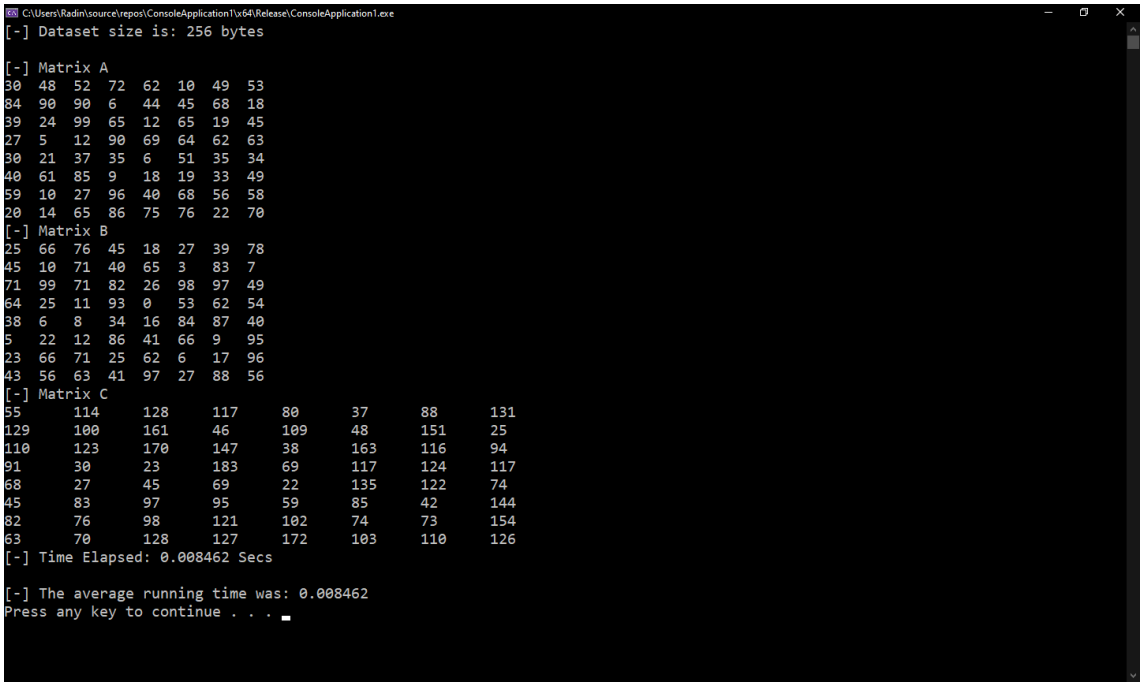
# ۱ مرحله اول

ابتدا تغییراتی در کد برنامه سریال می‌دهیم و امکاناتی مانند اندازه‌گیری زمان اجرا و زمان میانگین چند اجرا را به آن اضافه می‌کنیم. پس از انجام این تغییرات، برنامه را به صورت سریال اجرا می‌کنیم. یک نمونه از اجرای کد سریال را برای ماتریس ۸ در ۸ در شکل ۱ می‌بینیم.




شکل ۱: نمونه اجرای کد سریال

# ۲ مرحله دوم

برای تجزیه یک بعدی، با استفاده از راهنمای زیر، می‌توانیم اجراهای حلقه بیرونی را موازی کنیم. همچنین برای بهتر دیدن تجزیه، شماره نخ را به جای جمع ماتریس‌ها داخل ماتریس خروجی می‌نویسیم. حاصل نوشتن شماره نخ داخل ماتریس خروجی را در شکل ۲ می‌بینیم. **پس از اطمینان از درستی تجزیه، جهت اندازه‌گیری درست زمان اجرا مجدد برنامه را به حالت قبلی (جمع ماتریس) برمی‌گردانیم.** همچنین یک نمونه از جمع صحیح دو ماتریس با تجزیه یک بعدی در شکل ۳ نشان داده شده است.

```
#pragma omp parallel for
for (i = 0; i < dataSet.n; i++) {
    for (j = 0; j < dataSet.m; j++) {
        dataSet.C[i * dataSet.m + j] = omp_get_thread_num();
    }
}
```

```
[-] Matrix C
0       0       0       0       0       0       0       0
1       1       1       1       1       1       1       1
2       2       2       2       2       2       2       2
3       3       3       3       3       3       3       3
4       4       4       4       4       4       4       4
5       5       5       5       5       5       5       5
6       6       6       6       6       6       6       6
7       7       7       7       7       7       7       7
```

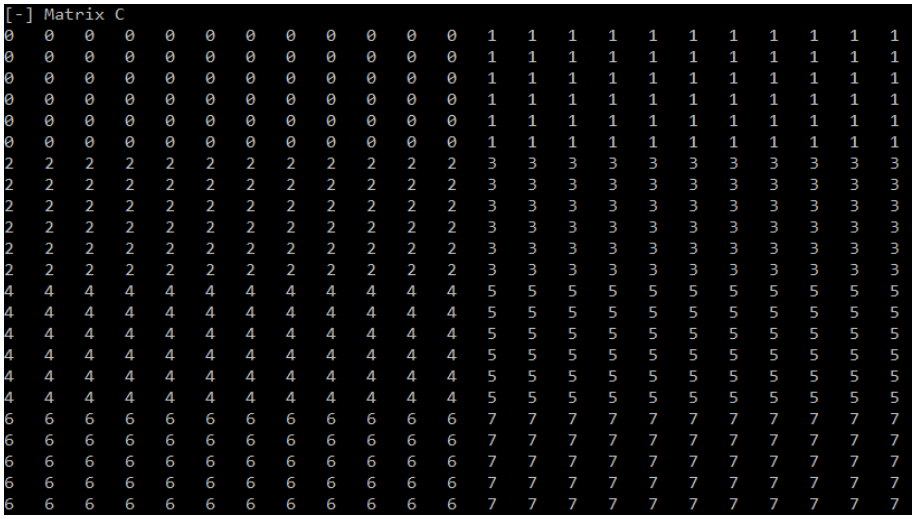
شکل ۲: نحوه تخصیص نخ‌ها در حالت تجزیه یک بعدی برای یک ماتریس ۸ در ۸

```
C:\Users\Radin\source\repos\ConsoleApplication1\x64\Release\ConsoleApplication1.exe
[-] Dataset size is: 256 bytes

[-] Matrix A
24  14  98  88  45  92  80  35
64  14  41  81  3   25  21  20
86  66  39  6   1   42  15  51
79  64  82  4   13  54  5   73
1   36  87  35  87  3   71  79
96  31  81  1   30  70  78  18
33  55  56  15  81  52  23  79
62  24  31  17  62  9   43  4
[-] Matrix B
55  53  98  19  43  6   45  32
76  1   67  18  36  25  72  58
23  48  26  70  18  35  96  40
60  98  95  60  0   50  4   46
4   85  27  2   29  98  31  99
88  55  48  67  51  29  91  53
62  3   74  17  38  98  14  44
45  34  46  97  52  45  59  58
[-] Matrix C
79      67      196     107     88      98      125     67
140     15      108     99      39      50      93      78
109     114     65      76      19      77      111     91
139     162     177     64      13      104     9       119
5       121     114     37      116     101     102     178
184     86      129     68      81      99      169     71
95      58      130     32      119     150     37      123
107     58      77      114     114     54      102     62
[-] Time Elapsed: 0.008798 Secs

[-] The average running time was: 0.008798
Press any key to continue . . .
```

شکل ۳: یک نمونه از جمع ماتریس با تجزیه یک بعدی

پس از تجزیه یک بعدی، این بار به سراغ تجزیه دوبعدی می‌رویم. برای این کار تابع دیگری به نام add2D را به کد اضافه می‌کنیم. کد این تابع در فایل **matadd.cpp** آمده است. مشابه حالت قبل، ابتدا برای دیدن

درستی تجزیه، شماره نخ را داخل ماتریس خروجی می‌نویسیم تا ببینیم هر خانه را کدام نخ پردازش می‌کند. به عنوان مثال در شکل ۴، تقسیم کار بر روی یک ماتریس ۲۳ در ۲۳ و برای هشت نخ را مشاهده می‌کنیم.




شکل ۴: نحوه تخصیص نخ‌ها در حالت تجزیه دو بعدی برای یک ماتریس ۲۳ در ۲۳

# ۳ مرحله سوم

برای اندازه‌گیری تسریع، ابتدا کد سریال (بدون هیچ راهنمای OpenMP) را به ازای اندازه ورودی‌های جدول ۱، ۱۰ بار اجرا می‌کنیم و نتایج را میانگین می‌گیریم.

جدول ۱: نتایج کد سریال

| | میانگین زمان اجرا (ثانیه) |
|---|---|
| 1MB | ۰/۰۰۰۵۳۸ |
| 10MB | ۰/۰۰۴۶۹۱ |
| 100MB | ۰/۰۴۶۰۴۶ |
| 1GB | ۰/۳۵۵۵۰۱ |

حال با تجزیه‌های یک بعدی و دو بعدی، جدول‌های ۲ و ۳ را با زمان اجرای **عمل جمع** پر می‌کنیم (برای هر خانه میانگین ۱۰ اجرا ثبت شده است). همچنین تسریع از تقسیم زمان اجرای سریال (جدول ۱) به هر یک از نتایج و میانگین گرفتن مقادیر تسریع هر سطر به دست آمده است.

همانطور که می‌بینم برنامه مقداری تسریع پیدا کرده است و در حالت تجزیه یک بعدی و با ۸ نخ، به تسریع

جدول ۲: نتایج روش اول

| تعداد نخ‌ها | میانگین زمان اجرا (ثانیه) | | | | تسریع |
|---|---|---|---|---|---|
| | 1MB | 10MB | 100MB | 1GB | |
| ۱ | ۰/۰۰۰۵۶۷ | ۰/۰۰۴۲۸۸ | ۰/۰۴۷۸۴۵ | ۰/۴۵۴۹۱۴ | ۰/۹۴ |
| ۲ | ۰/۰۰۰۵۴۵ | ۰/۰۰۲۸۲۶ | ۰/۰۲۷۶۲۳ | ۰/۲۸۳۴۲۸ | ۱/۳۹ |
| ۴ | ۰/۰۰۰۶۴۲ | ۰/۰۰۲۶۱۶ | ۰/۰۲۳۸۳۱ | ۰/۲۴۰۸۵۴ | ۱/۵۰ |
| ۸ | ۰/۰۰۰۲۳۷ | ۰/۰۰۲۲۸۹ | ۰/۰۲۱۱۷۸ | ۰/۲۳۰۹۵۰ | ۲/۰۰ |

جدول ۳: نتایج روش دوم

| تعداد نخ‌ها | میانگین زمان اجرا (ثانیه) | | | | تسریع |
|---|---|---|---|---|---|
| | 1MB | 10MB | 100MB | 1GB | |
| ۱ | ۰/۰۰۰۵۹۸ | ۰/۰۰۴۱۴۲ | ۰/۰۳۸۵۹۹ | ۰/۴۲۶۹۲۳ | ۱/۰۱ |
| ۲ | ۰/۰۰۰۵۶۶ | ۰/۰۰۲۹۵۹ | ۰/۰۲۵۹۹۳ | ۰/۲۷۵۳۶۵ | ۱/۳۹ |
| ۴ | ۰/۰۰۰۵۴۸ | ۰/۰۰۳۲۱۰ | ۰/۰۲۵۴۰۳ | ۰/۲۵۷۶۳۰ | ۱/۴۰ |
| ۸ | ۰/۰۰۰۳۱۷ | ۰/۰۰۲۸۰۹ | ۰/۰۲۲۴۳۱ | ۰/۲۱۳۱۸۳ | ۱/۷۷ |

۲ برابری دست یافته است. به نظر می‌رسد با توجه به پایین بودن شدت حسابی برنامه (تنها یک عمل جمع به ازای ۸ بایت خواندن اطلاعات یا به عبارتی شدت حسابی یک هشتم)، برنامه بیشتر memory-bound است و به همین دلیل، تسریع آن با وجود اینکه به خوبی تجزیه می‌شود خطی نیست.

با مقایسه نتایج روش اول و دوم، می‌بینیم که وضعیت کمی در روش اول (تجزیه یک بعدی) بهتر است. علت این امر می‌تواند استفاده بهتر تجزیه یک بعدی از حافظه کش باشد.