

به نام خدا

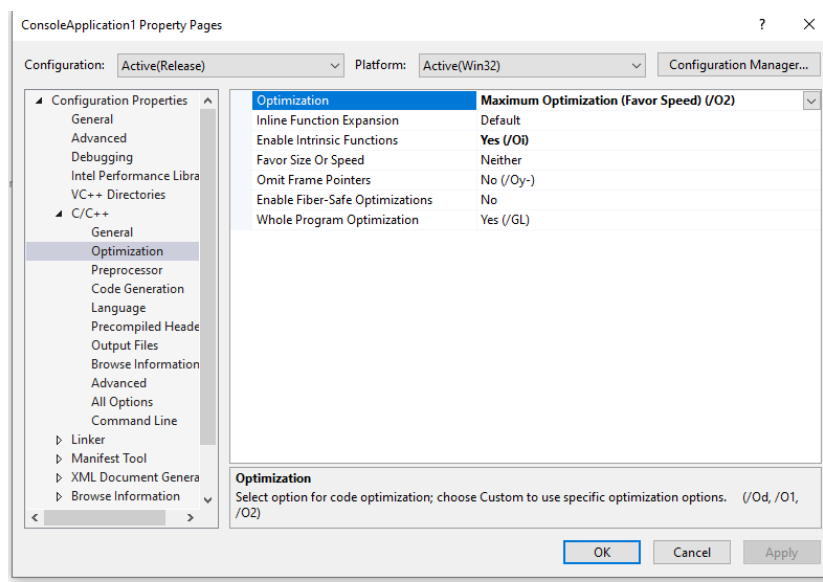
در اولین قدم از کار، این برنامه را به صورت سریال و بدون هیچ تنظیم خاصی از قبلی کامپایلر اینتل و ... اجرا می‌کنیم که مطابق تصویر زیر، زمان اجرا برای $N = 12$ در حالت سریال برابر با ۲۱۴ میلی‌ثانیه است:

```

Microsoft Visual Studio Debug Console
Hossein Soltanloo - 810195407
# solutions 14200 time:          214 ms

C:\Users\hossein\source\repos\ConsoleApplication1\Release\ConsoleAppli
cation1.exe (process 10860) exited with code 0.
Press any key to close this window . . .
  
```

در قدم بعد طبق تصویر زیر، Optimization را روی O2 تنظیم می‌کنیم:



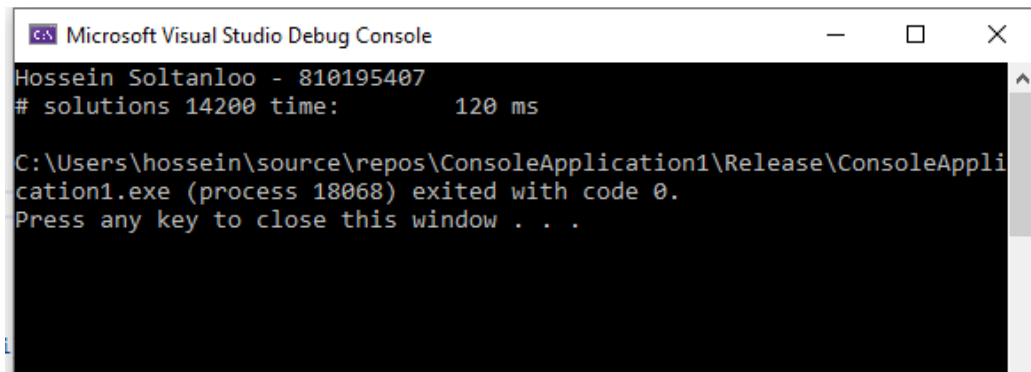
سپس مشاهده می‌شود که با همین کار، سرعت اجرای برنامه بهبود بسیاری یافته و زمان اجرای آن به ۱۲۳ میلی‌ثانیه بهبود پیدا می‌کند:

```

Microsoft Visual Studio Debug Console
Hossein Soltanloo - 810195407
# solutions 14200 time:          123 ms

C:\Users\hossein\source\repos\ConsoleApplication1\Release\ConsoleAppli
cation1.exe (process 10156) exited with code 0.
Press any key to close this window . . .
  
```

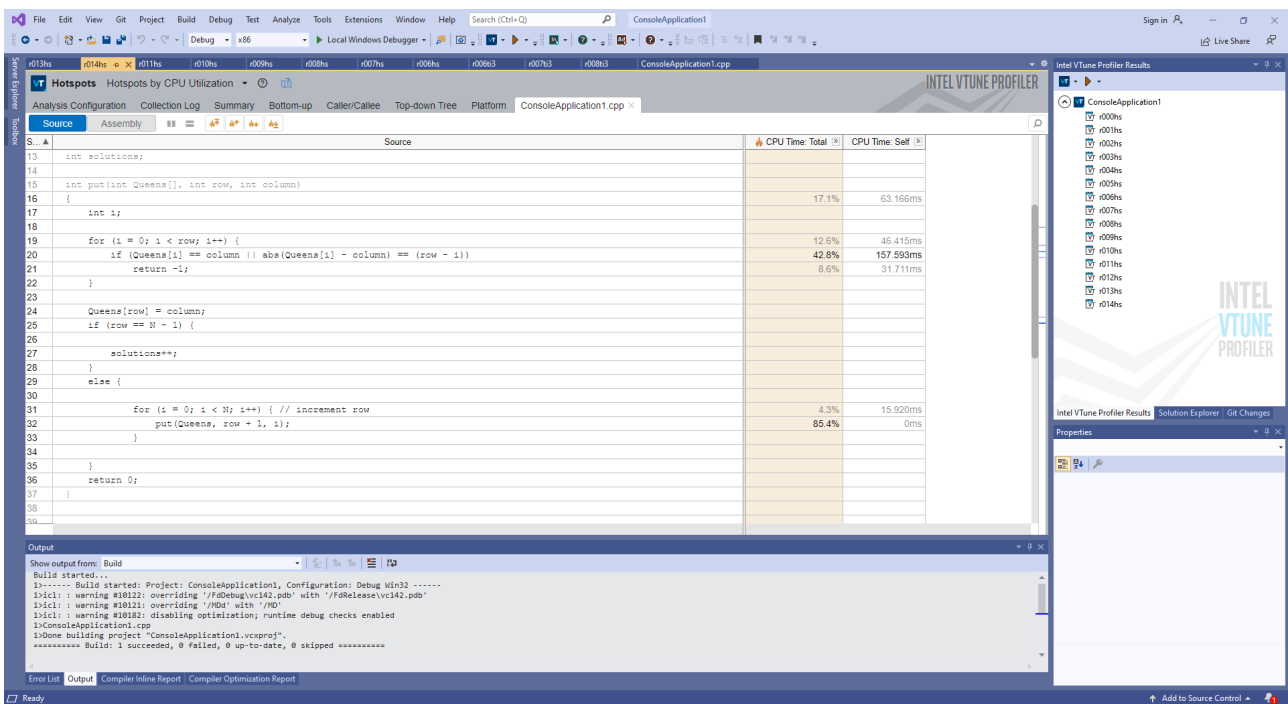
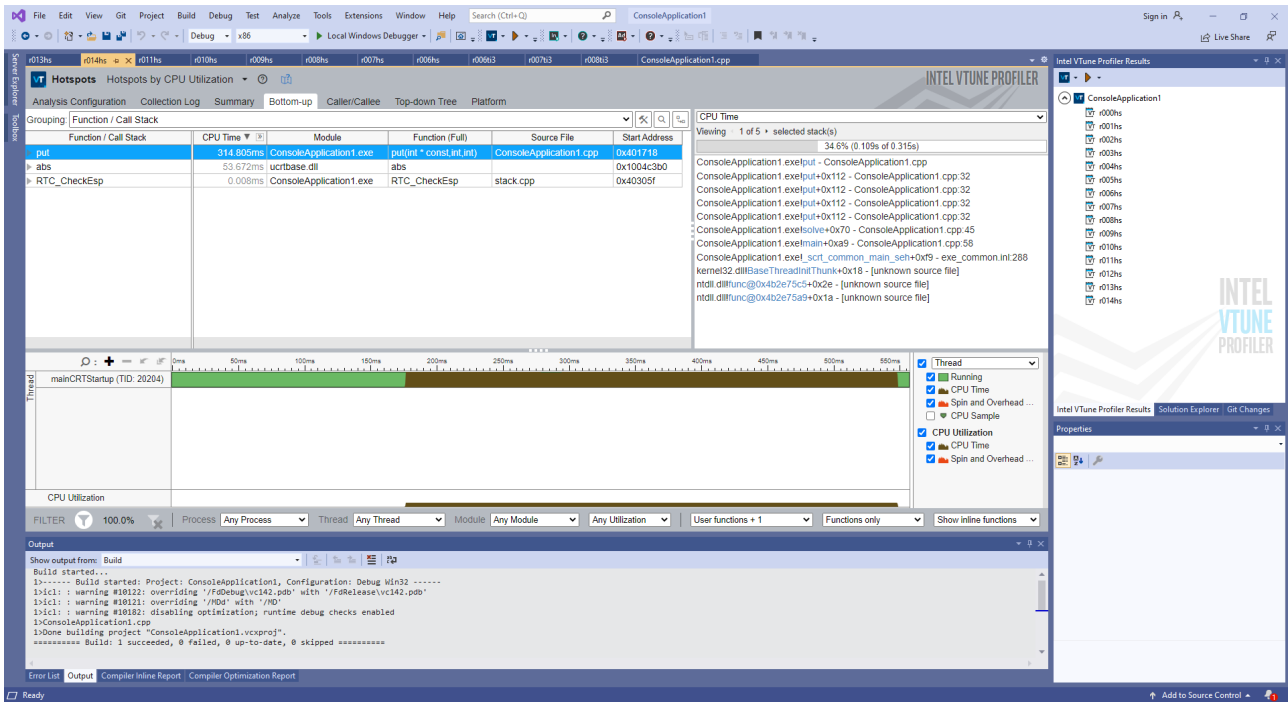
در بخش بعد، کامپایلر را بر روی Intel C++ Compiler قرار می‌دهیم و مجدداً برنامه را اجرا می‌کنیم که طبق تصویر زیر، زمان اجرا ۱۲۰ میلی‌ثانیه است:



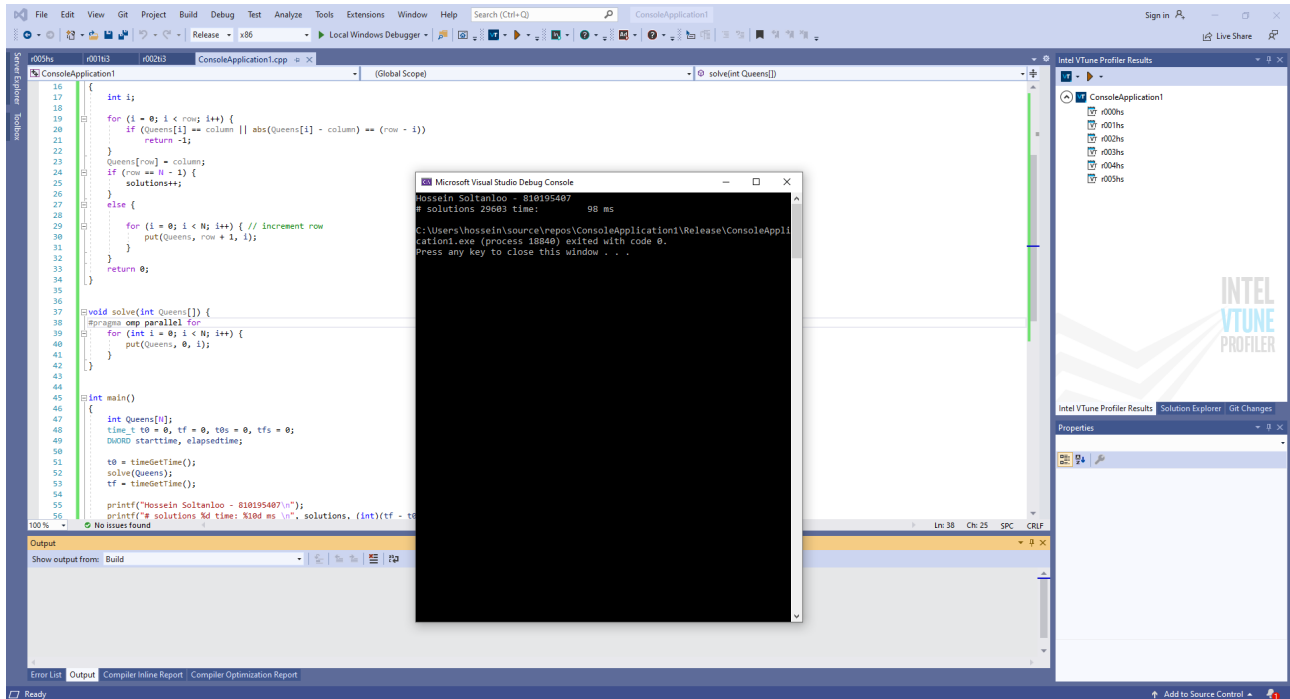
```
Microsoft Visual Studio Debug Console
Hossein Soltanloo - 810195407
# solutions 14200 time: 120 ms

C:\Users\hossein\source\repos\ConsoleApplication1\Release\ConsoleApplication1.exe (process 18068) exited with code 0.
Press any key to close this window . . .
```

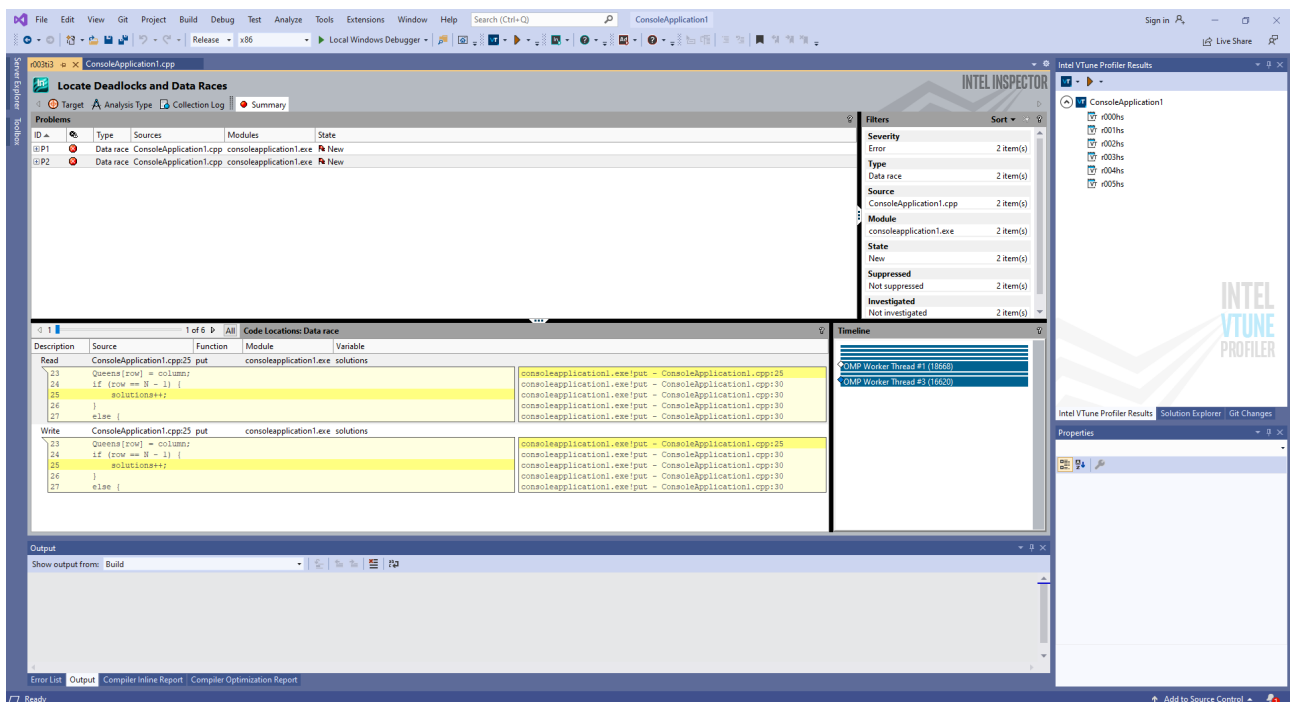
سپس با استفاده از VTune، اقدام به یافتن Hotspot های برنامه می کنیم. طبق تصویر مشاهده می شود که بخش اعظم زمان اجرا صرف اجرای تابع put و همچنین تابع abs درون آن می شود:

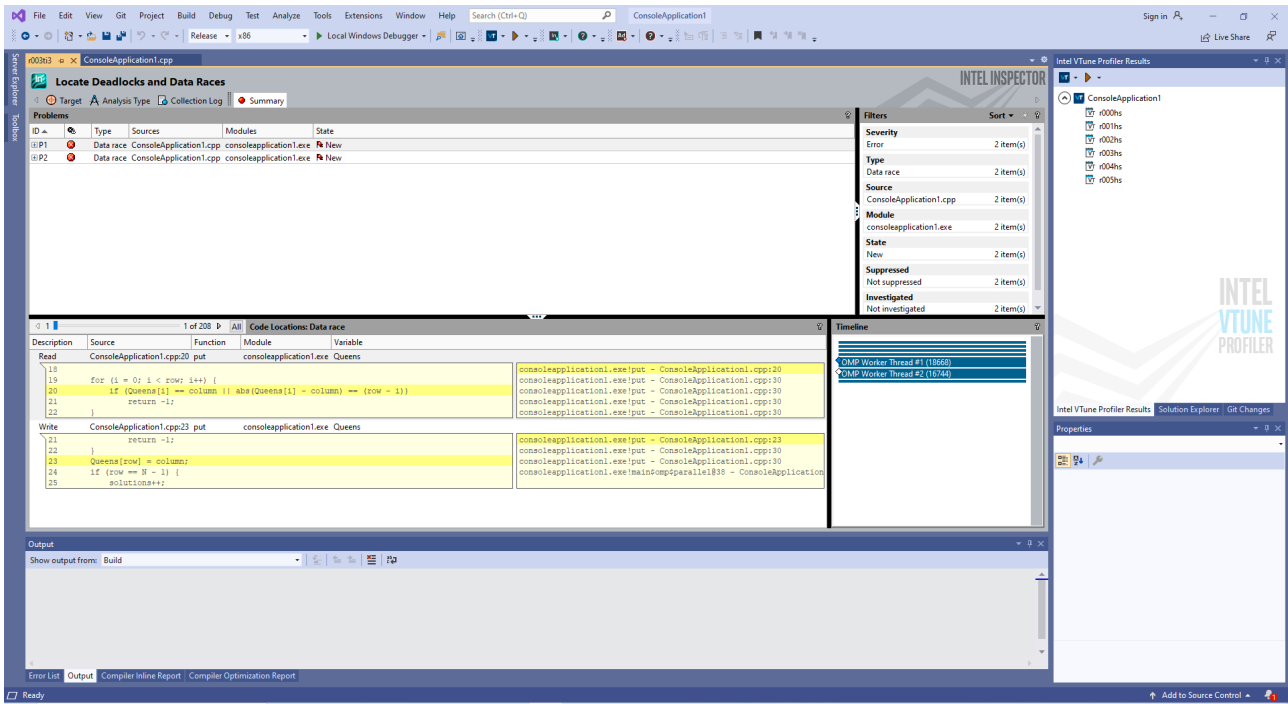


قدم بعدی این است که این Hotspot ها را از طریق موازی‌سازی برنامه، سریع‌تر کنیم. از آن‌جا که در ابتدا در تابع solve، تابع put به تعداد N بار فراخوانی می‌شود، به نظر می‌رسد که می‌توانیم این قسمت را با استفاده از OpenMP و ساختار for در آن، موازی‌سازی کنیم. پس از انجام این کار می‌بینیم که زمان اجرای برنامه کاهش یافته اما پاسخ صحیح دریافت نمی‌کنیم:

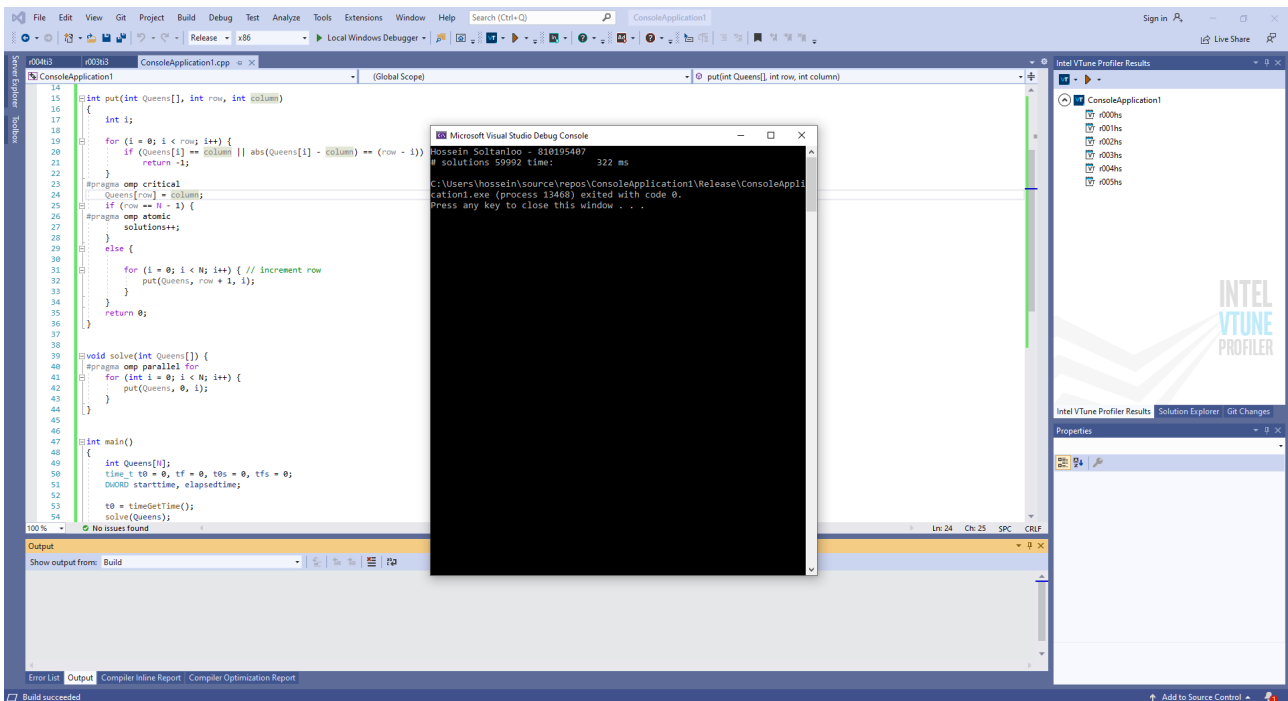


این یعنی این‌که در برنامه‌ی موازی‌سازی شده، Data Race وجود دارد. حالا به کمک Inspector می‌خواهیم این موارد را شناسایی و رفع کنیم. پس از دریافت گزارش این بخش، می‌بینیم که دو مورد در برنامه وجود دارد که در آن Data Race رخ می‌دهد. یکی مربوط به افزایش solutions است و دیگری مربوط به ست کردن Queens[row].

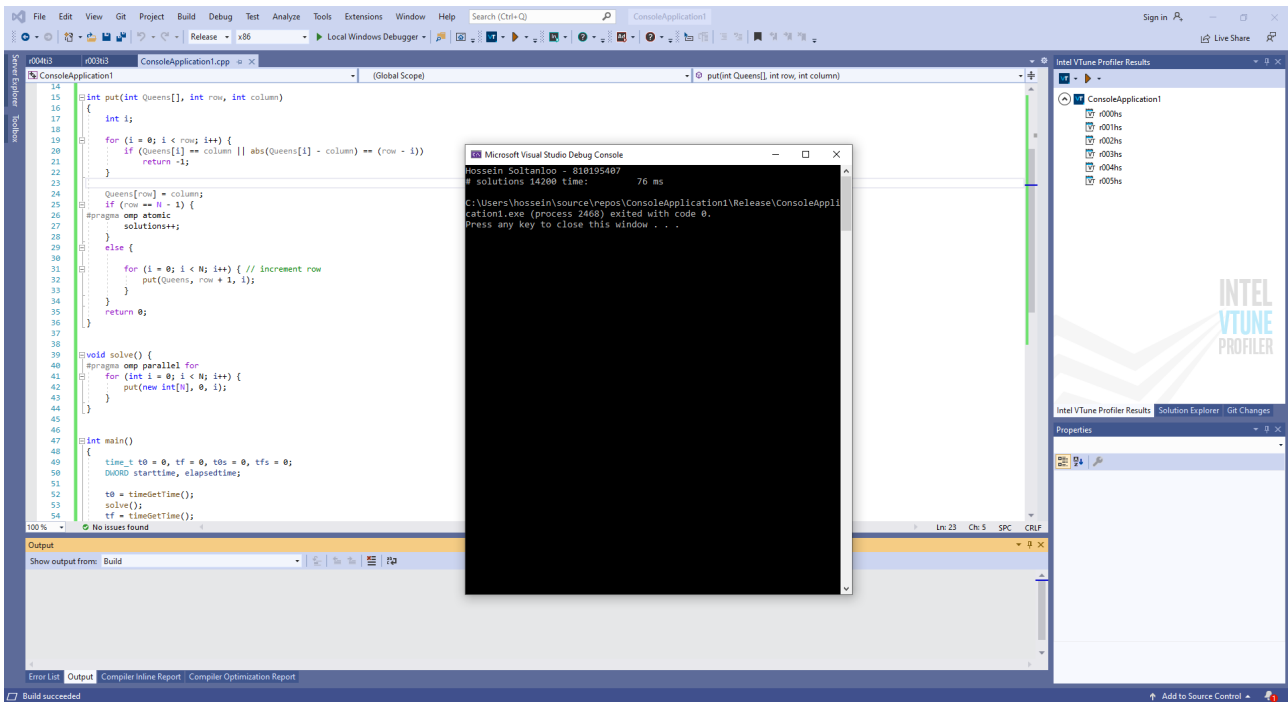




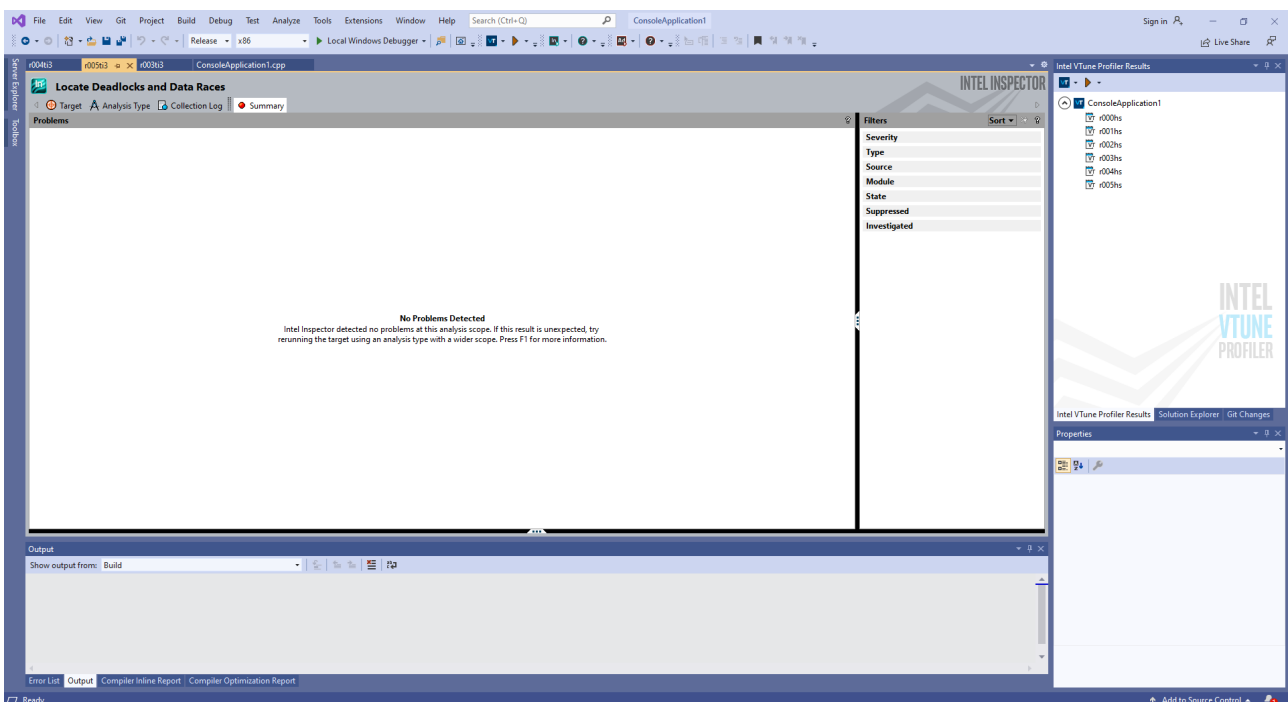
این دو خط را با استفاده از ساختارهای `critical` و `atomic` پوشش می‌دهیم تا دسترسی همزمان چند ریس به این خطوط کنترل شود. اما پس از اجرا می‌بینیم که سرعت بسیار کاهش یافته است و زمان اجرا به ۳۲۲ میلی‌ثانیه رسیده است. دلیل این امر می‌تواند سربار زیاد هم‌گام‌سازی جهت به‌روزرسانی دو متغیر مذکور باشد:



حال با استفاده از یک تغییر ساده در کد می‌توانیم ساختار critical برای دسترسی به Queens[row] را حذف کنیم و هر ریشه روی آرایه‌ی Queens مربوط به خودش کار کند. به این شکل متغیر مشترک از بین رفته و همچنین کنترل دسترسی نیز دیگر نیاز نخواهد بود. با این کار و فقط با ست کردن pragma omp atomic برای solutions++ می‌بینیم که سرعت کاهش چشم‌گیری داشته و زمان اجرا به ۷۶ میلی‌ثانیه رسیده است:

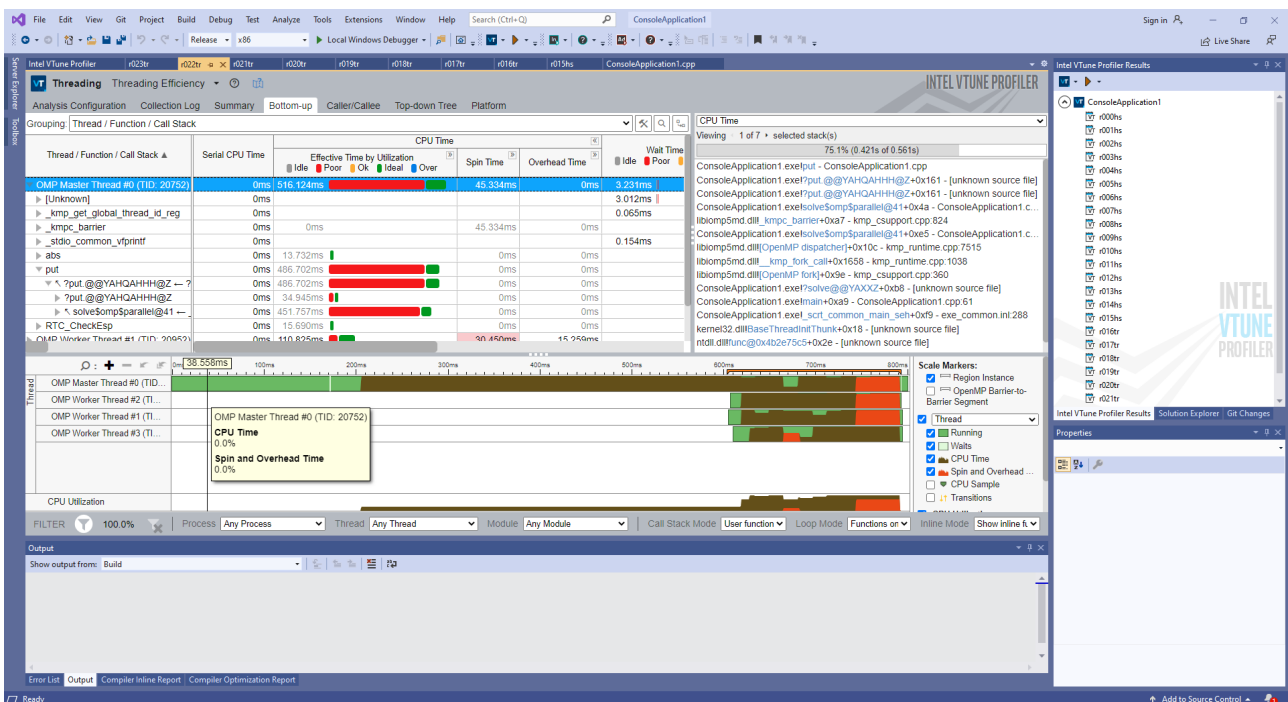


بار دیگر از Inspector استفاده می‌کنیم تا مطمئن شویم توانسته‌ایم موارد مربوط به Data Race را از بین ببریم که همین‌طور هم هست:



تمرین شماره ۶ برنامه‌نویسی موازی

در این مرحله باید دوباره از VTune استفاده کنیم تا ببینیم بهره‌گیری از ریسه‌ها تا چه میزان موثر بوده است. گزارشی که دریافت می‌کنیم طبق تصویر زیر است:

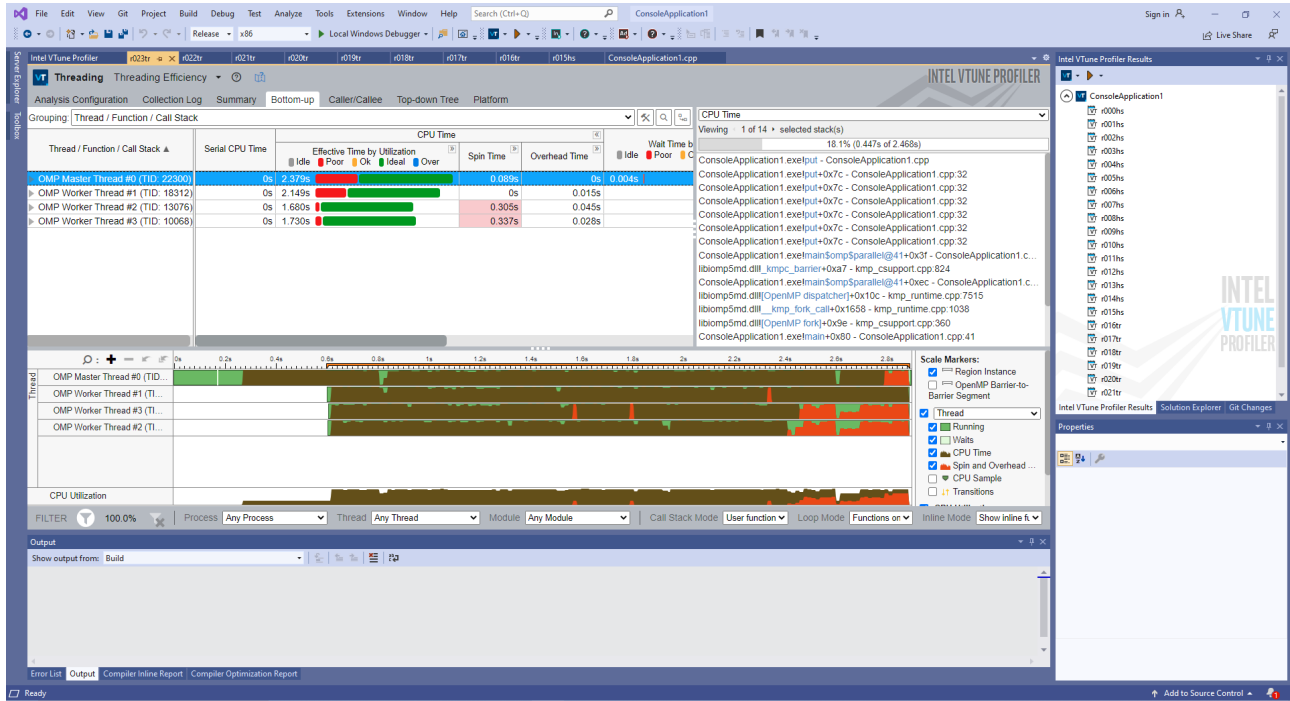


همان‌طور که مشاهده می‌شود، ترد اصلی زمان زیادی را صرف خودش می‌کند و در نهایت در بخش کوچکی از زمان همه‌ی تردها با هم مشغول به کار هستند.

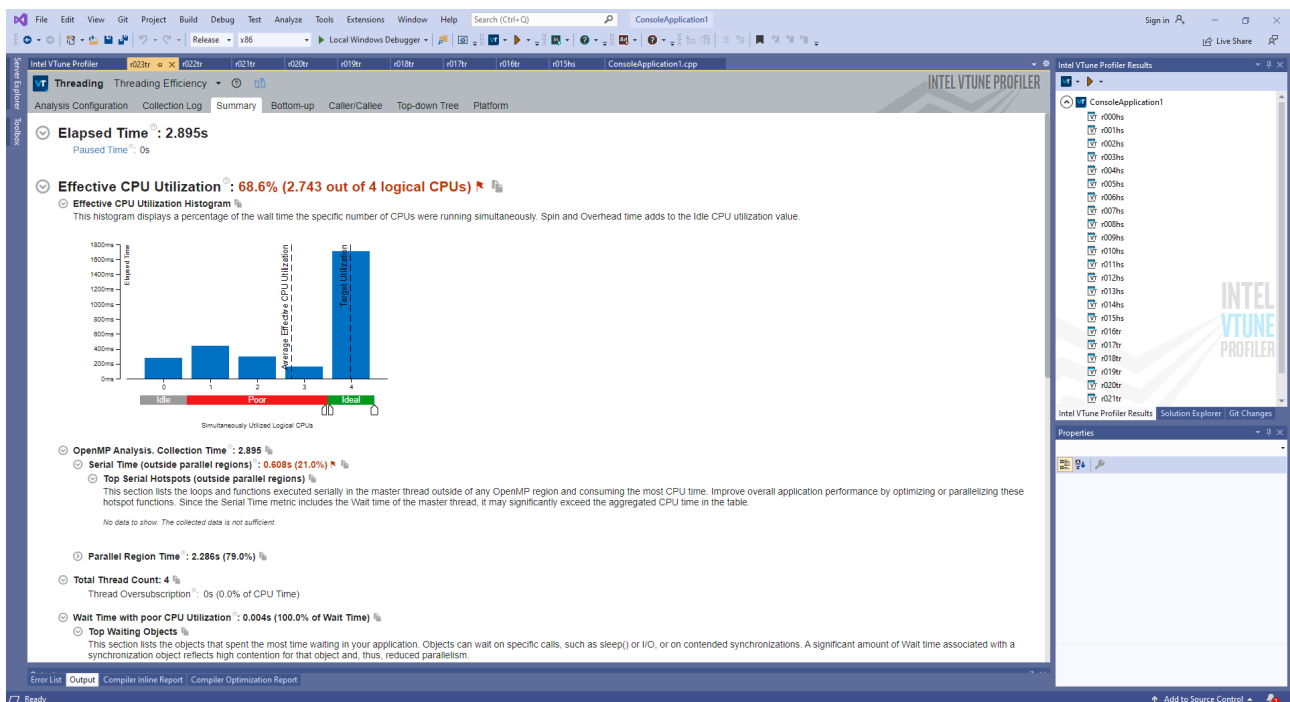
طبق بررسی‌های من، این زمان که تنها یک ترد مشغول به کار است، مربوط به سربار استفاده از `pragma omp paraller` است و کارهایی است که انجام می‌شود تا این موازی‌سازی ممکن شود. به‌طور خاص این زمان در `_kmp_get_global_thread_id_reg` صرف شده است.

از طرفی زمان اصلی محاسبات `solve` نسبت به زمان کل اجرا بسیار کوچک است و از همین رو این سربار بسیار زیاد به نظر می‌رسد.

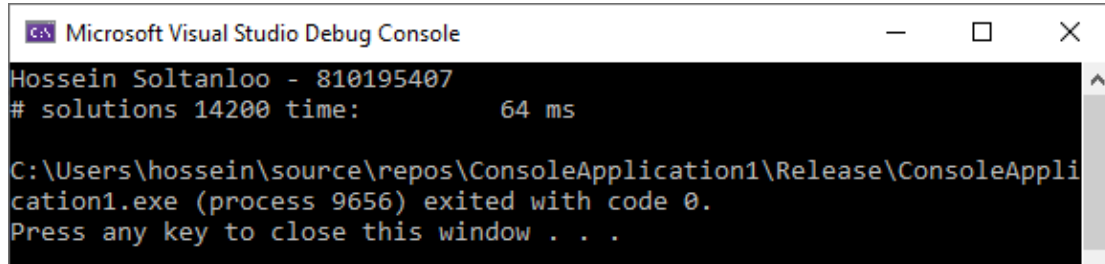
نهایتاً می‌توانیم ادعا کنیم که توانسته‌ایم موازی‌سازی را به حداکثر خود برسانیم و این سر بار نیز قابل حذف نیست. البته برای تأیید این ادعا، می‌توانیم N را بزرگ‌تر قرار دهیم و سپس دوباره این گزارش را بررسی کنیم. این کار را برای N برابر با ۱۴ انجام داده‌ایم که گزارش آن طبق تصاویر زیر است:



همان‌طور که مشاهده می‌شود، برای $N = 14$ این گزارش بسیار خوب به نظر می‌آید و در اکثر زمانی که برنامه مشغول اجرا بوده، هر ۴ ترد هم‌زمان فعال بوده‌اند که آن بخش اول که تنها یک ترد مشغول به کار بوده هم متعلق به همان سرباری است که پیش‌تر به آن اشاره شد



در نتیجه می‌توان ادعا کرد که حداکثر tuning انجام شده است و زمان مربوط به بعد از اجرای این بخش نیز همان زمان مرحله‌ی قبل است چون تغییر خاصی ایجاد نکرده‌ایم. آخرین تغییری که می‌توانیم برای بهبود انجام دهیم این است که از ساختار task به جای for استفاده کنیم که در صورت استفاده از آن مشاهده می‌شود که برای $N = 12$ زمان **۶۴ میلی‌ثانیه** گزارش می‌شود:



```
Microsoft Visual Studio Debug Console
Hossein Soltanloo - 810195407
# solutions 14200 time:          64 ms

C:\Users\hossein\source\repos\ConsoleApplication1\Release\ConsoleApplication1.exe (process 9656) exited with code 0.
Press any key to close this window . . .
```