Performance Analysis and tools

IIT (Prof. Xian-He Sun's Parallel Computing Class), Oct 10 2019

Sudheer Chunduri
Assistant Computer Scientist
Argonne Leadership Computing Facility

Outline

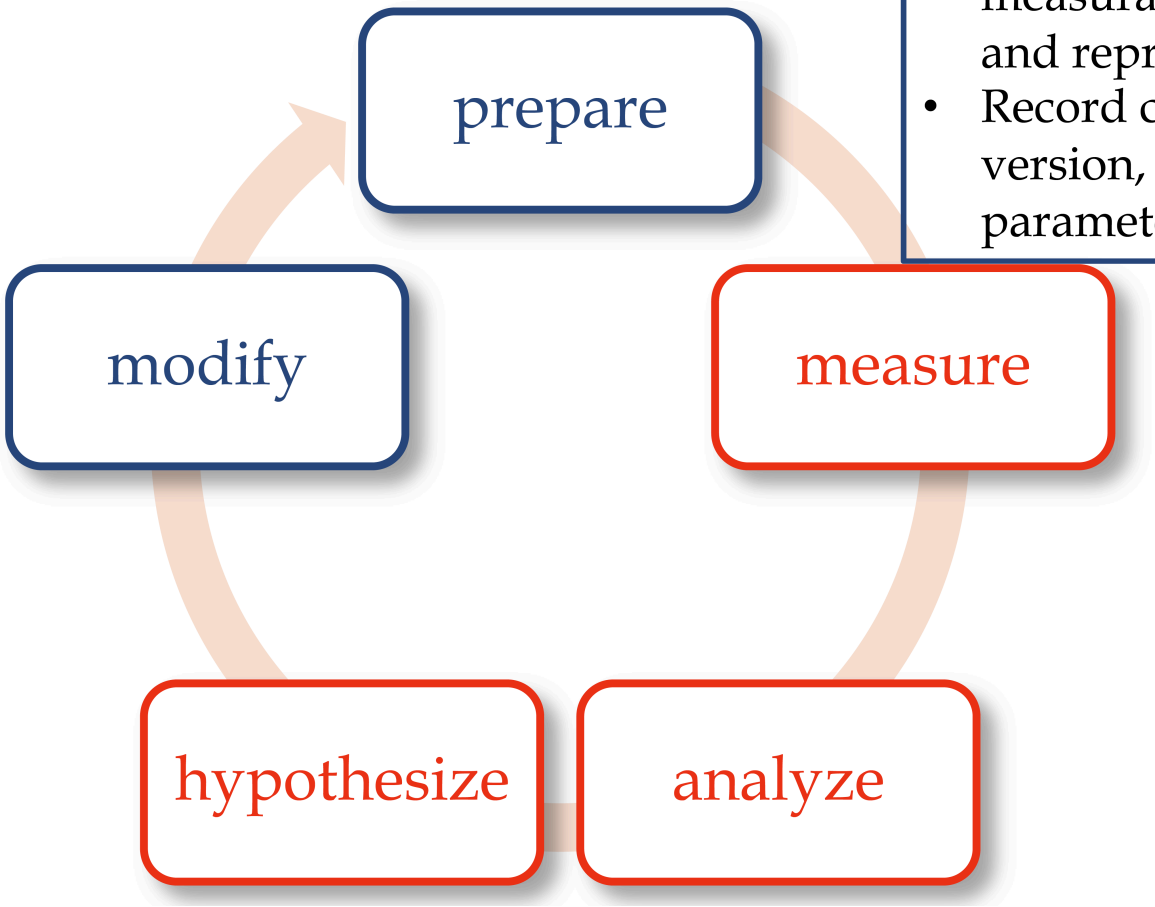
- **Basic Concepts in Performance Tuning**
  - What is performance tuning and why it matters?
  - Performance tuning workflow
  - Typical pitfalls wrt. single node performance
  - Performance tool overview
- **Performance Tools: Examples**
  - How to run basic timing experiments and what they can do
  - How to use hardware counters
  - How to deal with parallelism (vectorization and threads)
- **Goals**
  - Provide basic guidance on how to understand the performance of a code using tools
  - Provide starting point for performance optimizations

What is and why Performance Tuning

- What is performance tuning?
  - The process of improving the efficiency of an application to better utilize a given hardware resource
  - Requires some understanding about the performance features of the given hardware
  - Identifying bottlenecks, determining efficiency and eliminating the bottlenecks if possible
  - Incrementally complete tuning until the performance requirements are satisfied
- Why performance matters?
  - Energy efficiency
  - Today's applications only use a fraction of the machine due to Complex architectures
  - Mapping applications onto architectures is hard

Performance Tuning Workflow

- Change only **one thing at a time**
- Consider the ease (difficulty) of implementation
- Keep **track** of all **changes**
- Apply regression test to **ensure correctness** after each change
- Remember: fast computing of wrong result is completely irrelevant



- Choose an workload which is measurable, representative, static and reproducible, and quantifiable
- Record code generation, compiler version, compiler flags, input parameters, core count, affinity etc

Measure

- What to measure? Choose metrics which quantify the performance of your code
  - Time, energy etc
- How to measure?
  - Linux "time" command
    - Get an idea of overall run time, but can't pin performance bottlenecks
  - Put timer (e.g., gettimeofday, MPI_Wtime, omp_get_wtime) around loops/functions
    - Works for small code base to identify hotspots, but hard to maintain and require significant priori knowledge
  - Performance tools (recommended)
    - Collect a lot data with varying granularity, cost and accuracy
    - Trace back to source code (use –g compiler flag)
    - How to collect

Sampling
- Records system state at periodic intervals
- Useful to get an overview
- Low and uniform overhead
- Ex. Profiling

Instrumentation
- Records all events
- Provide detailed per event information
- High overhead for request events
- Ex. Tracing

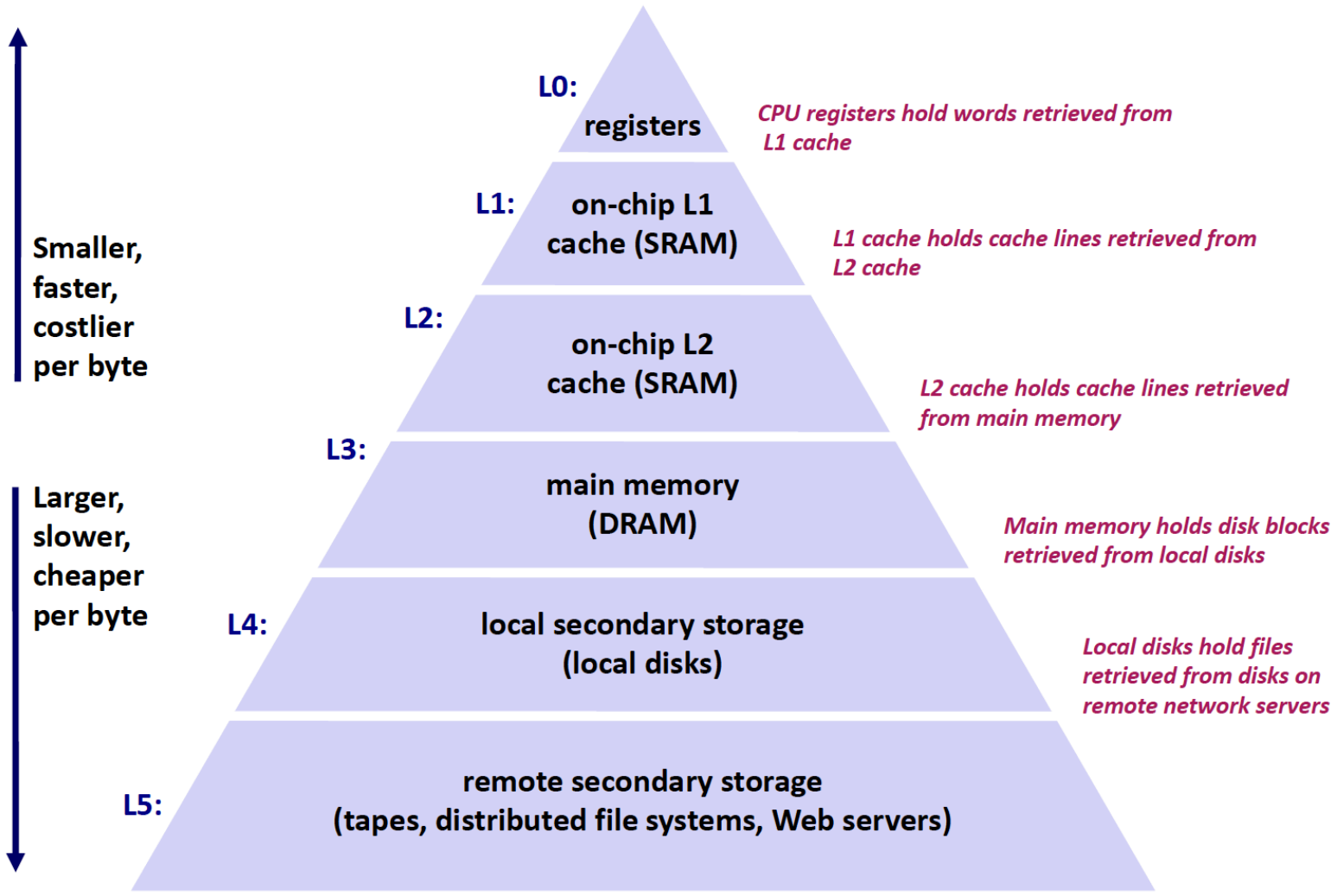
Performance Tools Overview

- Basic OS tools
  - Time
  - Gprof/perf
  - Valgrind/callgrind
- Hardware counter
  - PAPI API & tool set
- Community open source
  - HPCToolkit (Rice Univ.)
  - TAU (U of Oregon)
  - Open|SpeedShop (Krell)
- Commercial products
  - ARM MAP
  - Intel VTune Amplifier
  - Intel Advisor
  - Intel Trace Analyzer
- Vendor supplied (free)
  - CrayPat
  - Nvprof/pgprof

No tool can do everything. Choose the right tool for the right task.

Typical pitfalls with Performance: Sequential

- Where am I spending my time?
  - Find the hotspots
- Is my code computational or memory bounded?
  - Memory bounded
    - Data locality
    - TLB misses
    - L1/L2/L3 $ misses



Typical pitfalls with Performance: Sequential

- Where am I spending my time?
  - Find the hotspots
- Is my code computational or memory bounded?
  - Memory bounded
    - Data locality
    - TLB misses
    - L1/L2/L3 $ misses
  - Computational bounded
    - Vectorization Efficiency
    - Avoid type conversion

Typical pitfalls with Performance: Sequential

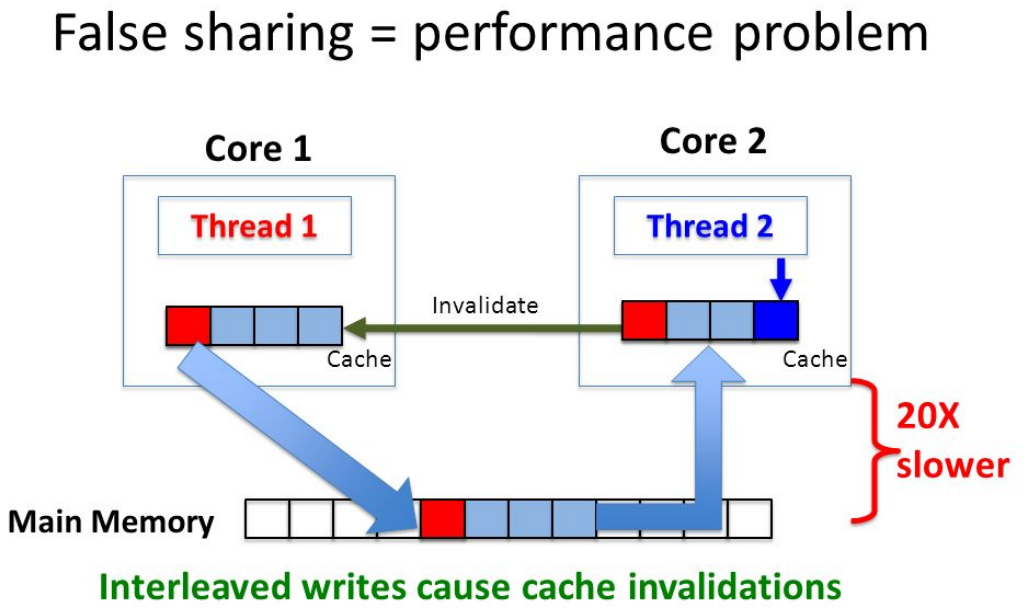
- Where am I spending my time?
  - Find the hotspots
- Is my code computational or memory bounded?
  - Memory bounded
    - Data locality
    - TLB misses
    - L1/L2/L3 $ misses
  - Computational bounded
    - Vectorization Efficiency
    - Avoid type conversion

```
float x=3.14; //bad: 3.14 is a double
float s=sin(x); //bad: sin() is a double precision function
long v=round(x); //bad: round takes and returns double
```

```
float x=3.14f; //good: 3.14f is a float
float s=sinf(x); //good: sinf() is a single precision function
long v=lroundf(x); //good: lroundf () takes and returns double
```

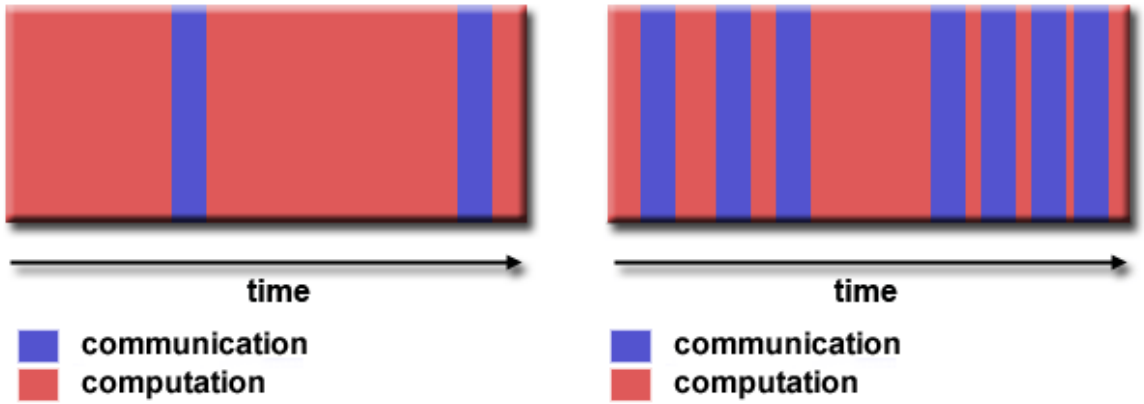
Typical pitfalls with Performance: Multithreading

- Load imbalance
- **False sharing**
  - Occurs when threads on different processors modify variables that reside on the same cache line
  - Caused by coherent caches
  - Cache line is 64 bytes wide
- Insufficient parallelism
- Synchronization
  - Avoid synchronization with private thread storage
- Non-optimal memory placement
  - Thread affinity
  - Allocation on first touch



Typical pitfalls with Performance: Distributed

- Scaling inefficiency
  - Serial bottlenecks
  - Load imbalance
    - Improper work distribution
- Granularity
  - Coarse-grained:
    - communication and synchronization overhead is low
    - Hard to load balance
  - Fine-grained:
    - Better load balance
    - Comm & sync overheads high
- Communication overhead
  - Communication-computation overlap
  - Network contention
  - Non-optimal process placement
    - Topology unaware
- IO overheads



Linux Perf

Linux Perf

- Perf is a performance analyzing tool in Linux, available in version 2.6.31
- How does it work
  - *perf record*: measure and save sampling data for a single program
    - -g: enable call-graph (callers/callee information)
  - *perf report*: analyze the file generated by perf record, can be flat profile or graph
    - -g: enable call-graph (callers/callee information)
  - *perf list*: list available events for measurement
    - Support a list of hardware and software events
  - *perf stat*: measure total event count for a single program
    - -e: event names provided in perf list
  - *etc*
- When compiling the code, use the following flags for easier interpretation
  - *-g*: need debug symbols in order to annotation source
  - *-fno-omit-frame-pointer*: provide stack chain/backtrace

Example: Matrix-Matrix Multiplication

```
int main(int argc, char *argv[])
{
  int matrix_size; //N*N matrix
  int max_iters=10; //number of times to call a matrix-matrix function

  //read command line input
  //set various paramaters
  if(argc<2) {
    cout<<"ERROR: expecting integer matrix size, i.e., N for NxN matrix"<<endl;
    exit(1);
  }
  else {
    matrix_size=atoi(argv[1]);
  }

  cout<<"using matrix size:"<<matrix_size<<endl;

  double **A, **B, **C; //2D arrays

  // create memory space for 2D matrixes
  create_matrix_2D(A, B, C, matrix_size);

  // initialize matrixes
  init_matrix_2D(A, B, C, matrix_size);

  for (int r=0; r < max_iters; r++) {
    zero_result(C,matrix_size);
#ifdef NAIVE
    compute_naive(A,B,C,matrix_size);
#elif INTERCHANGE
    compute_interchange(A,B,C,matrix_size);
#elif TRIANGULAR
    compute_triangular(A,B,C,matrix_size);
#endif
  }

  // free memeory space
  free_matrix_2D(A, B, C, matrix_size);

  return 0;
}
```

Two versions of 2D matrix-matrix multiplication

```
#ifdef NAIVE
//NAIVE: 2D matrix-matrix multiplication
__attribute__((noinline)) void compute_naive(double **A, double **B, double **C, int
matrix_size) {
  for (int i = 0 ; i < matrix_size; i++) {
    for (int j = 0;  j < matrix_size; j++) {
      for (int k = 0; k < matrix_size; k++) {
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}

#elif INTERCHANGE
//INTERCHANGE: 2D matrix-matrix multiplication
__attribute__((noinline)) void compute_interchange(double **A, double **B, double **C,
int matrix_size) {
  for (int i = 0 ; i < matrix_size; i++) {
    for (int k = 0; k < matrix_size; k++) {
      for (int j = 0;  j < matrix_size; j++) {
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}
```

Loop-interchange Optimization

- The *perf list* command lists all available CPU counters:

```
List of pre-defined events (to be used in -e):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
  branch-loads                                       [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-loads                                         [Hardware cache event]
  dTLB-store-misses                                  [Hardware cache event]
  dTLB-stores                                        [Hardware cache event]
  iTLB-load-misses                                   [Hardware cache event]
  iTLB-loads                                         [Hardware cache event]
  node-load-misses                                   [Hardware cache event]
  node-loads                                         [Hardware cache event]
  node-store-misses                                  [Hardware cache event]
  node-stores                                        [Hardware cache event]
```

- Check *man perf_event_open* to see what does each event measure

- The *perf stat* command instruments and summarizes selected CPU counters
  1. Compile the code
     - *g++ -g -fno-omit-frame-pointer -O3 -DNAIVE matmul_2D.cpp -o mm_naive.out*
  2. Run perf stat
     - *perf stat -e cpu-cycles,instructions,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores ./mm_naive.out 500*
  3. Record the numbers for each events
  4. Compile the code
     - *g++ -g -fno-omit-frame-pointer -O3 -DINTERCHANGE matmul_2D.cpp -o mm_interchange.out*
  5. Run perf stat
     - *perf stat -e cpu-cycles,instructions,L1-dcache-loads,L1-dcache-load-misses ./mm_interchange.out 500*
  6. Compare the numbers for both cases

Results Comparison (GCC on Skylake)

NAIVE

```
Performance counter stats for './mm_naive.out 500':

     5,690,894,860      cpu-cycles
    10,060,418,904      instructions              #    1.77  insn per cycle
     3,766,438,627      L1-dcache-loads
     1,477,256,051      L1-dcache-load-misses     #   39.22% of all L1-dcache hits
     1,258,966,720      L1-dcache-stores

       1.631281598 seconds time elapsed
```

INTERCHANGE

```
Performance counter stats for './mm_interchange.out 500':

     2,579,473,221      cpu-cycles
     8,868,040,843      instructions              #    3.44  insn per cycle
     3,143,159,042      L1-dcache-loads
       164,686,016      L1-dcache-load-misses     #    5.24% of all L1-dcache hits
     1,258,516,903      L1-dcache-stores

       0.740328866 seconds time elapsed
```

- The number of CPU cycles is much lowers for interchange, reflecting its shorter elapsed time
- The number of instructions are half in interchange
- Interchange has substantial fewer LL1 load misses, which indicates better data locality

What will happen if the matrix dimension is changed to 1000?

Results Comparison (GCC on Skylake)

NAIVE

```
Performance counter stats for './mm_naive.out 1000':

    48,170,169,478      cpu-cycles                                                    (55.56%)
    80,257,444,510      instructions              #    1.67  insn per cycle           (66.67%)
    30,074,772,443      L1-dcache-loads                                               (66.67%)
    13,717,515,476      L1-dcache-load-misses     #   45.61% of all L1-dcache hits    (66.67%)
    10,047,395,806      L1-dcache-stores                                              (66.67%)
           75,579       LLC-load-misses                                               (66.67%)
           47,390       LLC-store-misses                                              (44.44%)
     1,187,058,403      dTLB-load-misses                                              (44.44%)
           68,971       dTLB-store-misses                                             (44.45%)


      13.882172014 seconds time elapsed
```

INTERCHANGE

Higher LLC and TLB misses

```
Performance counter stats for './mm_interchange.out 1000':

    19,594,771,570      cpu-cycles                                                    (55.56%)
    70,426,885,060      instructions              #    3.59  insn per cycle           (66.67%)
    25,068,272,468      L1-dcache-loads                                               (66.68%)
     1,300,279,310      L1-dcache-load-misses     #    5.19% of all L1-dcache hits    (66.68%)
    10,039,107,897      L1-dcache-stores                                              (66.69%)
           23,069       LLC-load-misses                                               (66.69%)
           38,559       LLC-store-misses                                              (44.44%)
       11,975,830       dTLB-load-misses                                              (44.43%)
           70,397       dTLB-store-misses                                             (44.42%)


      7.069999994 seconds time elapsed
```

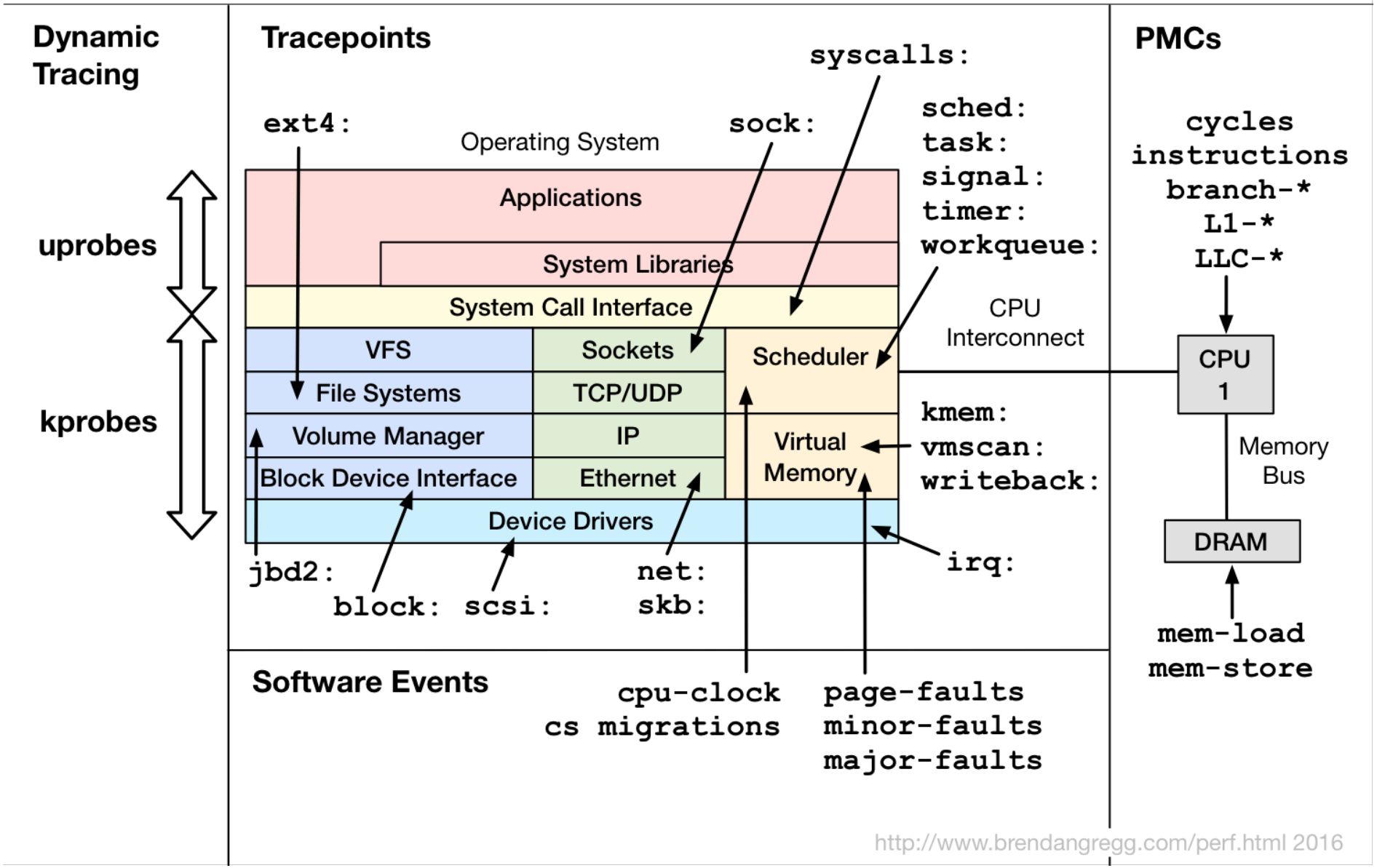
Find hotspot Using Perf

Perf record –g ./mm_interchange 500

```
Samples: 2K of event 'cycles:ppp', Event count (approx.): 2634218725
  Children      Self  Command          Shared Object       Symbol
+   99.42%     0.00%  mm_interchange.  libc-2.17.so        [.] __libc_start_main
+   99.42%     0.00%  mm_interchange.  mm_interchange.out  [.] main
-   98.83%    98.66%  mm_interchange.  mm_interchange.out  [.] compute_interchange
     98.66% __libc_start_main
        main
        compute_interchange
     0.33%     0.00%  mm_interchange.  [unknown]           [k] 0000000000000000
     0.33%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] page_fault
     0.33%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] do_page_fault
     0.33%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] __do_page_fault
     0.30%     0.03%  mm_interchange.  libc-2.17.so        [.] _int_malloc
     0.29%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] handle_mm_fault
     0.29%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] handle_pte_fault
     0.18%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] apic_timer_interrupt
     0.18%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] smp_apic_timer_interrupt
     0.18%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] local_apic_timer_interrupt
     0.18%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] hrtimer_interrupt
     0.18%     0.18%  mm_interchange.  libc-2.17.so        [.] __memset_sse2
     0.18%     0.18%  mm_interchange.  libc-2.17.so        [.] __random
     0.15%     0.15%  mm_interchange.  libc-2.17.so        [.] __random_r
     0.14%     0.04%  mm_interchange.  [kernel.kallsyms]   [k] __hrtimer_run_queues
     0.13%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] alloc_pages_vma
     0.13%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] __alloc_pages_nodemask
     0.11%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] tick_sched_timer
     0.10%     0.07%  mm_interchange.  [kernel.kallsyms]   [k] get_page_from_freelist
     0.10%     0.10%  mm_interchange.  [kernel.kallsyms]   [k] mem_cgroup_charge_common
     0.10%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] mem_cgroup_newpage_charge
     0.09%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] system_call_fastpath
     0.09%     0.09%  mm_interchange.  mm_interchange.out  [.] init_matrix_2D
     0.08%     0.00%  mm_interchange.  ld-2.17.so          [.] _etext
     0.07%     0.00%  mm_interchange.  ld-2.17.so          [.] _dl_map_object
     0.07%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] tick_sched_handle
     0.07%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] update_process_times
     0.05%     0.00%  mm_interchange.  ld-2.17.so          [.] open_path
     0.05%     0.00%  mm_interchange.  ld-2.17.so          [.] open64
     0.05%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] sys_open
     0.05%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] do_sys_open
     0.05%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] do_filp_open
     0.05%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] path_openat
     0.05%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] link_path_walk
     0.04%     0.04%  mm_interchange.  [kernel.kallsyms]   [k] perf_pmu_disable
     0.04%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] scheduler_tick
     0.04%     0.04%  mm_interchange.  [kernel.kallsyms]   [k] update_wall_time
     0.04%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] tick_sched_do_timer
     0.04%     0.00%  mm_interchange.  [kernel.kallsyms]   [k] tick_do_update_jiffies64
     0.04%     0.04%  mm_interchange.  [kernel.kallsyms]   [k] ktime_get_update_offsets_now
```

```
compute_interchange  /home/chunduri/IIT_class/codas_perftools/mm_interchange.out
  0.04 |       seta   %r8b
  0.04 |       test   %r8b,%al
       |     ↓ je     400ed0 <compute_interchange(double**, double**, 140
       |       lea    0x10(%rdi),%rax
       |       cmp    %rax,%rdx
       |       setae  %r8b
       |       cmp    -0x38(%rbp),%rdi
  0.07 |       setae  %al
       |       or     %al,%r8b
       |     ↓ je     400ed0 <compute_interchange(double**, double**, 140
  0.04 |       test   %r12d,%r12d
       |     ↓ je     400ea4 <compute_interchange(double**, double**, 114
       |     __attribute__((noinline)) void compute_interchange(double **A, double **B, double **C, int matrix_size) {
  0.04 |       xor    %eax,%eax
  0.04 |       xor    %r8d,%r8d
  0.14 |       xchg   %ax,%ax
       |             C[i][j] += A[i][k] * B[k][j];
  5.46 |  d0:  movsd  (%r9),%xmm1
  6.07 |       add    $0x1,%r8d
 14.71 |       movsd  (%rdi,%rax,1),%xmm2
  5.85 |       unpckl %xmm1,%xmm1
  8.89 |       movhpd 0x8(%rdi,%rax,1),%xmm2
  5.49 |       movsd  (%rdx,%rax,1),%xmm0
  9.04 |       mulpd  %xmm2,%xmm1
  4.55 |       movhpd 0x8(%rdx,%rax,1),%xmm0
 14.82 |       addpd  %xmm1,%xmm0
  9.54 |       movlpd %xmm0,(%rdx,%rax,1)
  5.24 |       movhpd %xmm0,0x8(%rdx,%rax,1)
  5.02 |       add    $0x10,%rax
       |       cmp    %r8d,%esi
  4.66 |     ↑ ja     400e60 <compute_interchange(double**, double**, d0
       |       cmp    %r12d,%ecx
       |     ↓ je     400eb7 <compute_interchange(double**, double**, 127
       | 114:  movsd  (%r9),%xmm0
       |       mulsd  (%rdi,%r15,1),%xmm0
       |       addsd  (%rbx),%xmm0
       |       movsd  %xmm0,(%rbx)
  0.04 | 127:  add    $0x1,%r10
       |         for (int k = 0; k < matrix_size; k++) {
       |       cmp    %r10d,%ecx
       |     ↓ jle    400f00 <compute_interchange(double**, double**, 170
  0.04 | 130:  mov    %r11,%r9
  0.04 |     ↑ jmpq   400e10 <compute_interchange(double**, double**, 80
       |       nop
       |     __attribute__((noinline)) void compute_interchange(double **A, double **B, double **C, int matrix_size) {
       | 140:  xor    %eax,%eax
       |       nop
       |             C[i][j] += A[i][k] * B[k][j];
       | 148:  movsd  (%r9),%xmm0
       |       mulsd  (%rdi,%rax,8),%xmm0
```

Linux perf_events Event Sources



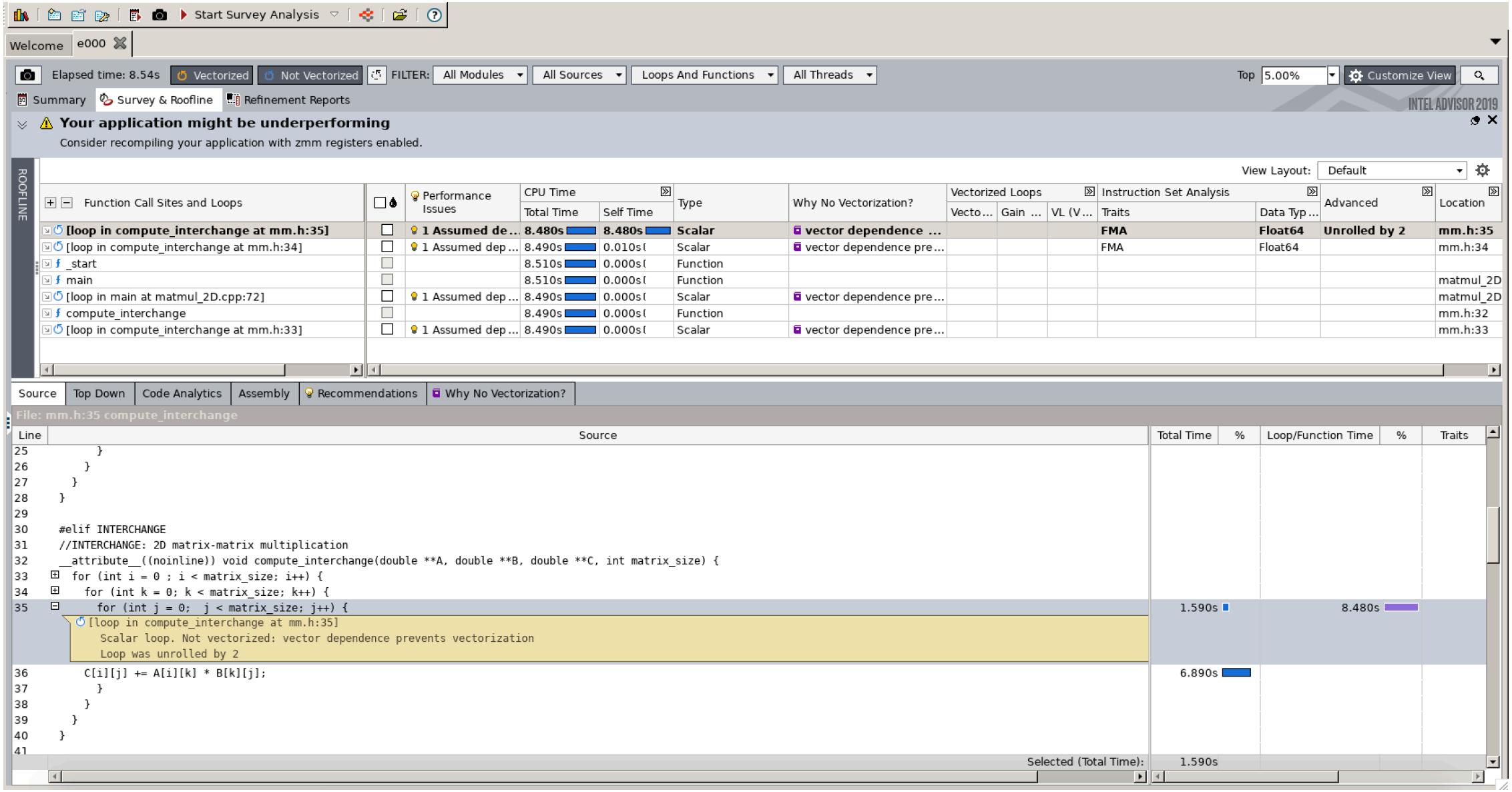
Supports

- Event counting
- Profiling
- Tracing
  - Static
  - Dynamic
- Reporting

Intel Advisor

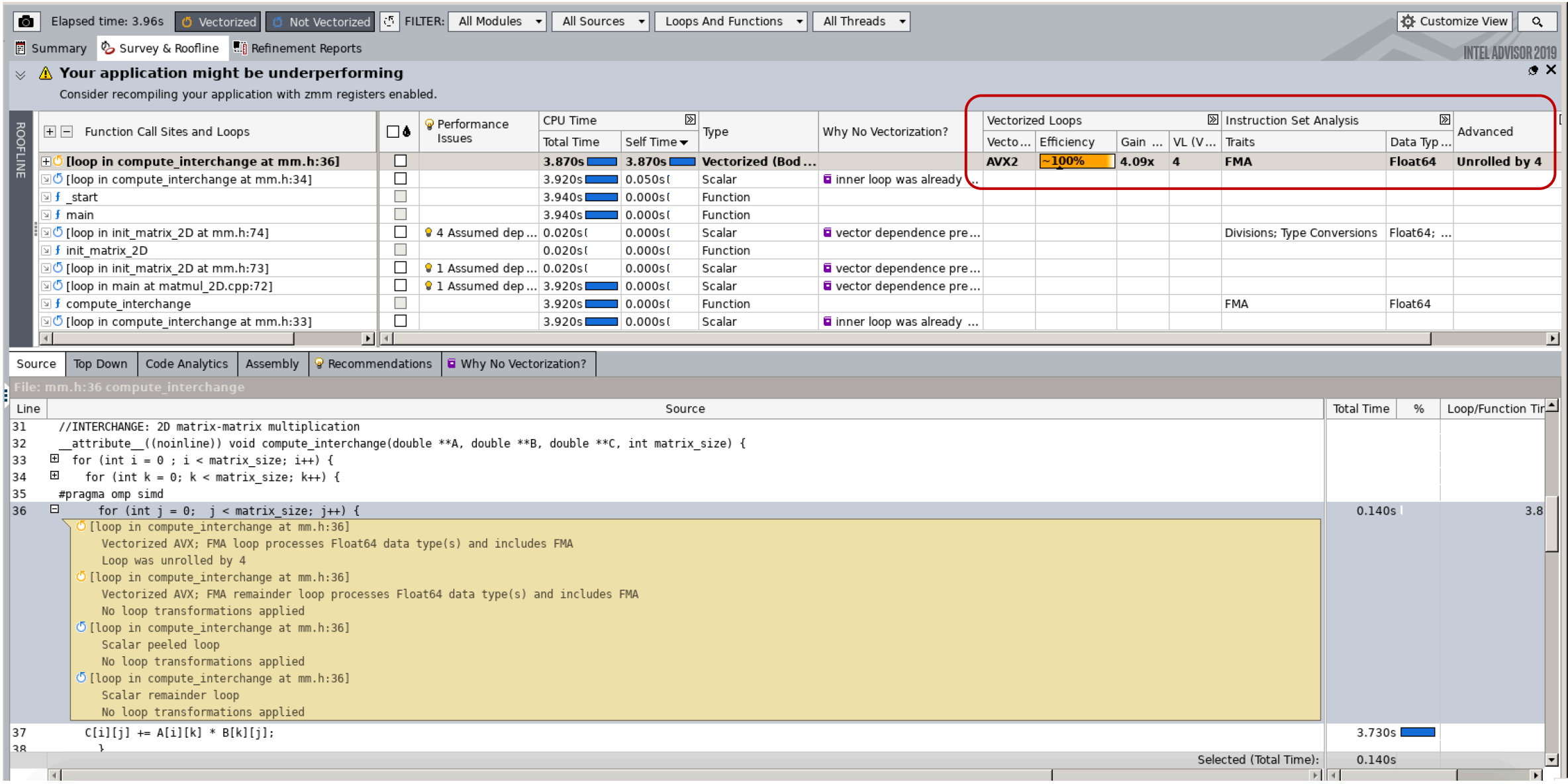
Survey Analysis

Compile the code: icpc -g -O3 –xhost -DINTERCHANGE matmul_2D.cpp -o mm_interchange_icpc.out
Collect the survey data: advixe-cl -c survey -project-dir mm-advisor -- ./mm_interchange_icpc.out 1000
Open the result in GUI: advixe-gui mm-advisor



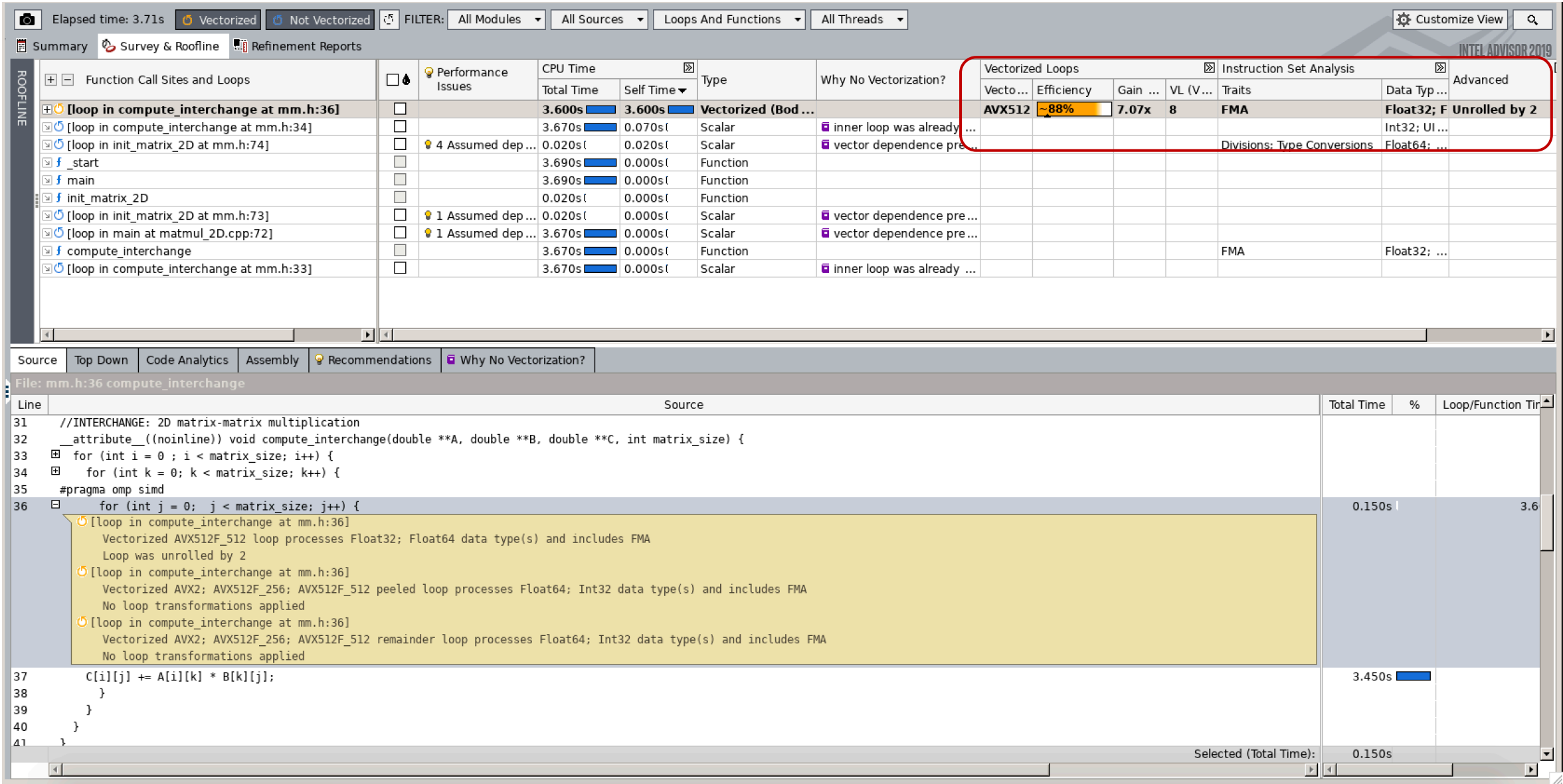
Survey with SIMD

Compile: icpc –g –O3 –xhost -qopenmp-simd –DINTERCHANGE matmul_2D.cpp –o mm_interchange_icpc.out
Collect the survey data: advixe-cl –c survey -project-dir mm-advisor -- ./mm_interchange_icpc.out 1000
Open the result in GUI: advixe-gui mm-advisor



512-bit ZMM register

Compile: icpc –g –O3 –xhost -qopenmp-simd –qopt-zmm-usage=high –DINTERCHANGE matmul_2D.cpp –o mm_interchange_icpc.out
Collect the survey data: advixe-cl –c survey -project-dir mm-advisor -- ./mm_interchange_icpc.out 1000
Open the result in GUI: advixe-gui mm-advisor



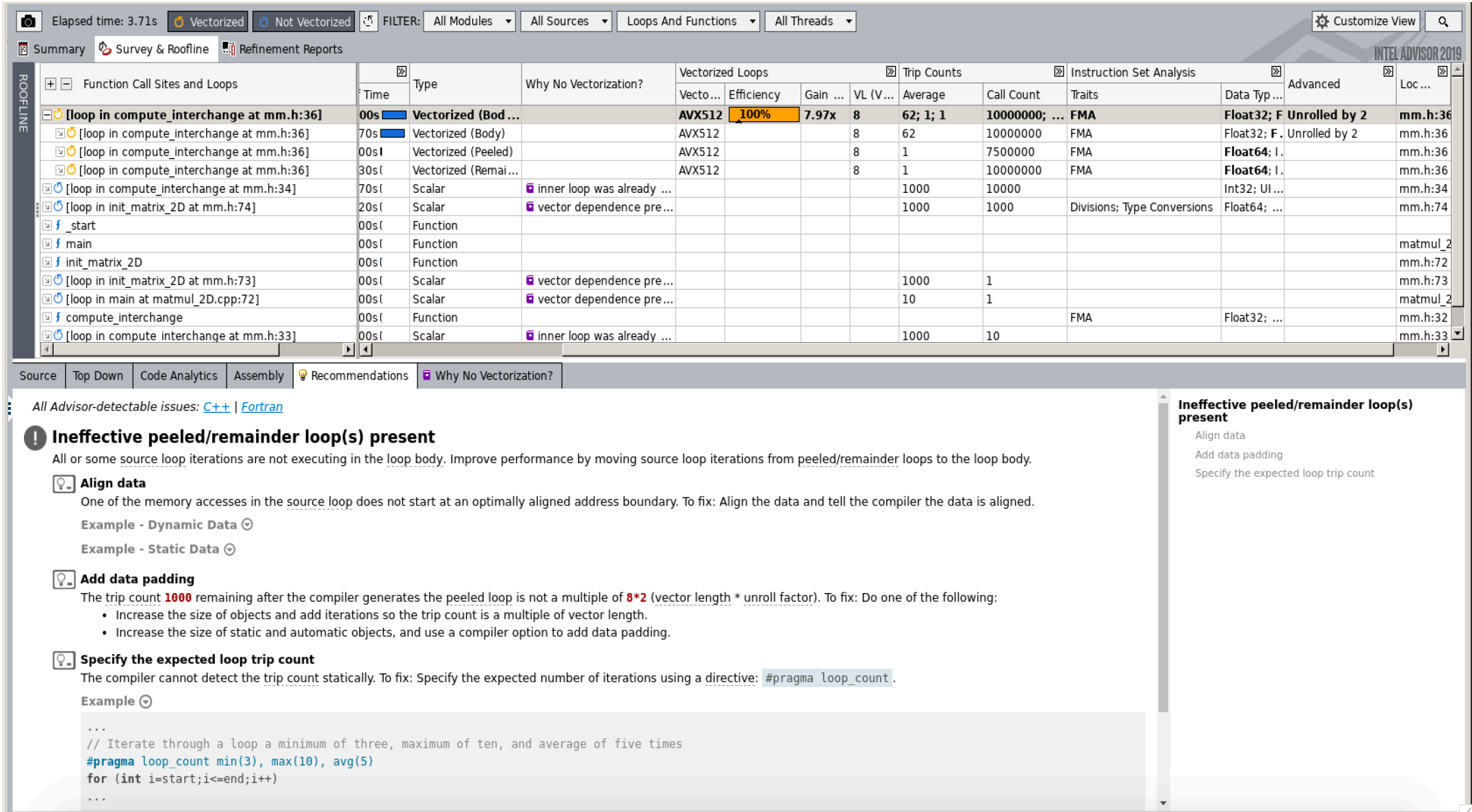
Trip Counts Analysis

Collect the trip counts data: advixe-cl -c tripcounts -project-dir mm-advisor -- ./mm_interchange_icpc.out 1000
Note: need to first do “survey” and use the same project for “tripcounts”

Trip Counts analysis shows you loop trip counts and call counts

The **best vectorization** requires the **scalar trip count to be divisible by the vector length**, or you get remainder loops

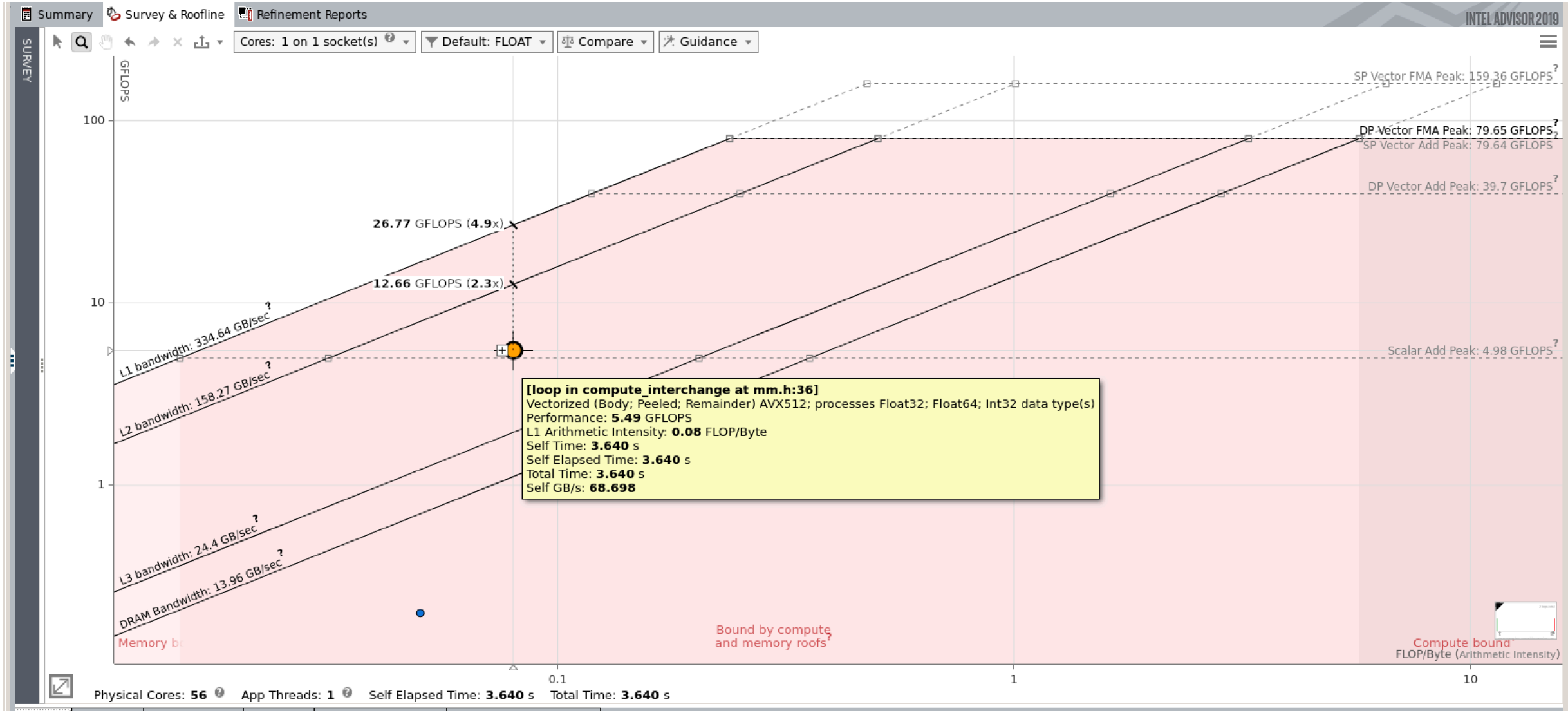
Call counts amplify the importance of tuning a given loop



Roofline Analysis

Collect the dependency data: advixe-cl -c tripcounts -flop -project-dir mm-advisor -- ./mm_interchange_icpc.out 1000

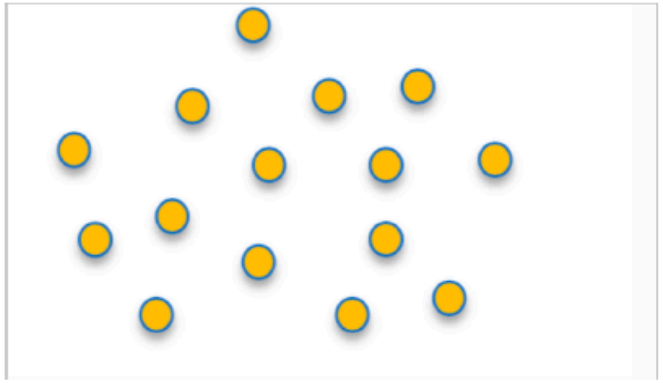
A visual representation of application performance in relation to hardware limitations, including memory bandwidth and computational peaks



N-Body problem

```
struct Particle {
  float x, y, z;
  float vx, vy, vz;
};
```

```
for (int i = 0; i < nParticles; i++) {
  // Components of the gravity force on particle i
  float Fx = 0, Fy = 0, Fz = 0;

  const float xi = particle[i].x;
  const float yi = particle[i].y;
  const float zi = particle[i].z;

  for (int j = 0; j < nParticles; j++) {
    // Newton's law of universal gravity
    const float dx = particle[j].x - xi;
    const float dy = particle[j].y - yi;
    const float dz = particle[j].z - zi;

    const float drPower32  = pow(drSquared, 3.0/2.0);

    const float drPower32Inv = 1.0f / drPower32;
    // Calculate the net force
    Fx += dx * G * drPower32Inv;
    Fy += dy * G * drPower32Inv;
    Fz += dz * G * drPower32Inv;
  }

  // Accelerate particles in response to the gravitational force
  particle[i].vx += dt*Fx;
  particle[i].vy += dt*Fy;
  particle[i].vz += dt*Fz;
}
```

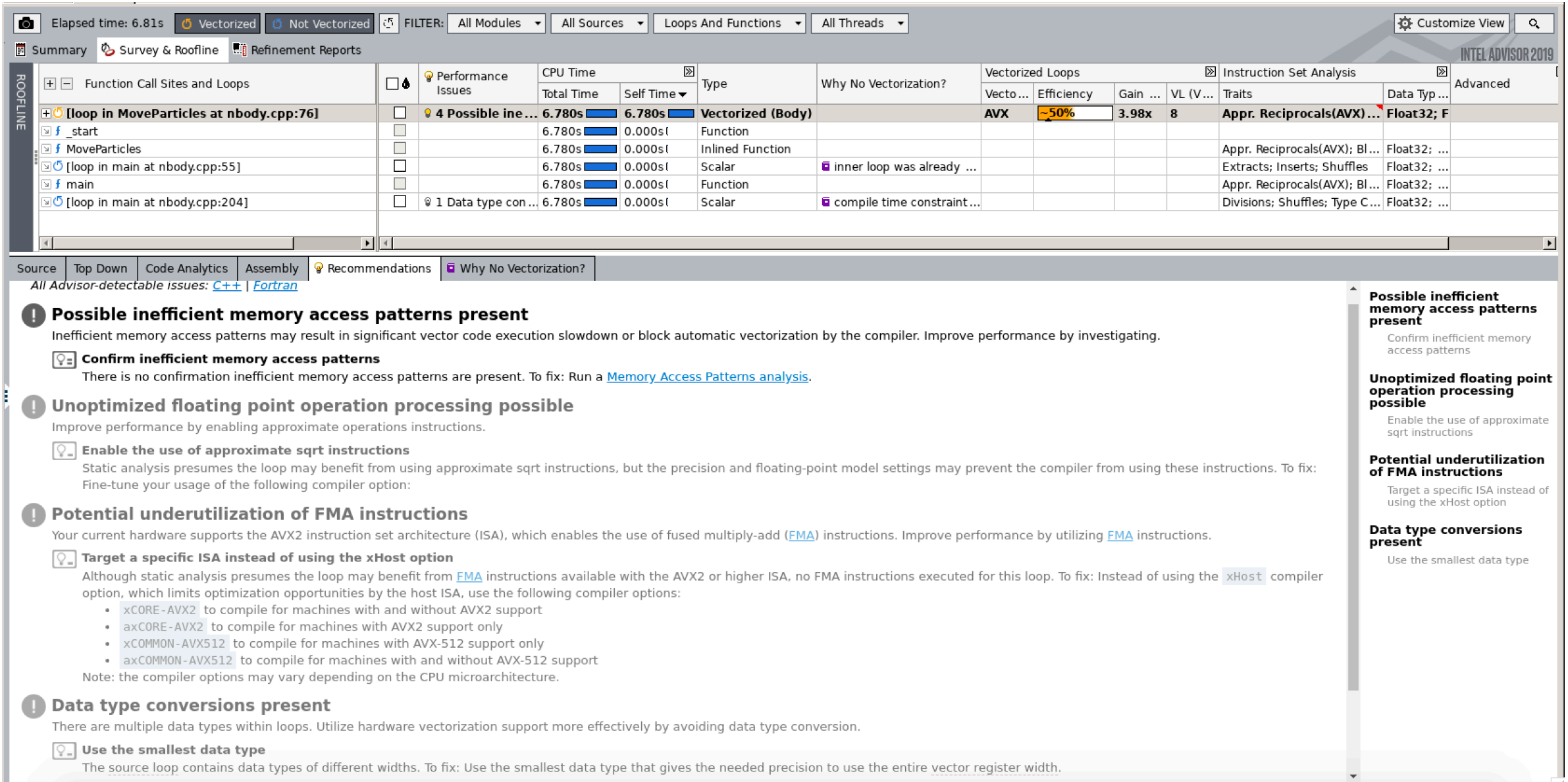


$$\vec{F}_{ij} = \frac{G m_i m_j}{\left|\vec{r}_j - \vec{r}_i\right|^3}\left(\vec{r}_j - \vec{r}_i\right)$$

$$\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = m\frac{d^2\vec{x}}{dt^2}$$

The example code assumes m=1 for all particles

N-body Performance issues?



Revisit N-body code

```
struct Particle {
  float x, y, z;
  float vx, vy, vz;
};
```

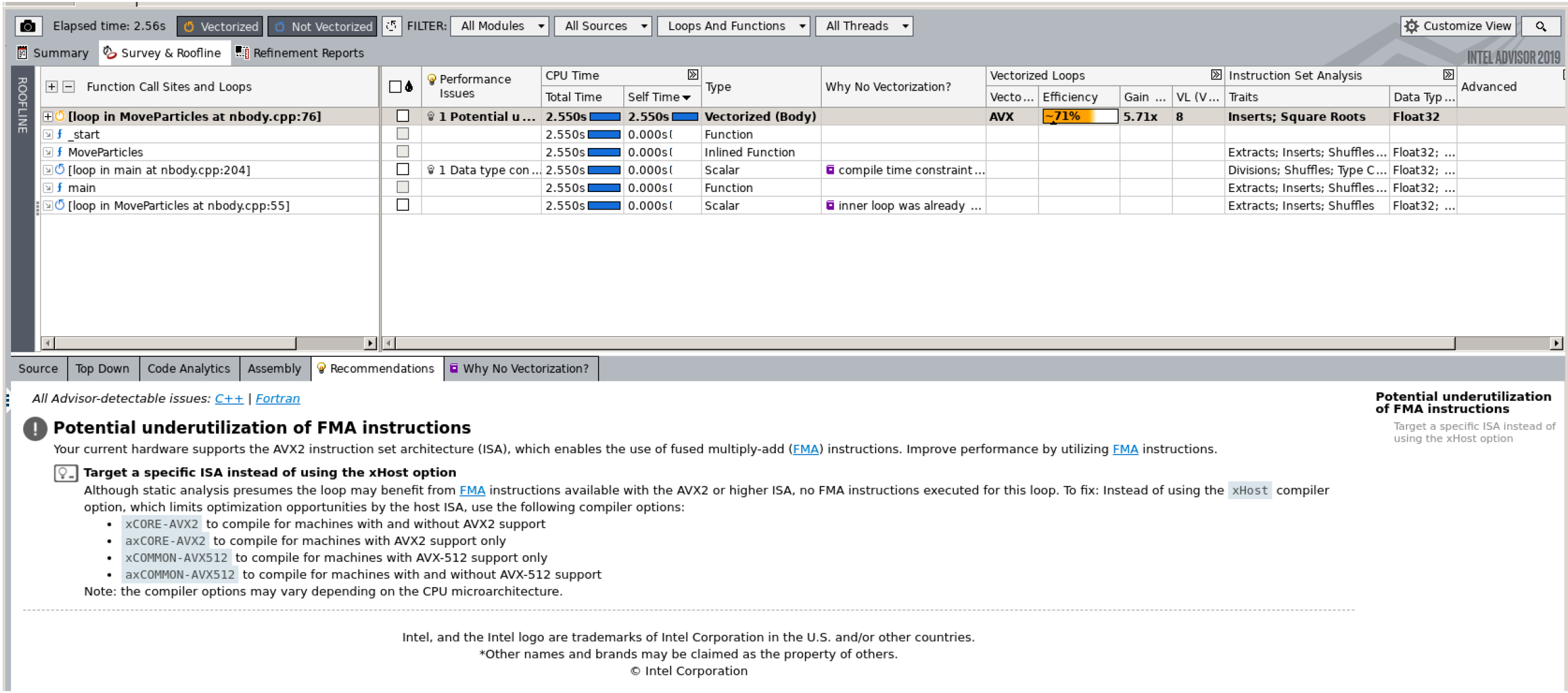
```
struct ParticleArrays {
  float *x, *y, *z;
  float *vx, *vy, *vz;
};
```

```
for (int i = 0; i < nParticles; i++) {
  // Components of the gravity force on particle i
  float Fx = 0, Fy = 0, Fz = 0;

  const float xi = particle[i].x;
  const float yi = particle[i].y;
  const float zi = particle[i].z;

  for (int j = 0; j < nParticles; j++) {
    // Newton's law of universal gravity
    const float dx = particle[j].x - xi;
    const float dy = particle[j].y - yi;
    const float dz = particle[j].z - zi;

    const float drPower32  = pow(drSquared, 3.0/2.0);

    const float drPower32Inv = 1.0f / drPower32;
    // Calculate the net force
    Fx += dx * G * drPower32Inv;
    Fy += dy * G * drPower32Inv;
    Fz += dz * G * drPower32Inv;
  }

  // Accelerate particles in response to the gravitational force
  particle[i].vx += dt*Fx;
  particle[i].vy += dt*Fy;
  particle[i].vz += dt*Fz;
}
```

```
const float drPower32  = powf(drSquared, 3.0f/2.0f);
```

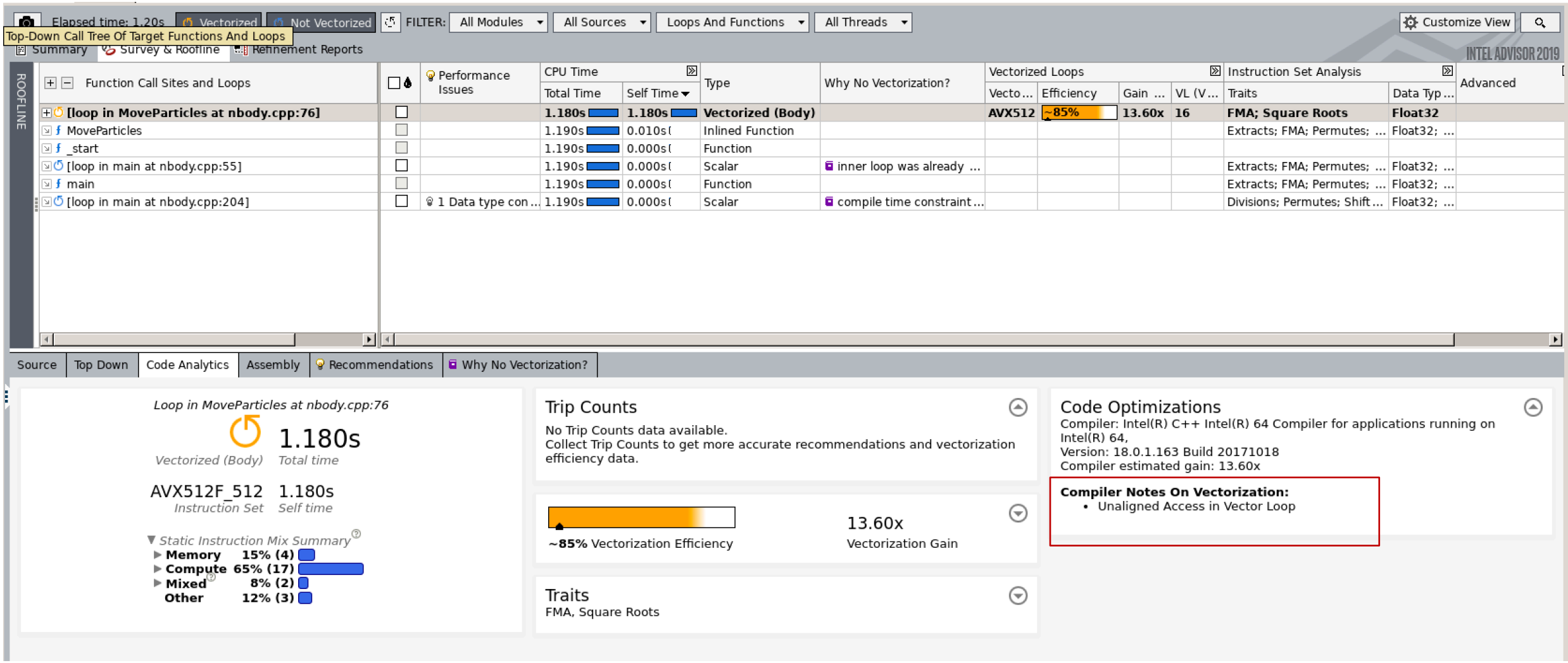
N-Body revised

Compile with -xhost –DSoA -DNo_FP_Conv



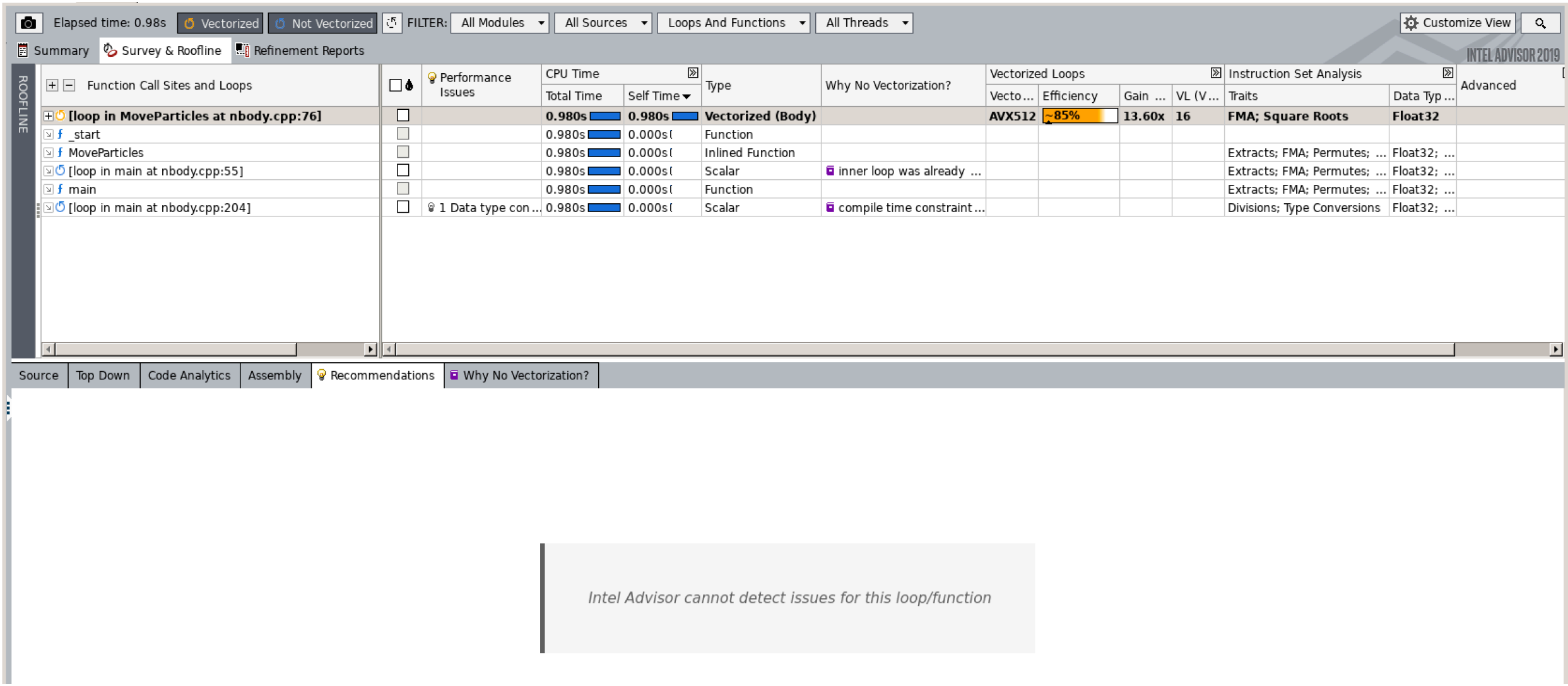
Any remaining performance issues?

Compile with -xCOMMON-AVX512 –DSoA -DNo_FP_Conv

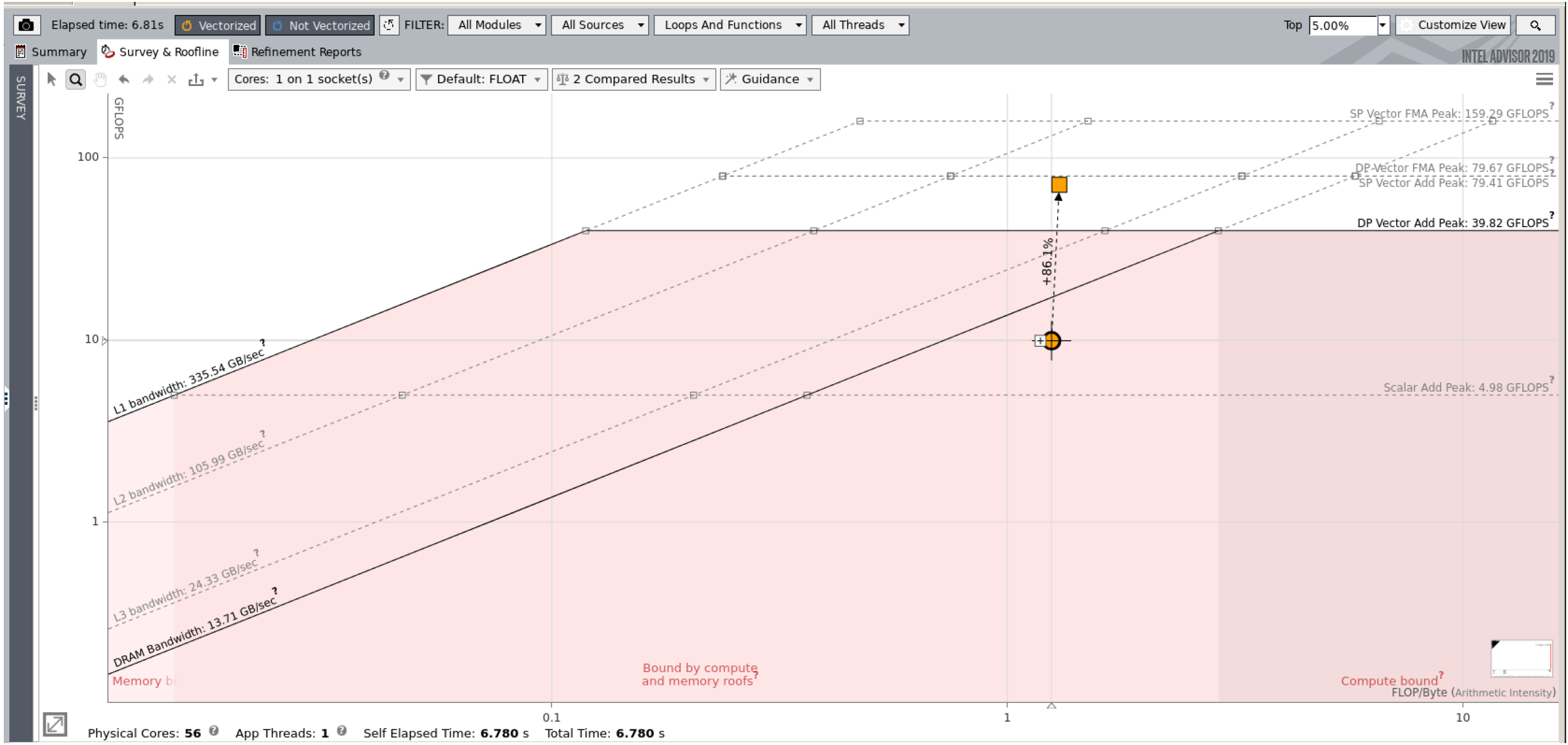


N-Body optimized

Compile with -xCOMMON-AVX512 –DSoA -DNo_FP_Conv -DAligned



N-Body Roofline comparison



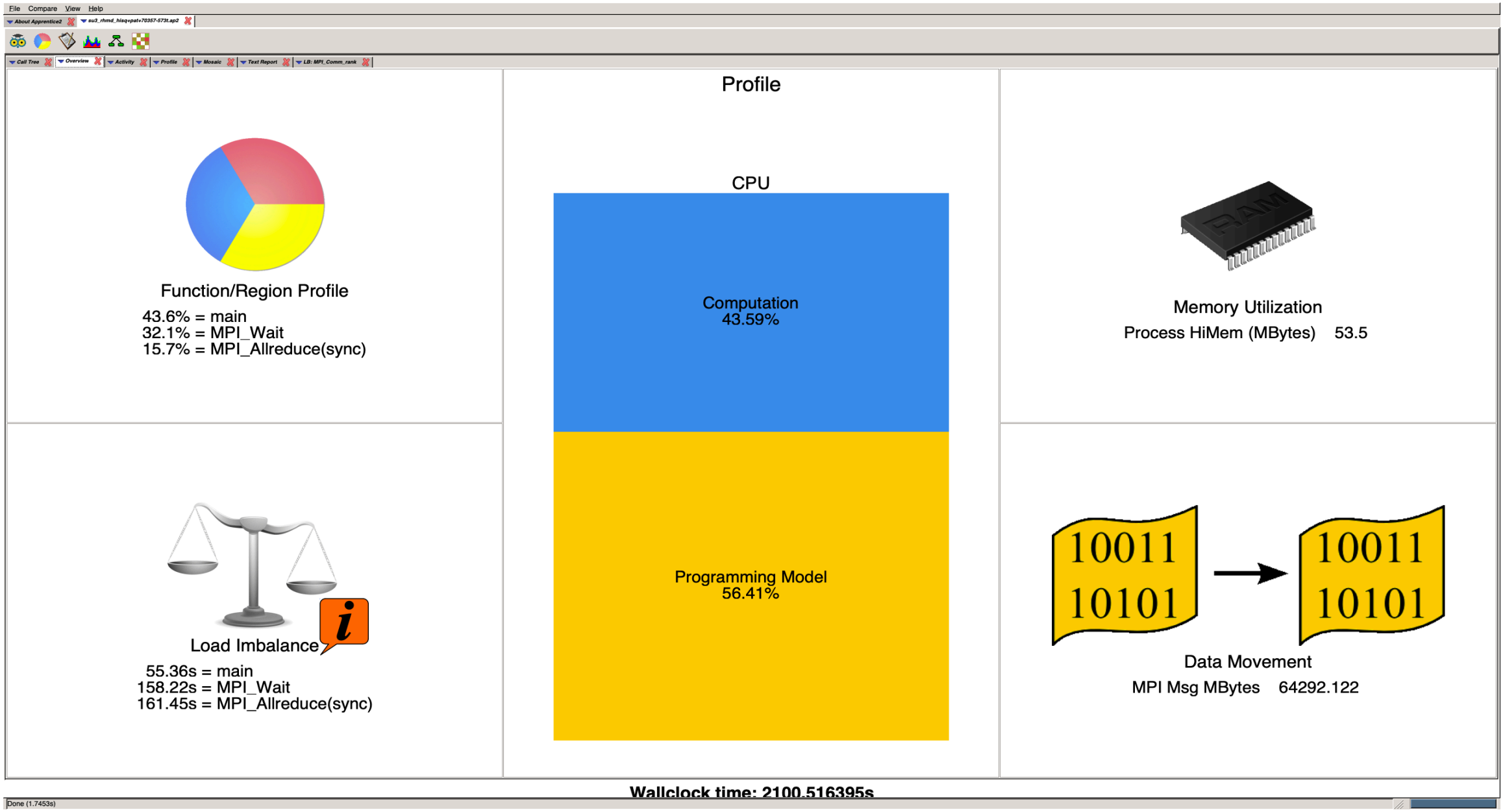
Cray Performance Analysis Tools (perftools)

Cray Performance Analysis Tools

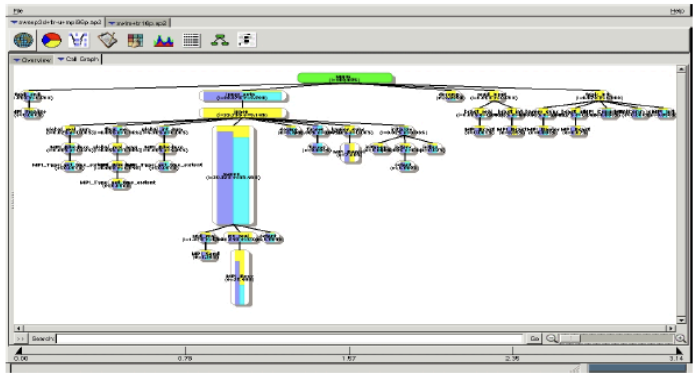
- Whole program performance analysis with
  - Novice and advanced user interfaces
  - Support for MPI, SHMEM, OpenMP, UPC, CAF, OpenACC, CUDA
  - Load imbalance detection
  - HW counter metrics (hit rates, computational intensity, etc.)
  - Observations and inefficiencies
  - Data correlation to user source
  - Minimal program perturbation

- Sampling, tracing with runtime summarization, full trace (timeline) modes available
- Supports CCE, Intel and GCC compilers

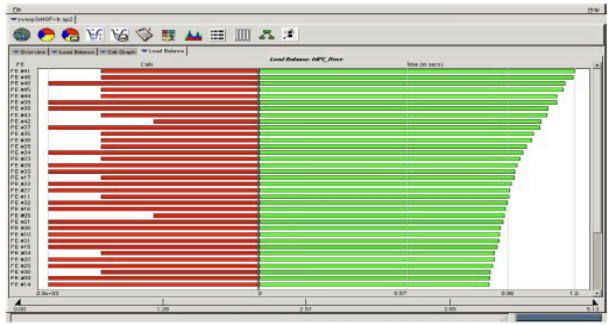
Cray PAT Overview



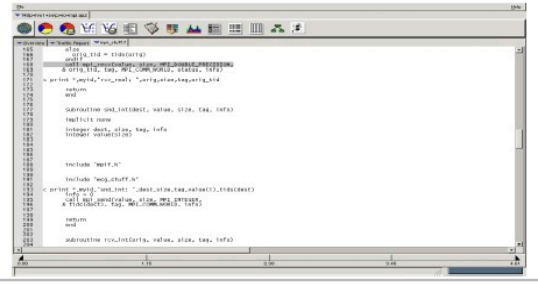
Cray Apprentice^2

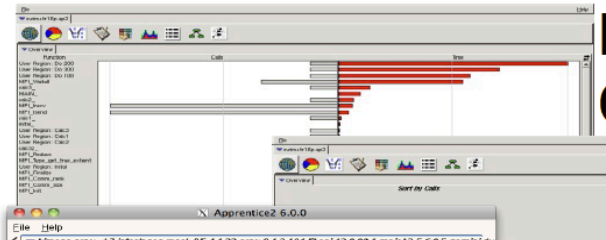


Load balance views

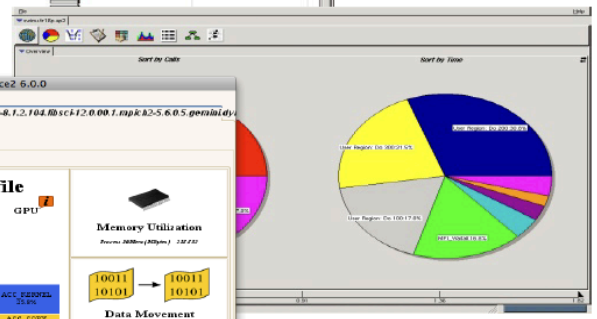


Source code mapping

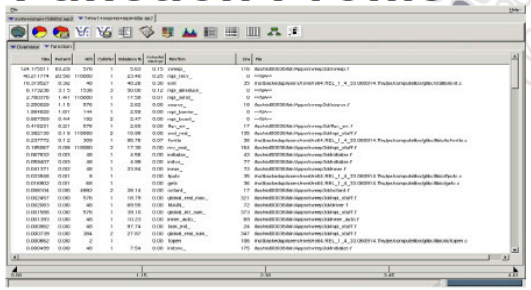


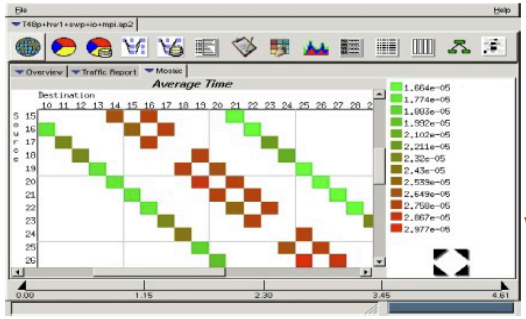


Program Overview

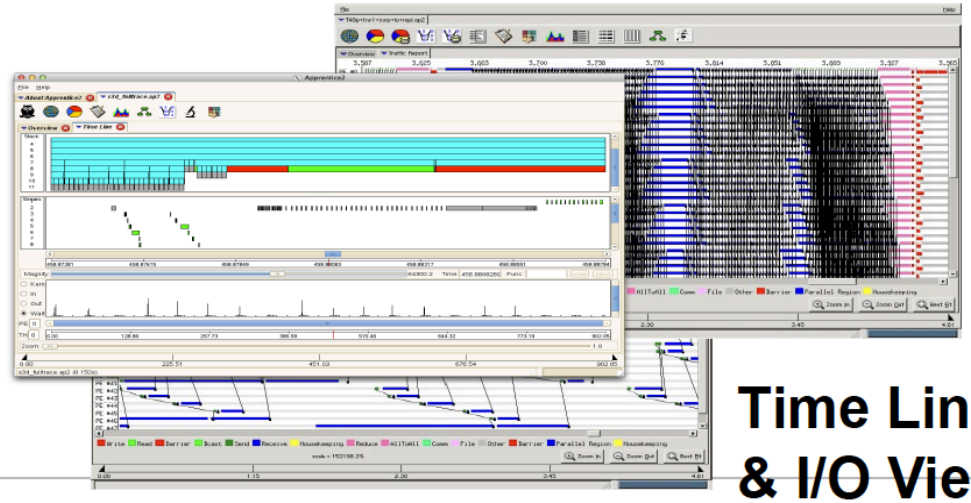


Function Profile

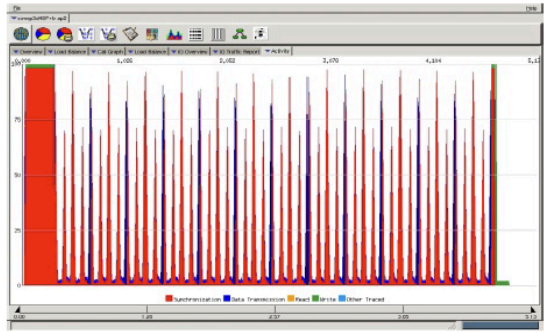




Pair-wise Communication View



Time Line & I/O Views



Communication & I/O Activity View

Profiling with CrayPat

- Application built with “pat_build -g mpi”
- Pat_report generates the CrayPat report

- Note the MPI call times, calls
- Load imbalance across the ranks

```
Table 1:  Profile by Function Group and Function
  Time% |       Time | Imb. Time |  Imb. |        Calls | Group
        |            |           | Time% |              |  Function
        |            |           |       |              |   PE=HIDE
 100.0% | 667.935156 |        -- |    -- | 49,955,946.2 | Total
|-------------------------------------------------------------------------
|  40.0% | 267.180169 |        -- |    -- | 49,798,359.2 | MPI
||------------------------------------------------------------------------
||  24.0% | 160.400193 | 28.907525 | 15.3% |  2,606,756.0 | MPI_Wait
||   6.4% |  42.897564 |  0.526996 |  1.2% |    157,477.0 | MPI_Allreduce
||   4.8% |  31.749303 |  3.923541 | 11.0% | 42,853,974.0 | MPI_Comm_rank
||   3.5% |  23.303805 |  1.774076 |  7.1% |  1,303,378.0 | MPI_Isend
||   1.1% |   7.658009 |  0.637044 |  7.7% |  1,303,378.0 | MPI_Irecv
||========================================================================
|  39.1% | 260.882504 |        -- |    -- |          2.0 | USER
||------------------------------------------------------------------------
||  39.1% | 260.882424 | 17.270557 |  6.2% |          1.0 | main
||========================================================================
|  20.9% | 139.872482 |        -- |    -- |    157,585.0 | MPI_SYNC
||------------------------------------------------------------------------
||  20.4% | 136.485384 | 36.223589 | 26.5% |    157,477.0 | MPI_Allreduce(sync)
|=========================================================================
```

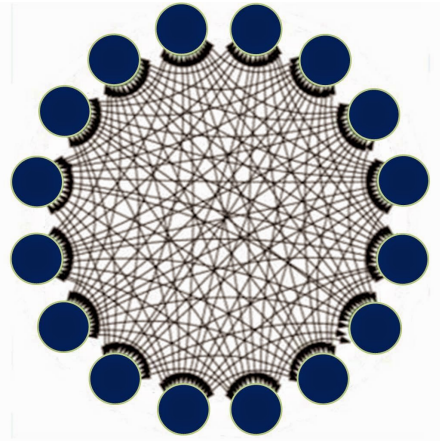
Profiling with CrayPat

- MPI message sizes are reported

```
=============================================================
  Total
-------------------------------------------------------------
  MPI Msg Bytes%                              100.0%
  MPI Msg Bytes                    18,052,938,280.0
  MPI Msg Count                         1,460,959.0 msgs
  MsgSz <16 Count                         157,529.0 msgs
  16<= MsgSz <256 Count                       65.0 msgs
  256<= MsgSz <4KiB Count                  2,815.0 msgs
  4KiB<= MsgSz <64KiB Count             1,300,511.0 msgs
  64KiB<= MsgSz <1MiB Count                   39.0 msgs
=============================================================
  MPI_Isend
-------------------------------------------------------------
  MPI Msg Bytes%                              100.0%
  MPI Msg Bytes                    18,051,670,432.0
  MPI Msg Count                         1,303,378.0 msgs
  MsgSz <16 Count                             16.0 msgs
  16<= MsgSz <256 Count                        0.0 msgs
  256<= MsgSz <4KiB Count                  2,812.0 msgs
  4KiB<= MsgSz <64KiB Count             1,300,511.0 msgs
  64KiB<= MsgSz <1MiB Count                   39.0 msgs
=============================================================
```

Topology Mapping and Rank reordering

- Topology mapping
  - Minimize communication costs through interconnect topology aware *task mapping*
  - Could ***potentially*** help reduce congestion
  - Node placement for the job could be a factor (no explicit control available to request a specific placement)
- *Application communication pattern*
  - MPI process topologies expose this in a portable way
  - Network topology agnostic
- *Rank reordering*
  - Can override the default mapping scheme
  - The default policy for **aprun** launcher is SMP-style placement
  - To display the MPI rank placement information, set **MPICH_RANK_REORDER_DISPLAY**.



Rank Reordering

- MPICH_RANK_REORDER_METHOD
  - Vary rank placement to optimize communication (Maximize on-node communication between MPI ranks)
  - Use CrayPat with “-g mpi” to produce a specific **MPICH_RANK_ORDER** file to maximize intra-node communication
  - Or, use perf_tools **grid_order** command with your application's grid dimensions to layout MPI ranks in alignment with data grid
  - To use:
    - name your custom rank order file: MPICH_RANK_ORDER
    - This approach is physical system topology agnostic

    export MPICH_RANK_REORDER_METHOD=3

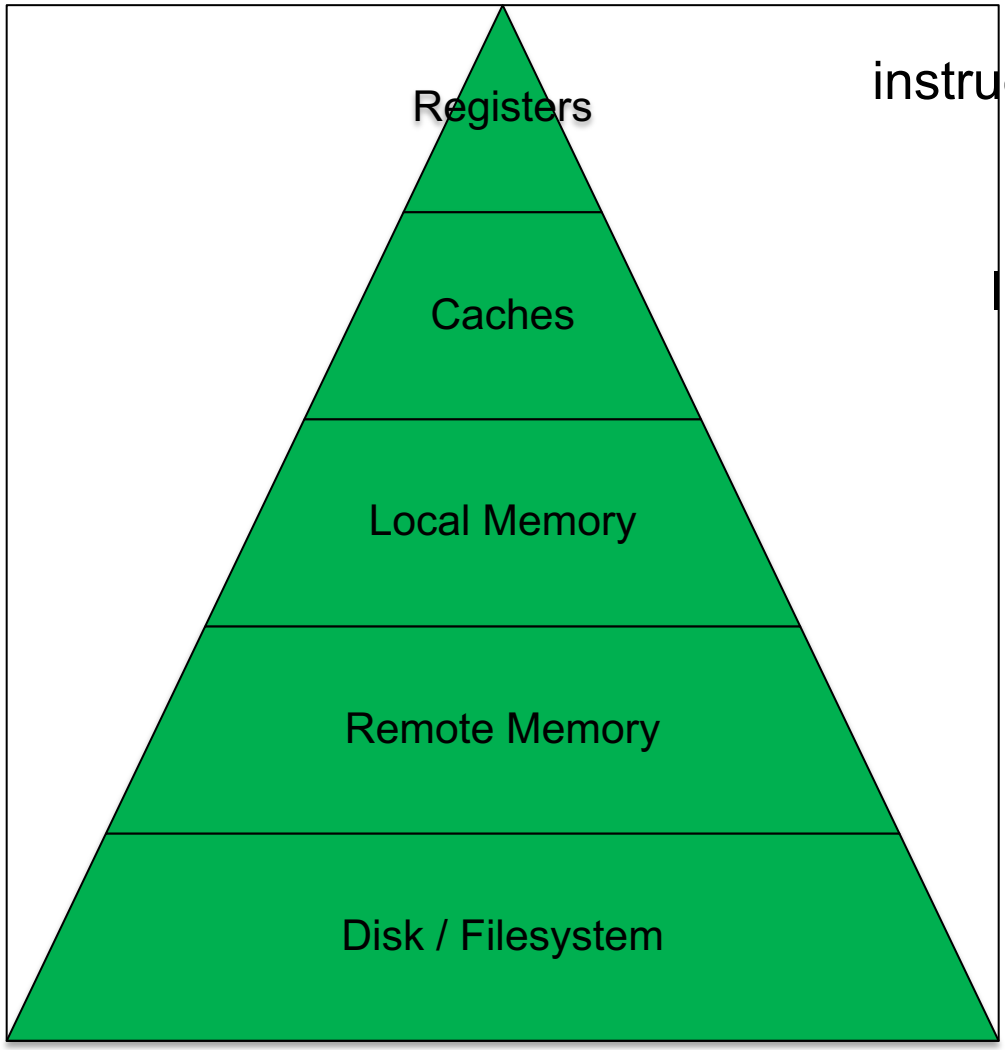
Rank Reordering

- MPICH_RANK_REORDER_METHOD (cont.)
  - A topology and placement aware reordering method is also available
  - Optimizes rank ordering for Cartesian decompositions using the layout of nodes in the job
  - To use:
    - **export MPICH_RANK_REORDER_METHOD=4**
    - **export MPICH_RANK_REORDER_OPTS="–ndims=3 –dims=16,16,8"**

```
MPI Grid Detection:
There appears to be point-to-point MPI communication in a 96 X 8 grid pattern. The 52% of the total execution time
spent in MPI functions might be reduced with a rank order that maximizes communication between ranks on the same node.
The effect of several rank orders is estimated below.

A file named MPICH_RANK_ORDER.Grid was generated along with this report and contains usage instructions and the Custom
rank order from the following table.

          Rank     On-Node   On-Node   MPICH_RANK_REORDER_METHOD
         Order     Bytes/PE  Bytes/PE%
                   of Total
                   Bytes/PE

         Custom     2.385e+09 95.55%   3
         SMP        1.880e+09 75.30%   1
         Fold       1.373e+06 0.06%    2
         RoundRobin 0.000e+00 0.00%    0
```

Summary on performance tools

Performance is Hierarchical



instructions & operands

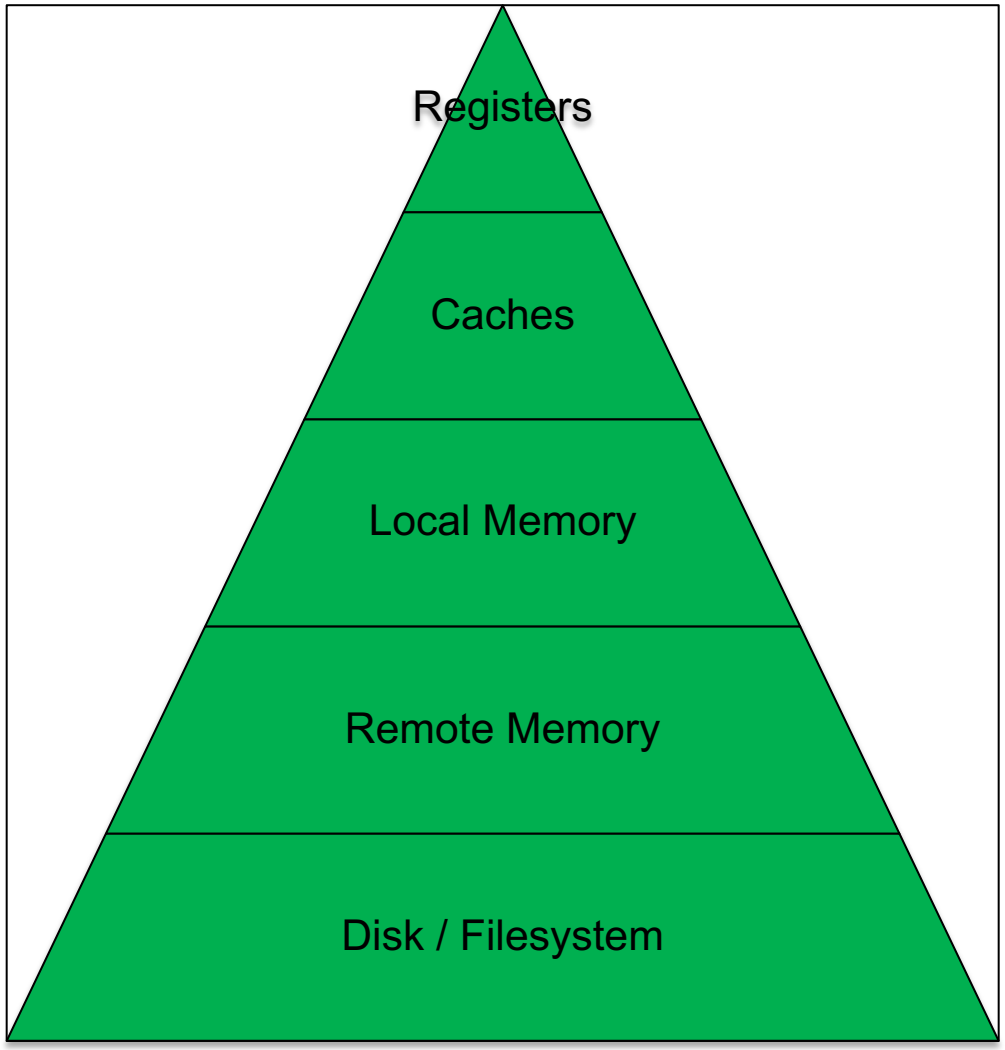
lines

pages

messages

blocks, files

Tools are Hierarchical



PAPI

Valgrind

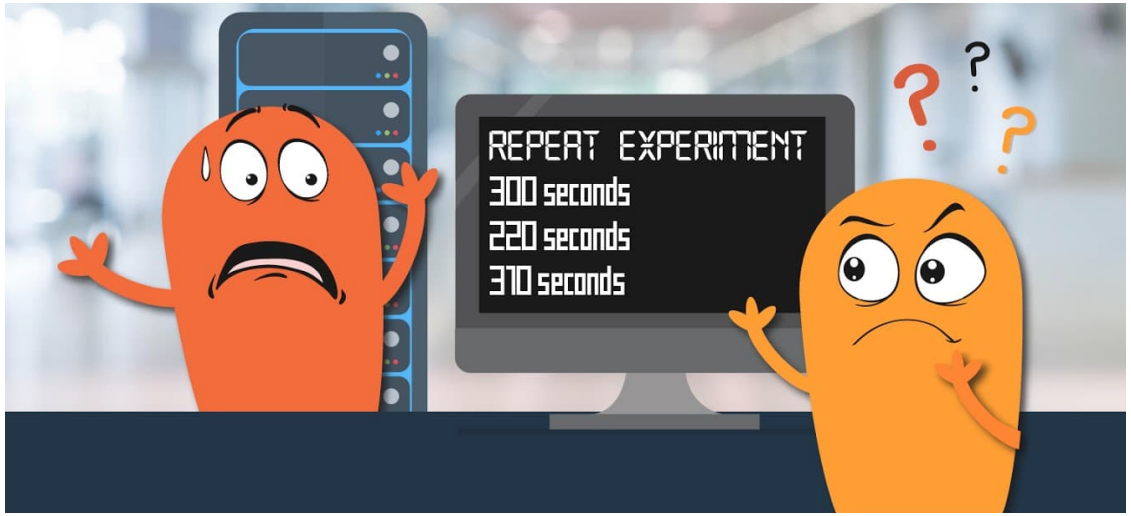
PMPI

POSIX

Craypat
IPM
TAU
HPCToolKit
...

My Research work

Run-to-run Variability

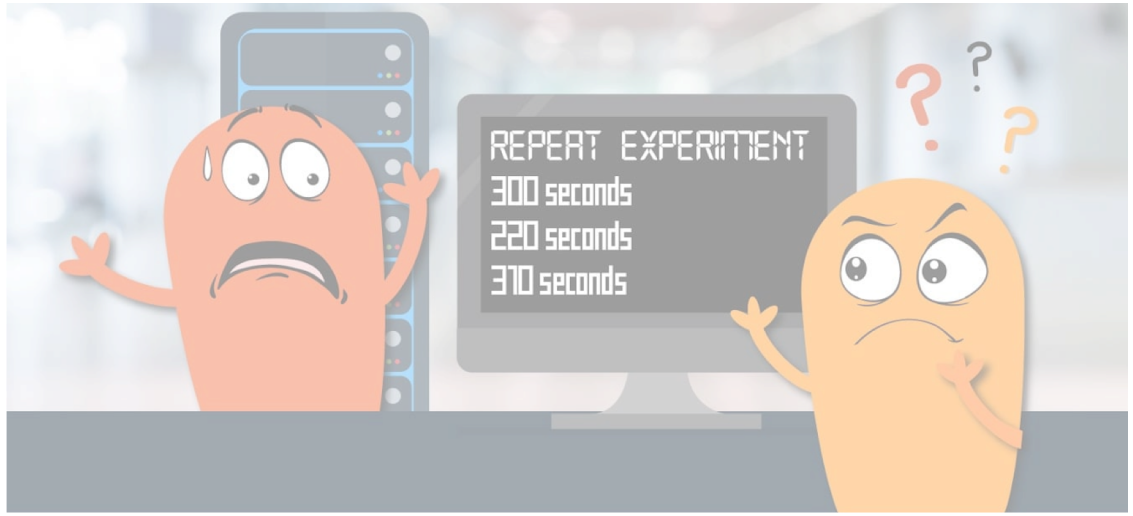


Equal work is not Equal time

Image courtesy: https://concertio.com/2018/07/02/dealing-with-variability/

Equal work is not Equal time

- **Sources of Variability**
  - Core-level
    - OS noise effects
    - Dynamic frequency scaling
    - Manufacturing variability
  - Node level
    - Shared cache contention on a multi-core
  - System level
    - Network congestion due to inter-job interference

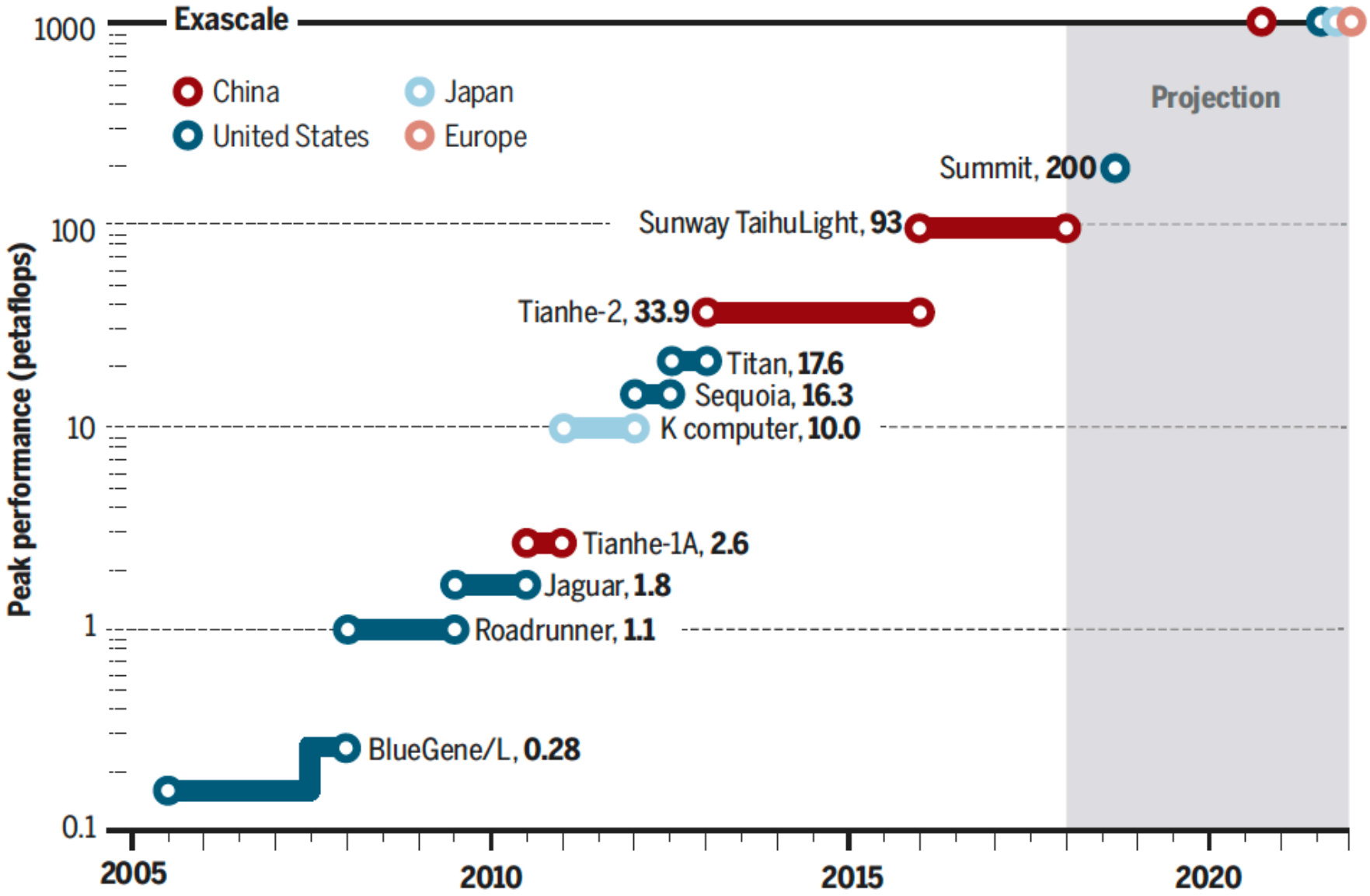


Equal work is not Equal time

- **Challenges**
  - Less reliable performance measures (multiple repetitions with statistical significance analysis is required)
  - Performance tuning – quantifying the impact of a code change is difficult
  - Difficult to predict job duration
    - Less user productivity
    - Inefficient system utilization
    - Complicates job scheduling

THE RACE TO EXASCALE

Top 500 #1 Supercomputers

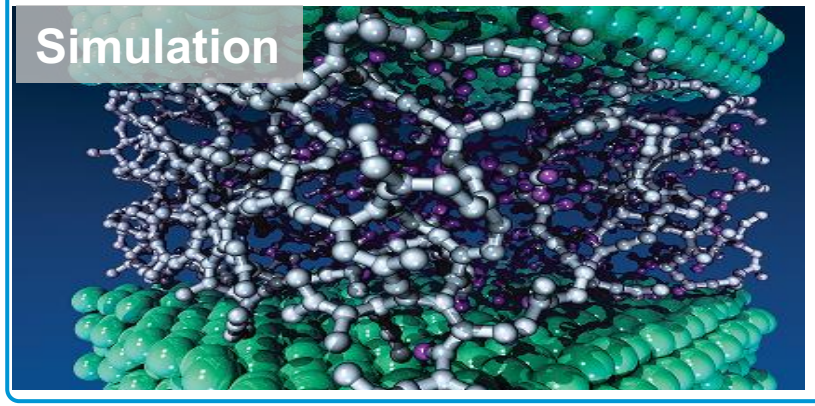


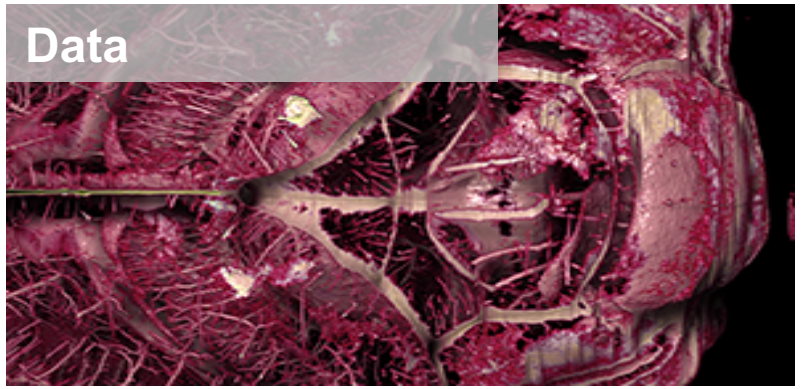
Credits: K. SUTLIFF/SCIENCE; DATA TOP500

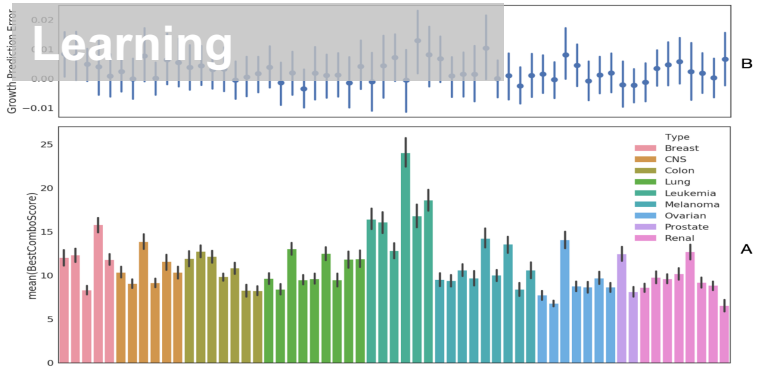
ALCF 2021 Exascale Supercomputer – Aurora

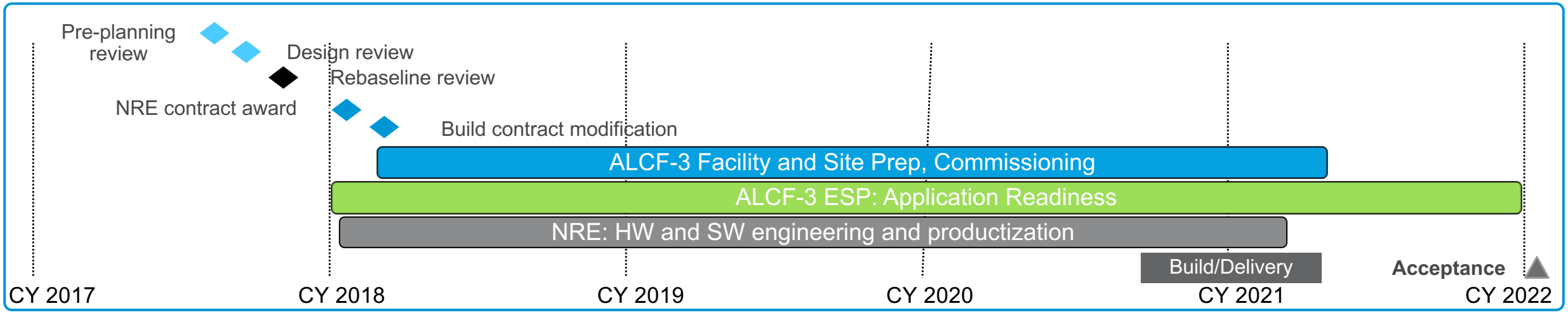
Intel Aurora supercomputer planned for 2018 shifted to 2021
Scaled up from **180 PF to over 1000 PF**

Support for three “pillars”









Aurora high-level configuration (public)

System Spec	Aurora
Delivery	CY2021
Sustained Performance	≥1EF DP
Compute Node	Intel Xeon scalable processors X^e arch based GP-GPUs
GPU Architecture	X^e arch based GPU Tile based, chiplets, HBM stack, Foveros 3D integration
CPU-GPU interconnect	PCIe
Aggregate System Memory	>10 PB
System Interconnect	Cray Slingshot Dragonfly topology with adaptive routing
Network Switch	25.6 Tb/s per switch, from 64 - 200 Gbs ports (25GB/s per direction)
High-Performance Storage	≥230 PB, ≥25 TB/s (DAOS)
Programming Models	Intel OneAPI, OpenMP, DPC++/SYCL
Software stack	Cray Shasta software stack + Intel enhancements + Data and Learning
Platform	Cray Shasta
# Cabinets	>100