



Intel® Advisor XE 2013

Threading Prototyping Tool



Code the Future

Additional Material

Intel® Advisor XE - Threading Prototyping Tool for Architects

Intel Advisor XE:

- [Product page](#) – overview, features, FAQs...
- [Training materials](#) – movies, tech briefs, documentation...
- [Evaluation guides](#) – step by step walk through
- [Case studies](#)
- [Support](#) – forums, secure support...

More Analysis Tools:

- [Intel® Inspector XE](#) - memory and thread checker / debugger
- [Intel® VTune™ Amplifier XE](#) – performance profiler

[Intel Software Development Products](#)

Data Driven Threading Design

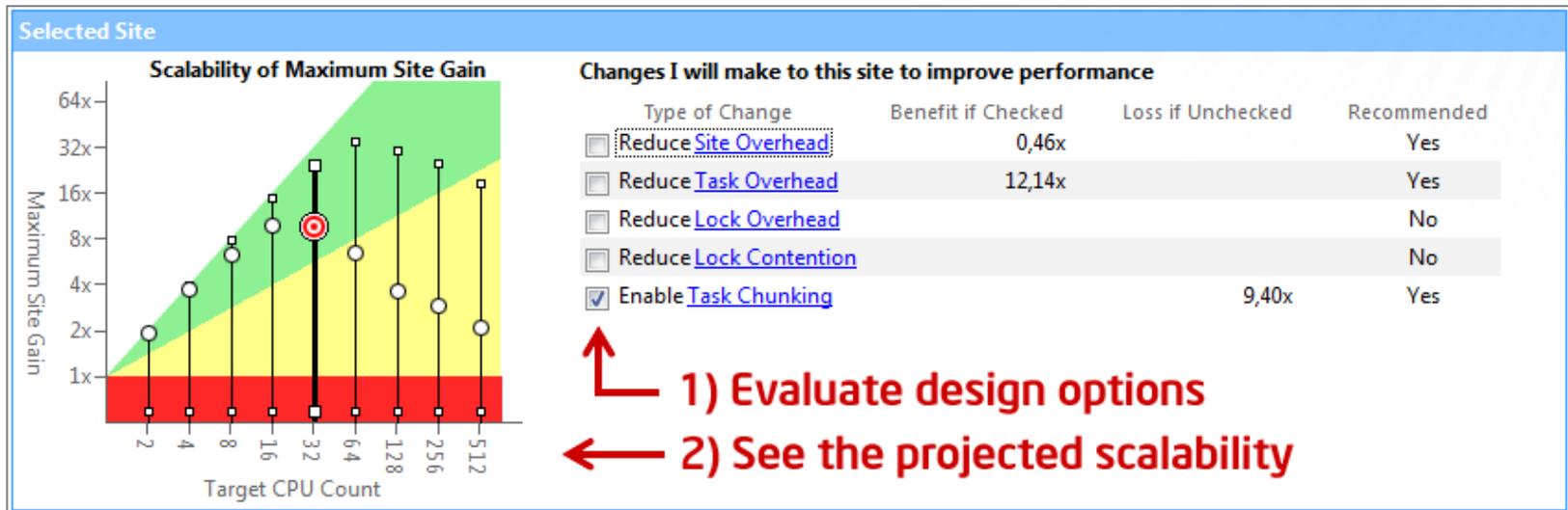
Intel® Advisor XE – Threading Prototyping Tool for Architects

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

Design Then Implement

Intel® Advisor XE 2013 – Threading Assistant

Design Parallelism

- No disruption to regular development
- All test cases continue to work
- Tune and debug the design before you implement it

1) Analyze it.

2) Design it.
(Compiler ignores these annotations.)

3) Tune it.

4) Check it.

5) Do it!

Implement Parallelism

Less Effort, Less Risk, More Impact

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

Amdahl's Law

(paraphrased) “The benefit from parallelism is limited by the computation which remains serial”

If you perfectly execute $\frac{1}{2}$ of your application in parallel you will achieve $< 2x$ speedup

The implication of this is that you must focus your attention where your application spends its time

Survey

Advisor XE Workflow

Benchmarks - e000 X potential_mt.cs kernel.cs potential.cs main.cs ThreadHelperClass.cs nbodies.cs potential_native_tpool.c

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.

Where should I add parallelism?

Intel Advisor XE 2013

Summary Survey Report Annotation Report Suitability Report Correctness Report

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Source Location
Total	100.0%	17.7508s	0s	
VTuneAmplifierXE::Examples::Benchmarks::Main	56.1%	9.9535s	0s	main.cs:17
VTuneAmplifierXE::Examples::kernel::startPotential	56.1%	9.9535s	0s	kernel.cs:32
VTuneAmplifierXE::Examples::POTENTIAL::start [loop]	56.1%	9.9535s	0s	potential.cs:62
VTuneAmplifierXE::Examples::POTENTIAL::start	56.1%	9.9535s	0s	potential.cs:63
VTuneAmplifierXE::Examples::POTENTIAL::computePot_st [loop]	51.9%	9.2164s	0s	potential.cs:39
VTuneAmplifierXE::Examples::POTENTIAL::computePot_st [loop]	51.9%	9.2164s	0s	potential.cs:41
VTuneAmplifierXE::Examples::POTENTIAL::computePot_st	4.2%	0.7371s	0.7371s	potential.cs:45
VTuneAmplifierXE::Examples::Benchmarks::Main	43.8%	7.7818s	0s	main.cs:16
VTuneAmplifierXE::Examples::kernel::startNBodies	43.8%	7.7818s	0s	kernel.cs:18
VTuneAmplifierXE::Examples::NBODIES::start [loop]	41.8%	7.4189s	0s	nbodies.cs:106
VTuneAmplifierXE::Examples::NBODIES::start	2.0%	0.3628s	0s	nbodies.cs:103
[Benchmarks.exe]	0.1%	0.0156s	0.0156s	

Find the places that are important to your application

Two Candidate loops

- 56%: POTENTIAL::start (loop)

Line	Source	Total Time	%	Loop Time	%
60					
61	for (int i = 0; i < constants.POT_ITERATION; i++)				
62	{				
63	potentialTotal = 0.0;			10.022s	
64	computePot_st();	10.012s			
65					
66	if (i % 10 == 0)				
67	Console.WriteLine("{0} - (Potential = {1:F5})", i, pote				
68					
69	updatePositions();	0.010s			
70	}				
71	}				
72					
Selected (Total Time):		0s			

- 41.8%: NBODIES::start (loop)

Line	Source	Total Time	%	Loop Time	%
96	public void start()				
97	{				
98	for (int i = 0; i < constants.NB_NUM_BODIES; i++)				
99	body[i] = new body();				
100					
101	// Loop over various sizes of the problem				
102	for (int n = 2; n <= constants.NB_NUM_BODIES; n *= 2)				
103	{				
104	startBodies(n);			7.451s	
105	runBodies(n);	7.451s			
106	}				
107	}				
108					

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

Add Annotation NBODIES::start (loop)

```
public void start()
{
    for (int i = 0; i < constants.NB_NUM_BODIES; i++)
        body[i] = new body();

    // Loop over various sizes of the problem
    for (int n = 2; n <= constants.NB_NUM_BODIES; n *= 2)
    {
        startBodies(n);
        runBodies(n);
    }
}
```

Generate Sequence Diagram...
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Go To Definition F12
Find All References
View Call Hierarchy Ctrl+K, Ctrl+T
Intel Advisor XE 2013
Breakpoint
Run To Cursor Ctrl+F10
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Outlining

Annotation Wizard...
Annotate Site
Annotate Task
Annotate Loop Site
Annotate Iteration Task
Annotation Definitions Reference

```
public void start()
{
    for (int i = 0; i < constants.NB_NUM_BODIES; i++)
        body[i] = new body();

    Annotate.SiteBegin( "nbody site" );
    // Loop over various sizes of the problem
    for (int n = 2; n <= constants.NB_NUM_BODIES; n *= 2)
    {
        Annotate.TaskBegin( "nbody tasks" );
        startBodies(n);
        runBodies(n);
        Annotate.TaskEnd();
    }
    Annotate.SiteEnd();
}
```

Add Annotation POTENTIAL::start (loop)

```
VTuneAmplifierXE.Examples.POTENTIAL start()
Annotate.SiteBegin( "potential site" );
for (int i = 0; i < constants.POT_ITERATION; i++)
{
    potentialTotal = 0.0;
    Annotate.TaskBegin( "potential task" );
    computePot_st();
    Annotate.TaskEnd();

    if (i % 10 == 0)
        Console.WriteLine("{0} - (Potential = {1:F5})", i, potentialTotal);

    updatePositions();
}
Annotate.SiteEnd();
```

Suitability – How Fast Will It Be?

What are the performance implications? Intel Advisor XE 2013

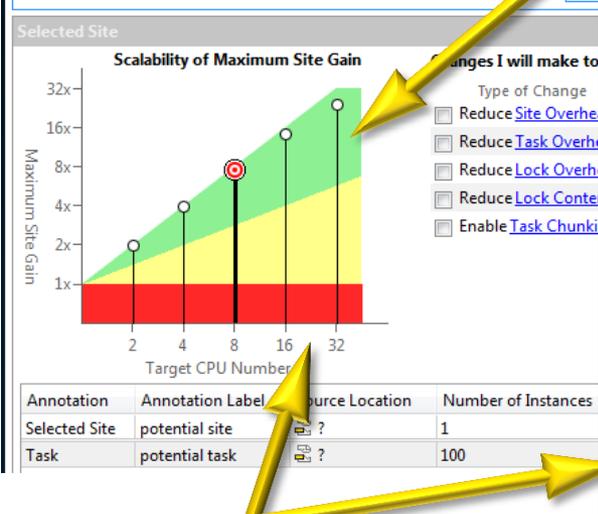
Summary Survey Report Correctness Report

Estimated Overall Speed-up

Maximum Program Gain For All Sites: 2.41x

Target CPU Number: 8 Threading Model: Microsoft TPL

Annotation Label	Source Location	Maximum Site Gain	Maximum Total Gain	Average Instance Time	Total Time
potential site	?	7.64x	1.92x	10.2441s	10.2441s



Scalability Graph

What are the performance implications? Intel Advisor XE 2013

Summary Survey Report Annotation Report

Recommended Improvement

Maximum Program Gain For All Sites: 2.41x

Target CPU Number: 8 Threading Model: Microsoft TPL

Annotation Label	Source Location	Maximum Site Gain	Maximum Total Gain	Average Instance Time	Total Time
nbody site	?	1.32x	1.12x	7.9730s	7.9730s
potential site	?	7.64x	1.92x	10.2441s	10.2441s

Selected Site

Scalability of Maximum Site Gain

Changes I will make to this site to improve performance

Type of Change	Benefit if Checked	Loss if Unchecked	Recommended
Reduce Site Overhead			No
Reduce Task Overhead			No
Reduce Lock Overhead			No
Reduce Lock Contention			No
Enable Task Chunking			No

Annotation	Annotation Label	Source Location	Number of Instances	Maximum Instance Time	Average Instance Time	Minimum Instance Time	Total Time
Selected Site	nbody site	?	1	7.9730s	7.9730s	7.9730s	7.9730s
Task	nbody tasks	?	7	6.0198s	1.1389s	0.0069s	7.9725s

Analyze the performance of your proposal

Correctness – Any Data Sharing Bugs?

Did the annotated tasks expose data sharing problems? Intel Advisor XE 2013

Summary Survey Report Annotation Report Suitability Report Correctness Report

ID	Problem	Site Name	Sources	Modules	State
P1	Parallel site information	nbody site	nbodies.cs	Benchmarks.exe	New
P2	Memory reuse	nbody site	body.cs; nbodies.cs	Benchmarks.exe	New
P3	Memory reuse	nbody site	nbodies.cs	Benchmarks.exe	New
P4	Memory reuse	nbody site	body.cs; nbodies.cs	Benchmarks.exe	New
P5	Memory reuse	nbody site	body.cs; nbodies.cs	Benchmarks.exe	New

4 Memory reuse conditions found!

Memory reuse: Code Locations

ID	Description	Source	Function	Module	State
X2	Allocation site	body.cs:12	VTuneAmplifierXE::Examples::body::ctor	Benchmarks.exe	New
X5	Parallel site	nbodies.cs:101	VTuneAmplifierXE::Examples::NBODIES::start	Benchmarks.exe	New
X6	Read	nbodies.cs:19	VTuneAmplifierXE::Examples::NBODIES::addAcc	Benchmarks.exe	New
X7	Read	nbodies.cs:20	VTuneAmplifierXE::Examples::NBODIES::addAcc	Benchmarks.exe	New
X8	Read	nbodies.cs:21	VTuneAmplifierXE::Examples::NBODIES::addAcc	Benchmarks.exe	New
X16	Write	nbodies.cs:84	VTuneAmplifierXE::Examples::NBODIES::startBodies	Benchmarks.exe	New

```
10 public body()
11 {
12     pos = new double[3];
13     vel = new double[3];
14     acc = new double[3];
}

99 body[i] = new body();
100
101 Annotate.SiteBegin( "nbody site" );
102 // Loop over various sizes of the problem
103 for (int n = 2; n <= constants.NB_NUM_BODIES; n *= 2)

82 {
83     for (int j = 0; j < 3; ++j)
84         body[i].pos[j] = rand.NextDouble() * Math.Abs(maxPos - minPos) + minPos;
85
86     for (int j = 0; j < 3; ++j)
```

Observations help identify problem

Sort By Item Name

Analyze your design for errors

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 behavior

Intel Advisor XE helps you choose an application to locate where it spends its time. Insert Intel Advisor XE annotations to identify your code's parallel behavior.

Potential program gain[®]: 1.12x (8 CPUs, Microsoft TPL Threading Model)

These annotated parallel sites were detected:

Parallel Site	Maximum Site Gain [®]	Correctness Problems
Start Bodies (nbodies.cs:103)	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 [®]
VTuneAmplifierXE::Examples::POTENTIAL::start	potential.cs:62	9.8229s
VTuneAmplifierXE::Examples::NBODIES::start	nbodies.cs:106	7.4341s

Potential program gain[®]: 2.41x (8 CPUs, Microsoft TPL Threading Model)

These annotated parallel sites were detected:

Parallel Site	Maximum Site Gain [®]	Correctness Problems
potential site (potential.cs:61)	7.64x	?
nbody site (nbodies.cs:101)	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 [®]
<>c_DisplayClass1::<ForWorker>b_c	?	10.9345s
VTuneAmplifierXE::Examples::NBODIES::start	nbodies.cs:105	7.3358s
VTuneAmplifierXE::Examples::POTENTIAL::computePot_st	potential.cs:41	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

Intel® Advisor XE

Available in these performance suites

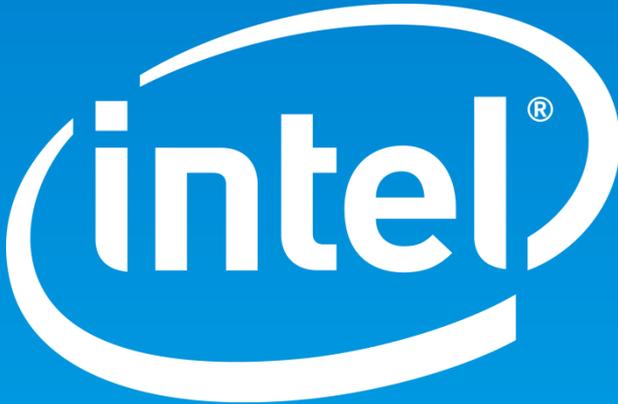
	Intel® Cluster Studio XE	Intel® Parallel Studio XE	
Analysis	●	●	Intel® VTune™ Amplifier XE - Performance Profiler
	●	●	Intel® Inspector XE - Memory & Thread Analyzer
	●	●	Static Analysis & Pointer Checker - Find Coding & Security Errors
	●	●	Intel® Advisor XE - Threading Prototyping Tool
	●		Intel® Trace Analyzer & Collector - MPI Optimizing Tool
Compilers & Libraries	●	●	Intel® Compiler - Optimizing Compiler for C, C++ and Fortran
	●	●	Intel® Integrated Performance Primitives[†] - Media and Data Optimizations
	●	●	Intel® Threading Building Blocks[†] - Parallelize Applications for Performance
	●	●	Intel® Math Kernel Library - High Performance Math
	●		Intel® MPI Library - Flexible, Efficient and Scalable Messaging

<http://software.intel.com/en-us/intel-advisor-xe>

† Available for C, C++ only

C, C++ only and Fortran only versions of Parallel Studio XE are also available.

Create fast, reliable code



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