Perfilamento com Intel Advisor e Vtune

Silvio Stanzani , Raphael Cóbe , Rogério Iope, Jefferson Fialho

Núcleo de Computação Científica – UNESP

ERAD-SP 2019

Agenda

- Parallel Architectures
- Roofline Model
- Optimization Workflow
- Intel Advisor
- Intel Vtune

Parallel Architectures

- Heterogeneous computational systems:
  - Scalar and Vector Instructions

Vector Instructions (SIMD)

A7	A6	A5	A4	A3	A2	A1	A0
			+				
B7	B6	B5	B4	B3	B2	B1	B0
			=				
A7+B7	A6+B6	A5+B5	A4+B4	A3+B3	A2+B2	A1+B1	A0+B0

Scalar Instructions

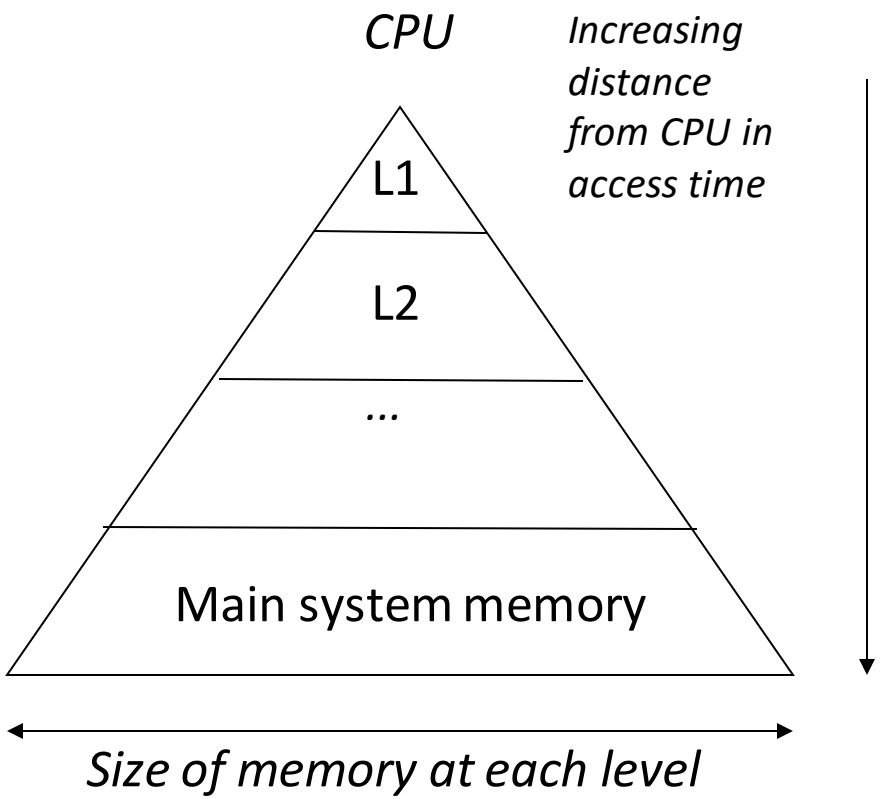
A
+
B
=
A+B

  - Multi-level memory
    - RAM Memory;
    - Multi-level Cache.

Processor 1			Processor 2		
Core 1	Core 2	Core N	Core 1	Core 2	Core N
L1	L1	L1	L1	L1	L1
L2	L2	L2	L2	L2	L2
L3			L3		
RAM					

Parallel Architecture

- Temporal locality: if an item was referenced, it will be referenced again soon (e.g. cyclical execution in loops);
- Spatial locality: if an item was referenced, items close to it will be referenced too (the very nature of every program – serial stream of instructions)
- Stride: Step size between consecutive access of array elements;



Parallel Architecture

- AoS vs SoA (Array of Structures vs Structure of Arrays)

```
// Array of Structures (AoS)
struct coordinate {
    float x, y, z;
} crd[N];
    …
for (int i = 0; i < N; i++)
    … = … f(crd[i].x, crd[i],y, crd[i].z);
```

Non-Unit Stride


```
// Structure of Arrays (SoA)
struct coordinate {
    float x[N], y[N], z[N];
} crd;
    …
for (int i = 0; i < N; i++)
    … = … f(crd.x[i], crd.y[i],
crd.z[i]);
```

Unit Stride - Sequential Access

Parallel Architecture

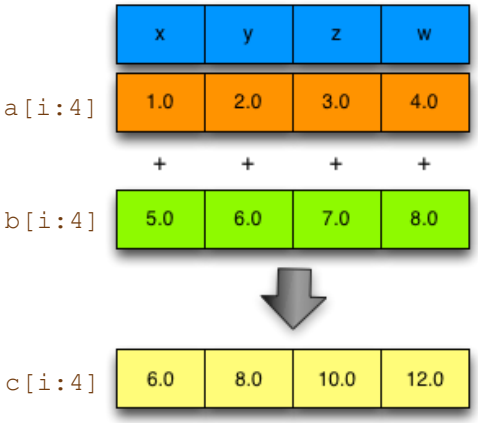
- Vectorization
  - Loading data into cache accordingly;
  - Store elements on SIMD registers or vectors;
  - Apply the same operation to a set of Data at the same time;
  - Iterations need to be independent;
  - Usually on inner loops.

Scalar loop

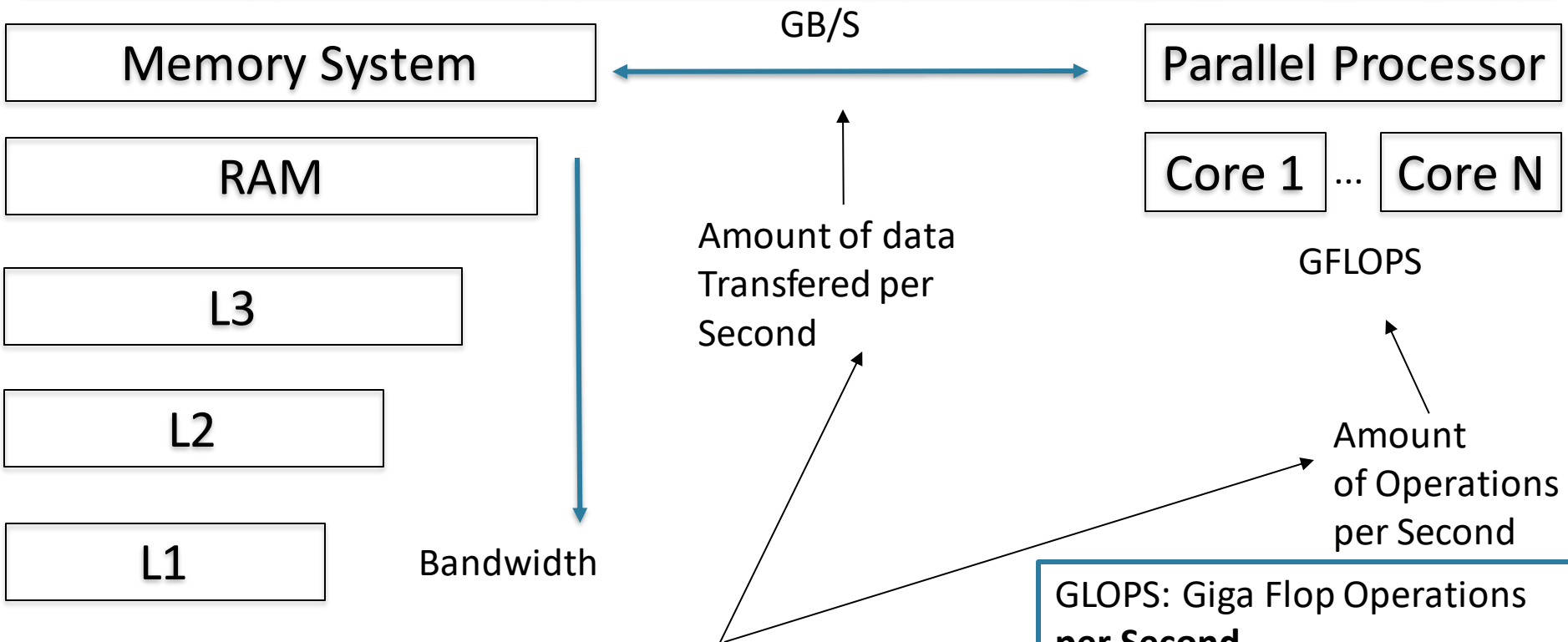
```
for (int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
```

SIMD loop (4 elements)

```
for (int i = 0; i < N; i += 4)
    c[i:4] = a[i:4] + b[i:4];
```



Performance Model



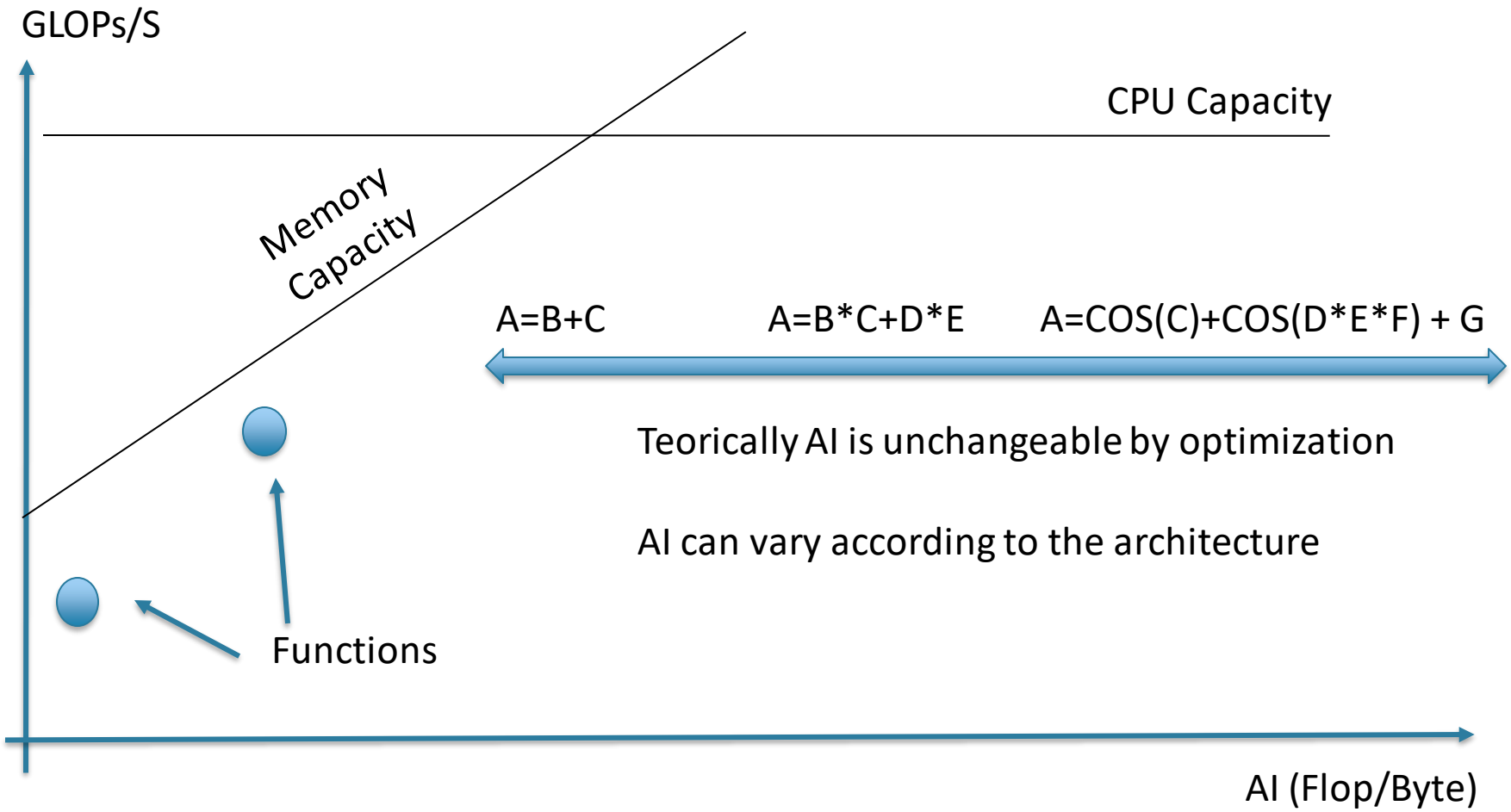
Balance between processing and transfer:

- Higher GFLOPS Demands more GB/S
- Lower GLOPS limits potential GB/s

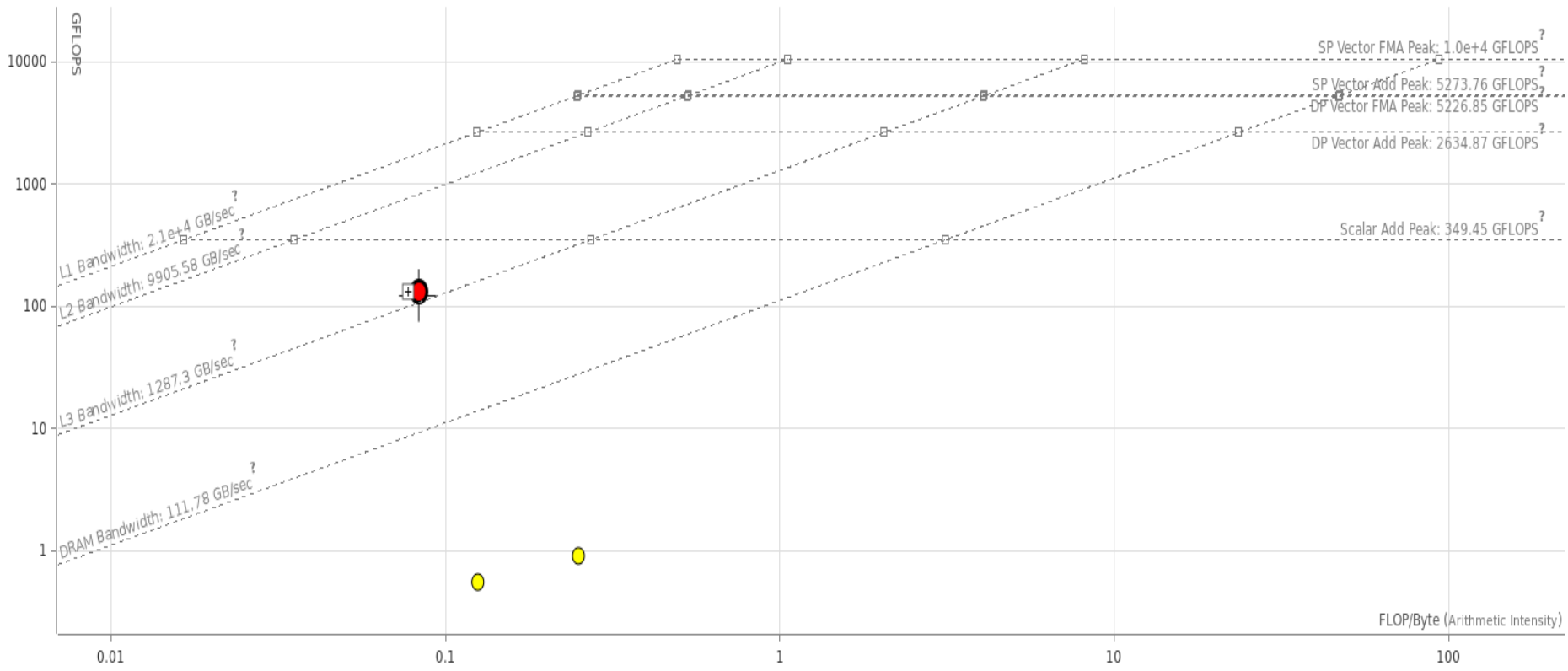
GLOPS: Giga Flop Operations **per Second**
GLOPs: Giga Flop Operations

How to Measure such Balance?
Arithmetic Intensity (AI): Ratio between work performed and data transfered: Flop/Byte.

Roofline Model



Roofline Model

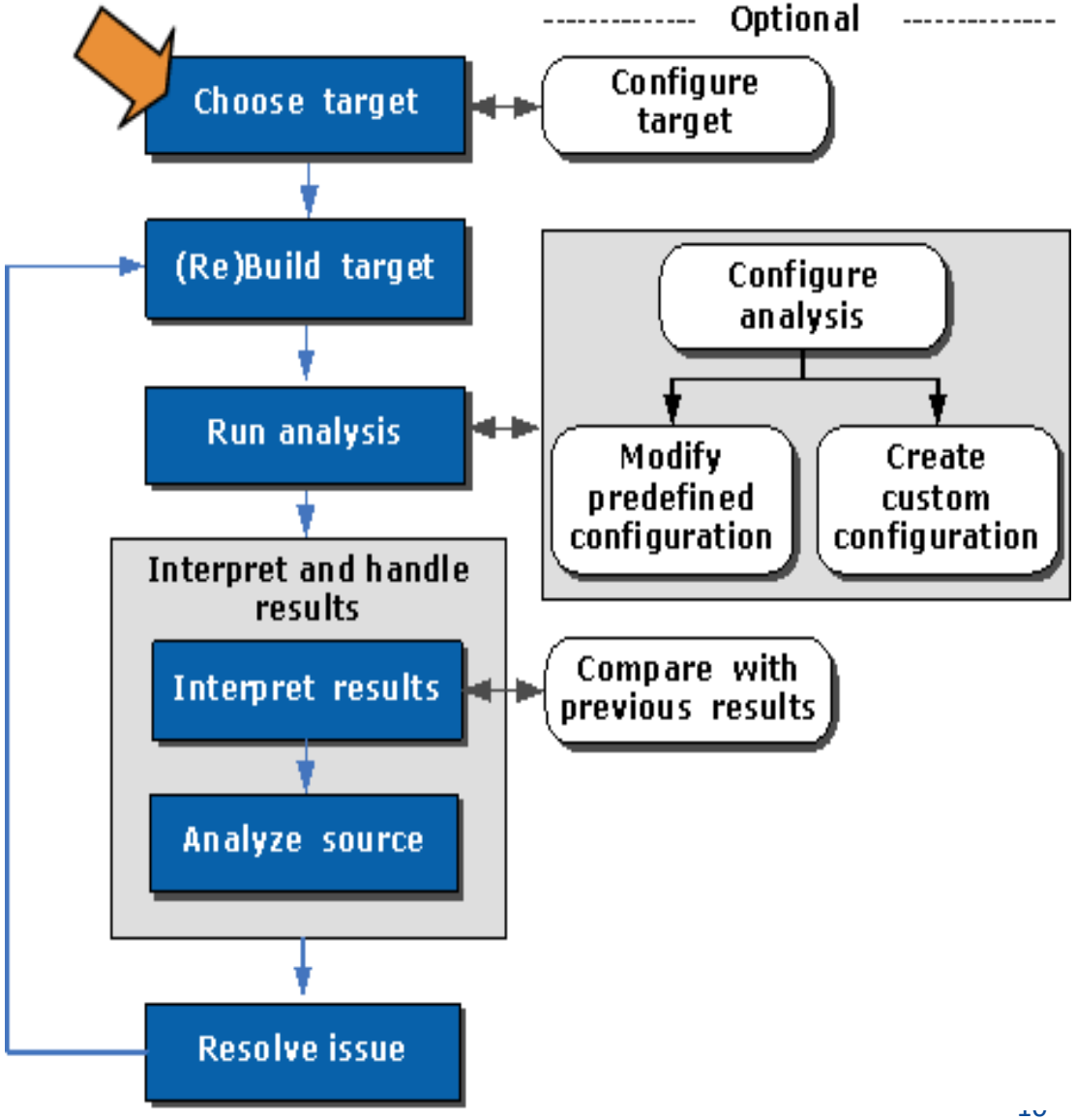


Optimization Workflow

Profiling Techniques include:

- Instrument Code
- Measure Hardware Events
- Performance Monitoring Units

Iterative process to improve Performance



Intel Advisor

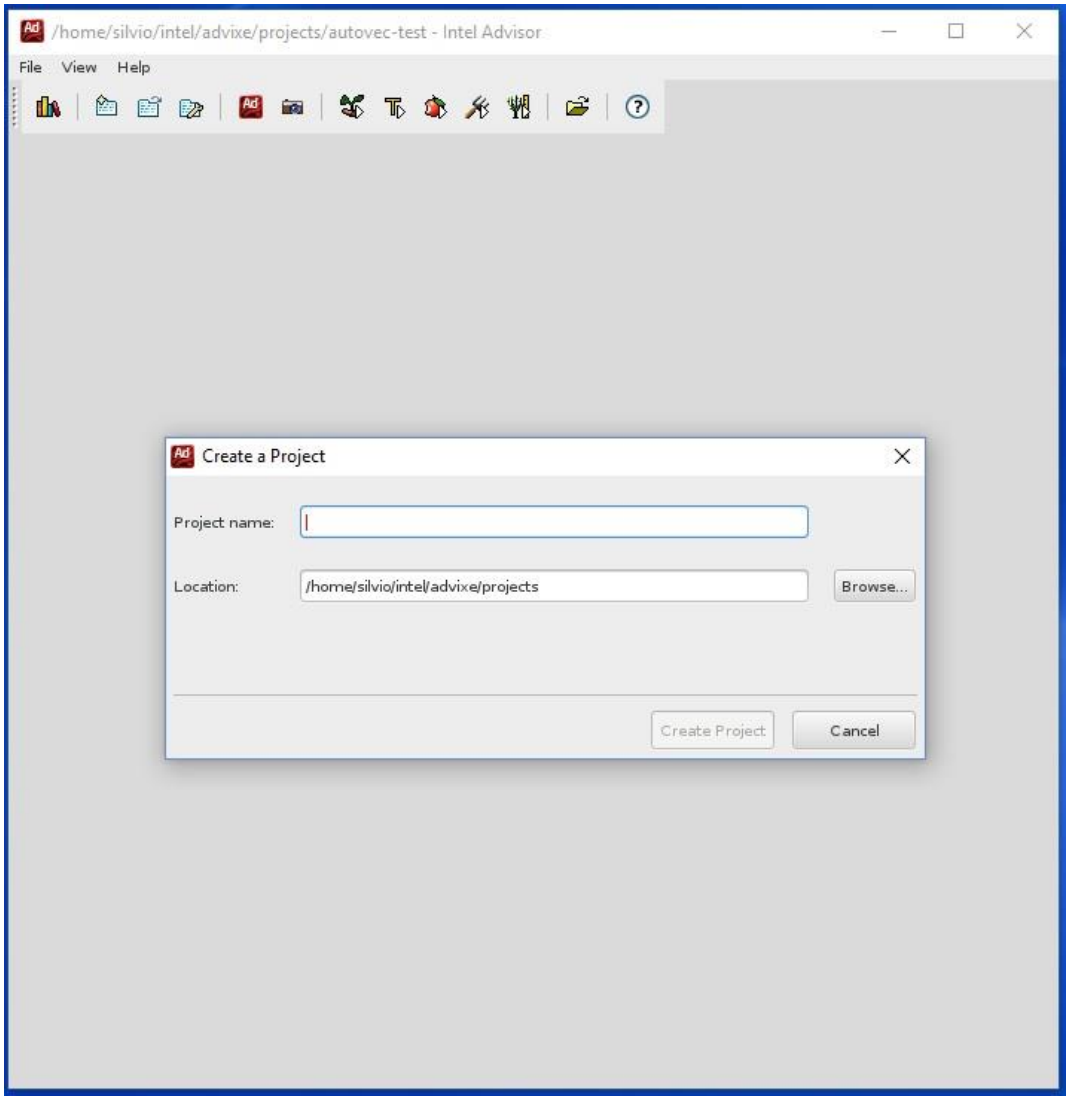
- Evaluate multi-threading parallelization

- Intel® Advisor XE
  - ❑ Performance modeling using several frameworks for multi-threading in processors and co-processors:
    - OpenMP, Intel® Cilk ™ Plus, Intel® Threading Bulding Block
    - C, C++, Fortran (OpenMP only) and C# (Microsoft TPL)
  - ❑ Identify parallel opportunities
    - Detailed information about vectorization;
    - Check loop dependencies;
    - Memory Access Patterns
  - ❑ Scalability prediction: amount of threads/performance gains
  - ❑ Correctness (deadlocks, race conditions)

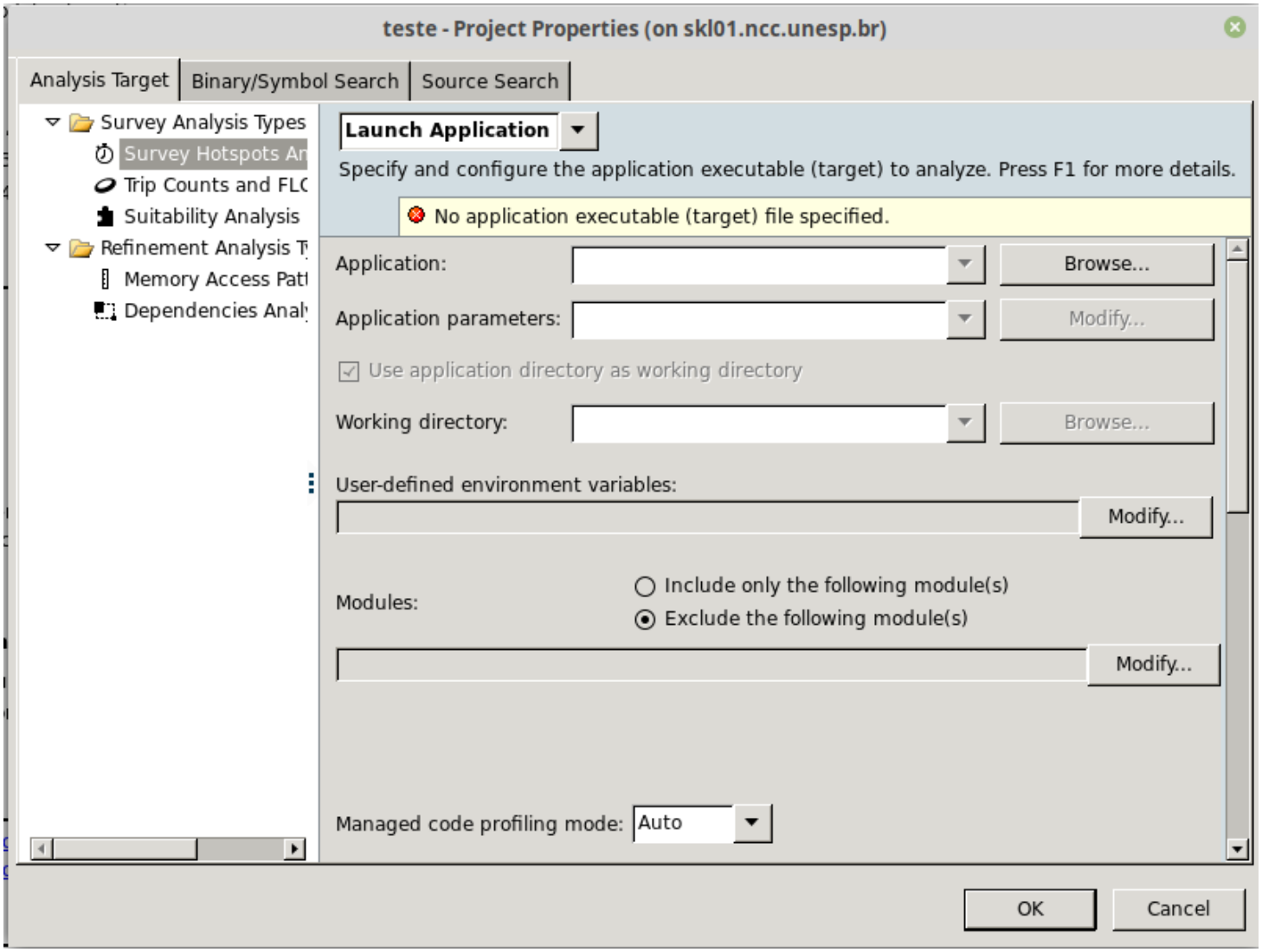
Intel Advisor

- Roofline Model
  - Obtain the Roofline of an application
- Survey Target;
  - Vectorization of loops: detailed information about vectorization;
  - Total Time: elapsed time on each loop considering the time involved in internal loops;
  - Self Time: elapsed time on each loop not considering the time involved in internal loops;
- Find Trip Counts;
  - Analysis to identify how many time particular loops run;
- Check Dependencies;
  - Analysis it there are loop-carried dependencies;
- Check Memory Access Patterns.
  - Analysis to identify how your code is iterating with memory.

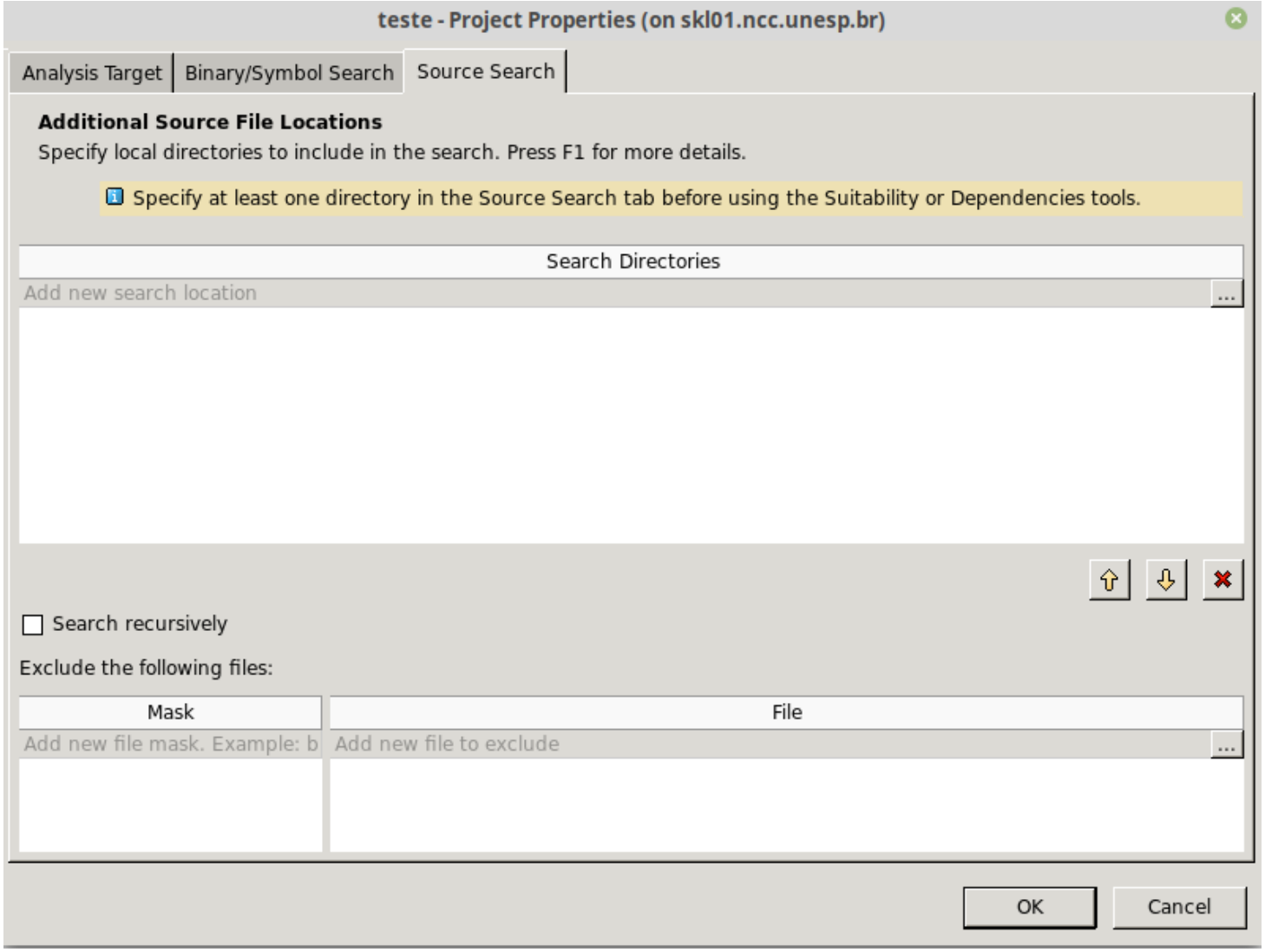
Advisor – New Project



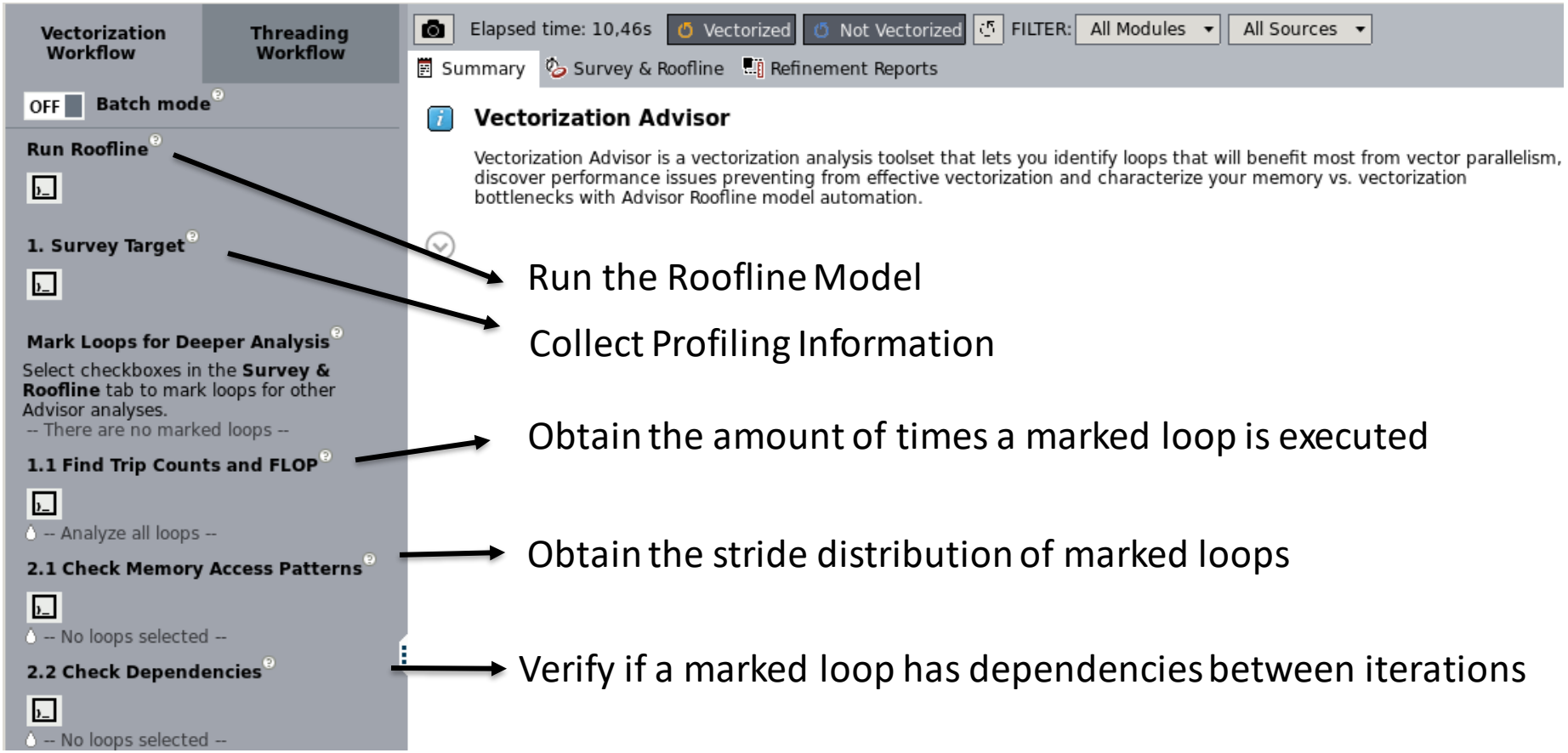
Advisor – Application and Parameters



Advisor – Source



Advisor - Analysis



Advisor – Survey Target

Summary | Survey Report | Refinement Reports | Annotation Report

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Vectorized Loops: Vec...	Efficiency	Gai...	VL (Vector Length)	Compiler Estimated Gain	Instruction Set Analysis: Traits
[loop in main at vec.c:50]	2 Inefficient memory access patterns present	0.730s	0.730s	Vectorized (Body)	AVX2		2.20x	16	2.20x	Extracts; Type Conversions
[loop in main at vec.c:63]	2 Data type conversions present	0.680s	0.680s	Vectorized Versions	AVX	71%	5.71x	2; 8	<5.71x	Extracts; Inserts; Type Conversions
[loop in main at vec.c:55]	1 Data type conversions present	0.490s	0.490s	Vectorized (Body)	AVX2	72%	5.79x	8	5.79x	Extracts; Inserts; Type Conversions

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: vec.c:50 main

Line	Source	Total Time	%	Loop Time	%	Tra
36	randV=0;					
37						
38	a = (double*) mm malloc(msize * msize *sizeof(double),64);					
39	b = (double*) mm malloc(msize * msize *sizeof(double),64);					
40	c = (double*) mm malloc(msize * msize * sizeof(double),64);					
41						
42	srand(time(NULL));					
43	randV=rand();					
44	randV=randV*0.11;					
45	printf("randV %f\n", randV);					
46						
47						
48	for(i=0; i<10000; i++) {			1,899.980m		
49						
50	for(i=0; i<40000; i++) {			729.970ms		
51	aosobj[i].x=i + randV;	229.961ms				Extracts; Type Conversions
52	aosobj[i].y=i*i+ randV;	290.000ms				Extracts; Type Conversions
53	aosobj[i].z=i+i+ randV;	210.009ms				Extracts; Type Conversions
54	}					
55	for(i=0; i<40000; i++) {	30.003ms		490.029ms		
56	soaobj.x[i]=i+i+ randV;	129.997ms				Extracts; Inserts; Type Conversions
57	soaobj.y[i]=i-i+ randV;	40.001ms				Type Conversions
58	soaobj.z[i]=i*i+ randV;	290.027ms				Extracts; Inserts; Type Conversions
59	}					
60	randV=rand();					Type Conversions
61	randV=randV*0.11;					
62						
63	for(i=0; i<40000; i++) {	9.999ms		679.985ms		
64	aosobj[i].x= aosobj[i].y+ aosobj[i].z + randV;	489.987ms				Extracts; Inserts; Type Conversions
65	}					
66						
67	for(i=0; i<40000; i++) {					
68						
69	soaobj.x[i]= soaobj.y[i]+ soaobj.z[i] + randV;	179.999ms				Extracts; Inserts; Type Conversions
70	}					
71	}					

Advisor – Survey Target

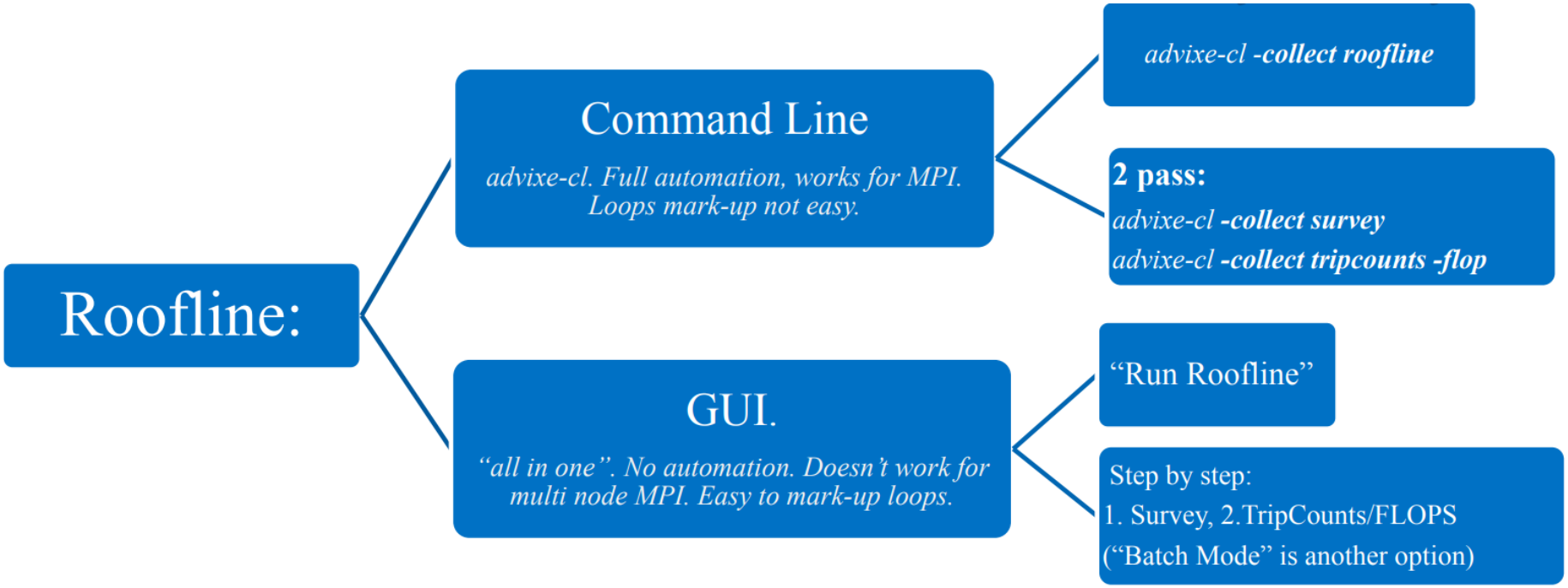
Instruction Set Analysis

Traits	Data Types	Number of Vector Registers	Vector Widths	Instruction Sets
Extracts; Type Conversions	Float32; Float64; Int32; UIn...	15	128/256	AVX; AVX2
Extracts; Inserts; Type Conversions	Float32; Float64	14; 15	128; 256	AVX
Extracts; Inserts; Type Conversions	Float32; Float64; Int32; UIn...	16	256	AVX; AVX2

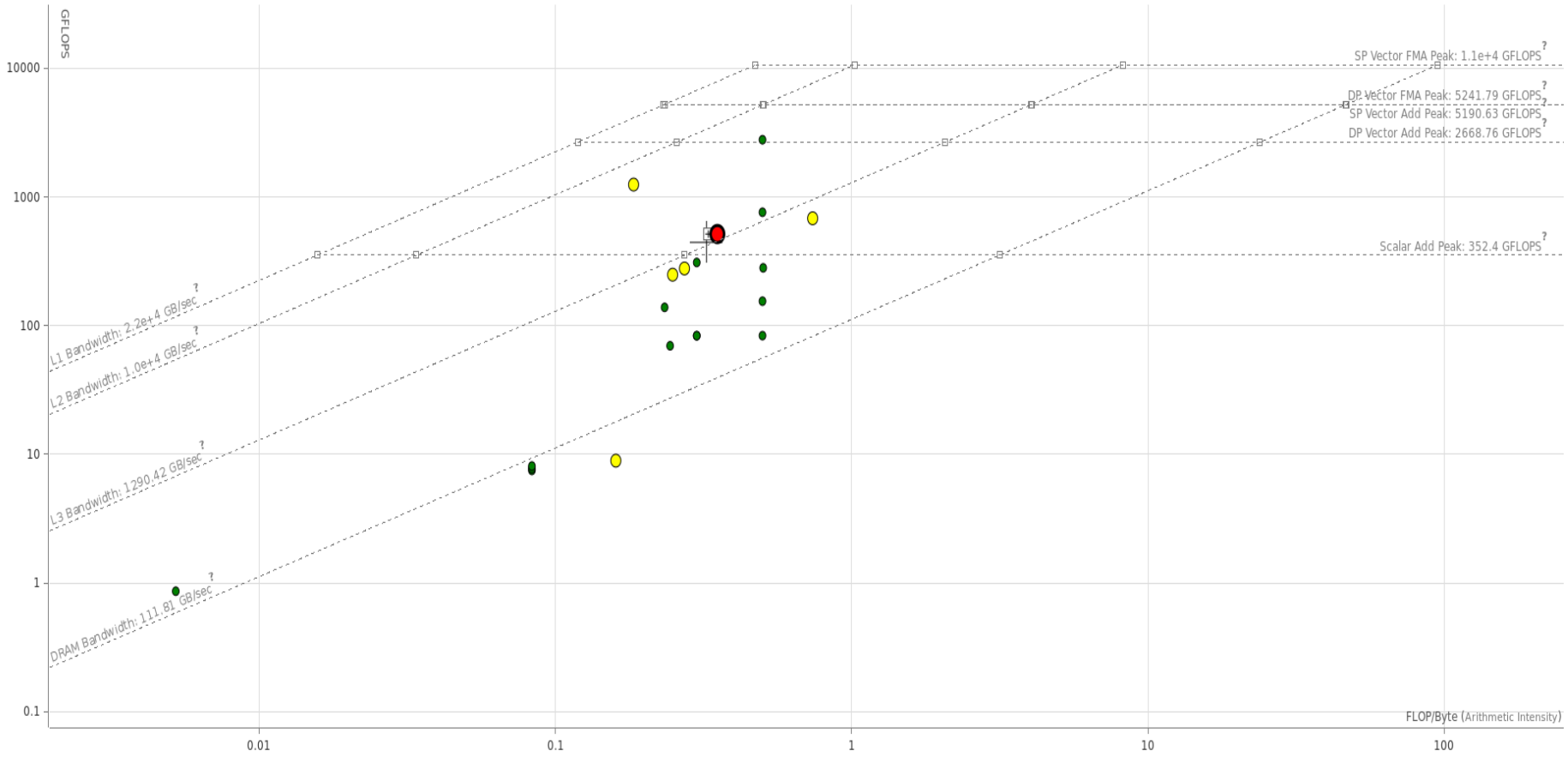
Advanced

Transformations	Unroll Factor	Vectorization Details	Optimization Details	Location
				vec.c:50
Fused; Unrolled	4		LOOP WAS DISTRIBUTED, CHUNK 1; LOOP WAS DISTRIBUTED, C...	vec.c:63
				vec.c:55

Roofline no Advisor



Roofline Example



Advisor – Memory Access Patterns

Vectorization Workflow | Threading Workflow

OFF Batch mode

1. Survey Target

Collect

1.1 Find Trip Counts

Collect

Mark Loops for Deeper Analysis

Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.

4 loops are marked

2.1 Check Dependencies

Collect

2.2 Check Memory Access Patterns

Collect

Check memory access patterns in your application

Elapsed time: 1.90s | Vectorized | Not Vectorized | FILTER: All Modules

Summary | Survey Report | Refinement Reports | Annotation Report

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in main at vec.c:50]	No information available	0% / 100% / 0%	All const strides	loop_site_7
[loop in main at vec.c:55]	No information available	100% / 0% / 0%	All unit strides	loop_site_4
[loop in main at vec.c:63]	No information available	0% / 100% / 0%	All const strides	loop_site_8
[loop in main at vec.c:63]	No information available	0% / 100% / 0%	All const strides	loop_site_9

Memory Access Patterns Report | Dependencies Report | Recommendations

ID	Stride	Type	Source	Site Name	Nested Func
P1	48	Constant stride	vec.c:51	loop_site_7	

```
49
50  for(i=0; i<40000; i++) {
51    aosobj[i].x=i + randV;
52    aosobj[i].y=i*i+ randV;
53    aosobj[i].z=i+i+ randV;
```

P2	48	Constant stride	vec.c:52	loop_site_7	

```
50  for(i=0; i<40000; i++) {
51    aosobj[i].x=i + randV;
52    aosobj[i].y=i*i+ randV;
53    aosobj[i].z=i+i+ randV;
54  }
```

P3	48	Constant stride	vec.c:53	loop_site_7	

```
51    aosobj[i].x=i + randV;
52    aosobj[i].y=i*i+ randV;
53    aosobj[i].z=i+i+ randV;
54  }
55  for(i=0; i<40000; i++) {
```

P6		Parallel site information	vec.c:50	loop_site_7	

```
48  for(j=0; j<10000; j++) {
49
50  for(i=0; i<40000; i++) {
51    aosobj[i].x=i + randV;
52    aosobj[i].y=i*i+ randV;
```

Random-Stride

Constant-Stride

Unit-Stride

“Top-Down” Methodology Definition

- hierarchical organization of event-based metrics that identifies the dominant performance bottlenecks in an application;
- Such methodology aims at show, on average, how well the CPU’s pipeline(s) were being utilized while running an application;
- Several hardware events are monitored according to the micro architecture;
- The results of this monitoring guides performance optimization .

Intel® VTune™ Amplifier XE

- **Compute-Intensive Application Analysis**
  - HPC Performance Characterization
- **Algorithm Analysis:**
  - Basic Hotspots
  - Advanced Hotspots
  - Concurrency
  - Locks & Waits
  - Memory Consumption

Views:
- Summary
- Bottom-up
- Top-down tree
- Tasks and frames
- Caller/Callee
- Source code
- Assembly
- Thread timeline
- Filters

View points:
- Hardware Event Counts
- Hardware Event Sample
- Hardware Issues
- Hotspots
- General Exploration
- Task Time
- Outros...

Intel® VTune™ Amplifier XE

- **Platform Analysis**
  - CPU/GPU Concurrency
  - System Overview
  - GPU Hotspots
  - GPU In-kernel Profiling
  - Disk Input and Output
  - Graphics Rendering (preview)
  - CPU/FPGA Interaction (preview)
- **Microarchitecture Analysis**
  - General Exploration
  - Memory Access
  - TSX Exploration
  - TSX Hotspots
  - SGX Hotspots

SGX Intel® Software Guard Extensions

TSX Intel® Transactional Synchronization Extensions

Intel® VTune™ Amplifier XE

Projects

Analysis

Results

Project Navigator

/home/silvio/i...

- matrix
  - r003macc
  - r004macc
- nbody
- option-price
- zmatmul

r003macc

Memory Access General Exploration viewpoint (change)

INTEL VTUNE AMPLIFIER 2018

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Event Count | Platform

Elapsed Time: 2.205s

Clockticks:	492,714,600,000	
Instructions Retired:	5,294,100,000	
CPI Rate:	93.069	
Front-End Bound:	2.5%	of Pipeline Slots
Bad Speculation:	0.0%	of Pipeline Slots
Back-End Bound:	94.9%	of Pipeline Slots
Memory Bound:	92.9%	of Pipeline Slots
L1 Bound:	3.0%	of Clockticks
L2 Bound:	N/A*	of Clockticks
L3 Bound:	32.6%	of Clockticks
DRAM Bound:	N/A*	of Clockticks
Store Bound:	0.0%	of Clockticks
Core Bound:	2.0%	of Pipeline Slots
Retiring:	2.6%	of Pipeline Slots
Total Thread Count:	98	
Paused Time:	0s	

*N/A is applied to metrics with undefined value. There is no data to calculate the metric.

Intel® VTune™ Amplifier XE

Project Navigator

/home/silvio/i...

- matrix
  - r003macc
  - r004macc
  - r005macc
- nbody
- option-price
- zmatmul

r005macc

Memory Access General Exploration viewpoint (change)

INTEL VTUNE AMPLIFIER 2018

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Event Count | Platform

Grouping: Function / Call Stack

Function / Call Stack	red	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound	Retiring	Module	
native_queued_spin_lock_slowpath	,000	59.928	1.3%	0.1%	96.6%	2.0%	vmlinux	native_queued_spin
_INTERNAL_25_____src_kmp_barrie	,000	79.593	3.2%	1.0%	84.4%	11.4%	libiomp5.so	void _INTERNAL_25
multiply3ompparallel_for@207	,000	7.487	1.6%	0.6%	83.3%	14.5%	matrixv3.icc	multiply3omppara
_INTERNAL_25_____src_kmp_barrie	,000	113.939	3.6%	0.9%	82.9%	12.6%	libiomp5.so	void _INTERNAL_25
smp_call_function_many	,000	47.211	1.6%	0.0%	97.2%	2.0%	vmlinux	smp_call_function_r
sysret_check	,000	83.000	46.5%	0.0%	43.2%	36.1%	vmlinux	sysret_check
system_call_after_swapgs	0		18.6%	26.0%	44.3%	11.1%	vmlinux	system_call_after_s
_INTERNAL_25_____src_kmp_barrie	,000	7.625	7.0%	0.0%	8.7%	100.0%	libiomp5.so	_INTERNAL_25___
__kmp_yield	0		82.7%	0.0%	0.0%	60.2%	libiomp5.so	__kmp_yield
__kmp_x86_pause	0		0.0%	0.0%	92.3%	99.5%	libiomp5.so	__kmp_x86_pause
clear_page_c_e	,000	39.000	0.0%	0.0%	100.0%	0.0%	vmlinux	clear_page_c_e
__kmp_x86_pause	0		14.8%	0.0%	85.2%	29.6%	libiomp5.so	__kmp_x86_pause
__schedule	,000	29.000	44.3%	88.7%	0.0%	14.8%	vmlinux	__schedule
ktime_get	0		15.9%	31.7%	52.4%	0.0%	vmlinux	ktime_get

0ms 100ms 200ms 300ms 400ms 500ms

Thread

- OMP Worker Thread #91 (TI...
- OMP Worker Thread #83 (TI...
- OMP Worker Thread #8 (TI...
- OMP Worker Thread #19 (TI...
- OMP Worker Thread #37 (TI...

CPU Time

System Bandwidth ■114

Ruler Area:

- Region Instance
- OpenMP Barrier-to-Barrier Segment
- Thread
  - Running
  - CPU Time
- CPU Time
  - CPU Time
- System Bandwidth

FILTER 100.0% | Any Process | Thread Any Thread | Any Module | User functions | Show inline f | Functions on

General Exploration Analysis – Intel Xeon

- Results:
  - Intel Xeon: high level of LLC hit

- Suggestion: improve data locality in order to make better use of cache
  - loop interchange;

Loop Interchange

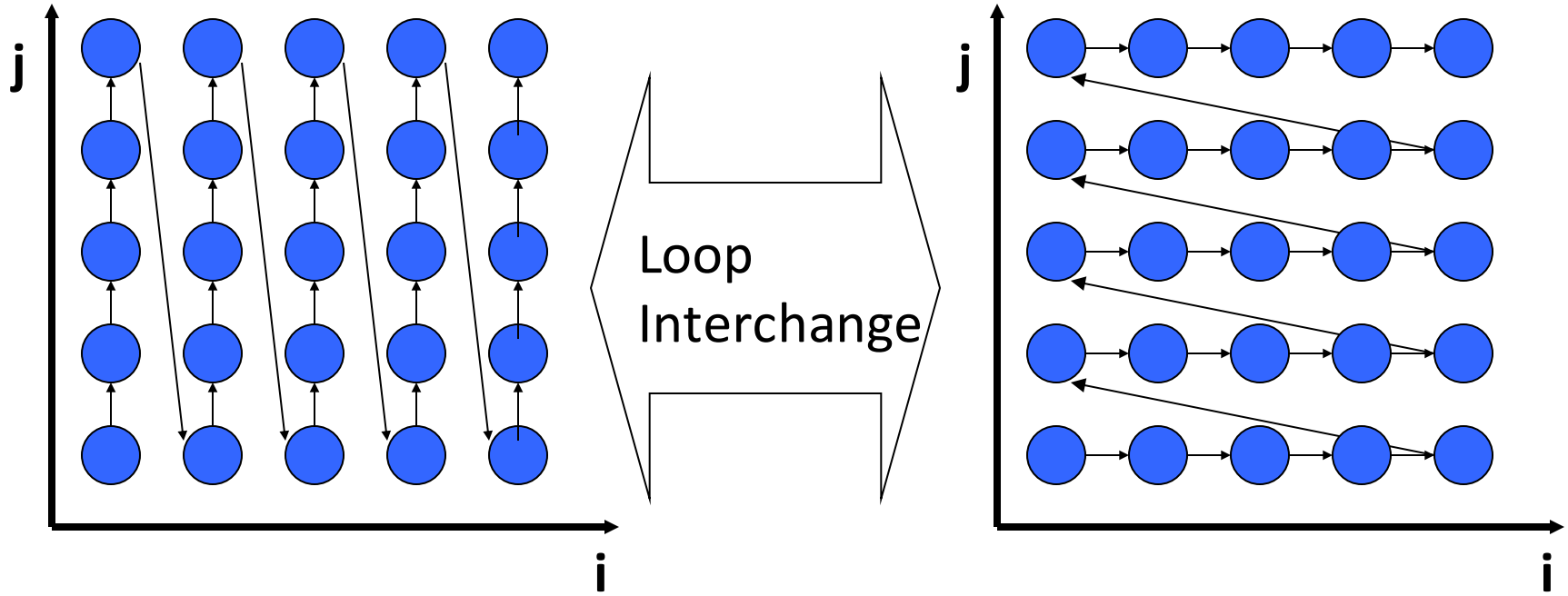
- Row-major order storage.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Loop Interchange

Reading in Row –Major order improves locality




```
for(i=0; i<W; i++)
  for(j=0; j<H; j++)
    A[i][j] = …;
```

```
for(j=0; j<H; j++)
  for(i=0; i<W; i++)
    A[i][j] = …;
```

Matrix Multiplication Loop Interchange

```
void multiply1(int msize, int tidx, int numt, TYPE a[][NUM], TYPE
b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{
  int i,j,k;
  #pragma omp parallel for
  for(i=0; i<msize; i++) {
        for(k=0; k<msize; k++) {
       for(j=0; j<msize; j++) {
                    c[i][j] = c[i][j] + a[i][k] * b[k][j];
                }
            }
      }
}
```

Matrix Multiplication Loop Interchange

Interchange

Clockticks:	86,272,200,000	
Instructions Retired:	3,641,400,000	
CPI Rate:	23.692	
Front-End Bound:	2.8%	of Pipeline Slots
Bad Speculation:	0.0%	of Pipeline Slots
Back-End Bound:	88.2%	of Pipeline Slots
Memory Bound:	66.3%	of Pipeline Slots
L1 Bound:	3.5%	of Clockticks
FB Full:	100.0%	of Clockticks
L2 Bound:	0.1%	of Clockticks
L3 Bound:	4.5%	of Clockticks
Contested Accesses:	0.9%	of Clockticks
Data Sharing:	0.0%	of Clockticks
L3 Latency:	6.1%	of Clockticks
DRAM Bound:	2.1%	of Clockticks
Memory Bandwidth:	6.5%	of Clockticks
Memory Latency:	1.7%	of Clockticks
Local DRAM:	1.5%	of Clockticks
Remote DRAM:	0.0%	of Clockticks
Remote Cache:	0.2%	of Clockticks
Store Bound:	0.0%	of Clockticks
Core Bound:	21.8%	of Pipeline Slots
Port Utilization:	3.3%	of Clockticks
Retiring:	9.0%	of Pipeline Slots
Total Thread Count:	98	
Paused Time:	0s	

Without Interchange

Clockticks:	492,714,600,000	
Instructions Retired:	5,294,100,000	
CPI Rate:	93.069	
Front-End Bound:	2.5%	of Pipeline Slots
Bad Speculation:	0.0%	of Pipeline Slots
Back-End Bound:	94.9%	of Pipeline Slots
Memory Bound:	92.9%	of Pipeline Slots
L1 Bound:	3.0%	of Clockticks
FB Full:	N/A*	of Clockticks
L2 Bound:	N/A*	of Clockticks
L3 Bound:	32.6%	of Clockticks
Contested Accesses:	0.7%	of Clockticks
Data Sharing:	0.0%	of Clockticks
L3 Latency:	3.5%	of Clockticks
DRAM Bound:	N/A*	of Clockticks
Memory Bandwidth:	38.6%	of Clockticks
Memory Latency:	22.7%	of Clockticks
Local DRAM:	4.2%	of Clockticks
Remote DRAM:	0.2%	of Clockticks
Remote Cache:	0.6%	of Clockticks
Store Bound:	0.0%	of Clockticks
Core Bound:	2.0%	of Pipeline Slots
Port Utilization:	1.4%	of Clockticks
Retiring:	2.6%	of Pipeline Slots
Total Thread Count:	98	
Paused Time:	0s	

HPC Performance Characterization

SP GFLOPS: **28.970**

Effective Physical Core Utilization: 35.0% (16.788 out of 48)

Effective Logical Core Utilization: 32.1% (30.819 out of 96)

Serial Time (outside parallel regions): 0.062s (29.9%)

Parallel Region Time: 0.145s (70.1%)

Estimated Ideal Time: 0.085s (41.1%)

OpenMP Potential Gain: 0.060s (29.0%)

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	OpenMP Potential Gain	(%)	OpenMP Region Time
multiply3ompparallel:96@unknown:207:215	0.060s	29.0%	0.145s

Effective CPU Utilization Histogram

Memory Bound: 60.2% of Pipeline Slots

Cache Bound: 10.5% of Clockticks

DRAM Bound: 9.6% of Clockticks

NUMA: % of Remote Accesses: 0.0%

Bandwidth Utilization Histogram

HPC Performance Characterization

FPU Utilization: 0.3%

SP FLOPs per Cycle: 0.221 Out of 64

Vector Capacity Usage: 100.0%

FP Instruction Mix:

% of Packed FP Instr.: 100.0%

% of 128-bit: 0.0%

% of 256-bit: 0.0%

% of 512-bit: 100.0%

% of Scalar FP Instr.: 0.0%

FP Arith/Mem Rd Instr. Ratio: 0.132

FP Arith/Mem Wr Instr. Ratio: 0.363

Top Loops/Functions with FPU Usage by CPU Time

This section provides information for the most time consuming loops/functions with floating point operations.

Function	CPU Time	FPU Utilization	Vector Instruction Set	Loop Type
[Loop at line 211 in multiply3ompparallel_for @207]	5.701s	0.3%	AVX512F_512(512)	Body
__kmp_fork_barrier	4.700s	0.4%		
osq_lock	0.112s	1.7%		
[Loop at line 208 in multiply3ompparallel_for @207]	0.073s	3.9%	AVX(256); AVX2(256); AVX512F_256(256); AVX512F_512(512)	Body