

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 screenshot shows the Intel Advisor 2018 interface. At the top, there are filters for 'All Modules', 'All Sources', 'Loops', and 'All Threads'. Below the filters, there are tabs for 'Summary', 'Survey & Roofline', and 'Refinement Reports'. The main area displays a table of function call sites and loops. The table has columns for 'Function Call Sites and Loops', 'Performance Issues', 'Self Time', 'Total Time', 'Type', 'Why No Vectorization?', 'Vectorized Loops', and 'FLOPS'. The 'Vectorized Loops' column is further divided into 'Vect...', 'Efficiency', 'Gain...', 'VL (...)', and 'Com..'. The 'FLOPS' column is further divided into 'Self GFLOPS' and 'Self AI'. The table shows several loops, with one loop highlighted in orange, indicating 100% efficiency. The highlighted loop is '[loop in runOMPRawLoopsSomp\$parallel@64]' with a self time of 4.190s and a total time of 4.190s. It is a 'Vectorized+Thr...' type loop, and the 'Why No Vectorization?' column shows 'vector dependence...'. The 'Vectorized Loops' column shows 'AVX' and the 'Efficiency' column shows '100%'. The 'FLOPS' column shows 'Self GFLOPS' of 6.767I and 'Self AI' of 0.02486.

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops					FLOPS	
						Vect...	Efficiency	Gain...	VL (...)	Com..	Self GFLOPS	Self AI
[loop in runOMPRawLoopsSomp\$parallel_...	2 Assumed d...	15.484s	578.046s	Threaded (Op...	vector dependence...						2.235I	0.02459
[loop in runCRawLoops at runCRawLoops.c...	2 Assumed d...	11.766s	11.766s	Scalar	vector dependence...						0.995I	0.08333
[loop in runCForallLambdaLoops at runCFora...	2 Assumed d...	11.766s	11.766s	Scalar	vector dependence...						0.995I	0.08333
[loop in runCRawLoops at runCRawLoops.c...	2 Assumed d...	5.156s	5.156s	Scalar	vector dependence...						1.512I	0.11458
[loop in runCForallLambdaLoops at runCFora...	2 Assumed d...	5.125s	5.125s	Scalar	vector dependence...						1.521I	0.11458
[loop in runOMPRawLoopsSomp\$parallel@64	1 Ineffective ...	4.190s	4.190s	Vectorized+Thr...		AVX	100%	5.10x	4	5.28x	6.767I	0.02486
[loop in runOMPRawLoopsSomp\$parallel@...		3.768s	3.768s	Remainder+Th...							4.138I	0.02083
[loop in runOMPRawLoopsSomp\$parallel@...		0.406s	0.406s	Vectorized (Bo...		AVX			4	5.28x	27.368I	0.03125
[loop in runOMPRawLoopsSomp\$parallel@...		0.016s	0.016s	Peeled+Thread...							0.113I	0.02083

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, \dots, r_n and have masses m_1, \dots, 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

The image shows a composite of three screenshots from the Intel Advisor 2018 GUI. The leftmost screenshot displays the 'Welcome to Intel Advisor 2018' screen with the 'New Project...' button highlighted in a red box. The middle screenshot shows the 'Launch Application' dialog box, where the application path '/home/intel-workshop/day1/lab4/build/nbody.x' is entered and highlighted in a red box. The rightmost screenshot shows the 'Run Roofline' dialog box, where the '1. Survey Target' section is highlighted in a red box.

Activity 1. Screenshot

Summary Survey & Roofline Refinement Reports

Higher instruction set architecture (ISA) available
Your application was compiled using the SSE2 instruction set, which is lower than AVX512 - the highest ISA available on the target machine. To compile the application using the highest ISA available on the target machine: Use the `-xCORE-AVX512` or `-xHost` option. For compatibility with other AVX-512 processors use the `-xCOMMON-AVX512` option.
Do not show this message again. Use

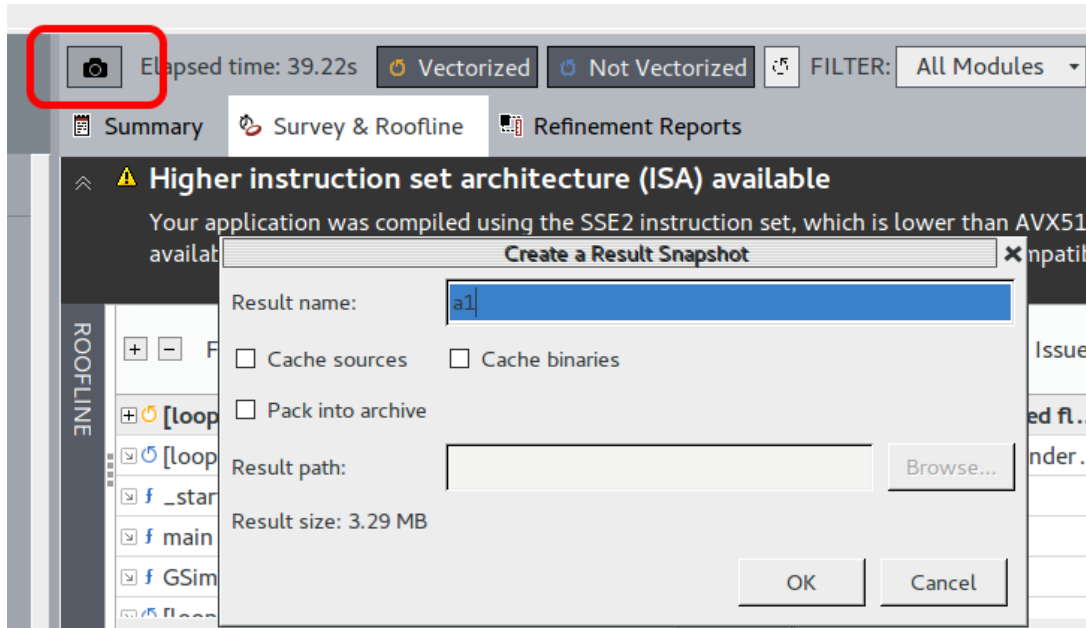
Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops	Efficiency
[loop in GSimulation::start at GSimulation.cpp:106]	4 Unoptimized fl...	39.124s	39.124s	Vectorized (Body)		SSE2	~90%
[loop in GSimulation::start at GSimulation.cpp:129]	1 Potential under...	0.008s	0.008s	Scalar	vectorization possible but ...		
f _start		0.000s	39.144s	Function			
f main		0.000s	39.144s	Function			
f GSimulation::start		0.000s	39.144s	Function			
f loop in GSimulation::start at GSimulation.cpp:106		0.000s	39.124s	Scalar	inner loops already use		

Source Top Down Code Analytics Assembly Recommendations Why No Vectorization?

File: GSimulation.cpp:106 GSimulation::start

Line	Source	Total Time	%	Loop/Func
99	const double t0 = time.start();			
100	for (int s = 1; s <= get_nsteps(); ++s)			
101	{			
102	ts0 += time.start();			
103	for (i = 0; i < n; i++)// update acceleration			
104	{			
105	float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;			
106	for (j = 0; j < n; j++)	0.400s		
107	{			
108	float dx, dy, dz;			
109	float distanceSqr = 0.;			
110	float distanceInv = 0.;			

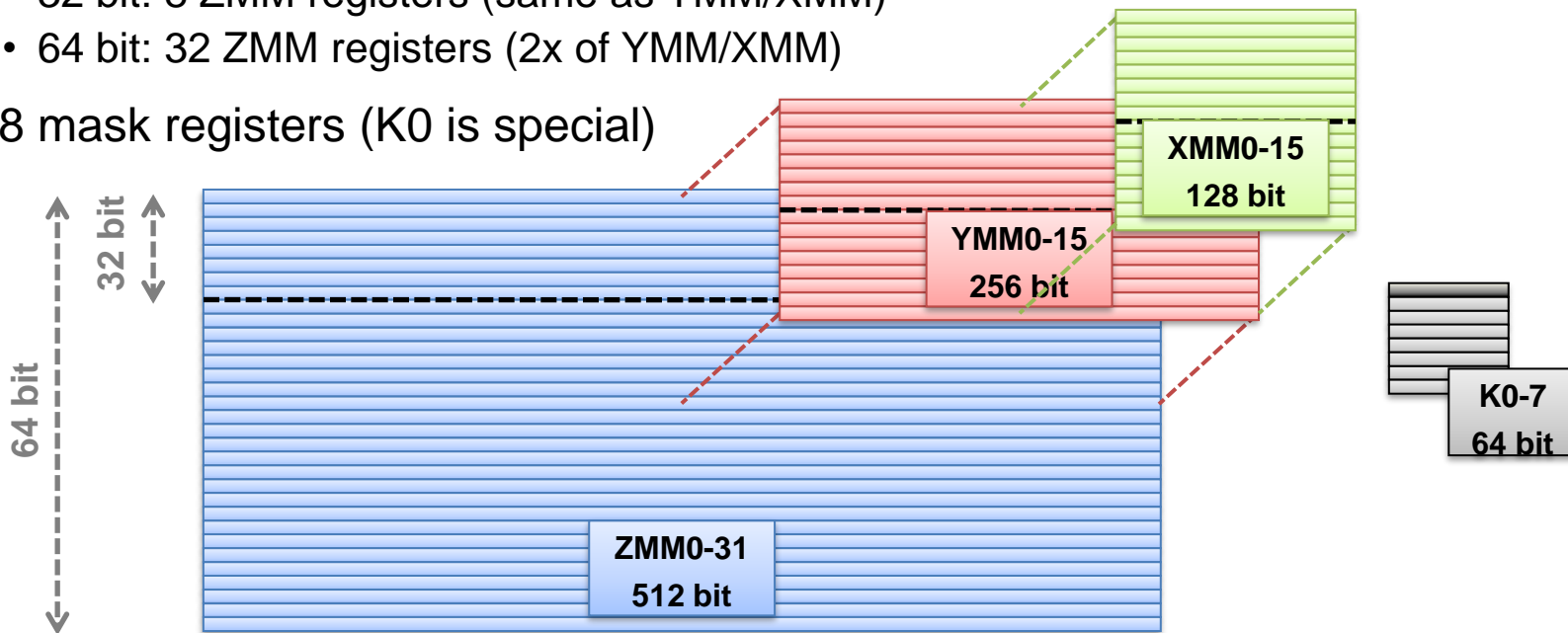
Create a snapshot



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

Elapsed time: 39.22s

Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA.

Function Call Sites and Loops	Performance Issues	Self Time	Total Time
[Loop in GSImulation::start at GSImulation.cpp:106]	4 Unoptimized fl...	39.124s	39.124s
[Loop in GSImulation::start at GSImulation.cpp:129]	1 Potential under...	0.008s	0.008s
f _start		0.000s	39.144s
f main			
f GSImulation::start			
[Loop in GSImulation::start at GSImulation.c...			
[Loop in GSImulation::start at GSImulation.c...			

Run Roofline

1. Survey Target

Mark Loops for Deeper Analysis
Select checkboxes in the Survey & Roofline tab to mark loops for other Advisor analyses.
-- There are no marked loops --

Elapsed time: 6.84s

Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

Function Call Sites and Loops	Performance Issues	Self Time	Total Time
[Loop in GSImulation::start at GSImulation.cpp:106]	2 Possible ineffic...	5.383s	6.789s 99.6%
f __svml_linvsqrtf16_z0		1.406s	1.406s
[Loop in GSImulation::start at GSImulation.cpp:103]	1 Data type conv...	0.029s	6.818s
f _start		0.000s	6.818s
f main		0.000s	6.818s
f GSImulation::start		0.000s	6.818s
[Loop in GSImulation::start at GSImulation.cpp:100]	1 Data type conv...	0.000s	6.818s

Run Roofline

1. Survey Target

Mark Loops for Deeper Analysis
Select checkboxes in the Survey &

Activity 3: Doing roofline analysis



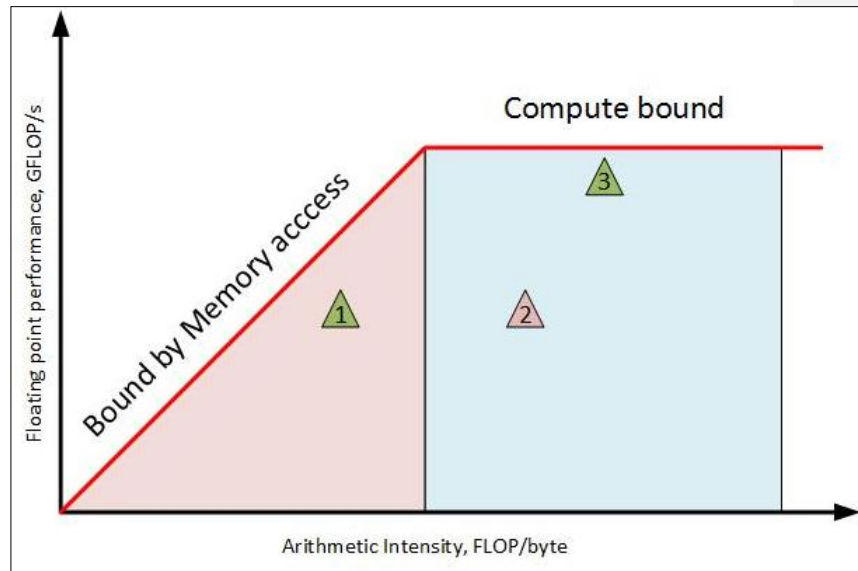
Roofline model

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

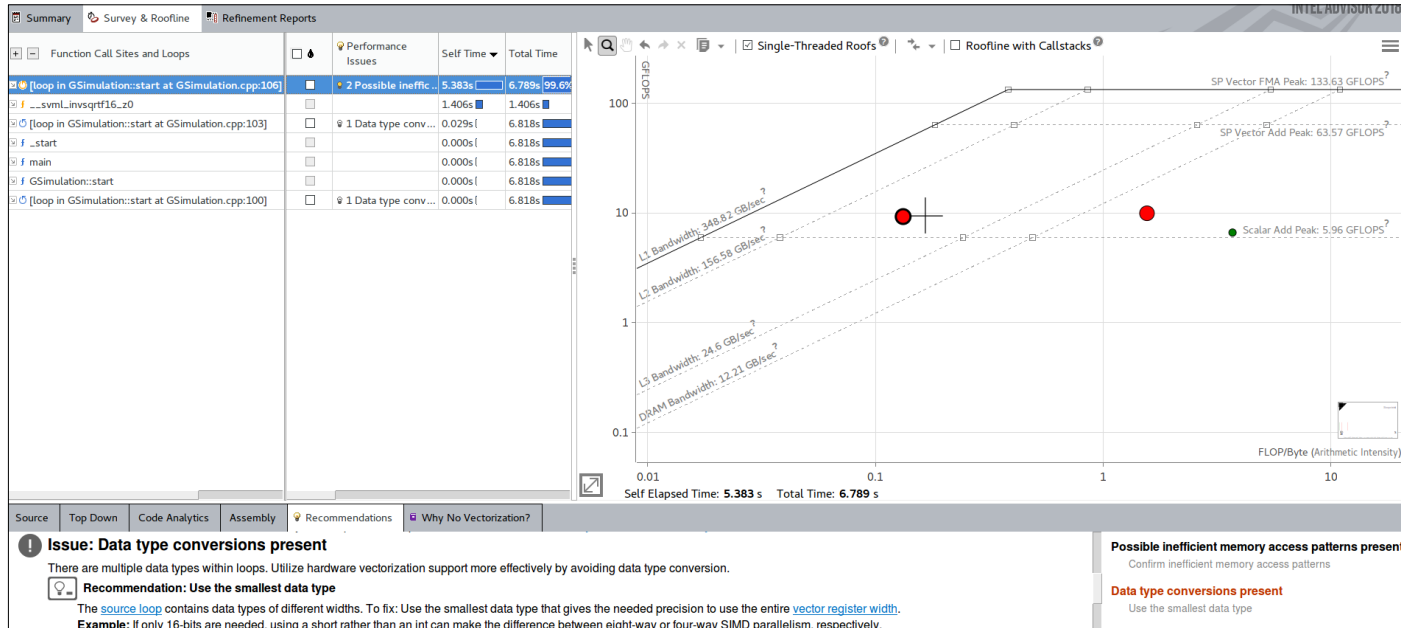
Press “Collect” button in “1.1 Find Trip Counts and FLOP” section

Create a snapshot

The screenshot displays the Intel Advisor interface. On the left, the 'Vectorization Workflow' is active, and the 'Threading Workflow' is also visible. The 'Batch mode' is turned OFF. Under the 'Run Roofline' section, the 'Collect' button is highlighted with a red box. Below this, the '1.1 Survey Target' section has its 'Collect' button also highlighted with a red box. In the 'Mark Loops for Deeper Analysis' section, the 'FLOP' checkbox is checked and highlighted with a red box. The right side of the interface shows the 'Roofline' tab, displaying a list of function call sites and loops, including '[loop in GSimulation::start at GSimulation.cpp:106]'. Below the roofline, the 'Source' tab is active, showing the source code for 'GSimulation.cpp:106 GSimulation::start' with line numbers 99 to 103.

```
99     const double t0 = time.start();
100     for (int s = 1; s <= get_nsteps(); ++s)
101     {
102         ts0 += time.start();
103         for (i = 0; i < n; i++)// update a
```

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

Elapsed time: 6.84s

Vectorized

Not Vectorized

FILTER: All Modules | All Sources | Loops And Functions

Summary | Survey & Roofline | Refinement Reports

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type
[loop in GSimulation::start at GSimulation.cpp:106]	2 Possible inefficiencies	5.383s	6.789s	99.6% Vectorized (Body)

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

All Advisor-detectable issues: [C++](#) | [Fortran](#)

Issue: Possible inefficient memory access patterns present
Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization.

Recommendation: Confirm inefficient memory access patterns
There is no confirmation inefficient memory access patterns are present. To confirm: Run a [Memory Access Patterns](#) analysis.

Issue: Data type conversions present
There are multiple data types within loops. Utilize hardware vectorization support for the smallest data type.

Recommendation: Use the smallest data type
The [source loop](#) contains data types of different widths. To fix: Use the smallest data type.

Elapsed time: 3.59s

Vectorized

Not Vectorized

FILTER: All Modules | All Sources | Loops And Functions

Summary | Survey & Roofline | Refinement Reports

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type
[loop in GSimulation::start at GSimulation.cpp:106]	1 Possible inefficiency	2.436s	3.551s	99.2% Vectorized (Body)

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

All Advisor-detectable issues: [C++](#) | [Fortran](#)

Issue: Possible inefficient memory access patterns present
Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization.

Recommendation: Confirm inefficient memory access patterns
There is no confirmation inefficient memory access patterns are present. To confirm: Run a [Memory Access Patterns](#) analysis.

Activity 5: Checking memory access patterns



Types OF MEMORY Access patterns

Unit-Stride access

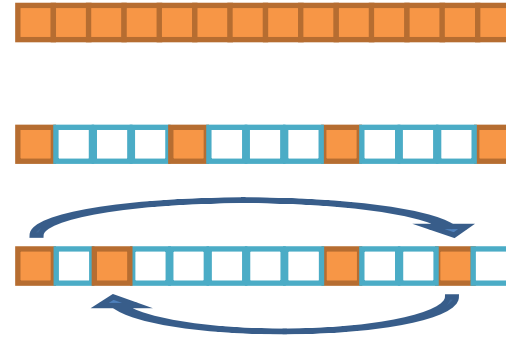
```
for (i=0; i<N; i++)  
  A[i] = C[i]*D[i]
```

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

The screenshot displays the Intel Advisor interface. On the left, the 'Threading Workflow' is active, and the '2.1 Check Memory Access Patterns' section is expanded, with its 'Collect' button highlighted by a red box. The 'Survey Target' section shows '1 Survey Target' and '1.1 Find Trip Counts and FLOP' with 'Trip Counts' and 'FLOP' checked. The 'Mark Loops for Deeper Analysis' section is also visible. On the right, the 'Survey & Roofline' tab is active, showing a list of loops. The first loop, '[loop in GSimulation::start at GSimulation.cpp:106]', is selected and has its checkbox checked, also highlighted by a red box. Below the roofline, the source code for 'GSimulation.cpp:106' is displayed, showing a nested loop structure. The code includes variables for time (ts0, ts1, t0), a loop over 's' (lines 100-101), and a nested loop over 'i' (lines 102-104) and 'j' (lines 105-106). The 'j' loop is highlighted in blue, corresponding to the selected loop in the roofline. The 'Recommendations' tab is partially visible at the bottom right.

Activity 5. Screenshot

The screenshot displays the Intel Advisor interface, specifically the 'Refinement Reports' tab. The top navigation bar includes 'Summary', 'Survey & Roofline', and 'Refinement Reports'. Below this, a table provides a summary of the report:

Site Location	Strides Distribution	Access Pattern	Performance Issues	Footprint Estimate	
[loop in start at GSimulation.cpp:1...	75% / 25% / 0%	Mixed strides	1 inefficient memory access patterns present	Max. Per-Instruction Addr. Range: 56KB	First Instance Site Footprint: 58KB

The code snippet shown is:

```
104 {
105     float acc_x = 0.f, acc_y = 0.f, acc_z = 0.f;
106     for (j = 0; j < n; j++)
107     {
108         float dx, dy, dz;
```

The 'Memory Access Patterns Report' section is expanded, showing a warning icon and the text: **Issue: Inefficient memory access patterns present**. Below this, a detailed explanation states: 'There is a high of percentage memory instructions with irregular (variable or random) stride accesses. Improve performance by investigating and handling accordingly.'

A recommendation icon is followed by the text: **Recommendation: Use Intel SDLT**. The explanation continues: 'The cost of rewriting code to organize data using SoA instead of AoS may outweigh the benefit. To fix: Use Intel SIMD Data Layout Templates (Intel SDLT), introduced in version 16.1 of the Intel compiler, to mitigate the cost. Intel SDLT is a C++11 template library that may reduce code rewrites to just a few lines. Example: Using SDLT instead of STL containers may improve the memory access pattern for more efficient vector processing.'

The 'Original code:' section shows:

```
...
struct kValues {
    float Kx;
    float Ky;
    float Kz;
    float PhiMag;
};
std::vector<kValues> dataset(count);
...
```

The 'Revised code:' section shows:

```
#include <sdl/sdlt.h>

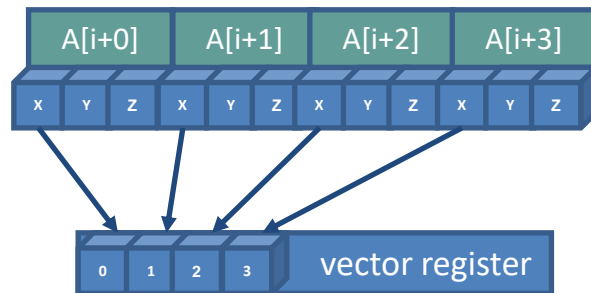
struct kValues {
    float Kx;
```

Activity 6: Using SDLT

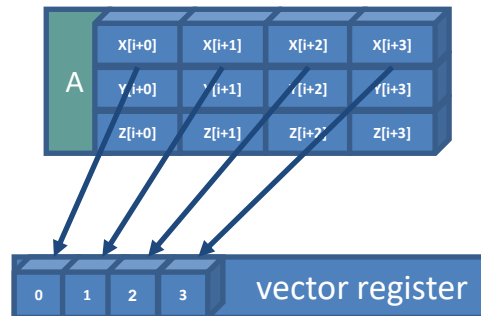


SIMD Is Effective With Unit Stride Access

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

e000 a1 (read-only) a2 (read-only) a3 (read-only) a4 (read-only) a5 (read-only) ✖ a6 (read-only)

Elapsed time: 3.59s Vectorized Not Vectorized FILTER: All Modules All Sources Loops And Functions All T

Summary Survey & Roofline Refinement Reports

ROOFLINE	Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Vectorized Loops				
					Vector...	Efficiency	Gain E...	VL	
	[loop in GSimulation::start at GSimulation.cpp:106]	1 Inefficient me ...	2.436s	3.551s	99.2%	AVX512	~47%	7.45x	16
	f __svml_invsqrtf16_z0		1.115s	1.115s		AVX512			
	[loop in GSimulation::start at GSimulation.cpp:103]		0.020s	3.571s	99.8%				

e000 a1 (read-only) a2 (read-only) a3 (read-only) a4 (read-only) a5 (read-only) a6 (read-only) ✖

Elapsed time: 1.18s Vectorized Not Vectorized FILTER: All Modules All Sources Loops And Functions All T

Summary Survey & Roofline Refinement Reports

ROOFLINE	Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Vectorized Loops				
					Vector...	Efficiency	Gain E...	VL	
	[loop in GSimulation::start at GSimulation.cpp:107]		0.722s	1.142s	99.3%	AVX512	~89%	14.22x	16
	f __svml_invsqrtf16_z0		0.290s	0.290s		AVX512			
	f _ZN4sdl4v2_58internal34value_binary_operator_pro		0.052s	0.052s					

Activity 7: Comparing roofline charts



ACTIVITY 7

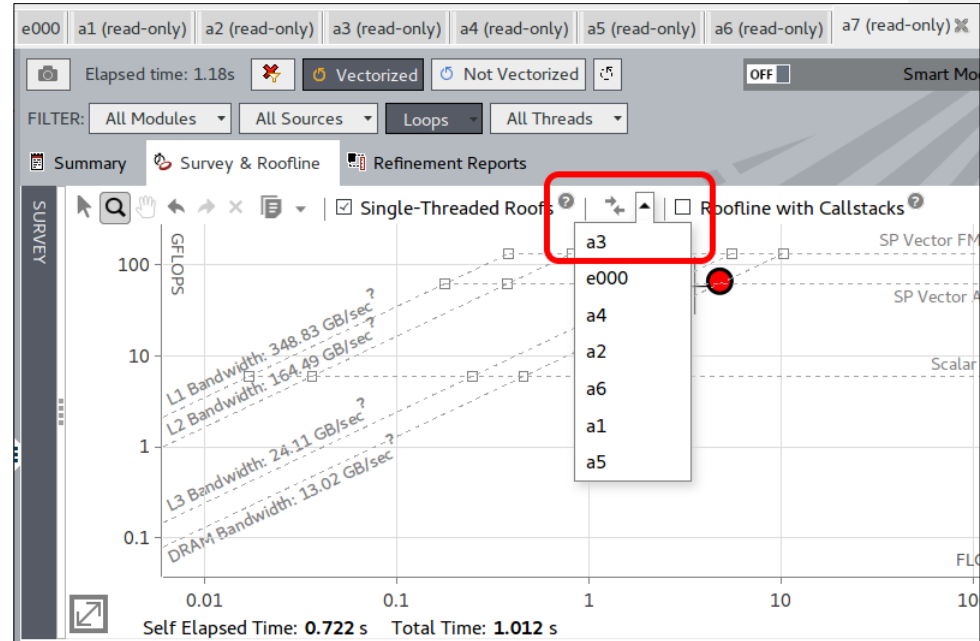
Purpose: Graph roofline chart for optimized version, and compare with initial chart

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

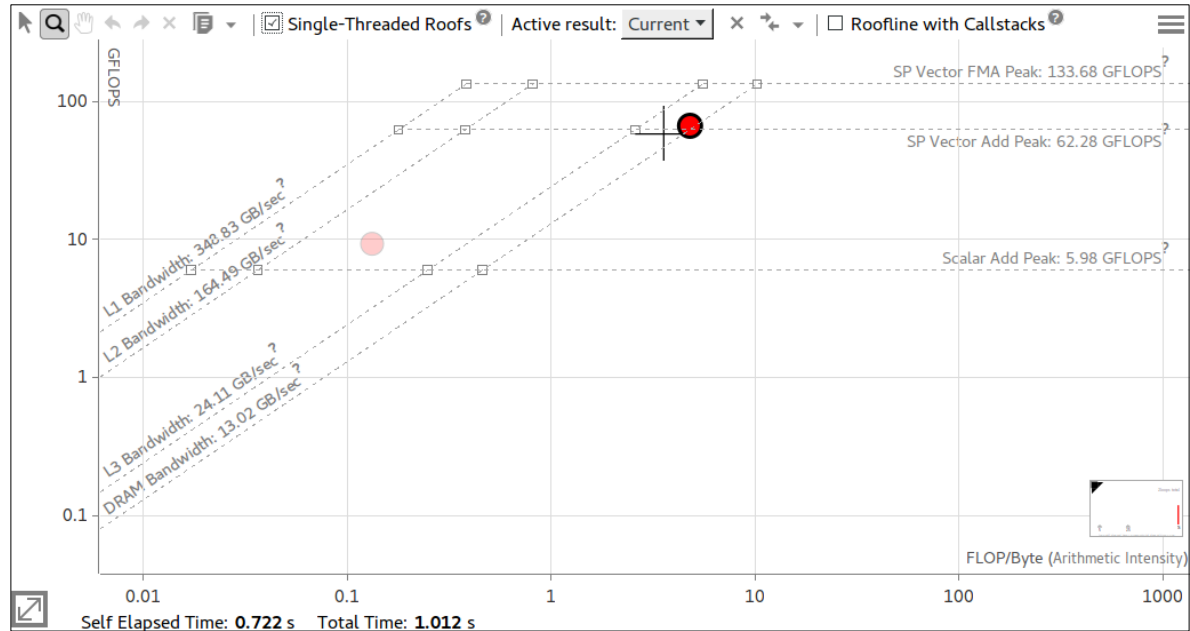
Create a snapshot

Compare with chart created in Activity

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

Summary	Survey & Roofline	Refinement Reports	
Program metrics			
Elapsed Time	1.18s		
Vector Instruction Set	AVX512	Number of CPU Threads	1
Total GFLOP Count	68.19	Total GFLOPS	57.99
Total Arithmetic Intensity [®]	1.58173		

Summary	Survey & Roofline	Refinement Reports	
Program metrics			
Elapsed Time	0.51s		
Vector Instruction Set	AVX512	Number of CPU Threads	4
Total GFLOP Count	68.21	Total GFLOPS	132.99
Total Arithmetic Intensity [®]	1.57628		



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.
- Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.