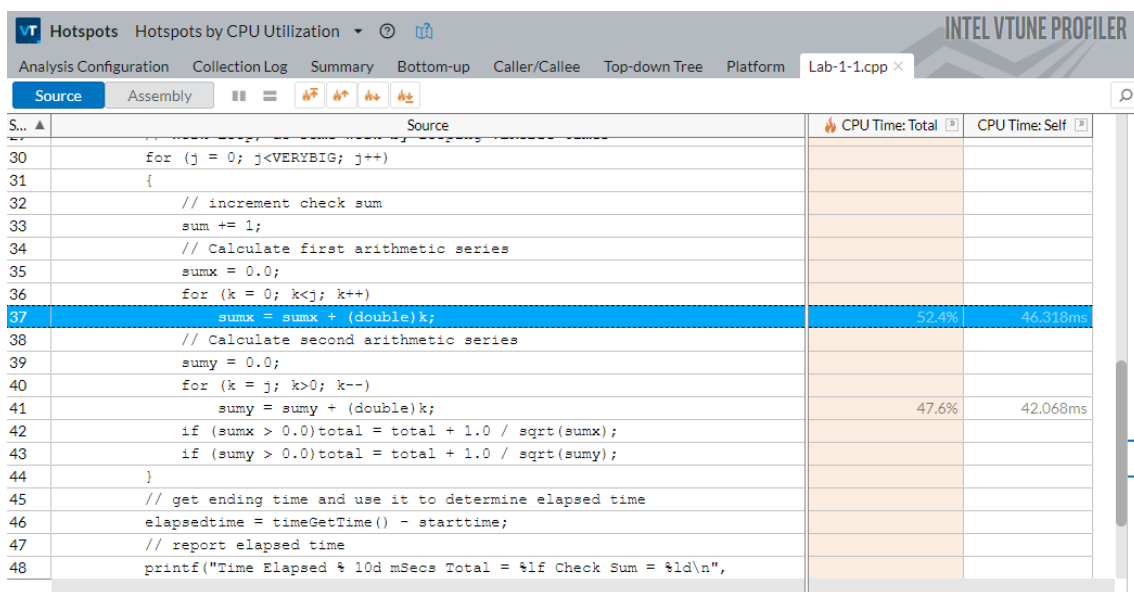


۱ آنالیز کد سریال

ابتدا Hotspot های برنامه را بررسی می‌کنیم تا متوجه شویم چه قسمت‌هایی از برنامه بیشترین زمان اجرا را به خود اختصاص داده‌اند.

برای این کار با تبدیل مقدار VERYBIG به ۱۰ هزار و تعداد تکرارهای برنامه به ۱، آن را به کمک VTune آنالیز می‌کنیم. پس از آنالیز شدن برنامه، با دو بار کلیک بر روی تابع main، به سورس کد برنامه (شکل ۱) می‌رویم و خطوطی که بیشترین زمان اجرا را به خود اختصاص داده‌اند را می‌بینیم.



Source	CPU Time: Total	CPU Time: Self
for (j = 0; j<VERYBIG; j++)		
{		
// increment check sum		
sum += 1;		
// Calculate first arithmetic series		
sumx = 0.0;		
for (k = 0; k<j; k++)		
sumx = sumx + (double)k;	52.4%	46.318ms
// Calculate second arithmetic series		
sumy = 0.0;		
for (k = j; k>0; k--)		
sumy = sumy + (double)k;	47.6%	42.068ms
if (sumx > 0.0) total = total + 1.0 / sqrt(sumx);		
if (sumy > 0.0) total = total + 1.0 / sqrt(sumy);		
}		
// get ending time and use it to determine elapsed time		
elapsedtime = timeGetTime() - starttime;		
// report elapsed time		
printf("Time Elapsed % 10d mSecs Total = %lf Check Sum = %ld\n",		

شکل ۱: نتایج آنالیز با VTune

۲ موازی‌سازی به کمک OpenMP

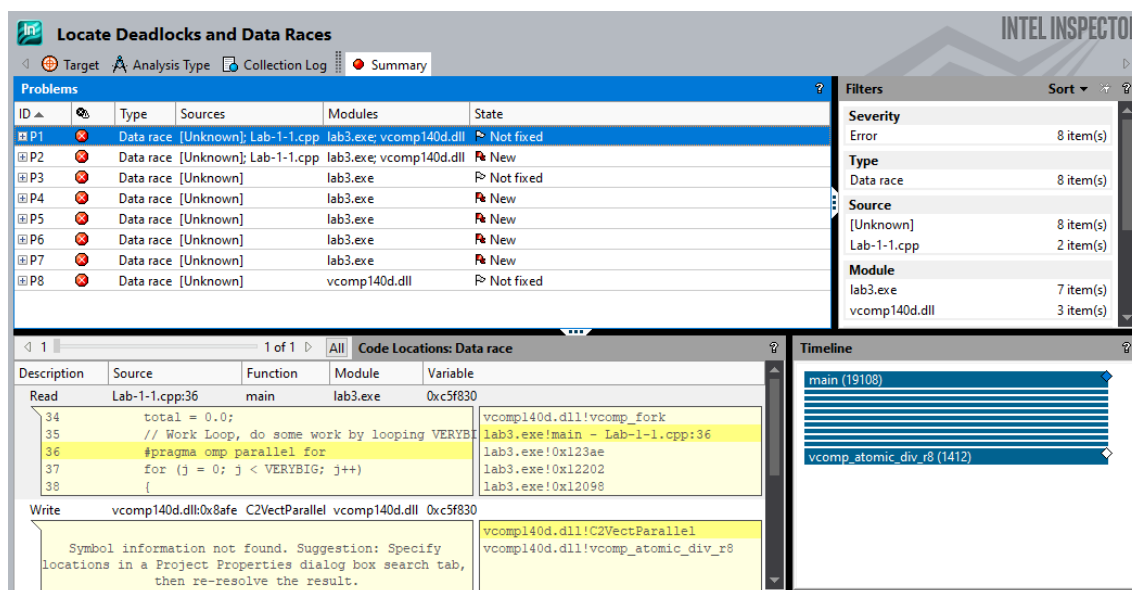
پس از مشاهده قسمت‌هایی از برنامه که زمان اجرای آن‌ها طولانی‌تر است، سعی می‌کنیم با موازی‌سازی این بخش‌ها تسریع بگیریم. با گذاشتن خط زیر پیش از حلقه work، اجراهای آن را موازی می‌کنیم.

```
#pragma omp parallel for
```

مشاهده می‌شود که در این حالت زمان اجرای برنامه حتی بیشتر شده است و برنامه به درستی اجرا نمی‌شود (مقدار متغیرهای sum و total درست نیست).

۳ دیباگ و رفع خطاها

با ابزار Inspector و کاهش دادن مقدار VERYBIG به هزار، برنامه را تحلیل می کنیم. همانطور که در شکل ۲ می بینیم، برنامه دارای شرایط مسابقه برای متغیرهای مشترک (sumx, total, sum, sumy و k) است.



شکل ۲: وجود شرایط مسابقه برای متغیرهای مشترک بین نخها

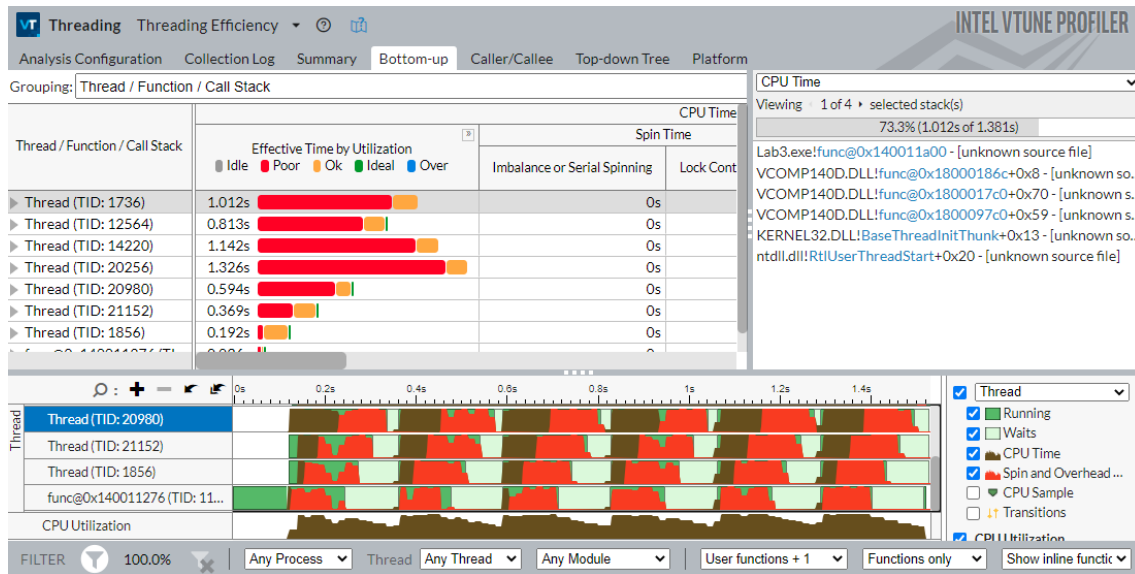
با استفاده از خط زیر، متغیرهایی که می توانند خصوصی باشند را خصوصی می کنیم. همچنین متغیرهایی که باید مقدار آنها بین همه نخها مشترک باشد را می توانیم با استفاده از عبارت reduction، جلوی شرایط مسابقه آن را بگیریم.

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

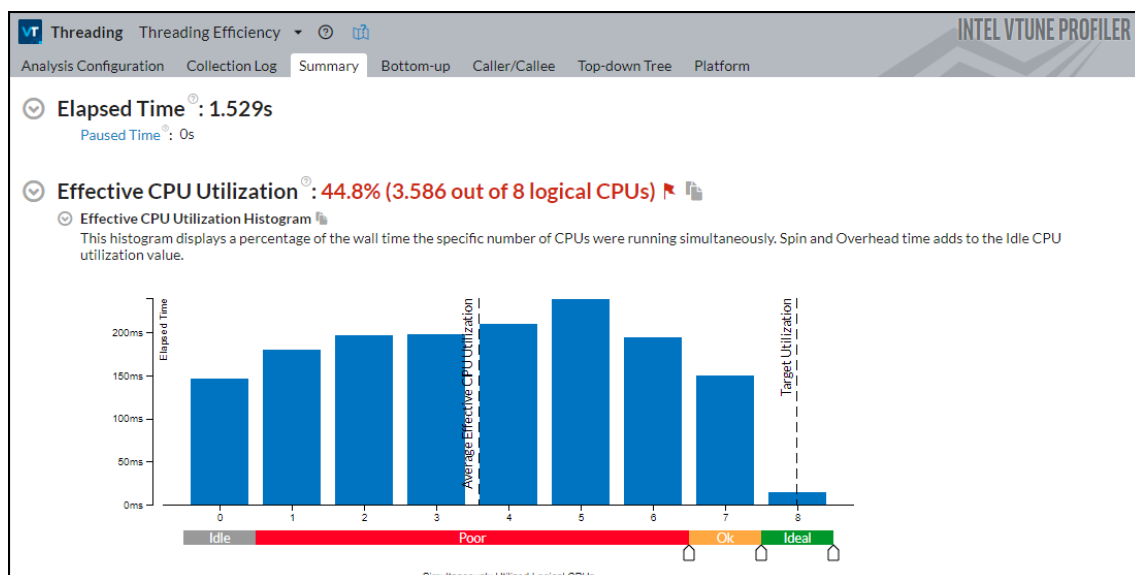
۴ تنظیم و سرعت بخشیدن به برنامه OpenMP

با آنالیز برنامه به کمک VTune مطابق شکل ۳، مشاهده می کنیم که کار به صورت نامتوازن بین نخها پخش شده است. در واقع برخی نخها زودتر کارشان به اتمام می رسد و به ناچار منتظر دیگر نخها می ماند. همچنین در شکل ۴ می بینیم که مدت زمان بسیار کمی هر ۸ نخ همزمان با هم فعال هستند (ستون سمت

راست) و این به معنی استفاده نامناسب از تمام توان پردازشی پردازنده است.



شکل ۳: پخش نامتوازن کار بین نخ‌ها

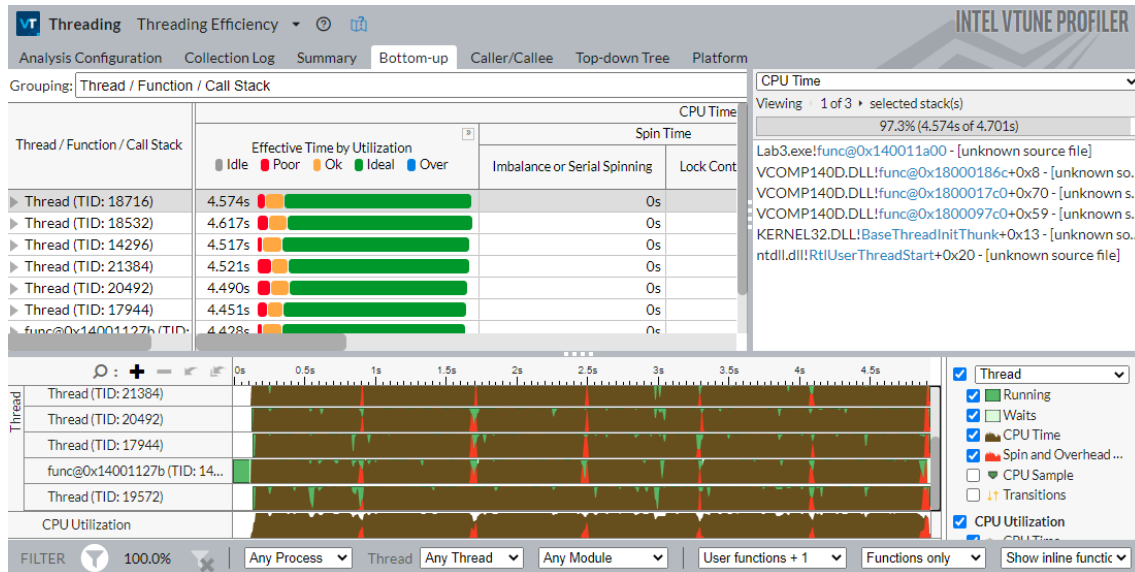


شکل ۴: عدم استفاده از تمام توان پردازشی پردازنده به علت پخش نامتوازن کارها

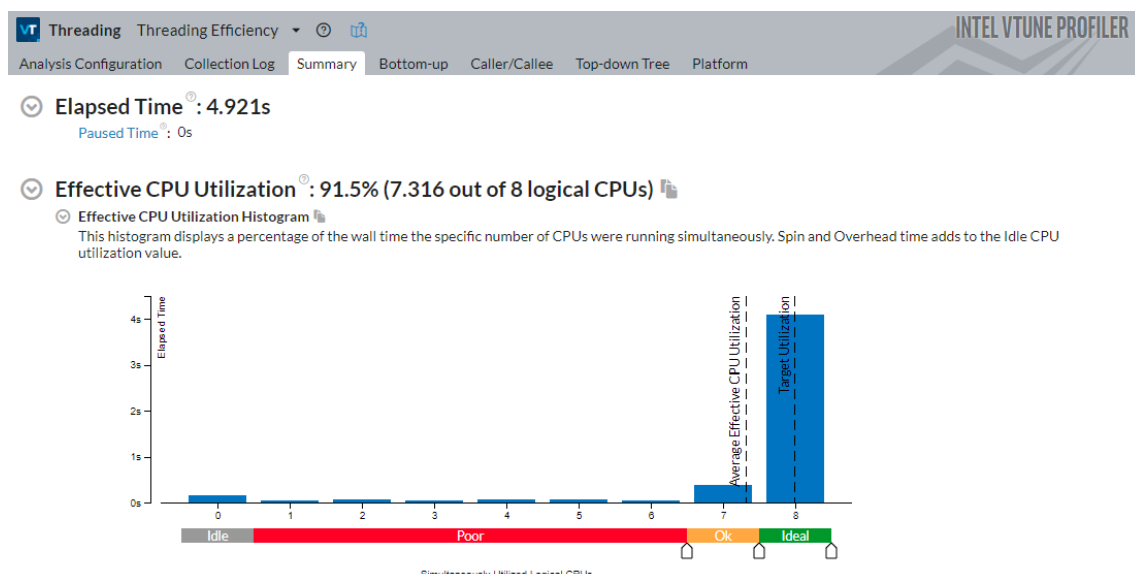
این پخش نامتوازن به علت محاسبات بیشتر نخ‌های پایانی در حلقه k است. با استفاده از عبارت زیر می‌توانیم پخش کارها را متوازن کنیم.

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

با آنالیز مجدد کارکرد نخ‌ها با استفاده از VTune، می‌بینیم که پخش کارها بسیار متوازن‌تر است و به بهره‌وری ایده‌آل بسیار نزدیک‌تر شده‌ایم (شکل‌های ۵ و ۶)



شکل ۵: پخش بسیار بهتر کارها پس از استفاده از schedule(static) و با chunk size برابر ۲۰۰۰



شکل ۶: نزدیک شدن به میزان بهره‌وری ایده‌آل پردازنده