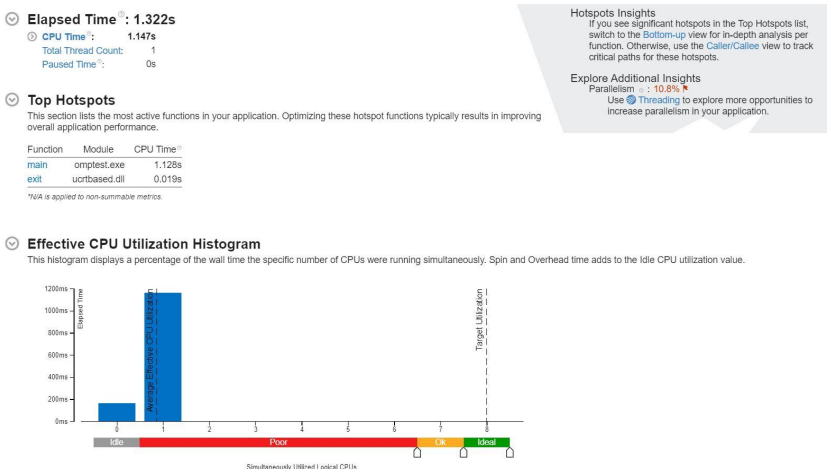
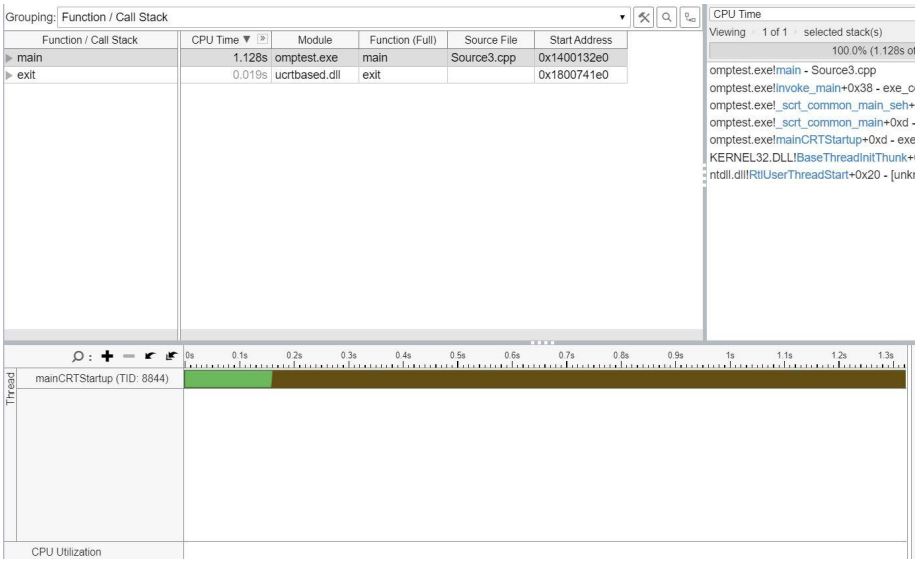
اجرای حالت سریال:

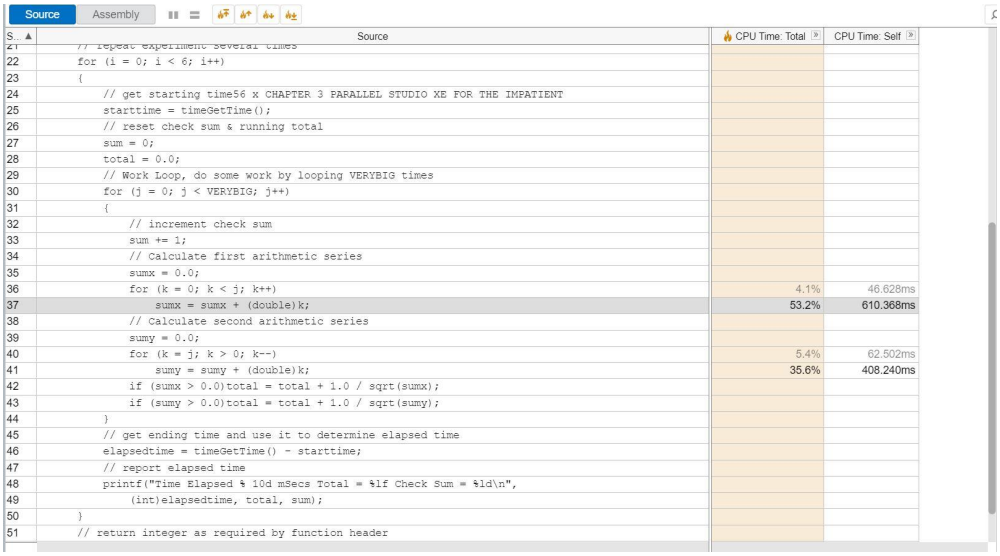
ابتدا کد سریال را اجرا کردیم.



همانطور که مشاهده می‌کنید، برنامه از یک هسته استفاده کرده و آن را به خوبی استفاده نمی‌کند.



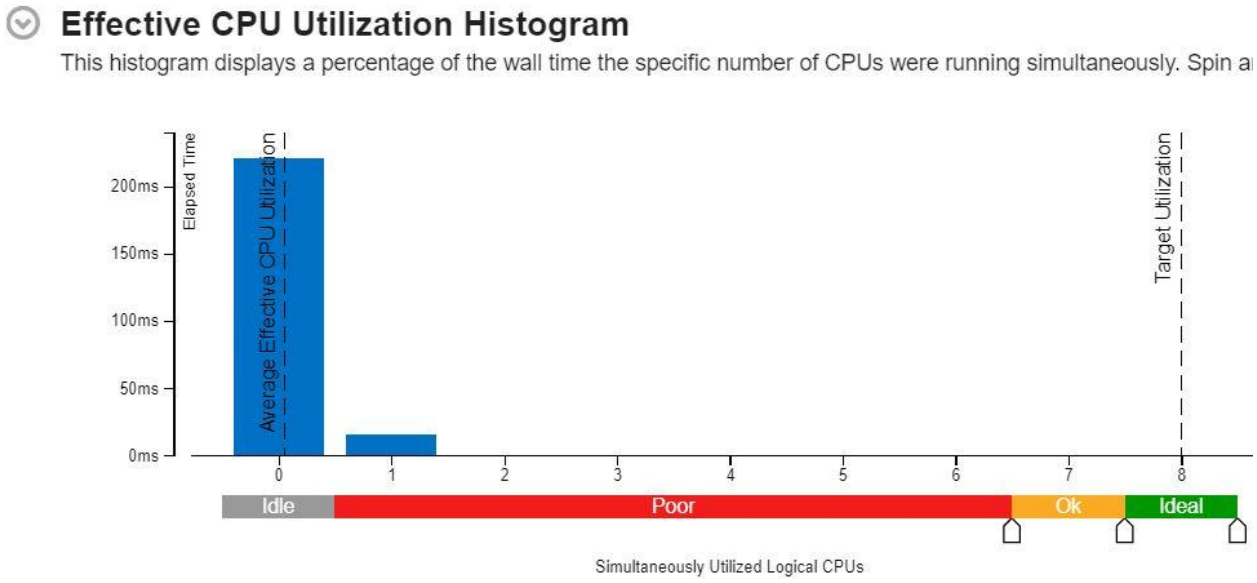
برنامه به غیر از بخش اول در بقیه اوقات به صورت غیر مناسبی از یک هسته استفاده می‌کند.



همانطور که در آزمایش‌های قبلی اشاره شد، اکثر زمان اجرای برنامه در دو خط ۳۷ و ۴۱ می‌گذرد.

اجرای موازی سازی با مشکل:

پس از قرار دادن pragma در بالای حلقه، حلقه موازی سازی شده ولی تعداد زیادی race condition بوجود خواهد آمد.

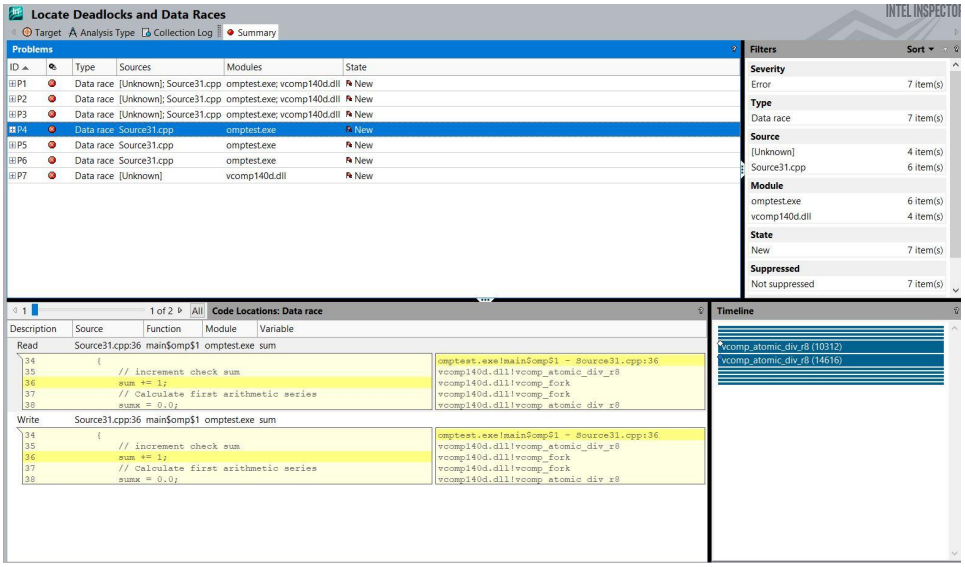


همانطور که مشاهده می‌کنید، تقریبا اصلا استفاده مناسبی از CPUها نشده است.

امیرحسین پاشایی هیر ۹۷۳۱۰۱۳ – محمد مهدی منتظر ۹۷۳۱۱۲۰

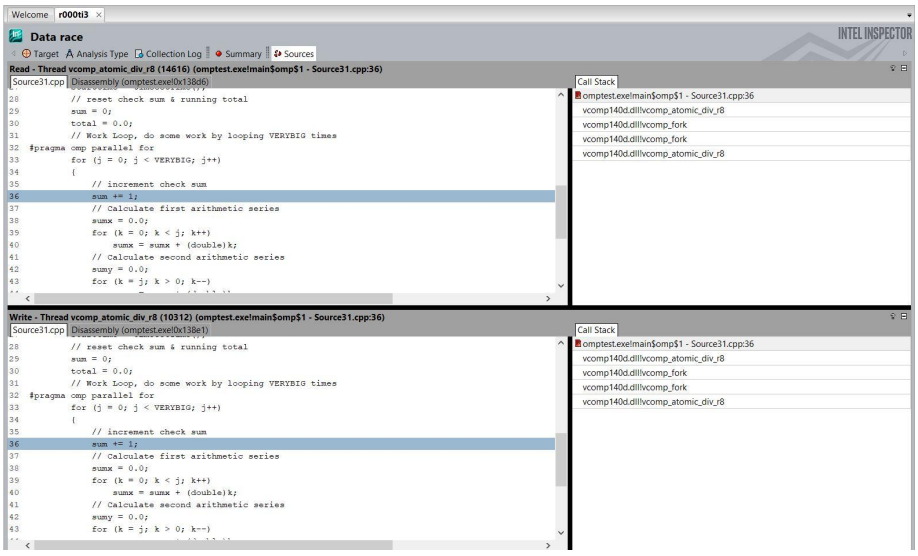
تشخیص مشکلات کد موازی سازی شده:

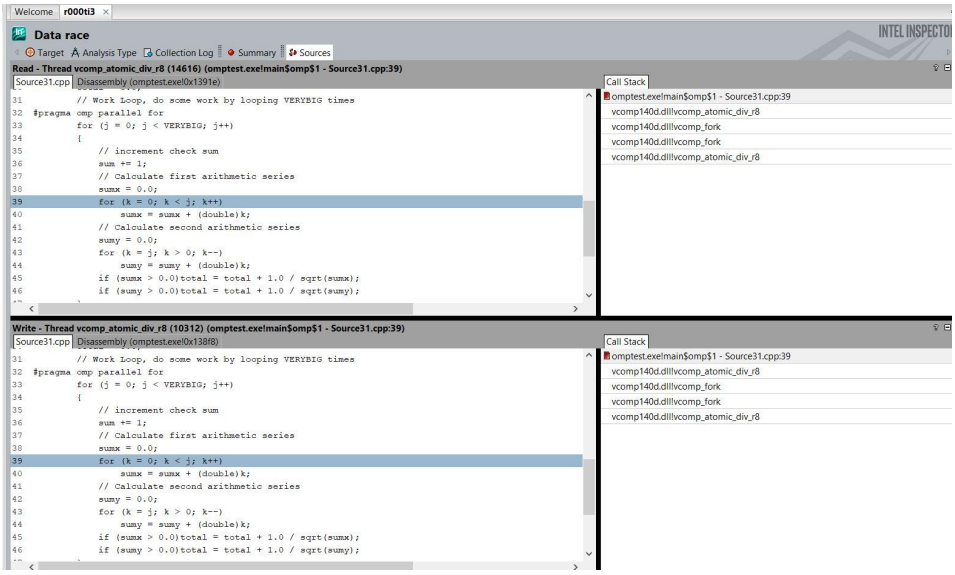
پس از اجرای کد موازی سازی شده و تحلیل آن توسط inspector مشاهده می‌کنیم:

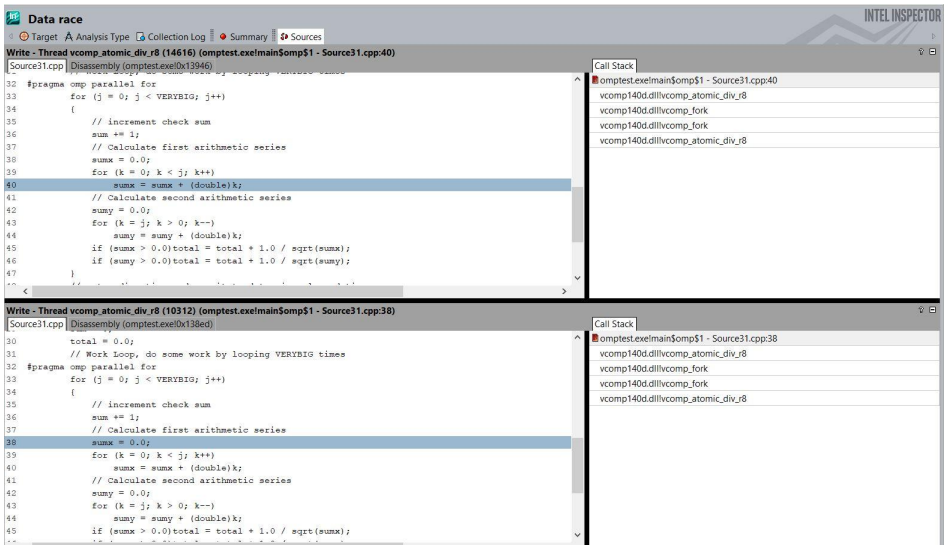


لیست مشکلات و مکان‌هایی که race condition پیش آمده است در عکس مشخص است.

مشکلاتی که در کد وجود داشت عبارت‌اند از:



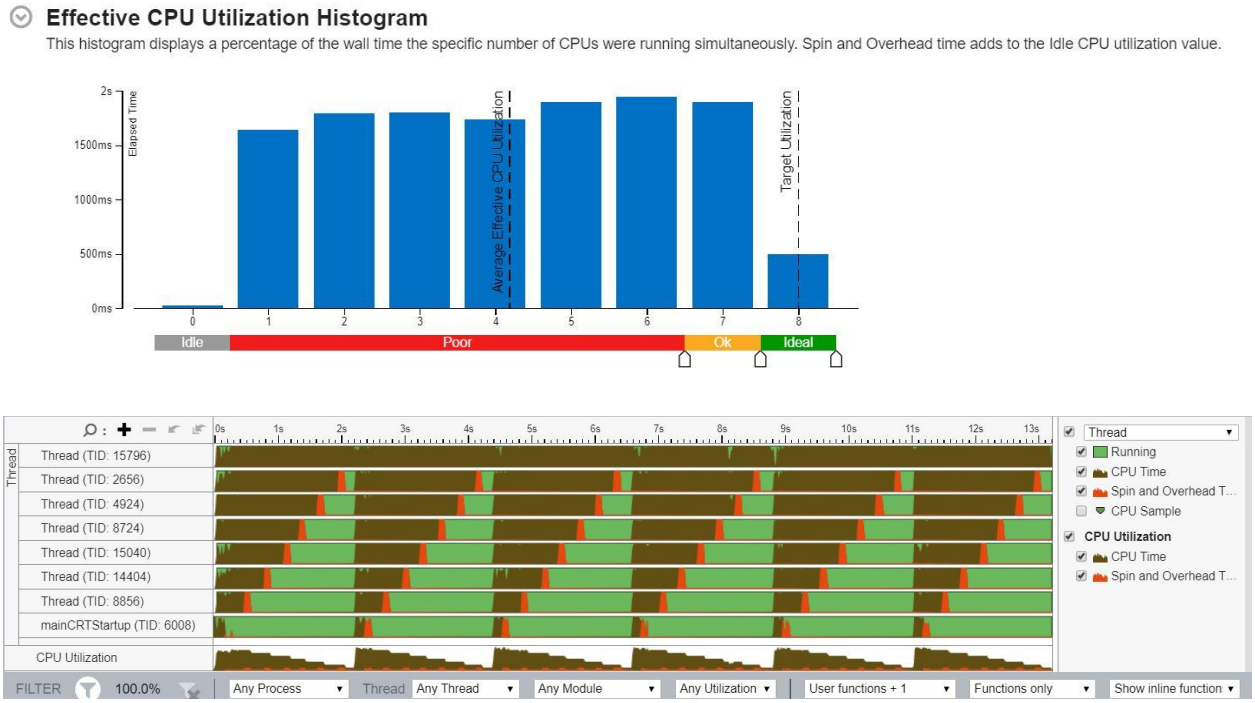




همانطور مشکلی که بوجود آمده با private کردن sumx, sumy و k درست می‌شود، همچنین برای درست شدن checksum باید مقدار sum را به صورت جمع و total را داخل reduction قرار داد.

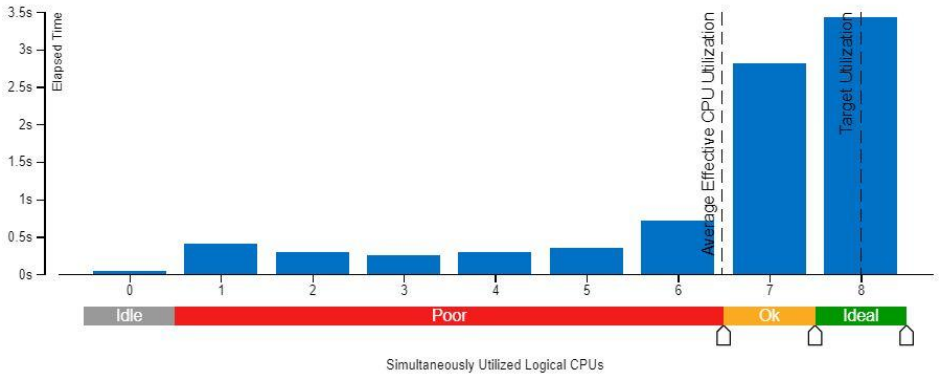
اجرای کد موازی سازی درست شده:

پس از درست کردن کد، آن را اجرا می‌کنیم.

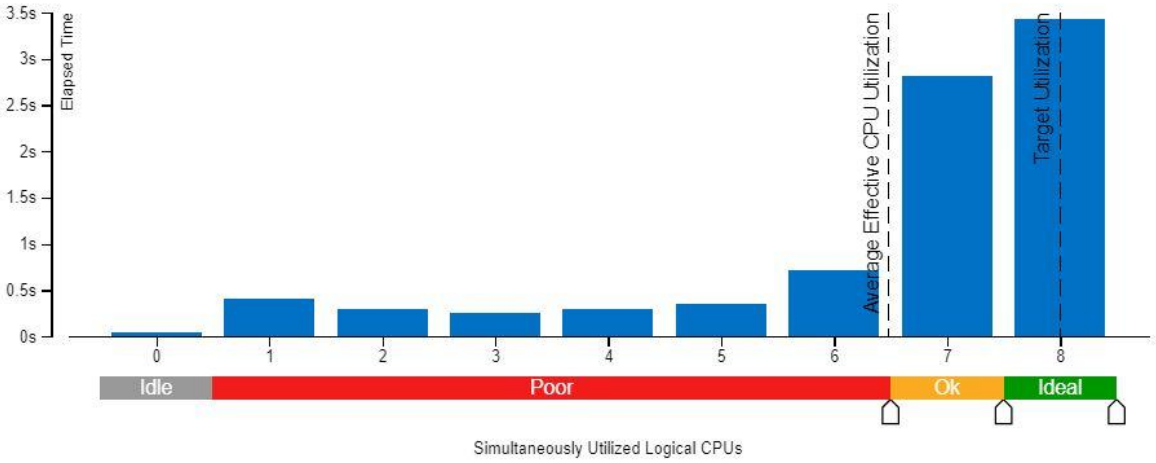


همانطور که می‌بینید از همه‌ی CPUها استفاده شده اما نه به خوبی، دلیل این اتفاق این است که OpenMP به صورت مساوی تقسیم می‌کند بازه‌ی حلقه را اما بازه‌های با طول مساوی در برنامه، حجم محاسبات مساوی نمی‌خواهند و این حجم صعودی می‌باشد، به همین دلیل اگه با استفاده از schedule، بازه ها را به تعداد بیشتر با طول کمتر تقسیم کنیم، تعادل بین هسته‌ها بهتر رعایت می‌شود.

اجرای کد با schedule dynamic با بازه‌های ۲۰۰۰تایی:



اجرای کد با schedule dynamic با بازه‌های ۱۰۰۰تایی:



همانطور که می‌بیند، استفاده از CPUها بهتر شد، همچنین بازه‌های ۱۰۰۰ تایی بهتر از ۲۰۰۰ تایی می‌باشد زیرا تعداد بازه‌ها بیشتر شده و تعادل بین هسته‌ها بهتر رعایت می‌شود.