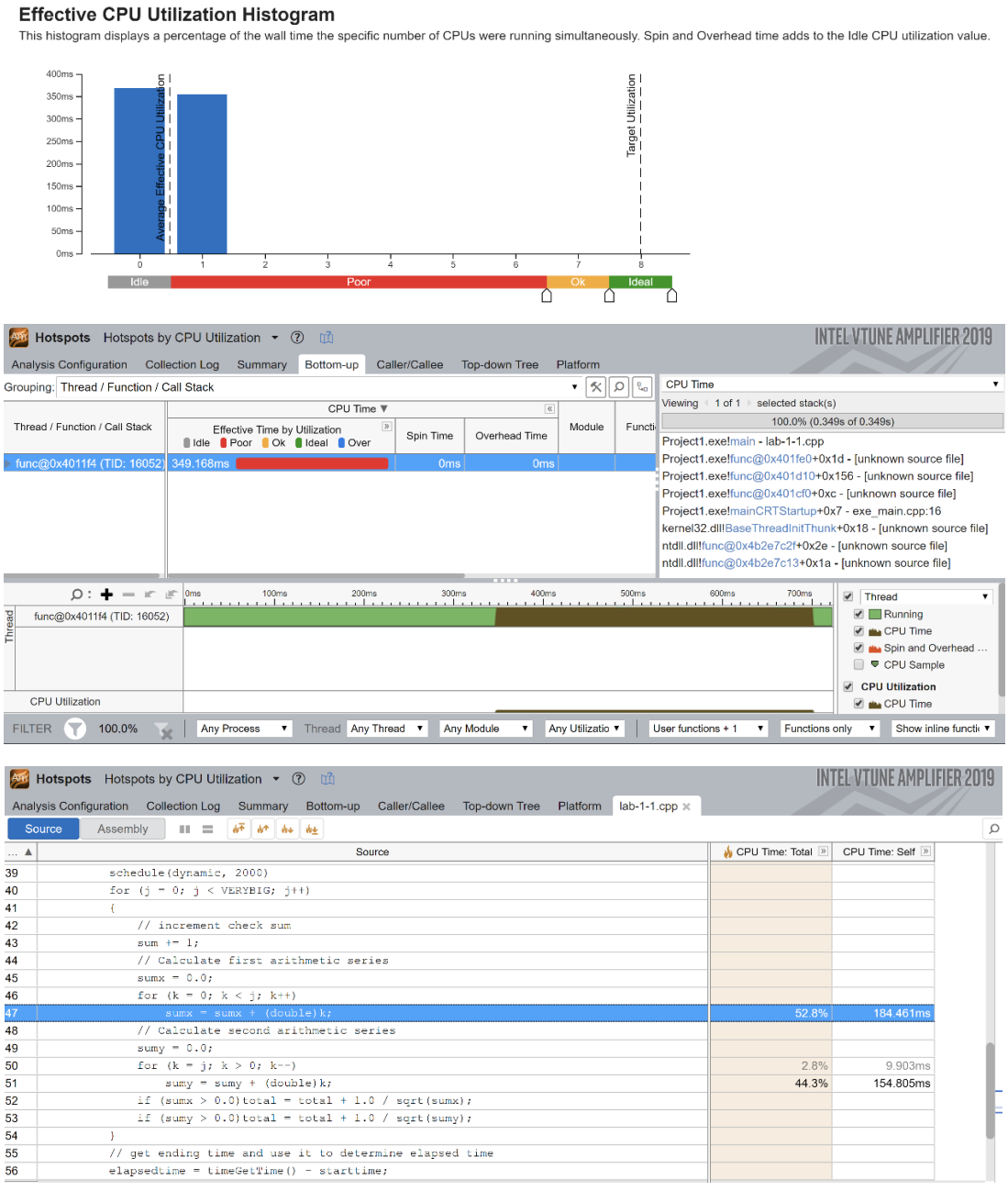
"بسمه تعالی"

گزارش آزمایش دوم برنامه نویسی چندهسته‌ای – آیدا احمدی پارسا – ۹۹۲۳۰۶۹ - زهرا لطیفی – ۹۹۲۳۰۶۹

در این آزمایش تلاش کردیم کد آزمایش اول را به صورت اصولی موازی کنیم. برای این کار Intel Parallel Studio را نصب کرده و با دو ابزار VTune Amplifier و Intel Inspector به آنالیز کد سری پرداختیم تا گلوگاه‌ها را یافته و موازی‌سازی را شروع کرده و تا حد ممکن به بهره‌وری ایده‌آل نزدیک شویم. این کار طبق دستورکار دارای ۴ مرحله آنالیز، پیاده‌سازی، دیباگ و Tune است. در وهله اول برای کاهش زمان اجرا تعداد تکرارهای حلقه خروجی را به یک مرتبه و همینطور متغیر VERYBIG را به ۱۰۰۰۰ کاهش دادیم. آنالیز را با VTune آغاز کردیم تا خطوطی از کد که بیشترین زمان را به خود اختصاص داده‌اند پیدا کنیم. حاصل به شرح زیر بود: (شاهدیم که با بهره‌برداری ایده‌آل از CPU چه قدر فاصله داریم.)

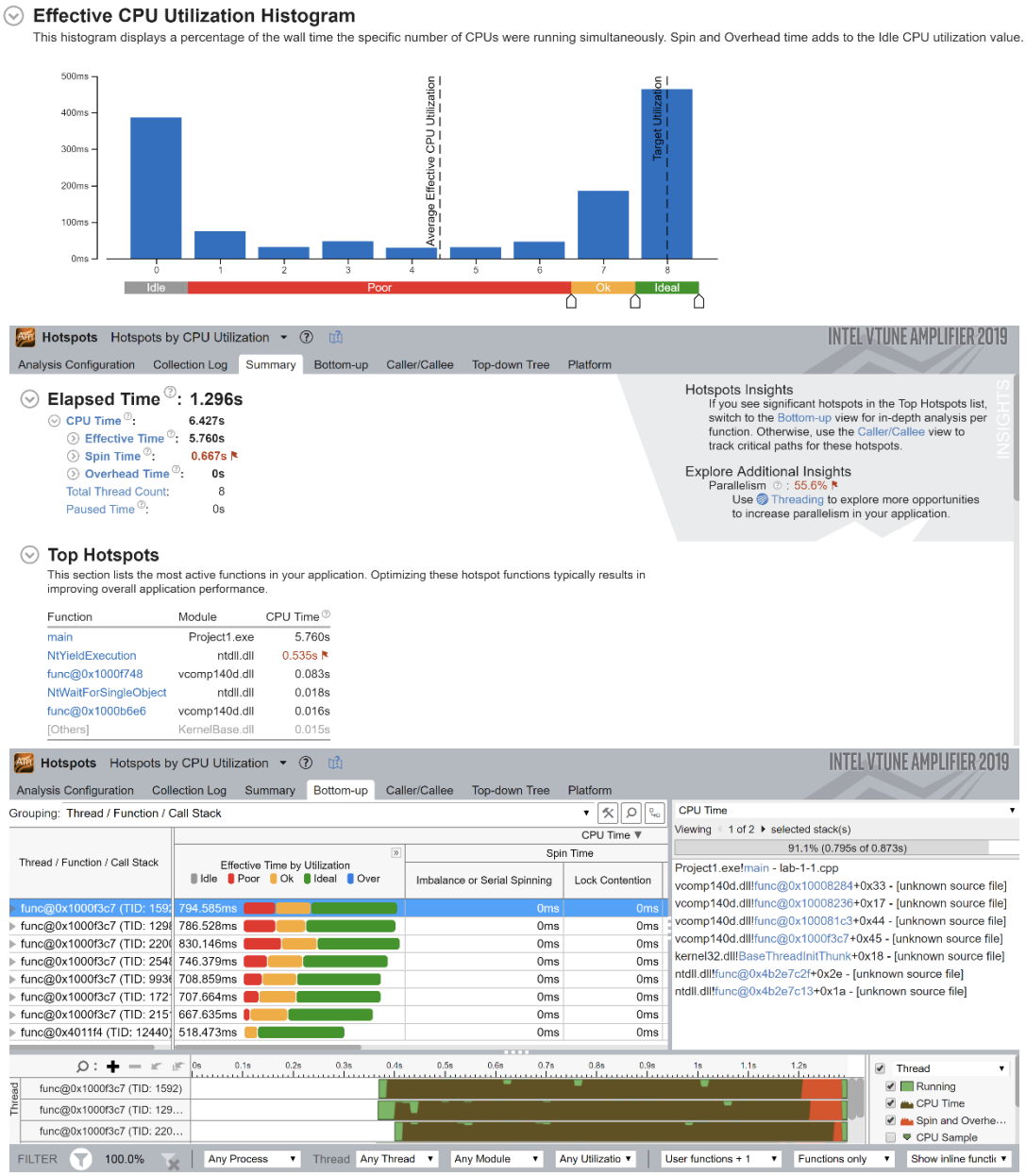


پس از یافتن قسمت‌هایی که زمان اجرای طولانی داشتند، سعی کردیم تا این بخش‌ها را موازی کنیم. این کار را با قرار دادن

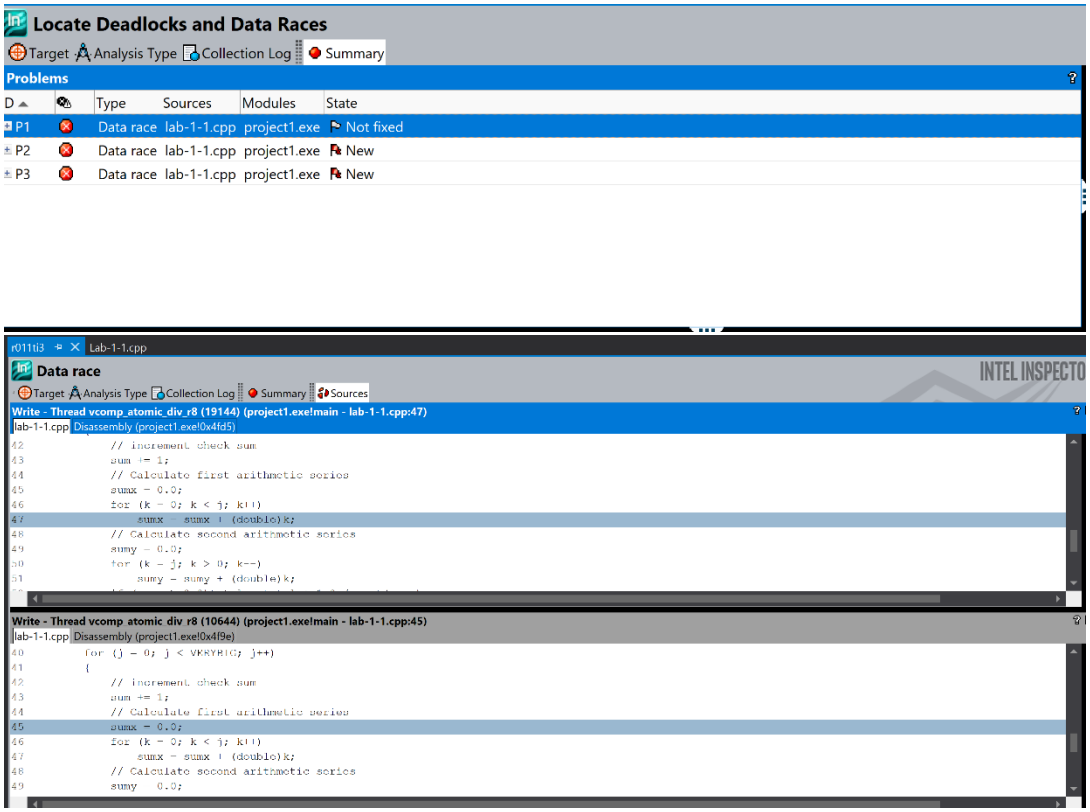
```
#pragma omp parallel for
```

پیش از حلقه `j` شروع شد.

مجددا با VTune آنالیز کرده و دیدیم که زمان اجرای برنامه بیشتر از قبل شد!! و بدتر اینکه مقادیر متغیرهای `sum` و `total` هم دیگر صحیح محاسبه نمی‌شوند.



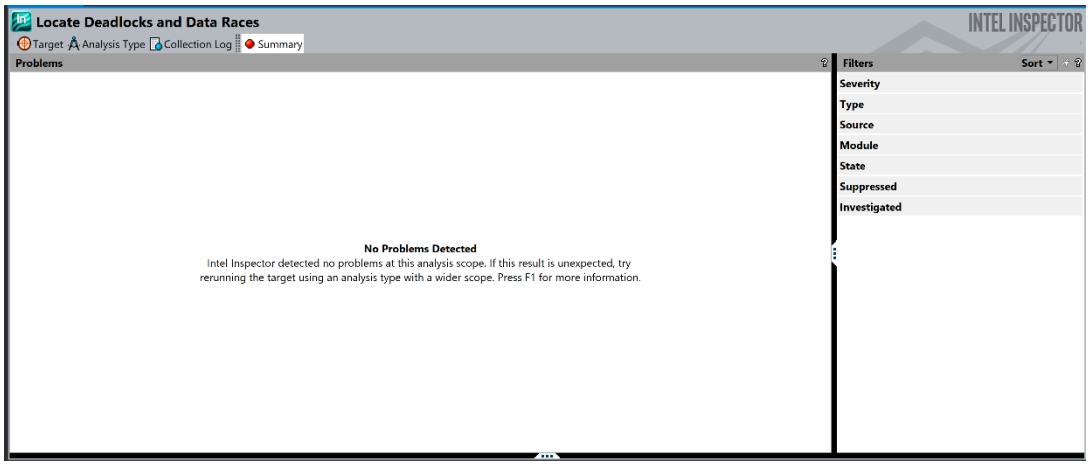
پس به سراغ مرحله debug رفتیم تا اشکال موازی‌سازی انجام شده را پیدا کنیم. این کار با ابزار `Intel Inspector` انجام شد. البته برای انجام این بخش متغیر VERYBIG را به ۱۰۰۰ کاهش دادیم.



شاهدیم که برروی متغیرهای sumx، sum و k خطای Data Race داریم. در این بخش متوجه شدیم که باید تا جایی که می‌توانیم متغیرهایی که این مشکل را دارند را private کنیم. و با عبارت reduction مانع از Data Race برای متعیرهای مشترک بین نخها مثل sum و total شویم. پس کد را به صورت زیر اصلاح کردیم:

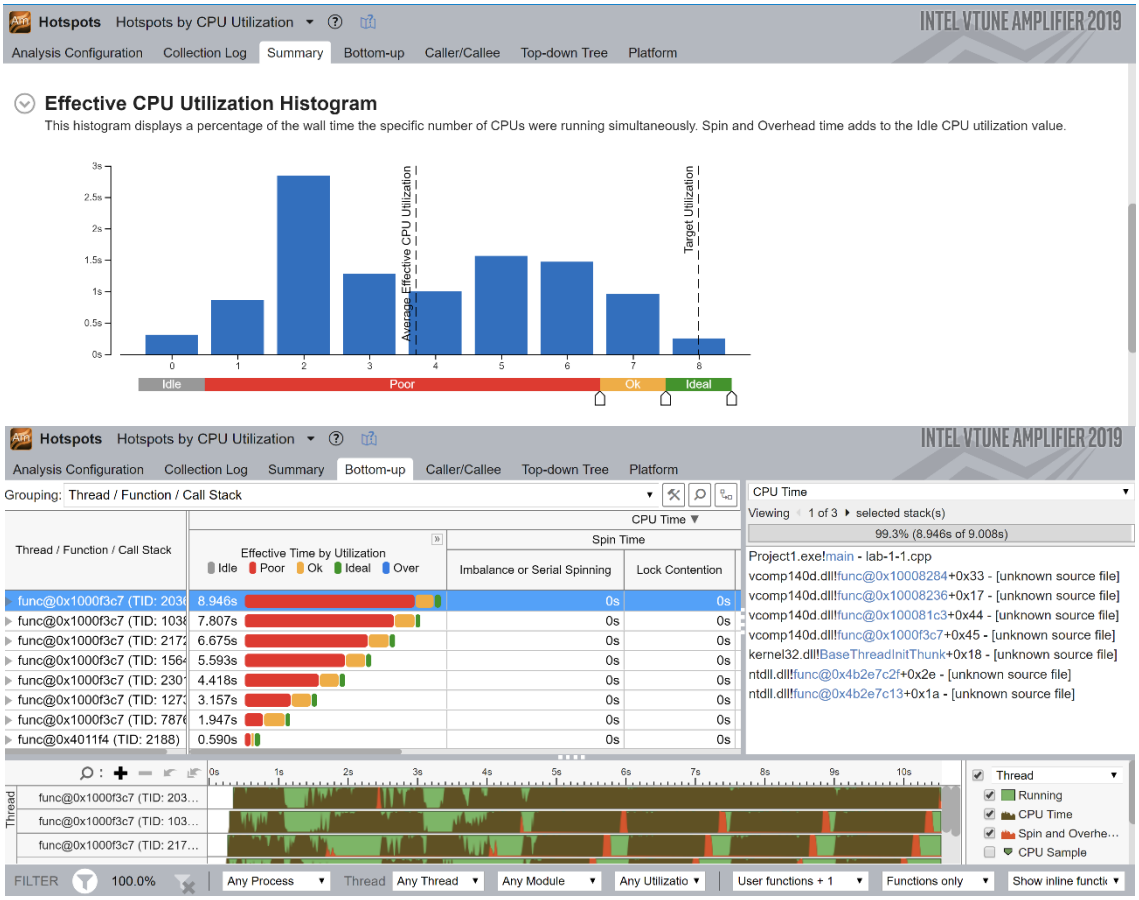
```
#pragma omp parallel for private(sumx, sumy, k) reduction(+:sum, total)
```

دیدیم که خطاها رفع شدند:



در اینجا بخش Debug به پایان می‌رسد اما مرحله Tune هنوز باقی مانده. در این بخش به آنالیز concurrency برای ارزیابی concurrency نخها و استفاده از CPU پرداختیم. درواقع تلاش کردیم تا به تقسیم متعادل load در بین نخها با تنظیم chunk size در عبارت Schedule(dynamic) دست پیدا کنیم.

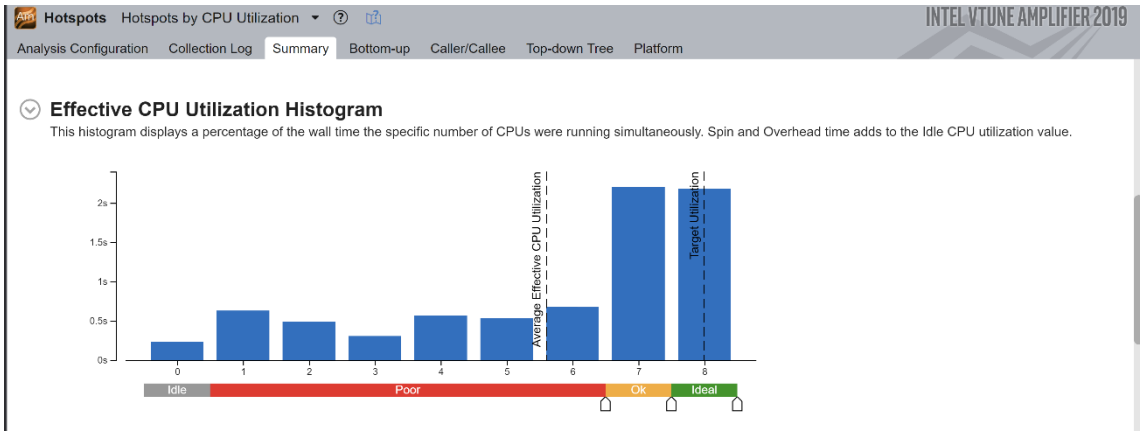
برنامه را دیگر بار با VTune آنالیز کردیم (با افزایش مجدد VERYBIG تا ۵۰۰۰۰) و دیدیم که کار به صورت نامتوازن بین نخ‌ها تقسیم شده. به شکلی که برخی نخ‌ها زودتر عملیات مربوط به خود ار تمام کرده و ناچارا در انتظار سایر نخ‌ها بیکار می‌مانند. در حقیقت مدت زمان بسیار کمی هر ۸ نخ با هم درحال انجام کارند.

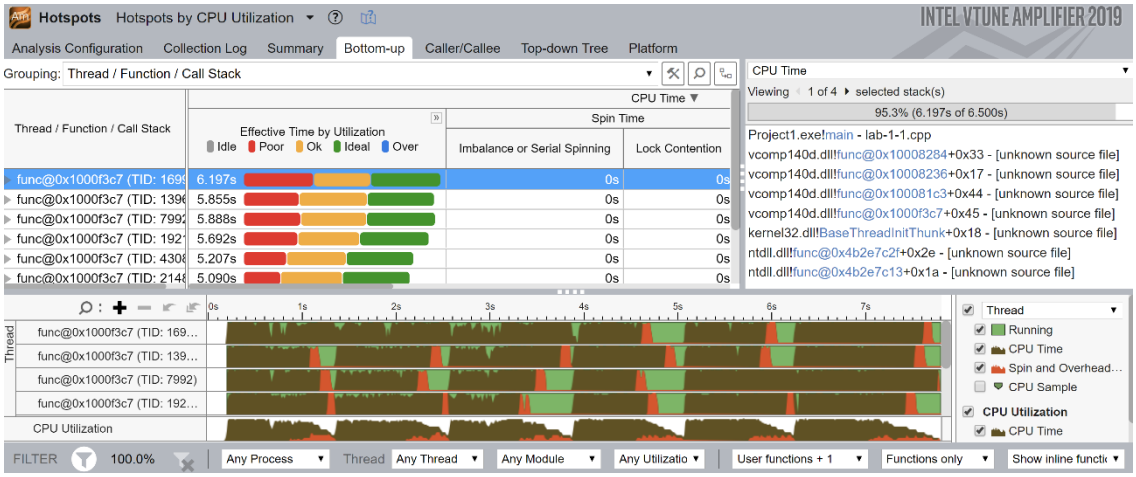


پس کد را به شکل زیر با chunk size = 2000 و schedule(dynamic) اصلاح کردیم:

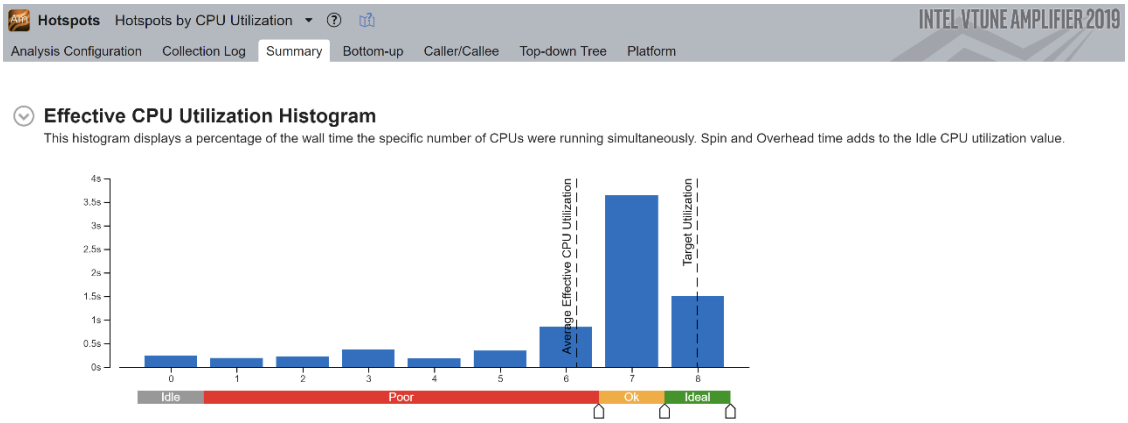
```
#pragma omp parallel for private(sumx, sumy, k) reduction(+:sum, total)
schedule(dynamic, 2000)
```

نتیجه به شکل زیر تغییر کرد.





یک بار دیگر هم با `chunk size = 1000` همین کار را تکرار کردیم و نتایج بهتری درباره تقسیم متوازن کار و بهره‌برداری ایده‌آل از CPU گرفتیم:



نهایتا یک بار دیگر هم با `chunk size = 500` همین کار را تکرار کردیم و نتایج بهتری درباره تقسیم متوازن کار و بهره‌برداری ایده‌آل از CPU گرفتیم: