

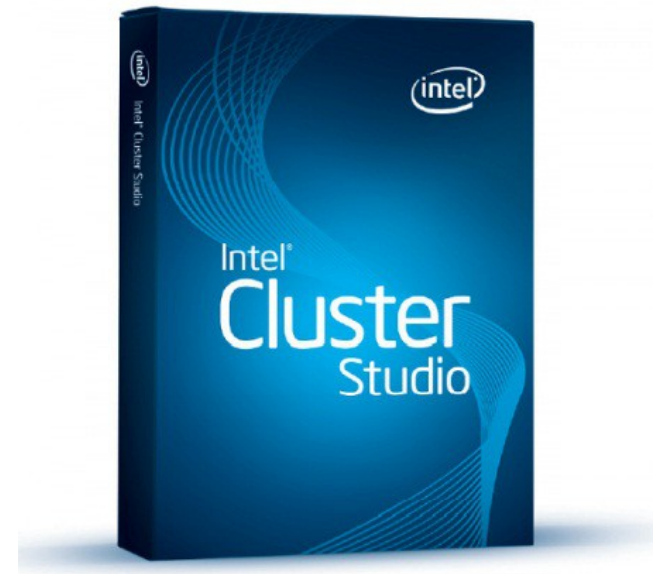
Intel Cluster Studio

HiPerCH April 2013



TECHNISCHE
UNIVERSITÄT
DARMSTADT

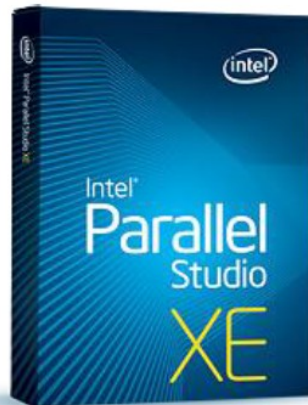
Michael Burger
FG Scientific Computing
TU Darmstadt
michael.burger@sc.tu-darmstadt.de



- What is the Intel Cluster Studio?
- Planning with help of Intel Advisor
- Optimizations with the help of Intel Compiler
- Searching errors with Intel Inspector
- Finding MPI problems: Intel Trace Analyzer
- Analyzing code with Vtune Amplifier
- Summary

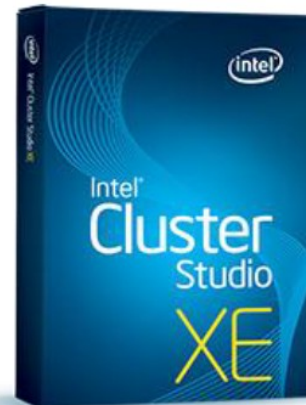
What is the Intel Cluster Studio?

- Collection of different tools
- Should cover the hole software development process
- Different packages for different platforms



Advanced Performance

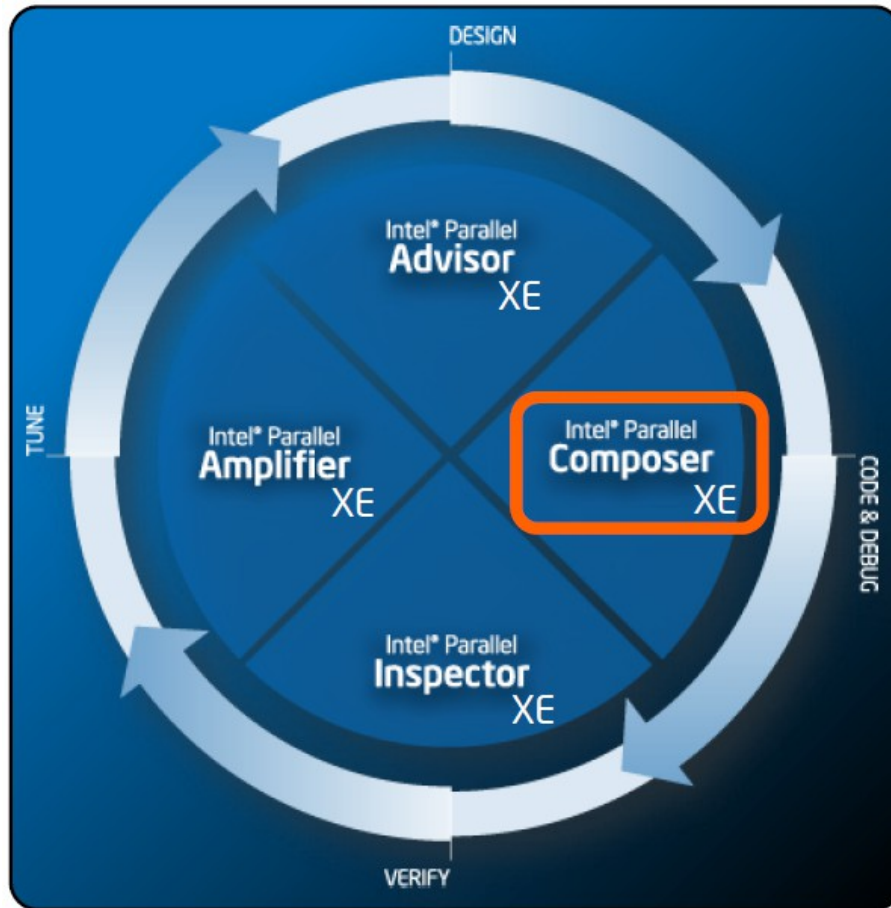
C++ and Fortran Compilers, MKL Libraries & Analysis Tools for Windows*, Linux* developers on IA based multi-core and many-core nodes



Distributed Performance

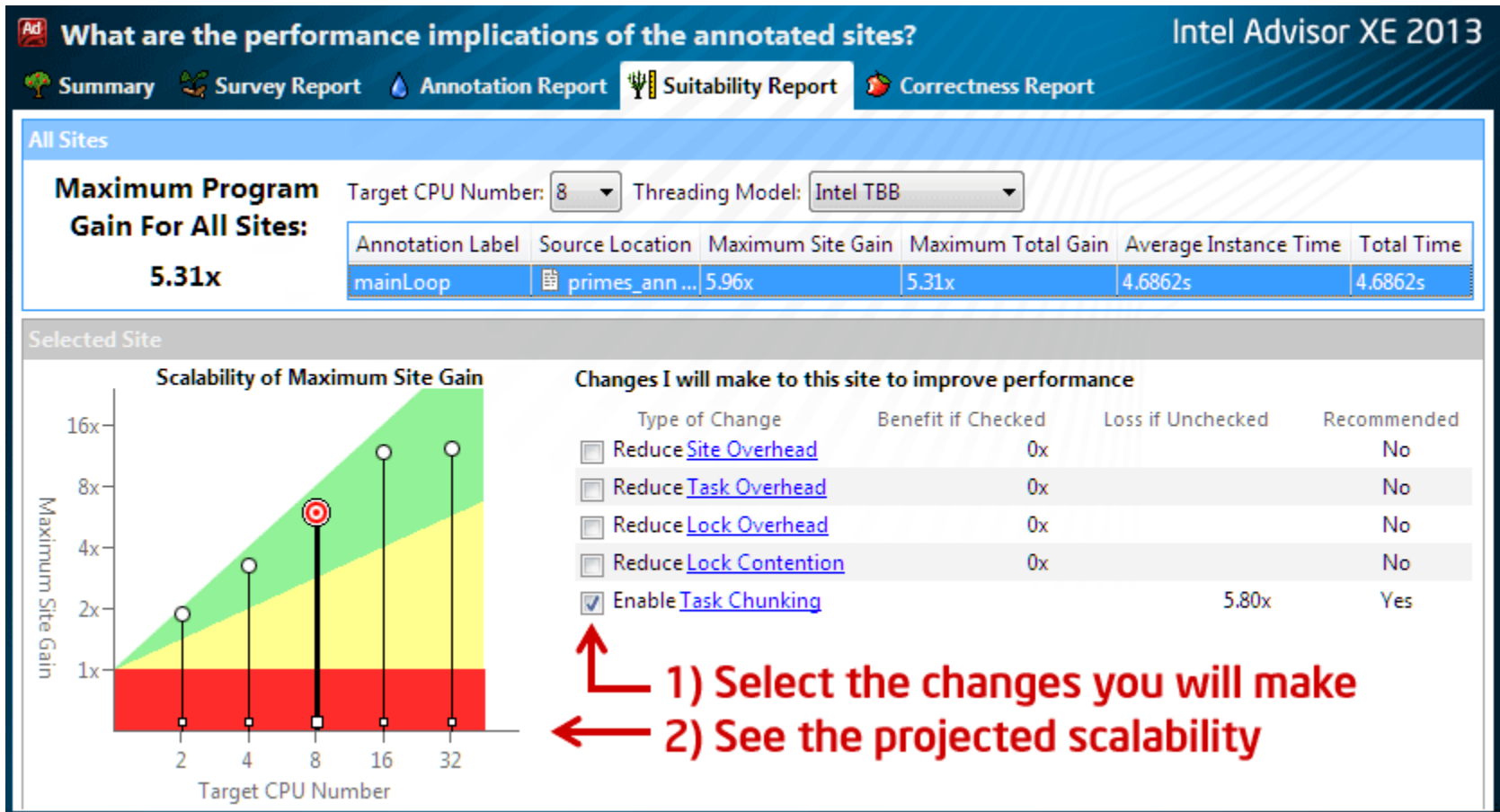
MPI Cluster Tools with C++ and Fortran Compiler, MKL Libraries and Analysis Tools for Windows*, Linux* developers on IA based clusters

What is the Intel Cluster Studio?

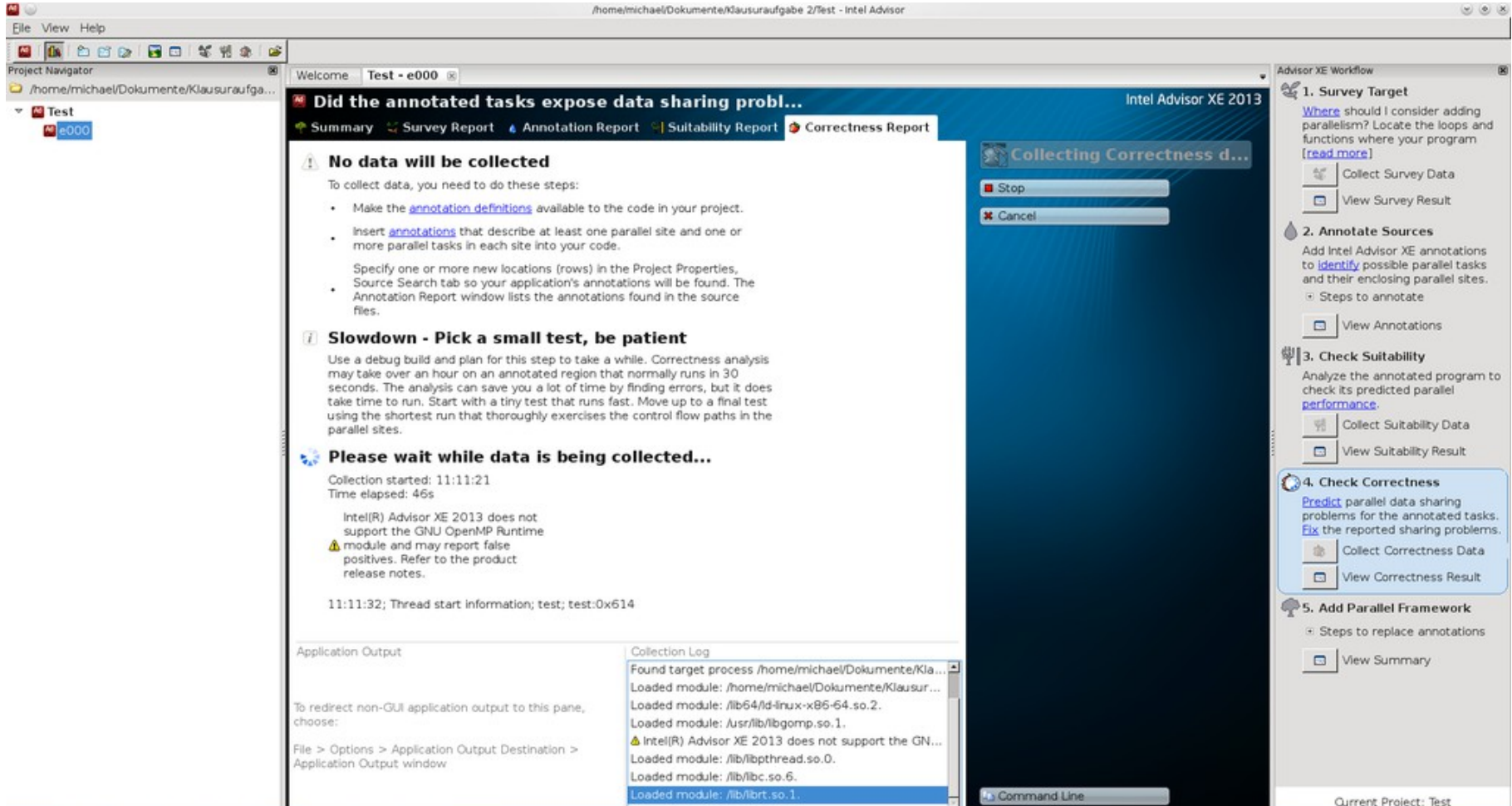


- **Advisor:** Help to parallelize code
- **Composer:** No IDE, compiler, libraries
- **Inspector:** Correctnesschecking
- **Analyzer:** Correctnesschecking (MPI)
- **Amplifier:** Parallel/serial tuning

- Threading assistant for C (C++/#) and Fortran
- Should lead through the process of designing software
- Parallelize existing code
- Compare different alternatives before implementing it
- Help in finding locks and points for synchronization
- Short demo in a few moments



www.intel.com



The screenshot shows the Intel Advisor XE 2013 interface. The main window displays a warning: "No data will be collected". Below this, it explains that to collect data, several steps must be followed, including making annotation definitions available, inserting annotations, and specifying new locations in the Project Properties. A "Slowdown - Pick a small test, be patient" section advises using a debug build and a small test. A "Please wait while data is being collected..." section shows the collection started at 11:11:21 and elapsed 46s. It also notes that Intel(R) Advisor XE 2013 does not support the GNU OpenMP Runtime module and may report false positives. The "Collection Log" at the bottom shows the target process and loaded modules.

Did the annotated tasks expose data sharing probl...

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

No data will be collected

To collect data, you need to do these steps:

- Make the [annotation definitions](#) available to the code in your project.
- Insert [annotations](#) that describe at least one parallel site and one or more parallel tasks in each site into your code.
- Specify one or more new locations (rows) in the Project Properties, Source Search tab so your application's annotations will be found. The Annotation Report window lists the annotations found in the source files.

Slowdown - Pick a small test, be patient

Use a debug build and plan for this step to take a while. Correctness analysis may take over an hour on an annotated region that normally runs in 30 seconds. The analysis can save you a lot of time by finding errors, but it does take time to run. Start with a tiny test that runs fast. Move up to a final test using the shortest run that thoroughly exercises the control flow paths in the parallel sites.

Please wait while data is being collected...

Collection started: 11:11:21
Time elapsed: 46s

Intel(R) Advisor XE 2013 does not support the GNU OpenMP Runtime module and may report false positives. Refer to the product release notes.

11:11:32; Thread start information; test; test:0x614

Collecting Correctness data...

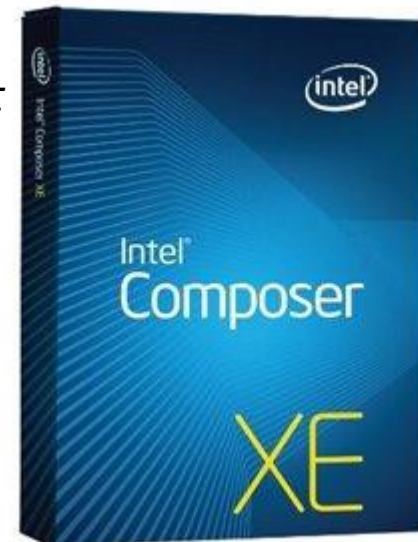
Stop | Cancel

Advisor XE Workflow

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

- Contains compiler (icc/fort)
- „Special“ support for Intel processors
- *„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. 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.“*



Intel (Parallel) Composer

- Support of C++ and Fortran. Additionally contains:
 - Template Library: Intel Threading Building Blocks (TBB)
 - Library: Intel Integrated Performance Primitives (IPP)
 - Library: Intel Math Kernel Libraries (MKL)
 - OpenMP / Intel MPI Support

Intel® Threading Building Blocks

Widely used C++ template library for parallelism

*Open Sourced
Supported product too*

Domain-Specific Libraries

Intel® Integrated Performance Primitives

Intel® Math Kernel Library

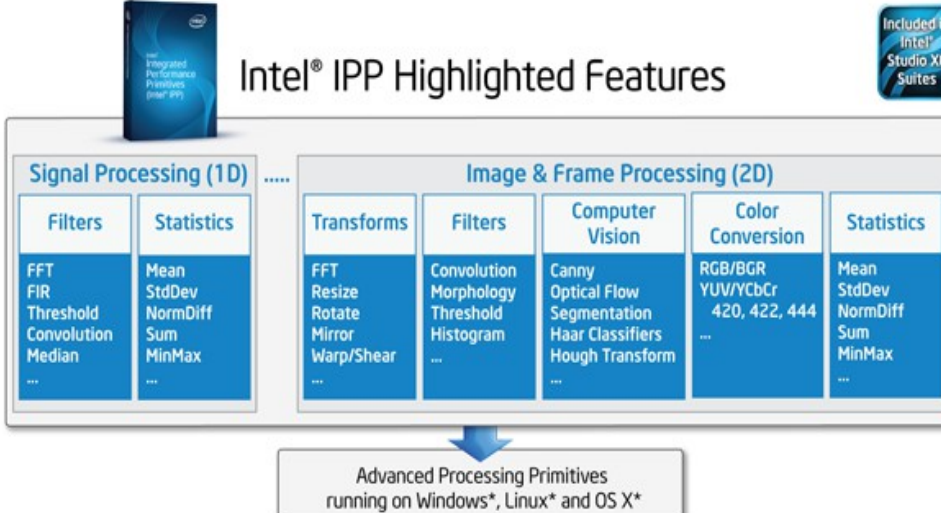
Established Standards

OpenMP*

Coarray Fortran

(Message Passing Interface)

Intel® IPP Highlighted Features



Signal Processing (1D)

Filters	Statistics
FFT	Mean
FIR	StdDev
Threshold	NormDiff
Convolution	Sum
Median	MinMax
...	...

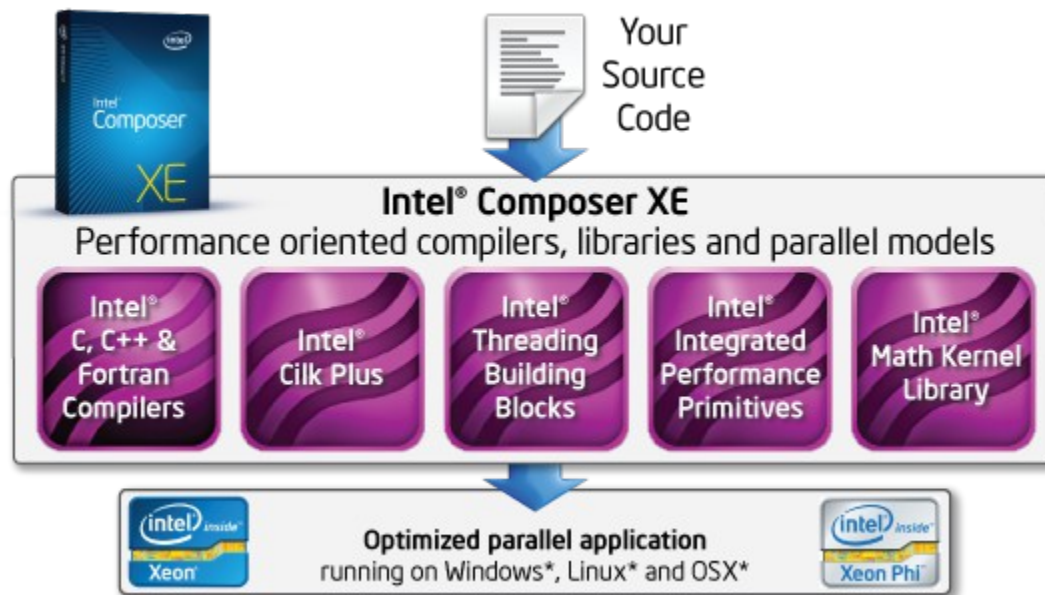
Image & Frame Processing (2D)

Transforms	Filters	Computer Vision	Color Conversion	Statistics
FFT	Convolution	Canny	RGB/BGR	Mean
Resize	Morphology	Optical Flow	YUV/YCbCr	StdDev
Rotate	Threshold	Segmentation	420, 422, 444	NormDiff
Mirror	Histogram	Haar Classifiers	...	Sum
Warp/Shear	...	Hough Transform	...	MinMax
...

Advanced Processing Primitives running on Windows*, Linux* and OS X*

Included in Intel® Studio XE Suites

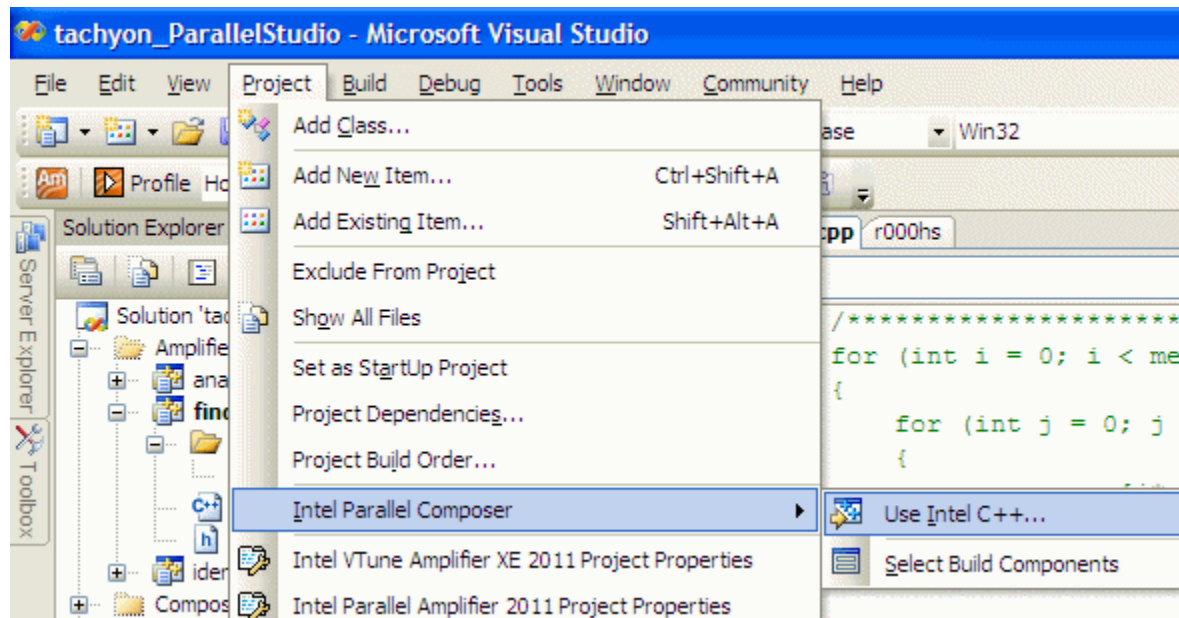
- The wish / aim:
- Use as much existing libraries and automatization as possible



www.intel.com

Intel (Parallel) Composer

- For Windows:
- Microsoft Visual Studio is prerequisite for composer
- Integration in existing IDE (**DEMO**):



- In Linux:
- Supports GNU tool chain
 - Integration in eclipse possible (tutorials online*)
 - Demo
 - Or use command line

Compiler (GCC)

Linker (LD)

Debugger (GDB)

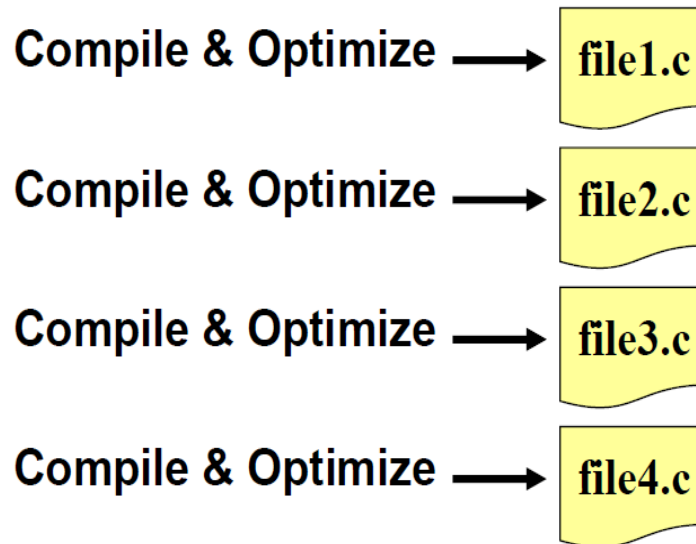
Toolchain

* <http://software.intel.com/en-us/articles/intel-c-compiler-for-linux-using-intel-compilers-with-the-eclipse-ide-pdf#installing>

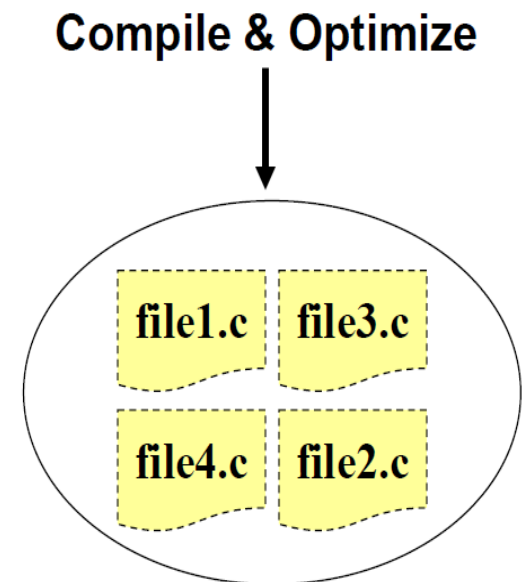
- It may already help to change the / optimize with compiler:
 - More performance through appropriate flags
 - Indication of problematic parts by reports

- standardflags: - O3 / Ox
- - vec
- - parallel
- - ip / ipo
- - fast

Without IPO



With IPO



Intel (Parallel) Composer



Modell	MSVC (2008)	<u>ICC (11)</u>
Venus (7200)	27,02 sec	19,76 sec
Fishes (20000)	52,36 sec	42,82 sec
Fertility (55000)	211,35 sec	186,66

- Reports
 - i.e. vectorization-report
 - -vec-report3
 - optimization-report
 - Contains Vec-Report
 - Inlining
 - Loop unrolling / fusing
 - ...

```
%sipsol.F(194): (col. 10) remark: loop was not vectorized: vectorization possible but seems inefficient.  
%sipsol.F(206): (col. 12) remark: loop was not vectorized: existence of vector dependence.  
%sipsol.F(296): (col. 7) remark: vector dependence: assumed OUTPUT dependence between fi_n line 296 and fi_n line 309.  
%sipsol.F(309): (col. 7) remark: vector dependence: assumed OUTPUT dependence between fi_n line 309 and fi_n line 296.
```

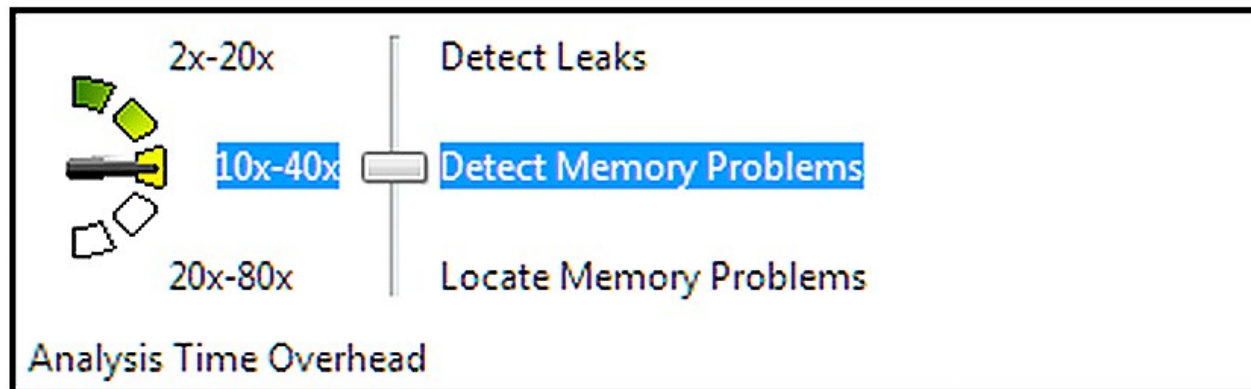
```
35:      subroutine fd( y )
36:      integer :: i
37:      real, dimension(10), intent(inout) :: y
38:      do i=2,10
39:          y(i) = y(i-1) + 1
40:      end do
41:      end subroutine fd
```

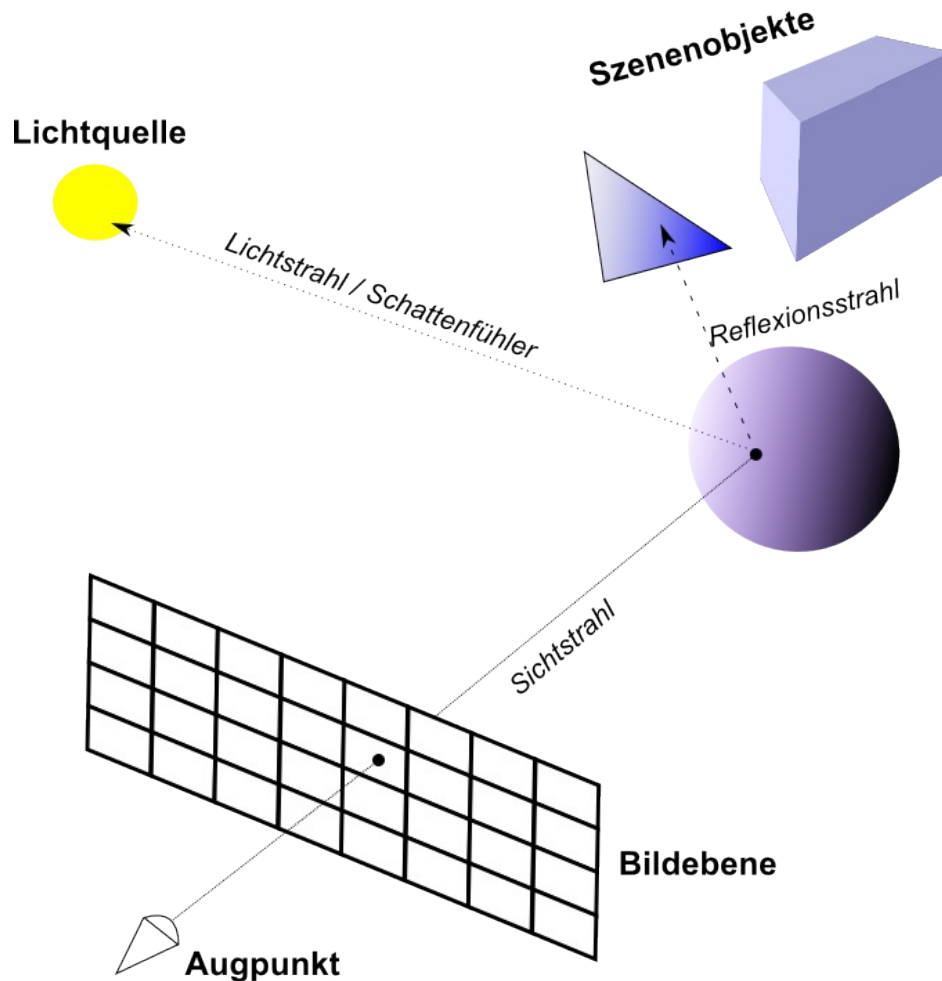
```
novec.f90(38): (col. 3) remark: loop was not vectorized: existence of
vector dependence.
novec.f90(39): (col. 5) remark: vector dependence: proven FLOW
dependence between y line 39, and y line 39.
novec.f90(38:3-38:3):VEC:MAIN_: loop was not vectorized: existence of
vector dependence
```

DEMO: Intel (Parallel) Composer

- **Show** how to turn on reporting!
- **Compile** code as RELEASE
- **Explain** dependencies
- **Remove** by reversing loop
- **Explain** ivdep

- Memory error checker (**DEMO**)
 - Leaks, corruption
- Threading checker
 - Data races, deadlocks

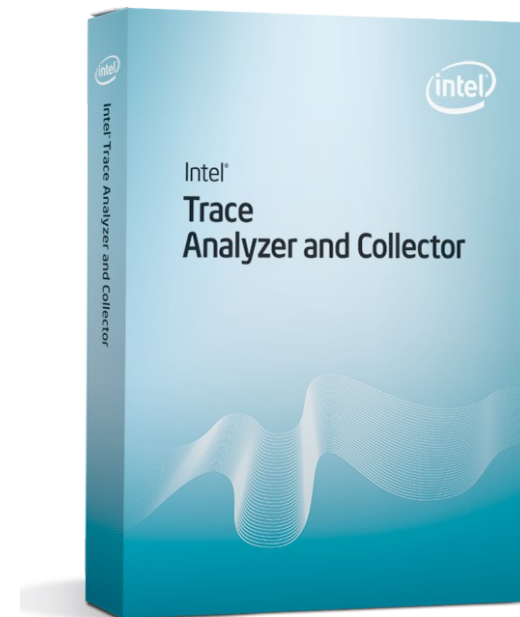




1. Calculate eye ray
2. Test intersection with objects
3. Find nearest intersection point
4. Create reflection and shadow rays
5. Calculate local color
6. Repeat steps 2-5 with reflection rays

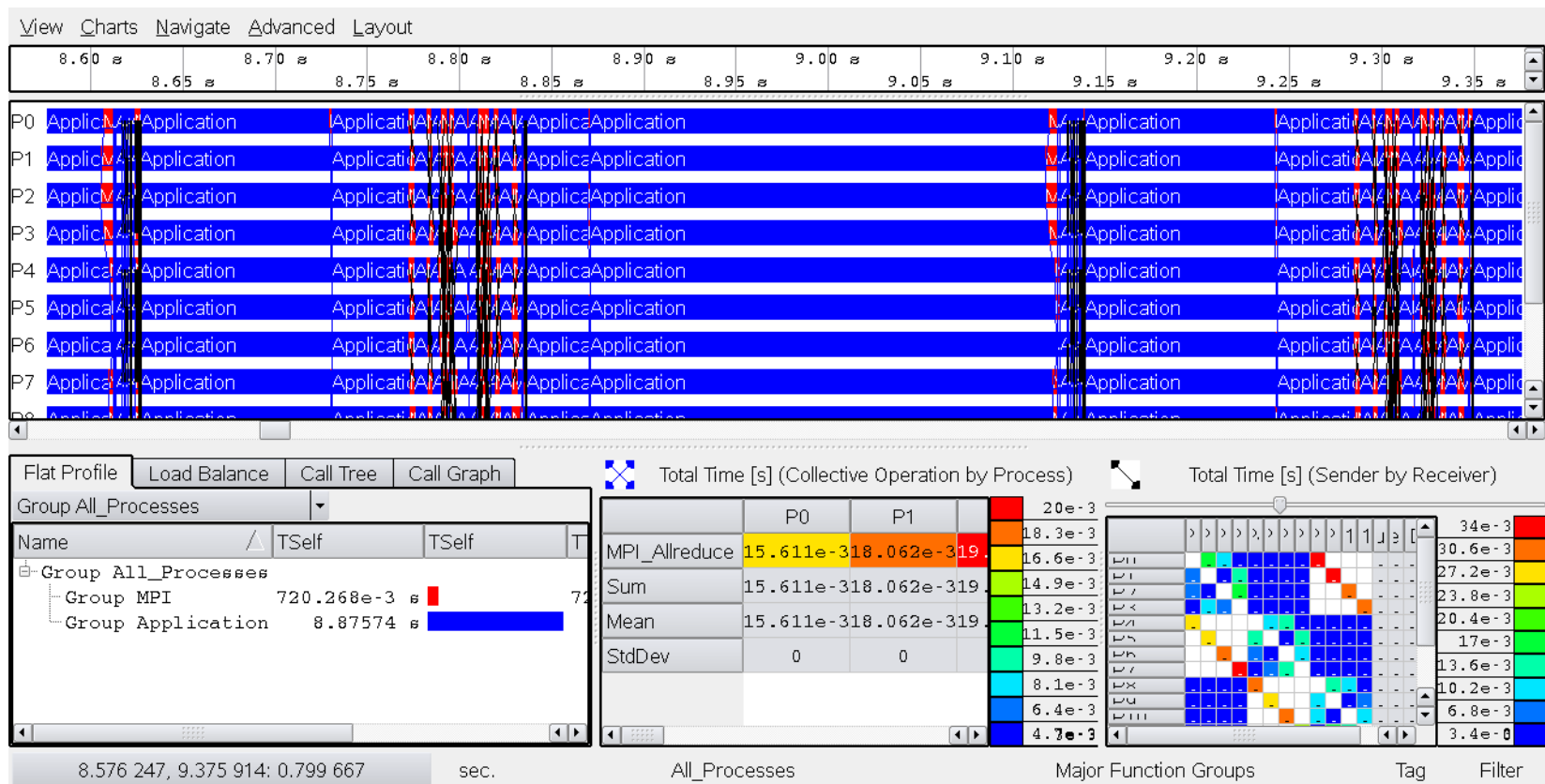
Intel Trace Analyzer and Collector

- Analyzing MPI behavior
- Search for bottlenecks, deadlocks, data corruption
- Debugging (call stack, debug infos)
- Supports „Intel architecture based cluster systems“
- Only guaranteed to work with Intel MPI

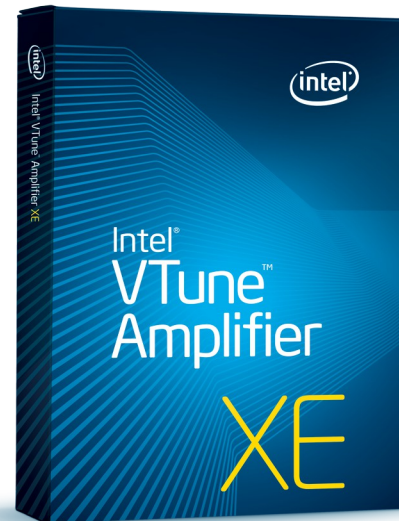


Intel Trace Analyzer and Collector

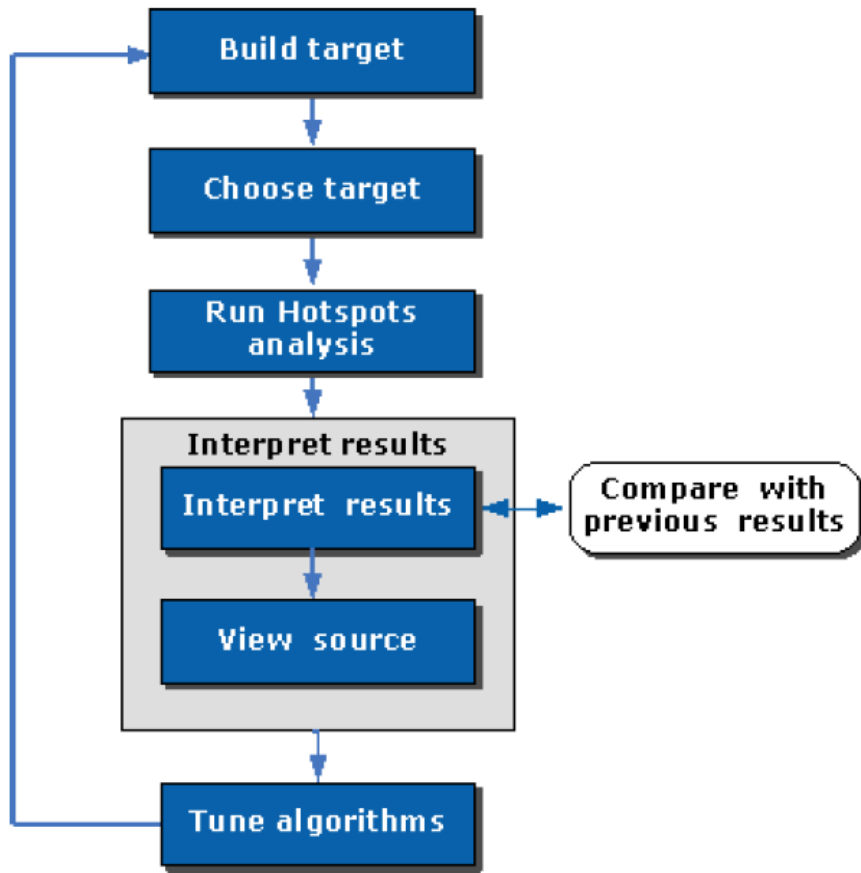
- Analyzer example screenshot:



- Supports C (++/#), Fortran, Assembler, Java
- Serial and parallel tuning
- Sample based
- Normal build with -g can be analyzed
 - Use releasebuild
- Graphical or commando line based execution
- Less overhead → real runtimes and results

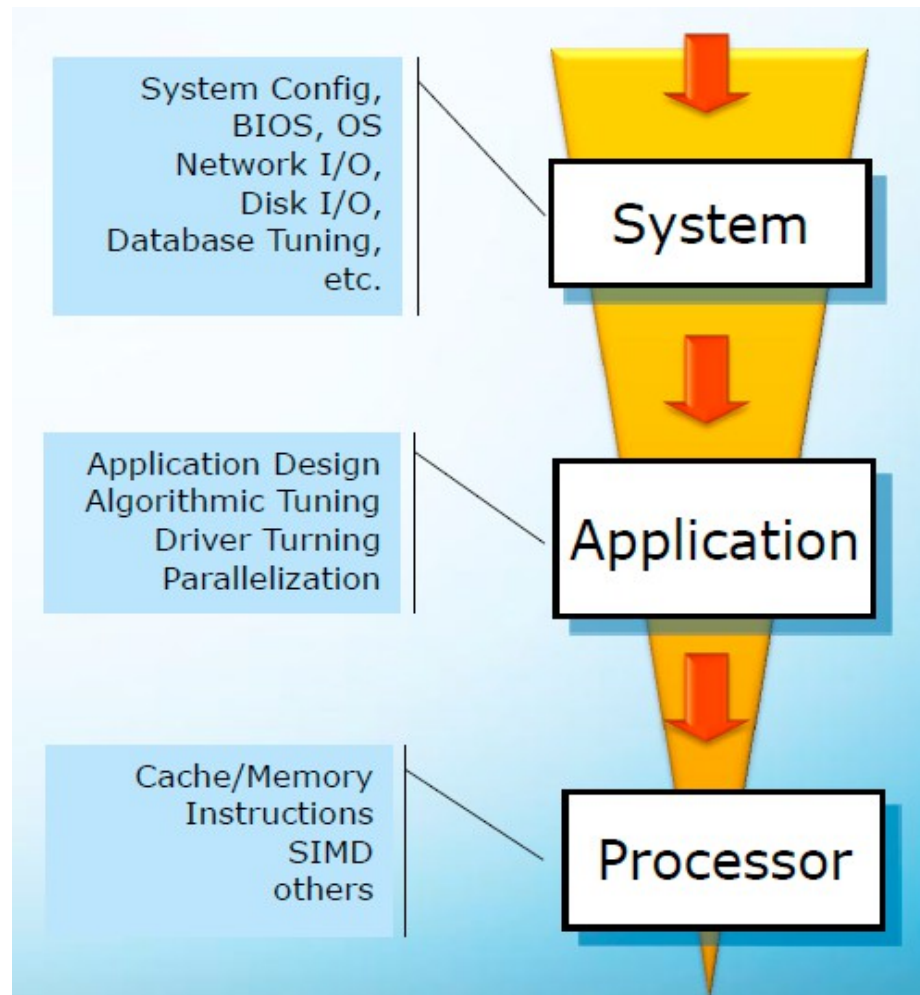


- Workflow for optimizing
- Different types of analyses (see next slides)
- Questions:
 - Why optimizing?
 - What should be optimized?
 - Aim of optimizing?
 - How to optimize?



Intel Vtune Amplifier

- Use top-down-method
- Amount of work increases with the depth of the level
- Additional aims
 - Portability
 - Code readability
 - Maintainability
 - Reliability



- **Scale** and **Vectorize**
- Example: 9-point stencil image blur filter

	Single Thread	1 Thread & Vectorized	122 Threads & Vectorized
Xeon Phi	2838,342 s	623,302 s	8,772 s
Xeon	244,178 s	186,585 s	43,862 a

- Example: Diffusion Simulation from Maruyama

	Single Thread	183 Threads	244 Threads & Vectorized
Xeon Phi	5699,55 s	84,622 s	18,664 s

Taken from: "Intel Xeon Phi Coprocessor High-Performance Programming"

Intel Vtune Amplifier



TECHNISCHE
UNIVERSITÄT
DARMSTADT

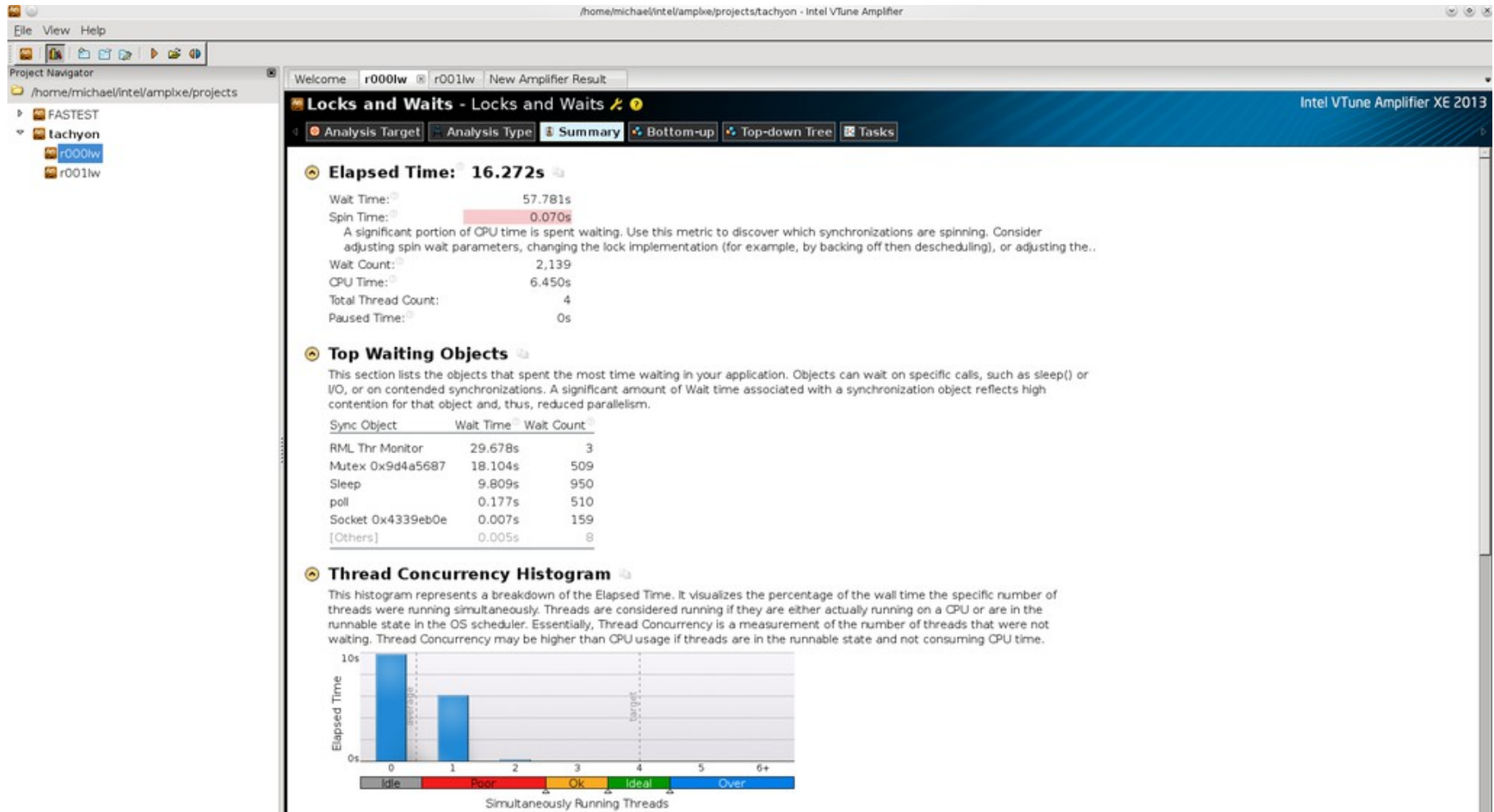
The screenshot shows the Intel Vtune Amplifier interface. On the left is a tree view of analysis categories: Algorithm Analysis (with sub-items: Lightweight Hotspots, Hotspots, Concurrency, Locks and Waits), Intel Core 2 Processor Analysis, Nehalem / Westmere Analysis, and Sandy Bridge / Ivy Bridge Analysis (with sub-items: General Exploration, Bandwidth, Access Contention, Branch Analysis, Client Analysis, Core Port Saturation, Cycles and uOps, Loop Analysis, Memory Access, Port Saturation). Below these are Intel Atom Processor Analysis, Knights Corner Platform Analysis, Power Analysis, and Custom Analysis. The 'Hotspots' item under 'Algorithm Analysis' is selected and highlighted. The main panel on the right is titled 'Hotspots' and contains a 'Copy' button. It includes a description: 'Identify your most time-consuming source code. Unlike Lightweight Hotspots, Hotspots collects stack and call tree information. This analysis type cannot be used to profile the system but must either launch an application/process or attach to one. This analysis type uses user-mode sampling and tracing collection. Press F1 for more details.' Below this is a 'CPU sampling interval, ms:' field with a value of '1' and a spinner. There is an unchecked checkbox for 'Analyze user tasks'. A 'Details' section is expanded, showing a list of settings: CPU sampling interval, ms: 1; Collect CPU sampling data: With stacks; Collect signalling API data: No; Collect synchronization API data: No; Collect I/O API data: No; Analyze user tasks: No; Stack unwinding mode: During collection; Stitch stacks: Yes; Collect timeline data: Yes; Collect sleep data: No; Collect frequency data: No.

- Depends on underlying architecture
- Our new Cluster: Sandybridge

Intel Vtune Amplifier



TECHNISCHE
UNIVERSITÄT
DARMSTADT

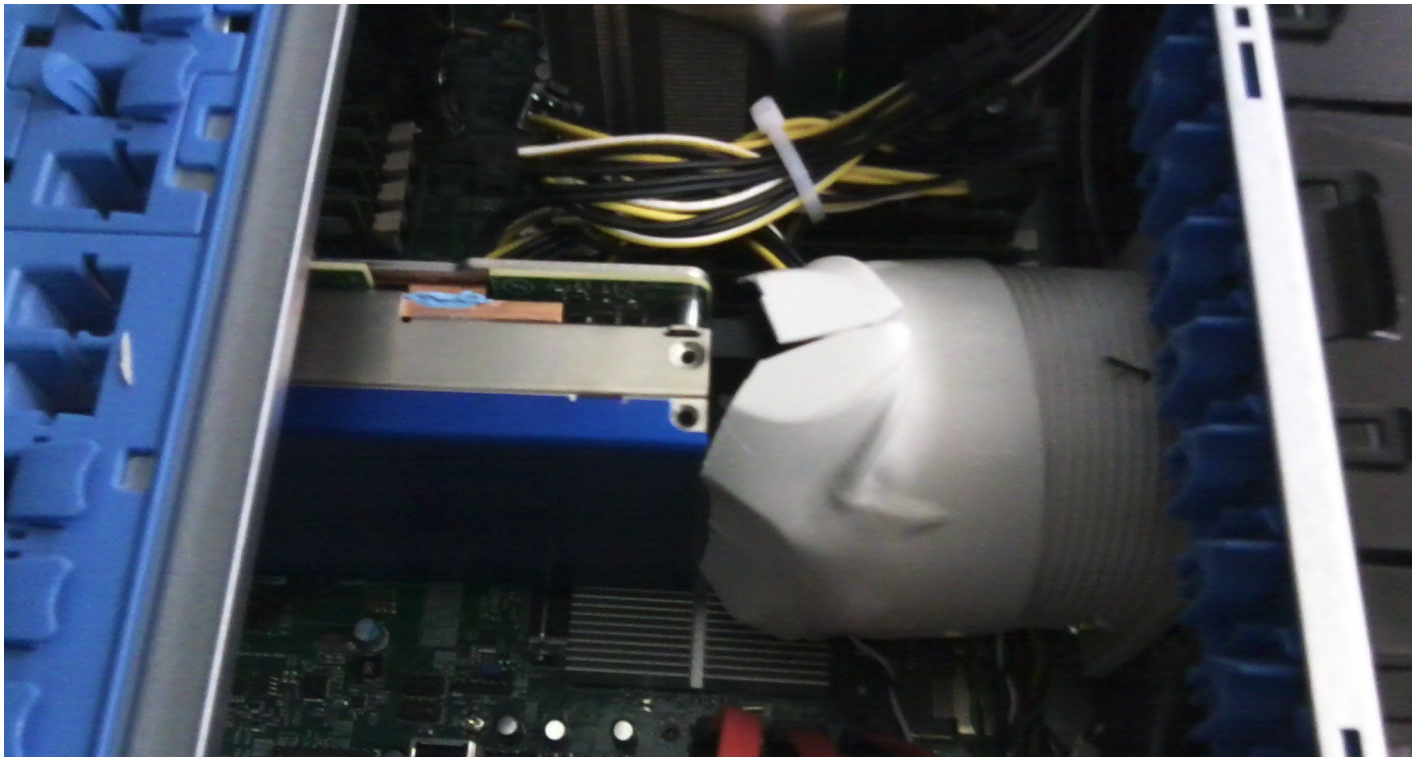


- Intel delivers code samples with the Amplifier
- Considered here:
 - The Tachyon-Raytracer
 - Matrix Matrix Multiplication
- Include intentional errors / problems
- Little changes result in big impacts on performance

- **DEMO**
 - Raytracer with hotspots und locs
 - Matrix-Multiplication and LL-Cache Misses
 - Also on Xeon Phi
 - Hotspots / Assembler
 - (Hardwarecounter)

Vtune Amplifier: DEMO

- Xeon Phi also supported
- 240 Threads, 60 cores, 8 GB GDDR5 RAM, PCI-E 2.0



Vtune Amplifier: DEMO

- First compile the program and copy to Phi

```
mburger@mice:~/matrix/linux$ source /opt/intel/bin/compilervars.sh intel64
mburger@mice:~/matrix/linux$ make mic
icc -g -O3 -debug inline-debug-info -vec-report0 -mmic -c ../src/util.c -D_ICC -D_LINUX
icc -g -O3 -debug inline-debug-info -vec-report0 -mmic -c ../src/multiply.c -D_ICC -D_LINUX
icc -g -O3 -debug inline-debug-info -vec-report0 -mmic -c ../src/matrix.c -D_ICC -D_LINUX
icc -g -O3 -debug inline-debug-info -vec-report0 -mmic util.o multiply.o matrix.o -o matrix.mic -lpthread -lm
scp matrix.mic mic0:/tmp
Enter passphrase for key '/home/mburger/.ssh/id_rsa':
matrix.mic 100% 32KB 31.5KB/s 00:00
touch mic-pushed
mburger@mice:~/matrix/linux$
```

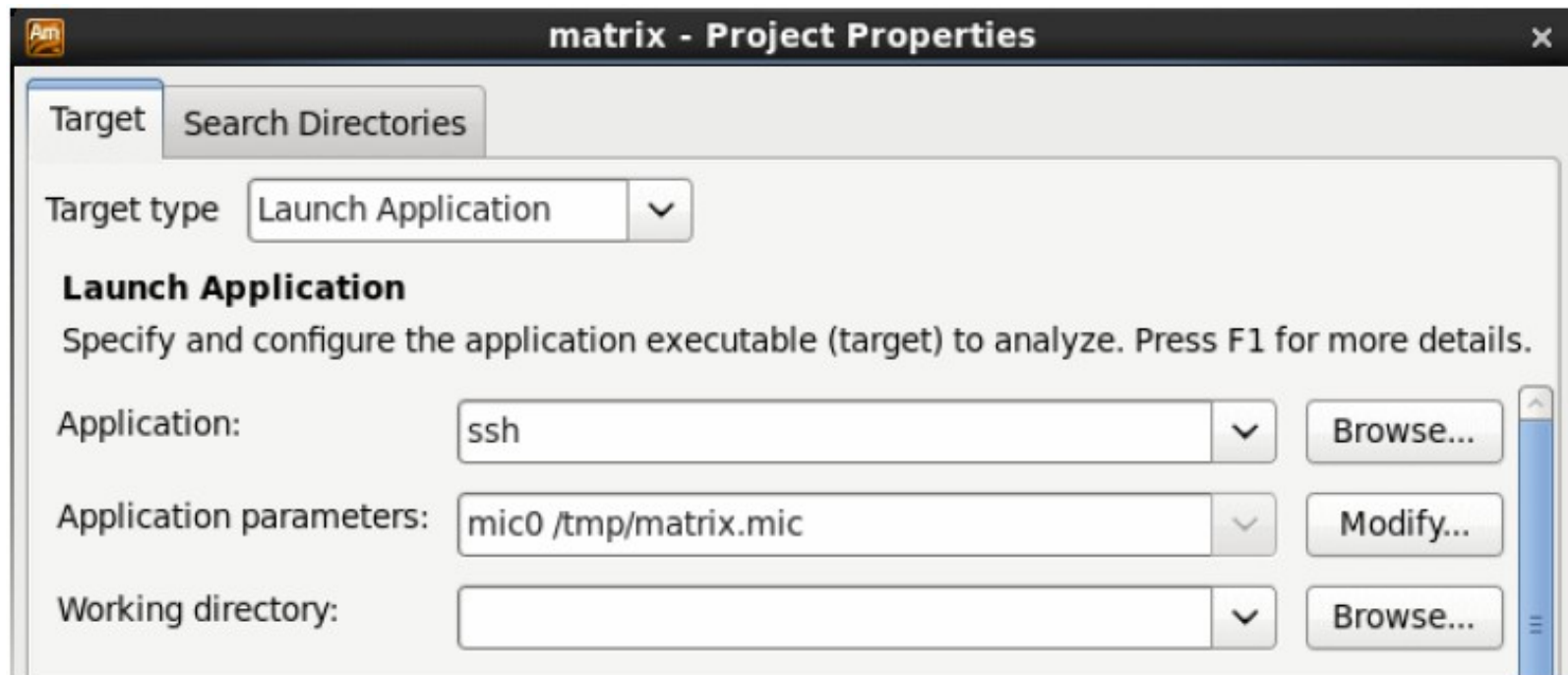
Vtune Amplifier: DEMO

- Run in native mode:

```
mburger@mice:~/matrix/linux$ ssh mic0
Enter passphrase for key '/home/mburger/.ssh/id_rsa':
[mburger@mice-mic0 mburger]$ cd /tmp/
[mburger@mice-mic0 /tmp]$ ./matrix.mic
Addr of buf1 = 0x0x7fa3f72cd010
Offs of buf1 = 0x0x7fa3f72cd180
Addr of buf2 = 0x0x7fa3f024c010
Offs of buf2 = 0x0x7fa3f024c1c0
Addr of buf3 = 0x0x7fa3e91cb010
Offs of buf3 = 0x0x7fa3e91cb100
Matrix size: 3840
Using multiply kernel: multiply1
Threads #: 240, Execution time = 26.170 seconds
[mburger@mice-mic0 /tmp]$ █
```

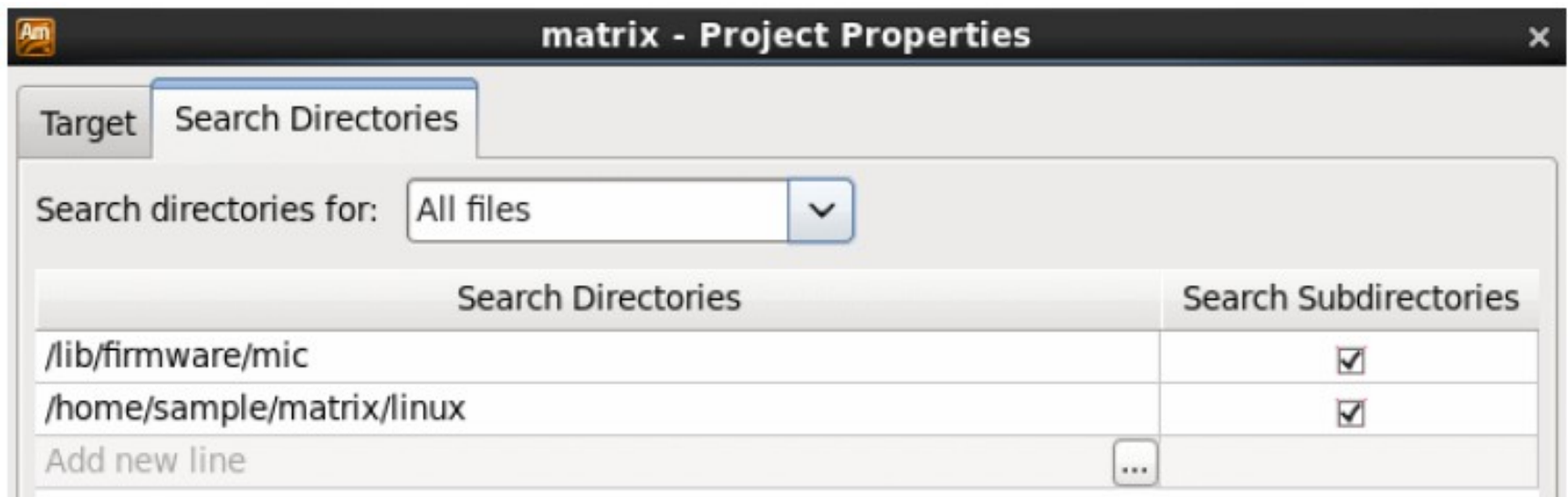
Vtune Amplifier: DEMO

- Setup project correctly:



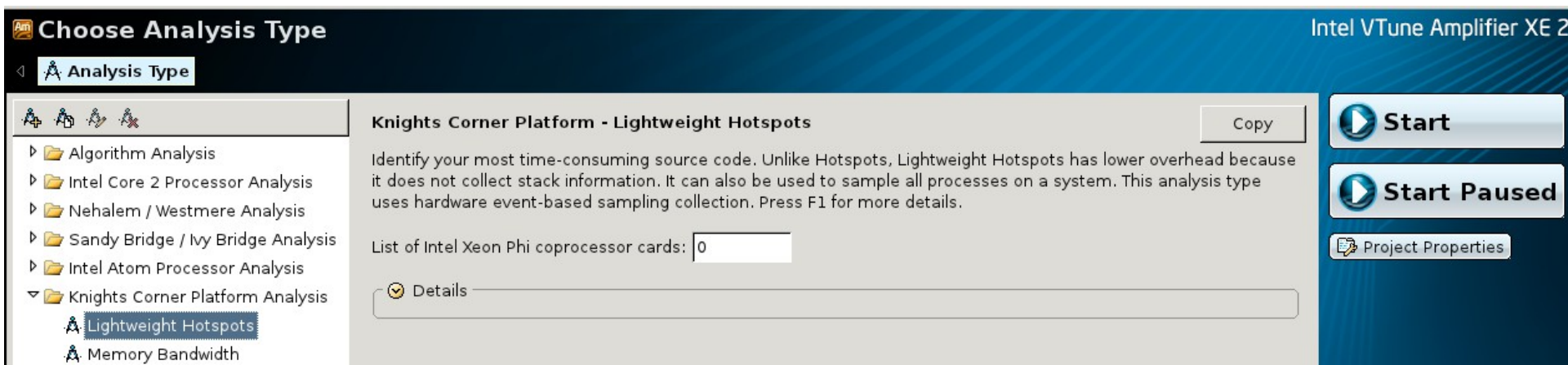
Vtune Amplifier: DEMO

- Setup project correctly:



Vtune Amplifier: DEMO

- Start correct analyses type
- Sample driver must be loaded and configured correctly



Vtune Amplifier: DEMO

- Look at the results
- Summary
- Tree View
- Clock ticks per instruction

Elapsed Time: 79.174s

CPU Time: 16853.340s

Instructions Retired: 744,720,000,000

CPI Rate: 22.630

Lightweight Hotspots - Hotspots

Analysis Target Analysis Type Collection Log

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Instructions Retired	CPI Rate
multiply1	16720.170s	736,220,000,000	22.711
__do_softirq	90.260s	3,630,000,000	24.865
default_send_IPI_mask	7.660s	540,000,000	14.185
__raw_spin_unlock_irq	6.160s	650,000,000	9.477
weighted_cpuload	2.420s	140,000,000	17.286
source_load	2.240s	110,000,000	20.364
Selected 1 row(s):	16720.170s	736,220,000,000	22.711

Vtune Amplifier: DEMO

- Look in the code:

Source		Assembly			
Sou..	Source	CPU Time★	Instructions Re...		
36	// Naive implementation				
37	for(i=tidx; i<msize; i=i+numt) {				
38	for(j=0; j<msize; j++) {	0.550s	80,000,000		
39	for(k=0; k<msize; k++) {	2.400s	130,000,000		
40	c[i][j] = c[i][j] + a[i][k] * b[k][j];	16717.220s	736,010,000,000		
41	}				
42	}				
43	}				
44	}				
45	void multiply2(int msize, int tidx, int numt, TYPE a[][NUM				

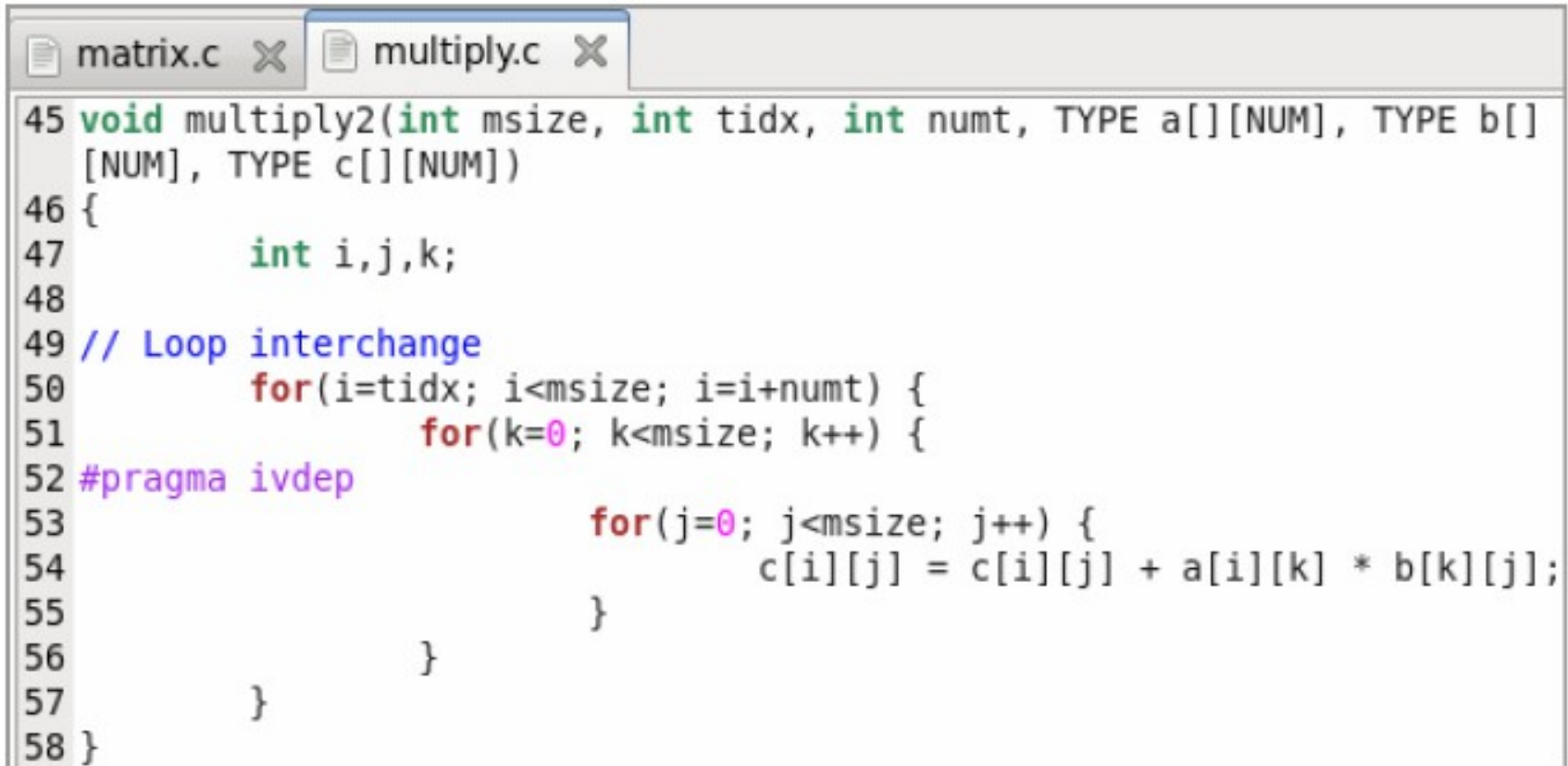
Vtune Amplifier: DEMO

- Change the code:

```
matrix.c x multiply.c x
34 // Select which multiply kernel to use via the following macro so that the
35 // kernel being used can be reported when the test is run.
36 #define MULTIPLY multiply1
37
38 int NTHREADS = MAXTHREADS;
39
40 //static TYPE a[NUM][NUM], b[NUM][NUM], c[NUM][NUM];
41 typedef TYPE array[NUM];
42 typedef unsigned long long UINT64;
```

Vtune Amplifier: DEMO

- Change the code:



```
45 void multiply2(int msize, int tidx, int numt, TYPE a[][NUM], TYPE b[]  
    [NUM], TYPE c[][NUM])  
46 {  
47     int i,j,k;  
48  
49 // Loop interchange  
50     for(i=tidx; i<msize; i=i+numt) {  
51         for(k=0; k<msize; k++) {  
52 #pragma ivdep  
53             for(j=0; j<msize; j++) {  
54                 c[i][j] = c[i][j] + a[i][k] * b[k][j];  
55             }  
56         }  
57     }  
58 }
```

- Test if problem is resolved:

```
[mburger@mice-mic0 /tmp]$ ./matrix.mic
Addr of buf1 = 0x0x7fdb838ce010
Offs of buf1 = 0x0x7fdb838ce180
Addr of buf2 = 0x0x7fdb7c84d010
Offs of buf2 = 0x0x7fdb7c84d1c0
Addr of buf3 = 0x0x7fdb757cc010
Offs of buf3 = 0x0x7fdb757cc100
Matrix size: 3840
Using multiply kernel: multiply2
Threads #: 240, Execution time = 3.563 seconds
[mburger@mice-mic0 /tmp]$ █
```

- Approach for the investigation of the MMM-problems:
 - Hardware counter
- Count certain „events“:
 - Cache / Memory accesses
 - Using of INT / FP-Units
 - SIMD instructions
 -

→ must be supported by hardware!

- Cluster Studio is an extensive collection of tools
- Assists in a lot of parts of software development process
- Several pitfalls are still there however
- Only minimal examples covered here

The End :)

Thanks for your attention !