

Advisor Hand-On: Stencil Example

Samuel Williams

Computational Research Division
Lawrence Berkeley National Lab

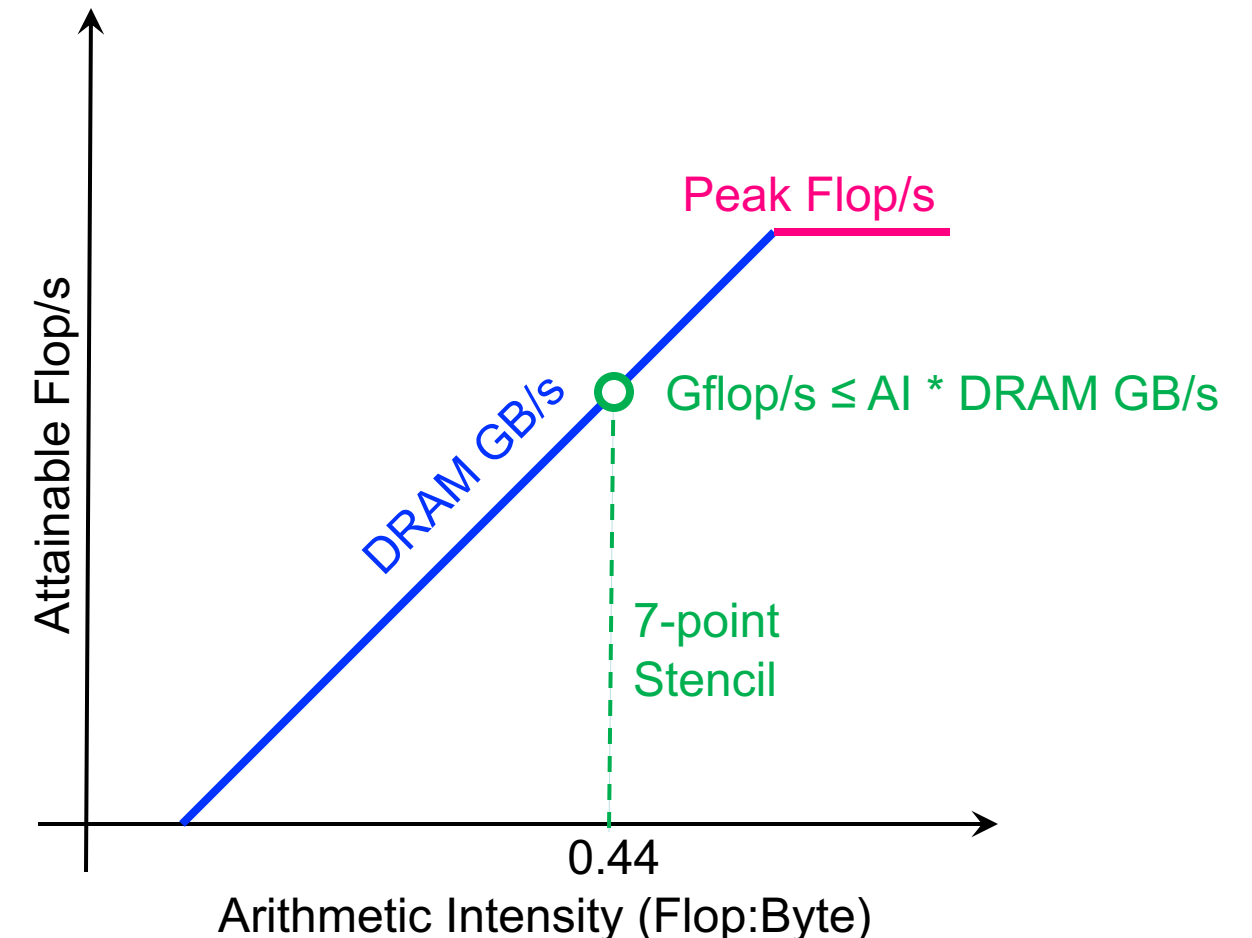
SWWilliams@lbl.gov

Roofline Model:

Arithmetic Intensity and Bandwidth

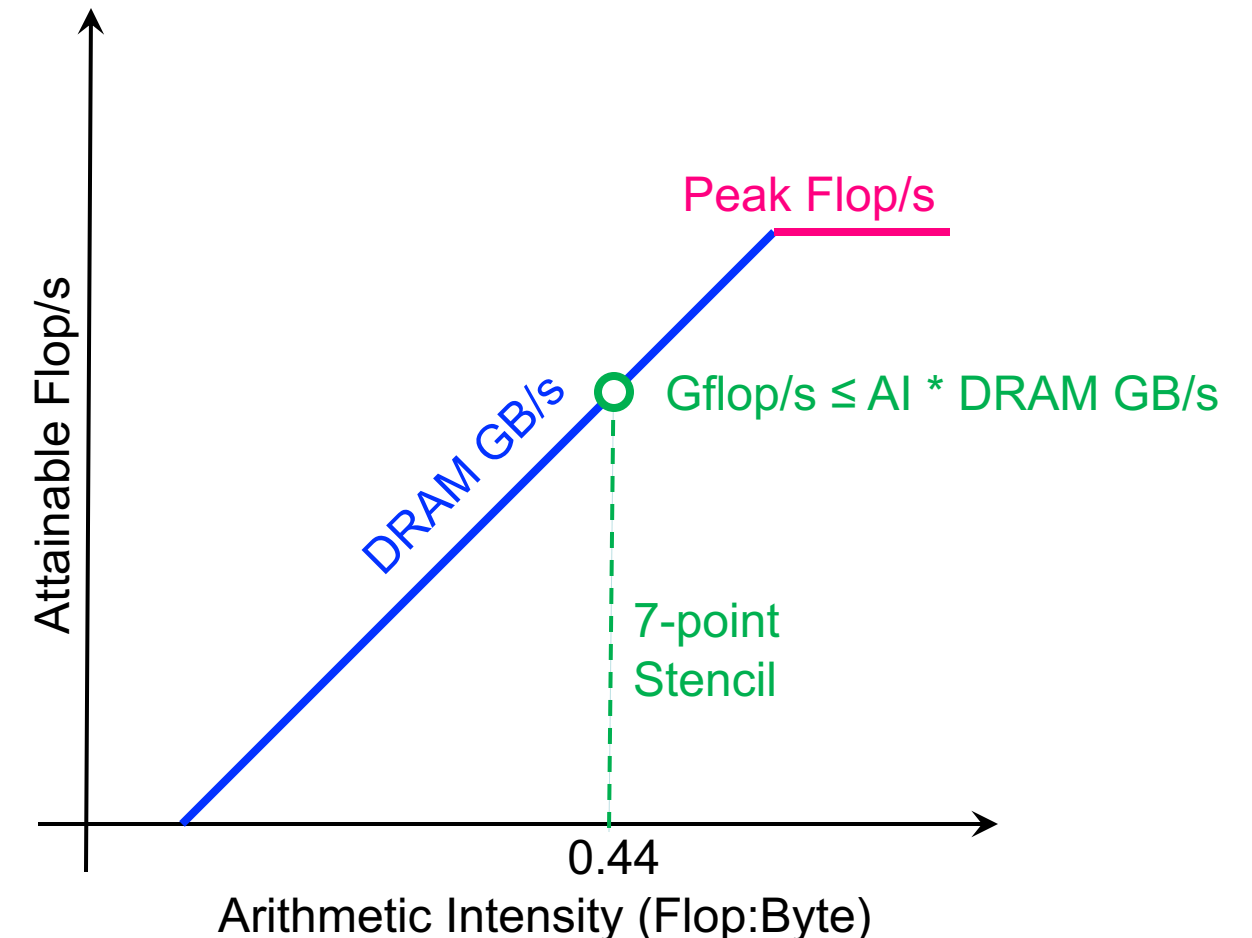
Stencil Example

- Consider our 7-point constant coefficient stencil...
 - 7 flops per point
 - 8 memory references (7 reads, 1 store)
 - An ideal cache can filter all but 1 read and 1 write per point (compulsory misses)
 - **AI = 0.44 flops per byte**
- Actual performance can suffer from...
 - Capacity misses in the L2/L3
 - Failure to vectorize / optimally vectorize
 - Superfluous write allocations



Stencil Example in Advisor

- Let's walk thru a series of progressively more optimized stencil implementations
- We'll use Advisor's Integrated Roofline functionality¹ to highlight how optimization changes AI at different levels of the cache



¹Technology Preview, not in official product roadmap so far.
This version will be made available during the hands-on component of this tutorial.

Naive Version (ver0)

- Naive version (ver0)...
 - 7-point stencil
 - 512^3 grid (padded to 514^3)
 - OpenMP on outer loop
 - **pragma to prevent vectorization**
 - **8MB/thread cache working set**
 - **Low DRAM/L3 Arithmetic Intensity**

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
    #pragma novector
    for(i=1;i<dim+1;i++){
        int ijk = i*iStride + j*jStride + k*kStride;
        new[ijk] = -6.0*old[ijk
                        + old[ijk-iStride]
                        + old[ijk+iStride]
                        + old[ijk-jStride]
                        + old[ijk+jStride]
                        + old[ijk-kStride]
                        + old[ijk+kStride];
    }}}
}}
```

Baseline Version (ver1)

- Eliminate novector
- Simplify Address calculation
- Still has issues...
 - 8MB/thread cache working set
 - 8MB*16 threads > L3
 - Low DRAM/L3 Arithmetic Intensity

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
    int ijk = i + j*jStride + k*kStride;
    new[ijk] = -6.0*old[ijk
                        + old[ijk-iStride]
                        + old[ijk+iStride]
                        + old[ijk-jStride]
                        + old[ijk+jStride]
                        + old[ijk-kStride]
                        + old[ijk+kStride];
}}}

```

Tiled Version (ver2)

- Apply 2D (8x8x512) loop tiling...
 - Cache working set reduced to ~164KB
 - Should improve L3 and DRAM AI (but not L2)

```
#pragma omp parallel for schedule(static,1)
for(tile=0;tile<jTiles*kTiles;tile++){
    int kLo = TILE*(tile/jTiles);
    int jLo = TILE*(tile%jTiles);
    for(k=kLo;k<kLo+TILE;k++){
        for(j=jLo;j<jLo+TILE;j++){
            for(i=0;i<dim;i++){
                int ijk = i + j*jStride + k*kStride;
                new[ijk] = -6.0*old[ijk
                                + old[ijk-1
                                + old[ijk+1
                                + old[ijk-jStride]
                                + old[ijk+jStride]
                                + old[ijk-kStride]
                                + old[ijk+kStride];
            }
        }
    }
}
```

Padded Version (ver3)

- Same code, but
 - Unit-stride padded to 520 (multiple of 8)
 - First non-ghost zone aligned to 64B
 - Should facilitate vectorization / eliminate peel and remainder loops
- Larger dimension = larger array...
 - **Als should be a 1% lower**

```
#pragma omp parallel for schedule(static,1)
for(tile=0;tile<jTiles*kTiles;tile++){
    int kLo = TILE*(tile/jTiles);
    int jLo = TILE*(tile%jTiles);
    for(k=kLo;k<kLo+TILE;k++){
        for(j=jLo;j<jLo+TILE;j++){
            for(i=0;i<dim;i++){
                int ijk = i + j*jStride + k*kStride;
                new[ijk] = -6.0*old[ijk
                                + old[ijk-1
                                + old[ijk+1
                                + old[ijk-jStride]
                                + old[ijk+jStride]
                                + old[ijk-kStride]
                                + old[ijk+kStride];
            }
        }
    }
}
```


SIMD/Cache Bypass Version (ver4)

- Once data has been aligned, force SIMDization and alignment
- Write allocate cache is inflating total data movement by 50%...
 - Use non-temporal store to bypass cache
- **DRAM AI should be a 50% higher and close to the ideal 0.44 flop/byte**

```
#pragma omp parallel for schedule(static,1)
for(tile=0;tile<jTiles*kTiles;tile++){
    int kLo = TILE*(tile/jTiles);
    int jLo = TILE*(tile%jTiles);
    for(k=kLo;k<kLo+TILE;k++){
        for(j=jLo;j<jLo+TILE;j++){
            #pragma omp simd aligned(new,old:64)
            #pragma vector nontemporal
            for(i=0;i<jStride;i++){
                int ijk = i + j*jStride + k*kStride;
                new[ijk] = -6.0*old[ijk
                                + old[ijk-1
                                + old[ijk+1
                                + old[ijk-jStride]
                                + old[ijk+jStride]
                                + old[ijk-kStride]
                                + old[ijk+kStride];
            }
        }
    }
}
```

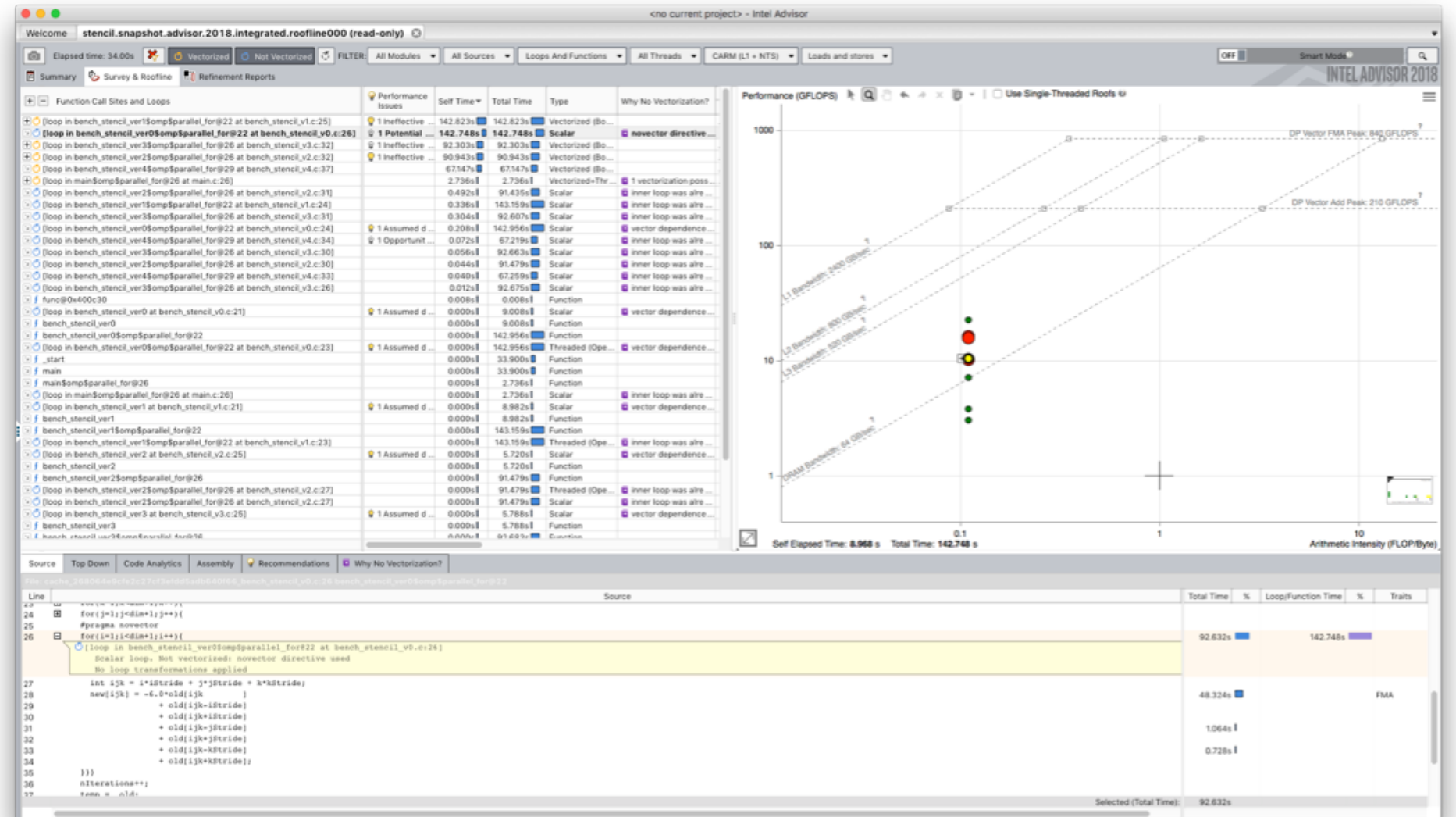
Using Advisor

Open <https://nxcloud01.nersc.gov> in Browser

- Login using your temporary account.
- Create a shell on cori using your temporary account
- Load Advisor
 - `module load advisor/2018.integrated_roofline`**
- We've pre-collected/surveyed the stencil benchmark for you:
 - `cp $ADVISOR_XE_2018_DIR/ECP-meeting-tutorial/* ~`**
- Launch advisor:
 - `advixe-gui`**

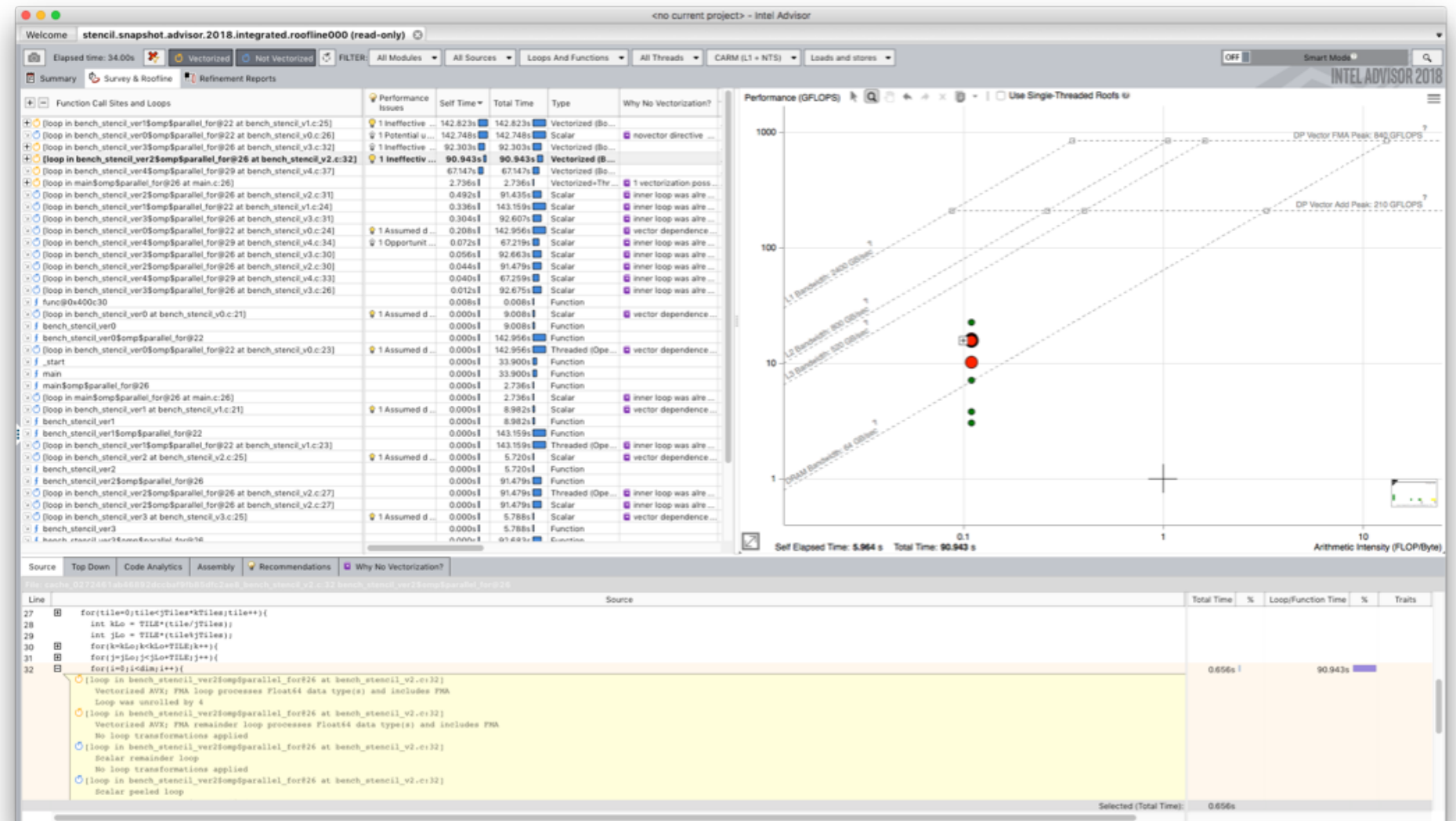
Ver0

- Advisor notes that ver0 didn't vectorize



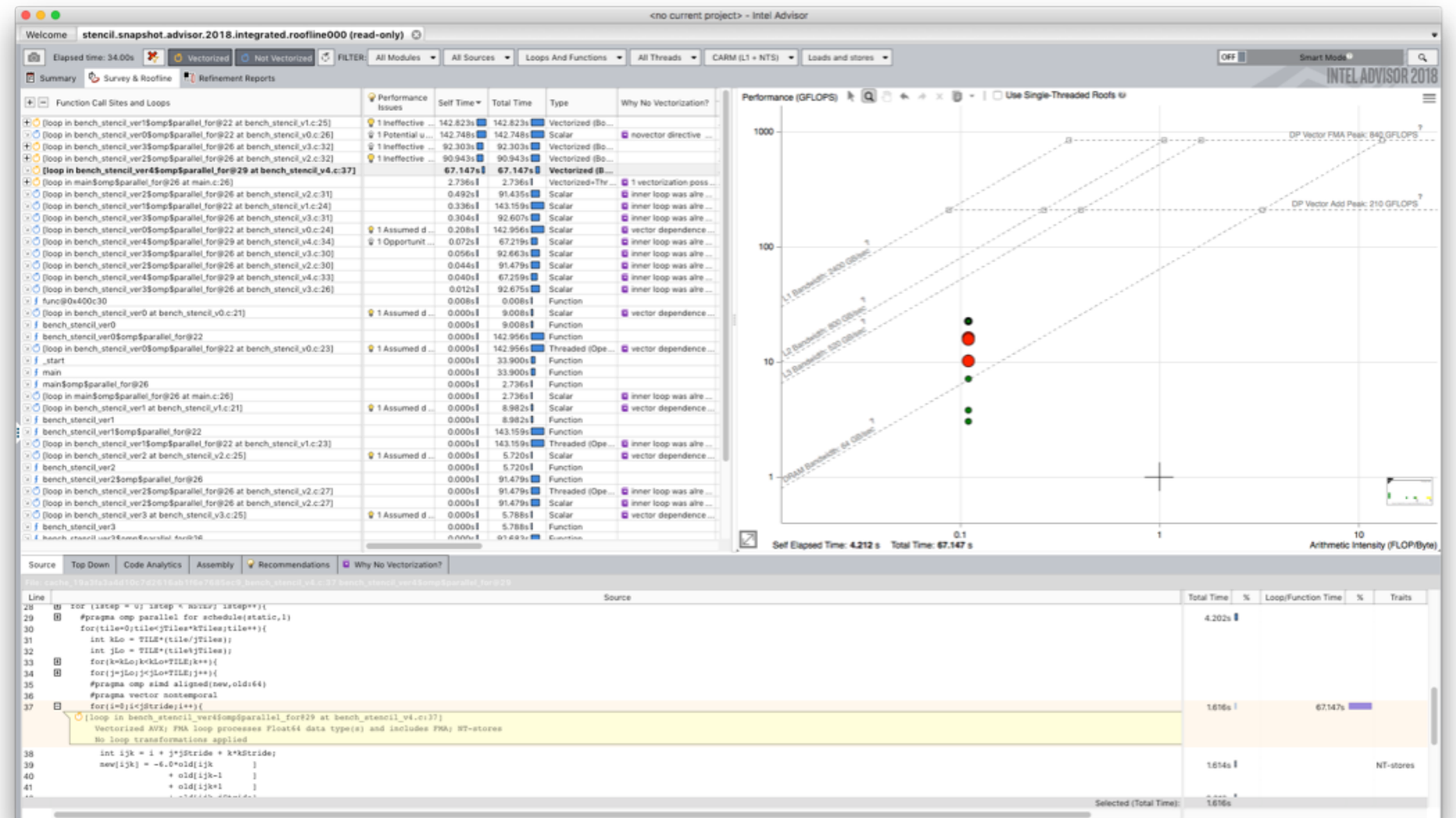
Ver2 (tiled)

- Ver2 showed no improvement in L1 AI.
- All variants present the same number of loads/stores to the L1 cache (compiler failed to register block)



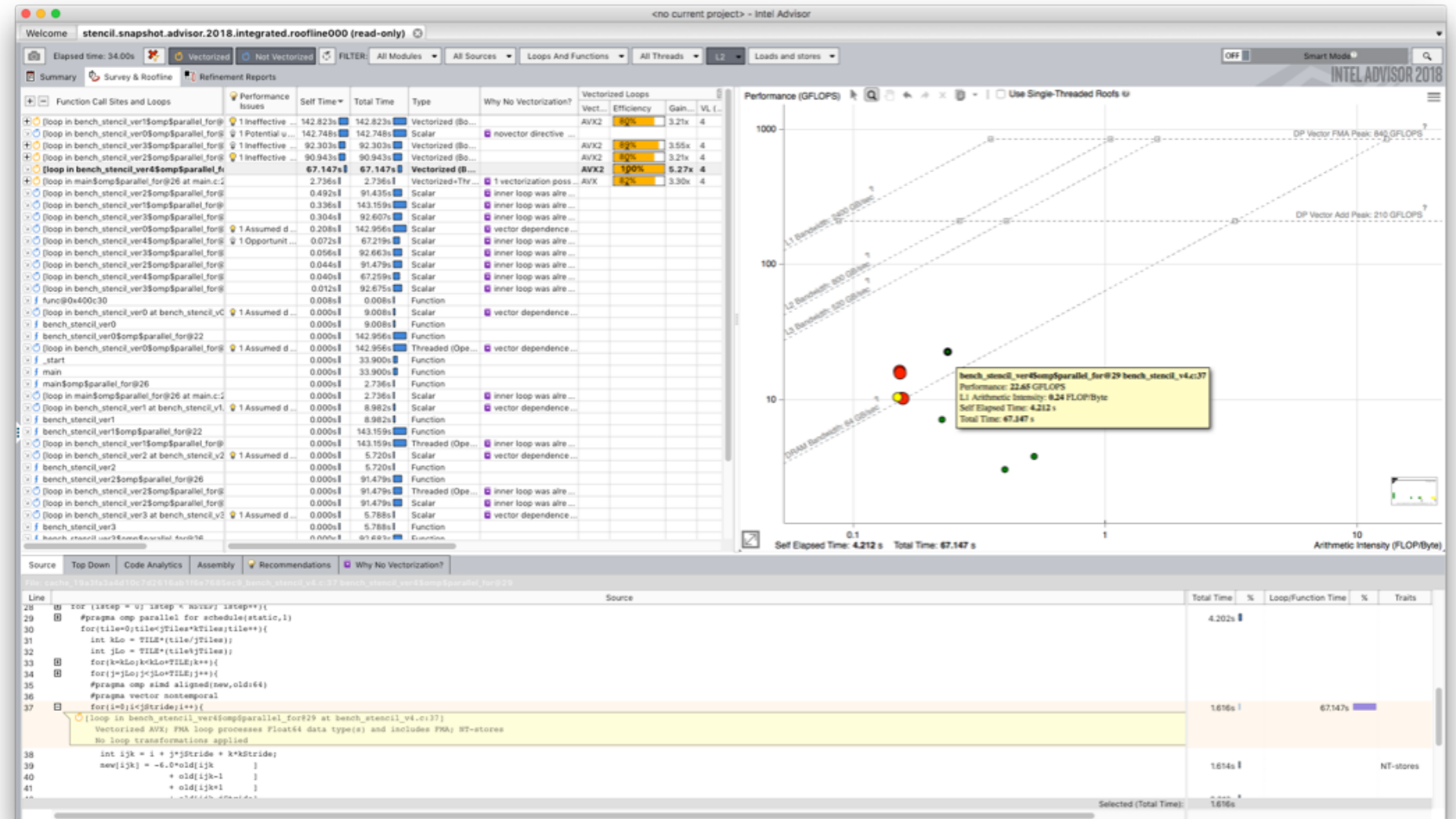
Ver4 (cache bypass)

- Advisor notes the presence of NTS
- L1 AI is flops divided by load+store+NTS
- Hence, NTS does not change L1 AI



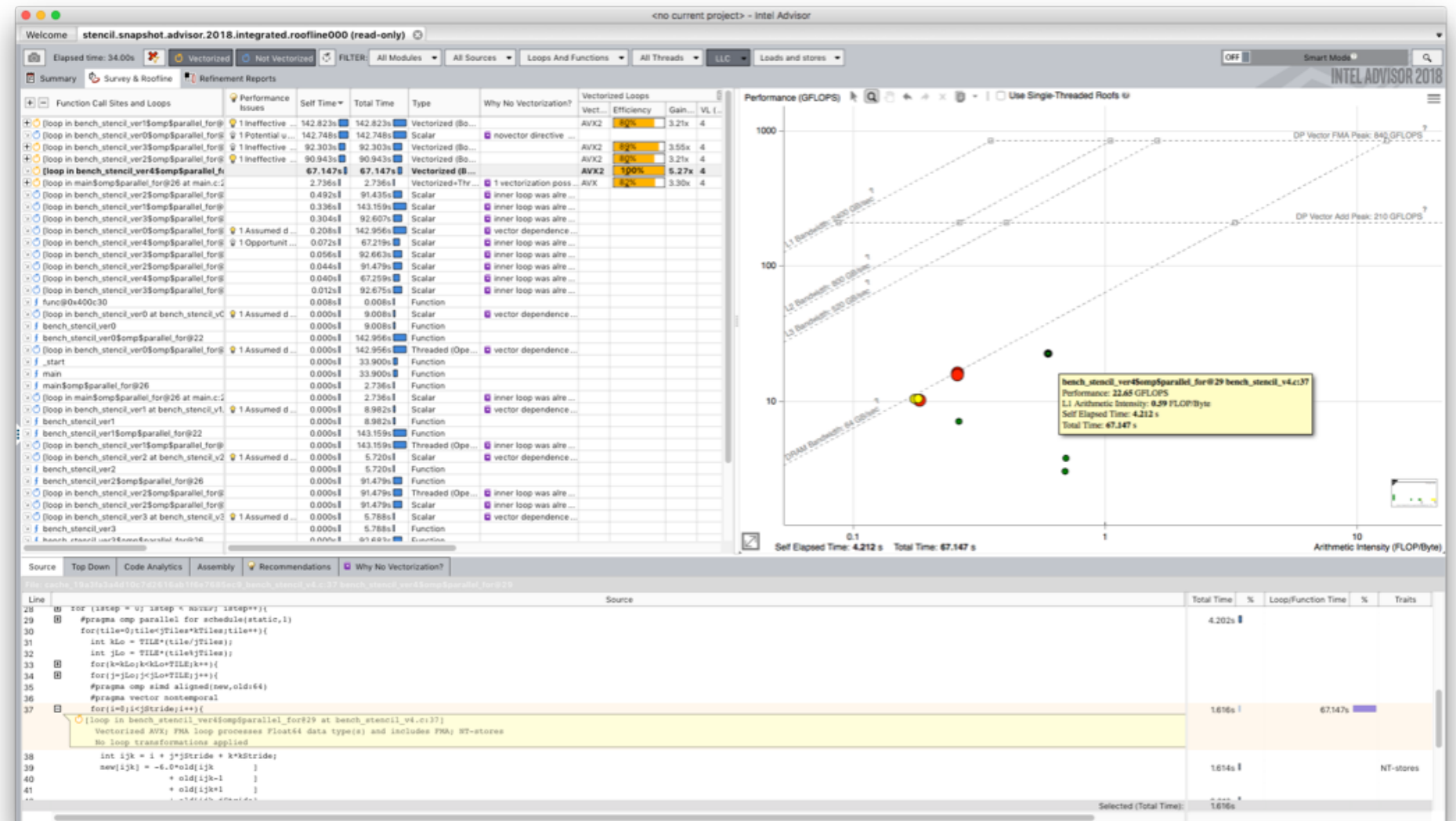
L2 AI

- Observe tiling did not change the L2 AI
- Why? Although working set is smaller, it still doesn't fit in the L1 (164KB > 32KB)
- Ver4 has higher AI.
- Why? NTS bypassed the L2.



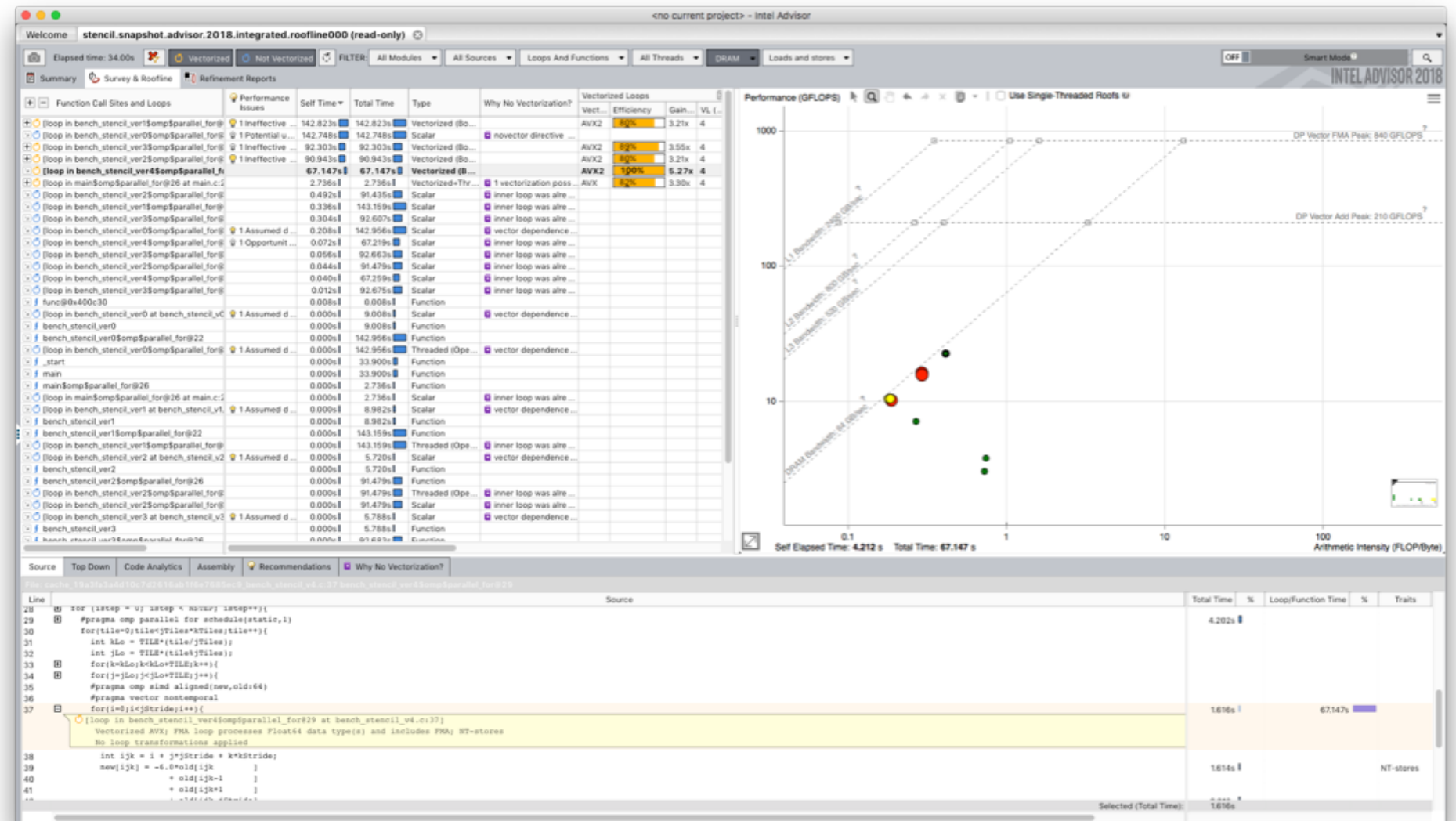
LLC AI

- ver0 and ver1 have low LLC AI
- ver2 and ver3 (tiled) have substantially higher AI (working set fits in LLC)
- Ver4 has even higher LLC AI because NTS bypassed the LLC
- Performance looks correlated with DRAM BW?



DRAM AI

- Ver4 DRAM AI drops, but others remain the same
- Performance is now correlated with STREAM bandwidth
- **Ver4 DRAM AI ~ 0.41 (close to ideal 0.44)**



Questions

Open <https://nxcloud01.nersc.gov> in Browser

- Login using your temporary account.
- Create a shell on cori using your temporary account
- Load Advisor
 - `module load advisor/2018.integrated_roofline`**
- Source the environment variables:
 - `source $ADVISOR_XE_2018_DIR/advixe-vars.sh`**
- We've pre-collected data for you:
 - `cp $ADVISOR_XE_2018_DIR/ECP-meeting-tutorial/* ~`**
- Launch advisor:
 - `advixe-gui`**