

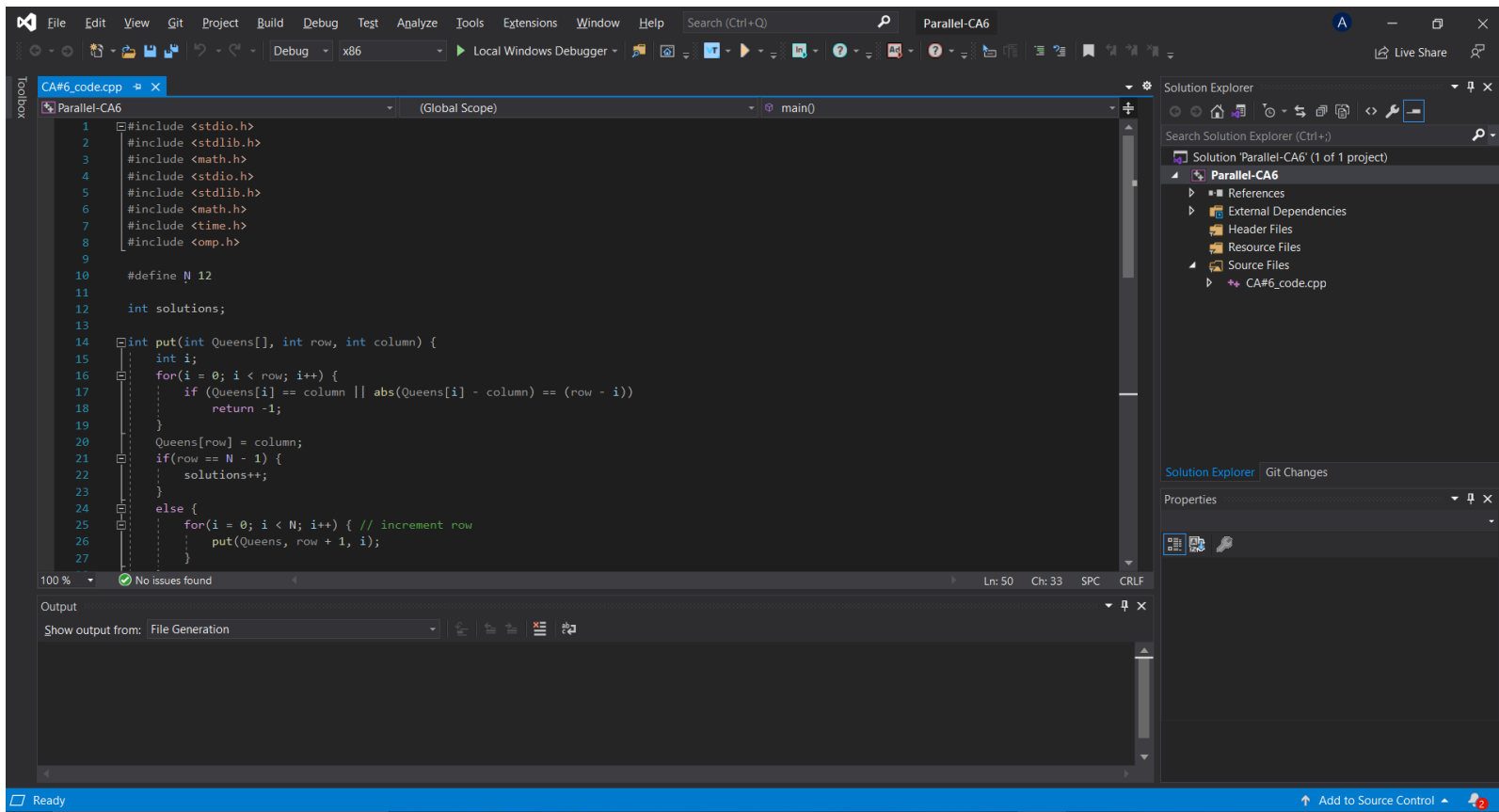
آرین حدادی ۸۱۰۱۹۶۴۴۸

ایمان مرادی ۸۱۰۱۹۶۵۶۰

تمرین کامپیوتری ششم درس برنامه نویسی موازی

مراحل انجام تمرین)

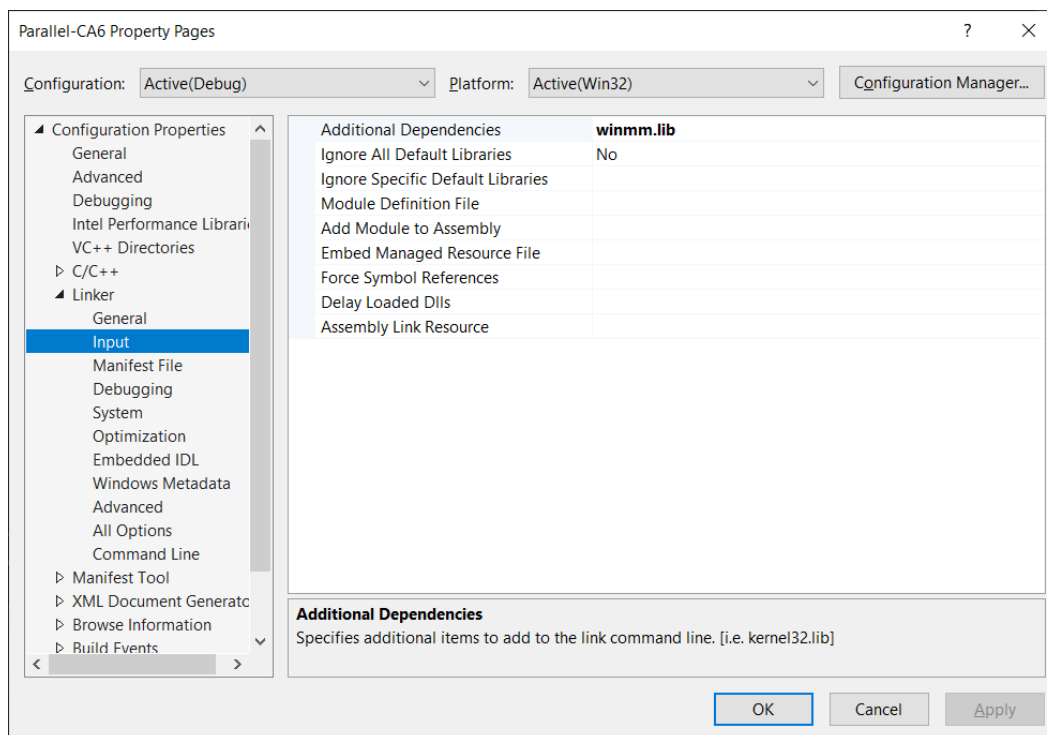
ابتدا intel parallel studio را نصب میکنیم و آن را با visual studio، integrate میکنیم.



همانطور که مشاهده می شود نوار مرتبط به ابزار های intel در visual studio قرار گرفته اند.

برای آنکه زمان گیری را دقیق تر کنیم تابع time(NULL) را با timeGetTime جایگزین می کنیم.

برای اینکار ابتدا باید winmm.lib را به dependency های پروژه اضافه کنیم.



سپس تابع main را به شکل زیر در می آوریم تا با تابع timeGetTime محاسبات را انجام دهد.

```
43 int main() {
44     int Queens[N];
45     DWORD start, end;
46
47     start = timeGetTime();
48     solve(Queens);
49     end = timeGetTime();
50
51     printf("# solutions %d time: %f milliseconds.\n", solutions, difftime(end, start));
52
53     return 0;
54 }
```

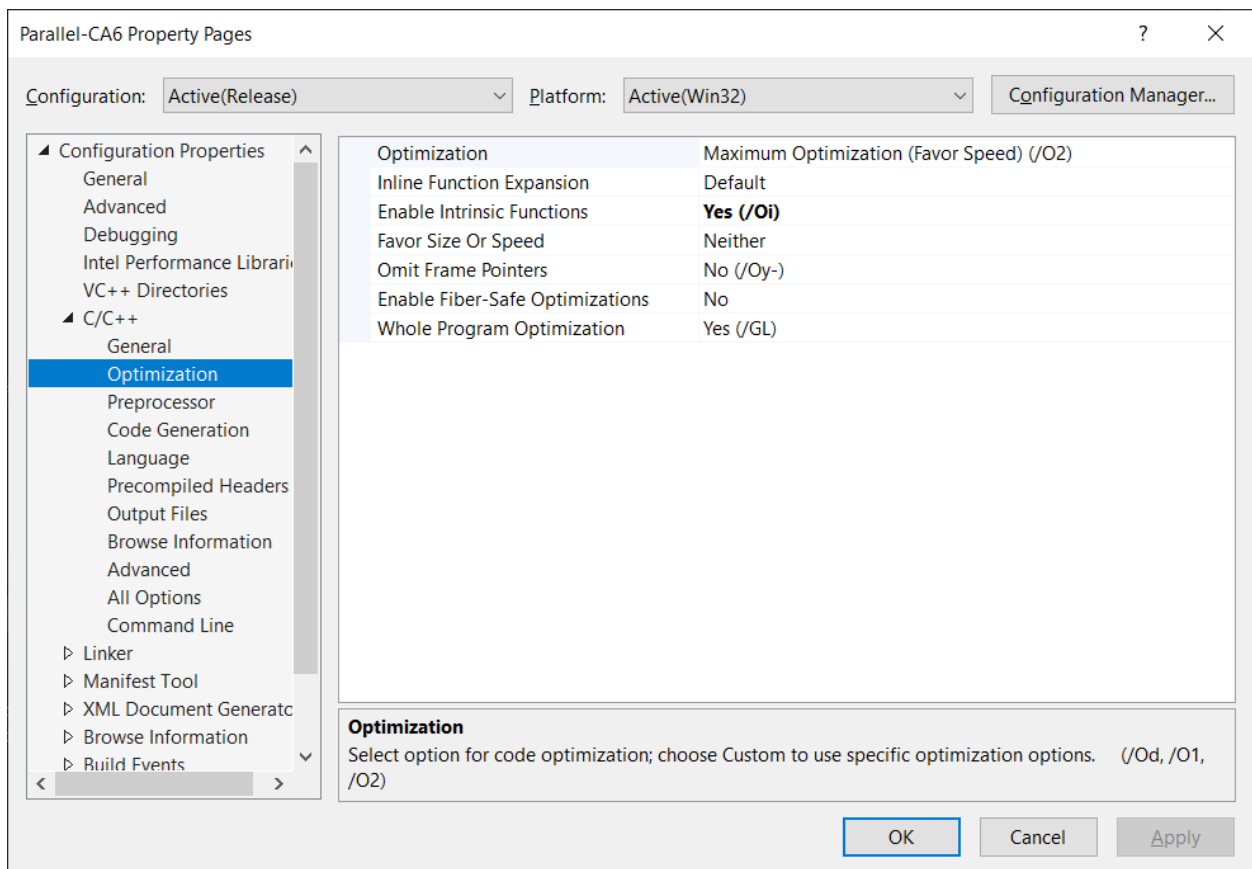
تابع timeGetTime زمان را به میلی ثانیه می دهد. سپس پروژه را در حالت Release قرار می دهیم.

یکبار این کد را ران میکنیم. خروجی زیر مشاهده می شود.

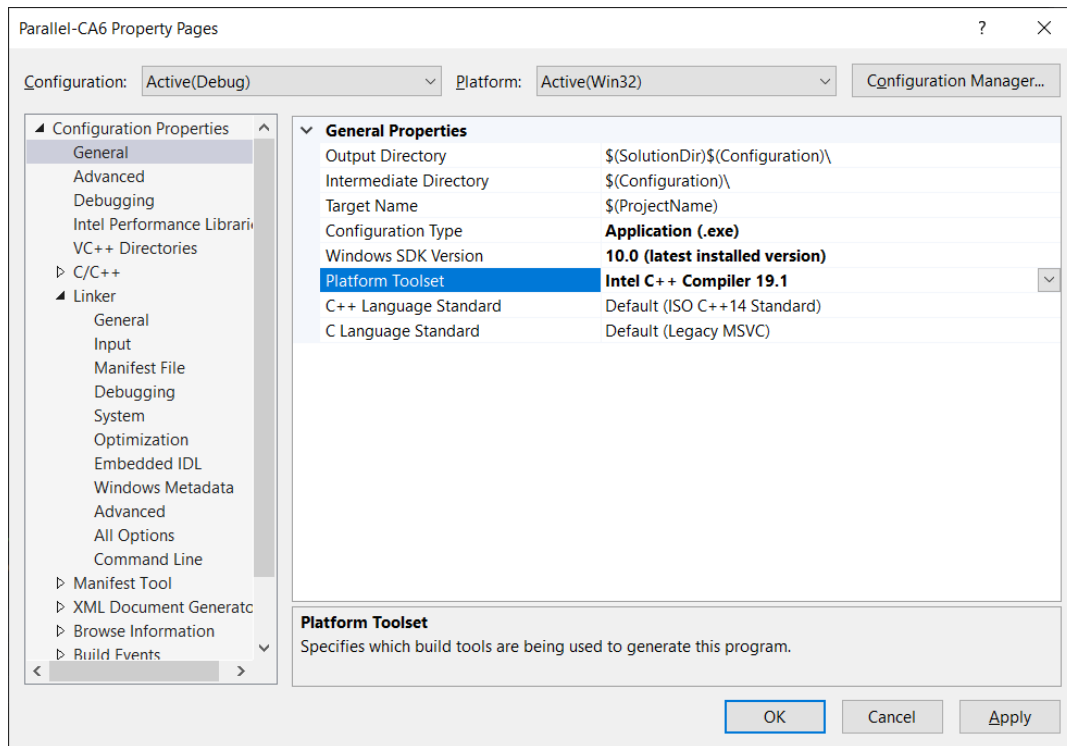
```
Microsoft Visual Studio Debug Console
Group Members:
1 - Aryan Haddadi 810196448
2 - Iman Moradi 810196560
-----
# solutions 14200 time: 119.000000 milliseconds.

E:\Codes\Visual Studio Projects\Parallel-CA6\Release\Parallel-CA6.exe (process 14464) exited with code 0.
Press any key to close this window . . .
```

توجه شود optimization برابر O2 قرار داده شده است.



حال کامپایلر پروژه را کامپایلر اینتل می کنیم.



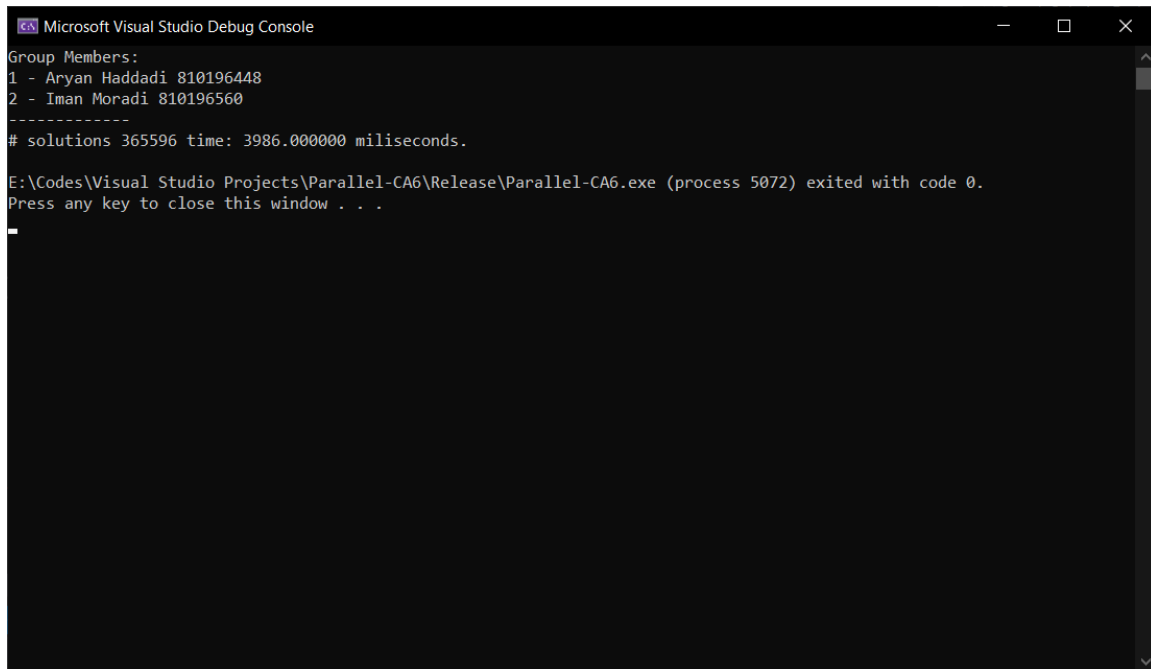
باید باز هم winmm.lib افزوده شود.

این بار با کامپایلر اینتل اجرا میکنیم. خروجی زیر مشاهده می شود.

```
Microsoft Visual Studio Debug Console
Group Members:
1 - Aryan Haddadi 810196448
2 - Iman Moradi 810196560
-----
# solutions 14200 time: 109.000000 milliseconds.

E:\Codes\Visual Studio Projects\Parallel-CA6\Release\Parallel-CA6.exe (process 1084) exited with code 0.
Press any key to close this window . . .
```

همانطور که مشاهده می شود زمان اجرا بسیار کم است. لذا N را بیشتر میکنیم تا کمی زمان اجرا بیشتر شود تا میزان تسریع هم بهتر مشخص شود. با تست های انجام شده در N از ۱۵ به بالا زمان اجرا بسیار طولانی می شود. مثلاً برای 15 تقریباً 26 ثانیه اجرا طول می کشد. اما در N برابر با ۱۴ زمان معقولی به دست می آید و لذا N را ۱۴ می کنیم. باز آن را اجرا میکنیم. خروجی زیر مشاهده می شود.

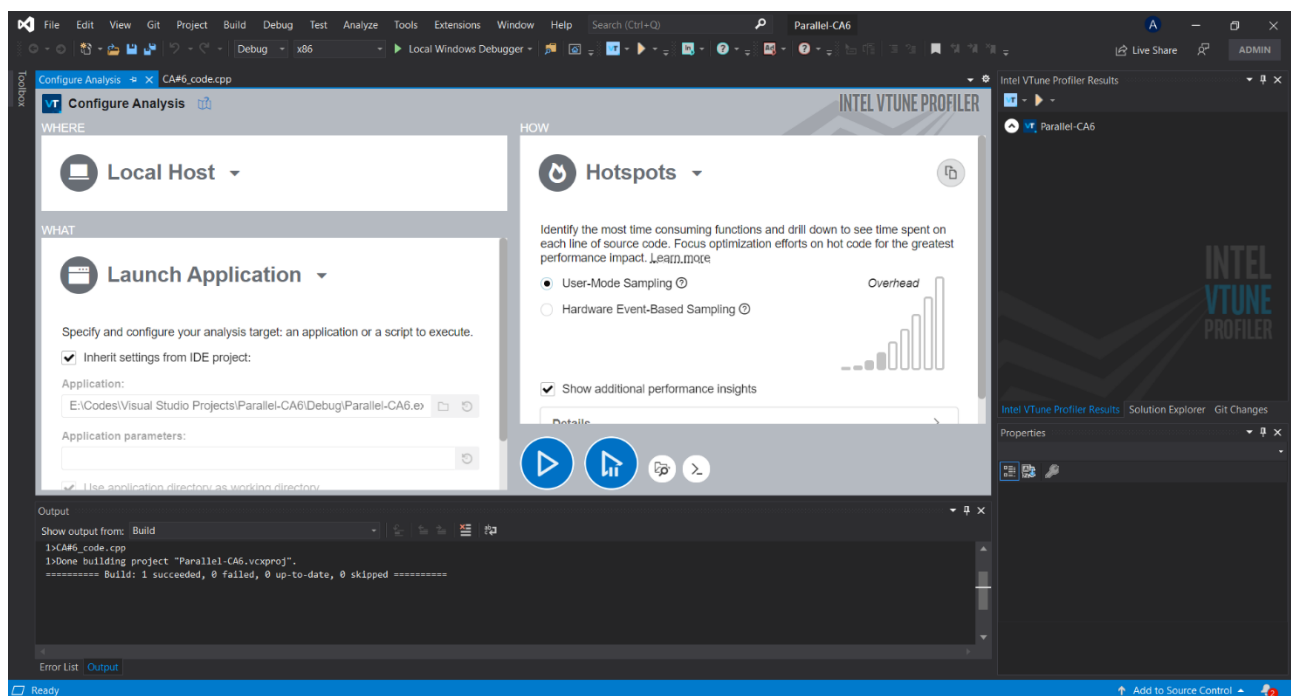


```
Microsoft Visual Studio Debug Console

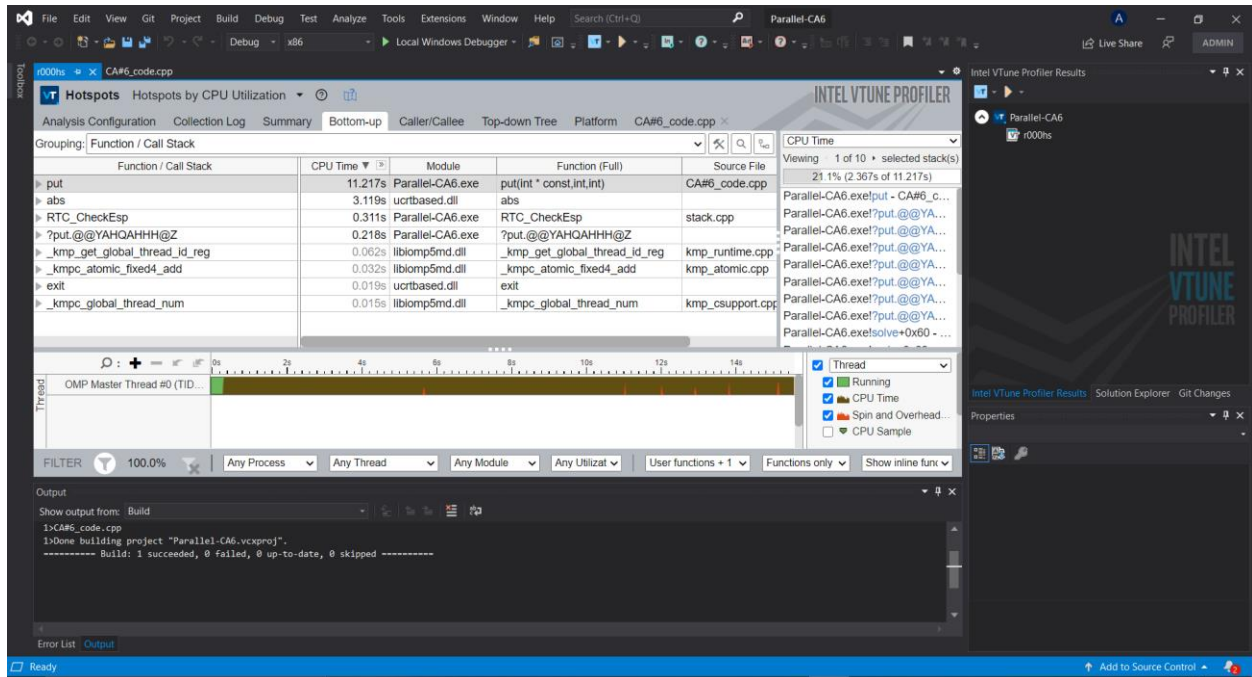
Group Members:
1 - Aryan Haddadi 810196448
2 - Iman Moradi 810196560
-----
# solutions 365596 time: 3986.000000 milliseconds.

E:\Codes\Visual Studio Projects\Parallel-CA6\Release\Parallel-CA6.exe (process 5072) exited with code 0.
Press any key to close this window . . .
```

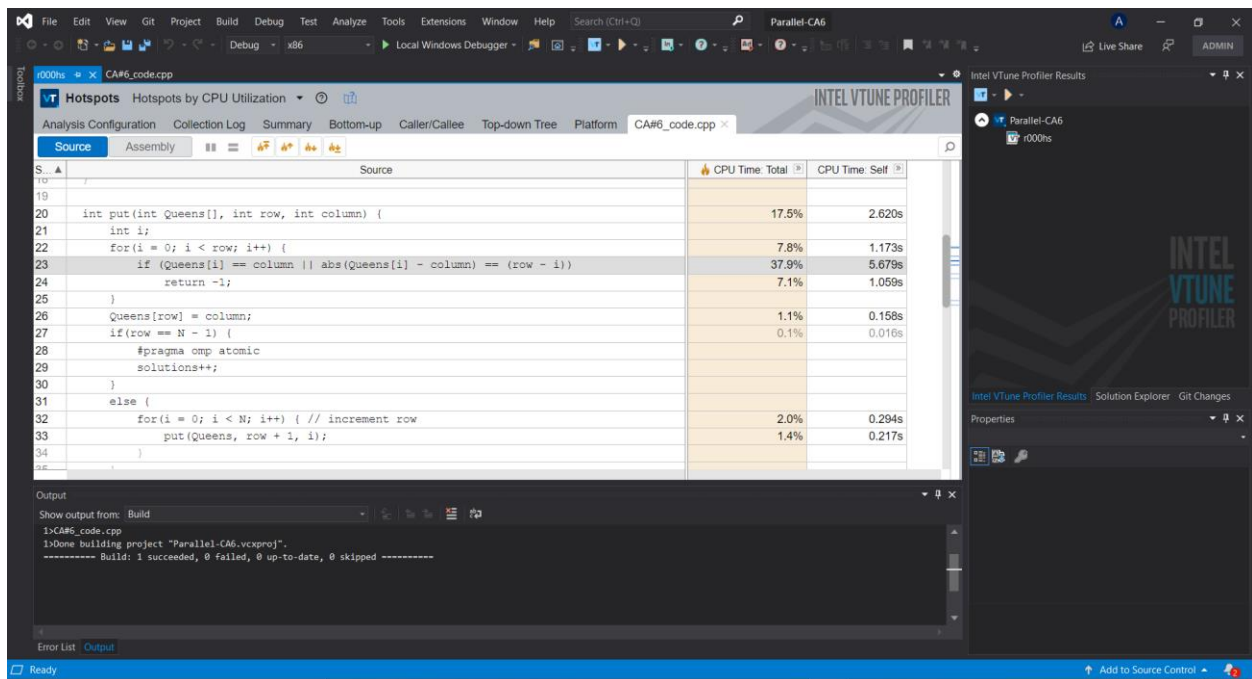
حال با استفاده از VTune Profiler از پکیج parallel studio شرکت اینتل hotspot های این برنامه را بدست می آوریم. ابتدا زمانی که میخواهیم آنالیز انجام دهیم با تصویر زیر روبرو می شویم:



دو گزینه وجود دارد که یکی user-mode sampling و hardware event-based sampling است. ابتدا با user-mode sampling کار را پیش می‌بریم. نتیجه زیر مشاهده می‌شود.

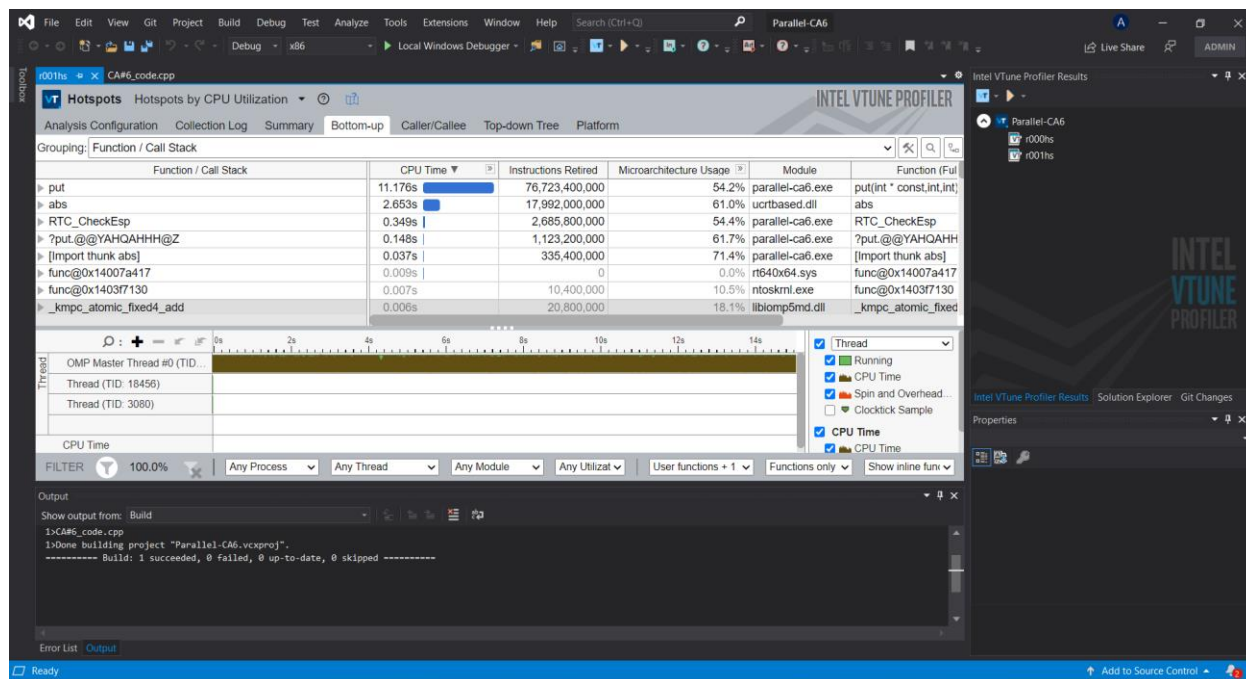


همانطور که مشاهده می‌شود hotspot اصلی برنامه تابع put است. همچنین اگر بخواهیم دقیقتر مشاهده کنیم که هر بخش کد چقدر مصرف کرده است روی put در تصویر بالا کلیک کرده و تصویر زیر را مشاهده می‌کنیم:



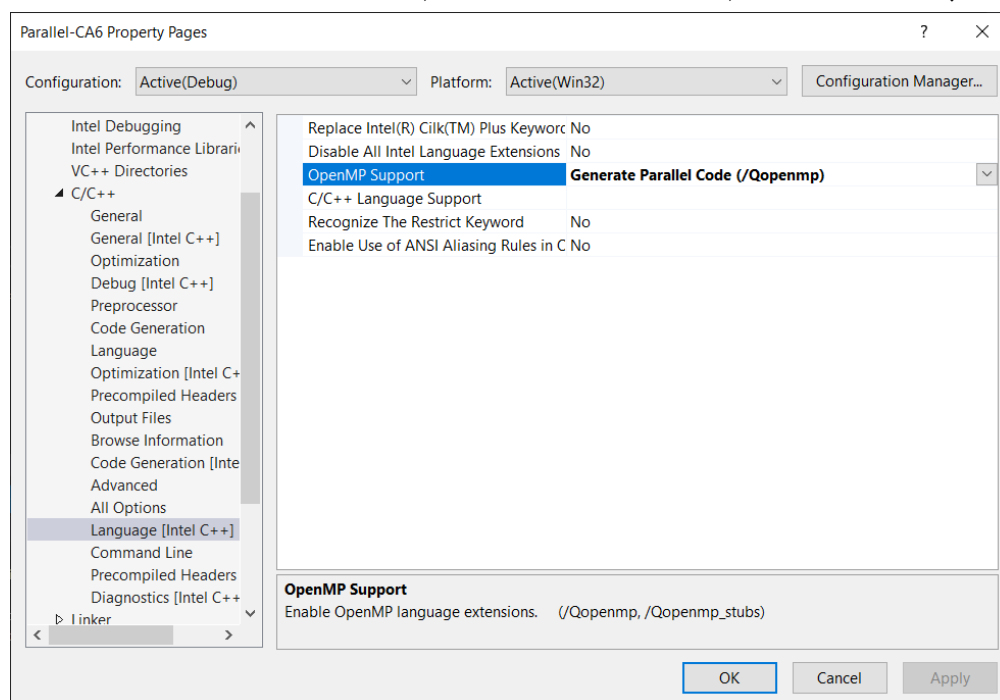
که همانطور که مشاهده می‌شود خط مربوط به چک کردن شرط تلاقی وزیری از قبل با این نقطه از نظر ستونی یا قطری بیشترین مقدار زمان اجرا را به خود اختصاص داده است.

حال روش دوم یعنی hardware event-based sampling را اجرا میکنیم. مقدار CPU sampling interval را ۱ میلی ثانیه قرار میدهیم. نتیجه زیر مشاهده می شود:



همچنان عمده برنامه تابع put بوده و در مرحله دوم نیز تابع abs قرار دارد. در تحلیل قبلی نیز چنین بود البته این تحلیل دقیق تر است.

حال برای بهتر کردن عملکرد سیستم این تابع را با استفاده از OpenMP موازی سازی میکنیم. ابتدا باید این کتابخانه را برای پروژه مان فعال کنیم. در زیر آن را فعال کرده ایم.



حال کد داده شده را با استفاده از OpenMP موازی سازی میکنیم. برای اینکار تابع solve را هدف قرار می دهیم. شکل اولیه تابع به شکل زیر است:

```
45 void solve(int Queens[]) {
46     int i;
47     for(i = 0; i < N; i++) {
48         put(Queens, 0, i);
49     }
50 }
```

این تابع، تابع put را با سطر اول (شماره صفر) و ستون های متفاوت صدا می زند. به این شکل که اولین queen که در صفحه شطرنج گذاشته می شود در سطر صفر و ستون های متفاوت باشد تا تمام حالات یافت شوند. همانطور که میتوان از منطق این تابع دریافت میتوان این بخش را موازی سازی کرد. اما اشتراک Queens ها مشکل ایجاد می کند. برای اینکار تابع solve را به شکل زیر در می آوریم:

```
void solve() {
    int i;
    #pragma omp parallel for num_threads(7)
    for (i = 0; i < N; i++) {
        int* Queens = new int[N];
        put(Queens, 0, i);
    }
}
```

به این شکل برای هر یک از iteration های توابع یک تابع جدید ایجاد می شود و لذا وابستگی داده ای بین iteration های مختلف این حلقه وجود نخواهد داشت. این جا مشکلی که ایجاد می شود در تابع put است زمانی که میخواهیم متغیر گلوبال solutions را یکی اضافه کنیم. همانطور که قبلا بحث آن شده است برای تغییر مقدار یک متغیر مشترک در شرایط solutions باید دسترسی به آن محدود شود به این نحو که تنها یک thread در حال انجام باشد. لذا باید از critical یا atomic استفاده شود که چون تنها یک statement برای افزایش solutions داریم (solutions++) از atomic استفاده می کنیم.

```
if(row == N - 1) {
    #pragma omp atomic
    solutions++;
}
```


این بار برنامه را کامپایل و اجرا می کنیم.

```
Microsoft Visual Studio Debug Console
Group Members:
1 - Aryan Haddadi 810196448
2 - Iman Moradi 810196560
-----
# solutions 365596 time: 951.000000 milliseconds.

E:\Codes\Visual Studio Projects\Parallel-CA6\Release\Parallel-CA6.exe (process 20456) exited with code 0.
Press any key to close this window . . .
```

که همانطور که مشاهده می شود تسریع قابل ملاحظه ای داشتیم و جواب نیز صحیح است. میزان تسریع برابر است با:

$$\frac{3986}{951} \cong 4.09$$

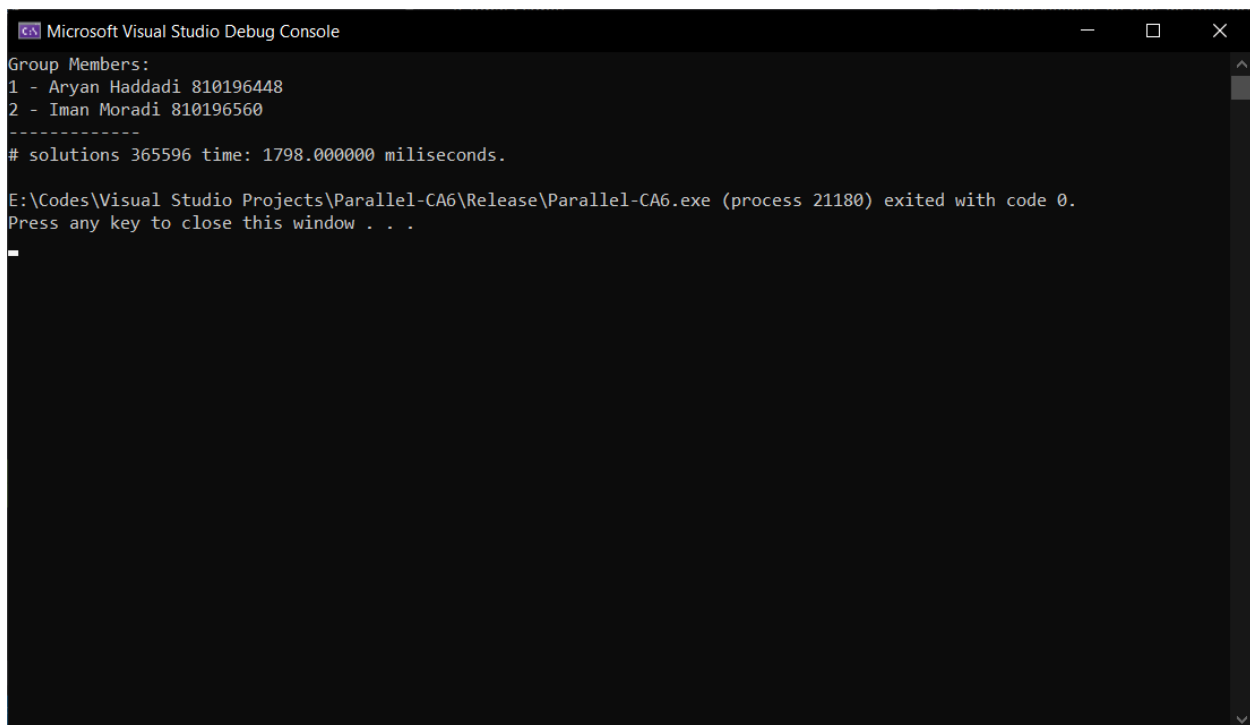
در این برنامه ۳ for دیگر نیز وجود دارد. یکی از for ها به شکل زیر است:

```
else {
    for(i = 0; i < N; i++) { // increment row
        put(Queens, row + 1, i);
    }
}
```

این for زمانی اجرا می شود که عدد سطر به انتها نرسیده باشد که باز تابع put را برای ستون های متفاوت ولی یک سطر بیشتر صدا می زند. برای آنکه این بخش را هم با استفاده از OpenMP موازی سازی کنیم آن را به شکل زیر تبدیل میکنیم.

```
else {  
    #pragma omp parallel for  
    for(i = 0; i < N; i++) { // increment row  
        put(Queens, row + 1, i);  
    }  
}
```

اما نتیجه زیر را دریافت می کنیم.



```
Microsoft Visual Studio Debug Console  
Group Members:  
1 - Aryan Haddadi 810196448  
2 - Iman Moradi 810196560  
-----  
# solutions 365596 time: 1798.000000 milliseconds.  
E:\Codes\Visual Studio Projects\Parallel-CA6\Release\Parallel-CA6.exe (process 21180) exited with code 0.  
Press any key to close this window . . .
```

همانطور که مشاهده می شود ایجاد تعداد زیاد thread ها نتیجه را بدتر کرده است و لذا این for را به حالت اولیه باز میگردانیم.

سومین for هم که داریم for زیر است:

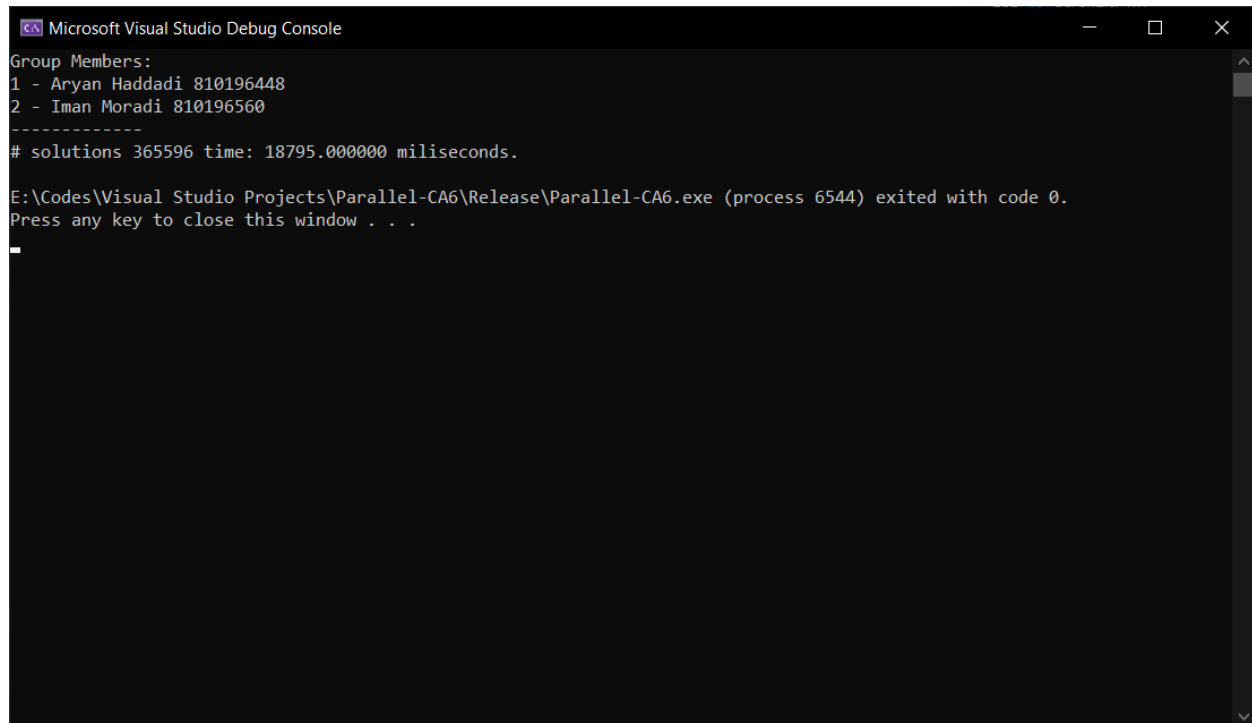
```
for(i = 0; i < row; i++) {  
    if (Queens[i] == column || abs(Queens[i] - column) == (row - i))  
        return -1;  
}
```

این for بررسی میکند تا سطر فعلی هیچ وزیری در این ستون یا در قطر این خانه (با توجه به row و column) قرار نداشته باشد. چون در این for عبارت return وجود دارد برای آنکه با ایجاد چندین ثرد مشکلی برای آن رخ

ندهد یک متغیر مشترک به نام hit تعریف میکنیم که در این نقطه true می شود و سپس بعد از انتهای کار تمام thread ها مقدار این متغیر خوانده می شود. کد به شکل زیر تبدیل می شود.

```
#pragma omp parallel for num_threads(row)
for(i = 0; i < row; i++) {
    if (Queens[i] == column || abs(Queens[i] - column) == (row - i))
        hit = true;
}
if (hit) return -1;
```

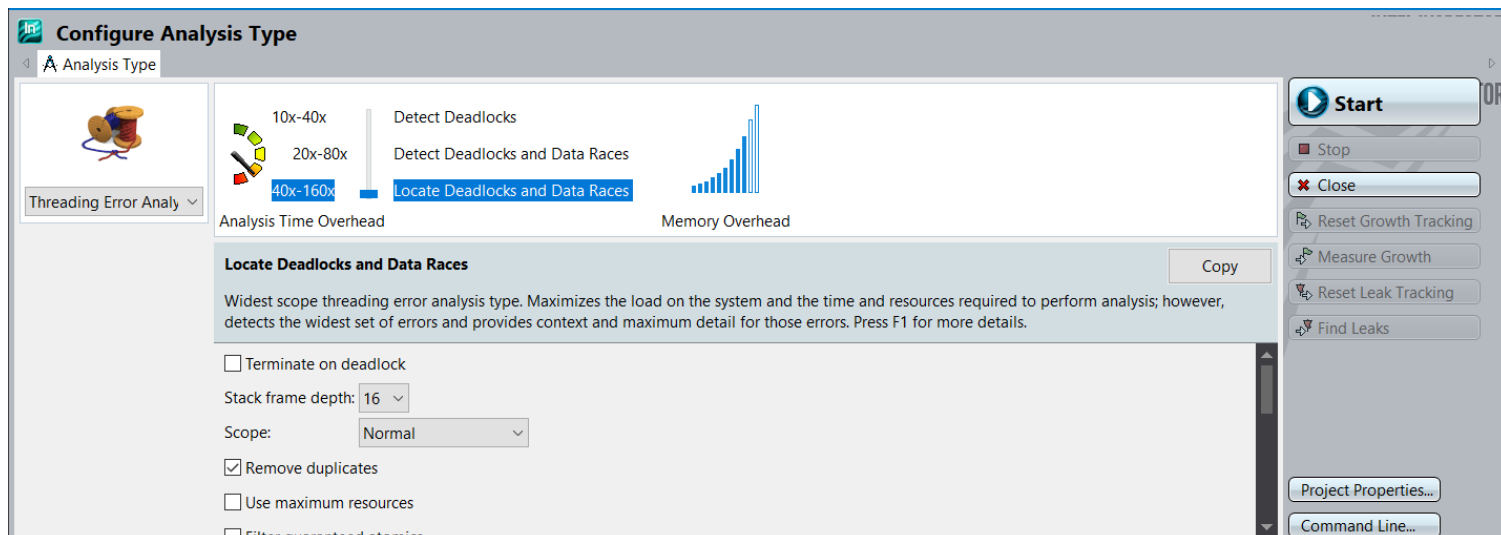
حال اجرا میکنیم و خروجی زیر مشاهده می شود.



```
Microsoft Visual Studio Debug Console
Group Members:
1 - Aryan Haddadi 810196448
2 - Iman Moradi 810196560
-----
# solutions 365596 time: 18795.000000 milliseconds.
E:\Codes\Visual Studio Projects\Parallel-CA6\Release\Parallel-CA6.exe (process 6544) exited with code 0.
Press any key to close this window . . .
```

همانطور که دیده می شود تولید تعداد زیاد thread ها باعث رخ دادن overhead بالایی شده و نتیجه بدتر شده در عین اینکه نتیجه صحیح است لذا این بخش را نیز parallel نمیکنیم.

حال به سراغ inspector می رویم. ابتدا به مد debug می رویم. سپس inspector را با تنظیمات زیر اجرا می کنیم.



خروجی زیر مشاهده می شود:

```
E:\Codes\Visual Studio Projects\Parallel-CA6\Debug\Parallel-CA6.exe
Group Members:
1 - Aryan Haddadi 810196448
2 - Iman Moradi 810196560
-----
# solutions 365596 time: 78389.000000 milliseconds.
Press any key to continue . . .
```

در inspector هم این خروجی مشاهده می شود که یعنی مشکلی نبوده است:

The screenshot shows the Intel Inspector interface with a data race error detected. The main window displays the 'Problems' tab with a table of issues. A single issue is listed: a 'Data race' in 'CA#6_code.cpp' at 'parallel-ca6.exe', marked as 'New'. The 'Filters' panel on the right shows the severity as 'Error', type as 'Data race', source as 'CA#6_code.cpp', and module as 'parallel-ca6.exe'. The 'Code Locations: Data race' panel shows two write operations to the 'solutions' variable in 'CA#6_code.cpp' at line 29, one from 'parallel-ca6.exe!put' and another from 'parallel-ca6.exe!put'. The 'Timeline' panel shows two threads: 'OMP Worker Thread #3 (8264)' and 'OMP Worker Thread #4 (16296)'. The bottom status bar shows 'Ready' and 'Add to Source Control'.

ID	Type	Sources	Modules	State
P1	Data race	CA#6_code.cpp	parallel-ca6.exe	New

Severity	Count
Error	1 item(s)

Type	Count
Data race	1 item(s)

Source	Count
CA#6_code.cpp	1 item(s)

Module	Count
parallel-ca6.exe	1 item(s)

Description	Source	Function	Module	Variable
Write	CA#6_code.cpp:29	put	parallel-ca6.exe	solutions
Write	CA#6_code.cpp:29	put	parallel-ca6.exe	solutions

Timeline:

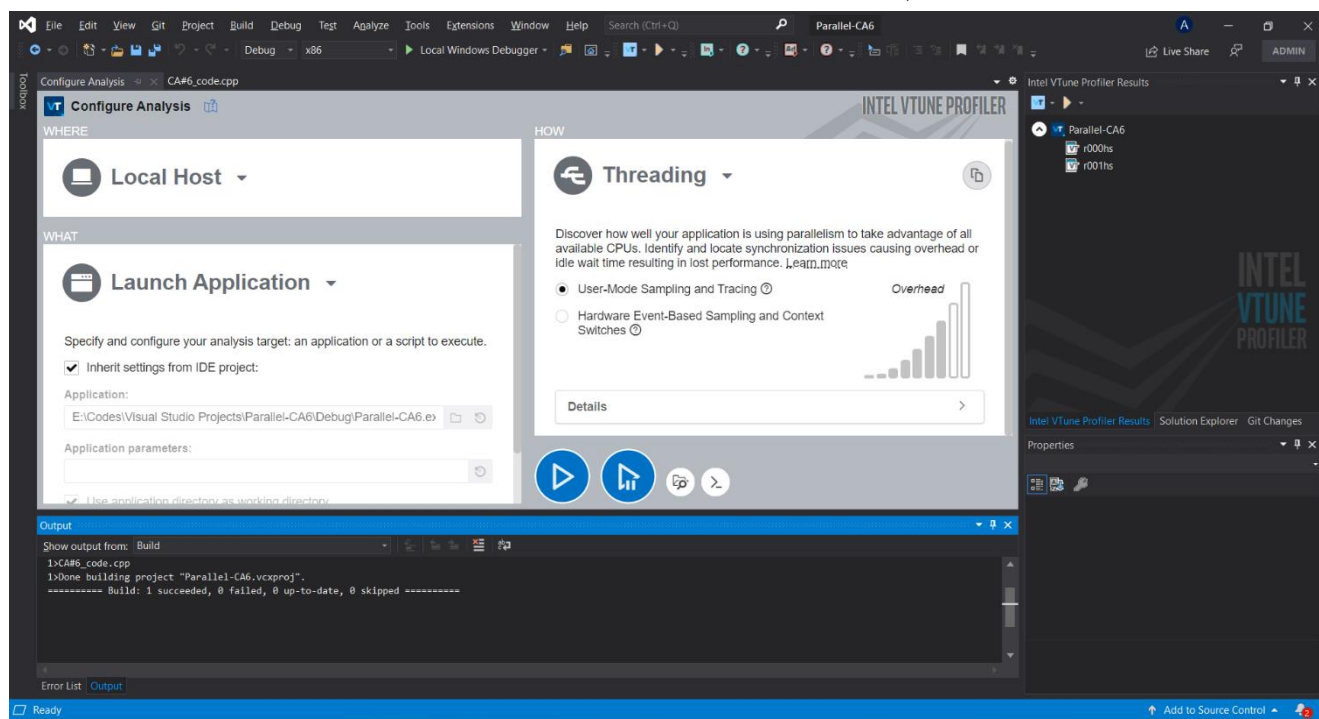
- OMP Worker Thread #3 (8264)
- OMP Worker Thread #4 (16296)

برای اینکه نمونه ارور هم مشاهده شود برای مثال بخش atomic را حذف میکنیم تا دسترسی به متغیر مشترک

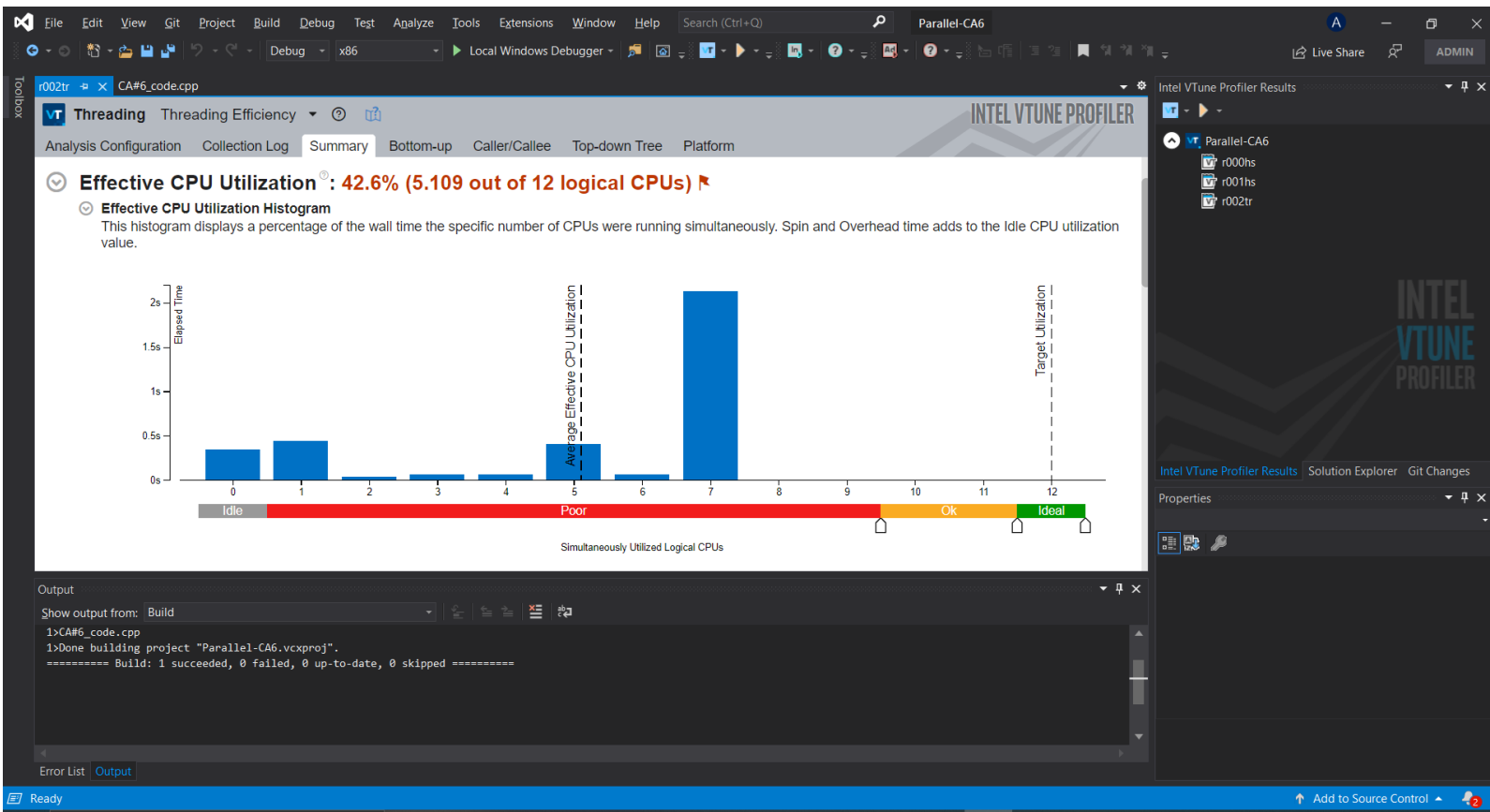
solutions محدود نشود. خروجی زیر مشاهده می شود:

همانطور که مشاهده می شود بخاطر کامنت کردن `#pragma omp atomic` امر `data race` رخ داده است که با توجه به توضیحات قبلی مورد انتظار بود.

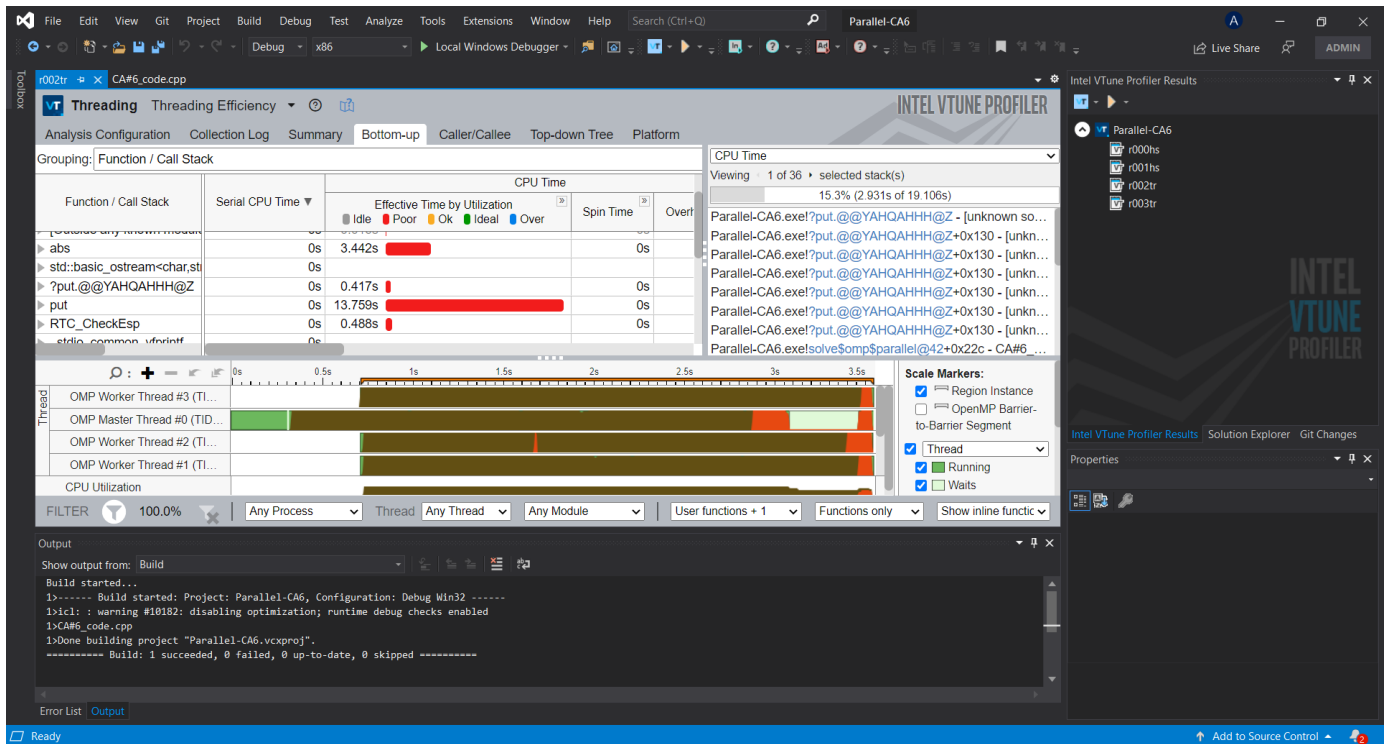
حال به سراغ tuning می رویم. این بار باز هم به سراغ VTune Profiler رفته ولی این بار به جای hotspots از Threading استفاده میکنیم.



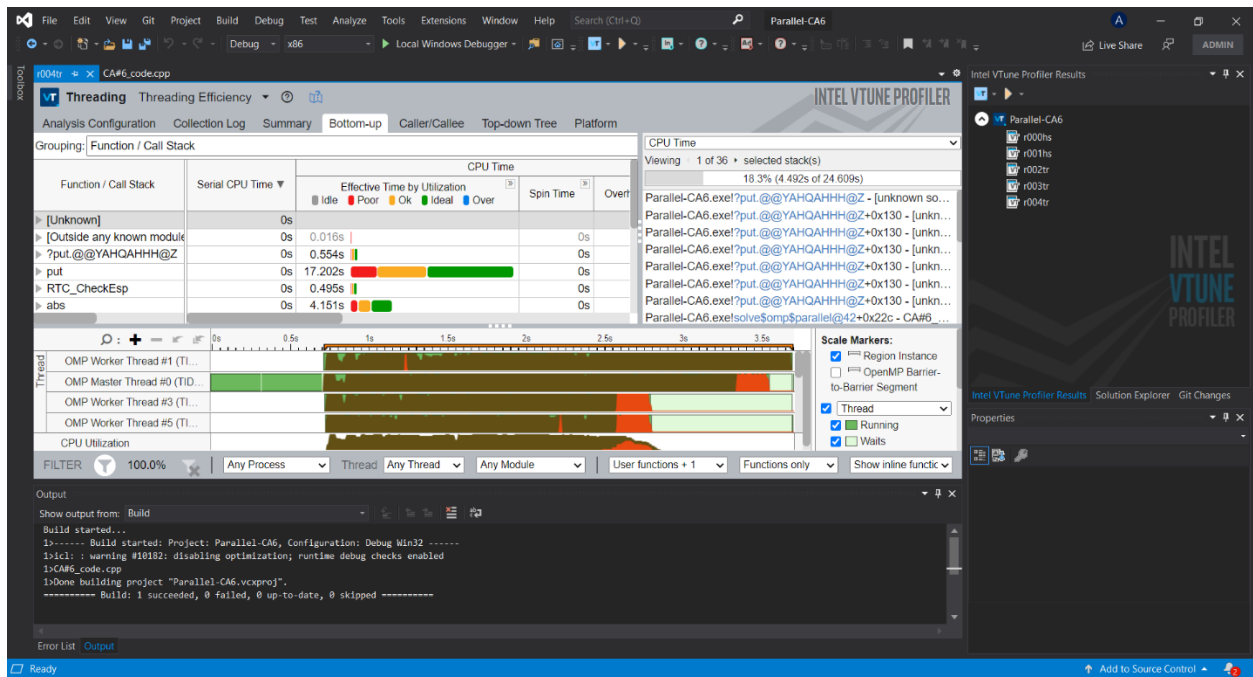
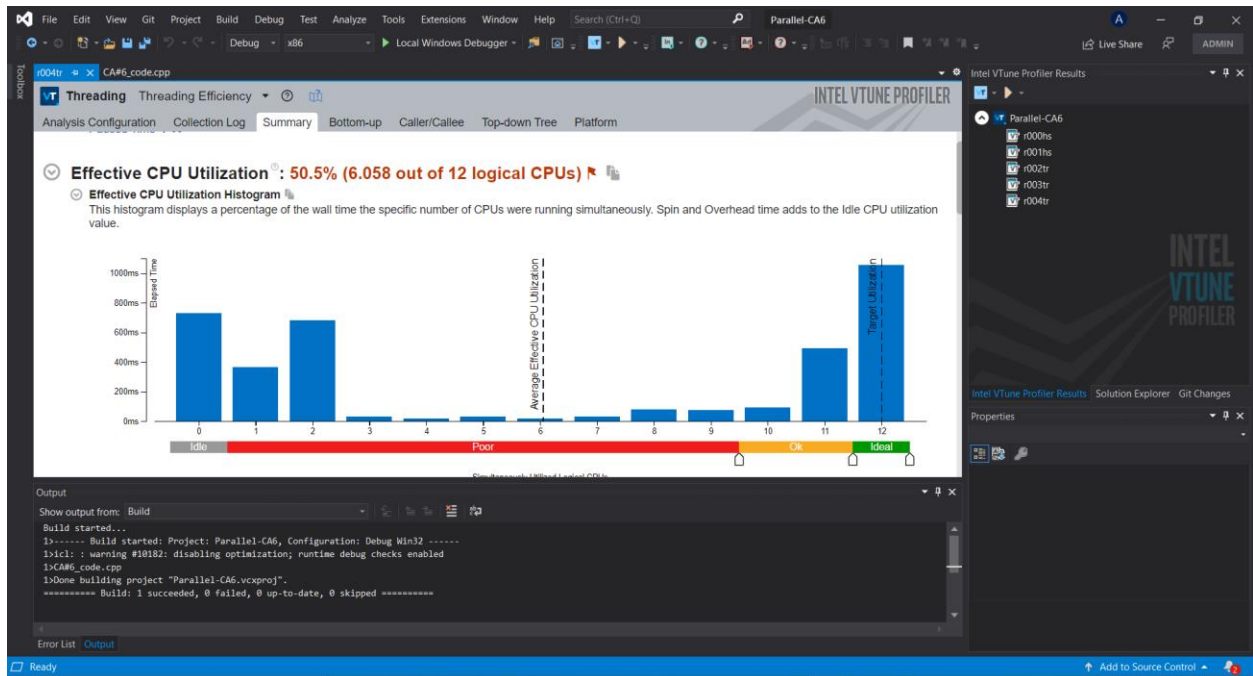
تصویر زیر مشاهده می شود.



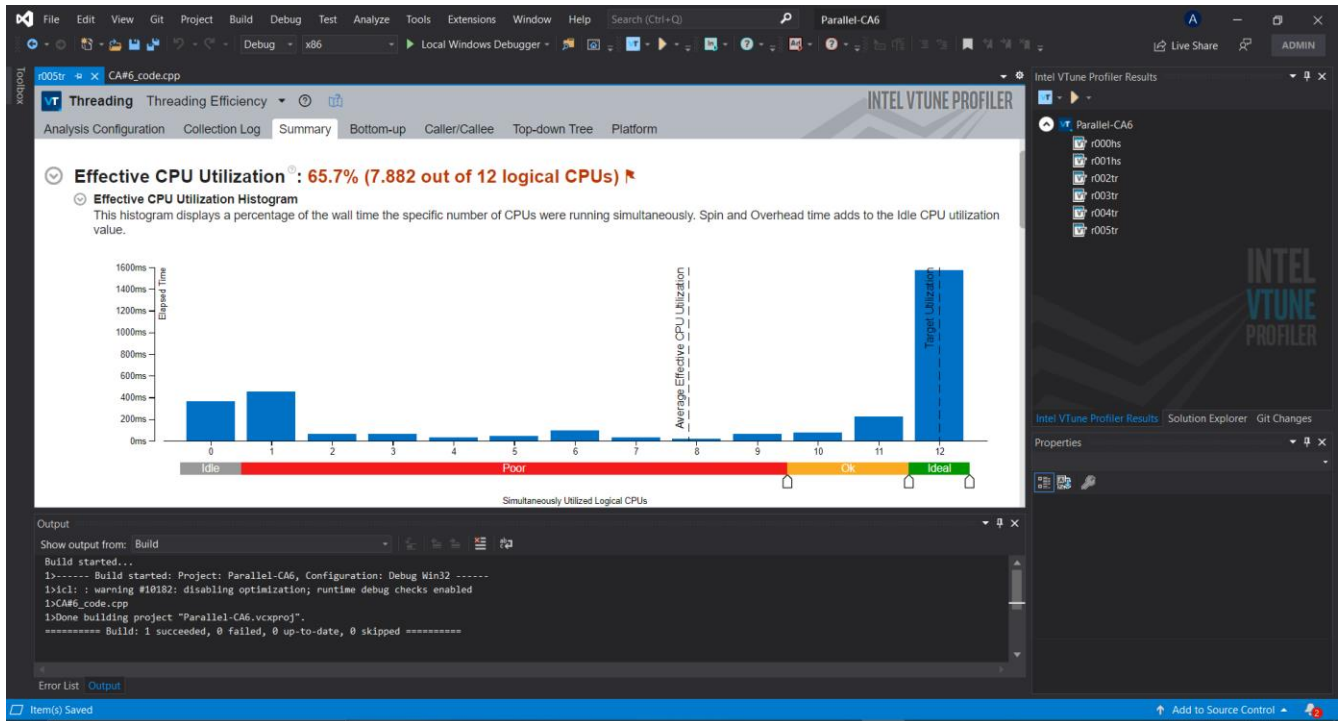
همانطور که مشاهده می شود وضعیت utilization مناسب نیست اگر گزارش bottom-up را ببینیم:



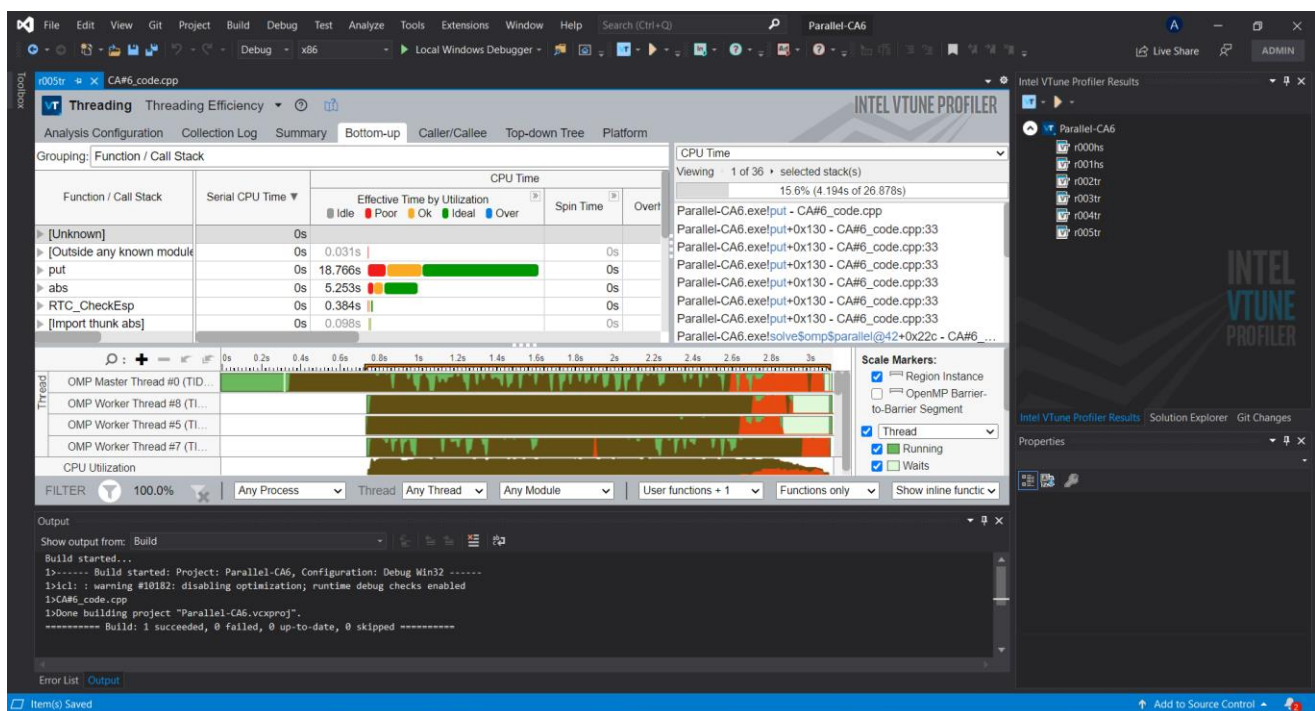
همانطور که مشاهده می شود وضعیت Effective Time By Utilization مناسب نیست. تعداد هسته های پردازنده پی سی ران کنند برابر ۱۲ تاست و با ۷ ثرد به وضعیت مطلوبی برای utilization نمی رسیم. لذا تعداد ثرد ها را بیشتر میکنیم. مثلا برابر تعداد هسته های logical که ۱۲ تاست قرار میدهم. وضعیت کمی بهتر شد.



اما همچنان چون مقدار N برابر ۱۴ است، چون ۱۲ ترد داریم به دو تا از ترد ها مقدار بیشتری می رسد. پس بهتر است تعداد ترد ها را برابر N یعنی ۱۴ تا کنیم که هر ترد به سراغ یکی از iteration ها برود. همانطور که انتظار میرفت خروجی بهتر شد:



شکل مرتبط با bottom-up هم به شکل زیر شد:



که نسبت به تصویر bottom-up قبلی به وضوح میزان ideal یا نوار سبز آن بیشتر است. در نهایت با تمام اصلاحات انجام شده یکبار دیگر در حالت release برنامه را اجرا میکنیم.

```
Microsoft Visual Studio Debug Console
Group Members:
1 - Aryan Haddadi 810196448
2 - Iman Moradi 810196560
-----
# solutions 365596 time: 751.000000 milliseconds.

E:\Codes\Visual Studio Projects\Parallel-CA6\Release\Parallel-CA6.exe (process 21552) exited with code 0.
Press any key to close this window . . .
```

میزان speedup نسبت به حالت اولیه بدون OpenMP برابر شد با

$$\frac{3986}{751} \cong 5.31$$

البته توجه شود در ران های متفاوت عدد ها هم تفاوت هایی خواهند داشت. برای داشتن دقت بیشتر باید چندین بار برنامه را ران کرد. برای مثال ران آخر که در عکس بالا ۷۵۱ میلی ثانیه است چندین بار اجرا شده است و حول و حوش همین مقدار آمده است.

	زمان اجرا (ms)	میزان تسریع نسبت به حالت سریال
Serial	3986	-
Parallel	951	4.09
Tuned Parallel	751	5.31

CPU: Intel Core i7-9750H 6 Cores, 12 Threads