataSet.m; p++)
            dataSet.C[k * dataSet.m + p] = dataSet.A[k * dataSet.m + p] + dataSet.B[k *
            dataSet.m + p];
    }
}
```

در این تابع سطرها و ستون‌ها را به سائز چانک‌ها تقسیم می‌کنیم سپس محدوده حرکت روی درایه‌ها را به این اعداد محدود می‌کنیم.

```
void add2D(DataSet dataSet) {
    int i, j;
    int row_block = (int)ceil(dataSet.n / (double)BLOCK_SIZE);
    int col_block = (int)ceil(dataSet.m / (double)BLOCK_SIZE);
    #pragma omp parallel for private(j) num_threads(8)
    for (i = 0; i < row_block; i++) {
        for (j = 0; j < col_block; j++) {
            block_add(dataSet, i, j);
        }
    }
}
```

```
C:\WINDOWS\system32\cmd. x + v
[-] Invalid No. of arguments.
[-] Try -> <n> <m>
>>> 5 5
[-] Matrix A
46 84 24 38 32
91 92 58 96 29
28 3 60 94 76
63 3 96 39 97
98 51 80 86 45
[-] Matrix B
75 5 36 43 75
76 77 51 23 46
94 13 20 40 44
53 32 62 13 98
64 41 3 93 86
[-] Matrix C
121 89 60 81 107
167 169 109 119 75
122 16 80 134 120
116 35 158 52 195
162 92 83 179 131
average running time : 0.000102
Press any key to continue . . .
```

نتایج

نتایج بدست آمده به شرح زیر است:

نتایج روش اول

تعداد نختها	اندازه هر ماتریس ورودی				تسریع
	۱MB	۱۰MB	۱۰۰MB	۱GB	
۱	0.000167	0.001623	0.016961	0.076567	1.02413
۲	0.000157	0.001056	0.011592	0.054585	1.41631
۴	0.000145	0.00091	0.011283	0.049192	1.55860
۸	0.000176	0.000848	0.00918	0.038219	1.75918

serial 0.000145 0.001869 0.016499 0.084525

نتایج روش دوم

تعداد نختها	اندازه هر ماتریس ورودی				تسریع
	۱MB	۱۰MB	۱۰۰MB	۱GB	
۱	0.000361	0.002379	0.023259	0.116834	0.65502
۲	0.000381	0.00185	0.018038	0.086178	0.82158
۴	0.000173	0.001161	0.017335	0.066568	1.16737
۸	0.000310	0.001340	0.015989	0.060380	0.82676

ابعاد ماتریس محاسبه شده برای حجم دیتای داده شده:

1 MB : 500 * 500

10 MB : 1500 * 1500

100 MB : 5000 * 5000

1 GB : 11000 * 11000

چنان که در جدول فوق مشاهده می‌شود، در روش اول برای دیتای 1MB با افزایش تعداد نخ‌ها، مرتبه تغییر زمان اجرا در حدود 10 به توان 4- است. برای دیتای 10MB این تغییر در مرتبه 10 به توان 3- بوده و همچنین تنها تغییر از 1 به 4 نخ می‌تواند تا حدودی توجیه‌پذیر باشد. برای داده 100MB تغییر در مرتبه 10 به توان 2- است و افزایش تعداد نخ‌ها به 8 عدد توجیه‌پذیر است. در نهایت برای داده‌های 1GB با اینکه مرتبه تغییر 10 به توان 2- است اما میزان کاهش زمان در این مرتبه بیشتر و محسوس‌تر از حالت قبل است. (لازم به ذکر است با توجه به اینکه ابعاد ماتریس 11000 است حجم داده‌ها 1GB نبوده و اگر سیستم قابلیت پردازش این حجم داده را داشت، میزان تغییرات محسوس‌تر بود).

نتایج روش دوم نیز از همین الگو تبعیت می‌کند.

با اندازه‌گیری زمان اجرای برنامه سریال می‌توان میزان تسریع را بدست آورد. به این صورت که برای هر تعداد نخ زمان اجرای برنامه سریال را بر زمان اجرای موازی تقسیم کرده و نهایتاً میانگین اعداد بدست آمده را به عنوان میزان تسریع برای یک نخ گزارش می‌کنیم.

چنانچه انتظار می‌رفت میزان تسریع با افزایش تعداد نخ، افزایش می‌یابد. اما نکته قابل توجه برای مقایسه میزان تسریع دو روش سطری و بلوکی این است که اعداد بدست آمده برای روش دوم کمتر است و این نشان‌دهنده این موضوع است که روش موازی سازی بلوکی برای این حجم داده مناسب نیست.