Multiarchitecture Programming for Accelerated Compute, Freedom of Choice for Hardware

Intel® oneAPI HPC Toolkit

Intel® Advisor

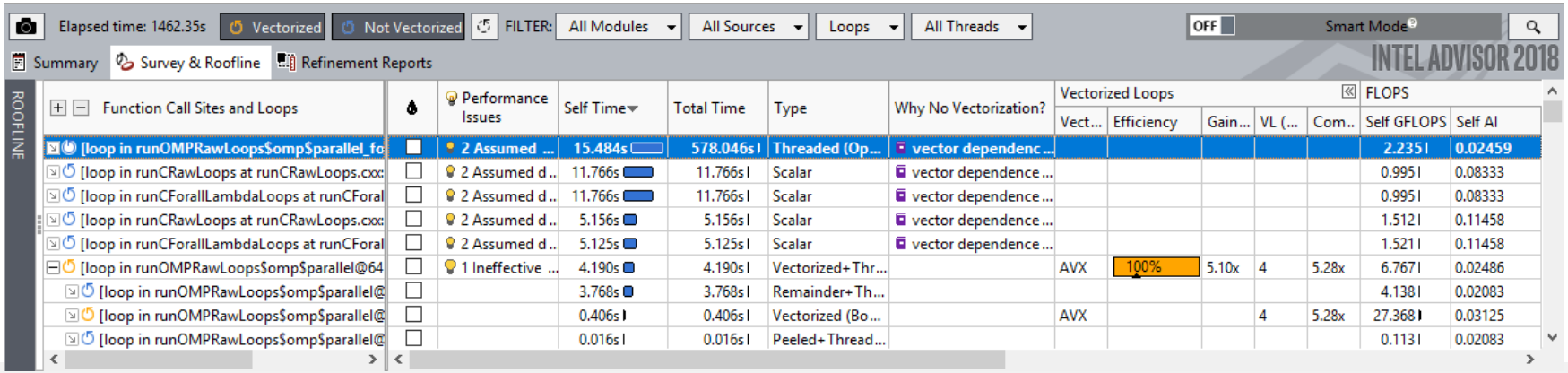
Intel® Advisor Vectorization Optimization

Intel® Advisor Vectorization Optimization

Have you:

Recompiled for AVX2 with little gain?
Wondered where to vectorize?
Recoded intrinsics for new arch.?
Struggled with compiler reports?

- Data Driven Vectorization:
  - What vectorization will pay off most?
  - What's blocking vectorization? Why?
  - Are my loops vector friendly?
  - Will reorganizing data increase performance?
  - Is it safe to just use pragma simd?



The Lab Activities

- Activity 0: Building N-body
- Activity 1: Doing Survey
- Activity 2: Fixing compilation option
- Activity 3: Doing roofline analysis
- Activity 4: Dealing with data type conversions
- Activity 5: Checking memory access patterns
- Activity 6: Using SDLT
- Activity 7: Comparing roofline charts
- Activity 8: Adding threading

N-BODY

N-body gravity simulation

- Let’s consider a distribution of point masses located at r_1,…,r_n and have masses m_1,…,m_n
- We want to calculate the position of the particles after a certain time interval using the Newton law of gravity

```
struct Particle
{
    float pos_x, pos_y, pos_z;
    float vel_x, vel_y, vel_z;
    float acc_x, acc_y, acc_z;
    float mass;
};


class GSimulation
{
…
private:
    std::vector<Particle> particles;
…
};
```

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        float distance, dx, dy, dz;
        float distanceSqr = 0.0;
        float distanceInv = 0.0;

        dx = particles[j].pos_x - particles[i].pos_x;
        …
        distanceSqr = dx*dx + dy*dy + dz*dz + softeningSquared;
        distanceInv = 1.0 / sqrt(distanceSqr);

        particles[i].acc_x += dx * G * particles[j].mass *
            distanceInv * distanceInv * distanceInv;
        particles[i].acc_y += …
        particles[i].acc_z += …
```

Activity 0: Building N-body

Build & Run

Purpose: Build an application, observe the performance

- Setup:

```
$ source /opt/intel/oneapi/setvars.sh
$ cd ~/day1/lab4/ver0
```

- Build & run

```
$ make
$ make run
```

Activity 0. Screenshot

```
================================
 Initialize Gravity Simulation
 nPart = 2000; nSteps = 500; dt = 0.1
----------------------------------------
 s       dt      kenergy     time (s)
----------------------------------------
 50      5       0.09659     3.8322
 100     10      1.6163      3.8374
 150     15      5.3975      3.8284
 200     20      11.879      3.8201
 250     25      21.908      3.8251
 300     30      37.079      3.821
 350     35      60.558      3.8171
 400     40      99.572      3.8149
 450     45      176.19      3.8152
 500     50      388.54      3.8322

# Total Time (s)      : 38.244
--------------------------------
```

Activity 1: Doing Survey

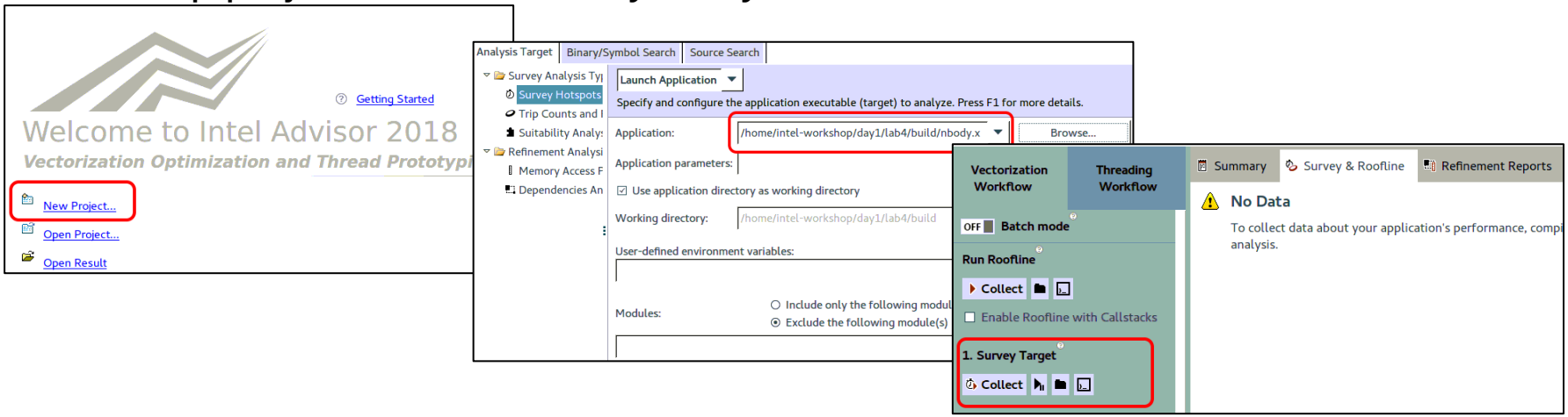
Advisor SURVEY Analysis

Purpose: Run Survey analysis in Advisor to get the baseline version

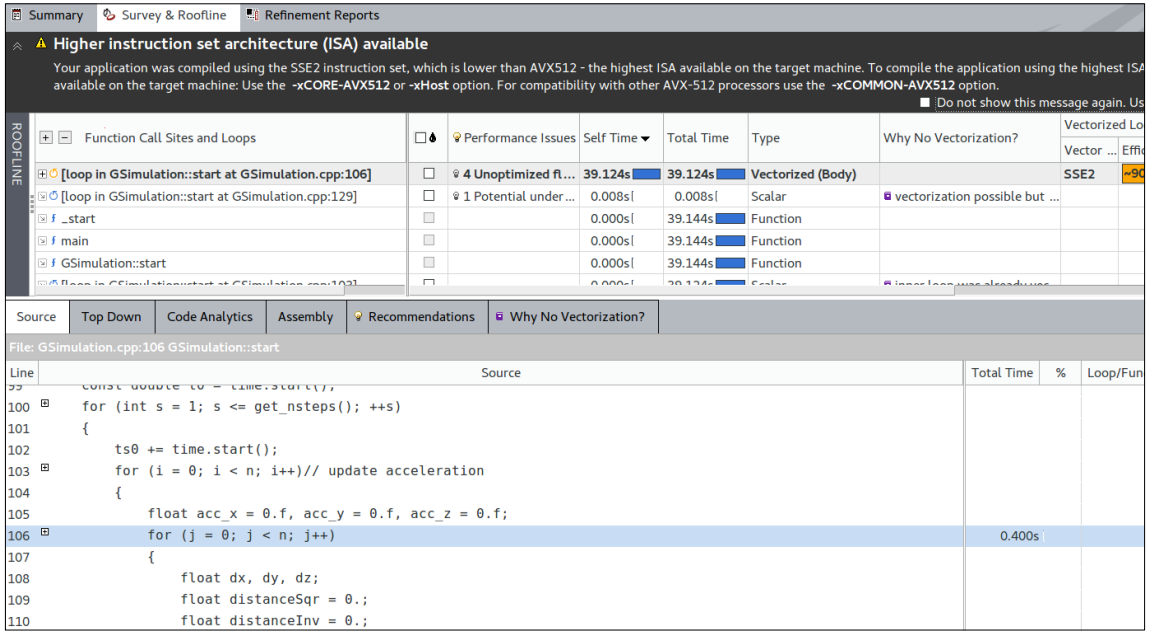
- Launch Advisor GUI:

```
$ advisor-gui
```

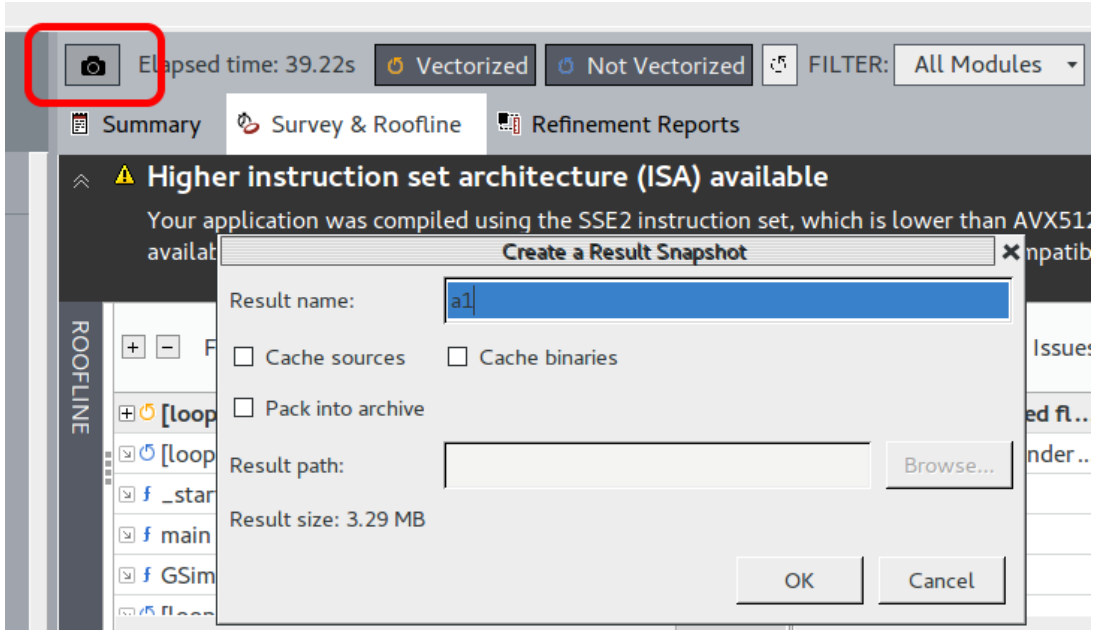
- Setup project and run Survey analysis



Activity 1. Screenshot



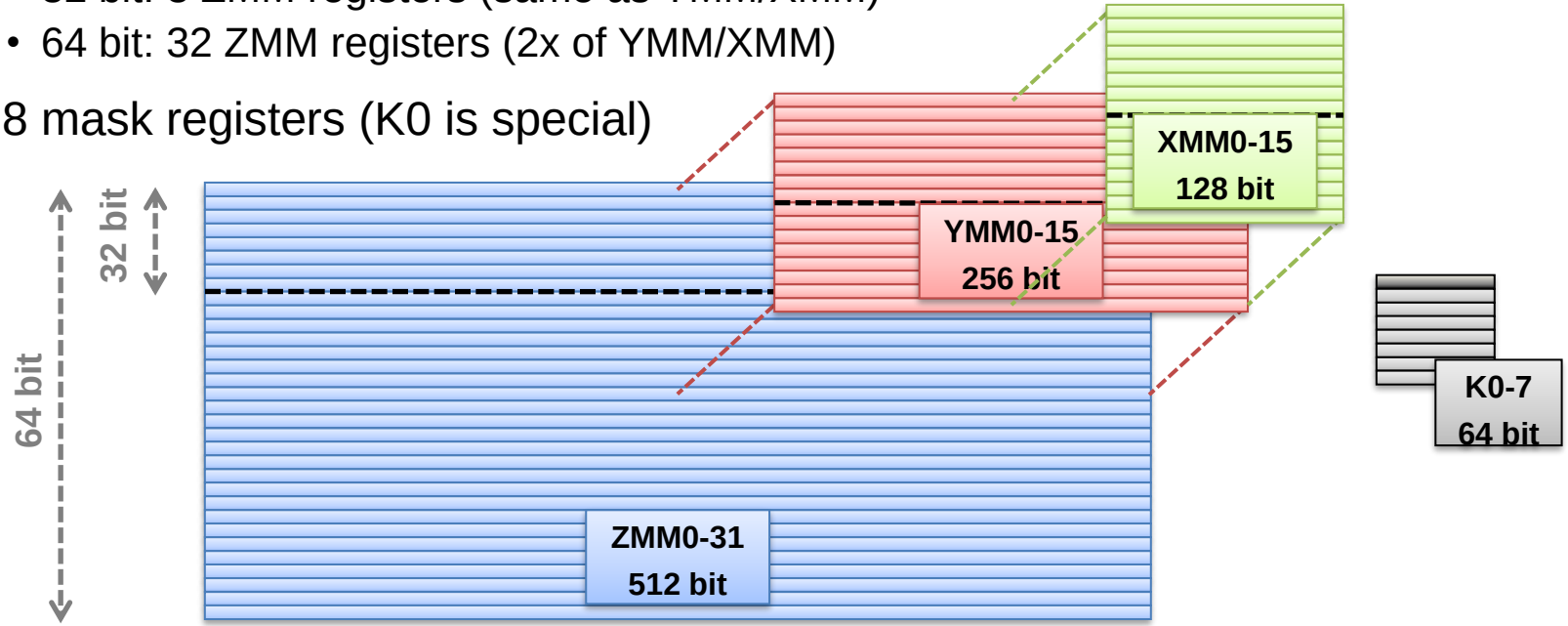
Create a snapshot



Activity 2: Fixing compilation option

Intel® AVX-512

- Extends previous AVX and SSE registers to 512 bit:
  - 32 bit: 8 ZMM registers (same as YMM/XMM)
  - 64 bit: 32 ZMM registers (2x of YMM/XMM)
- 8 mask registers (K0 is special)



Activity 2

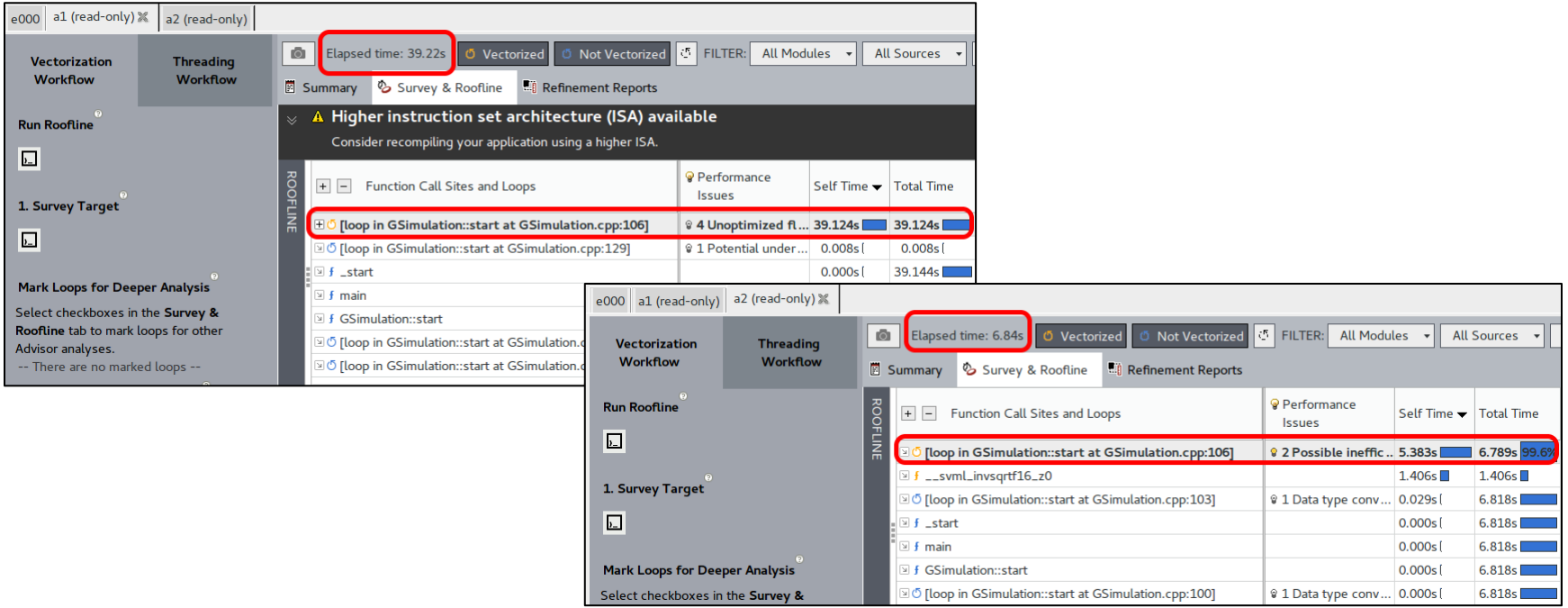
Purpose: Fix compilation options to use the highest available ISA

- Build a version with new compilation flags

```
$ cd ~/day1/lab4/ver1
$ make
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous activity

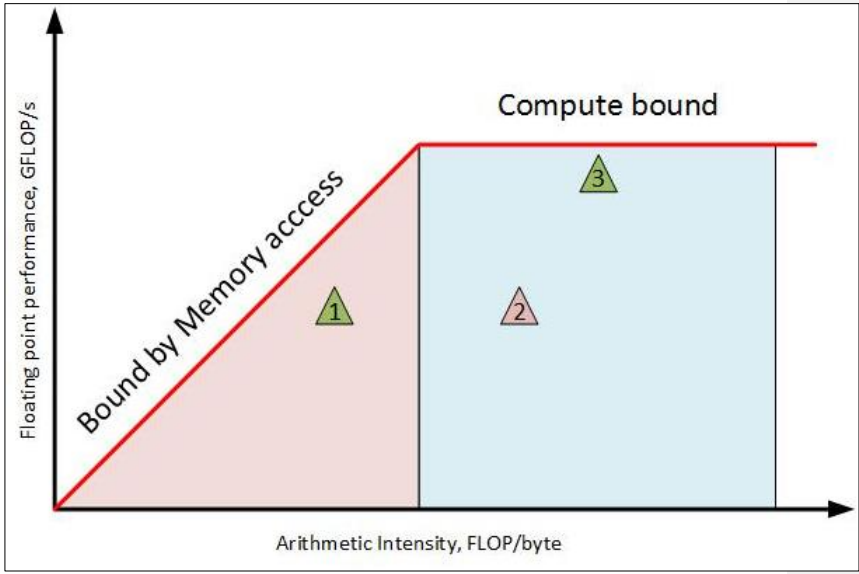
Activity 2. Screenshots



Activity 3: Doing roofline analysis

Roofline model

A roofline model helping you answer these questions:
Does my application work optimally on the current hardware? If not, what is the most underutilized hardware resource?
What limits performance? Is my application workload memory or compute bound?
What is the right strategy to improve application performance?



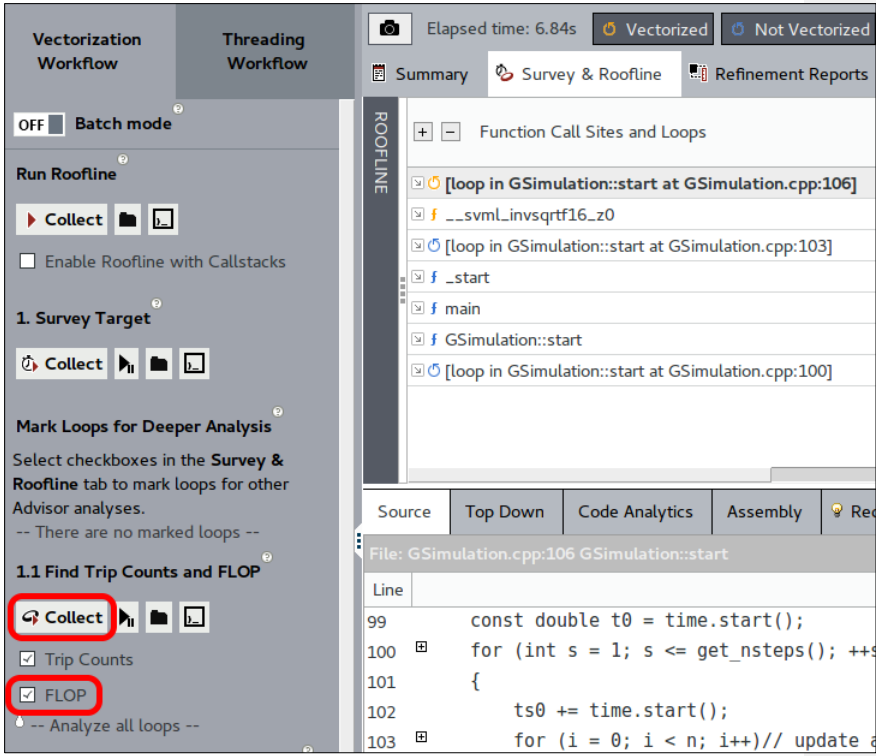
Collect FLOP data to GET ROOFLINE CHART

Purpose: Characterize the application using roofline model

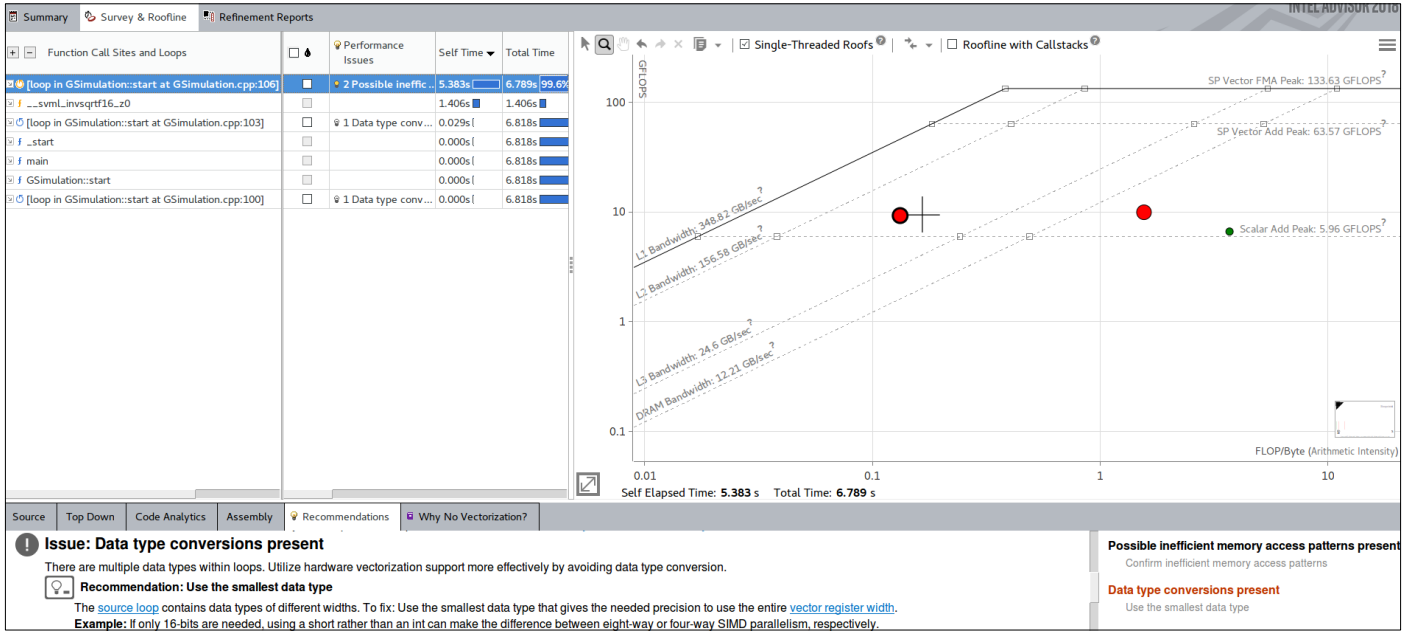
Click “FLOP” checkbox on workflow

Press “Collect” button in “1.1 Find Trip Counts and FLOP” section

Create a snapshot



Activity 3. Screenshot



Activity 4: Dealing with data type conversions

Activity 4

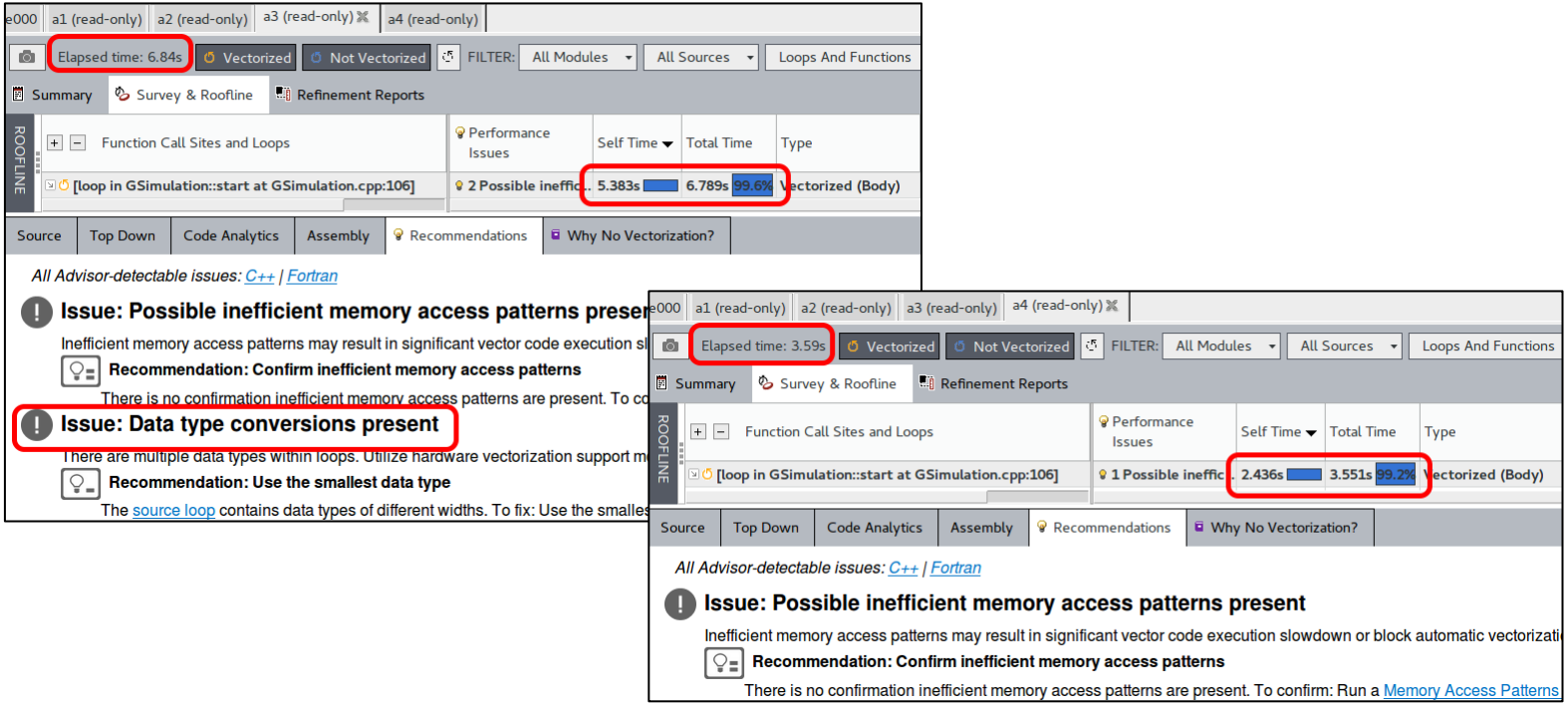
Purpose: Identify and fix data type conversion issue

- Build version without data type conversions

```
$ cd ~/day1/lab4/ver2
$ make
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous results

Activity 4. Screenshots



Activity 5: Checking memory access patterns

Types OF MEMORY Access patterns

Unit-Stride access

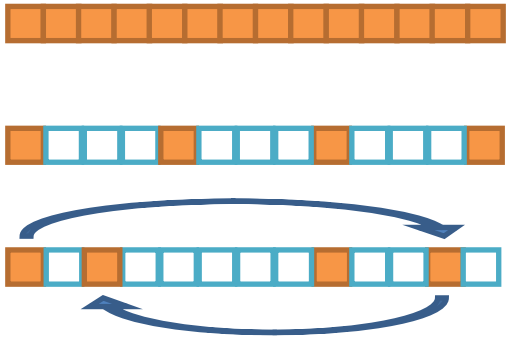
```
for (i=0; i<N; i++)
    A[i] = C[i]*D[i]
```

Constant stride access

```
for (i=0; i<N; i++)
    point[i].x = x[i]
```

Variable stride access

```
for (i=0; i<N; i++)
    A[B[i]] = C[i]*D[i]
```



Check Memory Access Patterns

Purpose: Investigate if poor memory access patterns are cause of poor vectorization efficiency

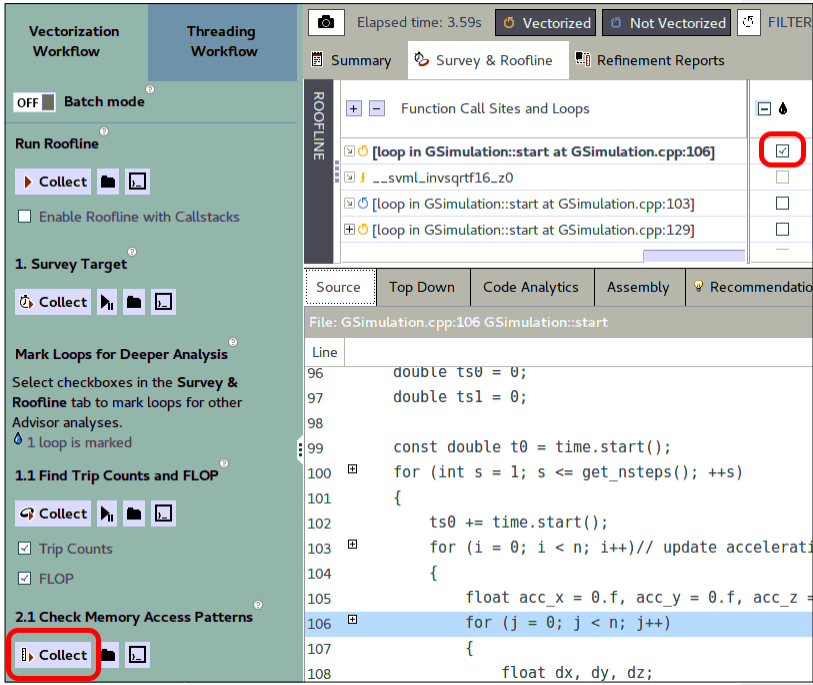
Mark the hottest loop using checkbox in Survey report

Press “Collect” button in “2.1 Check Memory Access Patterns” section

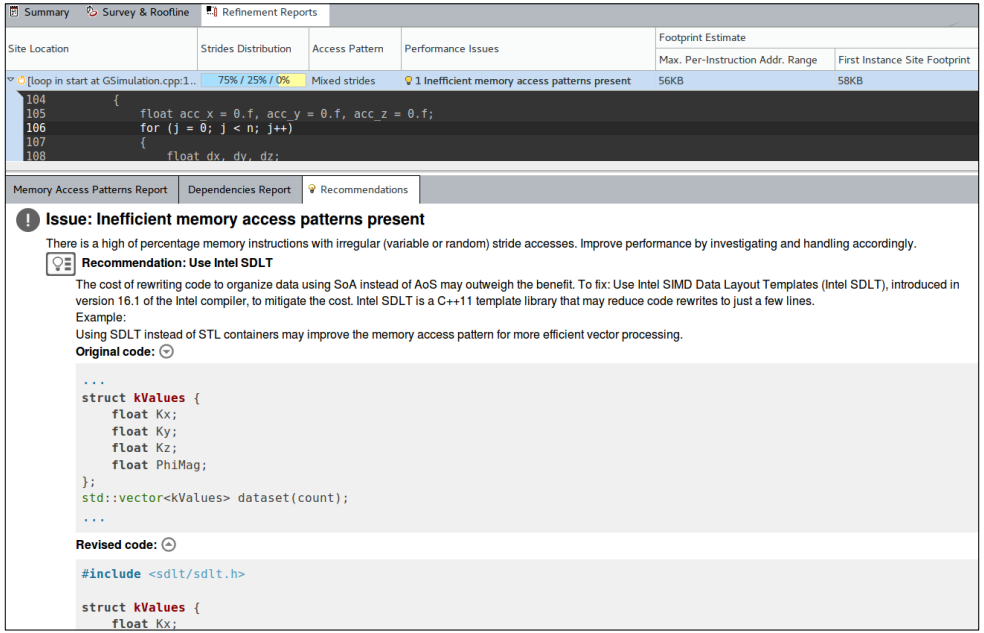
Create a snapshot

Investigate collected results

Review “Recommendations” tab



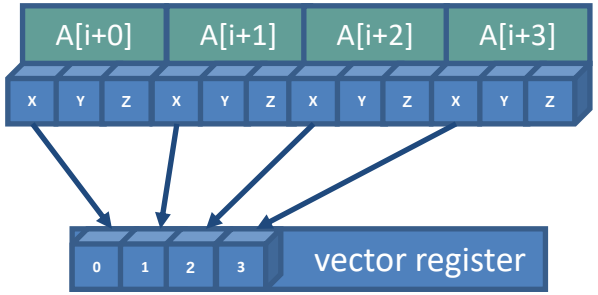
Activity 5. Screenshot



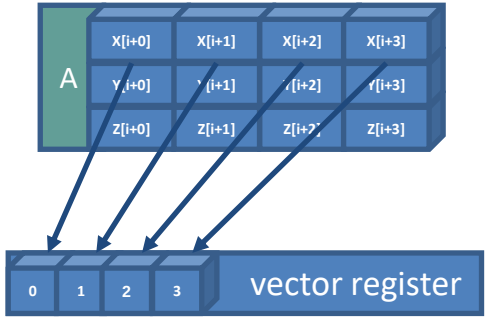
Activity 6: Using SDLT

SIMD Is Effective With Unit Stride Access

Getting Array of Structures (AoS) in memory data layout loaded into a vector register is a “strided” load/store operation requiring multiple load/shuffle/insert or gather instructions



A properly aligned Structure of Arrays (SoA) in memory data layout provides SIMD compatible Unit-Stride memory accesses



N-body SDLT code example

```
#include <sdlt/sdlt.h>
struct Particle
{
    float pos_x, pos_y, pos_z;
    float vel_x, vel_y, vel_z;
    float acc_x, acc_y, acc_z;
    float mass;
};
SDLT_PRIMITIVE(Particle, pos_x, pos_y, pos_z,
    vel_x, vel_y, vel_z, acc_x, acc_y, acc_z, mass)

class GSimulation
{
…
private:
    sdlt::soa1d_container<Particle> _particles;
…
};
```

```
auto particles = _particles.access();
…
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        float distance, dx, dy, dz;
        float distanceSqr = 0.0;
        float distanceInv = 0.0;

        dx       =       particles[j].pos_x()       -
particles[i].pos_x();
       …
        distanceSqr = dx*dx + dy*dy + dz*dz +
softeningSquared;
        distanceInv = 1.0f / sqrtf(distanceSqr);

        particles[i].acc_x() += dx * G *
particles[j].mass() *
            distanceInv * distanceInv * distanceInv;
        particles[i].acc_y() += …
        particles[i].acc_z() += …
```

ACTIVITY 6

Purpose: Use SDLT library to change a AOS to SOA format, thus improving vectorization

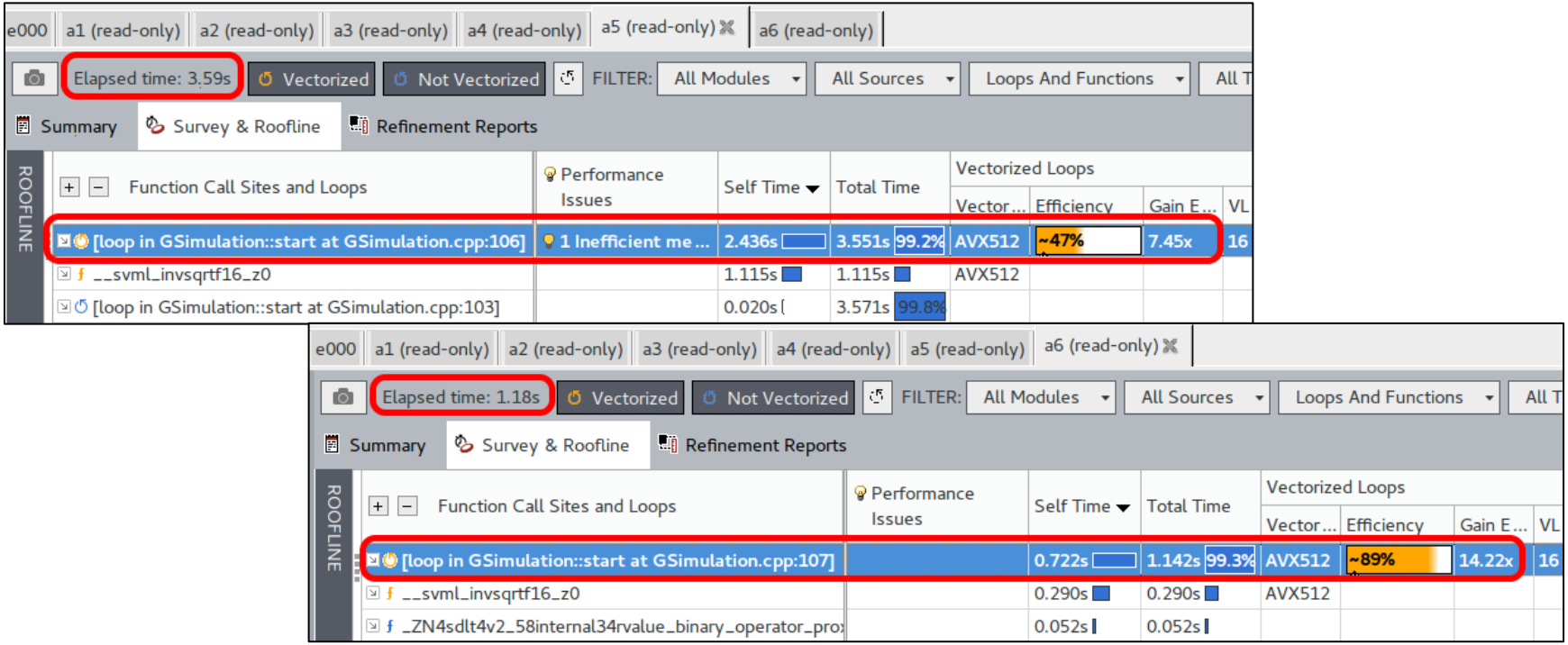
- Build version with optimized memory access pattern

```
$ cd ~/day1/lab4/ver3
```

```
$ make
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous results

Activity 6. Screenshots



Activity 7: Comparing roofline charts

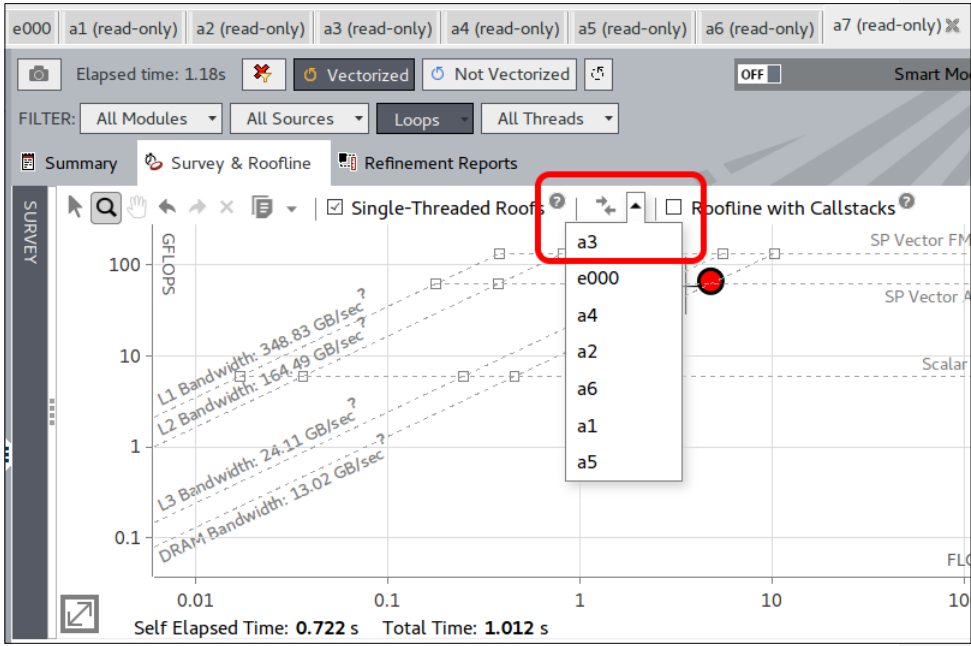
ACTIVITY 7

Purpose: Graph roofline chart for optimized version, and compare with initial chart

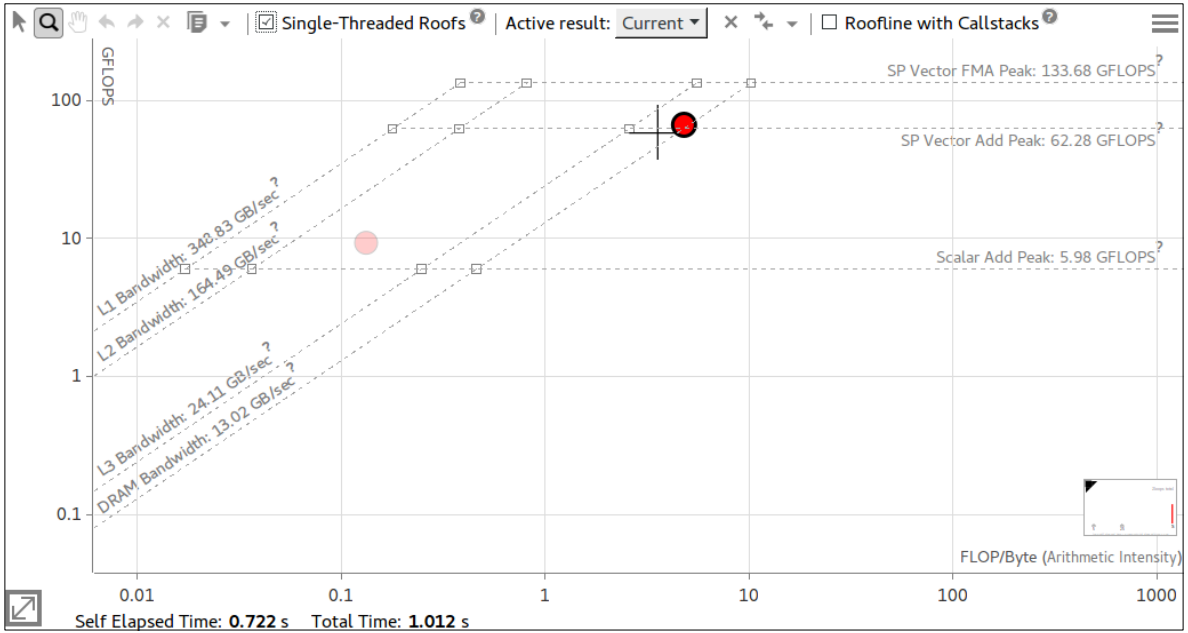
Press “Collect” button in “1.1 Find Trip Counts and FLOP” section

Create a snapshot

Compare with chart created in Activtiy 3



Activity 7. Screenshot



Activity 8: Adding threading

Activity 8

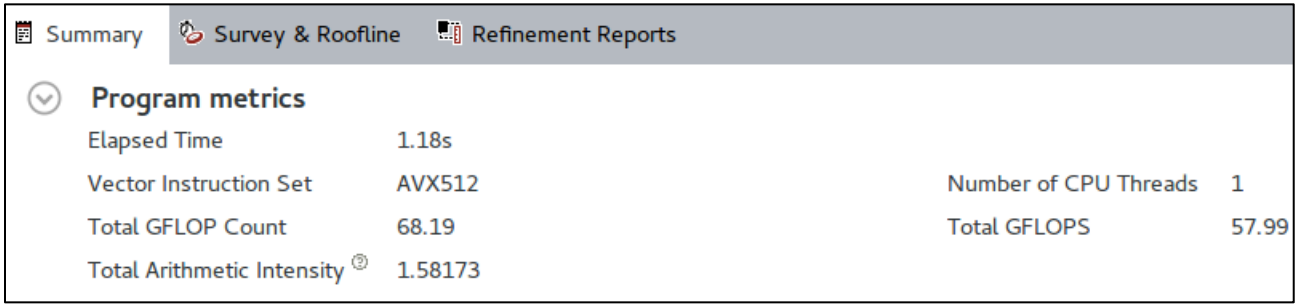
Purpose: Use OpenMP directives to enable threading parallelisation

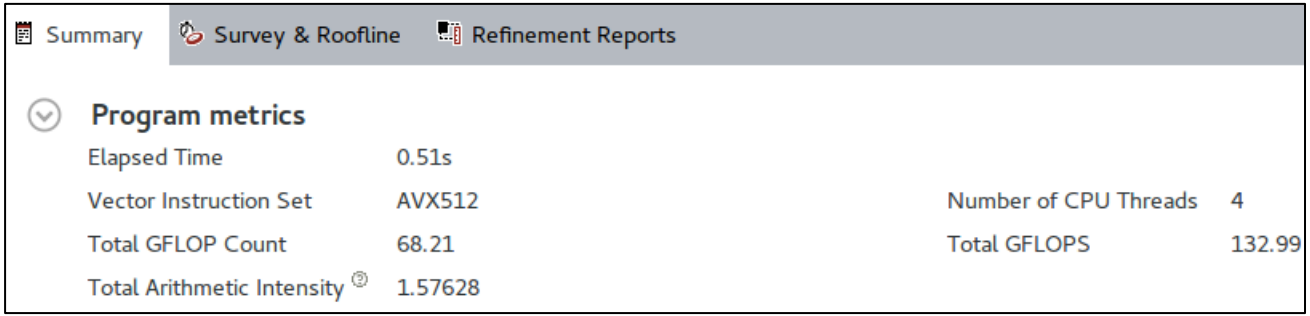
- Build threaded version

```
$ cd ~/day1/lab4/ver4
$ make
```

- Re-run Survey analysis
- Create a snapshot
- Compare with previous results

Activity 8. Screenshots





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.
-