



CA6 - Parallel Studio

Parallel Programming

امید بداقی 810196423

میینا شاه بنده 810196488

زمستان 99

دکتر صفری

مقدمه

در این پروژه کدی برای حل مسئله N-Queens ارائه شده است که در دو حالت موازی و سریال پیاده سازی شده است. با استفاده از ابزار Intel Parallel Studio تحلیل کد موازی صورت گرفته تا اشکالات آن نظیر deadlock ها و همچنین hotspot های آن پیدا شود تا بتوان کد موازی را بهینه تر و موازی سازی را بیشتر کرد.

محاسبه تعداد روش‌های قرار دادن N وزیر در صفحه

ابتدا آرایه‌ی Queens را در نظر می‌گیریم. در واقع در ردیف i ، وزیر را در ستون $Queens[i]$ قرار می‌دهیم. حال به روش backtracking عمل می‌کنیم. الگوریتم به این صورت است که:

هر بار یک وزیر را در ستون i و ردیف j قرار می‌دهیم. حال تلاش می‌کنیم وزیر بعدی را در ستون بعدی به صورتی قرار دهیم تا با وزیرهای قبلی تداخلی نداشته باشد. در هر مرحله، در صورتی که وزیر را در هیچ ستونی نتوانیم قرار دهیم، backtrack می‌کنیم و وزیر پدر را در ردیف مجاز دیگری قرار می‌دهیم. هر بار که بتوانیم تمام وزیرها را در صفحه قرار دهیم، یک راه‌حل برای چیدن به دست آورده‌ایم و solutions را یکی زیاد می‌کنیم.

در تابع put، ابتدا بررسی می‌شود هیچ وزیر دیگری در ستون یا خانه‌های موربی نباشد که وزیر کنونی را بزند. اگر وزیر آخر با موفقیت قرار داده شد، تعداد راه‌حل‌ها را یکی زیاد می‌کنیم و در غیر اینصورت، تلاش می‌کنیم وزیرهای بعدی را در ردیف‌های بعدی اضافه کنیم. در صورتی که وزیر نتواند اضافه شود، تابع $1 -$ باز می‌گرداند و مطابق توضیحات، به وزیر parent باز می‌گردیم.

پیاده سازی سری

در تابع solve()، تمام ستون‌های ردیف اول برای قرار دادن وزیر اول را بررسی می‌کنیم تا تعداد راه‌حل‌های مجاز را بدست آوریم.

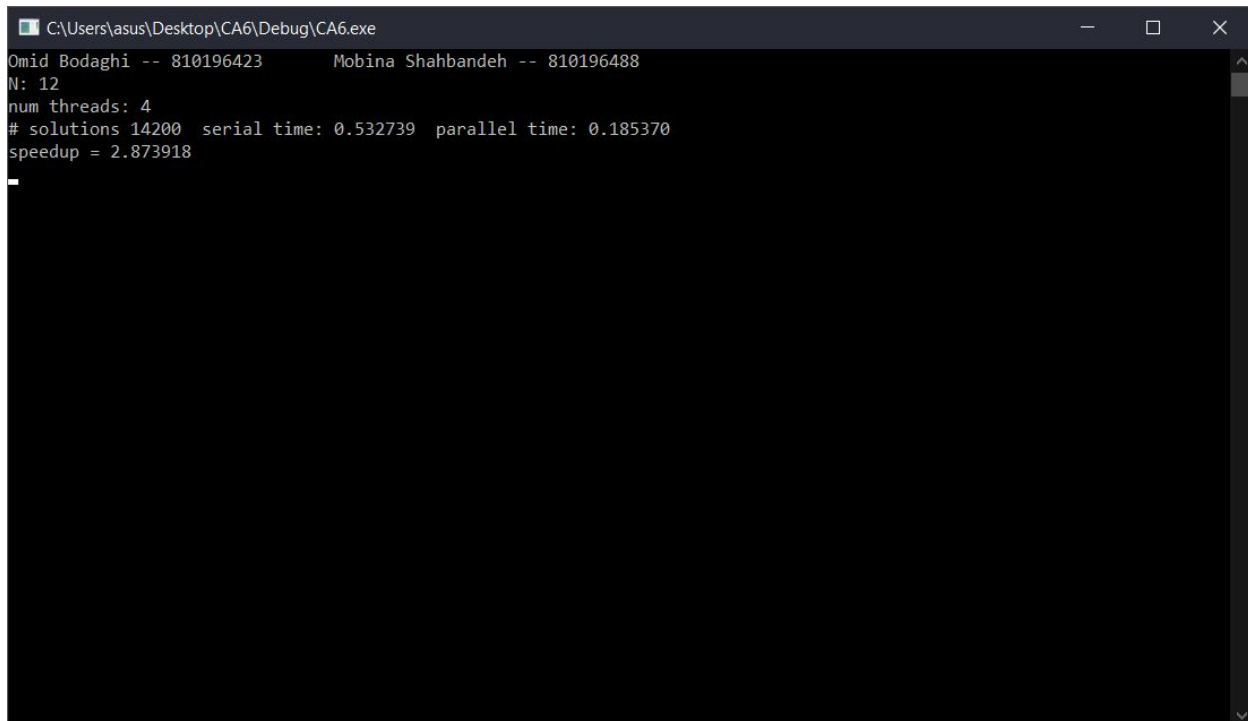
پیاده سازی موازی

در پیاده سازی موازی همان پیاده سازی سری را در یک ساختار parallel قرار داده و حلقه را نیز در یک ساختار for قرار می‌دهیم. همچنین آرایه Queens نیز باید در بدنه parallel، initialize شود، زیرا به صورت

`private` برای هر ترد تعریف می شود. از ساختار `atomic` برای قسمتی که تعداد راه حل ها را یکی زیاد می کنیم استفاده شده زیرا این متغیر اشتراکی است و توسط تمام تردها ممکن است تغییر کند.

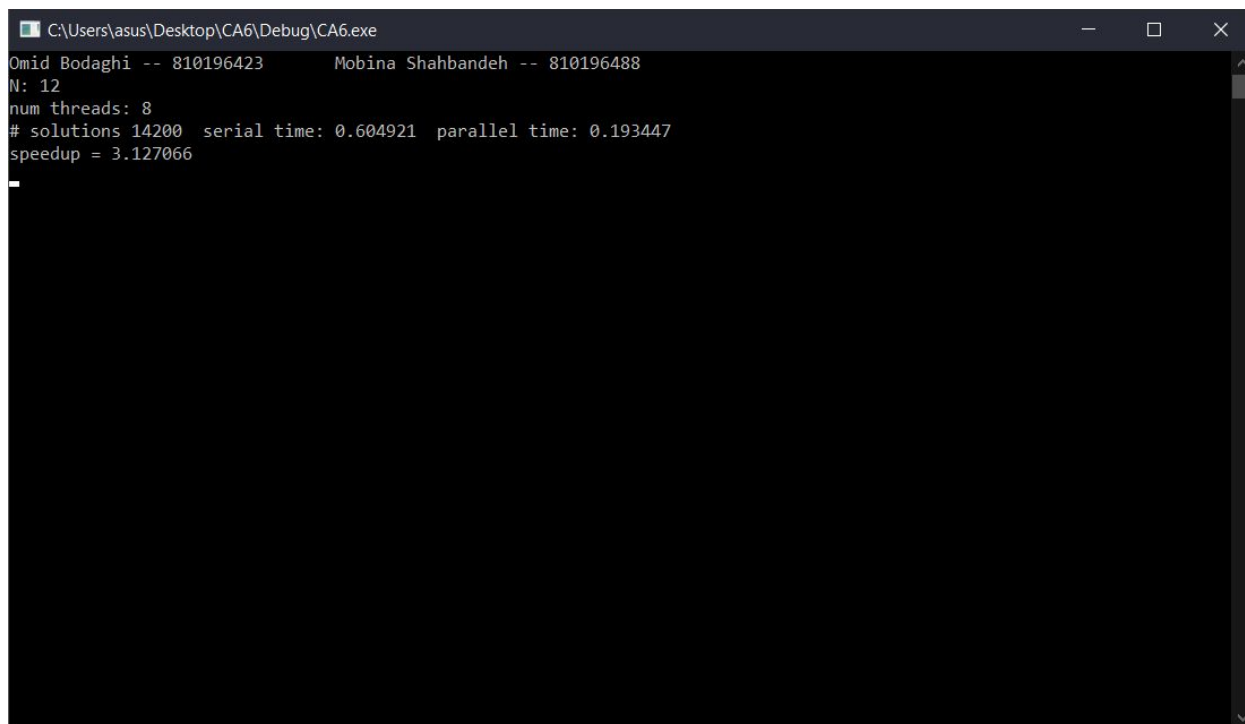
نتایج اجرا

`schedule(static), 4 threads`



```
C:\Users\asus\Desktop\CA6\Debug\CA6.exe
Omid Bodaghi -- 810196423      Mobina Shahbandeh -- 810196488
N: 12
num threads: 4
# solutions 14200  serial time: 0.532739  parallel time: 0.185370
speedup = 2.873918
```

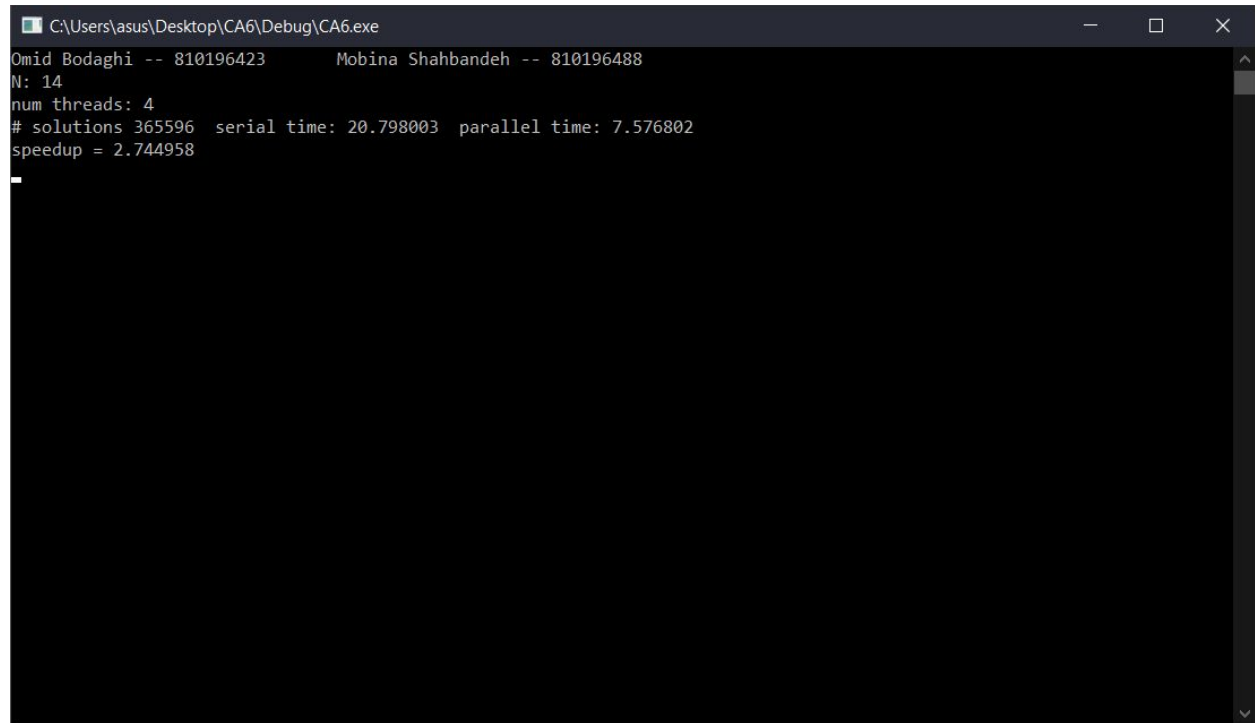
schedule(static), 8 threads



```
C:\Users\asus\Desktop\CA6\Debug\CA6.exe
Omid Bodaghi -- 810196423 Mobina Shahbandeh -- 810196488
N: 12
num threads: 8
# solutions 14200 serial time: 0.604921 parallel time: 0.193447
speedup = 3.127066
```

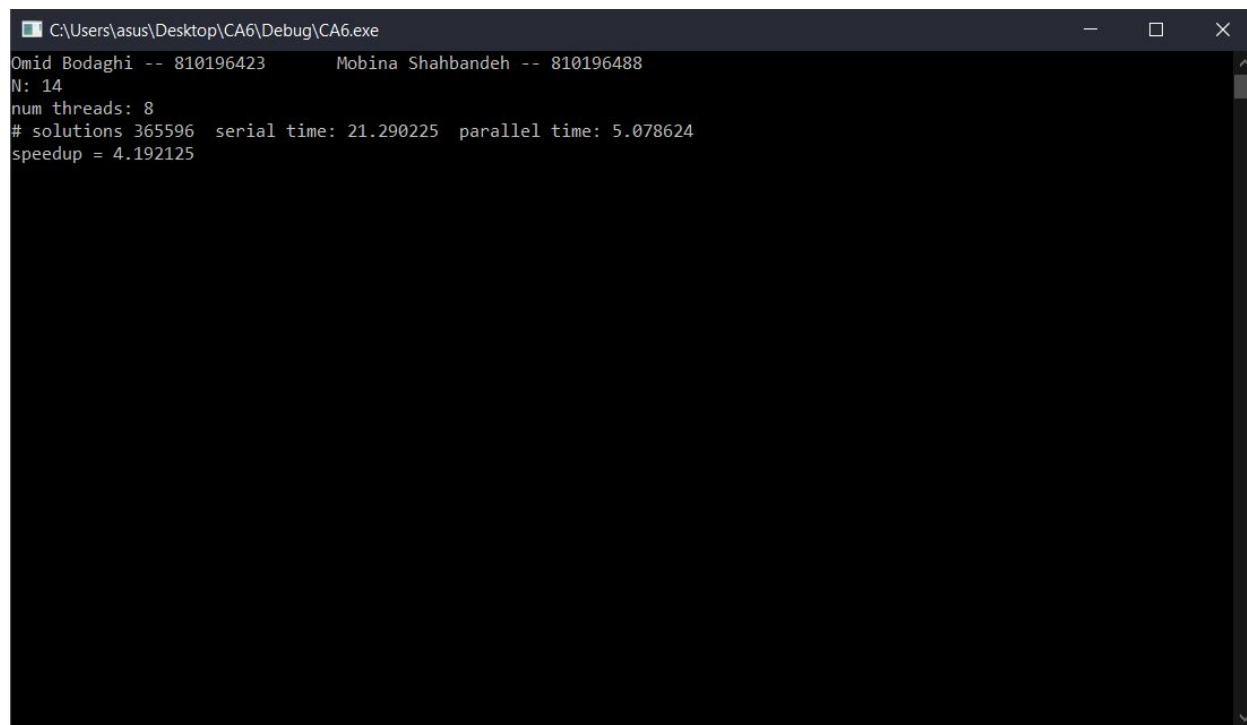
در ابتدا $N = 12$ بود اما نتیجه موازی سازی خوبی به دست نمی آمد. پس از این به بعد $N = 14$ قرار داده شد.

schedule(static), 4 threads



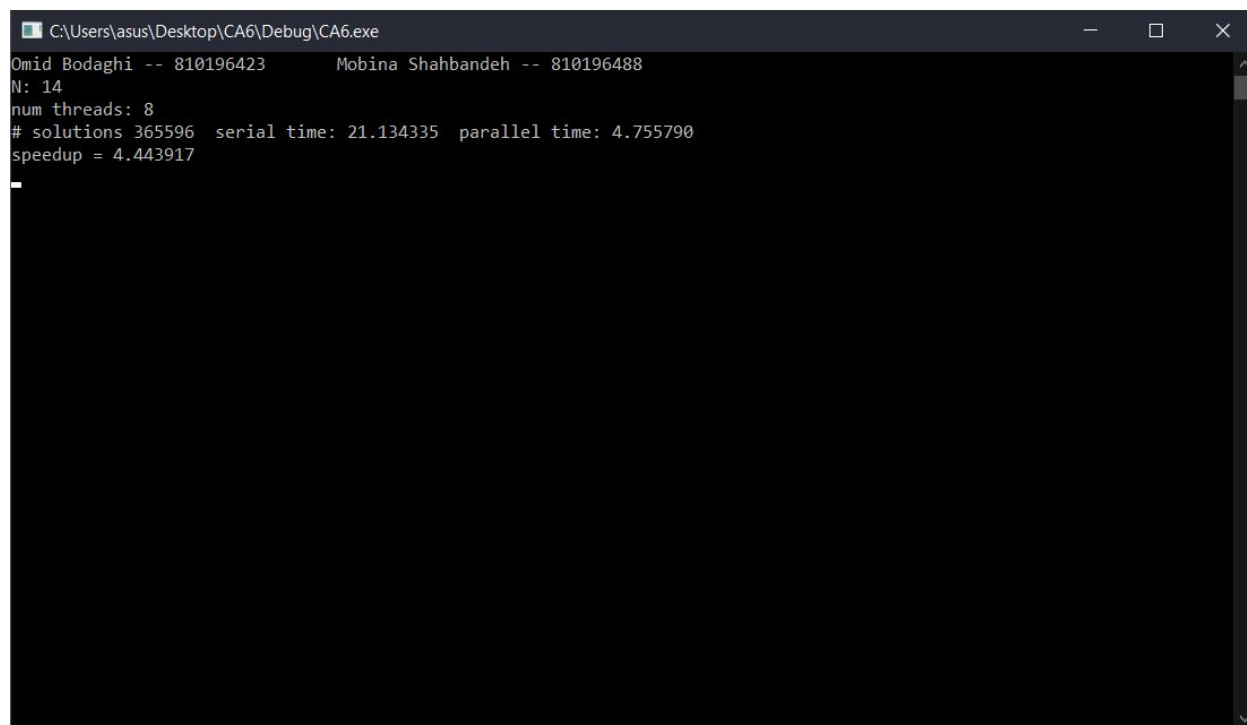
```
C:\Users\asus\Desktop\CA6\Debug\CA6.exe
Omid Bodaghi -- 810196423 Mobina Shahbandeh -- 810196488
N: 14
num threads: 4
# solutions 365596 serial time: 20.798003 parallel time: 7.576802
speedup = 2.744958
-
```

schedule(static), 8 threads



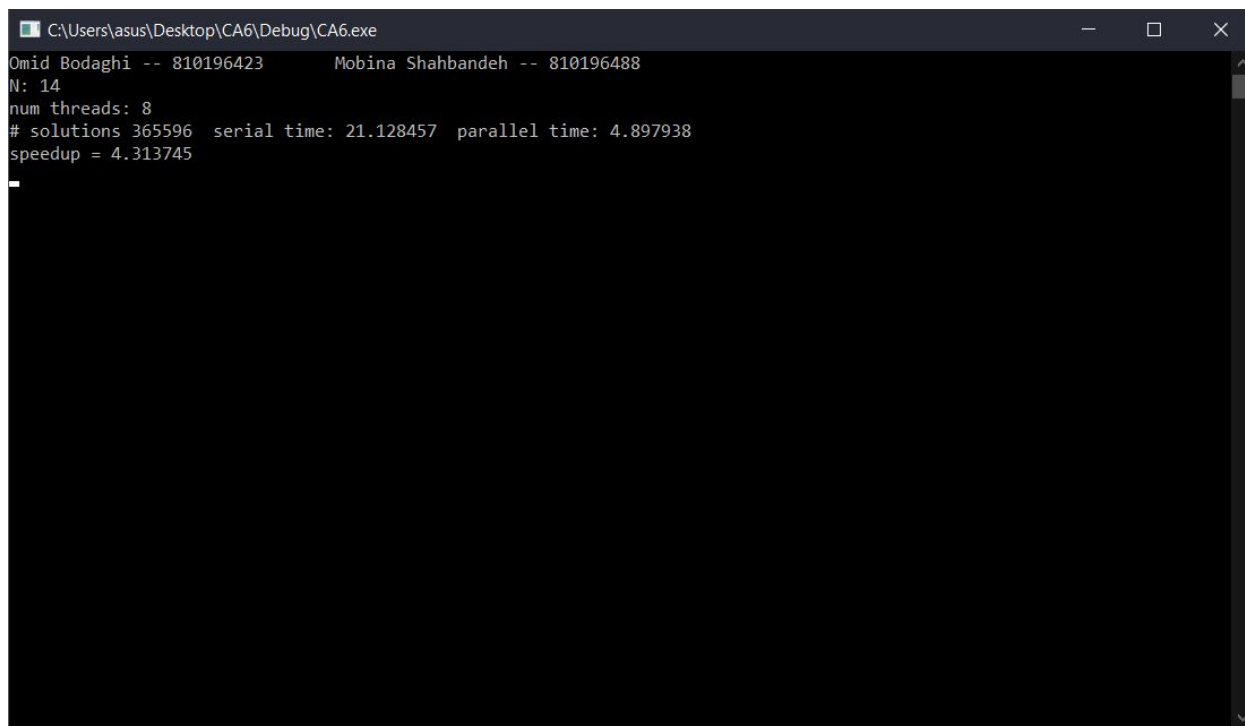
```
C:\Users\asus\Desktop\CA6\Debug\CA6.exe
Omid Bodaghi -- 810196423      Mobina Shahbandeh -- 810196488
N: 14
num threads: 8
# solutions 365596  serial time: 21.290225  parallel time: 5.078624
speedup = 4.192125
```

schedule(dynamic, 1), 8 threads



```
C:\Users\asus\Desktop\CA6\Debug\CA6.exe
Omid Bodaghi -- 810196423      Mobina Shahbandeh -- 810196488
N: 14
num threads: 8
# solutions 365596  serial time: 21.134335  parallel time: 4.755790
speedup = 4.443917
```

schedule(dynamic, 2), 8 threads



```

C:\Users\asus\Desktop\CA6\Debug\CA6.exe
Omid Bodaghi -- 810196423 Mobina Shahbandeh -- 810196488
N: 14
num threads: 8
# solutions 365596 serial time: 21.128457 parallel time: 4.897938
speedup = 4.313745

```

همانطور که مشخص است، بهترین حالت داینامیک با $\text{chunk size} = 1$ و 8 ترد است که نتایج Parallel Studio آن در ادامه آمده است.

نتایج تحلیل کد با Parallel Studio

پلتفرمی که روی آن کدها اجرا شده نیز به صورت زیر است:

⌵ CPU

Name: Intel(R) Processor code named Skylake

Frequency: 2.6 GHz

Logical CPU Count: 8

⌵ Cache Allocation Technology

Level 2 capability: not detected

Level 3 capability: not detected

در ابتدا کد سریال $N = 12$ را اجرا می کنیم تا hotspot های آن بدست آیند:

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time [®]
put	CA6.exe	0.997s
abs	ucrtbased.dll	0.061s
_kmpc_global_thread_num	libiomp5md.dll	0.015s

[®]N/A is applied to non-summable metrics.

همانطور که مشاهده می شود، hotspot های برنامه تابع put و abs هستند. تابع put به صورت بازگشتی بارها فراخوانی می شود که به این علت زمان بیشتری از زمان اجرا را به خود اختصاص داده است. تابع abs به صورت دستی نیز نوشته شد اما زمان اجرا بهتر نشد. پس در حلقه بیرونی موازی سازی را انجام می دهیم (حلقه های داخلی نیز موازی سازی شد اما نتیجه بدتری حاصل شد بنابراین از موازی سازی آنها صرف نظر شد).

حال موازی سازی را با 8 ترد و $N = 14$ به پیش می بریم.

schedule(dynamic, 1)

hotspots

Elapsed Time[®]: 5.889s

CPU Time [®] :	33.185s
Wait Time [®] :	3.309s
Total Thread Count:	8
Paused Time [®] :	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time [®]
put	CA6.exe	23.490s
abs	ucrtbased.dll	7.261s
_kmpc_barrier	libiomp5md.dll	0.974s
func@0x4010d7	CA6.exe	0.595s
omp_get_wtime	libiomp5md.dll	0.518s
[Others]	N/A [*]	0.345s

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

Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the [Bottom-up](#) view for in-depth analysis per function. Otherwise, use the [Caller/Callee](#) view to track critical paths for these hotspots.

Explore Additional Insights

Parallelism[®]: 67.1%
Use [Threading](#) to explore more opportunities to increase parallelism in your application.

threading

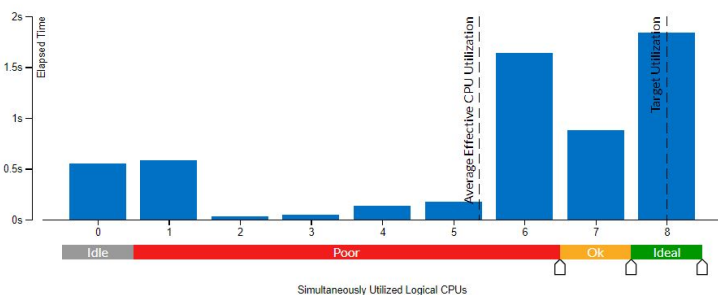
Elapsed Time[®]: 5.889s

Paused Time[®]: 0s

Effective CPU Utilization[®]: 67.1% (5.366 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 CPU utilization value.



OpenMP Analysis. Collection Time[®]: 5.889

Serial Time (outside parallel regions)[®]: 1.080s (18.3%) ▴

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. Improve overall application performance by optimizing or parallelizing these hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the aggregated CPU time in the table.

Function	Module	Serial CPU Time [®]
omp_get_wtime	libiomp5md.dll	0.518s
exit	ucrtbased.dll	0.025s

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

Parallel Region Time[®]: 4.809s (81.7%)

Total Thread Count: 8

Thread Oversubscription[®]: 0s (0.0% of CPU Time)

Wait Time with poor CPU Utilization[®]: 3.309s (100.0% of Wait Time)

Top Waiting Objects

This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep() or I/O, or on contended synchronizations. A significant amount of Wait time associated with a synchronization object reflects high contention for that object and, thus, reduced parallelism.

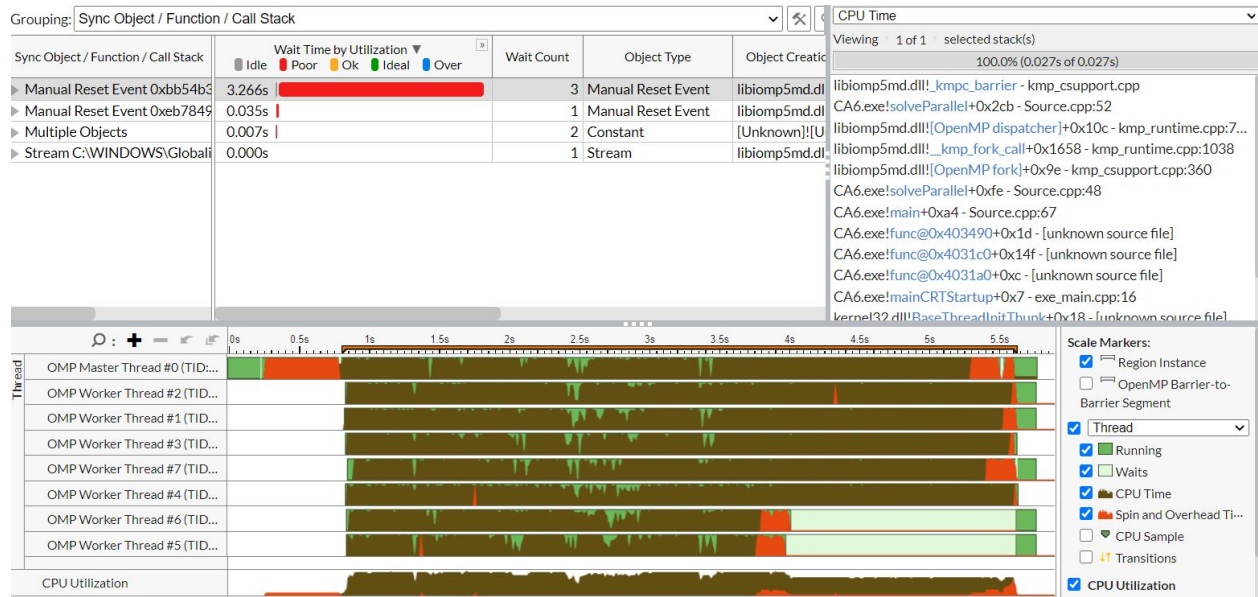
Sync Object	Wait Time with poor CPU Utilization [®]	(% from Object Wait Time) [®]	Wait Count [®]
Manual Reset Event 0xbb54b302	3.266s	100.0%	3
Manual Reset Event 0xeb784990	0.035s	100.0%	1
Multiple Objects	0.007s	100.0%	2
Stream C:\WINDOWS\Globalization\Sorting\sortdefault.nls 0x406c0781	0.000s	100.0%	1

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

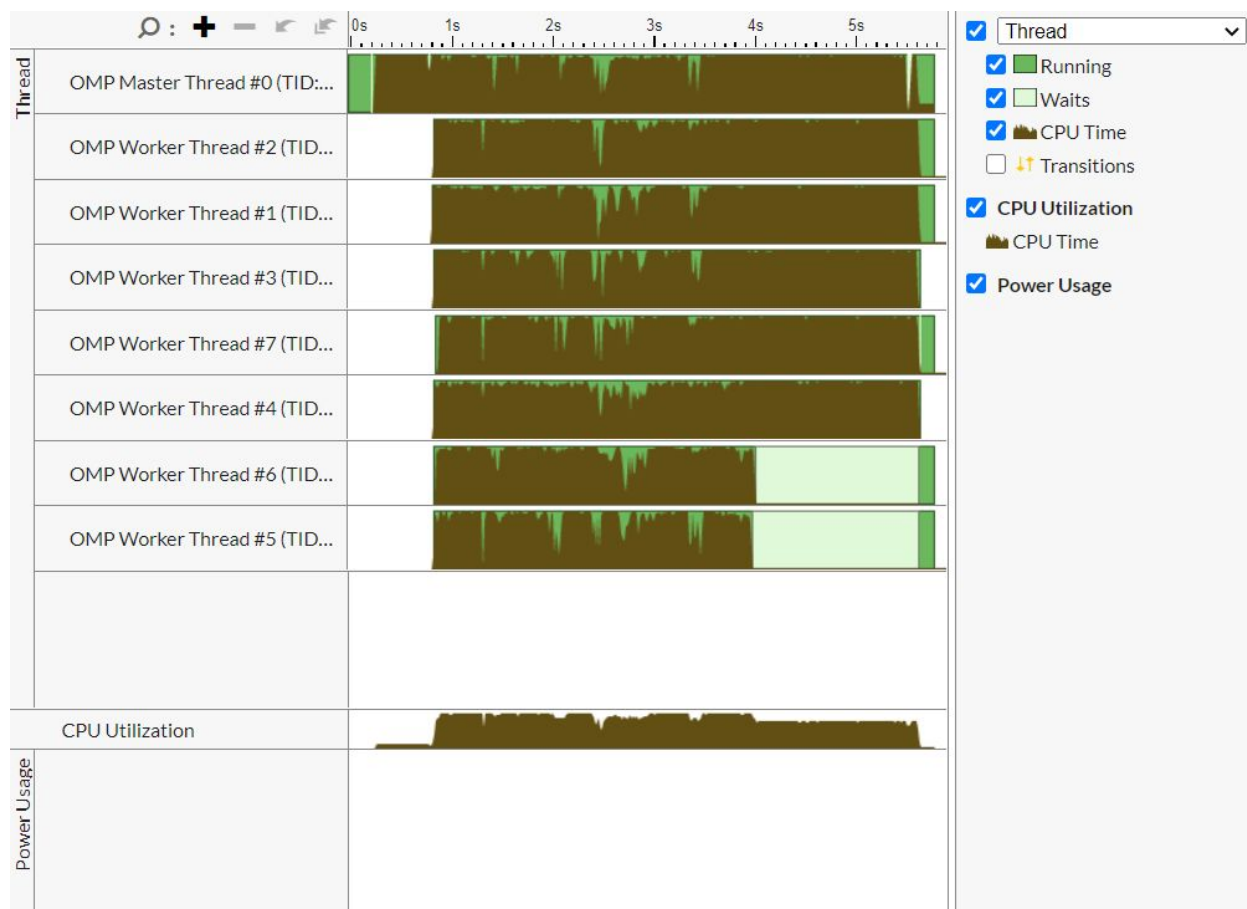
Spin and Overhead Time[®]: 1.586s (4.8% of CPU Time)

همانطور که مشاهده می شود، utilization برابر با 67.1 درصد شده است که از حالات دیگر با $N = 14$ بهتر است.

bottom-up

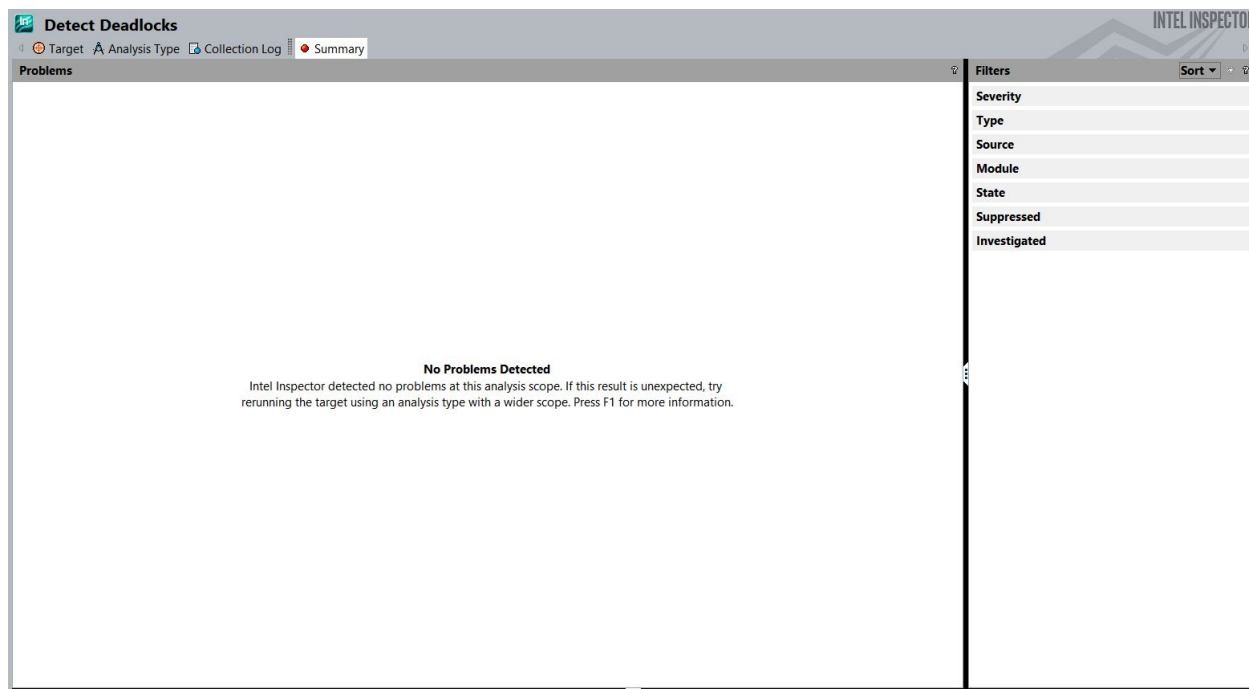


platform



همانطور که مشاهده می شود، تردها در اکثر زمان مشغول به کار هستند و idle نیستند. دو ترد 5 و 6 بخشی از زمان idle هستند و علت آنست که $N = 14$ است و ما 8 ترد داریم. پس دو ترد یک chunk ممکن است بگیرند که اینجا این دو ترد شده اند. اگر $N = 16$ بگذاریم این مشکل به وجود نمی آید و utilization بالاتر می رود اما زمان بسیار زیادی طول می کشد تا اجرا شود.

deadlocks



همانطور که مشاهده می شود مشکلی از نظر **race condition** و **deadlock** و ... وجود ندارد.

schedule(dynamic,2)

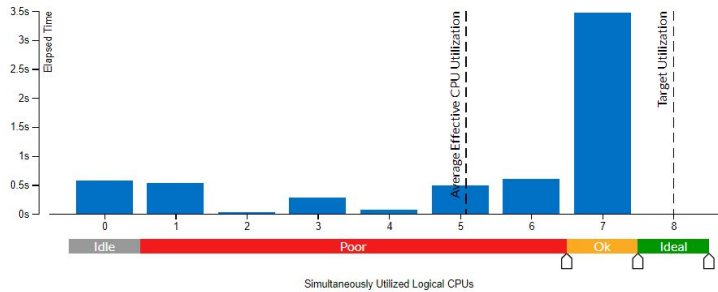
Elapsed Time[®]: 6.046s

Paused Time[®]: 0s

Effective CPU Utilization[®]: 63.5% (5.084 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 CPU utilization value.



OpenMP Analysis. Collection Time[®]: 6.046

Serial Time (outside parallel regions)[®]: 1.047s (17.3%) 📈

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. Improve overall application performance by optimizing or parallelizing these hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the aggregated CPU time in the table.

Function	Module	Serial CPU Time [®]
omp_get_wtime	libiomp5md.dll	0.473s

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

Parallel Region Time[®]: 4.999s (82.7%)

Estimated Ideal Time[®]: 4.060s (67.2%)

OpenMP Potential Gain[®]: 0.938s (15.5%) 📈

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain [®]	(%) [®]	OpenMP Region Time [®]
solveParallel\$omp\$parallel:8@unknown:48:56	0.938s 📈	15.5% 📈	4.999s

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

Total Thread Count: 8

Thread Oversubscription[®]: 0s (0.0% of CPU Time)

Wait Time with poor CPU Utilization[®]: 2.884s (46.7% of Wait Time)

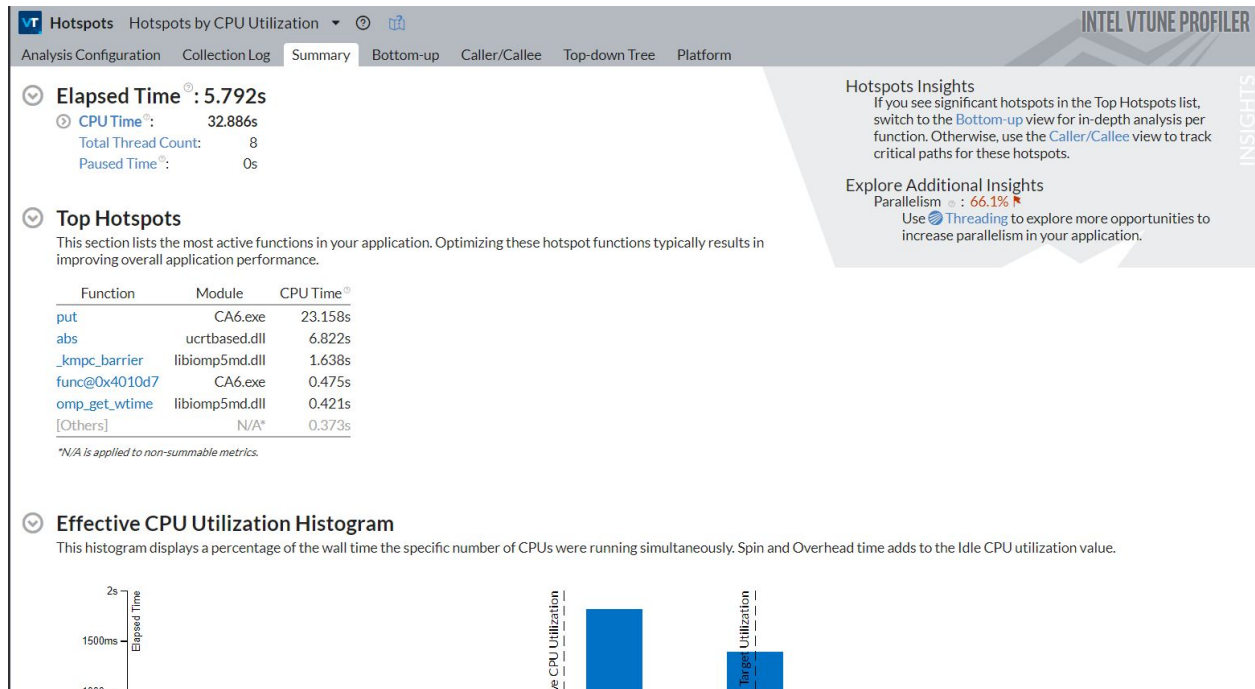
Top Waiting Objects

This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep() or I/O, or on contended synchronizations. A significant amount of Wait time associated with a synchronization object reflects high contention for that object and, thus, reduced parallelism.

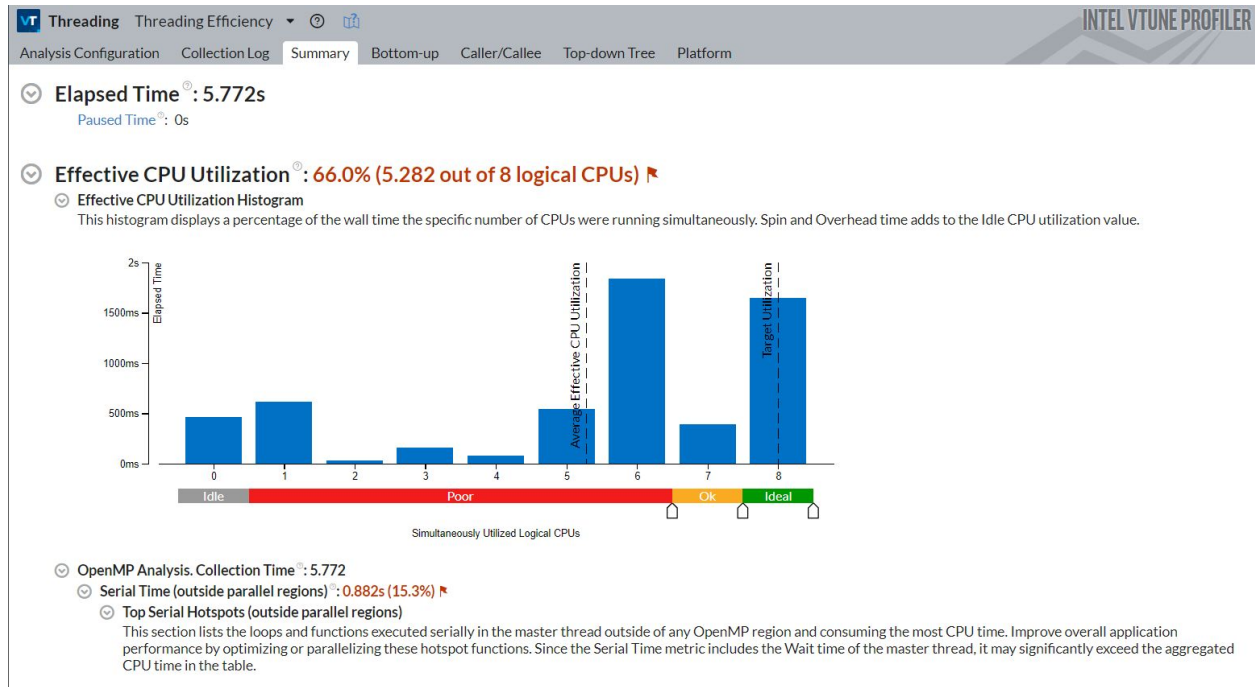
همانطور که مشاهده می شود utilization در این حالت از حالت قبلی کمتر شده است.

schedule(static)

hotspots

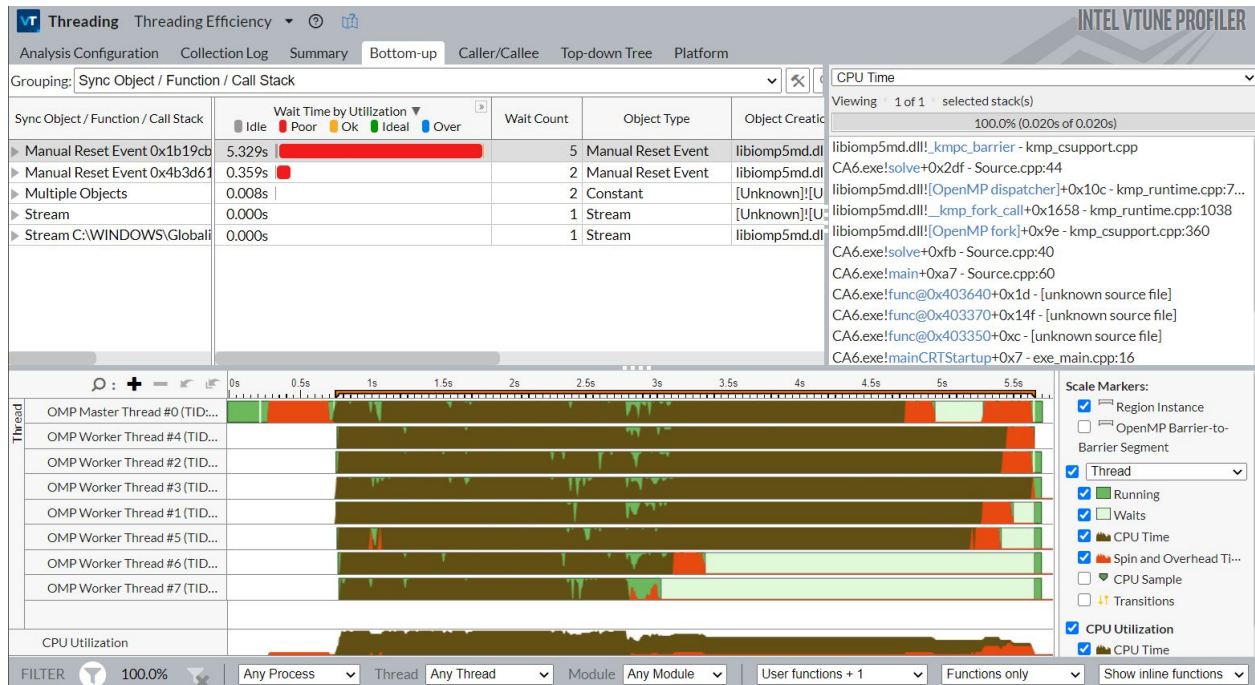


threading

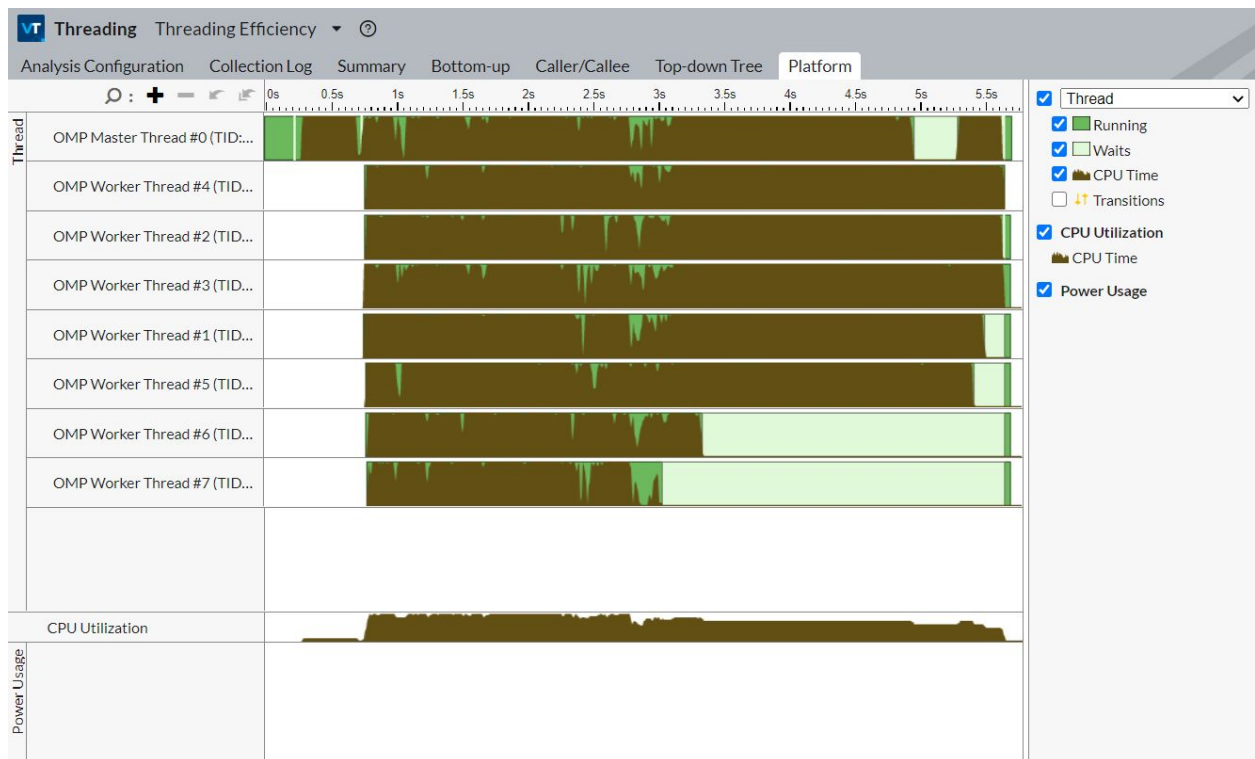


همانطور که مشاهده می شود **utilization** در این حالت اندکی از حالت **dynamic,1** کمتر است. به همین سبب حالت **dynamic,1** به عنوان بهترین حالت انتخاب شده است.

bottom-up



platform



این قسمت تقریباً مشابه **dynamic,1** است با این تفاوت که تردها زمان بیشتری در حالت **idle** هستند؛ زیرا **scheduling** به صورت **static** است و از ابتدا **chunk** های مساوی به تردها داده شده و چون $N = 14$ است و تردها 8 عدد هستند به دو ترد از اول یک **chunk** داده می شود و تفاوت با حالت **dynamic** این است که این دو ترد ممکن است محاسبات مربوط به **chunk** اول خود را سریع تر از سایر تردها که دو **chunk** دارند به اتمام برسانند.

در ادامه نتایج مربوط به 7 ترد نیز آمده است:

schedule(dynamic, 1)

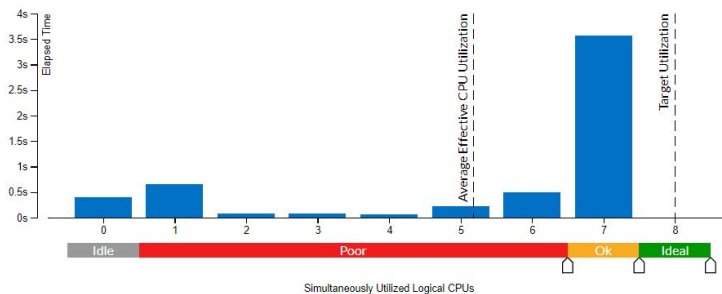
Elapsed Time[®]: 5.575s

Paused Time[®]: 0s

Effective CPU Utilization[®]: 64.8% (5.186 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 CPU utilization value.



OpenMP Analysis, Collection Time[®]: 5.575

Serial Time (outside parallel regions)[®]: 1.018s (18.3%) 📈

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. Improve overall application performance by optimizing or parallelizing these hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the aggregated CPU time in the table.

schedule(static)

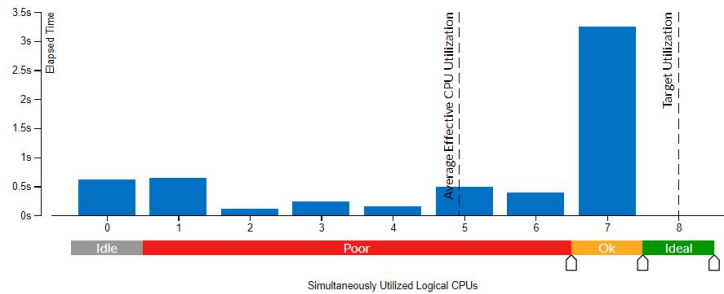
Elapsed Time[®]: 5.900s

Paused Time[®]: 0s

Effective CPU Utilization[®]: 61.7% (4.934 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 CPU utilization value.



OpenMP Analysis. Collection Time[®]: 5.900

Serial Time (outside parallel regions)[®]: 1.188s (20.1%) 📈

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. Improve overall application performance by optimizing or parallelizing these hotspot functions. Since the Serial Time metric includes the Wait time of the master thread, it may significantly exceed the aggregated CPU time in the table.

همانطور که مشاهده می شود در هر دوی این حالات utilization از حالت 8 ترد کمتر است.