



Parallel Programming CA 06

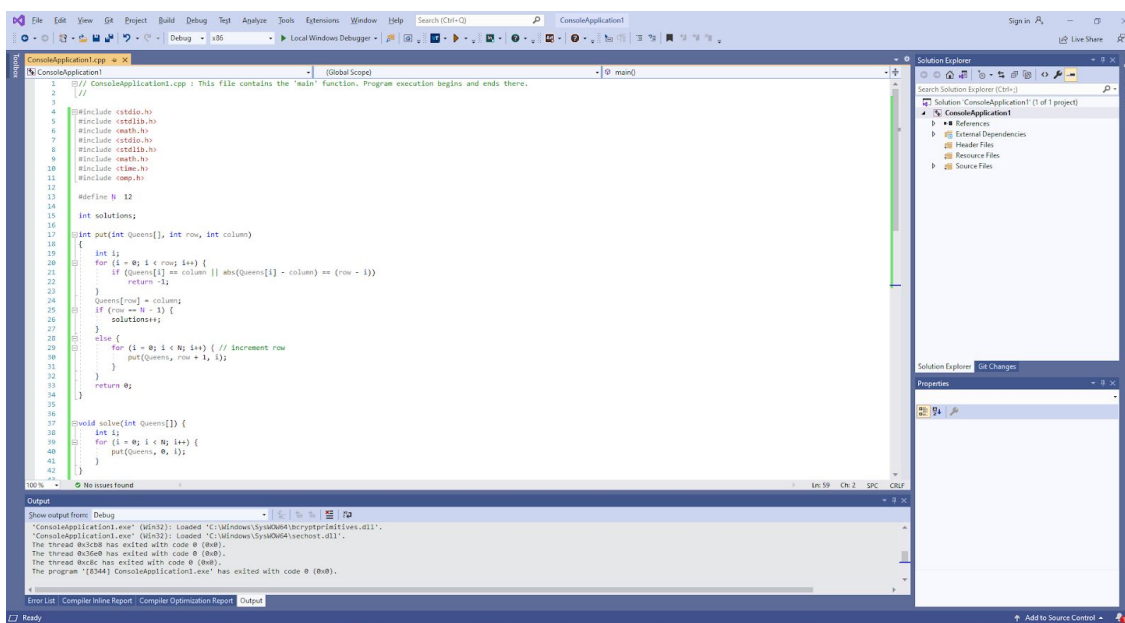
PREPARED BY

Mohsen Fayyaz - 810196650

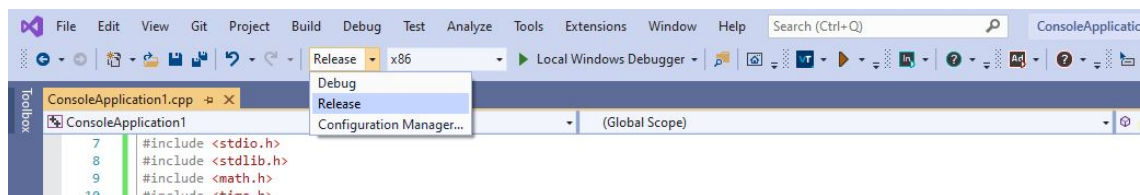
Mahdi Hadiloo - 810196587

1 مقدمه

ابتدا ویژوال استودیو و پارالل استودیو که برای این تمرین لازم بودند نصب شدند.



یک پروژه console application خالی ساخته شد
محتوای فایل cpp داده شده در آن قرار گرفت.
و ورژن Release پروژه انتخاب شد.



و سپس دیده شد که تایمیری که گذاشته شده در اجرا دقت کافی را ندارد و 0 می دهد.

```

Microsoft Visual Studio Debug Console

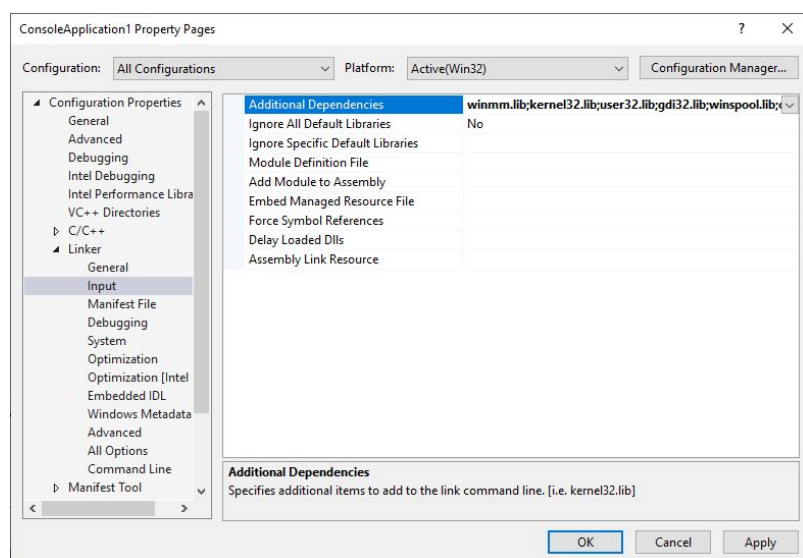
# solutions 14200 time: 0.000000

F:\Programming\University\07_Parallel_Programming\CA\
onsoleApplication1.exe (process 8152) exited with cod
To automatically close the console when debugging sto
le when debugging stops.
Press any key to close this window . . .

```

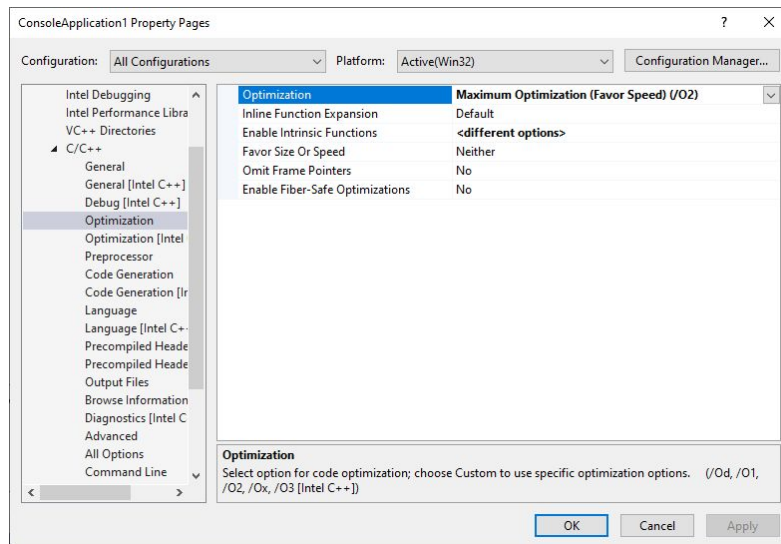
به همین دلیل آنرا جایگزین کردیم.

سپس winmm.lib را اضافه کردیم تا از timeGetTime() استفاده کنیم.

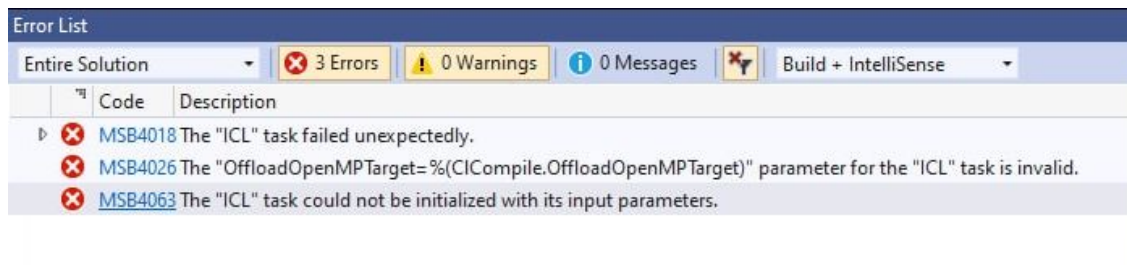


در این میان کد چاپ کردن اسم ها را نیز اضافه کردیم.

سپس بهینه سازی را روی حداکثر سرعت قرار دادیم.



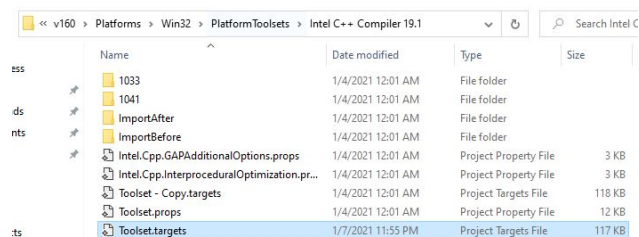
برای تغییر کامپایلر به کامپایلر اینتل ارور می گرفتیم.



در لینک زیر هم توضیح داده شده که نسخه آخر VS این مشکل را دارد که با یکی از راه حل های ارائه شده برطرفش کردیم.

<https://community.intel.com/t5/Intel-C-Compiler/VC-2019-16-8-does-not-work-with-Intel-C-2020-4-311/td-p/1227169>

فایل Toolset.targets را با فایل تغییر داده شده جایگزین کردیم.



و سپس دو حالت کامپایلر را امتحان کردیم.

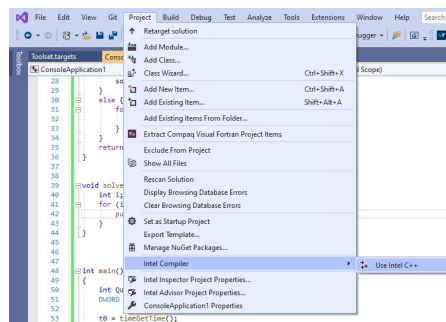
کامپایلر Visual cpp

```
Microsoft Visual Studio Debug Console

Mohsen Fayyaz - Mahdi Hadiloo
810196650 - 810196587
# solutions 14200 time: 132.000000

F:\Programming\University\07_Parallel_Programming\CA\git\Pa
onsoleApplication1.exe (process 2528) exited with code 0.
To automatically close the console when debugging stops, en
le when debugging stops.
Press any key to close this window . . .
```

تغییر کامپایلر به کامپایلر اینتل



```
Microsoft Visual Studio Debug Console

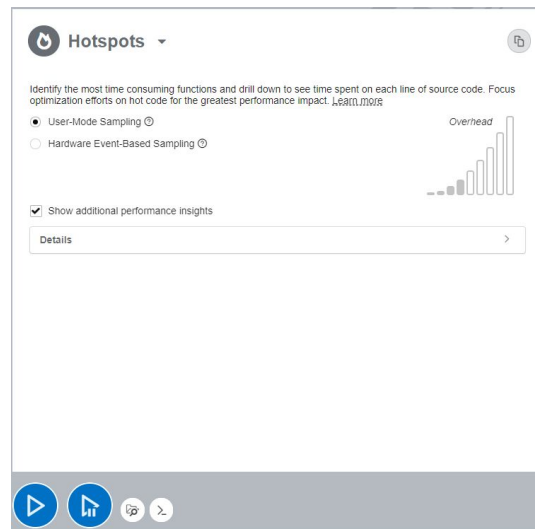
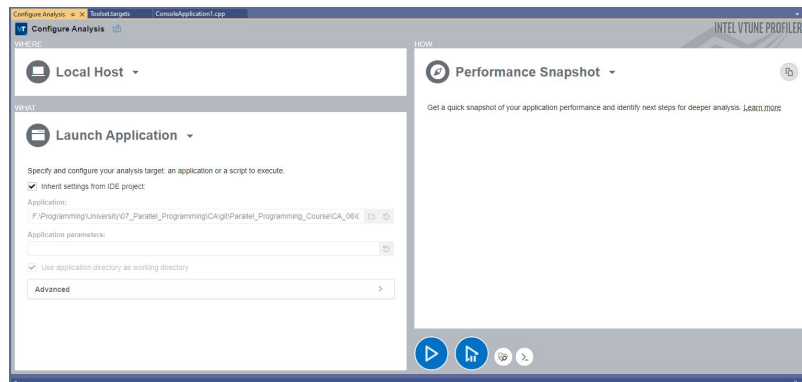
Mohsen Fayyaz - Mahdi Hadiloo
810196650 - 810196587
# solutions 14200 time: 104.000000

F:\Programming\University\07_Parallel_Programming\CA\git
onsoleApplication1.exe (process 10844) exited with code
To automatically close the console when debugging stops,
le when debugging stops.
Press any key to close this window . . .
```

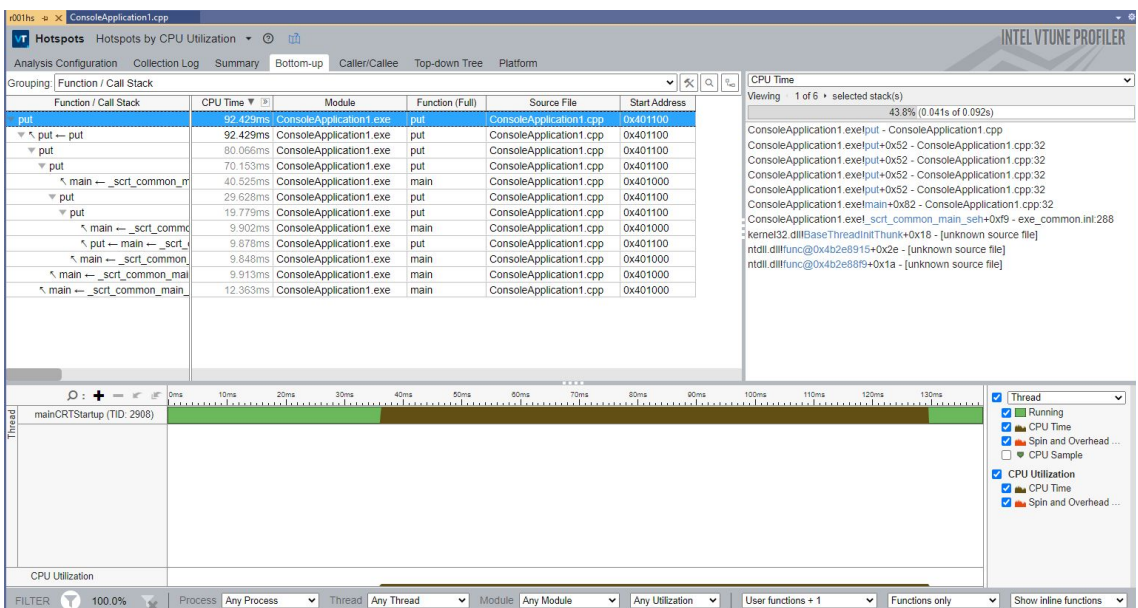
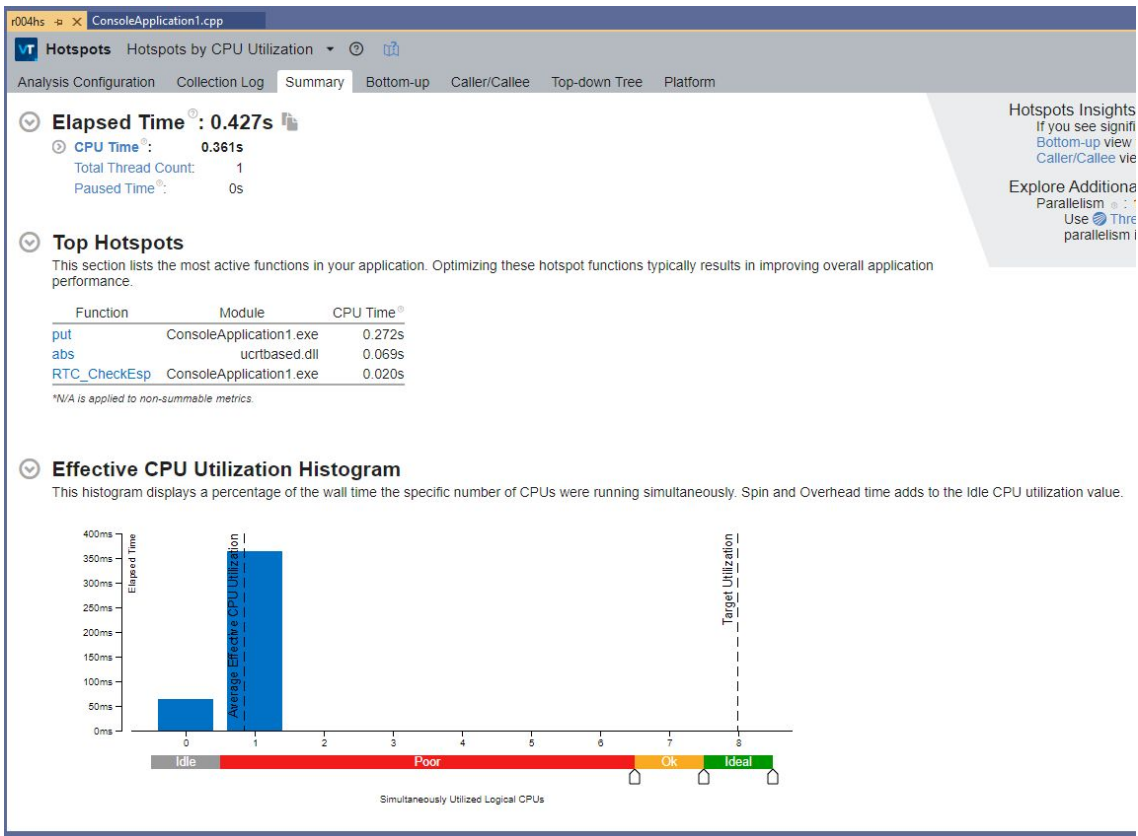
دیده می شود که زمانها نزدیک به هم هستند و اینتل کمی سریع تر است.

2 به دست آوردن Hotspot ها

یک آنالیز vtune جدید ایجاد می کنیم.



در نتیجه خلاصه زیر می بینیم که هشدار داده شده چون زمان اجرا خیلی کم است ممکن است تحلیل ها دقت بالایی نداشته باشند. و دیده می شود که هات اسپات اصلی برنامه تابع `put` تشخیص داده شده است.



همچنین تحلیل خط به خط کد در زیر آمده است.

19	int put(int Queens[], int row, int column)		
20	{	11.0%	39.728ms
21	int i;		
22	for (i = 0; i < row; i++) {	2.8%	9.986ms
23	if (Queens[i] == column abs(Queens[i] - column) == (row - i))	36.8%	132.813ms
24	return -1;	8.3%	29.806ms
25	}		
26	Queens[row] = column;	2.8%	9.992ms
27	if (row == N - 1) {	2.8%	9.927ms
28	#pragma omp atomic		
29	solutions++;		
30	}		
31	else {		
32	for (i = 0; i < N; i++) { // increment row	5.4%	19.659ms
33	put(Queens, row + 1, i);	75.3%	19.820ms
34	}		
35	}		
36	return 0;		
37	}		

سپس حالت تحلیل را روی Hardware Event-Based Sampling قرار دادیم که نوشته هم شده بود برای Profiles shorter than a few seconds از این حالت بهتر است استفاده شود و نتایج زیر را دیدیم.

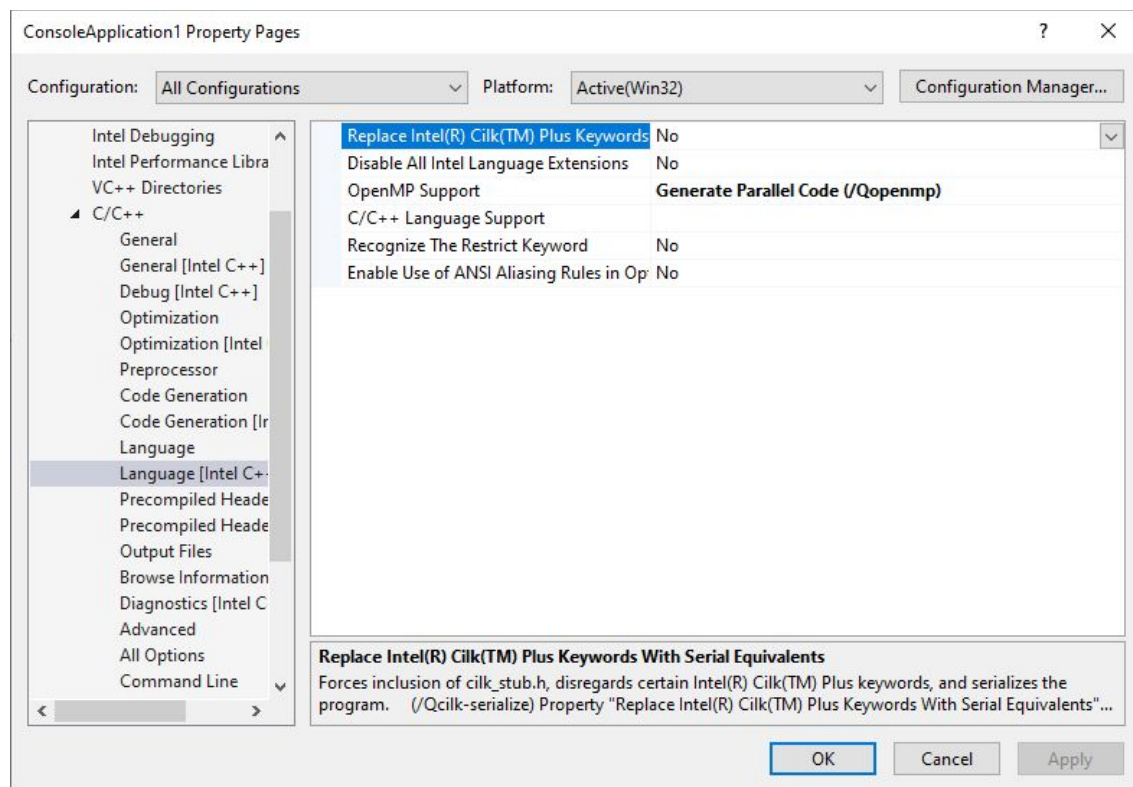
19	int put(int Queens[], int row, int column)		
20	{	11.000ms	108,000,000
21	int i;		
22	for (i = 0; i < row; i++) {	19.000ms	180,000,000
23	if (Queens[i] == column abs(Queens[i] - column) == (row - i))	38.000ms	198,000,000
24	return -1;	16.000ms	154,800,000
25	}		
26	Queens[row] = column;	0ms	3,600,000
27	if (row == N - 1) {	1.000ms	0
28	solutions++;	19.000ms	108,000,000
29	}		
30	else {		
31	for (i = 0; i < N; i++) { // increment row	0ms	36,000,000
32	put(Queens, row + 1, i);	3.000ms	25,200,000
33	}		
34	}		
35	return 0;	3.000ms	21,600,000
36	}		

هات اسپات ها به ترتیب دیده می شوند که ابتدا شرط بزرگ کد در خط 23 است و سپس باقی موارد که آمده مثل خود حلقه و یا اضافه کردن solutions. طبق عکس اول باید فراخوانی put در خط 32 را نیز در نظر داشت.

به این ترتیب هات اسپات ها به دست آمدند و تابع put باید موازی سازی شود.

3 موازی سازی کد

ابتدا openMP را فعال می کنیم.



برای این کار از openMP استفاده کردیم که در زیر دیده می شود.

ابتدا برای اینکه race پیش نیاید روی اضافه کردن متغیر solutions از atomic استفاده می کنیم.

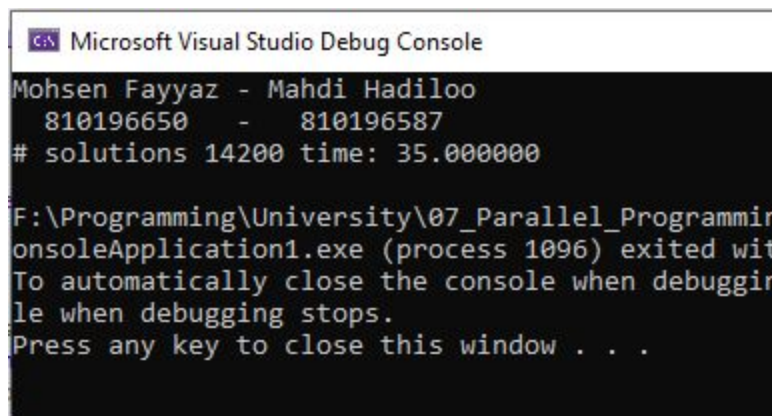
```
if (row == N - 1) {  
    #pragma omp atomic  
    solutions++;  
}
```

و موازی سازی را روی بزرگترین حلقه برنامه که در زیر آمده انجام می‌دهیم.

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

باید دقت داشت که در بالا به جای Queens یک int جدید را می‌دهیم، چون باید هر ترد روی دیتای خودش کار کند تا race پیش نیاید و پاسخ اشتباه نگیریم.

خروجی این تغییرات و نتیجه موازی سازی در زیر آمده است.



```
Microsoft Visual Studio Debug Console  
Mohsen Fayyaz - Mahdi Hadiloo  
810196650 - 810196587  
# solutions 14200 time: 35.000000  
  
F:\Programming\University\07_Parallel_Programming\ConsoleApplication1.exe (process 1096) exited with  
To automatically close the console when debugging  
le when debugging stops.  
Press any key to close this window . . .
```

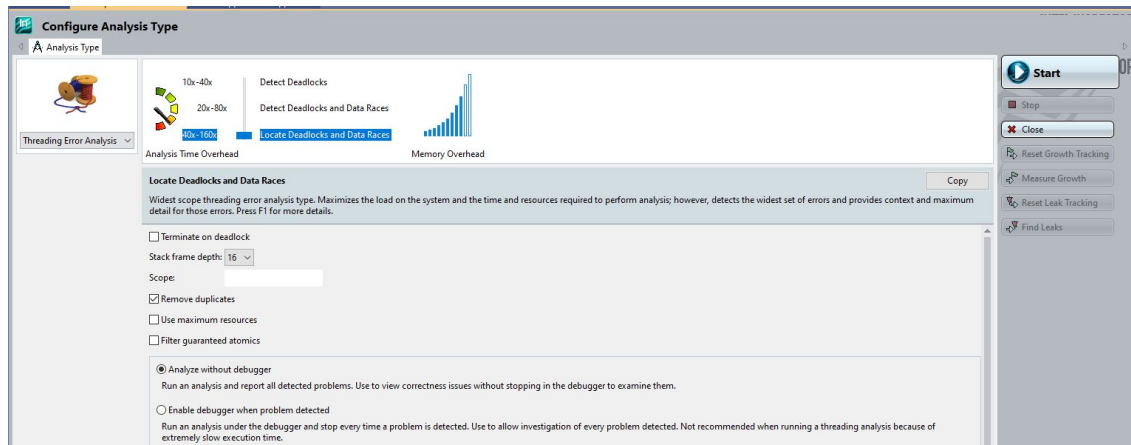
تصویر حالت سریال برنامه در بالاتر آمده بود که برابر 104 بود.

$$\text{Speedup} = 104/35 = 2.97$$

که نشان می‌دهد حدوداً 3 برابر تسریع گرفته‌ایم.

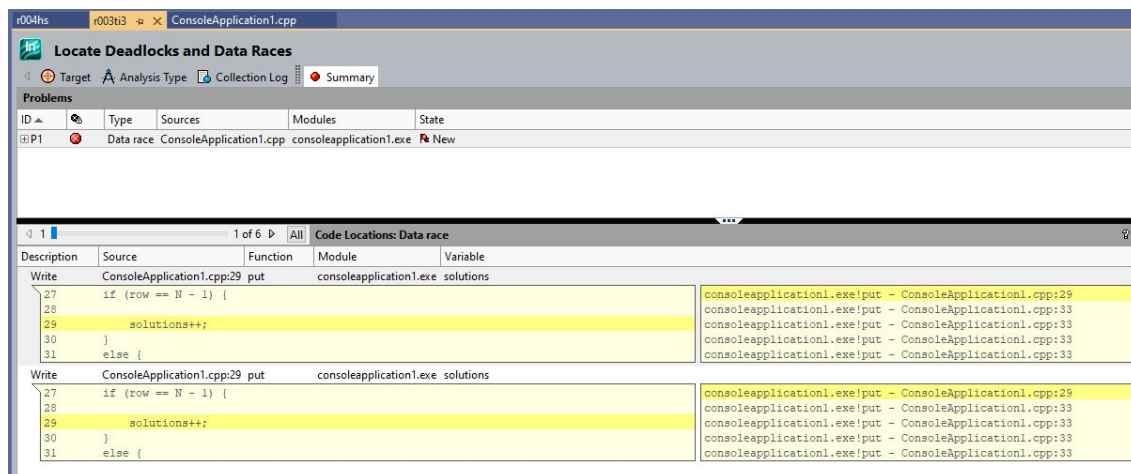
4 یافتن عیب های احتمالی با Inspector

می خواهیم ددلاک یا ریس را پیدا کنیم.

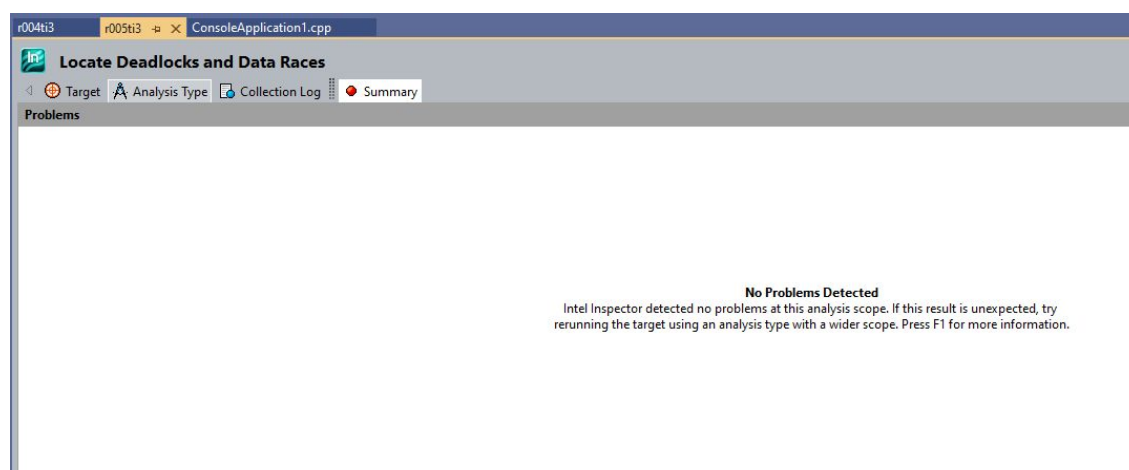


چون کد با دقت نوشته شده بود اشکال اساسی در کد موازی نبود.

به همین دلیل برای نمونه اگر اتمیک بودن را از روی متغیر مشترک به صورت عمدی برداریم و inspector را اجرا کنیم و می بینیم که دقیقاً مشخص می کند کدام بخش و کدام متغیر است.

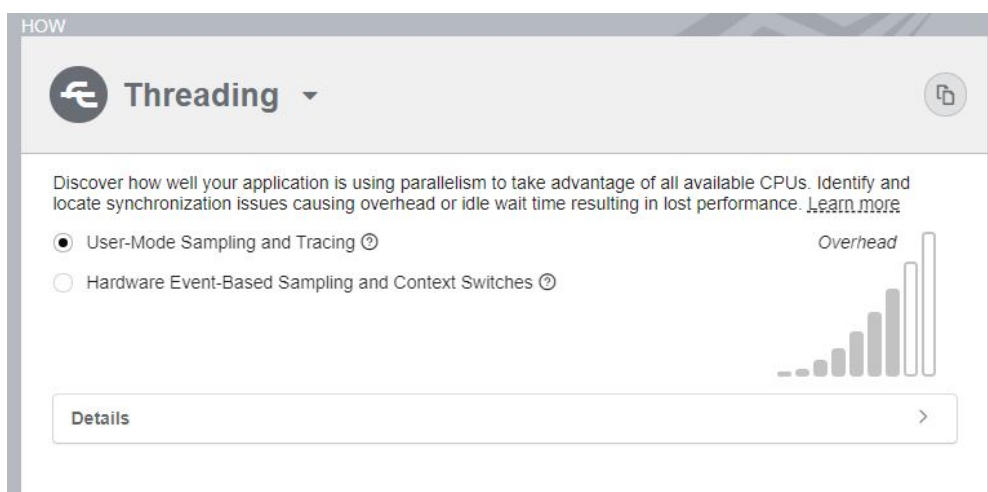


که برای برطرف کردنش دوباره اتمیک که از قصد برداشتیم را بر می گردانیم و می بینیم که دیگر مشکلی وجود ندارد.



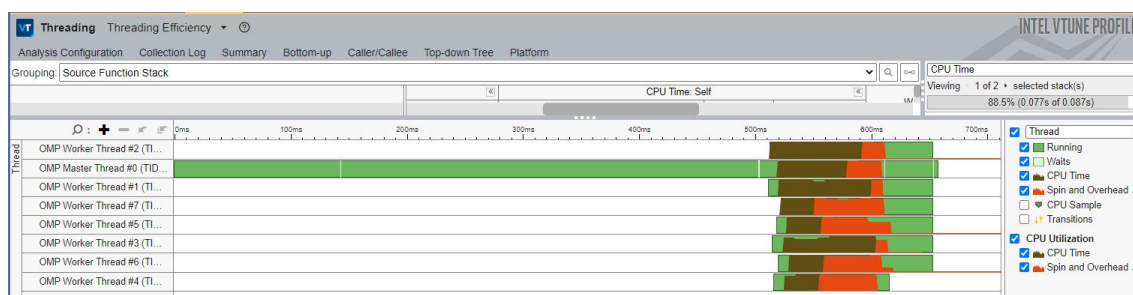
5, رسیدن به بیشترین همروندی با Tune کردن

برای این کار از بخش زیر استفاده کردیم



▼ Total	0	0ms	59.5%		40.5%
[No call stack information]	0	0ms			
▼ func@0x4b2e88f9	0	0ms	59.5%		40.5%
▼ func@0x4b2e8915	0	0ms	59.5%		40.5%
▼ BaseThreadInitThunk	0	0ms	59.5%		40.5%
▼ _scrt_common_main_seh	0	0ms	59.5%		35.5%
▼ main	0	0ms	59.5%		35.5%
▼ solve	0	0ms	59.5%		35.5%
▼ [OpenMP fork]	0	0ms	59.5%		35.5%
▼ _kmp_fork_call	0	0ms	59.5%		35.5%
▼ [OpenMP dispatcher]	0	0ms	59.5%		35.5%
▼ solve5omp\$parallel@42	0	0ms	59.5%		35.5%
▼ ?put_@YAHQAHHH@Z	0	0ms	59.5%		0.0%
▼ ?put_@YAHQAHHH@Z	0	0ms	59.5%		0.0%
▼ ?put_@YAHQAHHH@Z	0	0ms	33.2%		0.0%
▼ ?put_@YAHQAHHH@Z	0	0ms	21.5%		0.0%
▼ ?put_@YAHQAHHH@Z	0	0ms	8.8%		0.0%
▼ ?put_@YAHQAHHH@Z	0	0ms	4.4%		0.0%
▼ ?put_@YAHQAHHH@Z	0	0ms	1.4%		0.0%
▼ put	0	0ms	1.4%		0.0%
▼ put	0	0ms	2.9%		0.0%
▼ put	0	0ms	4.4%		0.0%
▼ put	0	0ms	12.6%		0.0%

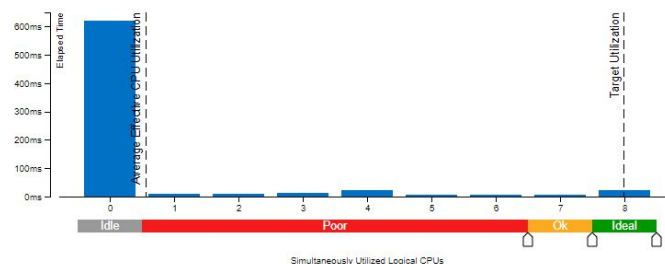
و در زیر می بینیم که هر ترد چگونه عمل کرده.



Effective CPU Utilization[®]: 7.1% (0.568 out of 8 logical CPUs) 📈

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead



Apply Threshold change may take time. Result will be re-opened.

OpenMP Analysis. Collection Time[®]: 0.711

Serial Time (outside parallel regions)[®]: 0.610s (85.8%) 📈

Top Serial Hotspots (outside parallel regions)

This section lists the loops and functions executed serially in the master thread outside of any OpenMP region and consuming hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the agg

No data to show. The collected data is not sufficient.

Parallel Region Time[®]: 0.101s (14.2%)

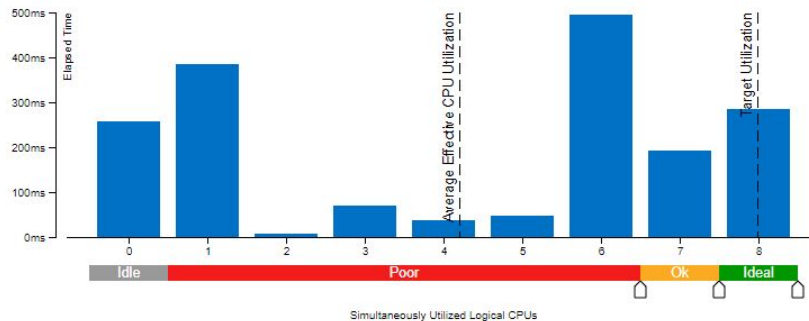
برای اینکه بتوانیم مقایسه و بررسی بهتری داشته باشیم و مقدمات اولیه نسبت به اصل محاسبات موازی بیشتر نباشد، N را برابر 14 می گذاریم تا نمودارهای دقیق تری بگیریم.

دوباره حالت اولیه را بررسی می کنیم

Effective CPU Utilization[®]: 52.7% (4.213 out of 8 logical CPUs) 📊

Effective CPU Utilization Histogram 📊

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle



OpenMP Analysis. Collection Time[®]: 1.778 📊

Serial Time (outside parallel regions)[®]: 0.611s (34.4%) 📊

Top Serial Hotspots (outside parallel regions) 📊

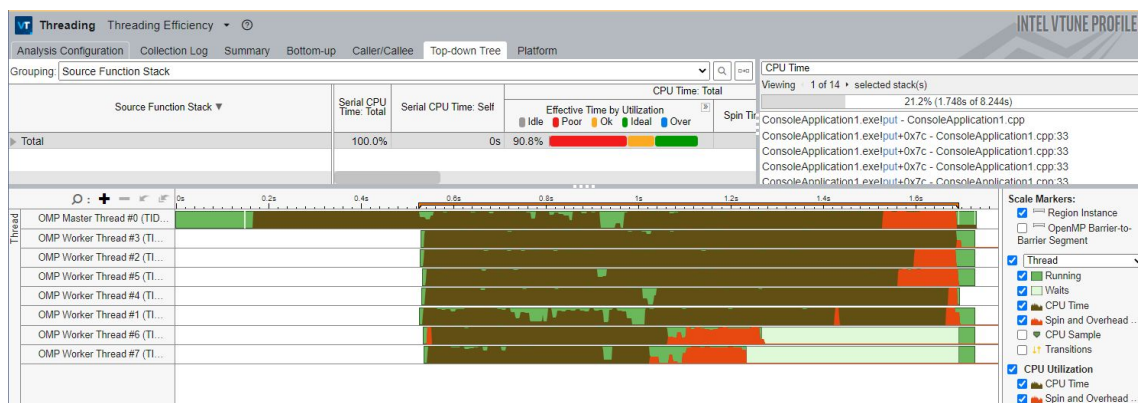
This section lists the loops and functions executed serially in the master thread outside of any OpenMP region and consuming the most CPU time. These hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the aggregated CPU

Function	Module	Serial CPU Time [®]
exit	ucrtbase.dll	0.009s

*N/A is applied to non-summable metrics.

Parallel Region Time[®]: 1.167s (65.6%) 📊

Estimated Ideal Time[®]: 0.953s (53.6%)

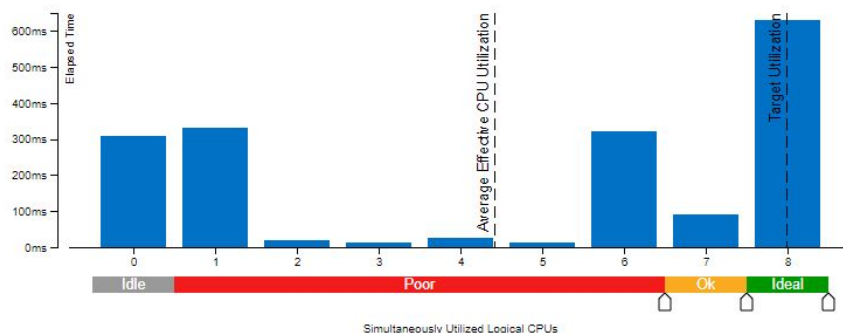


همانطور که دیده می شود دو تا از ترد ها خیلی زودتر کارشان تمام می شود و دیگر کاری نمی کنند، و در آخر بقیه هم این مورد بعضا دیده می شود ولی خیلی نیست. همچنین 6 ترد همزمان بیشترین زمان را به خود اختصاص داده اند. برای اینکه این اتفاق بهتر شود، از حالت schedule dynamic استفاده می کنیم و نتیجه را می بینیم.

Effective CPU Utilization: 55.4% (4.428 out of 8 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to 1



OpenMP Analysis. Collection Time: 1.755

Serial Time (outside parallel regions): 0.606s (34.5%)

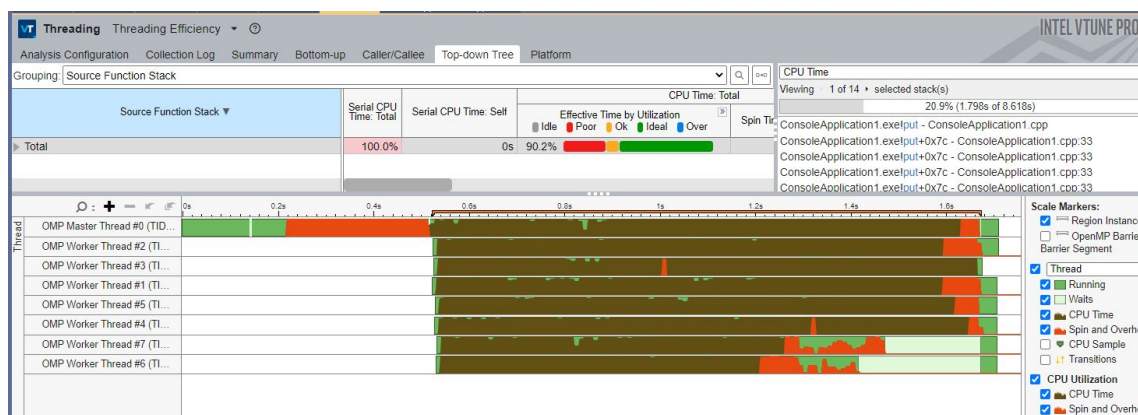
Top Serial Hotspots (outside parallel regions)

This section lists the loops and functions executed serially in the master thread outside of any OpenMP region and consuming the most CPU time. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the aggregate

Function	Module	Serial CPU Time
kmp_get_global_thread_id_reg	libiomp5md.dll	0.297s

*N/A is applied to non-summable metrics.

Parallel Region Time: 1.149s (65.5%)



همانطور که دیده می شود، حالا دیگر 8 ترد همزمان بیشترین سهم را به خود اختصاص داده است که نشان از بهتر شدن همروندی برنامه دارد. همچنین دو ترد آخر که از قبل از نصف زمان کارشان قبلا تمام شده بود، اینجا بیشتر کار کرده اند و کمتر منتظر بوده اند.

به این ترتیب همروندی را افزایش دادیم و اکثرا 8 ترد همزمان در حال کارند.

زمان اجرا در حالت tune شده:

```
Microsoft Visual Studio Debug Console
Mohsen Fayyaz - Mahdi Hadiloo
810196650 - 810196587
# solutions 14200 time: 33.000000

F:\Programming\University\07_Parallel_Programming\Parallel_Programming\ConsoleApplication1.exe (process 16036) exited with code 0.
To automatically close the console when debugging stops, enable the option 'Automatically close console when debugging stops'.
Press any key to close this window . . .
```

سرعت این حالت شاید آنچنان خود را در N های نسبتاً کمتر نشان ندهد و نسبت به حالت موازی اولیه نزدیک باشد چون آن کد هم از درجه همروندی خوبی برخوردار بود.

حالت	زمان اجرا	speedup
سریال	104	1
موازی اولیه	35	2.97
حالت Tune شده	33	3.15