



# Introduction to Modeling parallelism with Intel® Advisor XE

Technical Consulting Engineer



# Data Driven Threading Design

## Intel® Advisor XE - Threading Prototyping Tool for Architects

- Have you:
- Tried threading an app, but seen little performance benefit?
- Hit a “scalability barrier” ?  
Performance gains level off as you add cores?
- Delayed a release that adds threading because of synchronization errors?

Breakthrough for threading design:

Quickly prototype multiple options

Project scaling on larger systems

Find synchronization errors before implementing threading

Separate design and implementation, design without disrupting development

**Add Parallelism with Less Effort, Less Risk and More Impact**

# Agenda

- Survey
- Add Annotations
- Model Performance Suitability
- Check Correctness
- Add Parallel Framework
- Conclusion

# Intel® Advisor XE Workflow

Transforming many serial algorithms into parallel form takes 5 easy high-level steps:

1. **Survey and Summary tools**: where to add parallelism
2. **Annotations**: experiment with parallel program structure
3. **Suitability tool**: predict and model program scalability
4. **Correctness tool**: discover potential synchronization problems
5. Manually convert annotations to **parallel framework API** (with a little help of Annotations/Summary)

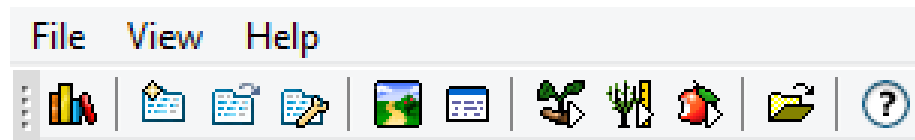
Advisor XE Workflow

- 1. Survey Target**  
[Where](#) should I consider adding parallelism? Locate the loops and functions where your program spends its time, and functions that call them.  
 Collect Survey Data  
 View Survey Result
- 2. Annotate Sources**  
Add Intel Advisor XE annotations to [identify](#) possible parallel tasks and their enclosing parallel sites.  
+ Steps to annotate  
 View Annotations
- 3. Check Suitability**  
Analyze the annotated program to check its predicted parallel [performance](#).  
 Collect Suitability Data  
 View Suitability Result
- 4. Check Correctness**  
[Predict](#) parallel data sharing problems for the annotated tasks. [Fix](#) the reported sharing problems.  
 Collect Correctness Data  
 View Correctness Result
- 5. Add Parallel Framework**  
+ Steps to replace annotations  
 View Summary

Current Project: Benchmarks

# Intel® Advisor XE Workflow

- Advisor XE guides you through these 5 steps
  - provides assisting tools
  - No auto-parallelization
- Model & evaluate potential return of parallelization investments.
- On your serial program



(Advisor XE toolbar)

A screenshot of the Intel Advisor XE Workflow window. The window title is 'Advisor XE Workflow'. It contains five numbered steps, each with an icon and a description:

- 1. Survey Target** (Plant icon): **Where** should I consider adding parallelism? Locate the loops and functions where your program spends its time, and functions that call them. Buttons: Collect Survey Data, View Survey Result.
- 2. Annotate Sources** (Water drop icon): Add Intel Advisor XE annotations to **identify** possible parallel tasks and their enclosing parallel sites. Button: Steps to annotate. View Annotations.
- 3. Check Suitability** (Tree icon): Analyze the annotated program to check its predicted parallel **performance**. Buttons: Collect Suitability Data, View Suitability Result.
- 4. Check Correctness** (Apple icon): **Predict** parallel data sharing problems for the annotated tasks. **Fix** the reported sharing problems. Buttons: Collect Correctness Data, View Correctness Result.
- 5. Add Parallel Framework** (Tree icon): Buttons: Steps to replace annotations, View Summary.

At the bottom right, it says 'Current Project: Benchmarks'.

# Intel® Advisor XE

## Advantages of Advisor XE modeling

Advisor XE modeling avoids the major design mistakes:

1. Measure performance, focus on hotspots.
2. Predict scalability, load balancing and overheads.
3. Predict data races

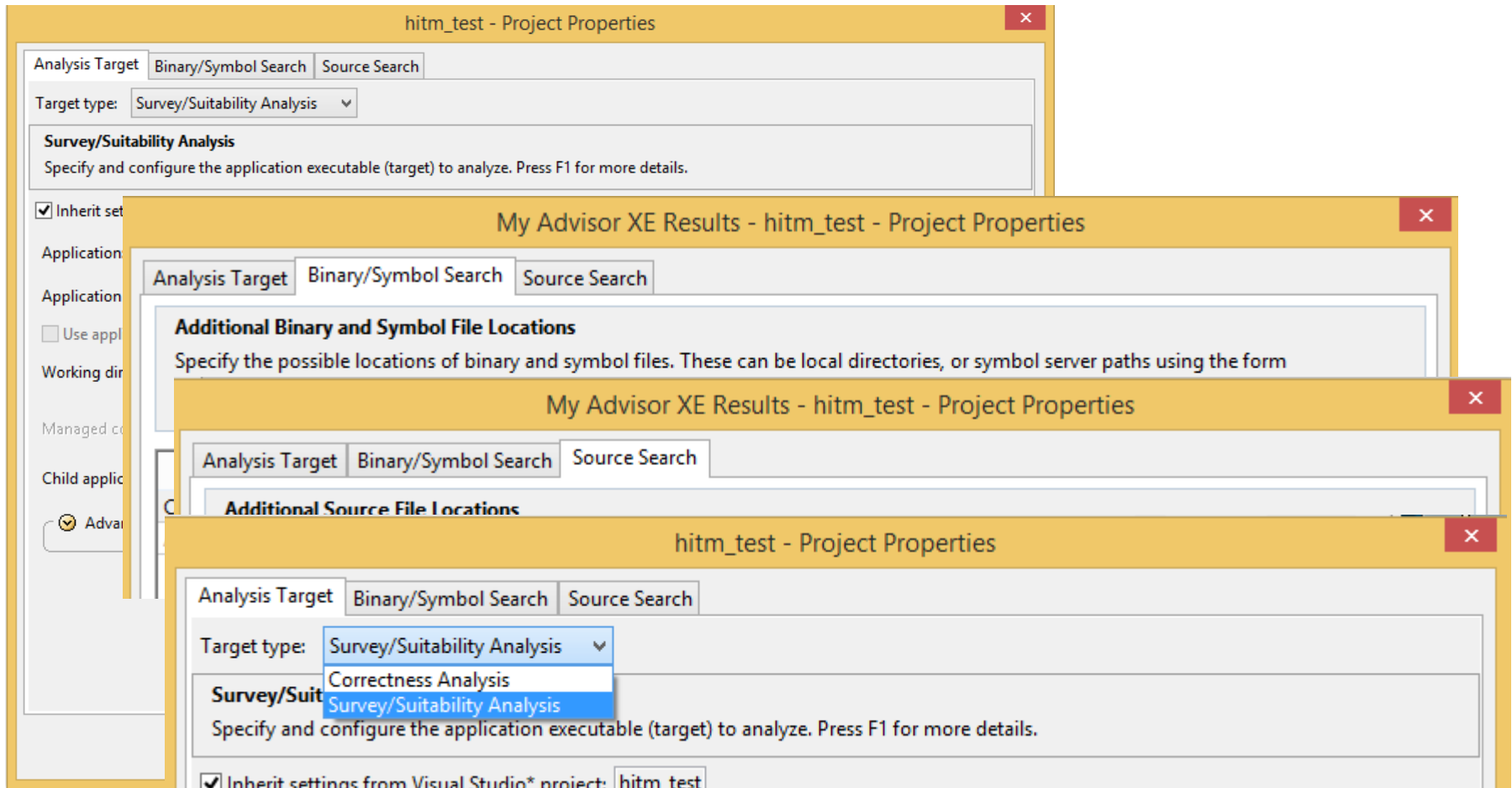
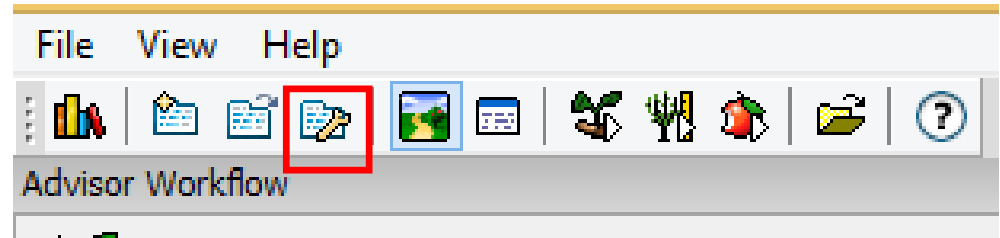
Automated analysis catches cases people miss.

Making good decisions early saves time.



Advisor XE increases parallelization ROI


# Project Set up




# Step 1: Survey Target



**1. Survey Target**  
**Where** should I consider adding parallelism?  
Locate the loops and functions where your program spends its time, and functions that call them.

 Collect Survey Data

 View Survey Result

hitm\_test - e000 x util.c hitm\_test.c

Summary of predicted parallel behavior

Summary | Survey Report | Annotation Report | Suitability Report | Correctness Report

**Intel Advisor helps you choose where to add parallelism to your program**  
Intel Advisor tools help you choose possible parallel code regions, and predict their approximate parallel performance and data sharing problems. View the Advisor Workflow to guide you.

**No Annotations found in your project source files.**  
After scanning 2 source files, 0 annotations have been found.

**Top time-consuming loops<sup>®</sup>**  
Consider adding parallel site and task annotations around these time-consuming loops found during Survey analysis.

Loop	Source Location	CPU Total Time <sup>®</sup>
<a href="#">work</a>	<a href="#">hitm_test.c:136</a>	70.1988s
<a href="#">work</a>	<a href="#">hitm_test.c:141</a>	70.1988s
<a href="#">init_arrays</a>	<a href="#">hitm_test.c:120</a>	0.0745s

**Collection Details**

**Survey**

Collection started: 18 September 2014, 13:28:47  
Collection finished: 18 September 2014, 13:29:09  
Collection time: 00 min 22 sec  
Finalization time: 00 min 02 sec  
Elapsed time: 00 min 24 sec  
Collection Log: [See log](#)  
Application Output: [See output](#)  
Collection Command Line: [See command line](#)

**Suitability (Not available)**

**Correctness (Not available)**

**Platform Information**

Frequency: 1.90 GHz  
Logical CPU Count: 4  
Operating System: Windows  
Computer Name: bbachmay-MOBL7.ger.corp.intel.com

Intel Advisor XE 2015



# Survey Report

hitm\_test - e000

Where should I add parallelism?

Summary Survey Report Annotation Report Suitability Report Correctness Report

View Source

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Hot Loops	Source Location
Total	100.0%	38.4208s	0s		
RtlUserThreadStart	100.0%	38.4208s	0.0100s		
work	99.8%	38.3252s	0s		hitm_test.c:126
[loop at advisor-annotate.h:136 in work]	99.8%	38.3252s	0.0199s		advisor-annotate.h:136
[loop at hitm_test.c:139 in work]	99.6%	38.2851s	38.2851s		hitm_test.c:139
TlsGetValue	0.1%	0.0201s	0.0201s		
_tmainCRTStartup	0.2%	0.0857s	0s		crtexe.c:473
main	0.2%	0.0857s	0s		hitm_test.c:152
init_arrays	0.2%	0.0701s	0s		hitm_test.c:115
[loop at hitm_test.c:118 in init_arrays]	0.2%	0.0701s	0.0100s		hitm_test.c:118
WaitForMultipleObjects	0.0%	0.0156s	0.0156s		

Example: Iteration Loop, Single Task How to add it using a wizard?

```
// To copy compiler options, select Build Settings from the drop-down list.
```

# Drill down to Source Code in the hotspot

Where should I add parallelism? (Source)

Summary Survey Report Annotation Report Suitability Report Correctness Report Survey Source x

File: hitm\_test.c:136 work

Line	Source	Total Time	%	Loop Time	%
125	}				
126	#ifdef WIN32				
127	DWORD WINAPI work(void *pArg) {				
128	#else				
129	void * work(void *pArg) {				
130	#endif				
131					
132	int j = 0, i = 0;				
133	int tid = (int) pArg;				
134					
135					
136					
137	for (j = 0; j < ITERATIONS; j++)				
138	{				
139					
140	for (i = tid; i < MAXSIZE; i+= NUM_PROCS)				
141	{				
142					
143					
144	a[i] = i + a[i] * b[i];				
145	localSum[tid] += a[i];				
146	}				
147					
148					

Selected (Total Time): 0usec

Call Stack with Loops

- work - hitm\_test.c:136
- work - hitm\_test.c:126
- RtlInitializeExceptionChain

Example: Iteration Loop, Single Task

Copy to Clipboard

```
// To copy compiler options, select Build Settings from the drop-down list.

#include "advisor-annotate.h" // Add to each module that contains Intel Advisor annotations

ANNOTATE_SITE_BEGIN( MySite1 ); // Place before the loop control statement to begin a parallel code region (parallel site).
// loop control statement
ANNOTATE_ITERATION_TASK( MyTask1 ); // Place at the start of loop body. This annotation identifies an entire body as a task.
// loop body
ANNOTATE_SITE_END(); // End the parallel code region, after task execution completes
```

# Advisor XE Annotation Concepts

Advisor XE uses 3 primary concepts to create a model

## ■ SITE

- A region of code in your application you want to transform into parallel code

## ■ TASK

- The region of code in a SITE you want to execute in parallel with the rest of the code in the SITE

## ■ LOCK

- Mark regions of code in a TASK which must be serialized

## NOTE

- All of these regions may be nested
- You may create more than one SITE
- Just macros, so work with any C/C++ compiler

The screenshot displays the 'Advisor Workflow' window with the following steps:

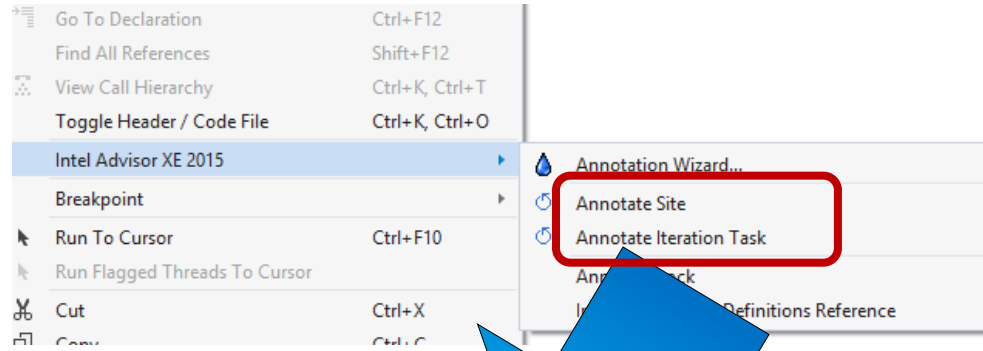
- 1. Survey Target**  
Where should I consider adding parallelism? Locate the loops and functions where your program spends its time, and functions that call them.  
Buttons: Collect Survey Data, View Survey Result
- 2. Annotate Sources**  
Add Intel Advisor annotations to [identify](#) possible parallel tasks and their enclosing parallel sites.  
Buttons: Steps to annotate, View Annotations
- 3. Check Suitability**  
Analyze the annotated program to check its predicted parallel [performance](#).  
Buttons: Collect Suitability Data, View Suitability Result
- 4. Check Correctness**  
[Predict](#) parallel data sharing problems for the annotated tasks. [Fix](#) the reported sharing problems.  
Buttons: Collect Correctness Data, View Correctness Result
- 5. Add Parallel Framework**  
Buttons: Steps to replace annotations, View Summary

Current Project: My Advisor XE Results - hitm\_test

# Candidate: ADD Annotation

## 48% Work::start

```
DWORD WINAPI work(void *pArg) {  
#else  
void * work(void *pArg) {  
#endif  
  
int j = 0, i = 0;  
int tid = (int) pArg;  
  
for (j = 0; j < ITERATIONS; j++)  
{  
for (i = tid; i < MAXSIZE; i+= NUM_PROCS)  
{  
a[i] = i + a[i] * b[i];  
localSum[tid] += a[i];  
}  
}  
return 0;  
}
```



```
void * work(void *pArg)  
#endif  
  
int j = 0, i = 0;  
int tid = (int) pArg;  
ANNOTATE_SITE_BEGIN( MySite5 );  
  
for (j = 0; j < ITERATIONS; j++)  
{  
ANNOTATE_ITERATION_TASK( MyTask9 );  
for (i = tid; i < MAXSIZE; i+= NUM_PROCS)  
{  
a[i] = i + a[i] * b[i];  
localSum[tid] += a[i];  
}  
}  
ANNOTATE_SITE_END();  
return 0;  
}
```

# Step 3:

Estimated Overall Speed-up

hitm\_test - e0

What of the annotated sites?

Summary Suitability Report Correctness Report

Maximum Program Gain For All Sites: 74.08x

Serial time: 68.5132s  
Predicted Parallel time: 0.9249s

Target System: CPU Threading Model: Other CPU Count: 256

Site Label	Source Location	Impact to Program Gain
MySite5	hitm_test.c:133	73.76x

Modeling of Runtime and loops

Site Performance Scalability Site Details

### Scalability of Maximum Site Gain

33.5% Load Imbalance: 0.1836s

### Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks):	Avg. Iteration (Task) Duration:
1000	0.0171s

0.008x, 0.040x, 0.200x, 1x (1000), 5x, 25x, 125x

0.008x, 0.040x, 0.200x, 1x (0.0171s), 5x, 25x, 125x

Apply

### Runtime Modeling

Type of Change	Gain Benefit if Checked
<input type="checkbox"/> Reduce Site Overhead	+1.21x
<input type="checkbox"/> Reduce Task Overhead	+35.75x
<input type="checkbox"/> Reduce Lock Overhead	
<input type="checkbox"/> Reduce Lock Contention	
<input type="checkbox"/> Enable Task Chunking	+26.35x

Scalability Graph

# Adjustable: Target architecture, threading models and number of CPU

hitm\_test - e000

What are the performance implications of the annotated sites?

Summary Survey Report Annotation Report Suitability Report Correctness Report

Maximum Program Gain For All Sites: 74.08x

Serial time: 68.5132s  
Predicted Parallel time: 0.9249s

Site Label	Program Gain	CPU Count
MySite5	hitm_test.c:133 73.76x	256

Target System: CPU Threading Model: Other CPU Count: 256

Site Instance Metrics, Parallel Time: 0.1599s

Collect Suitability data Start Paused Pause

To set up data collection determine the target architecture, threading model and number of CPU's

Collect the Scalability data, and determine how it differs between the architectures and threading models.

# After collecting data change your view

**What are the performance implications of the annotated sites?**

[Summary](#)
[Survey Report](#)
[Annotation Report](#)
[Suitability Report](#)
[Correctness Report](#)

**Maximum Program Gain For All Sites: 141.81x**

Target System: Intel Xeon Phi    Threading Model: Other    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
MySite5	hitm_test.c:133	141.69x	682.24s	1.9411s	351.47x	0.4853s

---

**Site Performance Scalability**    Site Details

**Scalability of Maximum Site Gain**

Coprocessor Threads	Maximum Site Gain
2	58x
4	~16x
8	~32x
16	~64x
32	~128x
64	~256x
128	~512x
256	~1024x

**Loop Iterations (Tasks) Modeling**

Avg. Number of Iterations (Tasks):	Avg. Iteration (Task) Duration:
1000	0.1705s
0.008x	0.008x
0.040x	0.040x
0.200x	0.200x
1x (1000)	1x (0.1705s)
5x	5x
25x	25x
125x	125x

Apply

**Runtime Modeling**

Type of Change	Gain Benefit if Checked
<input type="checkbox"/> Reduce <a href="#">Site Overhead</a>	+0.09x
<input type="checkbox"/> Reduce <a href="#">Task Overhead</a>	+3.42x
<input type="checkbox"/> Reduce <a href="#">Lock Overhead</a>	
<input type="checkbox"/> Reduce <a href="#">Lock Contention</a>	
<input type="checkbox"/> Enable <a href="#">Task Chunking</a>	+3.32x

# Review of of Lab 1

## Survey, Annotate, Suitability

What would be the best target architecture for this code?

Does loop scale well?

At how many iterations would you want to use Xeon Phi coprocessors instead of CPU' s

Does the Threading model make a difference in the scalability?



# Correctness Simulation

Find data sharing problems prior to implementation

- Data Sharing
- Data races

Intel Advisor XE provides a list of errors:

- shows a snippet of the code at all the related code locations
- Correctness Analysis watches the annotated sites for data sharing problems

The screenshot shows the 'Advisor Workflow' window with the following steps:

- 1. Survey Target**  
Where should I consider adding parallelism? Locate the loops and functions where your program spends its time, and functions that call them.  
Buttons: Collect Survey Data, View Survey Result
- 2. Annotate Sources**  
Add Intel Advisor annotations to [identify](#) possible parallel tasks and their enclosing parallel sites.  
Buttons: Steps to annotate, View Annotations
- 3. Check Suitability**  
Analyze the annotated program to check its predicted parallel [performance](#).  
Buttons: Collect Suitability Data, View Suitability Result
- 4. Check Correctness** (highlighted)  
[Predict](#) parallel data sharing problems for the annotated tasks. [Fix](#) the reported sharing problems.  
Buttons: Collect Correctness Data, View Correctness Result
- 5. Add Parallel Framework**  
Buttons: Steps to replace annotations, View Summary

Current Project: My Advisor XE Results - hitm\_test

# Correctness Analysis - Set up

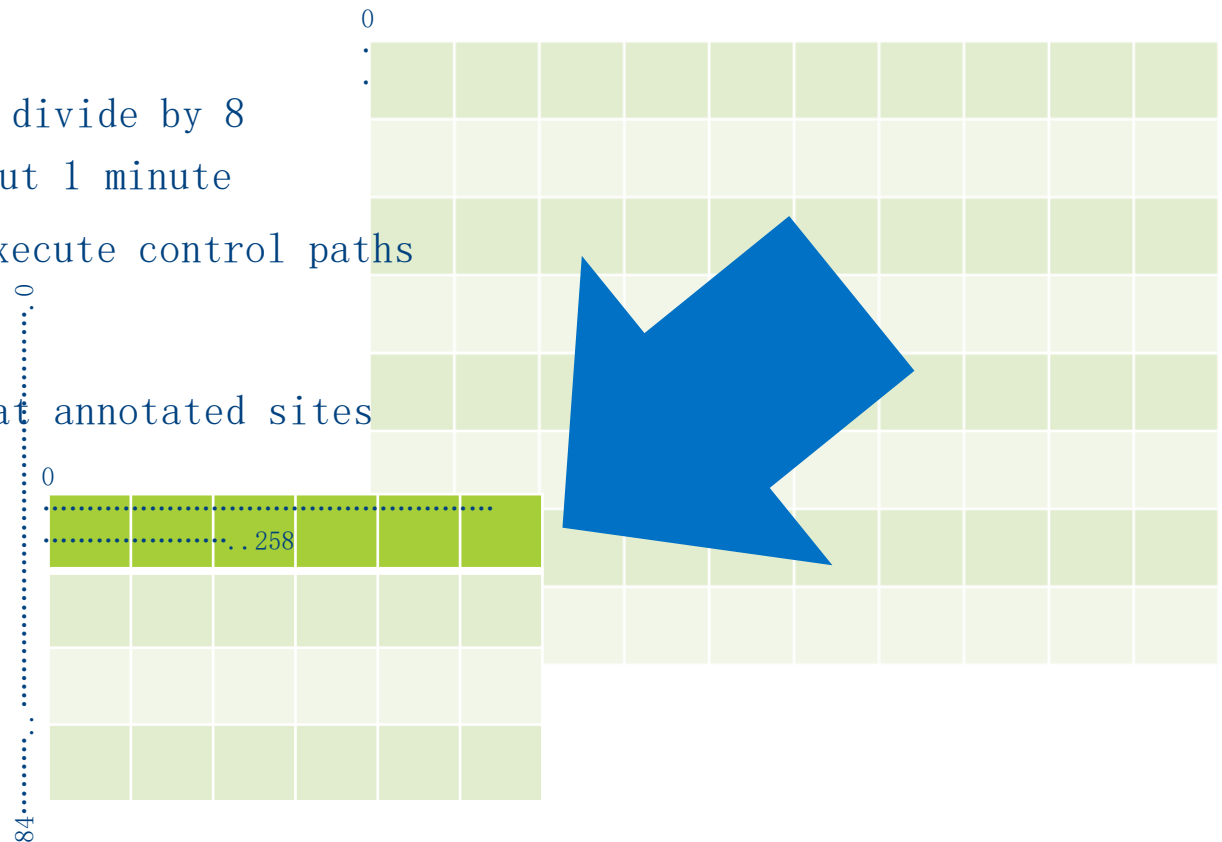
Build a target executable using debug configuration build

Choose a reduced data set that allows execution of all control paths

- Reduce data set size - divide by 8
  - 258x84 executes about 1 minute
- Be sure to thoroughly execute control paths

Annotate your program

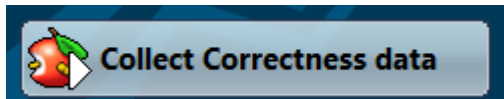
- Correctness only looks at annotated sites



# Check Correctness









Recompile using Debug

Execute your program using



Execution will take longer as the code is executing in debug and the annotations in the code

Advisor Workflow

- 1. Survey Target**  
[Where](#) should I consider adding parallelism? Locate the loops and functions where your program spends its time, and functions that call them.  
 Collect Survey Data  
 View Survey Result
- 2. Annotate Sources**  
Add Intel Advisor annotations to [identify](#) possible parallel tasks and their enclosing parallel sites.  
+ Steps to annotate  
 View Annotations
- 3. Check Suitability**  
Analyze the annotated program to check its predicted parallel [performance](#).  
 Collect Suitability Data  
 View Suitability Result
- 4. Check Correctness**  
[Predict](#) parallel data sharing problems for the annotated tasks. [Fix](#) the reported sharing problems.  
 Collect Correctness Data  
 View Correctness Result
- 5. Add Parallel Framework**  
+ Steps to replace annotations  
 View Summary

Current Project: hitm\_test\_correctness

# Correctness Report

Did the annotated tasks expose data sharing problems?

Summary Survey Report Annotation Report Suitability Report **Correctness Report**

Problems and Messages						
ID	Type	Site Name	Sources	Modules	State	
P1	Parallel site information	MySite1	hitm_test.c	hitm_test.exe	✓ Not a problem	
P2	Data communication	MySite1	hitm_test.c	hitm_test.exe	🚩 New	

**Data communication: Code Locations**

ID	Description	Source	Function	Module	State
X2	Parallel site	hitm_test.c:134	work	hitm_test.exe	🚩 New
<pre>132 int tid = (int) pArg; 133 134 ANNOTATE_SITE_BEGIN( MySite1 ); 135 136</pre>					
X3	Read	hitm_test.c:144	work	hitm_test.exe	🚩 New
<pre>142 143     a[i] = i + a[i] * b[i]; 144     localSum[tid] += a[i]; 145 } 146</pre>					
X4	Write	hitm_test.c:144	work	hitm_test.exe	🚩 New
<pre>142 143     a[i] = i + a[i] * b[i]; 144     localSum[tid] += a[i]; 145 } 146</pre>					

**Filter**

Severity	Count
Error	1 item
Remark	1 item

**Type**

Parallel site information	1 item
Data communication	1 item

**Site Name**

MySite1	2 items
---------	---------

**Source**

hitm_test.c	2 items
-------------	---------

**Module**

hitm_test.exe	2 items
---------------	---------

**State**

New	1 item
Not a problem	1 item

Analyze your annotations to see if you made a correct choice

# Drill down to source code to get more information

Intel Advisor XE 2015

Did the annotated tasks expose data sharing problems? (Source)

Summary Survey Report Annotation Report Suitability Report Correctness Report Correctness Source x

Focus Code Location: hitm\_test.c:144 - Read

```
137 for (j = 0; j < ITERATIONS; j++)
138 {
139     for (i = tid; i < MAXSIZE; i+= NUM_PROCS)
140     {
141         ANNOTATE_ITERATION_TASK( MyTask1 );
142
143         a[i] = i + a[i] * b[i];
144         localSum[tid] += a[i];
145     }
146 }
147 ANNOTATE_SITE_END();
148 return 0;
149 }
```

Call Stack

- work - hitm\_test.c:144

Collect Correctness data

Stop

Cancel

Related Code Location: hitm\_test.c:144 - Write

```
137 for (j = 0; j < ITERATIONS; j++)
138 {
139     for (i = tid; i < MAXSIZE; i+= NUM_PROCS)
140     {
141         ANNOTATE_ITERATION_TASK( MyTask1 );
142
143         a[i] = i + a[i] * b[i];
144         localSum[tid] += a[i];
145     }
146 }
147 ANNOTATE_SITE_END();
148 return 0;
149 }
```

Call Stack

- work - hitm\_test.c:144

Data communication: Code Locations

ID	Description	Source	Function	Module	State
X2	Parallel site	hitm_test...	work	hitm_test.exe	New
X3	Read	hitm_test...	work	hitm_test.exe	New
X4	Write	hitm_test...	work	hitm_test.exe	New

Relationship Diagram

```
graph LR
    X4[Write hitm_test.c:144] --> X3[Read hitm_test.c:144]
```

# Fix the issues shown in Advisor and then Repeat...

You do not have to choose the perfect answer the first time, so you can go back and modify your choices

Iterative refinement will either

- Create a suitable and correct annotation proposal
- Conclude no viable sites are possible

Efficiently arriving at either answer is valuable

# Add Parallel Framework

**1. Survey Target**  
Where should I consider adding parallelism? Locate the loops and functions where your program [read more](#)

**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](#).

**4. Check Correctness**  
[Predict](#) parallel data sharing problems for the annotated tasks. [Fix](#) the reported sharing problems.

**5. Add Parallel Framework**  
Steps to replace annotations

## Summary of predicted parallel b...

**Potential program gain<sup>®</sup>: 1.12x (8 CPUs, Microsoft TPL Threading Model)**

These annotated parallel sites were detected:

Parallel Site	Maximum Site Gain <sup>®</sup>	Correctness Problems
Start Bodies ( <a href="#">nbodies.cs:103</a> )	1.33x	?

The most time-consuming (hot) functions found during Survey analysis appear below. Consider adding parallel site and task annotations around these functions so Suitability and Correctness can predict their parallel behavior.

Function	Source Location	CPU Time <sup>®</sup>
<a href="#">VTuneAmplifierXE::Examples::POTENTIAL::start</a>	<a href="#">potential.cs:62</a>	9.8229s
<a href="#">VTuneAmplifierXE::Examples::NBODIES::start</a>	<a href="#">nbodies.cs:106</a>	7.4341s

**Potential program gain<sup>®</sup>: 2.41x (8 CPUs, Microsoft TPL Threading Model)**

These annotated parallel sites were detected:

Parallel Site	Maximum Site Gain <sup>®</sup>	Correctness Problems
potential site ( <a href="#">potential.cs:61</a> )	7.64x	?
nbody site ( <a href="#">nbodies.cs:101</a> )	1.32x	?

The most time-consuming (hot) functions found during Survey analysis appear below. Consider adding parallel site and task annotations around these functions so Suitability and Correctness can predict their parallel behavior.

Function	Source Location	CPU Time <sup>®</sup>
<a href="#">&lt;&gt;c_DisplayClass1::&lt;ForWorker&gt;b_c</a>	?	10.9345s
<a href="#">VTuneAmplifierXE::Examples::NBODIES::start</a>	<a href="#">nbodies.cs:105</a>	7.3358s
<a href="#">VTuneAmplifierXE::Examples::POTENTIAL::computePot_st</a>	<a href="#">potential.cs:41</a>	2.8775s

**Collection Details.**

**Survey**

- Collection started: 21 March 2012, 6:01:26 PM
- Collection finished: 21 March 2012, 6:01:39 PM
- Elapsed time: 00 min 13 sec
- Collector Log: See log
- Application Output: See output
- Collector Command Line: See command line

# Summary

The Intel Advisor XE is a unique tool

- assists you to work smarter through detailed modeling
- guides you through the necessary steps
- leaves you in control of code and architectural choices
- lets you transform serial algorithms into parallel form faster

The parallel modeling methodology

- maintains your original application's semantics and behavior
- helps find the natural opportunities to exploit parallel execution



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

## Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

