Intel Software Tools

Stephen Blair-Chappell

Intel Compiler Labs

Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013

Helping Developers Efficiently Produce Fast, Scalable and Reliable Applications

More Cores. Wider Vectors. Performance Delivered.

Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013

More Cores

Multicore Many-core

50+ cores

Wider Vectors

128 Bits

256 Bits

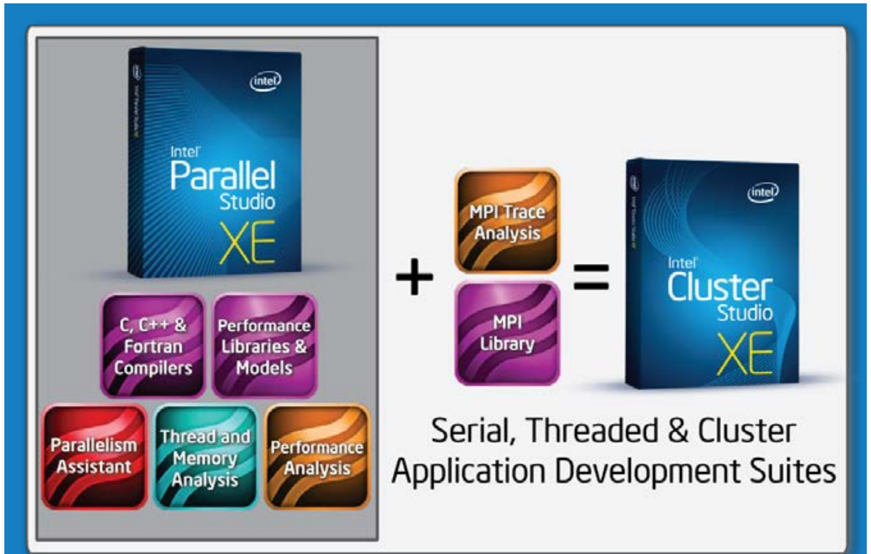
512 Bits

Scaling Performance Efficiently

Serial Performance

Task & Data Parallel Performance

Distributed Performance



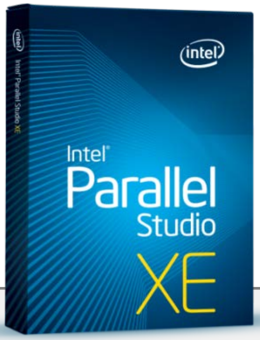
Serial, Threaded & Cluster Application Development Suites

- Industry-leading performance from advanced compilers
- Comprehensive libraries
- Parallel programming models
- Insightful analysis tools

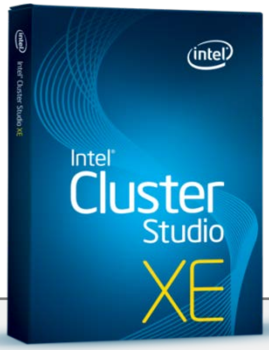
Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013 †

Phase	Product	Feature	Benefit
Build	**Intel® Advisor XE**	Threading design assistant (Studio products only)	• Simplifies, demystifies, and speeds parallel application design
	Intel® Composer XE	• C/C++ and Fortran compilers • Intel® Threading Building Blocks • Intel® Cilk™ Plus • Intel® Integrated Performance Primitives • Intel® Math Kernel Library	• Enabling solution to achieve the application performance and scalability benefits of multicore and forward scale to many-core
	Intel® MPI Library†	High Performance Message Passing (MPI) Library	• Enabling High Performance Scalability, Interconnect Independence, Runtime Fabric Selection, and Application Tuning Capability
Verify & Tune	**Intel® VTune™ Amplifier XE**	Performance Profiler for optimizing application performance and scalability	• Remove guesswork, saves time, makes it easier to find performance and scalability bottlenecks
	Intel® Inspector XE	Memory & threading dynamic analysis for code quality Static Analysis for code quality	• Increased productivity, code quality, and lowers cost, finds memory, threading , and security defects before they happen
	Intel® Trace Analyzer & Collector†	MPI Performance Profiler for understanding application correctness & behavior	• Analyze performance of MPI programs and visualize parallel application behavior and communications patterns to identify hotspots

Efficiently Produce Fast, Scalable and Reliable Applications



Top New Features



Performance	Performance Profiling	Reliability	Reproducibility	Standards	Parallelism Assistance
Improved compiler and library performance + Ivy Bridge microarchitecture + Haswell microarchitecture + Intel® Xeon Phi™ coprocessor	A dozen new analysis features Low overhead Java* profiling CPU Power Analysis	Pointer checker Heap growth analysis Improved MPI fault tolerance†	Conditional numerical reproducibility	Expanded C++ 11 Expanded Fortran 2008 MPI 2.2†	Analysis extended to include Linux*, Fortran and C# (in addition to Windows* and C/C++)

† Intel® Cluster Studio XE

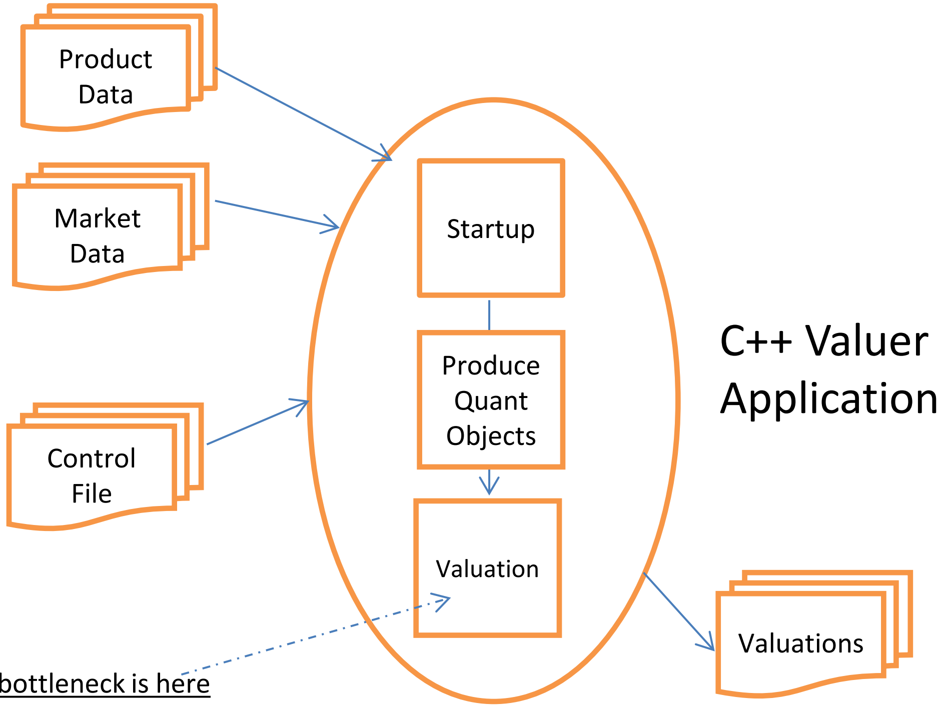
Efficiently produce fast, scalable and reliable applications running on Windows* and Linux*

Optimization Notice

A Story …

A Bank near you!

The Reason Why



Main bottleneck is here
One Line of Code (in a Monte Carlo Calculation)
takes 50% of the time.

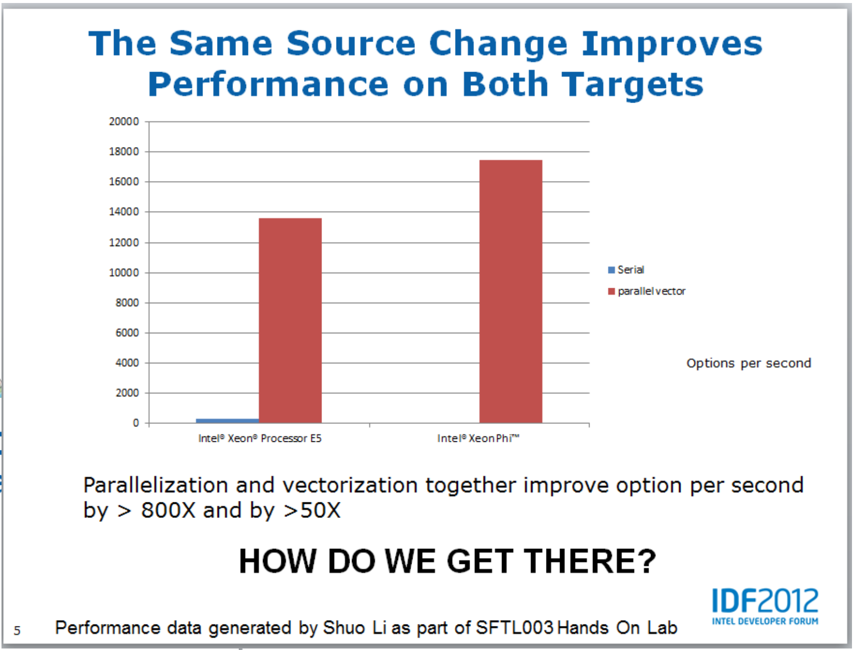
- Long overnight runtime
- Cost of renting space in data centres
- Power Consumption

ON A PRESENATION FROM:

2012
ELOPER FORUM

ogramming Continui Between Intel®
eon and Intel Xeon® Coprocessors
for High Performanc

Robert Geva, Princ



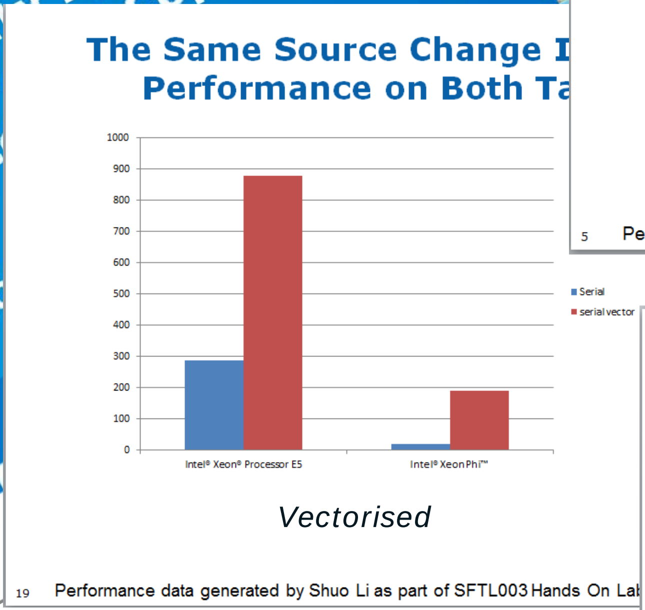
The Same Source Change Improves Performance on Both Targets

Parallelization and vectorization together improve option per second by > 800X and by >50X

HOW DO WE GET THERE?

5 Performance data generated by Shuo Li as part of SFTL003 Hands On Lab

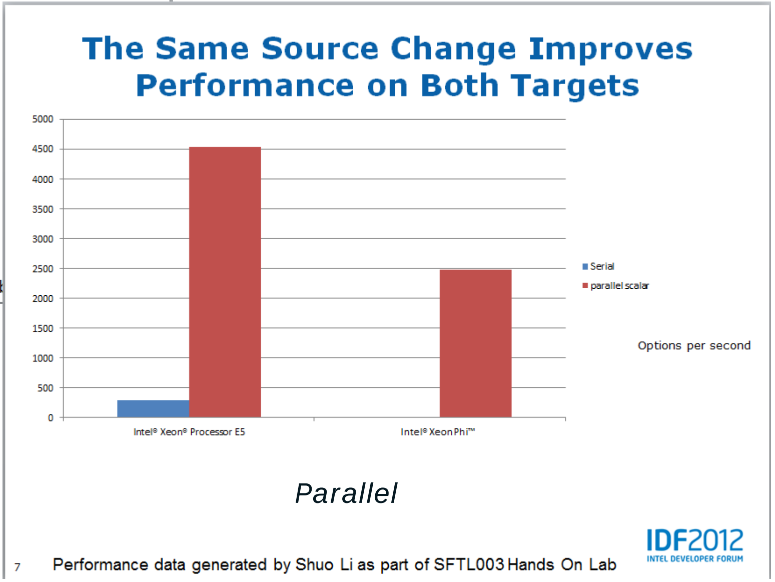
IDF2012 INTEL DEVELOPER FORUM



The Same Source Change I Performance on Both Ta

Vectorised

19 Performance data generated by Shuo Li as part of SFTL003 Hands On Lab



The Same Source Change Improves Performance on Both Targets

Parallel

7 Performance data generated by Shuo Li as part of SFTL003 Hands On Lab

IDF2012 INTEL DEVELOPER FORUM

On the graphs, bigger is better

Optimization Notice

Timing Summary

	initialization	calculation	total
MS VS 10 - CL , base line	1324	627	1951
ICL , base line	1172	487	1659
ICL, vectorized	1161	202	1363
ICL, vectorized + OMP threading	612	105	717
ICL, vectorized + Cilk tasking	608	117	725
ICL, vectorized + OMP + MKL VSL	99	103	202

Source code freely available : Please contact presenter in case you want to have the source code and build scripts to reproduce the measurements

ArraySection version available from software.intel.com (search for Black-Scholes)

Three Common Requests

"How can I make my program run **faster?**"

"How can I make my program **parallel?**"

"Will my code run on any CPU? - **compatibility**"

Optimization Notice

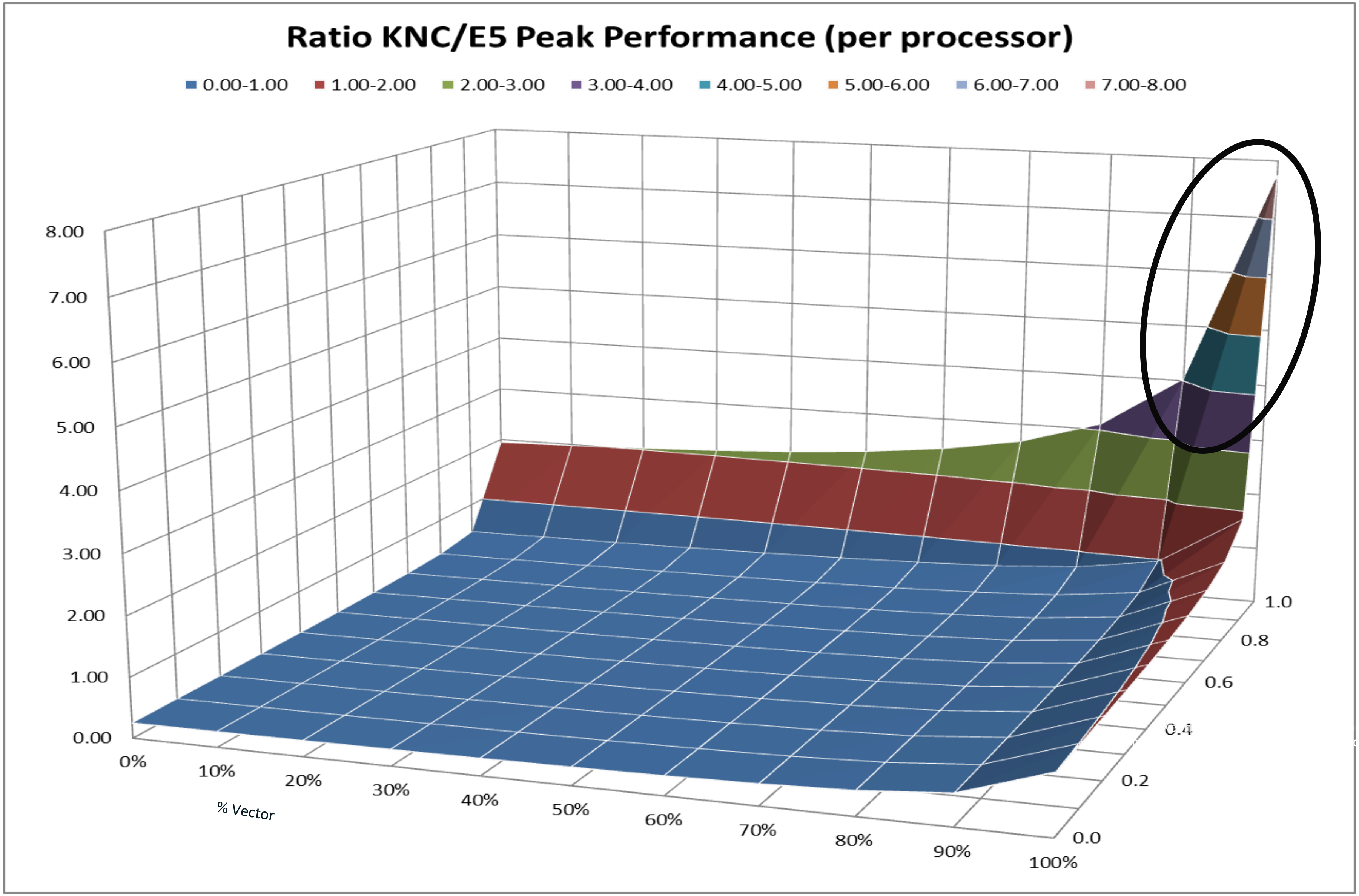
Programs on Xeon Phi . . .

Code must be

highly Parallel

effectively Vectorised

Application Performance: Intel® Xeon Phi™ Coprocessor



For more information go to http://www.intel.com/performance

Optimization Notice

And three more questions of late . . .

Will my code run on a Xeon Phi?

?

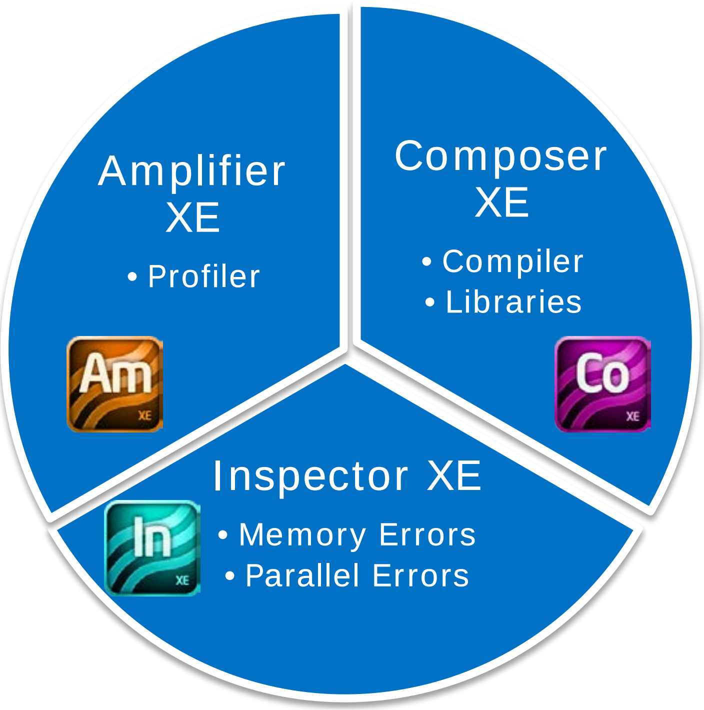
Do I have to change my code to much?

?

What performance will I get?

?

Intel® Parallel Studio XE



+ Advisor Ad

- **Intel® Parallel Advisor**
  - Use to model parallelism in your existing applications

- **Intel® Composer XE**
  - Use to generate fast, safe, parallel code (C/C++, Fortran)
- **Intel® VTune™ Amplifier XE**
  - Find hotspots and bottlenecks in you code.
- **Intel® Inspector XE**
  - Use to find memory and threading errors

Four Components

Three Common Requests

“How can I make my program run **faster?**”

“How can I make my program **parallel?**”

“Will my code run on any CPU? - **compatibility**”

Optimization Notice

The compiler uses many optimisation techniques

Faster Code

architecture-specific optimisations

profile guided optimisation

-Od

auto-Parallelism

auto-vectorisation

loop optimisations

function inlining

-O3

-O1

inter procedural optimisation

optimise for size

optimised runtime functions

-O2

fast intrinsic functions

fast floating point

http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

http://software.intel.com/sites/products/collateral/hpc/compilers/compiler_qrg12.pdf

Optimization Notice

Often we are happy with out-of-the-box experience

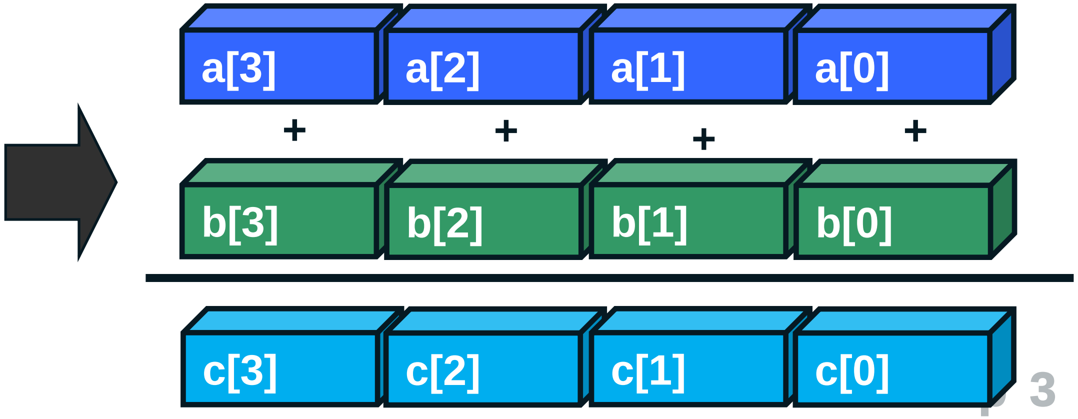
When was the last time you looked at some documentation?

Optimization Notice

SIMD Instruction Enhancements

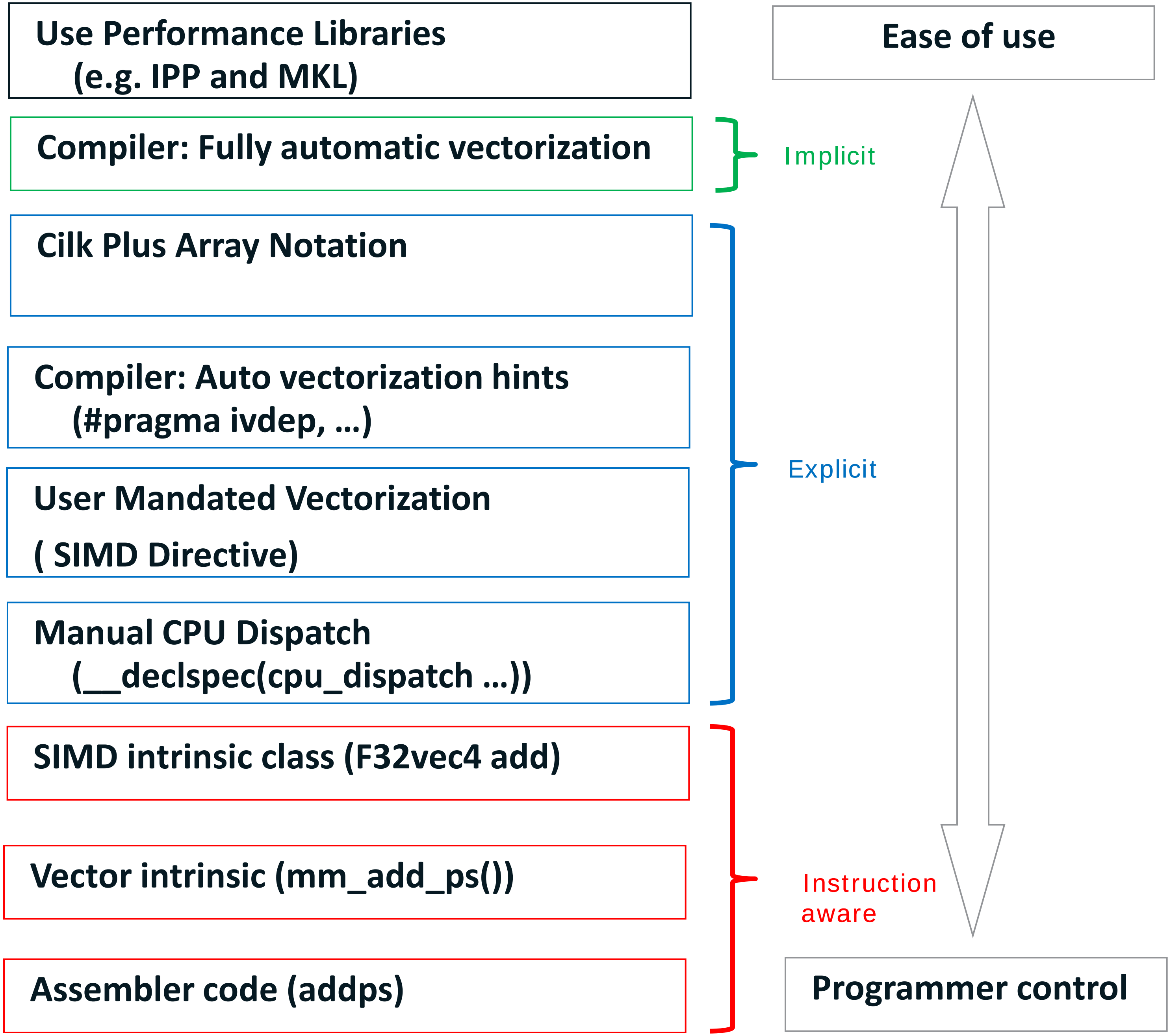
1999	2000	2004	2006	2007	2008	2009	2011	2012\2013	2012
SSE	SSE2	SSE3	SSSE3	SSE4.1	SSE4.2	AES-NI	AVX	AVX2	MIC
70 instr Single-Precision Vectors Streaming operations	144 instr Double-precision Vectors 8/16/32 64/128-bit vector integer	13 instr Complex Data	32 instr Decode	47 instr Video Graphics building blocks Advanced vector instr	8 instr String/XML processing POP-Count CRC	7 instr Encryption and Decryption Key Generation	~100 new instr. ~300 legacy sse instr updated 256-bit vector 3 and 4-operand instructions	Int. AVX expands to 256 bit Improved bit manip. fma Vector shifts Gather	512-bit vector

```
for (i=0;i<MAX;i++)
   c[i]=a[i]+b[i];
```



Optimization Notice

Other Ways of Inserting Vectorised Code



Use Performance Libraries (e.g. IPP and MKL)

Compiler: Fully automatic vectorization

Cilk Plus Array Notation

Compiler: Auto vectorization hints (#pragma ivdep, …)

User Mandated Vectorization

(SIMD Directive)

Manual CPU Dispatch (__declspec(cpu_dispatch …))

SIMD intrinsic class (F32vec4 add)

Vector intrinsic (mm_add_ps())

Assembler code (addps)

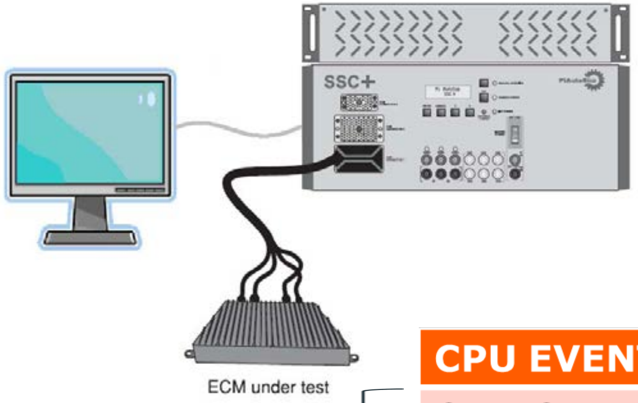
An example

Faster Code

Speedup by upgrading silicon

CPU	No Auto-Vectorisation	With Auto-Vectorisation	Speedup
P4	39.344	21.9	**1.80**
Core 2	5.546	0.515	**10.77**
Speedup	**7.09**	**45.52**	**76**

Speedup by swapping compiler



Verified using VTune

CPU EVENT	Without Vect	With Vect
CPU_CLK_UNHALTED.CORE	16,641,000,448	1,548,000,000
INST_RETIRED.ANY	3,308,999,936	1,395,000,064
X87_OPS_RETIRED.ANY	250,000,000	0
SIMD_INST_RETIRED	0	763,000,000

Full paper available here:
http://edc.intel.com/Link.aspx?id=1045

Three Common Requests

“How can I make my program run faster?”

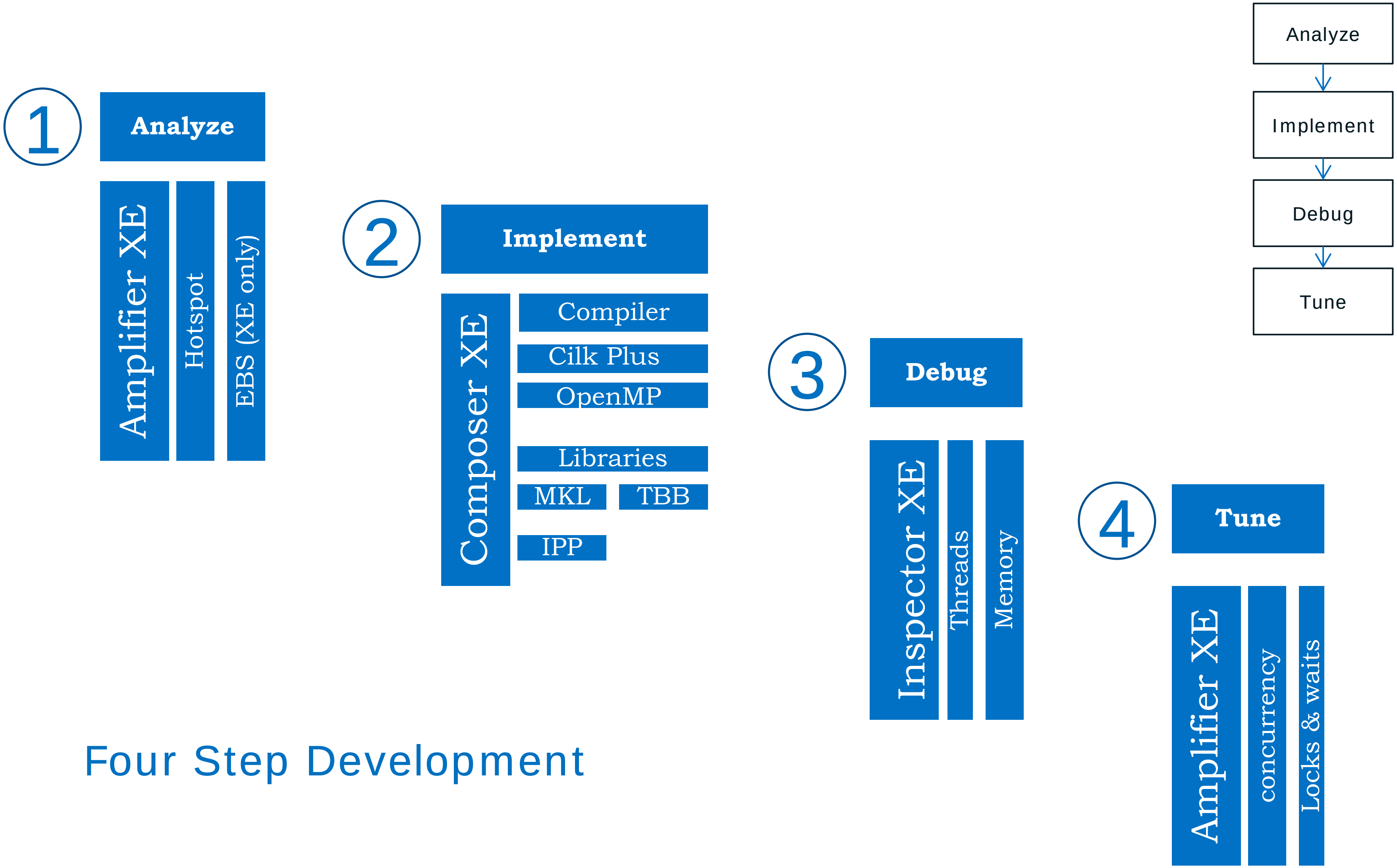
“How can I make my program parallel?”

“Will my code run on any CPU? - compatibility”

Optimization Notice

Speedup using parallelism

Parallel Code



Analyze

Implement

Debug

Tune

1 Analyze

Amplifier XE

Hotspot

EBS (XE only)

2 Implement

Composer XE

Compiler

Cilk Plus

OpenMP

Libraries

MKL

TBB

IPP

3 Debug

Inspector XE

Threads

Memory

4 Tune

Amplifier XE

concurrency

Locks & waits

Four Step Development

Language to help parallelism

Intel® Cilk™ Plus

OpenMP

```
#pragma omp parallel for
for(i=1;i<=4;i++) {
    printf(“Iter: %d”, i);
}
```

Intel® Threading Building Blocks

Intel® MPI

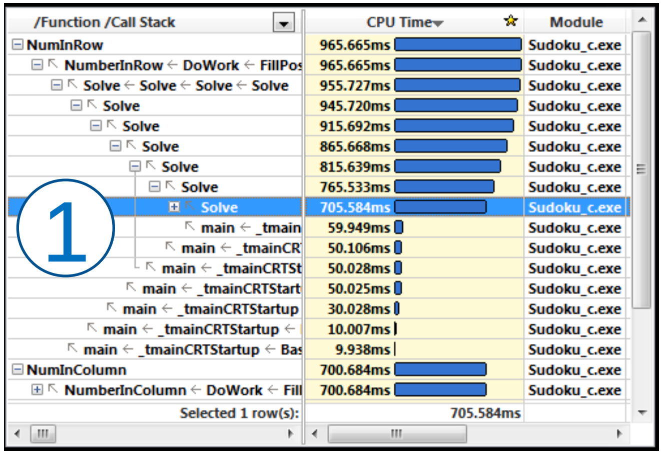
Fortran Coarrays

OpenCL

Native Threads

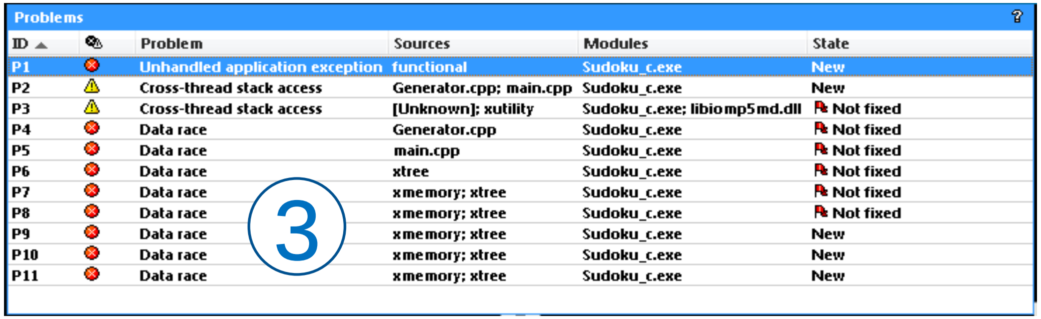
```
cilk_for (int i = 0; i < max_row; i++)
{
    for (int j = 0; j < max_col; j++ )
    {
        p[i][j] = mandel( complex(scale(i), scale(j)));
    }
}
```

An example …



/Function /Call Stack	CPU Time	Module
⊟ NumInRow	965.665ms	Sudoku_c.exe
⊟ NumberInRow ← DoWork ← FillPos	965.665ms	Sudoku_c.exe
⊟ Solve ← Solve ← Solve ← Solve	955.727ms	Sudoku_c.exe
⊟ Solve	945.720ms	Sudoku_c.exe
⊟ Solve	915.692ms	Sudoku_c.exe
⊟ Solve	865.668ms	Sudoku_c.exe
⊟ Solve	815.639ms	Sudoku_c.exe
⊟ Solve	765.533ms	Sudoku_c.exe
⊞ Solve	705.584ms	Sudoku_c.exe
main ← _tmain	59.949ms	Sudoku_c.exe
main ← _tmainCR	50.106ms	Sudoku_c.exe
main ← _tmainCRTSt	50.028ms	Sudoku_c.exe
main ← _tmainCRTStart	50.025ms	Sudoku_c.exe
main ← _tmainCRTStartup	30.028ms	Sudoku_c.exe
main ← _tmainCRTStartup ←	10.007ms	Sudoku_c.exe
main ← _tmainCRTStartup ← Bas	9.938ms	Sudoku_c.exe
⊟ NumInColumn	700.684ms	Sudoku_c.exe
⊞ NumberInColumn ← DoWork ← Fill	700.684ms	Sudoku_c.exe
Selected 1 row(s):	705.584ms	

1

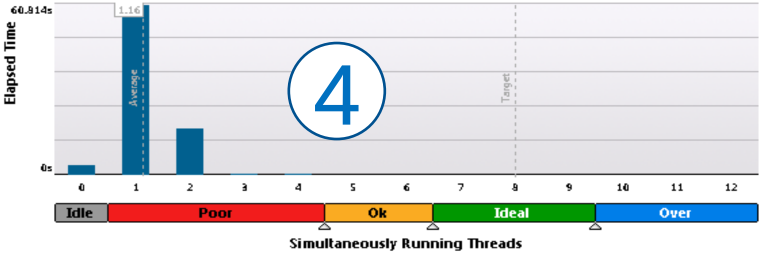


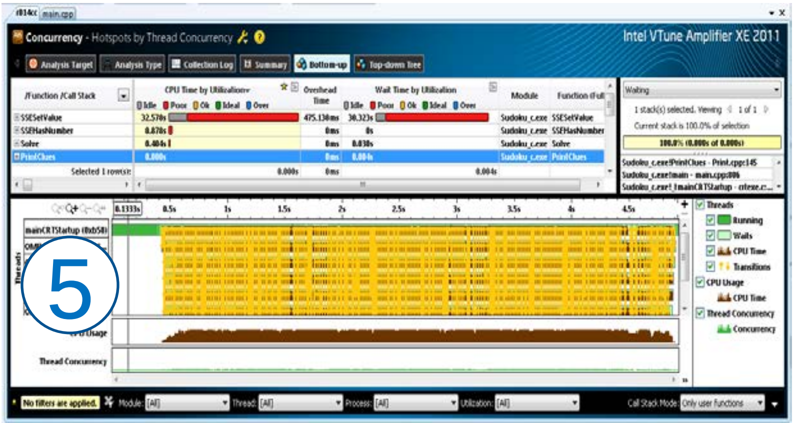
Problems

ID		Problem	Sources	Modules	State
P1	⊗	Unhandled application exception	functional	Sudoku_c.exe	New
P2	⚠	Cross-thread stack access	Generator.cpp; main.cpp	Sudoku_c.exe	New
P3	⚠	Cross-thread stack access	[Unknown]; xutility	Sudoku_c.exe; libiomp5md.dll	Not fixed
P4	⊗	Data race	Generator.cpp	Sudoku_c.exe	Not fixed
P5	⊗	Data race	main.cpp	Sudoku_c.exe	Not fixed
P6	⊗	Data race	xtree	Sudoku_c.exe	Not fixed
P7	⊗	Data race	xmemory; xtree	Sudoku_c.exe	Not fixed
P8	⊗	Data race	xmemory; xtree	Sudoku_c.exe	Not fixed
P9	⊗	Data race	xmemory; xtree	Sudoku_c.exe	New
P10	⊗	Data race	xmemory; xtree	Sudoku_c.exe	New
P11	⊗	Data race	xmemory; xtree	Sudoku_c.exe	New

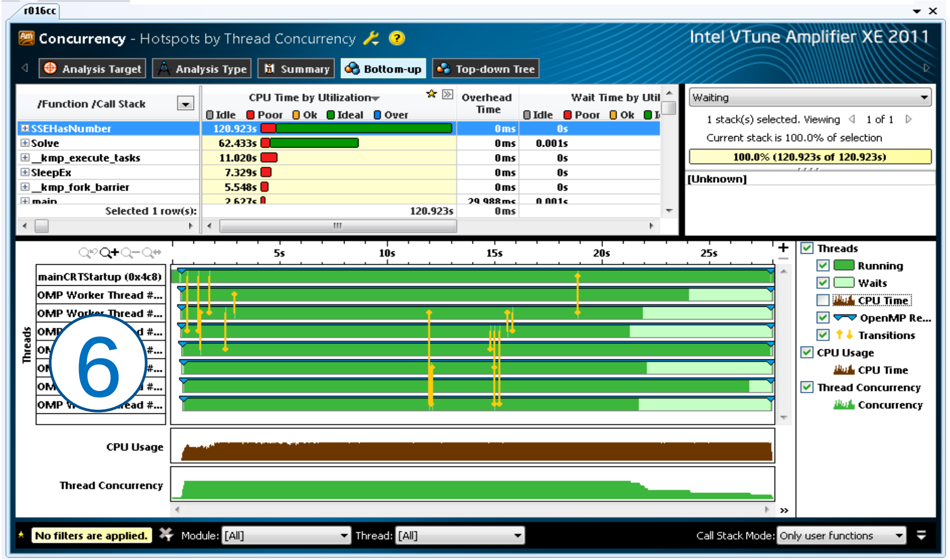
3

2





/Function /Call Stack	CPU Time by Utilization	Overhead Time	Wait Time by Utilization	Module	Function (Full)
SSESetValue	32.570s	475.130ms	30.323s	Sudoku_c.exe	SSESetValue
SSEHasNumber	0.870s	0ms	0s	Sudoku_c.exe	SSEHasNumber
Solve	0.404s	0ms	0.030s	Sudoku_c.exe	Solve
PrintClues	0.000s	0ms	0.000s	Sudoku_c.exe	PrintClues
Selected 1 row(s):	0.000s	0ms	0.004s		



/Function /Call Stack	CPU Time by Utilization	Overhead Time	Wait Time by Util
⊞ SSEHasNumber	120.923s	0ms	0s
⊞ Solve	62.433s	0ms	0.001s
⊞ _kmp_execute_tasks	11.020s	0ms	0s
⊞ SleepEx	7.329s	0ms	0s
⊞ _kmp_fork_barrier	5.548s	0ms	0s
⊞ main	2.627s	29.988ms	0.001s
Selected 1 row(s):	120.923s	0ms	

1. Hotspot Analysis
2. Implement
3. Find Threading Errors

4,5,6. Tune Parallelism

https://makebettercode.com/parallel_landing_required/lib/pdf/5373_IN_ParallelMag_Sudoku_060911.pdf

Optimization Notice

Three Common Requests

“How can I make my program run faster?”

“How can I make my program parallel?”

“Will my code run on any CPU? - compatibility”

Optimization Notice

Will my program run on any CPU?

Compatible Code

Compatibility

- run?

Future Proofing

- build?

OS-agnostic
CPU-agnostic
Language / Standards
Tools
Scalability

- Performance?

Optimization Notice

Thank You

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, 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

intel®