Intel® Software Development Products

Modeling parallelism with Intel® Advisor XE

Levent Akyil
Software and Services Group

Executive Summary

Challenge: *Parallel programming can be rewarding but daunting!*

Intel® Advisor XE *is a methodology and set of tools to help you easily add correct and effective* ***parallelism*** *to your program*

Intel® Advisor XE *supports C/C++, Fortran on Windows/Linux and C# on Windows*

But why should you care about Parallelism?

In a word: "**Performance**"

Serial optimizations may achieve less than 25%

Data Parallelism, e.g., Vectorization may gain **2-4X**

Task Parallelism may provide speed-ups proportional to the number of cores, e.g., **4-8X**

Don't leave all that potential performance on the table!

Optimization Notice

Agenda

Introduction

Advisor Workflow

- **Survey**
- Add **Annotations**
- Model **Suitability**
- Check **Correctness**
- Add **Parallel Framework**

Conclusion

Suppose you had a magical tool that

- Lets you quickly write a serial program to implement your algorithm,

- Causes your program to run correctly *even in the presence of coding bugs,*

- Helps you find and fix the bugs,

- And also tells you the best performance to expect from your algorithm.

- Would this make you more productive? Of course it would!

Suppose you had a magical tool that…

- Lets you quickly write a serial program to implement your algorithm,
- Causes your program to run correctly *even in the presence of coding bugs,*
- Helps you find and fix the bugs,
- And also tells you the best performance to expect from your algorithm.
- Would this make you more productive? Of course it would!

This is similar to how Intel® Advisor XE works when you add Parallelism to your Serial program.

Intel® Advisor XE

Advisor XE is a toolset

- design tool that assists in making good decisions to transform a serial algorithm to use multi-core hardware

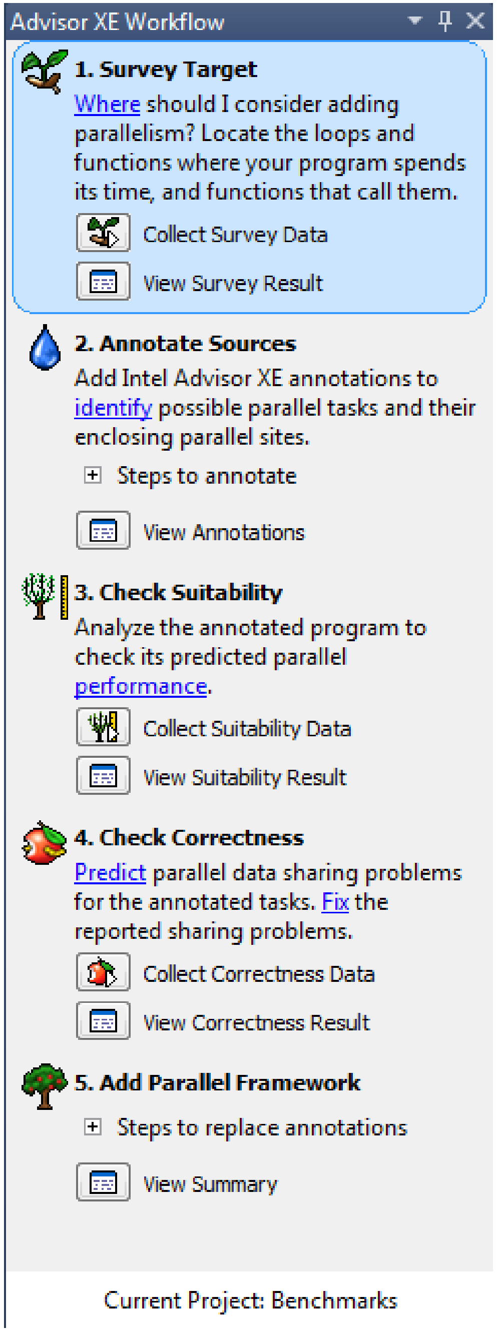
- parallel modeling tool that forecasts what might happen ***if*** that code were to execute in parallel
  - uses annotations in the serial code to calculate what

- A methodology and workflow to educate users on an effective method of using parallel programming

Intel® Advisor XE
Introduction

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 & benefits
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

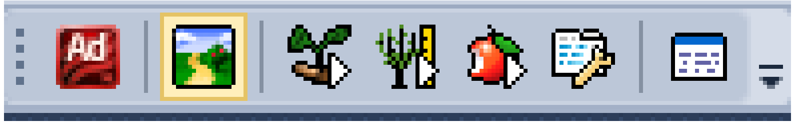
Current Project: Benchmarks

Intel® Advisor XE
Introduction

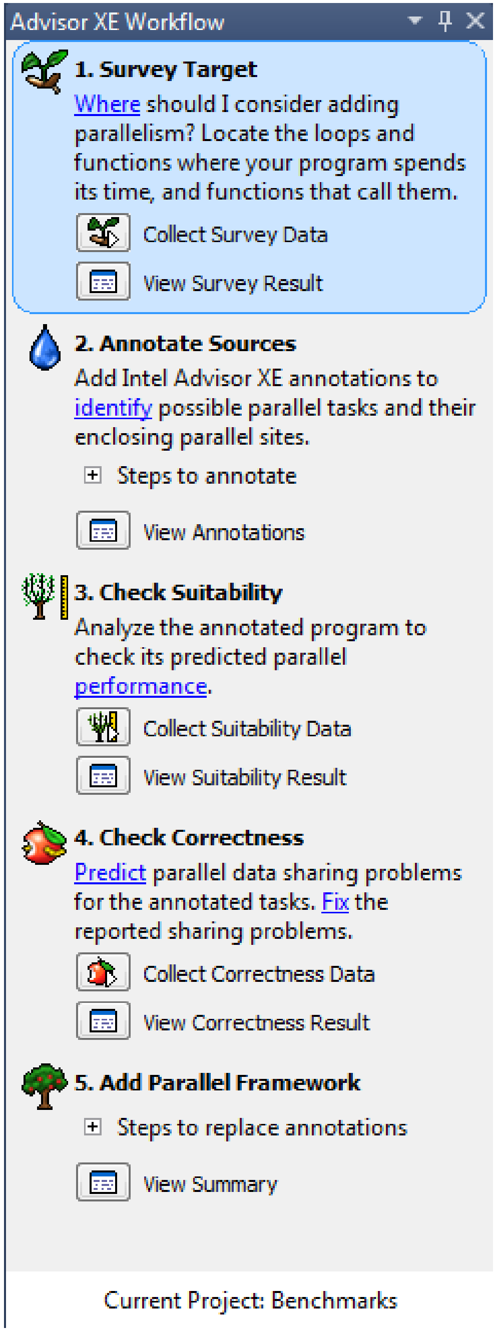
- Advisor XE guides you through these 5 steps, providing assisting tools
  - No auto-parallelization

- Model & evaluate potential return of parallelization investments.

- On your serial program



(Advisor XE toolbar)



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.

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

Current Project: Benchmarks

Intel® Advisor XE

Advantages of Advisor XE modeling

Serial modeling benefits:

1. **Your application can’t fail due to bugs caused by incorrect parallel execution (it’s running serially)**

2. **You can easily experiment with several different proposals before committing to a specific implementation**

3. **All of your test suites should still pass when validating the correctness of your transformations**

But you still can use Advisor XE on partially or completely parallelized code.

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

Parallel Advisor vs Advisor XE "what's new"

	Parallel Advisor	Advisor XE
Name	Intel® Parallel Advisor	Intel® Advisor XE
Component of	Intel® Parallel Studio	Intel® Parallel Studio and Cluster Studio XE
Windows OS	Windows XP and later; Windows Vista is deprecated.	Windows XP and later (but no support for Vista). Windows XP is deprecated.
VS integration	VS 2005, 2008, 2010	VS 2008, 2010 and later
Linux OS	No	**Yes**
Languages	C/C++	**C/C++** **Fortran** **C# .NET (Windows only)**
Standalone GUI	No	**Yes (Windows and Linux)**
CLI	No	**Yes (Windows and Linux)**

Where should we begin?

You either start with a blank sheet of paper … or you don’t

Almost no one starts from a blank sheet of paper

- Useful for writing explicitly parallel kernels and skeletons, but difficult to use when migrating legacy applications
- All others have to first get their ideas organized, usually by expressing a serial algorithm, which you then need to figure out how to express using parallelism

Almost everyone is worried about how to improve something which already has demonstrated value

- if you need to parallelize it
- it can’t already be parallel
- so therefore it must be serial

This is the assumption of this talk

Do you really mean that?

1

```
for (int I = 0; I < N; ++I)
        A[I] = B[I] + C[I];
```

This loop is equivalent to:

```
I = 0;
if (! (I < N)) goto done;

A[0] = B[0] + C[0]; // I = 0

++I;
if (! (I < N)) goto done;

A[1] = B[1] + C[1]; // I = 1
…
     done:
```

2

```
for (int I = 0; I < N; ++I)
        Work( &A[I] );
```

This loop is equivalent to:

```
I = 0;
if (! (I < N)) goto done;

Work( &A[0] ); // I = 0

++I;
if (! (I < N)) goto done;

Work( &A[1] ); // I = 1
…
     done:
```

Or did you really mean this…

1

```
for (int I = 0; I < N; ++I)
        A[I] = B[I] + C[I];
```

```
A[0…N-1] = B[0…N-1] + C[0…N-1]
```

2

```
for (int I = 0; I < N; ++I)
        Work( &A[I] );
```

```
foreach X in A[0…N-1]
        Work( &X );
```

or even…

```
Work( &A[0…N-1] )
```

Digression: debugging

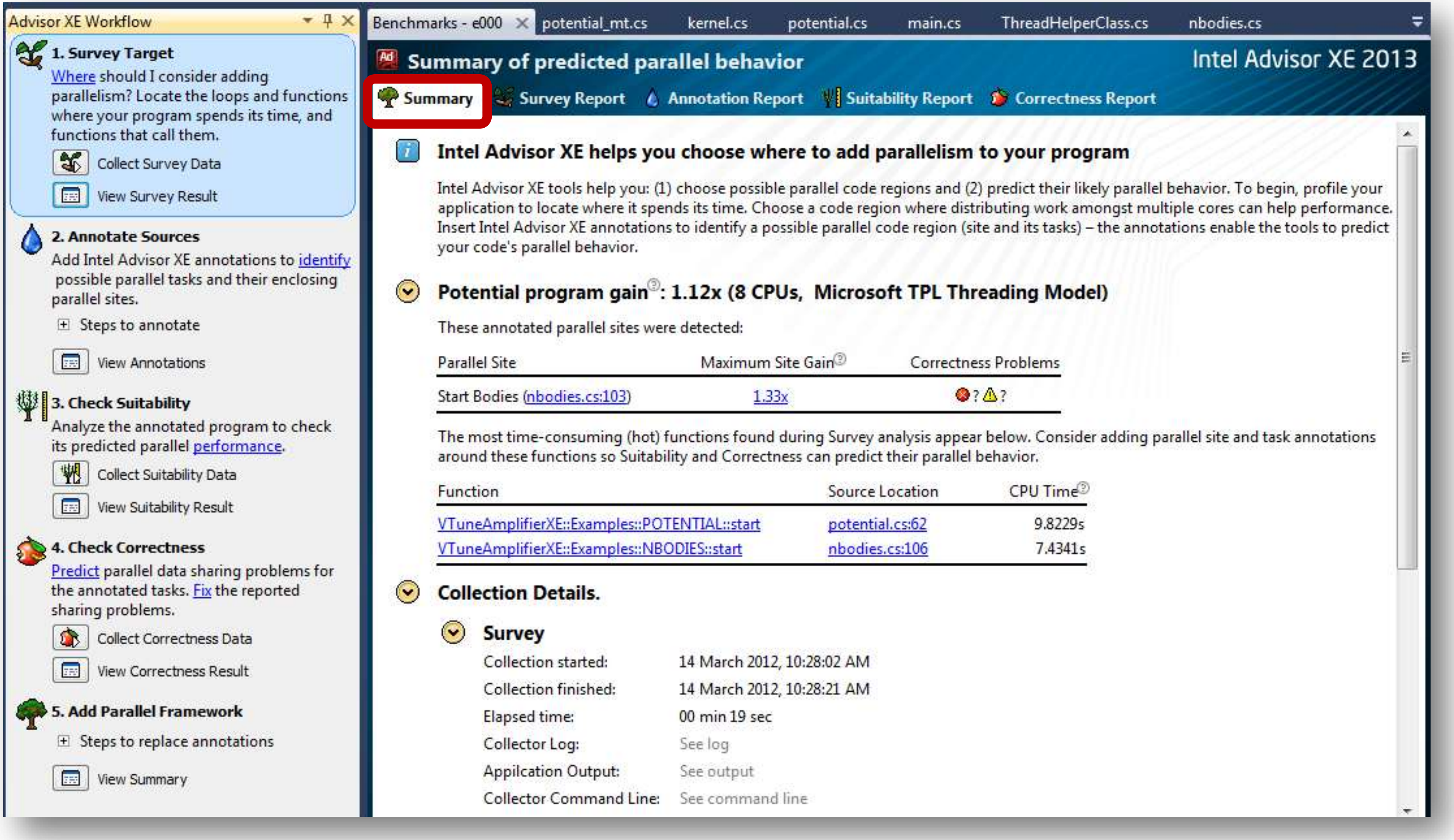
What is still claimed to be the #1 debugging tool in use today?

- A “print” statement

Inserting a “print” statement into your serial program typically does not change its behavior, but does allow you to observe what is happening

We can use this same approach in order to understand the “parallelism potential” present in your existing serial implementation

Step: 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.

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.

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 2013

Intel Advisor XE helps you choose where to add parallelism to your program

Intel Advisor XE tools help you: (1) choose possible parallel code regions and (2) predict their likely parallel behavior. To begin, profile your application to locate where it spends its time. Choose a code region where distributing work amongst multiple cores can help performance. Insert Intel Advisor XE annotations to identify a possible parallel code region (site and its tasks) – the annotations enable the tools to predict 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.

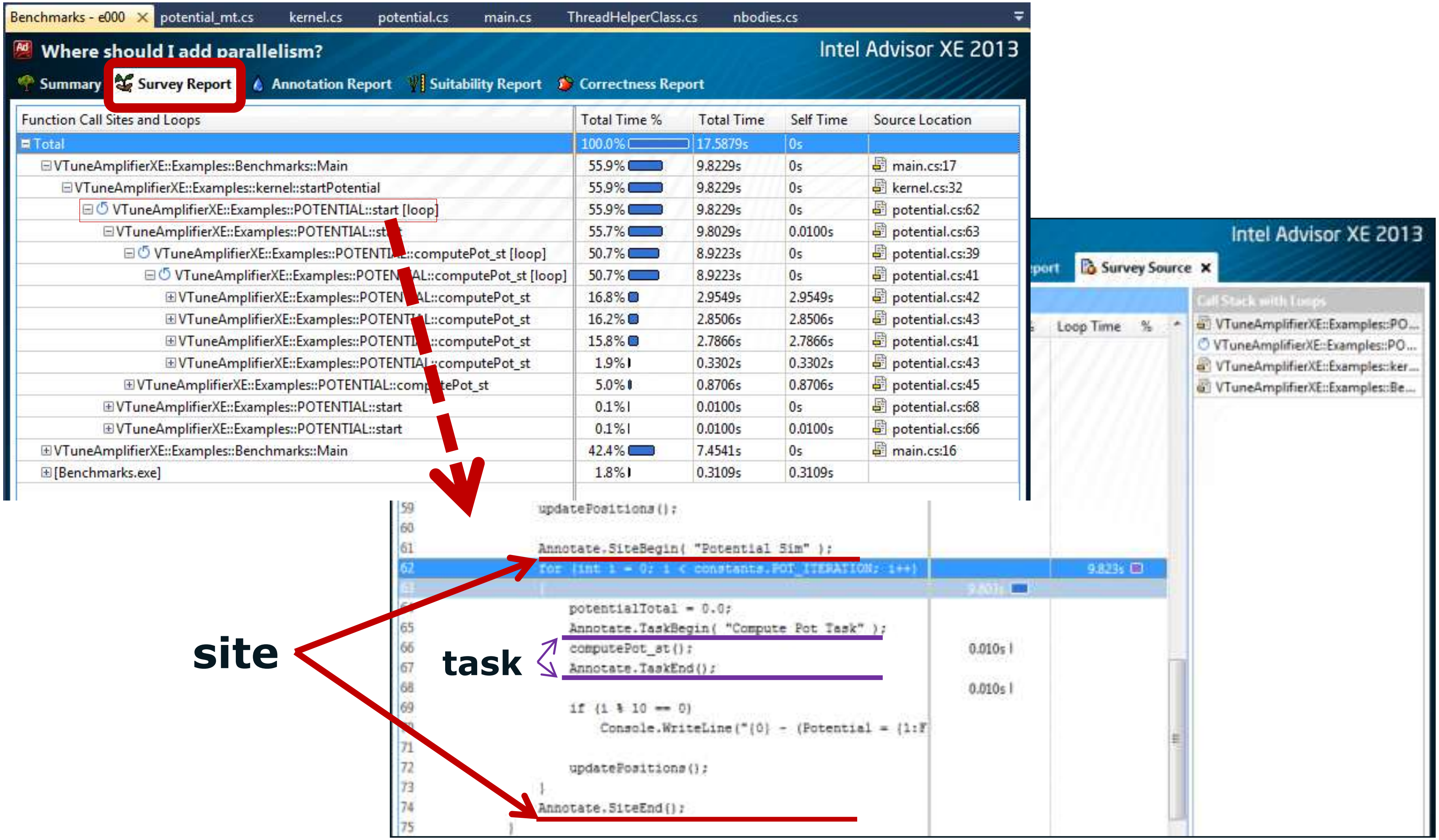
Function	Source Location	CPU Time
VTuneAmplifierXE::Examples::POTENTIAL::start	potential.cs:62	9.8229s
VTuneAmplifierXE::Examples::NBODIES::start	nbodies.cs:106	7.4341s

Collection Details.

Survey

Collection started:	14 March 2012, 10:28:02 AM
Collection finished:	14 March 2012, 10:28:21 AM
Elapsed time:	00 min 19 sec
Collector Log:	See log
Appilcation Output:	See output
Collector Command Line:	See command line

Step: Survey Target & Annotate Sources



Function Call Sites and Loops	Total Time %	Total Time	Self Time	Source Location
⊟ Total	100.0%	17.5879s	0s	
⊟ VTuneAmplifierXE::Examples::Benchmarks::Main	55.9%	9.8229s	0s	main.cs:17
⊟ VTuneAmplifierXE::Examples::kernel::startPotential	55.9%	9.8229s	0s	kernel.cs:32
⊟ VTuneAmplifierXE::Examples::POTENTIAL::start [loop]	55.9%	9.8229s	0s	potential.cs:62
⊟ VTuneAmplifierXE::Examples::POTENTIAL::st	55.7%	9.8029s	0.0100s	potential.cs:63
⊟ VTuneAmplifierXE::Examples::POTENTI[illegible]::computePot_st [loop]	50.7%	8.9223s	0s	potential.cs:39
⊟ VTuneAmplifierXE::Examples::POTEN[illegible]AL::computePot_st [loop]	50.7%	8.9223s	0s	potential.cs:41
⊞ VTuneAmplifierXE::Examples::POTEN[illegible]AL::computePot_st	16.8%	2.9549s	2.9549s	potential.cs:42
⊞ VTuneAmplifierXE::Examples::POTENT[illegible]L::computePot_st	16.2%	2.8506s	2.8506s	potential.cs:43
⊞ VTuneAmplifierXE::Examples::POTENTI[illegible]::computePot_st	15.8%	2.7866s	2.7866s	potential.cs:41
⊞ VTuneAmplifierXE::Examples::POTENTIAL::computePot_st	1.9%	0.3302s	0.3302s	potential.cs:43
⊞ VTuneAmplifierXE::Examples::POTENTIAL::comp[illegible]tePot_st	5.0%	0.8706s	0.8706s	potential.cs:45
⊞ VTuneAmplifierXE::Examples::POTENTIAL::start	0.1%	0.0100s	0s	potential.cs:68
⊞ VTuneAmplifierXE::Examples::POTENTIAL::start	0.1%	0.0100s	0.0100s	potential.cs:66
⊞ VTuneAmplifierXE::Examples::Benchmarks::Main	42.4%	7.4541s	0s	main.cs:16
⊞ [Benchmarks.exe]	1.8%	0.3109s	0.3109s	

- VTuneAmplifierXE::Examples::PO...
- VTuneAmplifierXE::Examples::PO...
- VTuneAmplifierXE::Examples::ker...
- VTuneAmplifierXE::Examples::Be...

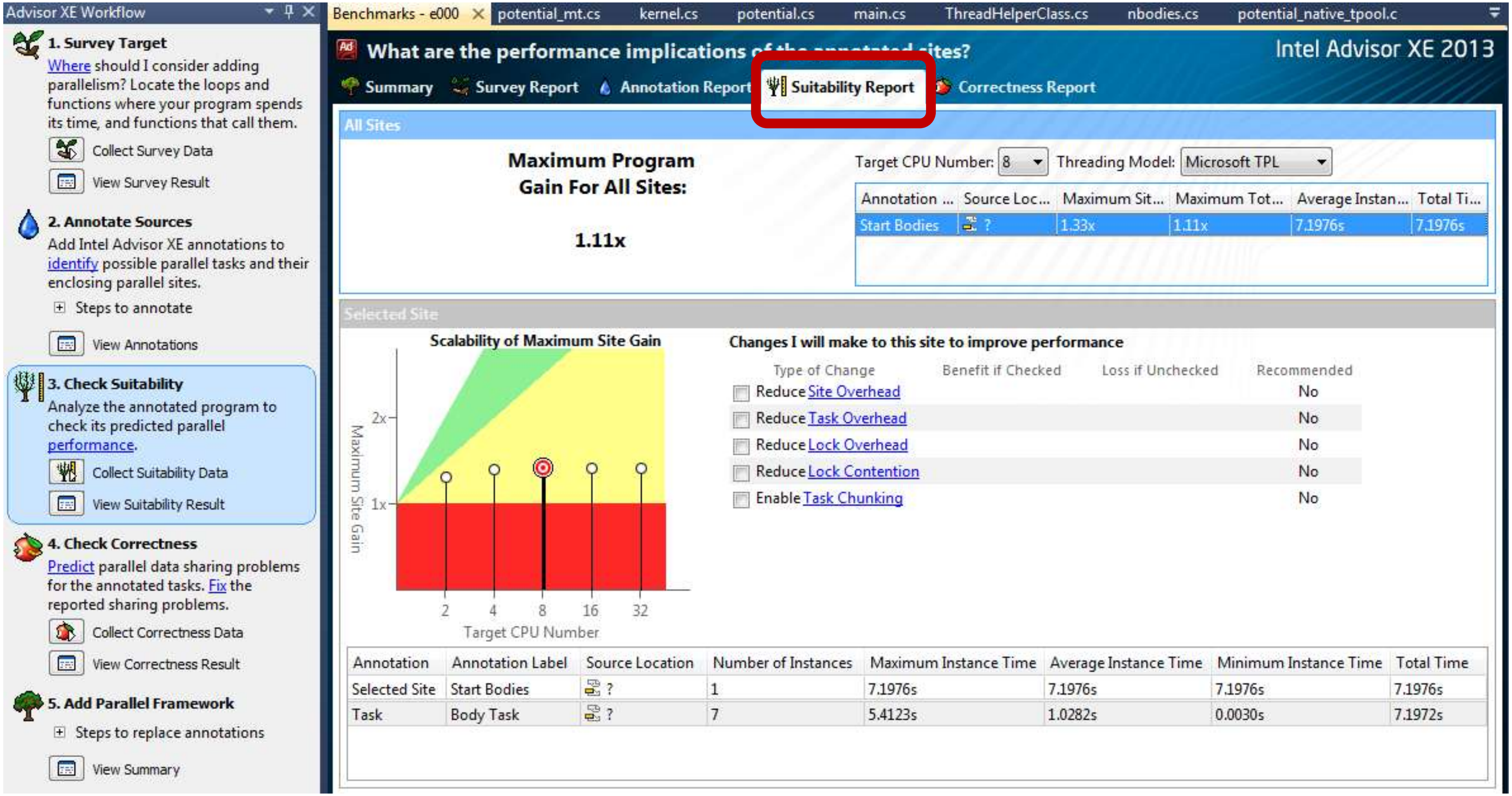
```
        updatePositions();

        Annotate.SiteBegin( "Potential Sim" );
        for (int i = 0; i < constants.POT_ITERATION; i++)          9.823s
        {                                                    9.803s
            potentialTotal = 0.0;
            Annotate.TaskBegin( "Compute Pot Task" );
            computePot_st();                                  0.010s
            Annotate.TaskEnd();
                                                              0.010s
            if (i % 10 == 0)
                Console.WriteLine("{0} - (Potential = {1:F

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

site

task

Optimization Notice

Step: Check Suitability



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

What are the performance implications of the annotated sites?

Maximum Program Gain For All Sites:

1.11x

Target CPU Number: 8

Threading Model: Microsoft TPL

Annotation ...	Source Loc...	Maximum Sit...	Maximum Tot...	Average Instan...	Total Ti...
Start Bodies	?	1.33x	1.11x	7.1976s	7.1976s

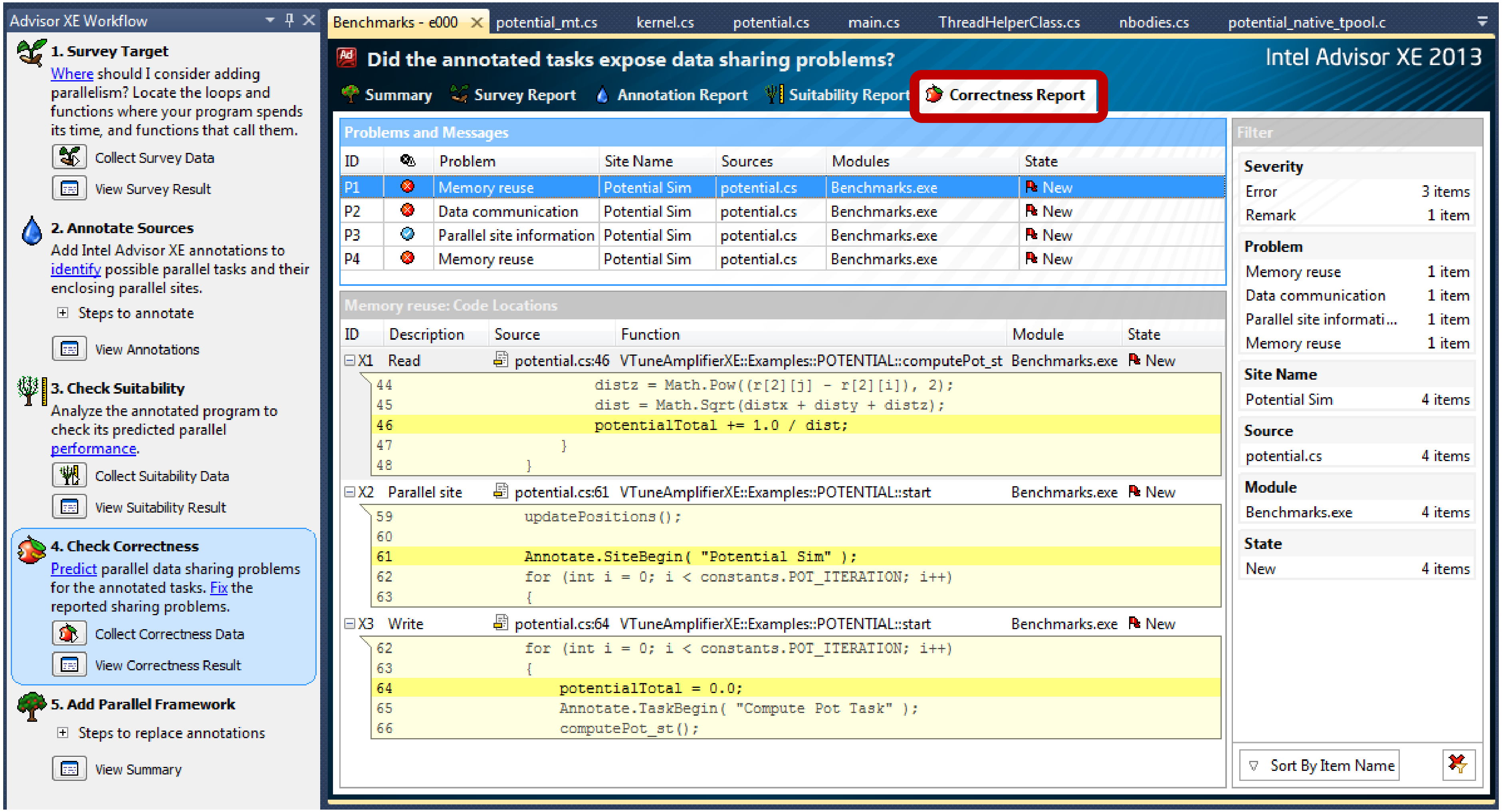
Changes I will make to this site to improve performance

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	Start Bodies	?	1	7.1976s	7.1976s	7.1976s	7.1976s
Task	Body Task	?	7	5.4123s	1.0282s	0.0030s	7.1972s

Optimization Notice

Step: Check Correctness



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

Did the annotated tasks expose data sharing problems?

Intel Advisor XE 2013

Problems and Messages

ID		Problem	Site Name	Sources	Modules	State
P1	●	Memory reuse	Potential Sim	potential.cs	Benchmarks.exe	New
P2	●	Data communication	Potential Sim	potential.cs	Benchmarks.exe	New
P3	○	Parallel site information	Potential Sim	potential.cs	Benchmarks.exe	New
P4	●	Memory reuse	Potential Sim	potential.cs	Benchmarks.exe	New

Memory reuse: Code Locations

ID	Description	Source	Function	Module	State
⊟X1	Read	potential.cs:46	VTuneAmplifierXE::Examples::POTENTIAL::computePot_st	Benchmarks.exe	New

```
44                  distz = Math.Pow((r[2][j] - r[2][i]), 2);
45                  dist = Math.Sqrt(distx + disty + distz);
46                  potentialTotal += 1.0 / dist;
47              }
48          }
```

ID	Description	Source	Function	Module	State
⊟X2	Parallel site	potential.cs:61	VTuneAmplifierXE::Examples::POTENTIAL::start	Benchmarks.exe	New

```
59          updatePositions();
60
61          Annotate.SiteBegin( "Potential Sim" );
62          for (int i = 0; i < constants.POT_ITERATION; i++)
63          {
```

ID	Description	Source	Function	Module	State
⊟X3	Write	potential.cs:64	VTuneAmplifierXE::Examples::POTENTIAL::start	Benchmarks.exe	New

```
62          for (int i = 0; i < constants.POT_ITERATION; i++)
63          {
64              potentialTotal = 0.0;
65              Annotate.TaskBegin( "Compute Pot Task" );
66              computePot_st();
```

Filter

Severity	
Error	3 items
Remark	1 item
Problem	
Memory reuse	1 item
Data communication	1 item
Parallel site informati...	1 item
Memory reuse	1 item
Site Name	
Potential Sim	4 items
Source	
potential.cs	4 items
Module	
Benchmarks.exe	4 items
State	
New	4 items

Optimization Notice

Case Study - Demo

Advisor Workflow – Case Study

- ***Survey***
- Add **Annotations**
- Model **Suitability**
- Check **Correctness**
- Add **Parallel Framework**

Conclusion

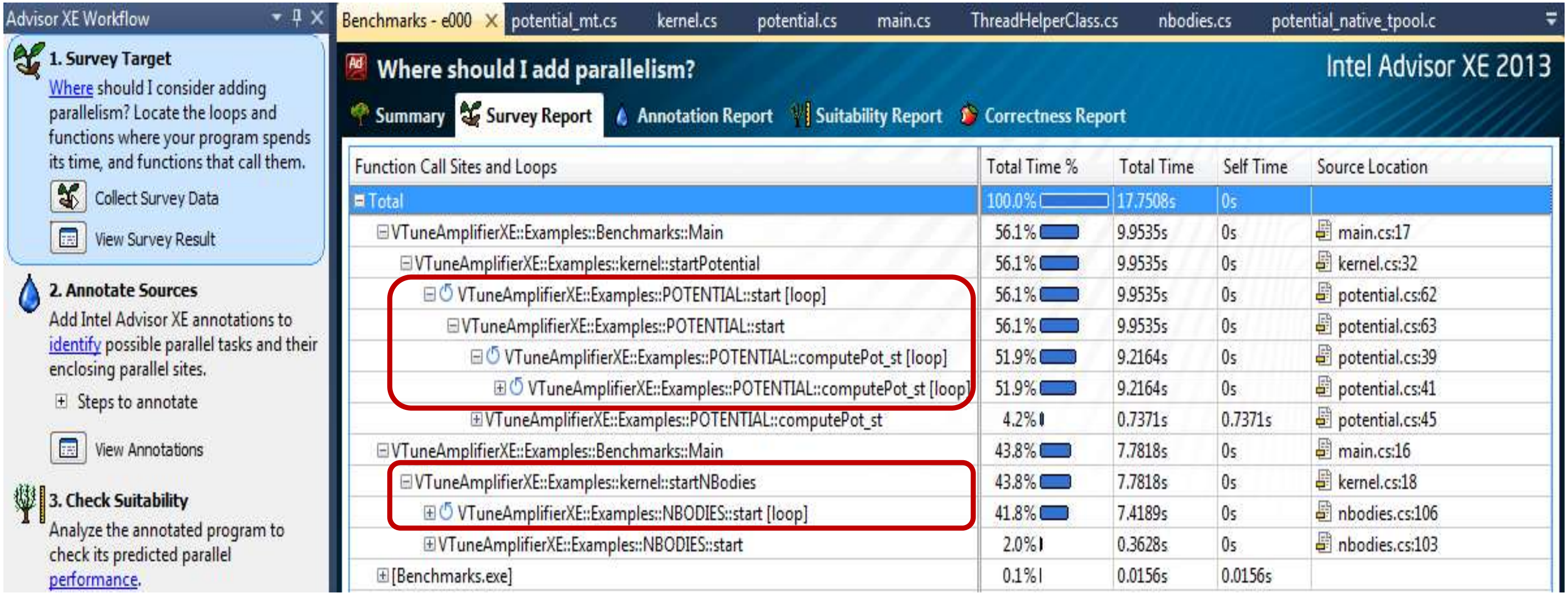
Amdahl's Law

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

If you perfectly execute ½ 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



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.

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.

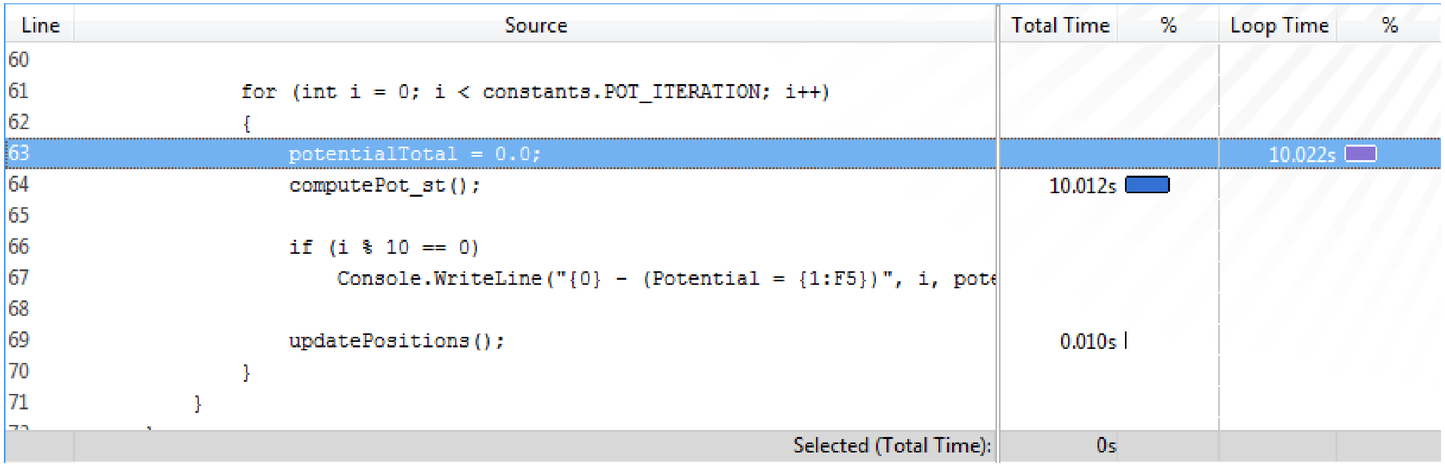
Where should I add parallelism?

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

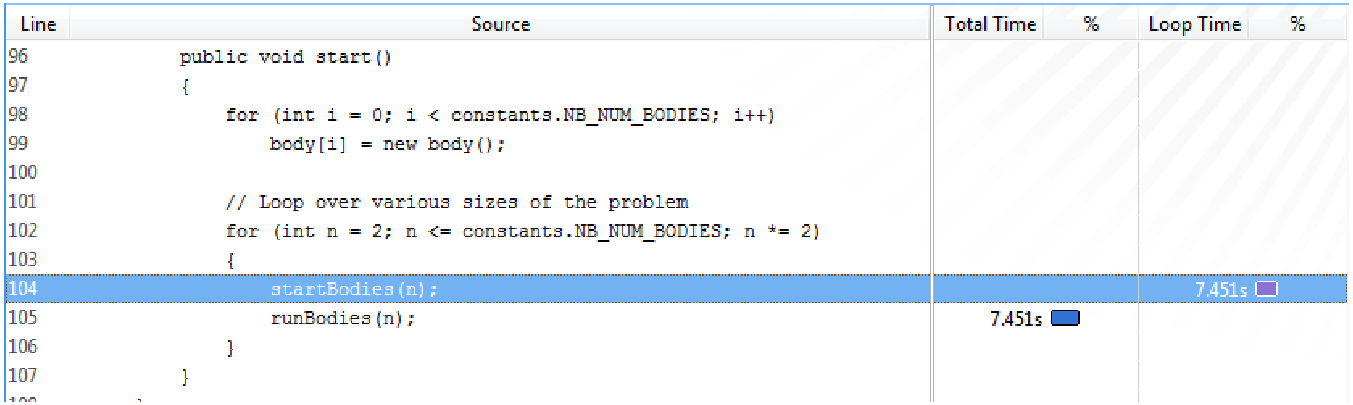
Two Candidate loops

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	}				
	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	}				

Optimization Notice

Agenda

Advisor Workflow – Case Study

- Survey
- *Add **Annotations***
- Model **Suitability**
- Check **Correctness**
- Add **Parallel Framework**

Conclusion

Advisor XE Annotation Concepts

Advisor uses 3 primary concepts to create a model

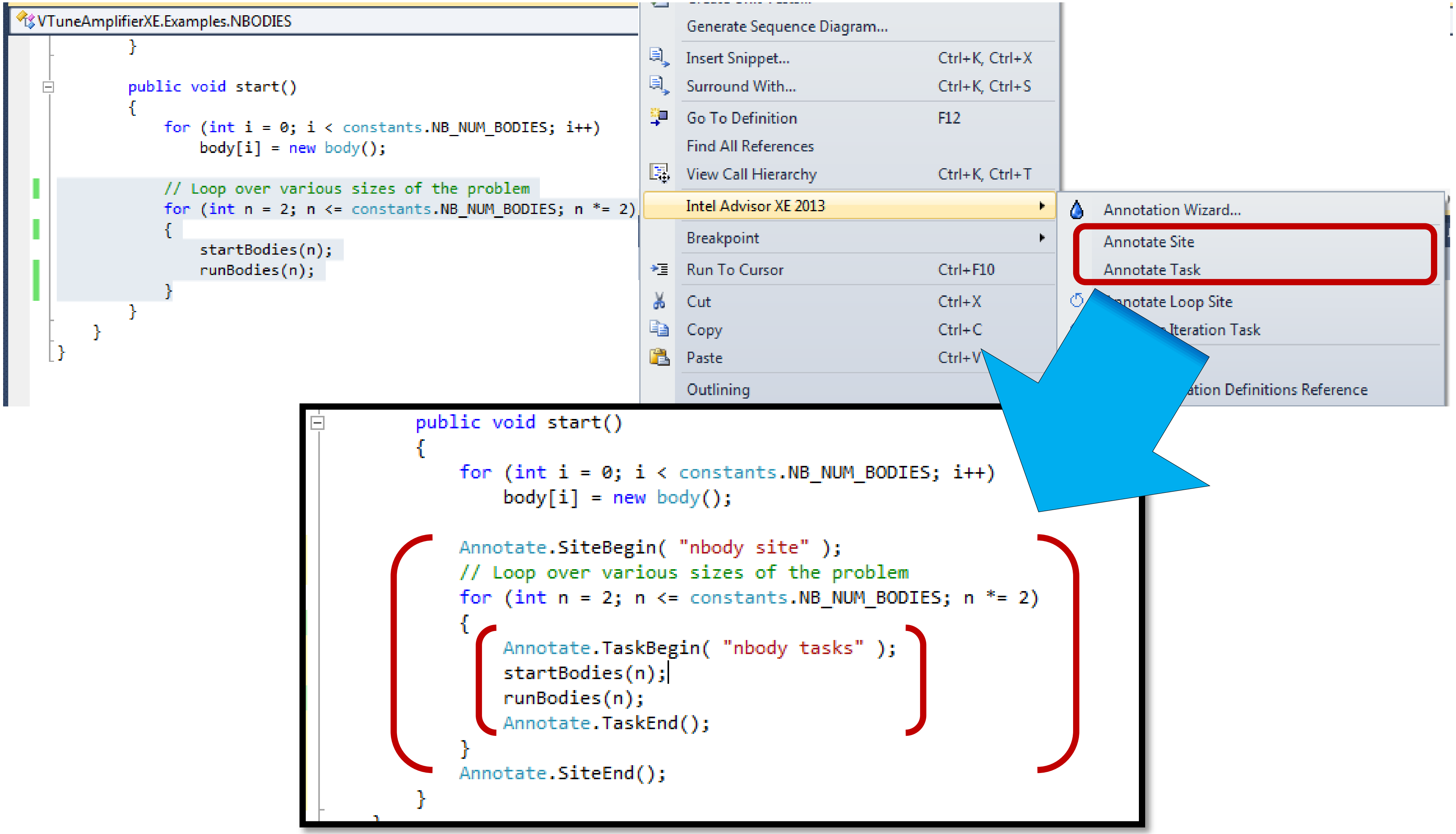
- **SITE**
  - A region of code in your application you want to transform into parallel code
- **TASK**
  - The region of code in a SITE you want to execute in parallel with the rest of the code in the SITE
- **LOCK**
  - Mark regions of code in a TASK which must be serialized

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)

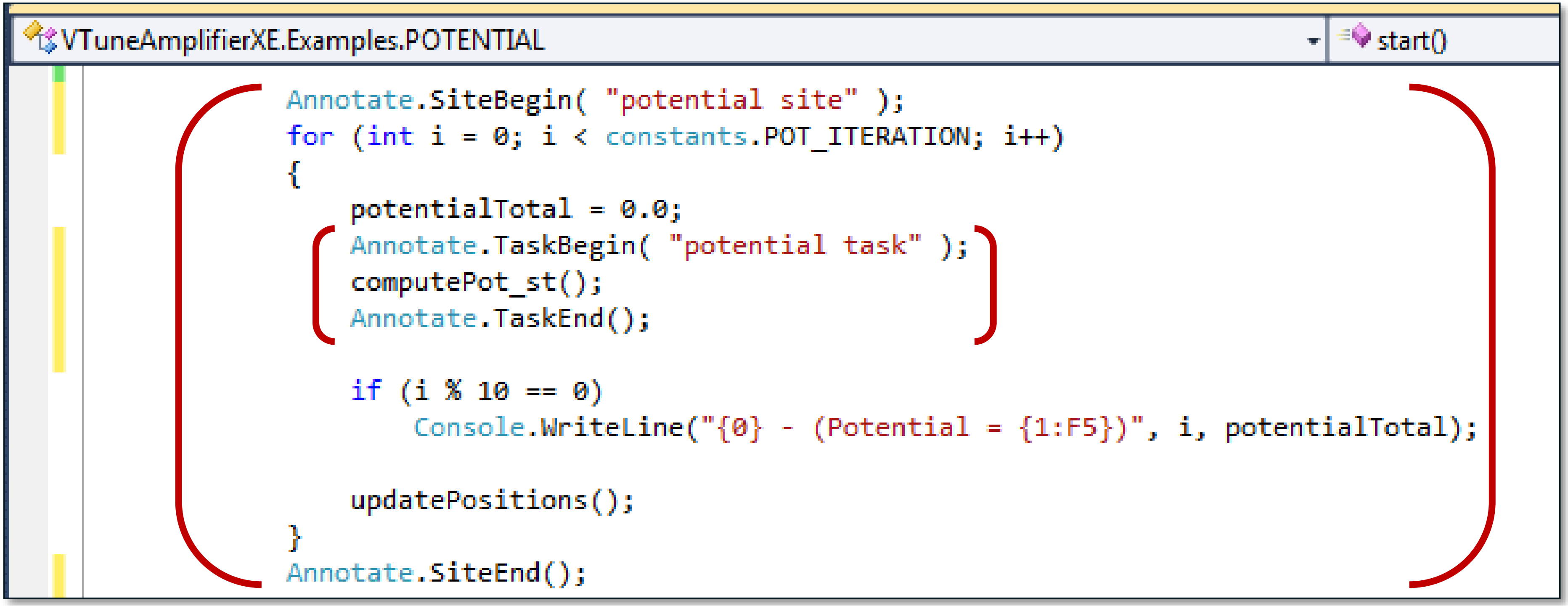


VTuneAmplifierXE.Examples.NBODIES

```
        }

        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);
            }
        }
    }
}
```

```
        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)



```
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();
```

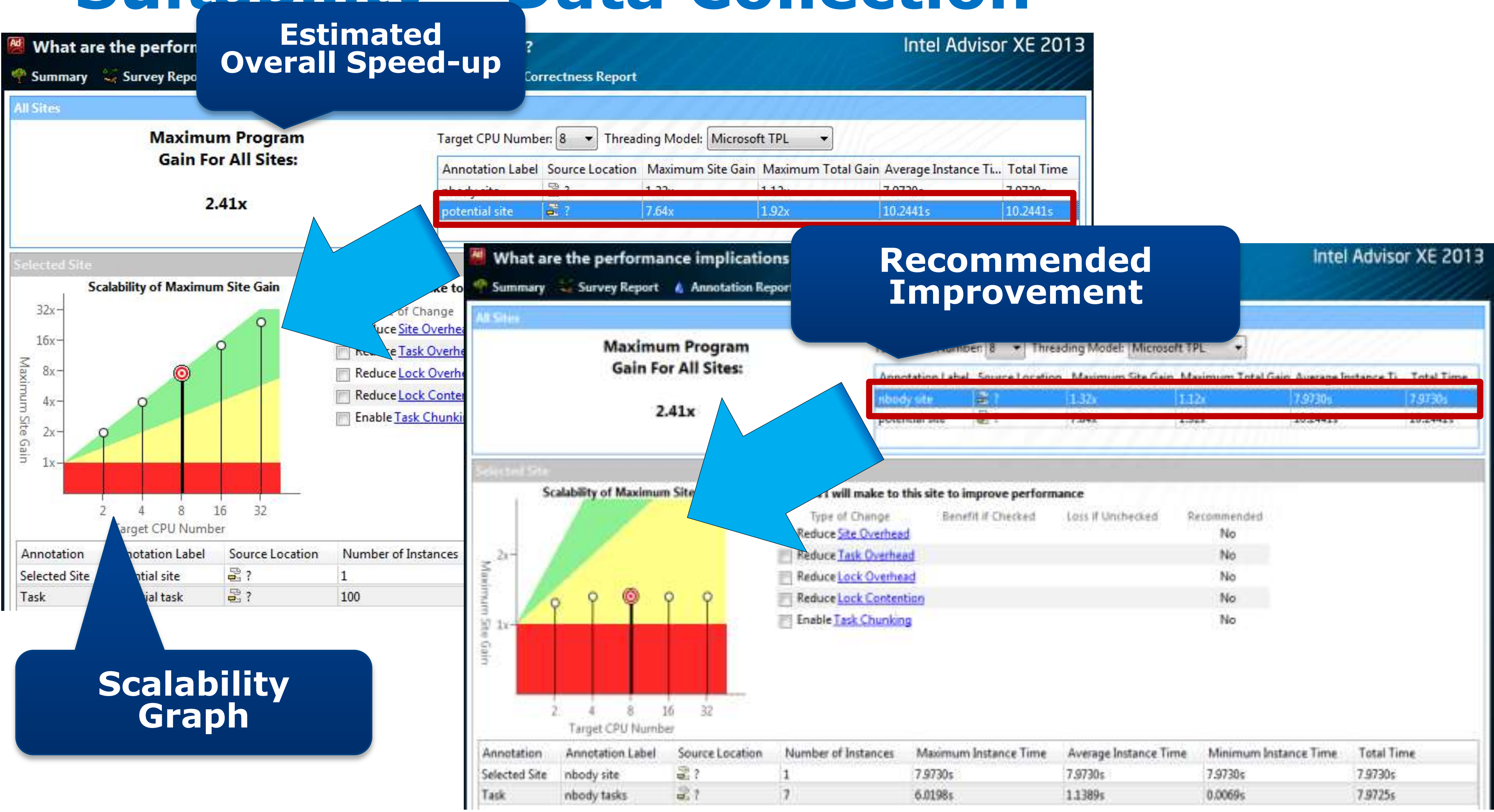
Agenda

Advisor Workflow – Case Study

- Survey
- Add Annotations
- *Model* ***Suitability***
- Check **Correctness**
- Add **Parallel Framework**

Conclusion

Suitability - Data Collection



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 Ti...	Total Time
potential site	?	7.64x	1.92x	10.2441s	10.2441s

Annotation	Annotation Label	Source Location	Number of Instances
Selected Site	ntial site	?	1
Task	ial task	?	100

Maximum Program Gain For All Sites:

2.41x

Threading Model: Microsoft TPL

Annotation Label	Source Location	Maximum Site Gain	Maximum Total Gain	Average Instance Ti...	Total Time
nbody site	?	1.32x	1.12x	7.9730s	7.9730s

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 your proposal to see if you made a suitable choice

Optimization Notice

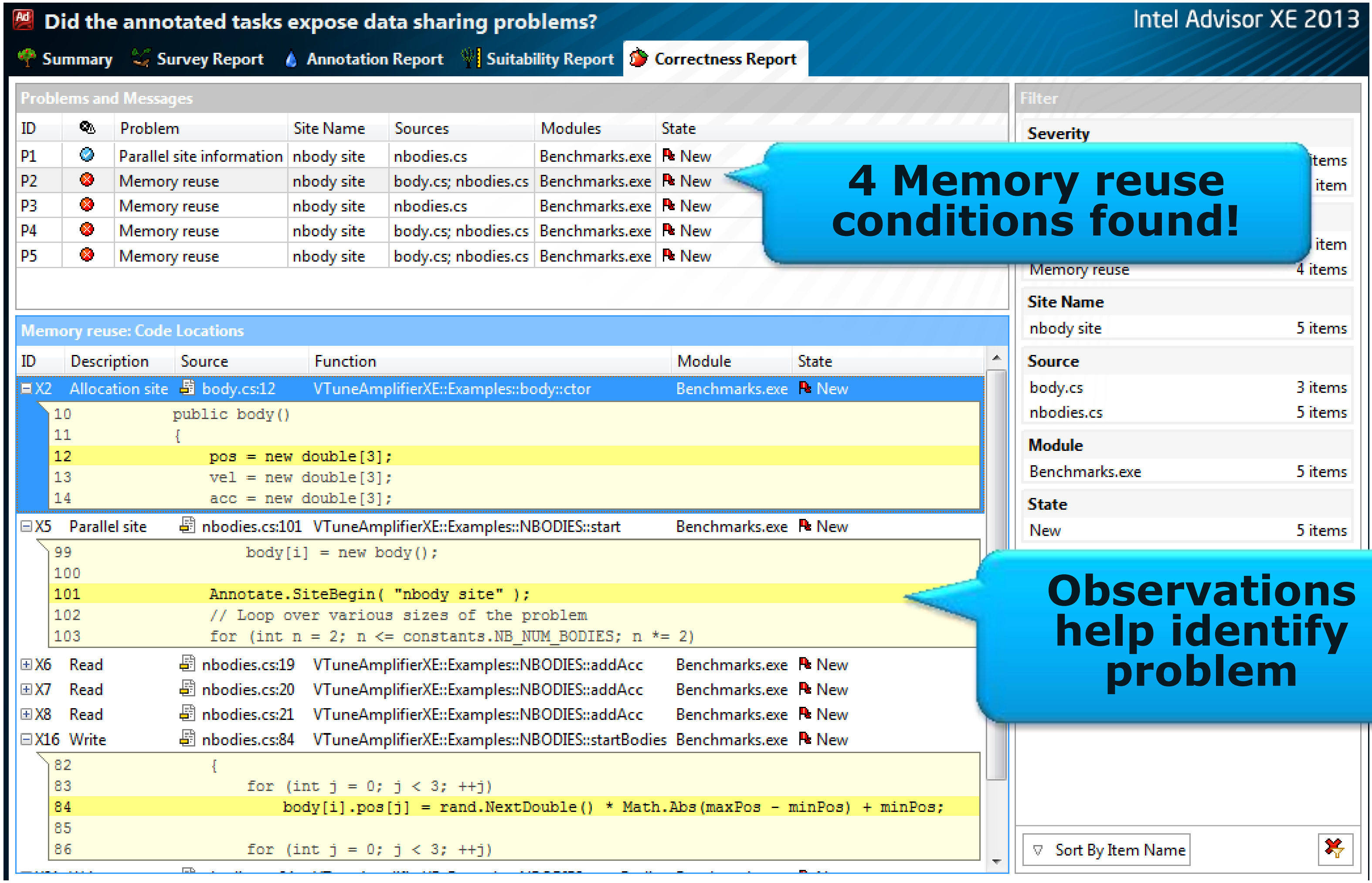
Agenda

Advisor Workflow – Case Study

- Survey
- Add Annotations
- Model Suitability
- *Check **Correctness***
- Add **Parallel Framework**

Conclusion

Correctness – Data Collection



Problems and Messages

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

```
        public body()
        {
            pos = new double[3];
            vel = new double[3];
            acc = new double[3];
```

ID	Description	Source	Function	Module	State
X5	Parallel site	nbodies.cs:101	VTuneAmplifierXE::Examples::NBODIES::start	Benchmarks.exe	New

```
                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)
```

ID	Description	Source	Function	Module	State
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

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

                for (int j = 0; j < 3; ++j)
```

Filter

Severity	
	items
	item
	item
Memory reuse	4 items

Site Name	
nbody site	5 items

Source	
body.cs	3 items
nbodies.cs	5 items

Module	
Benchmarks.exe	5 items

State	
New	5 items

Observations help identify problem

Analyze your annotations to see if you made a correct choice

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

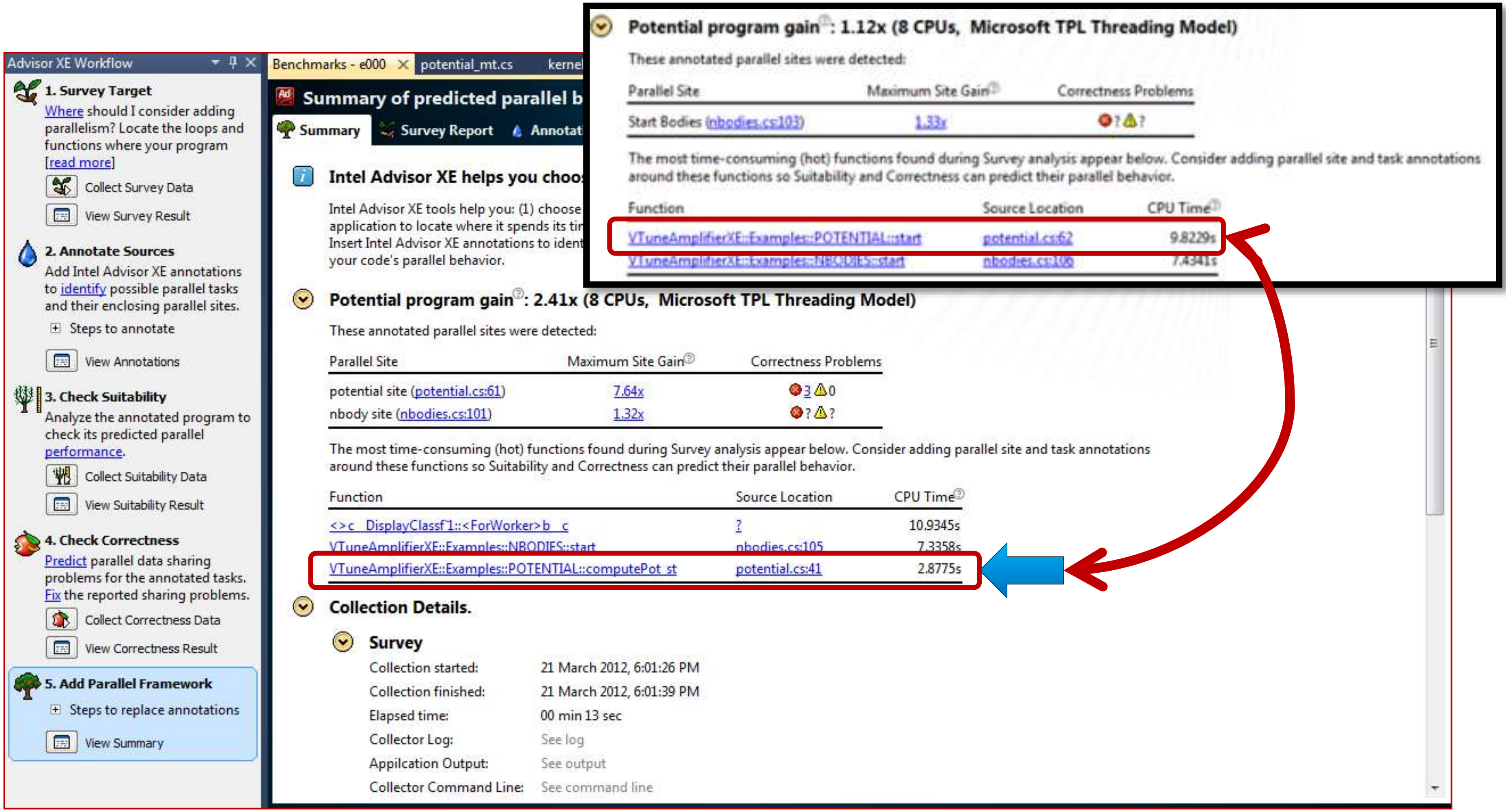
Agenda

Advisor Workflow – Case Study

- Survey
- Add Annotations
- Model Suitability
- Check Correctness
- *Add **Parallel Framework***

Conclusion

Add Parallel Framework



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

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

Summary of predicted parallel b

Intel Advisor XE helps you choo

Intel Advisor XE tools help you: (1) choose application to locate where it spends its tir Insert Intel Advisor XE annotations to ident your code's parallel behavior.

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	● 3 ⚠ 0
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__DisplayClassf`1::<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
Appilcation Output:	See output
Collector Command Line:	See command line

Agenda

Advisor Workflow – Case Study

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

Conclusion

Optimization Notice

Summary

The Intel Advisor XE is a unique tool

- assists you to work smarter though detailed modeling
- guides you through the necessary steps
- leaves you in full control of your 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® Parallel Studio XE

- Intel® Parallel Studio XE 2013 beta started! Join beta!

www.intel.com/go/parallel

Optimization Notice

intel®
Software

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.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

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