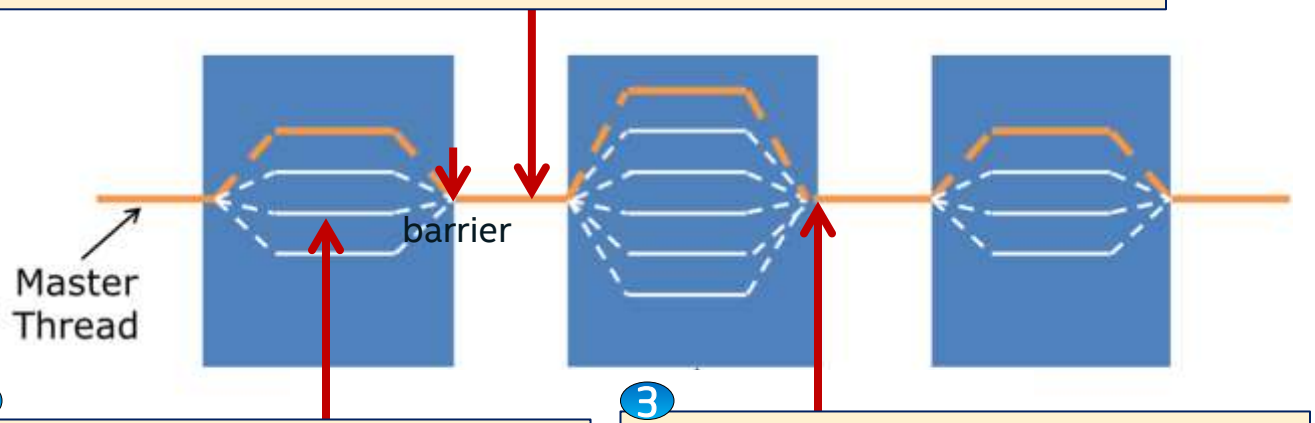
Intel® VTune Amplifier XE
Analysis of OpenMP applications

Kirill Rogozhin

Major reasons why working threads wait

1. When the master thread is executing a serial region, the worker threads are in the OpenMP runtime waiting for the next parallel region



2. When synchronization objects are used inside a parallel region, **threads can wait on a lock release**, contending with other threads for a shared resource **(Synchronization on locks)**

3.
- When a thread finishes a parallel region, it waits at a barrier for the other threads to finish. (**Load imbalance**)
- The number of loop iterations < the number of working threads so several threads from the team are waiting at the barrier not doing useful work at all **(Not enough parallel work)**

VTune Amplifier XE OpenMP Analysis

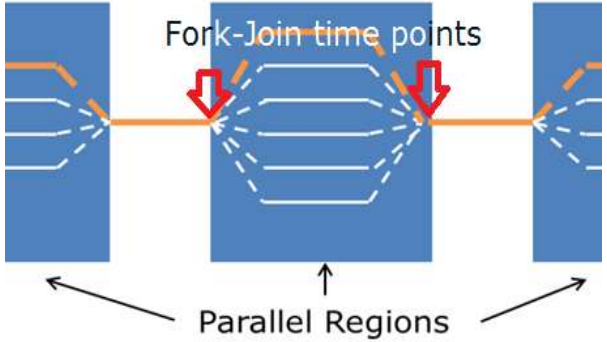
- Tracing of OpenMP is used to provide region/work sharing context

- Provided to VTune by Intel OpenMP Runtime:

- **Fork-Join time points of parallel regions with number of working threads**

→

- Overhead of tracing can be substantial– used carefully per region instance on region fork-join points



- Sampling to determine different kinds of overhead, synchronization spinning etc.

- Any type of VTune analysis that support CPU time calculation (such as hotspots, advanced-hotspots with or without stacks, etc.)
- With Hotspot Viewpoint selected →

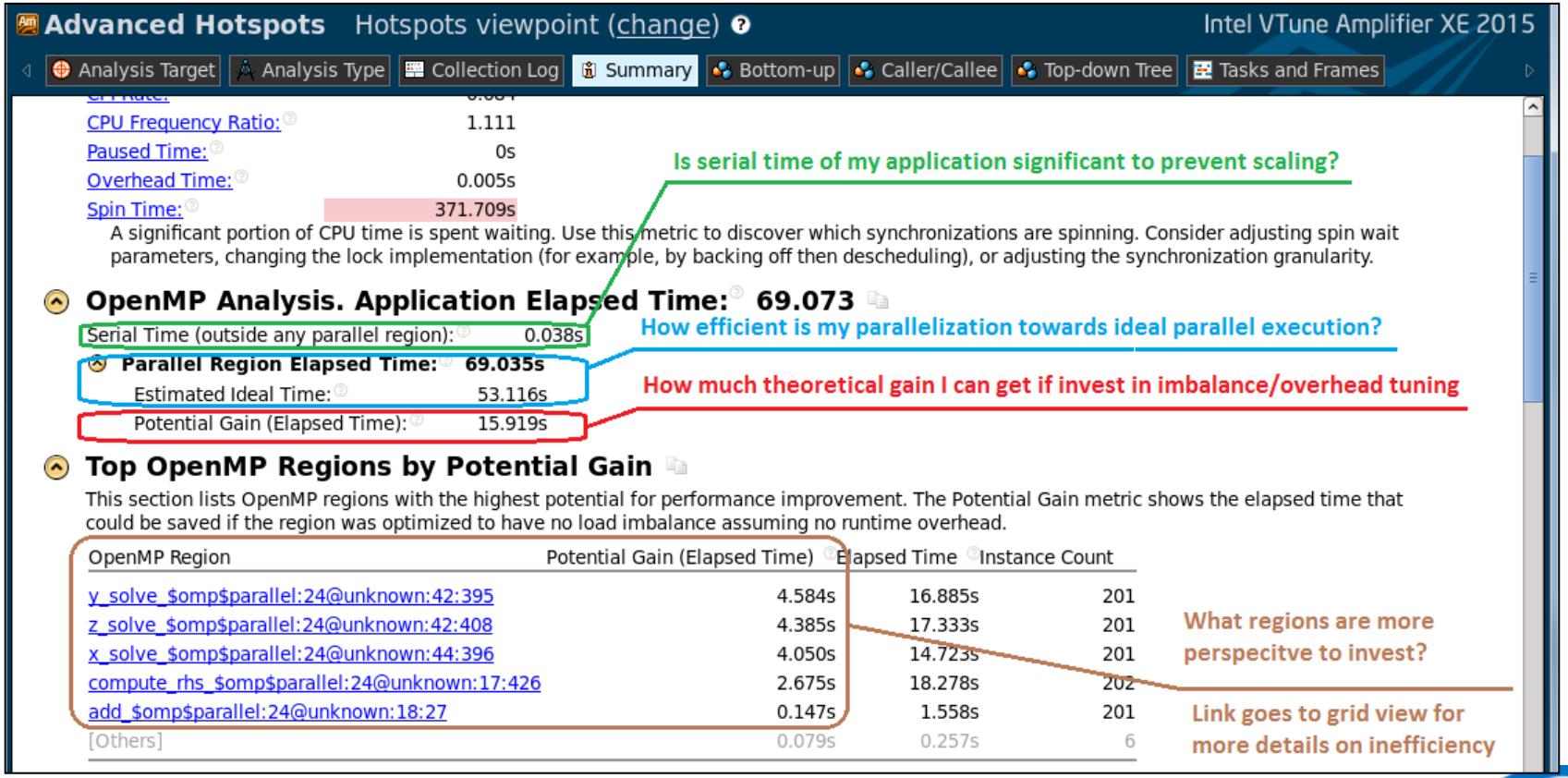
VTune Amplifier XE OpenMP Analysis

Enhancing OpenMP analysis with a set of metrics to answer the following questions:

- **Is serial time of my application significant to prevent scaling?**
- **How efficient is my OpenMP parallelization?**
- **How much gain I can take if invest in reducing load imbalance/overhead?**
- **What regions are more perspective to invest?**

Metrics are based on elapsed time ➡ direct improvement possibilities on application wall clock time

VTune Amplifier XE OpenMP Analysis



Definition of metrics

Serial time: time spent by the application outside any OpenMP* region in the master thread during collection: Elapsed time - ∑[Elapsed time of all Parallel regions]

Effective CPU time of a Parallel Region Instance:
([CPU time] – [Spin Time] – [Overhead Time])

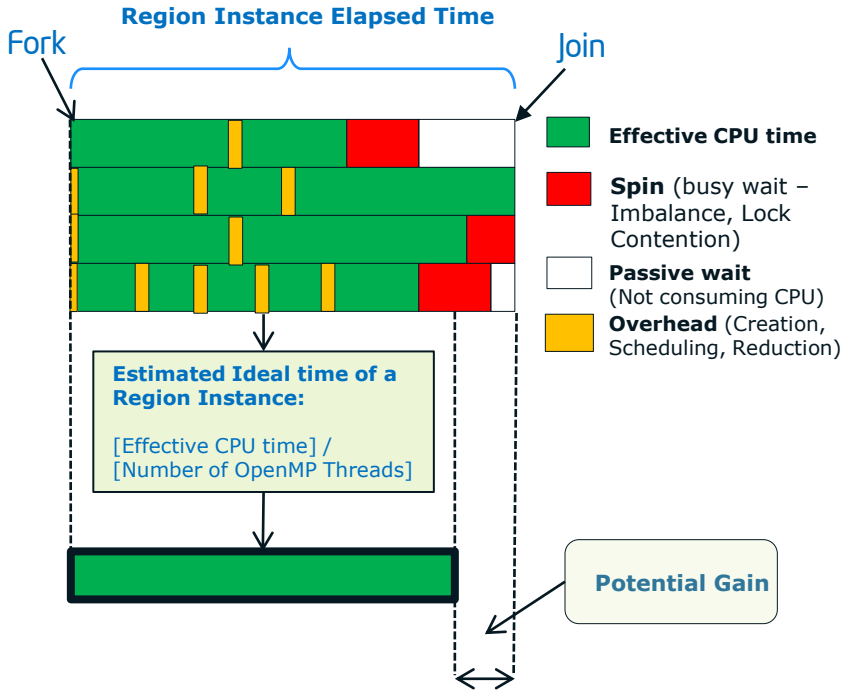
where CPU, Spin and Overhead time aggregated by threads in the Region instance

Estimated Ideal time of a Region Instance:
[Effective CPU time] / [Number of Threads]

Potential Gain of a Parallel Region Instance:
[Region Instance Elapsed Time] – [Estimated Ideal Time of the Region Instance]

Potential Gain of a Region: ∑[Potential Gain of all instances of a Region]

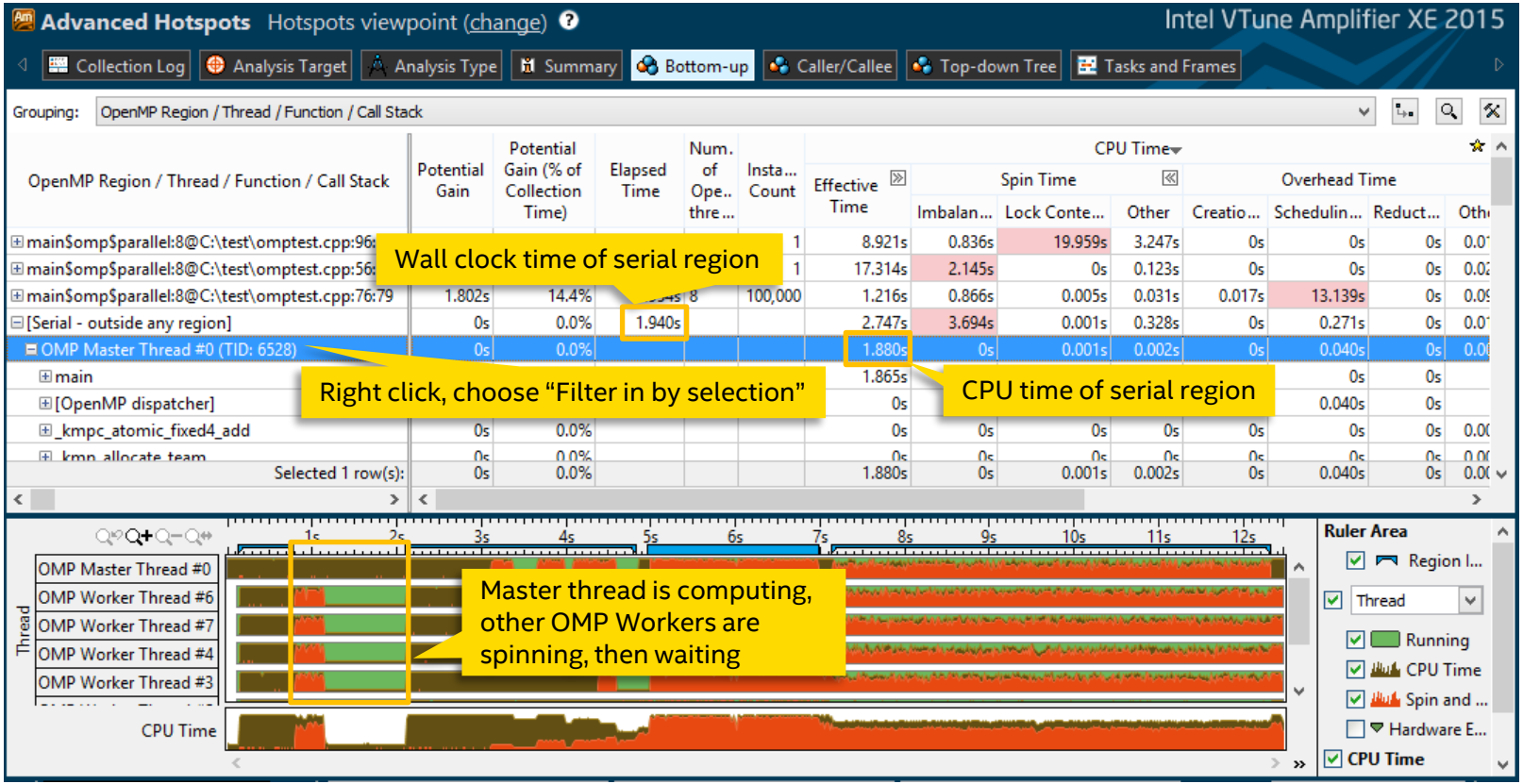
Potential Gain of a Program: ∑[Potential Gain of all Regions]



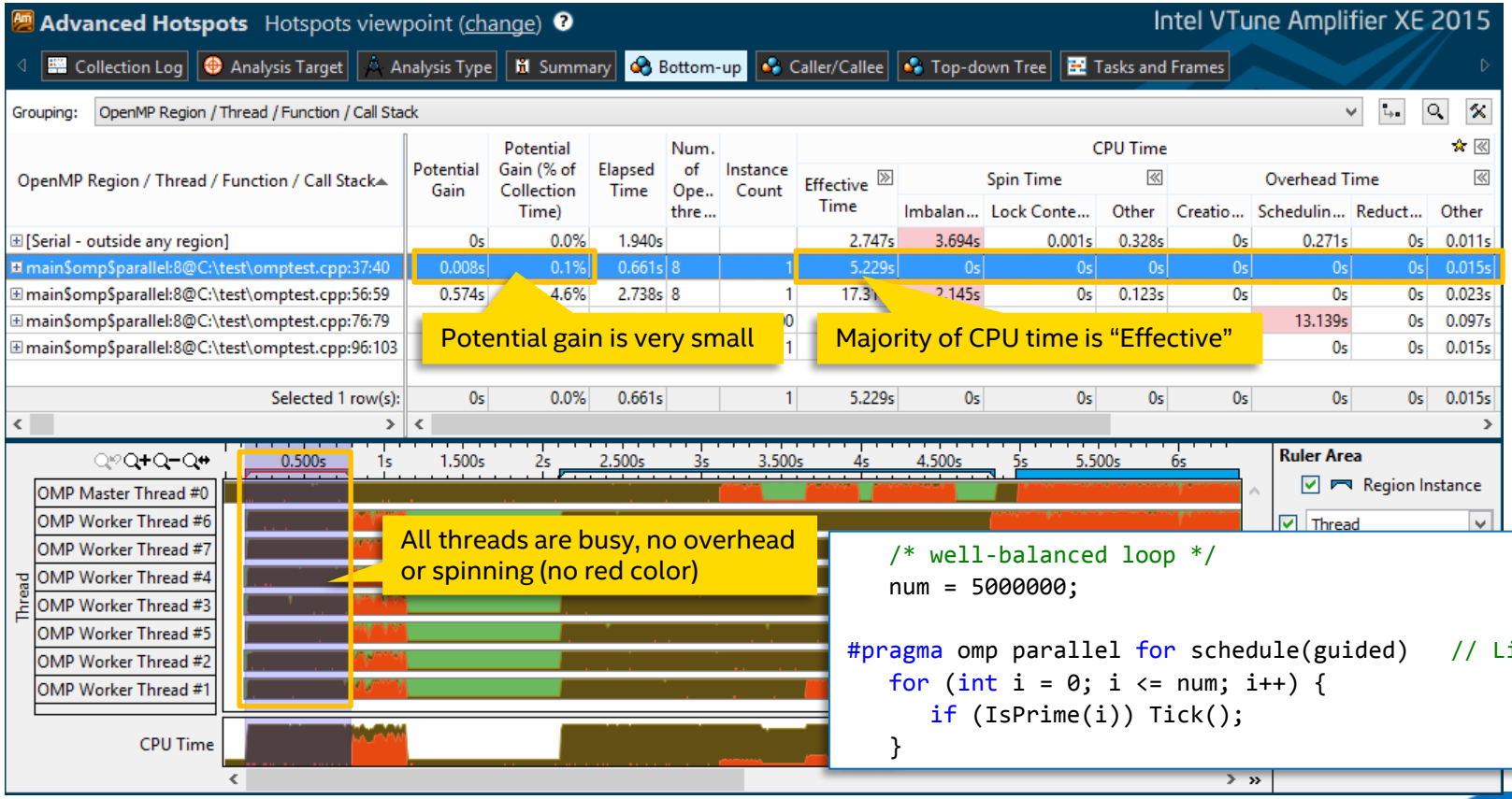
OpenMP region patterns in VTune Amplifier

- Serial region
- Well-balanced region
- Imbalanced region
- Region with runtime overhead
- Region with synchronization objects

Serial region



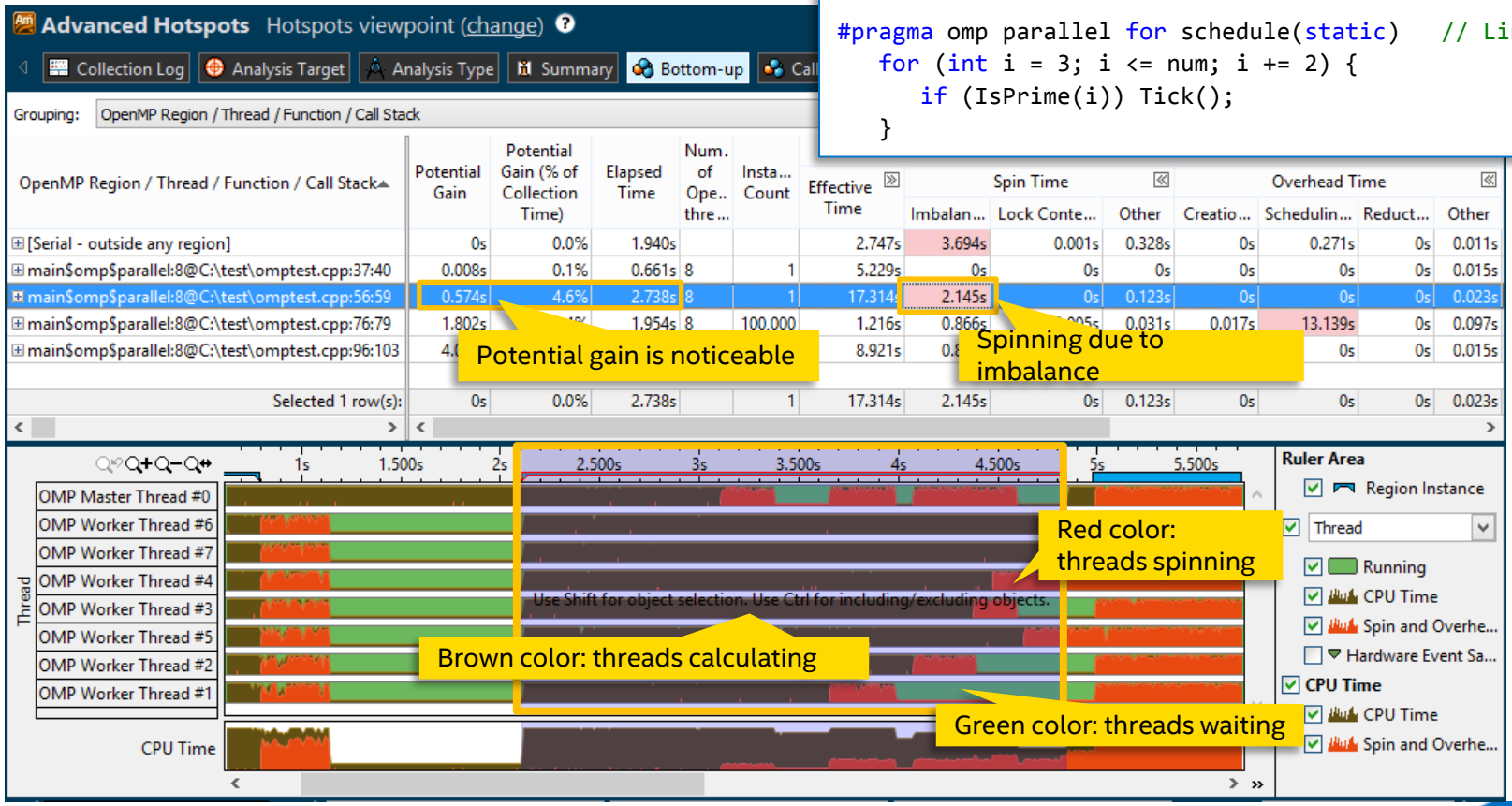
Well-balanced parallel region




```
   /* well-balanced loop */
   num = 5000000;

#pragma omp parallel for schedule(guided)   // Line 37
   for (int i = 0; i <= num; i++) {
      if (IsPrime(i)) Tick();
   }
```

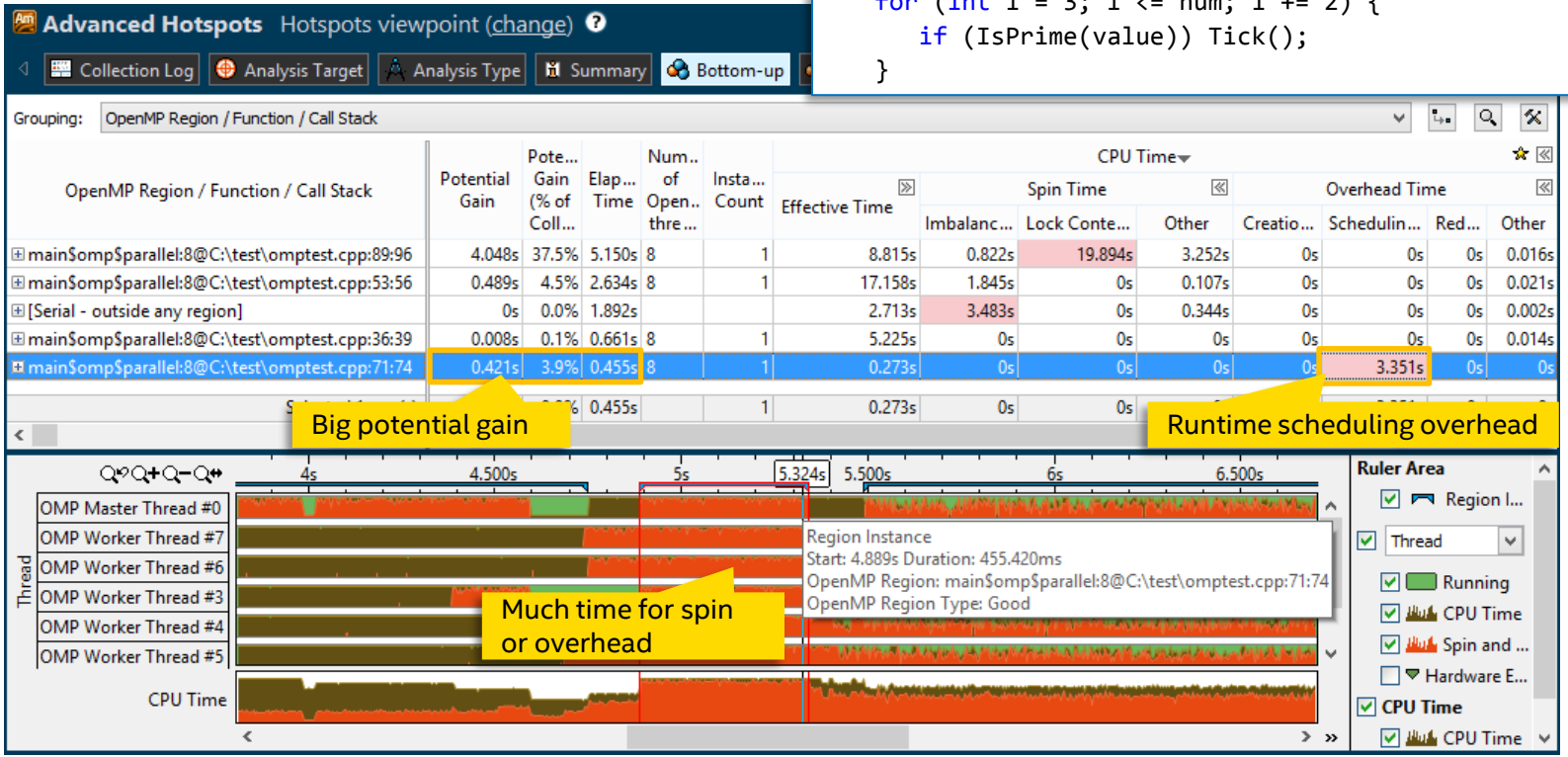
Load imbalance

```
   /* imbalanced loop */
   num = 20000000;

#pragma omp parallel for schedule(static)   // Line 56
   for (int i = 3; i <= num; i += 2) {
      if (IsPrime(i)) Tick();
   }
```

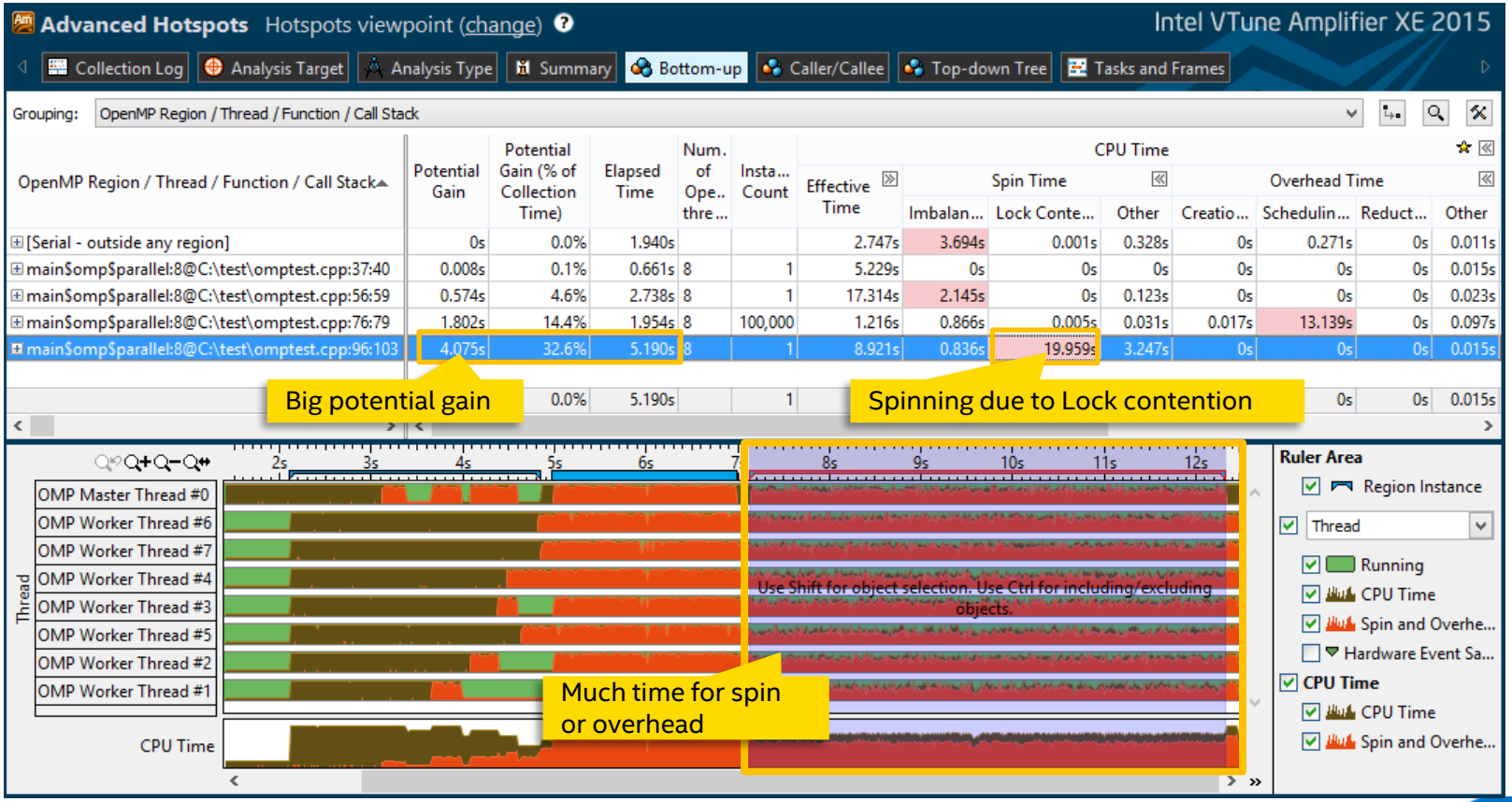


Runtime overhead

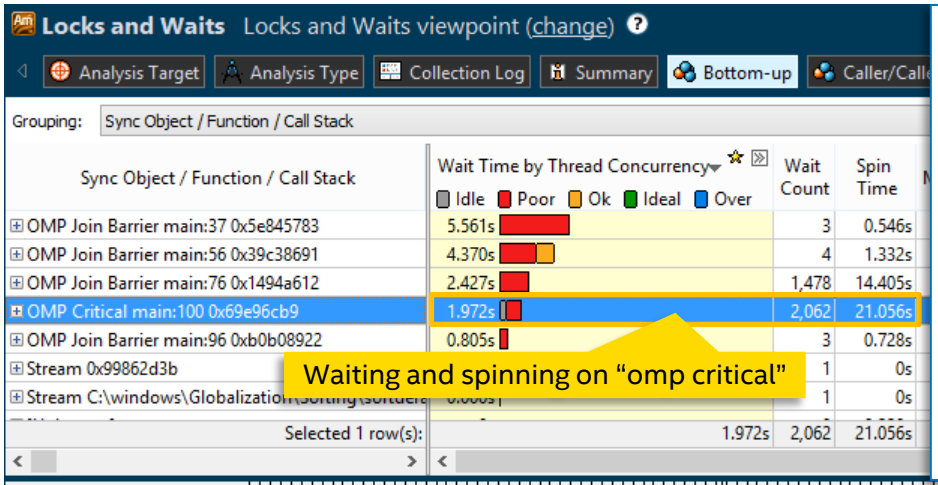
```
   /* overhead loop */
   num = 50000000;
   value=2000;

#pragma omp parallel for schedule (dynamic)   // Line 71
   for (int i = 3; i <= num; i += 2) {
      if (IsPrime(value)) Tick();
   }
```



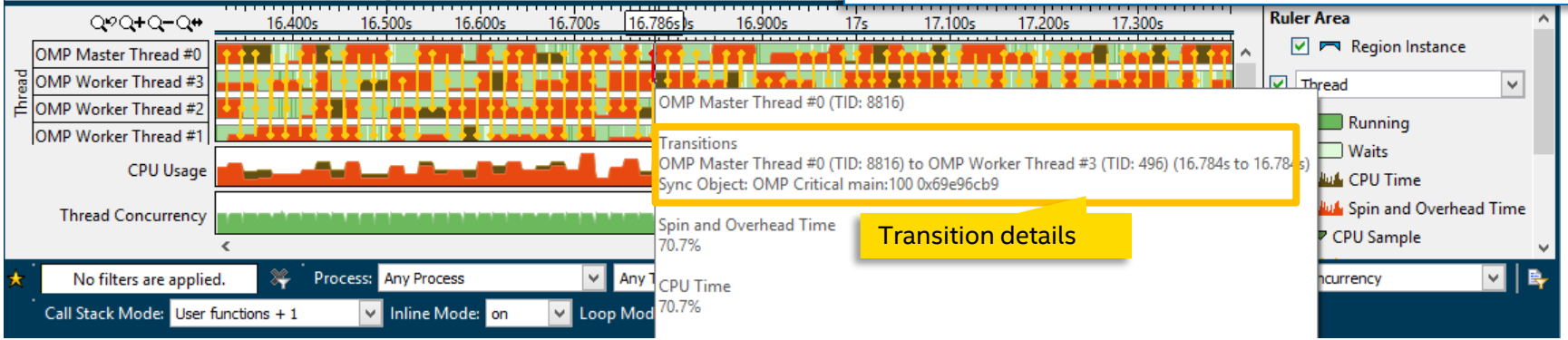
Synchronization objects



Synchronization objects – Locks & Waits analysis




```
      /* loop with lock */
   num = 1000000;

#pragma omp parallel for schedule(guided)   // Line 96
   for (int i = 3; i <= num; i += 2) {

      if (IsPrime(i)) {

         Tick();
#pragma omp critical                        // Line 100
         printf("prime: %d\n", i);
      }
   }
```



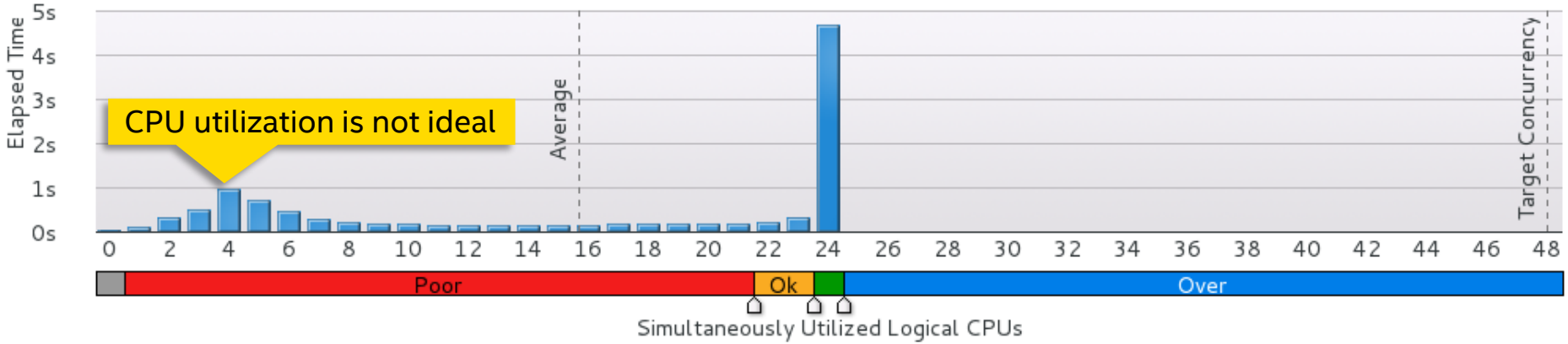
NASA Parallel Benchmark optimization

Setup

- CPU: Intel® Xeon® processor E5-2697 v2 @ 2.70GHz, 24 cores/48 threads.
- OS: RHEL 7.0 x64
- Compiler: Intel® Parallel Studio XE Composer Edition 2015 update 2
- Workload: NPB 3.3.1, “CG - Conjugate Gradient, irregular memory access and communication” module, class B.

CPU Usage 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 usage value.



OpenMP Analysis. Collection Time: 11.400

Serial Time (outside any parallel region): 0.017s (0.1%)

Parallel Region Time: 11.384s (99.9%)

Estimated Ideal Time: 7.408s (65.0%)

Potential Gain: 3.975s (34.9%)

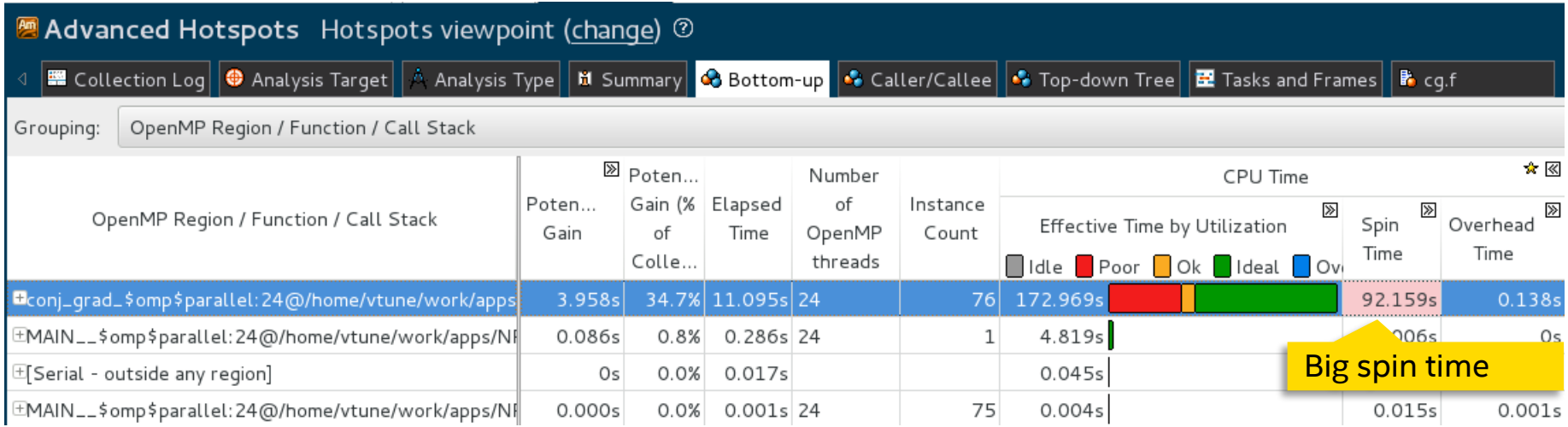
Potential gain is 34.9%

The time wasted on load imbalance or parallel work arrangement is significant and negatively impacts the application performance and scalability. Explore OpenMP regions with the highest metric values. Make sure the workload of the regions is enough and the loop schedule is..

Top OpenMP Regions by Potential Gain

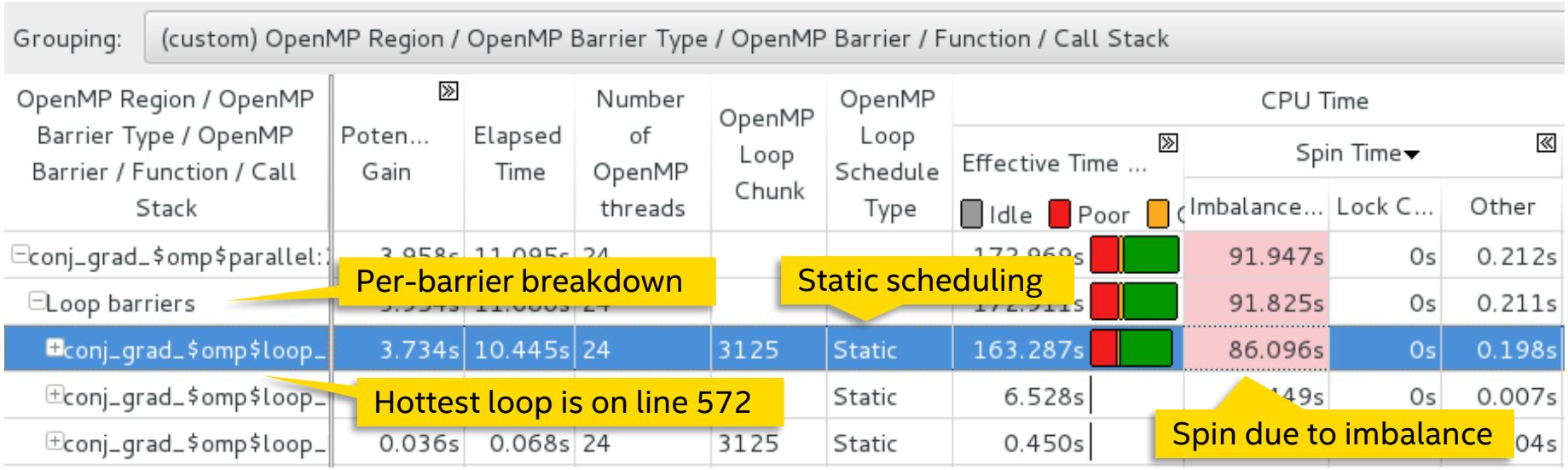
This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	Potential Gain	(%)	Elapsed Time
conj_grad_ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695	3.958s	34.7%	11.095s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:185:231	0.086s	0.8%	0.286s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:361:365	0.000s	0.0%	0.001s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:339:345	0.000s	0.0%	0.001s
MAIN__ompparallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:263:269	0.000s	0.0%	0.000s
[Others]	0.000s	0.0%	0.000s



```
514 !$omp parallel default(shared) private(j,k,cgit,suml,alpha,beta)
515 !$omp&  shared(d,rho0,rho,sum)
516
517 c---------------------------------------------------------------------
518 c  Initialize the CG algorithm:
519 c---------------------------------------------------------------------
520 !$omp do
521       do j=1,naa+1
522          q(j) = 0.0d0
523          z(j) = 0.0d0
524          r(j) = x(j)
525          p(j) = r(j)
526       enddo
527 !$omp end do
528
529
530 c---------------------------------------------------------------------
531 c  rho = r.r
532 c  Now, obtain the norm of r: First, sum squares of r elements locally...
533 c---------------------------------------------------------------------
534 !$omp do reduction(+:rho)
535       do j=1, lastcol-firstcol+1
536          rho = rho + r(j)*r(j)
537       enddo
538 !$omp end do
539
```

Many “omp do” in the same parallel region




```
572 !$omp do
573           do j=1,lastrow-firstrow+1
574              suml = 0.d0
575              do k=rowstr(j),rowstr(j+1)-1
576                 suml = suml + a(k)*p(colidx(k))
577              enddo
578              q(j) = suml
579           enddo
580 !$omp end do
```

Original – static scheduling by default

```
572 !$omp do
573           do j=1,lastrow-firstrow+1
574              suml = 0.d0
575              do k=rowstr(j),rowstr(j+1)-1
576                 suml = suml + a(k)*p(colidx(k))
577              enddo
578              q(j) = suml
579           enddo
580 !$omp end do
```

Changed to dynamic scheduling

```
572 !$omp do schedule (dynamic)
573           do j=1,lastrow-firstrow+1
574              suml = 0.d0
575              do k=rowstr(j),rowstr(j+1)-1
576                 suml = suml + a(k)*p(colidx(k))
577              enddo
578              q(j) = suml
579           enddo
580 !$omp end do
```

Grouping: (custom) OpenMP Region / OpenMP Barrier Type / OpenMP Barrier / Function / Call Stack

OpenMP Region / OpenMP Barrier Type / OpenMP Barrier / Function / Call Stack	Poten... Gain	Elapsed Time	OpenMP Loop Chunk	OpenMP Loop Schedule Type	CPU Time: Effective Time... (Idle, Poor)	CPU Time: Spin Time	CPU Time: Overhead Time: Creation ...	CPU Time: Overhead Time: Scheduling...	CPU Time: Overhead Time: Reduction...	CPU Time: Overhead Time: Other
conj_grad_ompparallel:24@					199.298s	5.866s	0.001s	75.051s	0.012s	0.083s
Loop barriers					199.272s	5.709s	0.001s	75.021s	0.012s	0.071s
conj_grad_omploop_bar	3.133s	11.102s	1	Dynamic	189.320s	0.368s	0s	74.990s	0s	0.009s
conj_grad_omploop_bar	0.128s	0.412s		Static	6.880s	0.969s	0s			
conj_grad_omploop_bar	0.031s	0.0			0.512		0s			

Elapsed time increased

Chunk size is only 1

Spin is fixed now

New problem: scheduling

Default chunk size is 1

```
572 !$omp do schedule (dynamic)
573           do j=1,lastrow-firstrow+1
574              suml = 0.d0
575              do k=rowstr(j),rowstr(j+1)-1
576                 suml = suml + a(k)*p(colidx(k))
577              enddo
578              q(j) = suml
579           enddo
580 !$omp end do
```

Set chunk size to 20

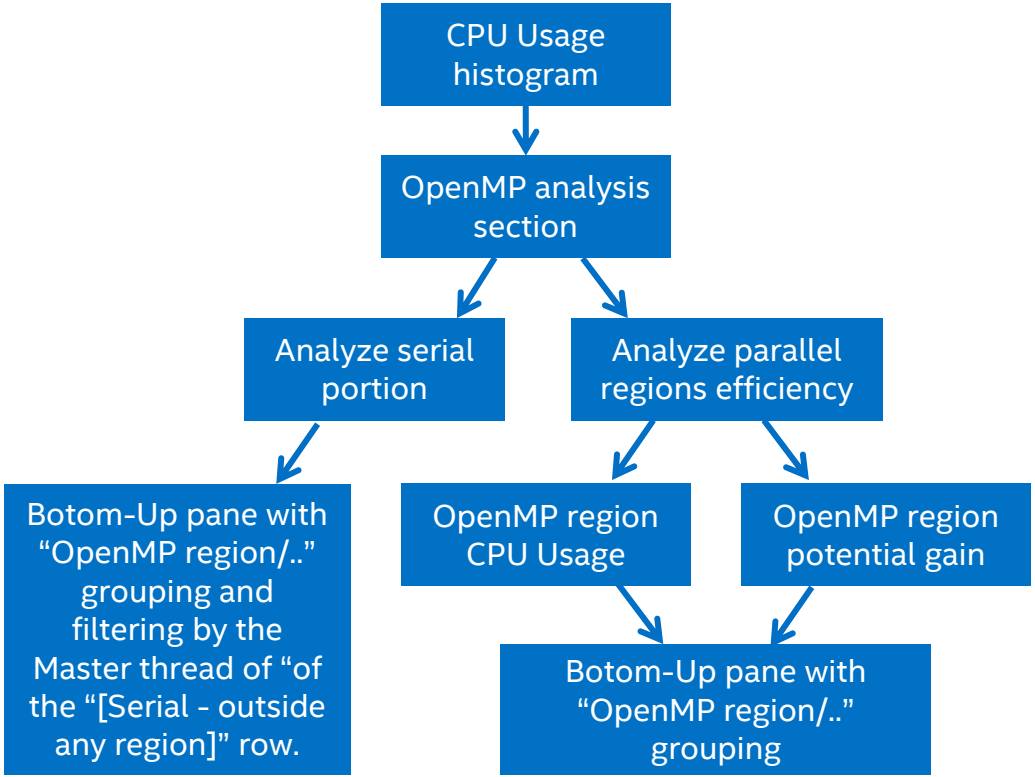
```
572 !$omp do schedule (dynamic,20)
573           do j=1,lastrow-firstrow+1
574              suml = 0.d0
575              do k=rowstr(j),rowstr(j+1)-1
576                 suml = suml + a(k)*p(colidx(k))
577              enddo
578              q(j) = suml
579           enddo
580 !$omp end do
```

Grouping: (custom) OpenMP Region / OpenMP Barrier Type / OpenMP Barrier / Function / Call Stack

OpenMP Region / OpenMP Barrier Type / OpenMP Barrier / Function / Call Stack	Potenti... Gain	Potential Gain (% of Collection Time)	Elapsed Time	Number of OpenMP threads	OpenMP Loop Chunk	OpenMP Loop Schedu... Type	CPU Time: Effective Time by ... (Idle / Poor / Ok)	CPU Time: Spin Time	CPU Time: Over... Time
conj_grad_ompparallel:24@/hc	0.264s	2.7%	9.568s	24			220.930s	6.135s	1.061s
Loop barriers	0.258s	2.6%	9.557s	24			220.904s	5.982s	1.019s
conj_grad_omploop_barrier(	0.119s	1.2%	0.406s	24	3125	Static	6.963s	2.769s	0.001s
conj_grad_omploop_barrier(	0.077s	0.8%	8.928s	24	20	Dynamic	210.848s	1.083s	0.893s
conj_grad_omploop_barrier(	0.028s	0.3%		24	3125	Static	0.538s	1.010s	0.083s
conj_grad_omploop_barri					3125	Static	1.365s	0.537s	0.022s

Elapsed time 8.928s vs 10.445s
Decreased ~17%

General OpenMP analysis workflow




intel®

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804