



Hands-on “modernization of an application using Intel Advisor”

# Matrix Transposition

- This Hands-on is based on chapter 24 (“Profiling-Guided Optimization”) - “High Performance Parallelism Pearls” book;
- This chapter exploits the use of Intel® VTune™ Amplifier XE reports to understand where to apply optimization on matrix transposition.



# Agenda

---

- Speedup Estimation Analysis
  - Create Advisor Project
  - Collect Survey Data
  - Include Annotations
  - Collect Suitability Data

# Agenda

---

- Speedup Estimation Analysis
  - Create Advisor Project
  - Collect Survey Data
  - Include Annotations
  - Collect Suitability Data
- ;

# Intel Advisor

- Intel® Advisor XE is a shared memory threading designing and prototyping tool for C, C++, C# and Fortran.
- This tool provides a mechanism called annotation which is used to inform the regions of the code which can be parallelized. Based on such annotations a model is built in order to compare the performance scaling of different threading designs without the cost and disruption of implementation.
- In this part of Hands-on you will execute the following steps in order to perform threading prototyping:
  - ✓ Create project;
  - ✓ Survey data: Discover opportunities for parallelization;
  - ✓ Annotate sources: include annotations on source code to check scalability;
  - ✓ Check suitability: evaluate the performance of annotated loops in different architectures and frameworks.

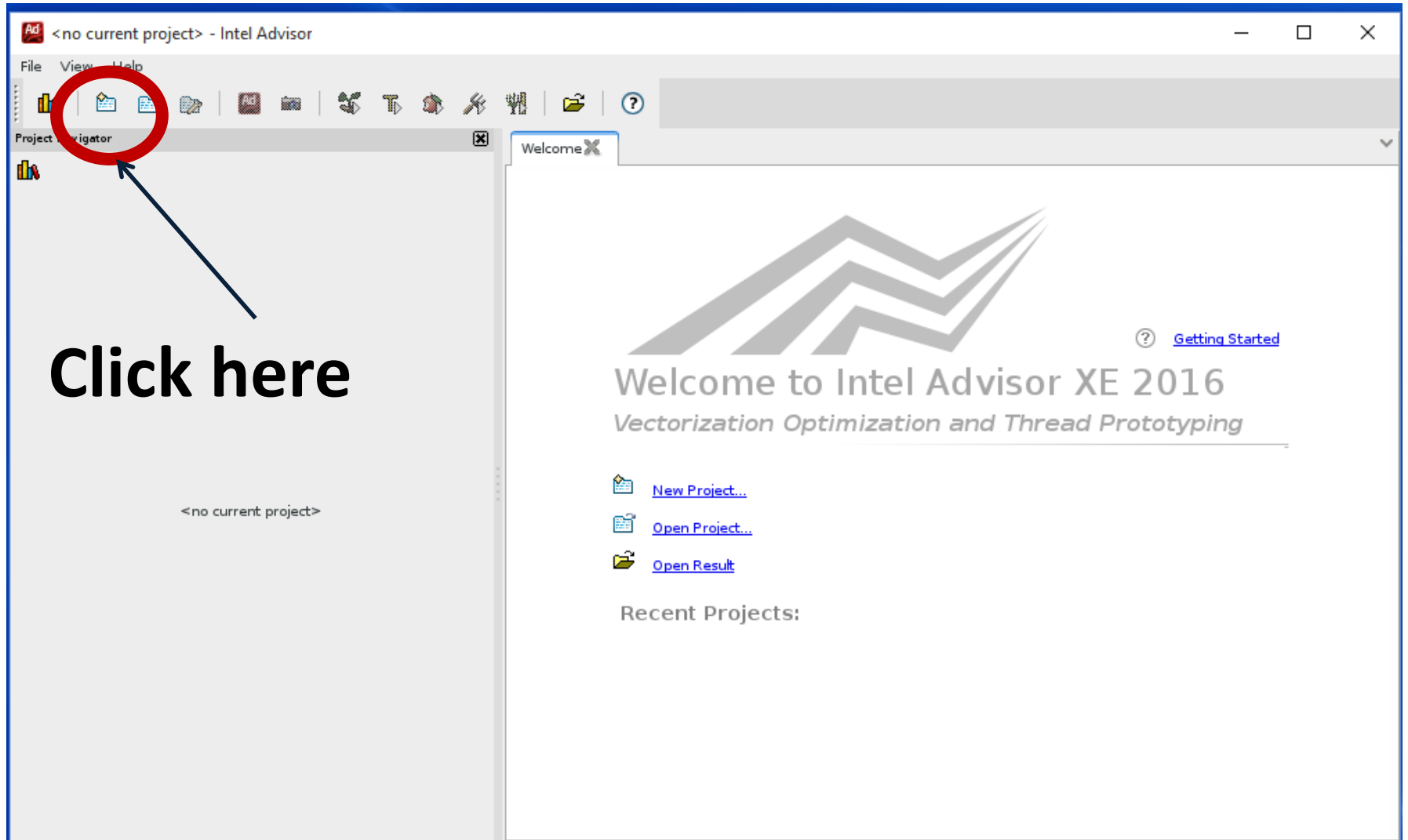
# Agenda

---

- Speedup Estimation Analysis
  - Create Advisor Project
  - Collect Survey Data
  - Include Annotations
  - Collect Suitability Data

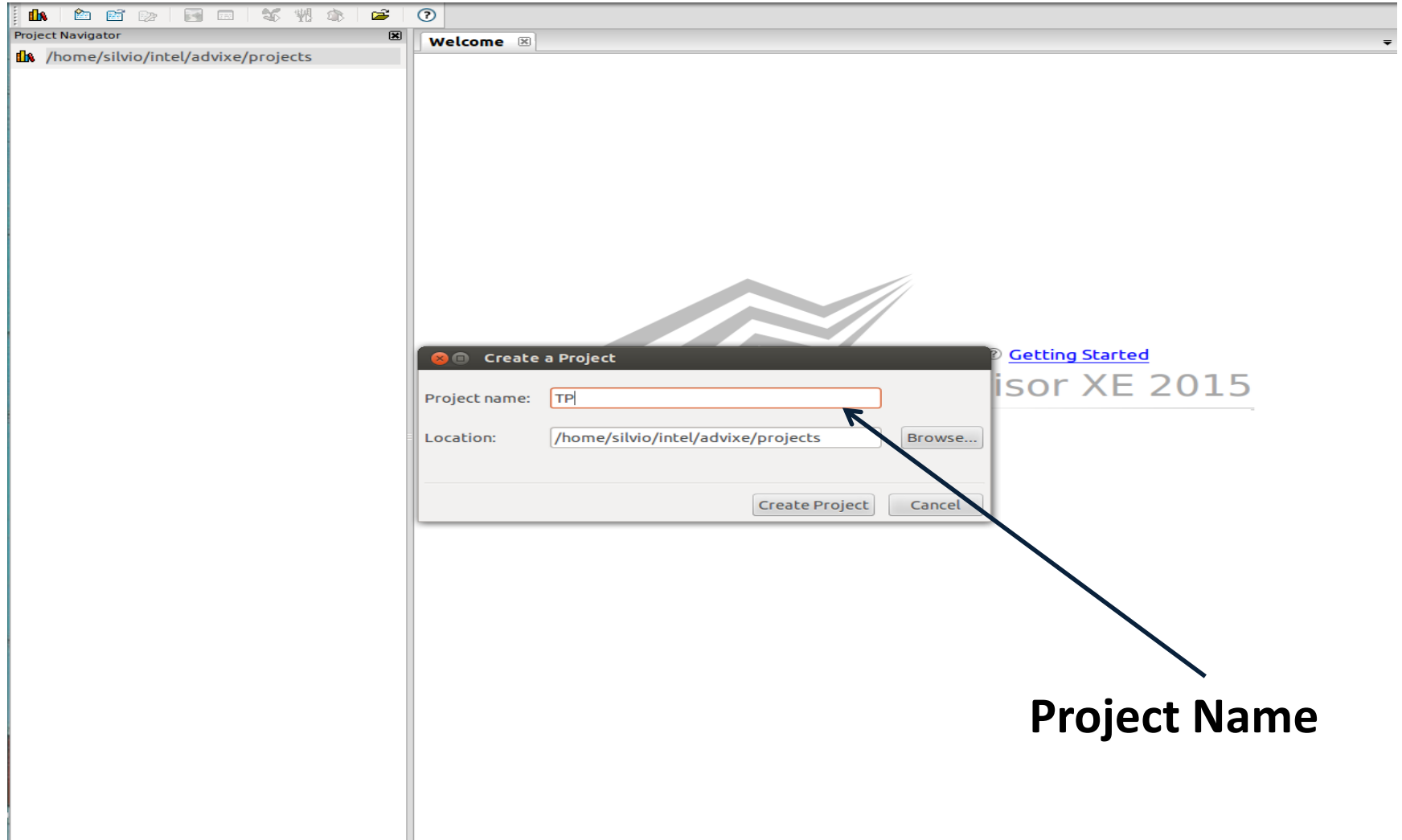
# Intel Advisor – Create New Project

- Execute Intel Advisor on terminal: `advixe-gui`
- create new Advisor project:
  - name: TP
  - application: `~/multiprogramming-Intel/hands-on/transposition/runme-CPU`
  - application parameters: 3000 100
  - Source Folder: `~/multiprogramming-Intel/hands-on/transposition/`





# Intel Advisor – Create New Project



# Intel Advisor – Create New Project

TP - Project Properties

Analysis Target Binary/Symbol Search Source Search

Survey Analysis Types

- Survey Hotspots Analysis
- Survey Trip Count Analysis
- Suitability Analysis

Refinement Analysis Types

- Dependencies Analysis
- Memory Access Patterns An.

Survey Launch Application

Specify and configure the application executable (target) to analyze. Press F1 for more details.

Application: /home/silvio/Pearl/1/runme-CPU Browse...

Application parameters: 4000 1000 Modify...

☒ Use application directory as working directory

Working directory: /home/silvio/Pearl/1 Browse...

User-defined environment variables: Modify...

Child application:

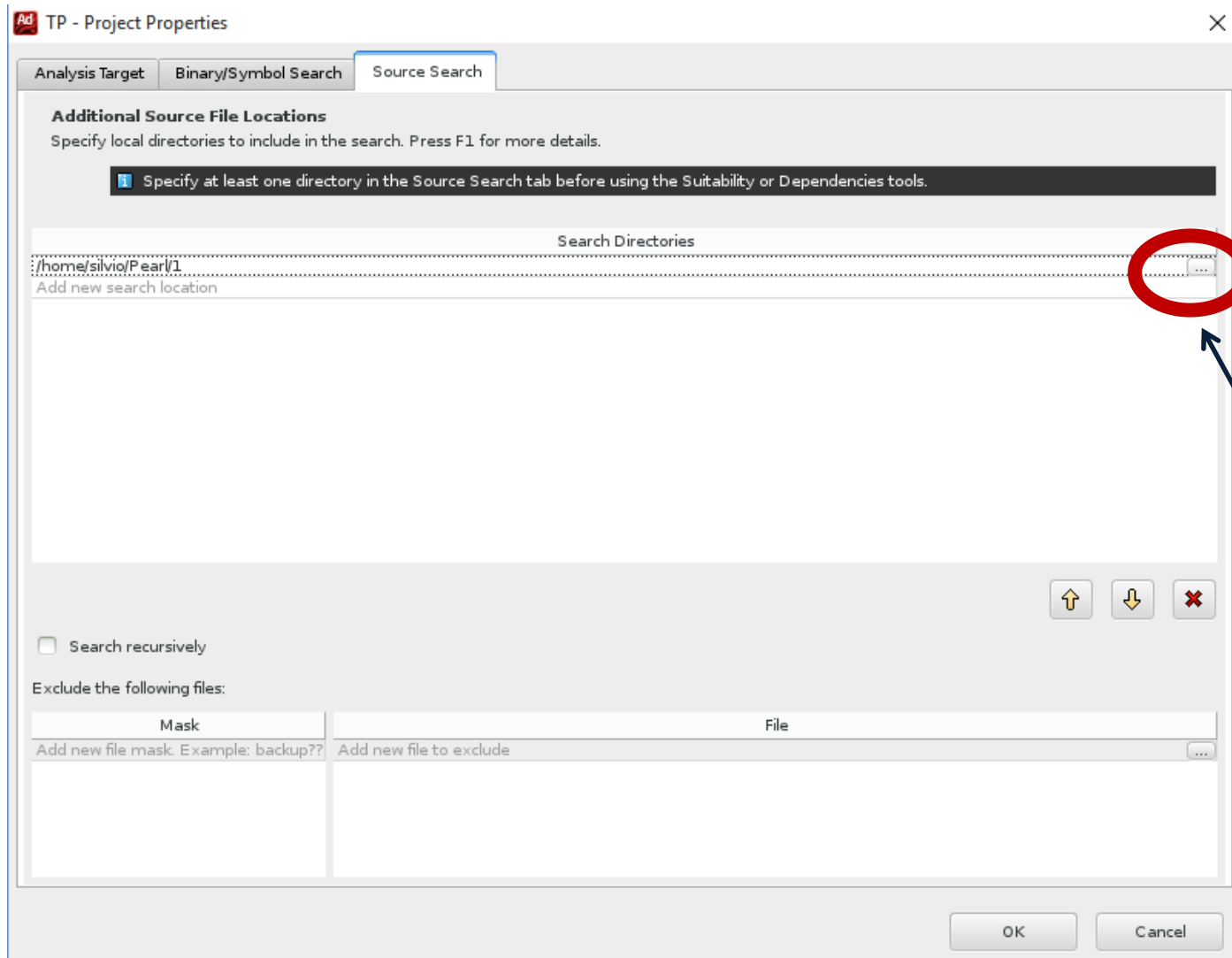
☒ Advanced

OK Cancel

Application

Application Parameters

# Intel Advisor – Source Search



Click here

# Agenda

---

- Speedup Estimation Analysis
  - Create Advisor Project
  - **Collect Survey Data**
  - Include Annotations
  - Collect Suitability Data

# Intel Advisor - collect survey data

The screenshot shows the Intel Advisor XE 2016 application window. The title bar indicates the path: `/home/silvio/intel/advixe/projects/TP - Intel Advisor`. The menu bar includes **File**, **View**, and **Help**. The toolbar contains various icons for file operations and analysis. The main workspace is titled **VECTORIZATION WORKFLOW** and displays a sidebar with the following steps:

- 1. Survey Target**  
Explore where to add efficient vectorization and/or threading.  
**Collect** (highlighted with a red circle and an arrow pointing to it with the text "Click here")  
Command Line
- 1.1 Find Trip Counts**  
Find how many iterations are executed.
- 2.1 Check Dependencies**  
Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.  
**Collect**  
Command Line
- 2.2 Check Memory Access Patterns**  
Identify and explore complex memory accesses for marked loops. Fix the reported problems.  
**Collect**  
Command Line

At the bottom of the sidebar, there is a section for switching between workflows:

Switch between Vectorization and Threading workflows  
**Threading Workflow**

The main content area shows a banner for **Where should I add vectorization and/or threading parallelism?** with tabs for **Summary**, **Survey Report**, **Refinement Reports**, and **Annotation Report**. Below the banner, a **No Data** warning is displayed:

**No Data**  
To collect data about your application's performance, compile your application with Release build settings and run [Survey](#) analysis.

# Intel Advisor - survey data results

## ⚠ Higher instruction set architecture (ISA) available

Consider recompiling your application using a higher ISA.

Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops			Compiler Estimated Gain	Instruction Set Analysis			Location
						Vector ISA	Gain Estimate	VL (Vector Length)		Traits	Data Types	Advanced	
[-] [loop in Transpose at Transpose.cc:20]		40.470s	40.470s	Vectorized (8...)		SSE	3.43x	2; 4	3.43x		Float32; Float64	Unrolled by 4	Transpose.cc
[-] [loop in Transpose at Transpose.cc:20]		40.470s	40.470s	Vectorized (8...)		SSE		2; 4	3.43x		Float32; Float64	Unrolled by 4	Transpose.cc
[+] [loop in __kmp_launch_thread at kmp_runtime.c:5900]		24.499s	24.820s	Scalar									kmp_runtime
[-] [loop in VerifyTransposed at Main.cc:24]	⚠ Data type conversions present	0.171s	0.171s	Vectorized (8...)		SSE2	2.73x	2; 4	2.73x	Type Conv...	Float32; Float64; Int32	Unrolled by 4	Main.cc:24
[-] [loop in VerifyTransposed at Main.cc:24]	⚠ Data type conversions present	0.171s	0.171s	Vectorized (8...)		SSE2		2; 4	2.73x	Type Conv...	Float32; Float64; Int32	Unrolled by 4	Main.cc:24
[+] [loop in Transpose at Transpose.cc:20]	⚠ Ineffective peeled/remainder loop...	0.080s	0.080s	Vectorized ...		SSE	1.86x	2; 4	1.86x		Float32; Float64		Transpose.cc
[-] [loop in Transpose at Transpose.cc:20]		0.050s	0.050s	Vectorized (R...)		SSE		2; 4	1.86x		Float32; Float64		Transpose.cc
[+] [loop in Transpose at Transpose.cc:20]		0.030s	0.030s	Remainder									Transpose.cc
[+] [loop in Transpose at Transpose.cc:17]		0.030s	40.500s	Scalar	inner loop was al...								Transpose.cc
[+] [loop in start_thread]		0.000s	24.820s	Scalar									
[+] [loop in __libc_start_main]		0.000s	40.670s	Scalar									
[+] [loop in main at Main.cc:74]		0.000s	40.600s	Scalar	loop with multipl...						Float32; Float64		Main.cc:74
[+] [loop in VerifyTransposed at Main.cc:23]		0.000s	0.171s	Scalar						Unpacks	Float32; Float64; Int32; Int64		Main.cc:23
[+] [loop in [OpenMP worker] at z_Linux_util.c:786]		0.000s	24.820s	Scalar									z_Linux_util.c

Source					Total Time		Loop Time		Traits
Line	Source					%		%	
7	// You are free to use, modify and distribute this code as long as you acknowledge								
8	// the above mentioned publication.								
9	// (c) Colfax International, 2013								
10									
11	#include "Transpose.h"								
12	#include <stdlib.h>								
13									
14									
15	void Transpose(FTYPE* const A, const int n) {								
16									
17	for (int i = 0; i < n; i++) {						40.499s		
18									
19									
20	for (int i = 0; i < i; i++) {				0.280s		40.550s		
21									
22	const FTYPE c = A[i*n + i];				29.950s				
23	A[i*n + i] = A[i*n + i];				10.180s				
24	A[i*n + i] = c;				0.190s				
25	}								
26									
27	}								
28									
29	}								
30									

# Intel Advisor - survey data results

Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops				Instruction Set Analysis		Advanced	Location
						Vector ISA	Gain Estimate	VL (Vector Length)	Compiler Estimated Gain	Traits	Data Types		
[-] [loop in Transpose at Transpose.cc:20]		40.470s	40.470s	Vectorized (8 ...)		SSE	3.43x	2; 4	3.43x		Float32; Float64	Unrolled by 4	Transpose.cc
[+] [loop in Transpose at Transpose.cc:20]		40.470s	40.470s	Vectorized (8 ...)		SSE		2; 4	3.43x		Float32; Float64	Unrolled by 4	Transpose.cc
[+] [loop in __kmp_launch_thread at kmp_runtime.c:5900]		24.499s	24.820s	Scalar									kmp_runtime
[-] [loop in VerifyTransposed at Main.cc:24]	1 Data type conversions present	0.171s	0.171s	Vectorized (8 ...)		SSE2	2.73x	2; 4	2.73x	Type Conv...	Float32; Float64; Int32	Unrolled by 4	Main.cc:24
[+] [loop in VerifyTransposed at Main.cc:24]	1 Data type conversions present	0.171s	0.171s	Vectorized (8 ...)		SSE2		2; 4	2.73x	Type Conv...	Float32; Float64; Int32	Unrolled by 4	Main.cc:24
[-] [loop in Transpose at Transpose.cc:20]	1 Ineffective peeled/remainder loop ...	0.080s	0.080s	Vectorized ...		SSE	1.86x	2; 4	1.86x		Float32; Float64		Transpose.
[+] [loop in Transpose at Transpose.cc:20]		0.050s	0.050s	Vectorized (R ...)		SSE		2; 4	1.86x		Float32; Float64		Transpose.cc
[+] [loop in Transpose at Transpose.cc:20]		0.030s	0.030s	Remainder									Transpose.cc
[+] [loop in Transpose at Transpose.cc:17]		0.030s	40.500s	Scalar	inner loop was al...								Transpose.cc
[+] [loop in start_thread]		0.000s	24.820s	Scalar									
[+] [loop in __libc_start_main]		0.000s	40.670s	Scalar									
[+] [loop in main at Main.cc:74]		0.000s	40.600s	Scalar	loop with multipl...						Float32; Float64		Main.cc:74
[+] [loop in VerifyTransposed at Main.cc:23]		0.000s	0.171s	Scalar						Unpacks	Float32; Float64; Int32; Int64		Main.cc:23
[+] [loop in [OpenMP worker] at z_Linux_util.c:786]		0.000s	24.820s	Scalar									z_Linux_util.c

Source <div>Top Down</div> Loop Assembly <div>Recommendations</div> <div>Compiler Diagnostic Details</div>													
Function Call Sites and Loops	Total Time %	Total Time	Self Time	Loop Type	Why No Vectorization?	Vectoriz...		Instruction Set Analysis		Advanced	Location		
						Vec...	VL ...	Traits	Data Types				
<div><div><div></div></div>Total</div>	100.0%	<div><div></div>65.490s</div>	<div><div></div>0s</div>										
<div><div><div></div></div><div><div><div></div></div>__libc_start_main</div></div>	62.1%	<div><div></div>40.670s</div>	<div><div></div>0s</div>										
<div><div><div><div></div></div><div>[loop in __libc_start_main]</div></div></div>	62.1%	<div><div></div>40.670s</div>	<div><div></div>0s</div>	Scalar									
<div><div><div><div></div></div><div>[Unknown stack frame(s)]</div></div></div>	62.0%	<div><div></div>40.600s</div>	<div><div></div>0s</div>										
<div><div><div><div></div></div><div>main</div></div></div>	62.0%	<div><div></div>40.600s</div>	<div><div></div>0s</div>								Main.cc:38		
<div><div><div><div></div></div><div>[loop in main at Main.cc:74]</div></div></div>	62.0%	<div><div></div>40.600s</div>	<div><div></div>0s</div>	Scalar	loop with multiple exits cannot be vectorized unless it meets search loop idiom criteria			Float32; Float64			Main.cc:74		
<div><div><div><div></div></div><div>Transpose</div></div></div>	62.0%	<div><div></div>40.600s</div>	<div><div></div>0.0200s</div>								Transpose.cc:15		
<div><div><div><div></div></div><div>[loop in Transpose at Transpose.cc:17]</div></div></div>	61.8%	<div><div></div>40.500s</div>	<div><div></div>0.0300s</div>	Scalar	inner loop was already vectorized						Transpose.cc:17		
<div><div><div><div></div></div><div>[loop in Transpose at Transpose.cc:20]</div></div></div>	61.8%	<div><div></div>40.470s</div>	<div><div></div>0s</div>								Transpose.cc:20		
<div><div><div><div></div></div><div>[loop in Transpose at Transpose.cc:20]</div></div></div>	61.8%	<div><div></div>40.470s</div>	<div><div></div>40.4696s</div>	Vectorized (Body)		SSE	2; 4	Float32; Float64	Unrolled by 4		Transpose.cc:20		
<div><div><div><div></div></div><div>[loop in Transpose at Transpose.cc:20]</div></div></div>	0.1%	<div><div></div>0.080s</div>	<div><div></div>0s</div>								Transpose.cc:20		
<div><div><div><div></div></div><div>[loop in Transpose at Transpose.cc:20]</div></div></div>	0.1%	<div><div></div>0.050s</div>	<div><div></div>0.0500s</div>	Vectorized (Remainder)		SSE	2; 4	Float32; Float64			Transpose.cc:20		
<div><div><div><div></div></div><div>[loop in Transpose at Transpose.cc:20]</div></div></div>	0.0%	<div><div></div>0.030s</div>	<div><div></div>0.0300s</div>	Remainder							Transpose.cc:20		
<div><div><div><div></div></div><div>main</div></div></div>	0.1%	<div><div></div>0.070s</div>	<div><div></div>0s</div>								Main.cc:38		
<div><div><div><div></div></div><div>_clone</div></div></div>	37.9%	<div><div></div>24.820s</div>	<div><div></div>0s</div>										

# Intel Advisor - survey data results

---

- What lines are the hot spots in this code?



# Agenda

---

- Speedup Estimation Analysis
  - Create Advisor Project
  - Collect Survey Data
  - Include Annotations
  - Collect Suitability Data

# Intel Advisor – Include Annotations

- Include the annotations on `~/multiprogramming-Intel/hands-on/transposition/transpose.cc` in the following way:
  - **#include "advisor-annotate.h"**: include header file
  - **ANNOTATE\_SITE\_BEGIN(id)**: before beginning of loop;
  - **ANNOTATE\_ITERATION\_TASK(id)**: first line inside the loop;
  - **ANNOTATE\_SITE\_END()**: after end of loop;

# Intel Advisor – Include Annotations

```
#include "Transpose.h"  
#include <cstdlib>
```

```
void Transpose(FTYPE* const A, const int n) {  
  
    for (int j = 0; j < n; j++) {  
  
        for (int i = 0; i < j; i++) {  
  
            const FTYPE c = A[i*n + j];  
            A[i*n + j] = A[j*n + i];  
            A[j*n + i] = c;  
        }  
  
    }  
  
}
```

```
#include "Transpose.h"  
#include <cstdlib>  
#include "advisor-annotate.h"  
  
void Transpose(FTYPE* const A, const int n) {  
    ANNOTATE_SITE_BEGIN( MySite1 );  
    for (int j = 0; j < n; j++) {  
        ANNOTATE_ITERATION_TASK( MyTask1 );  
        ANNOTATE_SITE_BEGIN( MySite2 );  
        for (int i = 0; i < j; i++) {  
            ANNOTATE_ITERATION_TASK( MyTask2 );  
            const FTYPE c = A[i*n + j];  
            A[i*n + j] = A[j*n + i];  
            A[j*n + i] = c;  
        }  
        ANNOTATE_SITE_END();  
    }  
    ANNOTATE_SITE_END();  
}
```

# Intel Advisor – Include Annotations

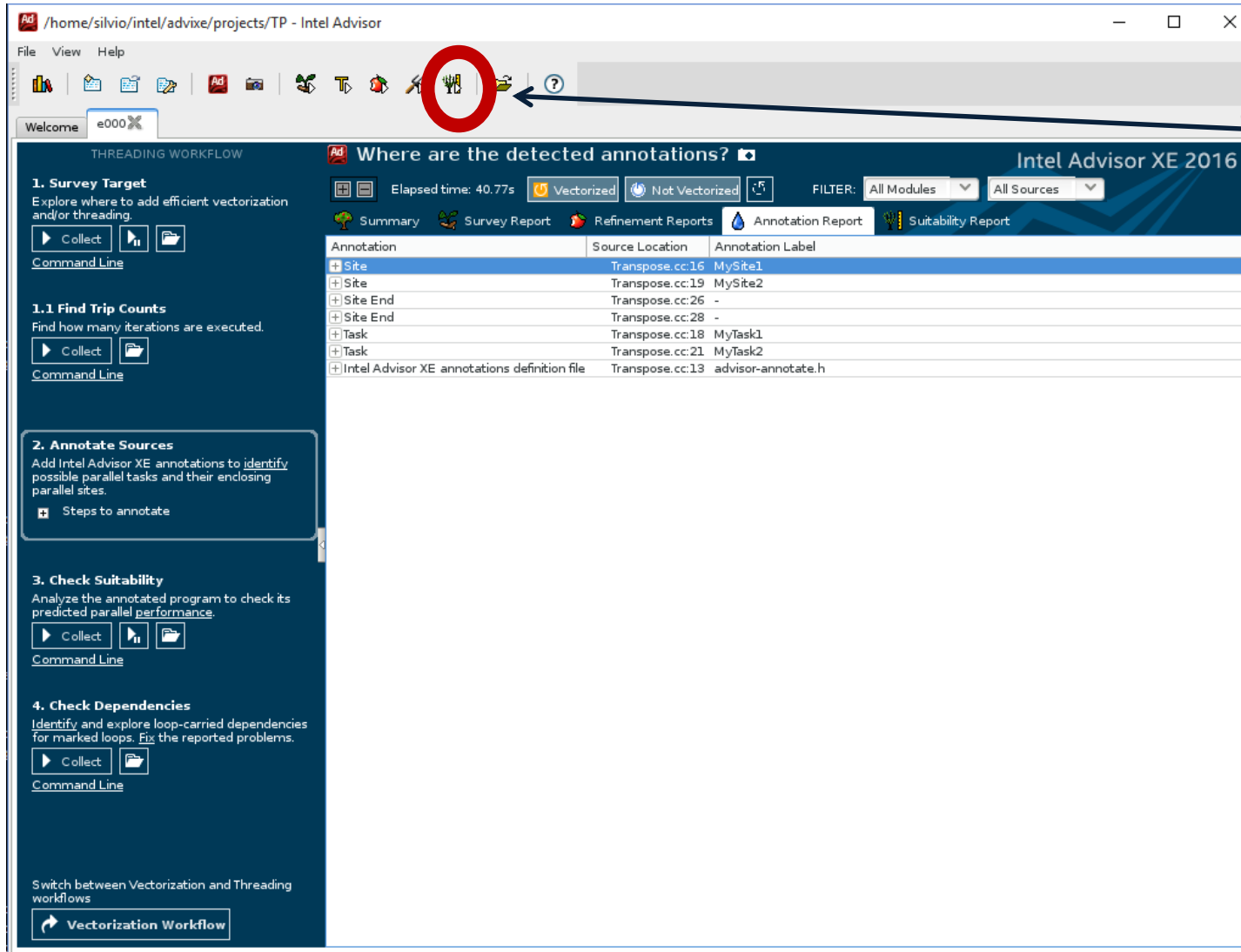
- Red Lines shows the lines that has to be included in original source code;
- After include these lines recompile application:
  - `Cd ~/~/multiprogramming-Intel/hands-on/transposition/`
  - `rm -rf runme-CPU`
  - `make`
- Check your annotations on “view annotations” options;

# Agenda

---

- Speedup Estimation Analysis
  - Create Advisor Project
  - Collect Survey Data
  - Include Annotations
  - Collect Suitability Data

# Intel Advisor - collect suitability data



The screenshot shows the Intel Advisor XE 2016 interface. The main window is titled "Where are the detected annotations?". The left sidebar contains a "THREADING WORKFLOW" section with four steps: 1. Survey Target, 1.1 Find Trip Counts, 2. Annotate Sources, 3. Check Suitability, and 4. Check Dependencies. The right pane displays a table of annotations. The toolbar at the top includes icons for various functions, with the "Suitability Report" icon (a green tree-like symbol) circled in red. An arrow points from the text "Click here" to this icon.

Intel Advisor XE 2016

Elapsed time: 40.77s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Summary | Survey Report | Refinement Reports | Annotation Report | Suitability Report

Annotation	Source Location	Annotation Label
+ Site	Transpose.cc:16	MySite1
+ Site	Transpose.cc:19	MySite2
+ Site End	Transpose.cc:26	-
+ Site End	Transpose.cc:28	-
+ Task	Transpose.cc:18	MyTask1
+ Task	Transpose.cc:21	MyTask2
+ Intel Advisor XE annotations definition file	Transpose.cc:13	advisor-annotate.h

Switch between Vectorization and Threading workflows

Vectorization Workflow

Click  
here

# Check Suitability Expected Results (Intel Xeon) Site 1

Ad

/home/silvio/intel/advixe/projects/TP - Intel Advisor

File View Help

Welcome e000X

THREADING WORKFLOW

1. Survey Target

Explore where to add efficient vectorization and/or threading.

Command Line

1.1 Find Trip Counts

Find how many iterations are executed.

Command Line

2. Annotate Sources

Add Intel Advisor XE annotations to identify possible parallel tasks and their enclosing parallel sites.

3. Check Suitability

Analyze the annotated program to check its predicted parallel performance.

Command Line

4. Check Dependencies

Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.

Command Line

Switch between Vectorization and Threading workflows

Vectorization Workflow

What are the performance implications of the annotated sites?

Intel Advisor XE 2016

Elapsed time: 40.77s

Vectorized Not Vectorized

FILTER: All Modules All Sources

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Instances of task MyTask2 are too small. Suitability data may be unreliable.

View Source

Maximum Program Gain For All Sites: 1.59x

Target System: CPU Threading Model: Intel TBB CPU Count: 8

Serial time: 189.6191s  
Predicted Parallel time: 119.0152s

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances				Site Instance Metrics, Parallel Time
			Total Serial Time	Total Parallel Time	Site Gain		
MySite1	Transpose.cc:16	7.60x	189.51s	24.84s	7.63x		0.0248s
MySite2	Transpose.cc:19	0.10x	187.64s	1859.07s	0.10x		0.0005s

Site Performance Scalability

Site Details

Scalability of Maximum Site Gain

CPU Count	Maximum Site Gain
2	~1.0x
4	~1.0x
8	~1.59x
16	~1.59x
32	~1.59x
64	~1.59x

Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 4000

Avg. Iteration (Task) Duration: < 0.0001s

0.008x  
0.040x  
0.200x  
1x (4000)  
5x  
25x  
125x

0.008x  
0.040x  
0.200x  
1x (< 0.0001s)  
5x  
25x  
125x

Apply

Runtime Modeling

Type of Change

Gain Benefit if Enabled

☐ Reduce Site Overhead

☐ Reduce Task Overhead +0.10x

☐ Reduce Lock Overhead

☐ Reduce Lock Contention

☐ Enable Task Chunking +0.13x

3.2% Load Imbalance: 0.7779s

3.7% Runtime Overhead: 0.9259s

0.0% Lock Contention: 0s

Total Parallel Time: 24.84s

# Check Suitability Expected Results (Intel Xeon) Site 2

Ad

/home/silvio/intel/advixe/projects/TP - Intel Advisor

File View Help

Welcome

e000X

THREADING WORKFLOW

1. Survey Target

Explore where to add efficient vectorization and/or threading.

Collect

Command Line

1.1 Find Trip Counts

Find how many iterations are executed.

Collect

Command Line

2. Annotate Sources

Add Intel Advisor XE annotations to identify possible parallel tasks and their enclosing parallel sites.

Steps to annotate

3. Check Suitability

Analyze the annotated program to check its predicted parallel performance.

Collect

Command Line

4. Check Dependencies

Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.

Collect

Command Line

Switch between Vectorization and Threading workflows

Vectorization Workflow

What are the performance implications of the annotated sites?

Elapsed time: 40.77s Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Instances of task MyTask2 are too small. Suitability data may be unreliable.

View Source

Maximum Program Gain For All Sites: 1.59x

Serial time: 189.6191s  
Predicted Parallel time: 119.0152s

Target System: CPU Threading Model: Intel TBB CPU Count: 8

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances			Site Instance Metrics, Parallel Time
			Total Serial Time	Total Parallel Time	Site Gain	
MySite1	Transpose.cc:16	7.60x	189.51s	24.84s	7.63x	0.0248s
MySite2	Transpose.cc:19	0.10x	187.64s	1859.07s	0.10x	0.0005s

Site Performance Scalability

Scalability of Maximum Site Gain

78.1% Load Imbalance: 1447.97s  
98.9% Runtime Overhead: 1839.47s  
0.0% Lock Contention: 0s

Total Parallel Time: 1859.07s

Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 1999  
0.008x  
0.040x  
0.200x  
1x (1999)  
5x  
25x  
125x

Avg. Iteration (Task) Duration: < 0.0001s  
0.008x  
0.040x  
0.200x  
1x (< 0.0001s)  
5x  
25x  
125x

Apply

Runtime Modeling

Type of Change Gain Benefit if Enabled

☐ Reduce Site Overhead

☐ Reduce Task Overhead +0.60x

☐ Reduce Lock Overhead

☐ Reduce Lock Contention

☐ Enable Task Chunking +0.98x

Warning

Current tasks are too fine-grain, and not effective for multi-threading. Suggestion: Increase task granularity/duration, reduce task overhead, or consider vectorization.





# Check Suitability Expected Results (Intel Xeon Phi) Site 2

Ad

/home/silvio/intel/advixe/projects/TP - Intel Advisor

File View Help

Welcome e000

THREADING WORKFLOW

1. Survey Target

Explore where to add efficient vectorization and/or threading.

Command Line

1.1 Find Trip Counts

Find how many iterations are executed.

Command Line

2. Annotate Sources

Add Intel Advisor XE annotations to identify possible parallel tasks and their enclosing parallel sites.

Steps to annotate

3. Check Suitability

Analyze the annotated program to check its predicted parallel performance.

Command Line

4. Check Dependencies

Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.

Command Line

Switch between Vectorization and Threading workflows

What are the performance implications of the annotated sites? Intel Advisor XE 2016

Elapsed time: 40.77s FILTER: All Modules All Sources

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Instances of task MyTask2 are too small. Suitability data may be unreliable.

Maximum Program Gain For All Sites: 28.40x

Serial time: 1896.1910s  
Predicted Parallel time: 66.7624s

Target System: Intel Xeon Phi Threading Model: Intel TBB Coprocessor Threads: 128

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances		Site Instance Metrics, Parallel Time	
			Total Serial Time	Total Parallel Time	Site Gain	Parallel Time
MySite1	Transpose.cc:16	105.47x	1895.08s	16.86s	112.37x	0.0169s
MySite2	Transpose.cc:19	0.75x	1876.37s	2511.25s	0.75x	0.0006s

Site Performance Scalability

Site Details

Scalability of Maximum Site Gain

83.2% Load Imbalance: 2084.40s

98.3% Runtime Overhead: 2469.50s

0.0% Lock Contention: 0s

Total Parallel Time: 2511.25s

Loop Iterations (Tasks) Modeling

Runtime Modeling

Avg. Number of Iterations (Tasks): 1999

Avg. Iteration (Task) Duration: < 0.0001s

Type of Change: ☐ Reduce Site Overhead +0.13x  
☐ Reduce Task Overhead +1.63x  
☐ Reduce Lock Overhead  
☐ Reduce Lock Contention  
☐ Enable Task Chunking +0.58x

Warning: Current tasks are too fine-grain, and not effective for multi-threading. Suggestion: Increase task granularity/duration, reduce task overhead, or consider vectorization.

Intel Xeon Phi Advanced Modeling

# OpenMP

- Put the following pragma in top of appropriate loop:
  - `#pragma omp parallel for`
- Could you notice some performance improvement?