# INTEL ARCHITECTURE AND TOOLS
# JURECA – TUNING FOR THE PLATFORM II

Dr. Heinrich Bockhorst Intel SSG/DPD/

Date: 23.11.2017

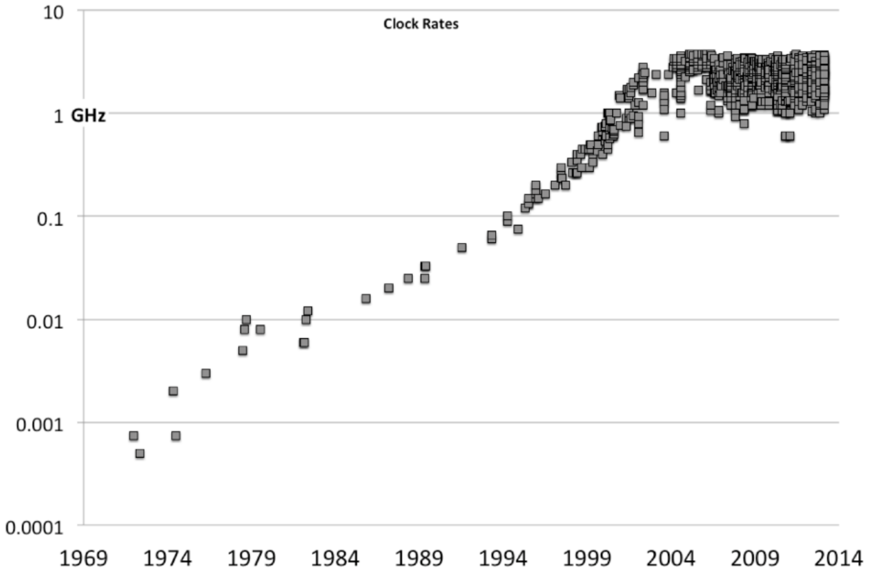
# AGENDA

- Introduction
- Processor Architecture Overview
- Composer XE – Compiler
- Intel Python
- APS – Application Performance Snapshot
- VTune Amplifier XE - Analysis
- Advisor XE  - Vectorization
- Selected Intel® Tools

# PROCESSOR ARCHITECTURE OVERVIEW

# THE "FREE LUNCH" IS OVER, REALLY

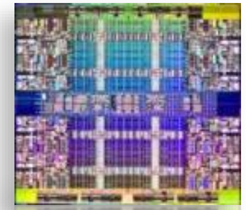
## PROCESSOR CLOCK RATE GROWTH HALTED AROUND 2005

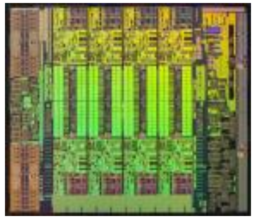


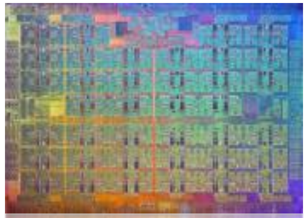

Source: © 2014, James Reinders, Intel, used with permission

Software must be parallelized to realize all the potential performance

# What platform should I use for code modernization?







*Future*

**The world is going parallel – stick with sequential code and you will fall behind.**

| | *Intel® Xeon® Processor E5-2600 v3 Product Family formerly codenamed* **Haswell** | *Intel® Xeon Phi™ x100 Product Family formerly codenamed* **Knights Corner** | *Intel® Xeon® Processor E5-2600 v4 Product Family codenamed* **Broadwell** | *Intel® Xeon Phi™ x200 Product Family codenamed* **Knights Landing** | **Skylake** |
|---|---|---|---|---|---|
| **Cores** | 18 | 61 | 22 | 72 | 28 |
| **Threads/Core** | 2 | 4 | 2 | 4 | 2 |
| **Vector Width** | 256-bit | 512-bit | 256-bit | 512-bit (x2) | 512-bit (x2) |
| **Peak Memory Bandwidth** | 68 GB/s | 352 GB/s | 77 GB/s | >500 GB/s | 228 GB/s |

Both Xeon and KNL are suitable platforms; KNL provides higher scale & memory bandwidth.

# HASWELL PROCESSOR AT JURECA: E5-2680V3

## SEE ARK.INTEL.COM FOR MORE DETAILS

| # Cores | 12 |
|---|---|
| Non-AVX Reference Frequency | 2500 MHz |
| Non-AVX Max Turbo Frequency | 3300 MHz |
| AVX Reference Frequency | 2100 MHz |
| AVX Max Turbo Frequency | 3100 MHz |
| L3 Cache Size | 30 MB |
| QPI | 9.6 GT/s |

E5-2680v3: Turbo bins in GHz for number of cores being used ( see here for more )

| Cores | 1-2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11+ |
|---|---|---|---|---|---|---|---|---|---|---|
| Non-AVX | 3.3 | 3.1 | 3 | 2.9 | 2.9 | 2.9 | 2.9 | 2.9 | 2.9 | 2.9 |
| AVX | 3.1 | 2.9 | 2.8 | 2.8 | 2.8 | 2.8 | 2.8 | 2.8 | 2.8 | 2.8 |

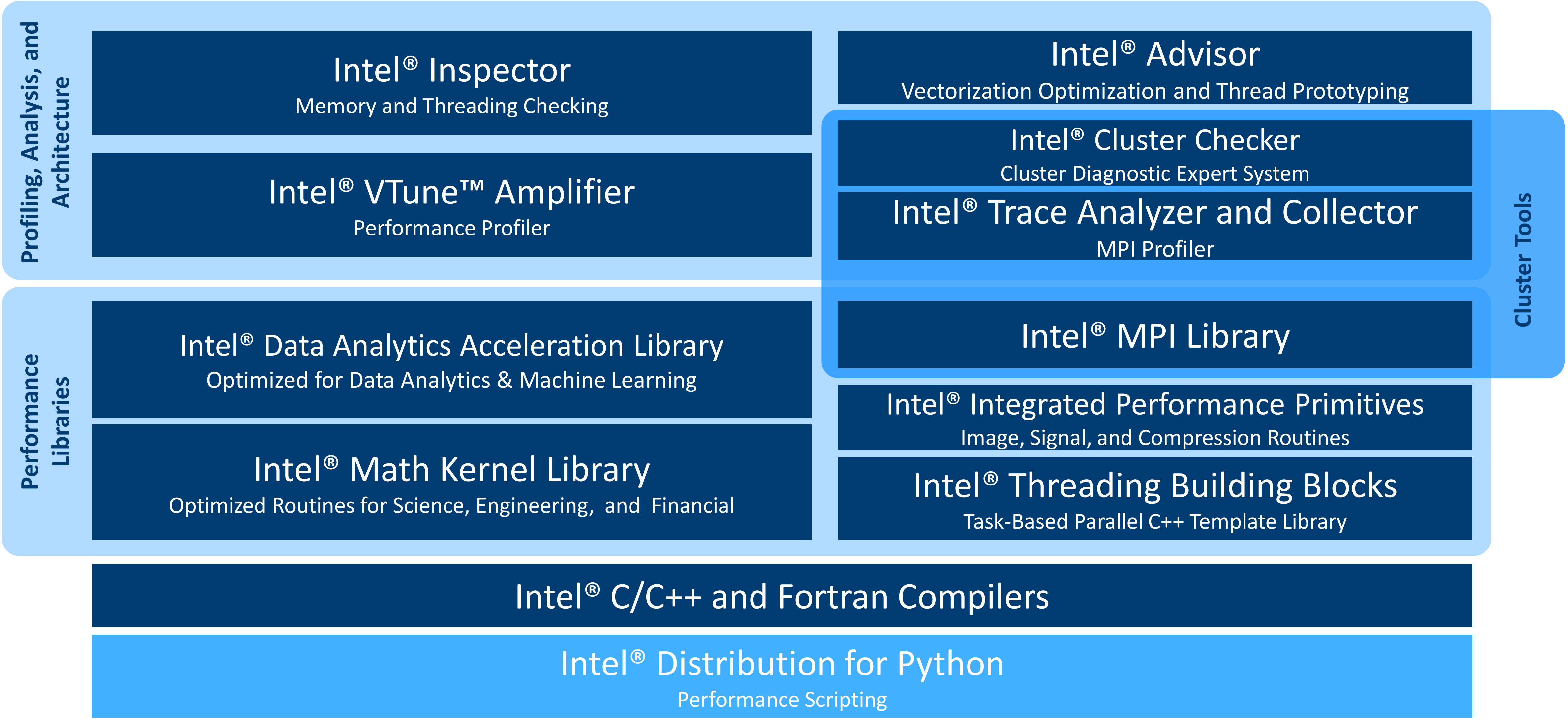
# DOCUMENTATION

- Intel® 64 and IA-32 Architectures Software Developer Manuals:
- Intel® 64 and IA-32 Architectures Software Developer's Manuals
  - Volume 1: Basic Architecture
  - Volume 2: Instruction Set Reference
  - Volume 3: System Programming Guide
- Software Optimization Reference Manual
- Related Specifications, Application Notes, and White Papers

- https://www-ssl.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html?iid=tech_vt_tech+64-32_manuals

# INTEL® PARALLEL STUDIO XE

# Intel® Parallel Studio XE



# INTEL® COMPOSER XE

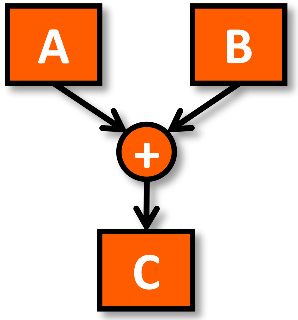
# COMMON OPTIMIZATION OPTIONS

| | Windows* | Linux*, OS* X |
|---|---|---|
| Disable optimization | /Od | -O0 |
| Optimize for speed (no code size increase) | /O1 | -O1 |
| Optimize for speed (default) | /O2 | -O2 |
| High-level loop optimization | /O3 | -O3 |
| Create symbols for debugging | /Zi | -g |
| Multi-file inter-procedural optimization | /Qipo | -ipo |
| Profile guided optimization (multi-step build) | /Qprof-gen<br>/Qprof-use | -prof-gen<br>-prof-use |
| Optimize for speed across the entire program (“prototype switch”)<br>***fast* options definitions changes over time!** | /fast<br>same as: /O3 /Qipo /Qprec-div-, /fp:fast=2 /QxHost) | -fast<br>same as:<br>Linux: -ipo –O3 -no-prec-div –static –fp-model fast=2 -xHost)<br>OS X: -ipo -mdynamic-no-pic -O3 -no-prec-div -fp-model fast=2 -xHost |
| OpenMP support | /Qopenmp | -qopenmp |
| Automatic parallelization | /Qparallel | -parallel |

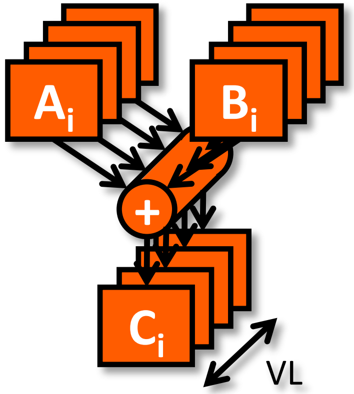
# VECTORIZATION

- Single Instruction Multiple Data (SIMD):
  - Processing vector with a single operation
  - Provides data level parallelism (DLP)
  - Because of DLP more efficient than scalar processing
- Vector:
  - Consists of more than one element
  - Elements are of same scalar data types (e.g. floats, integers, …)
- Vector length (VL): Elements of the vector

**Scalar Processing**




**Vector Processing**



# MANY WAYS TO VECTORIZE

•

**Compiler:**
**Auto-vectorization (no change of code)**

**Compiler:**
**Auto-vectorization hints (`#pragma vector`, …)**

**Compiler:**
**Intel® Cilk™ Plus Array Notation Extensions**

**SIMD intrinsic class**
**(e.g.: `F32vec`, `F64vec`, …)**

**Vector intrinsic**
**(e.g.: `_mm_fmadd_pd(…)`, `_mm_add_ps(…)`, …)**

**Assembler code**
**(e.g.: `[v]addps`, `[v]addss`, …)**

**Ease of use**

**Programmer control**

# BASIC VECTORIZATION SWITCHES I

- Linux*, OS X*: **`-x<feature>`**
  - Might enable Intel processor specific optimizations
  - Processor-check added to “main” routine:
    Application errors in case SIMD feature missing or non-Intel processor with appropriate/informative message
  - Example: -xCORE-AVX2

- Linux*, OS X*: **`-ax<features>`**
  - Multiple code paths: baseline and optimized/processor-specific
  - Multiple SIMD features/paths possible, e.g.: **`-axSSE2,AVX`**
  - Baseline code path defaults to **`–xSSE2`**

# BASIC VECTORIZATION SWITCHES II

- Special switch for Linux*, OS X*: **`-xHost`**
  - Compiler checks SIMD features of current host processor (where built on) and makes use of latest SIMD feature available
  - Code only executes on processors with same SIMD feature or later as on build host
  - As for **`-x<feature>`** if “main” routine is built with
    **`-xHost`** the final executable only runs on Intel processors

# CONTROL VECTORIZATION

- Verify vectorization:
  - Globally:
    Linux*, OS X*: **-qopt-repot[n]**
    `check for additional options (man icc)!`
- Advanced:
  - Ignore vector dependencies (IVDEP):
    C/C++: **#pragma ivdep**
    Fortran: **!DIR$ IVDEP**
  - "Enforce" vectorization:
    C/C++: **#pragma simd**
    Fortran: **!DIR$ SIMD**

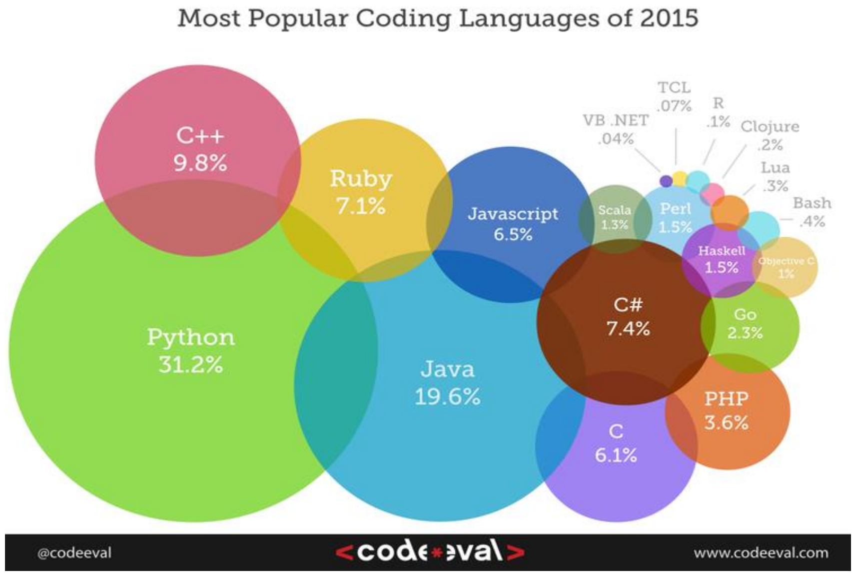
`Check new generic SIMD OpenMP pragmas!`

# INTEL® DISTRIBUTION FOR PYTHON*

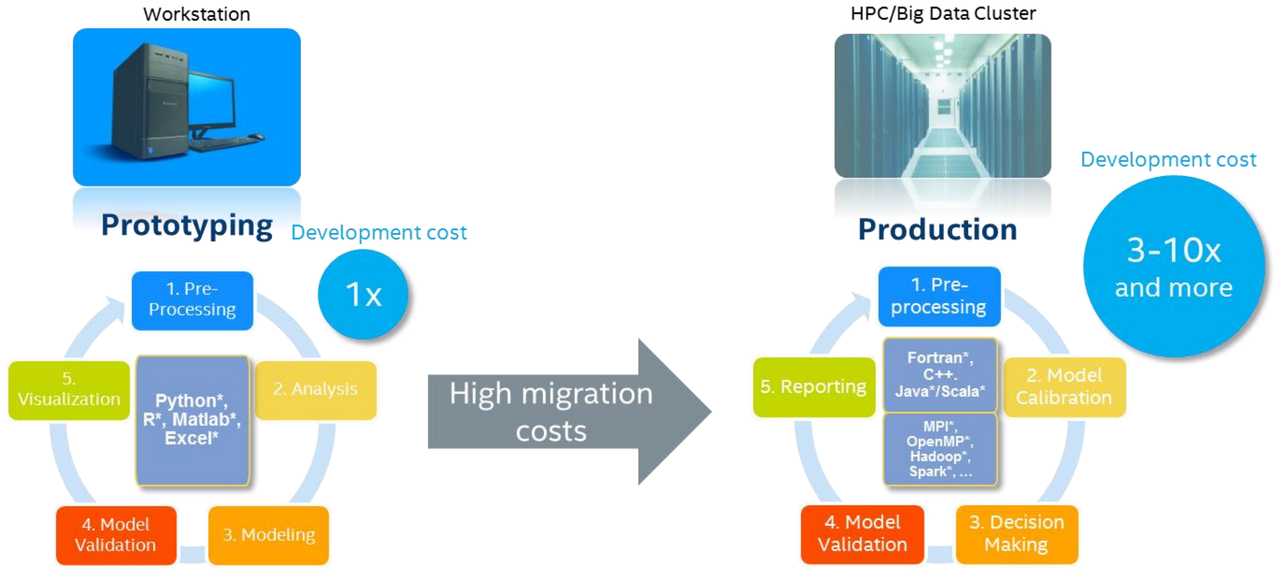
# PYTHON* LANDSCAPE

- Challenge#1:
  - Domain specialists are not professional software programmers.

**Adoption of Python** continues to grow among domain specialists and developers for its productivity benefits

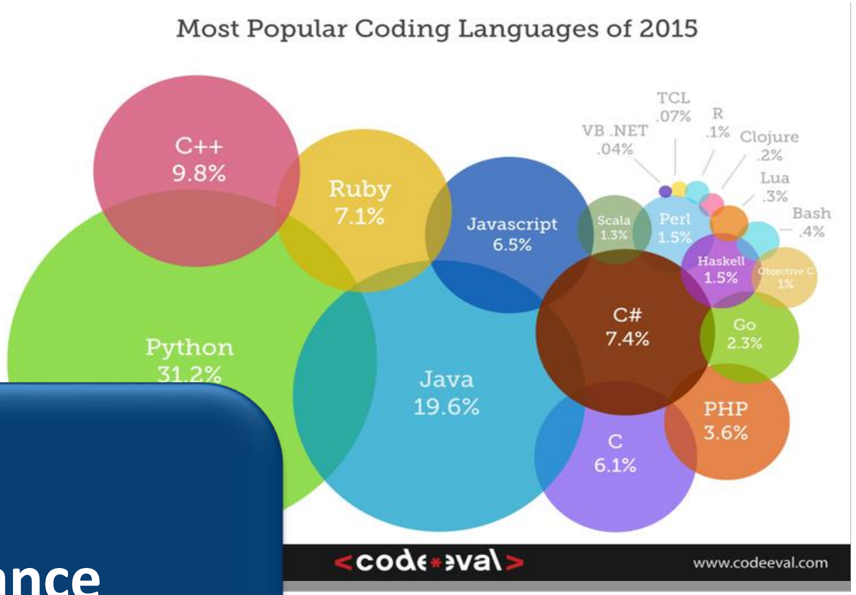



- Challenge#2:
  - Python performance limits migration to production systems



# PYTHON* LANDSCAPE

**Adoption of Python** continues to grow among domain specialists and developers for its productivity benefits

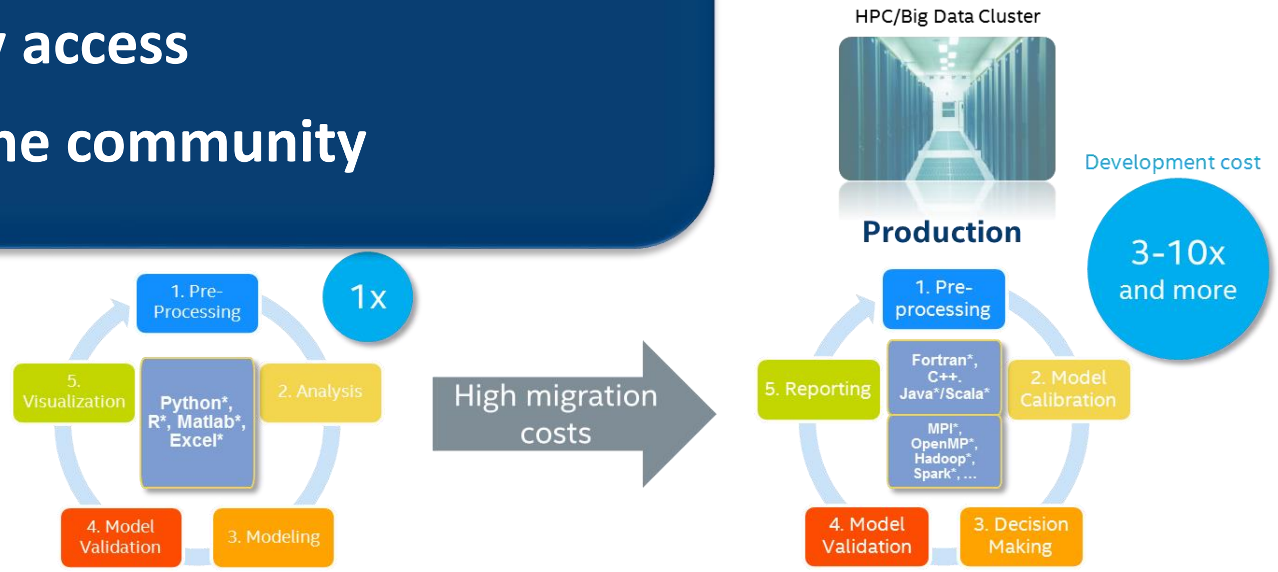



- Challenge#1:
  - Domain specialists are not prof
  software programmers.

- Challenge#2:
  - Python performance limits mig
  production systems

## Intel's solution is to…

- **Accelerate Python performance**
- **Enable easy access**
- **Empower the community**



# ACCESS MULTIPLE OPTIONS FOR FASTER PYTHON*

## INCLUDED IN INTEL® DISTRIBUTION FOR PYTHON

- Accelerate with native libraries
  - NumPy, SciPy, Scikit-Learn, Theano, Pandas, pyDAAL
  - Intel® MKL, Intel® DAAL
- Exploit vectorization and threading
  - Cython + Intel C++ compiler
  - Numba + Intel LLVM

Better/Composable threading

- Cython, Numba, Pyston
- Threading composability for MKL, CPython, Blaze/Dask, Numba

*"I expected Intel's numpy to be fast but it is significant that plain old python code is much faster with the Intel version too."*

Dr. Donald Kinghorn, Puget Systems [Review]

- Multi-node parallelism
  - Mpi4Py, Distarray
  - Intel native libraries: Intel MPI
- Integration with Big Data, ML platforms and frameworks
  - Spark, Hadoop, Trusted Analytics Platforr

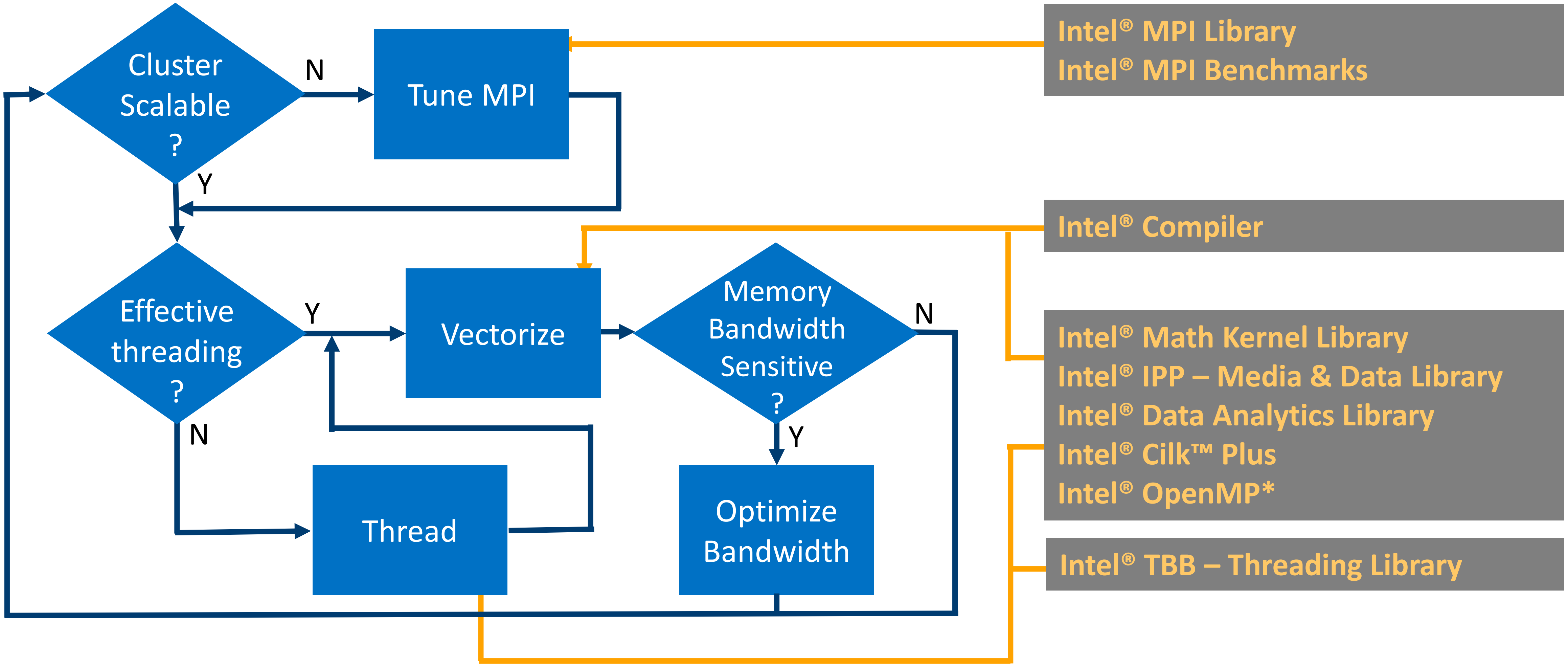
Work in Progress

- Better performance profiling
  - Extensions for profiling mixed Python & native/JIT codes

# WHICH TOOL SHOULD I USE?

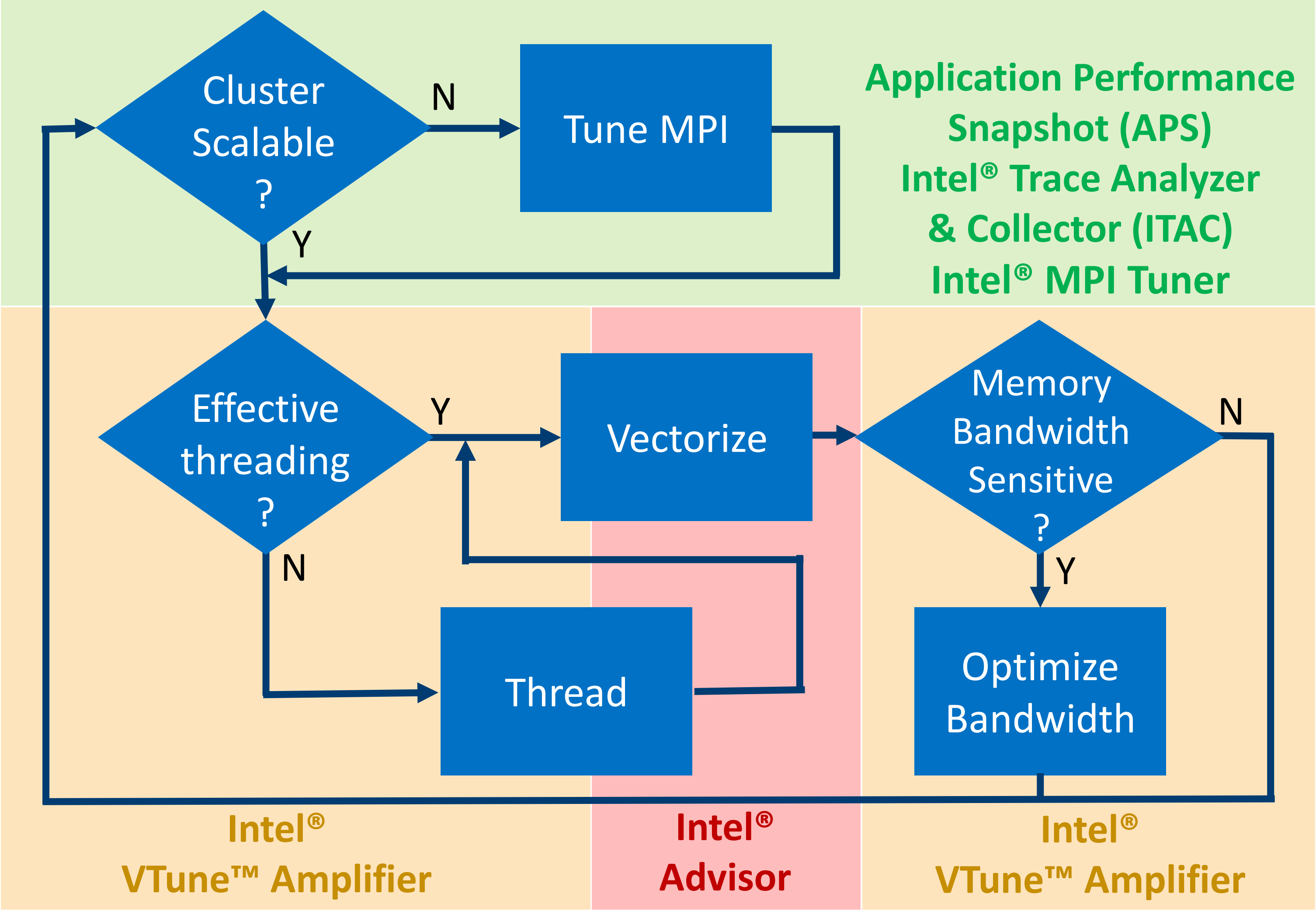
# TOOLS FOR HIGH-PERFORMANCE IMPLEMENTATION

## INTEL® PARALLEL STUDIO XE



# PERFORMANCE ANALYSIS TOOLS FOR DIAGNOSIS

## INTEL® PARALLEL STUDIO XE



# SUGGESTED ORDER OF TUNING STEPS

1. Application Performance Snapshot (APS)
2. Intel Trace Analyzer and Collector (ITAC) (MPI scalability issues)
3. VTune analysis: Advanced Hotspots (OpenMP profiling)
4. Intel Advisor (Vectorization)
5. VTune analysis: HPC (Adding Bandwidth and some Memory Analysis)
6. VTune analysis (Memory Access, General Exploration)

Check Code with Intel Inspector (Threading, Memory) and MPI with Message Checker (part of ITAC)

# Application Performance Snapshot Adds MPI

## Data in One Place: MPI+OpenMP+Memory Floating Point—Intel® VTune™ Amplifier

### Quick & Easy Performance Overview

- Does the app need performance tuning?

### MPI & non-MPI Apps†

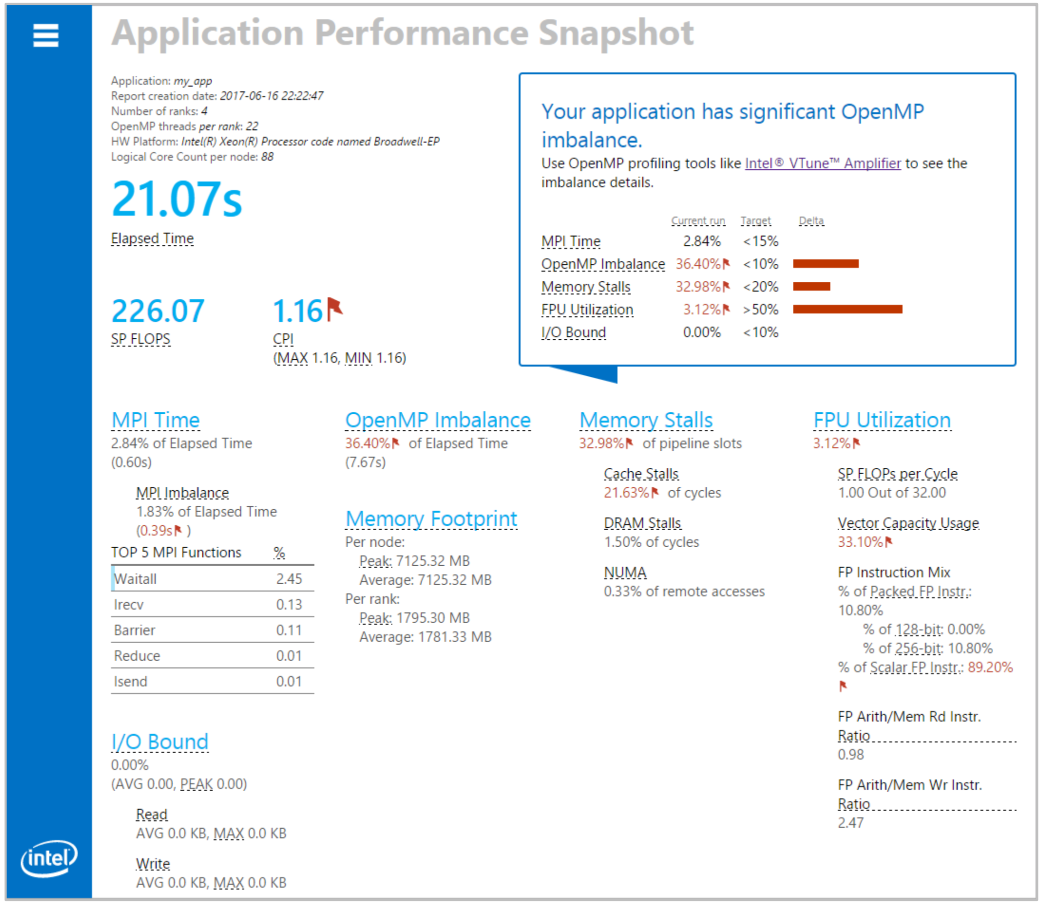
- Distributed MPI with or without threading
- Shared memory applications

### Popular MPI Implementations Supported

- Intel® MPI Library
- MPICH & Cray MPI

### Richer Metrics on Computation Efficiency

- CPU (processor stalls, memory access)
- FPU (vectorization metrics)




†MPI supported only on Linux*

# Boost Distributed Application Performance with Intel® MPI Library
## Performance, Scalability & Fabric Flexibility

- Standards Based Optimized MPI Library for Distributed Computing
  - Built on open source MPICH Implementation
  - Tuned for low latency, high bandwidth & scalability
  - Multi fabric support for flexibility in deployment

- What's New in 2018 edition[1]
  - Up to **11x** faster in job start-up time
  - Up to **25%** reduction in job finalization time
  - Supports the latest Intel® Xeon® Scalable processor

**Learn More: software.intel.com/intel-mpi-library**

[1]See following benchmarks slide for more details

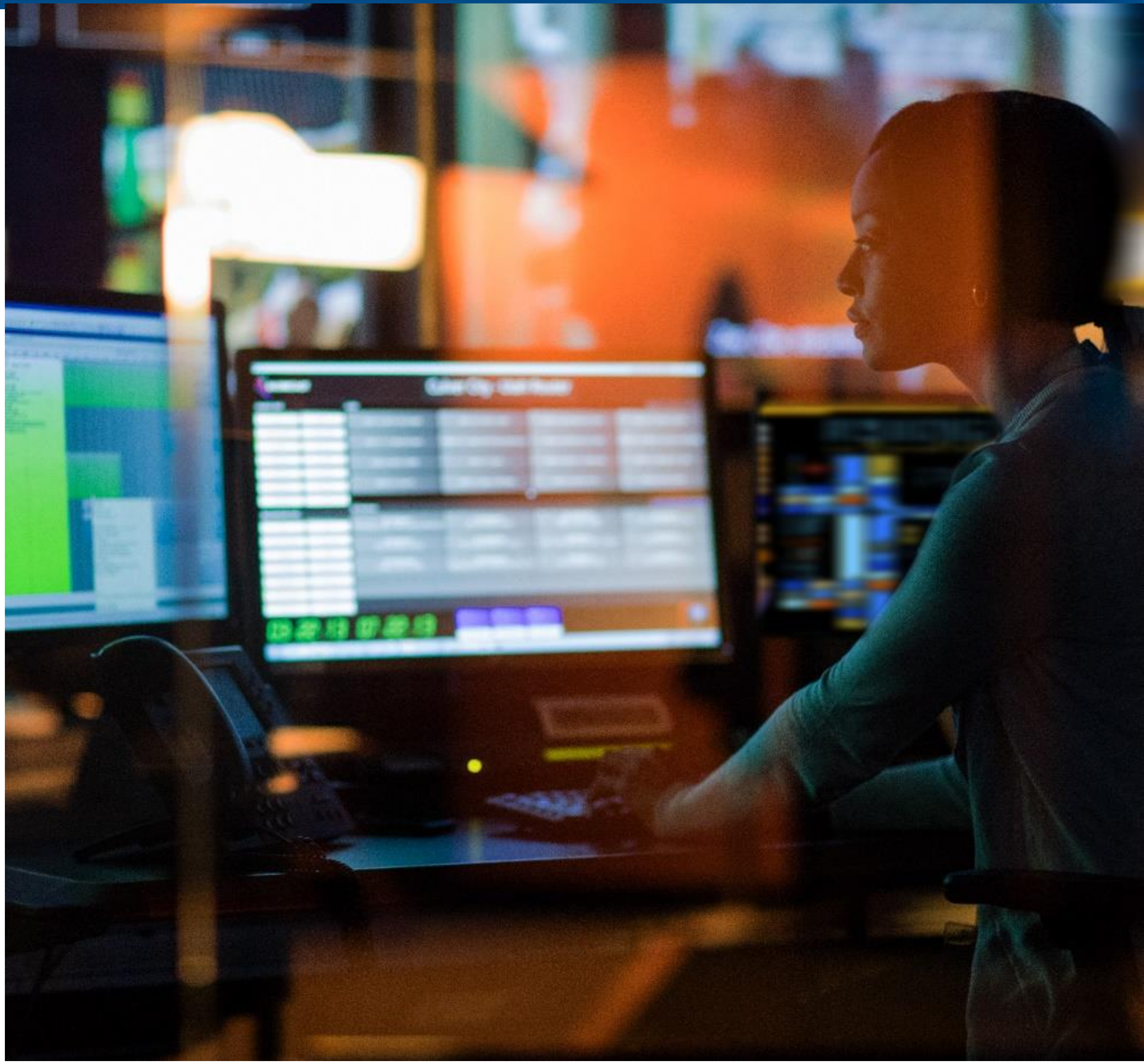
# Profile & Analyze High Performance MPI Applications

## Intel® Trace Analyzer & Collector

- Powerful Profiler, Analysis & Visualization Tool for MPI Applications
  - Low overhead for accurate profiling, analysis & correctness checking
  - Easily visualize process interactions, hotspots & load balancing for tuning & optimization
  - Workflow flexibility: Compile, Link or Run

- What's New in 2018 edition
  - Support of OpenSHMEM* applications
  - Supports the latest Intel® Xeon® Scalable and Intel® Xeon Phi™ processors

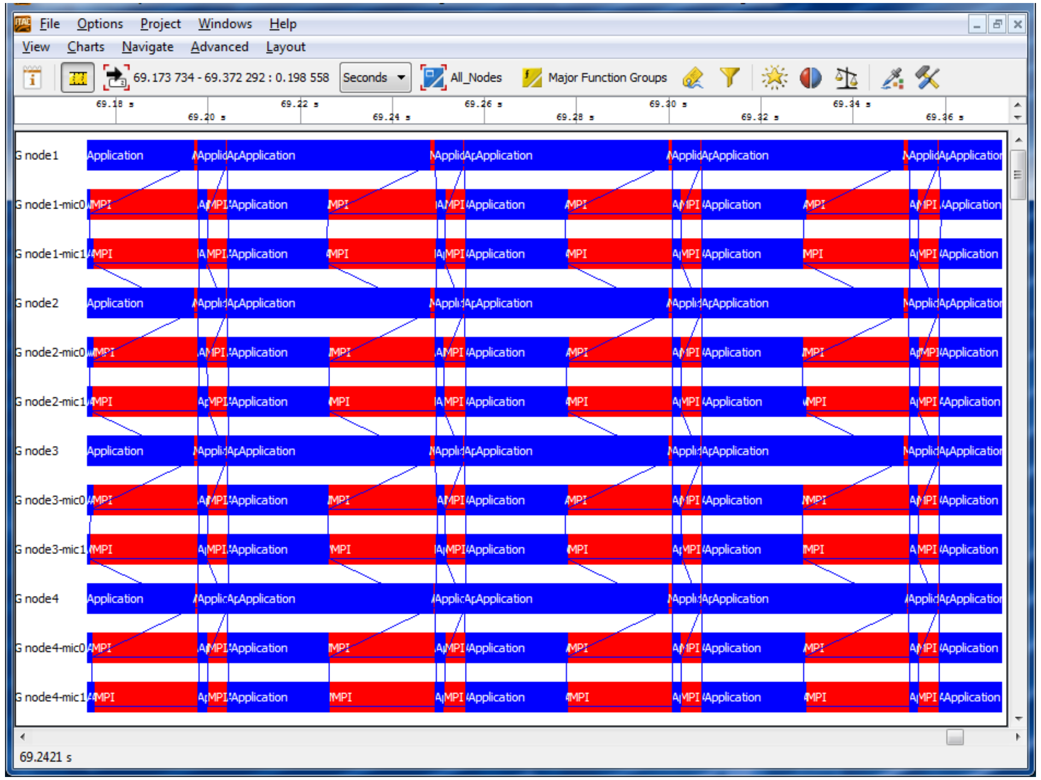
Learn More: software.intel.com/intel-trace-analyzer



# Efficiently Profile MPI Applications
## Intel® Trace Analyzer & Collector

- Helps Developers
  - ❿ Visualize & understand parallel application behavior
  - ❿ Evaluate profiling statistics & load balancing
  - ❿ Identify communication hotspots

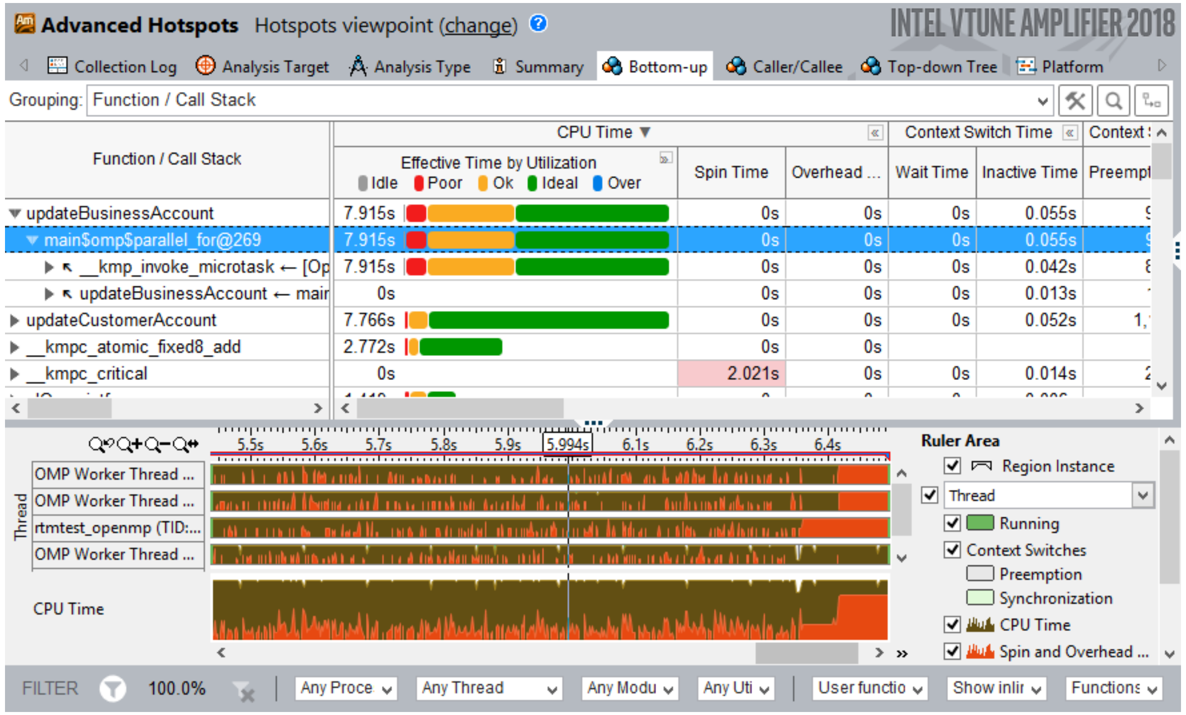
- Features
  - Event-based approach
  - Low overhead
  - Excellent scalability
  - Powerful aggregation & filtering functions
  - Idealizer
  - Scalable






intel Software

# Analyze & Tune Application Performance & Scalability with Intel® VTune™ Amplifier—Performance Profiler




- Save Time Optimizing Code
  - Accurately profile C, C++, Fortran*, Python*, Go*, Java*, or any mix
  - Optimize CPU, threading, memory, cache, storage & more
  - Save time: rich analysis leads to insight
- New for 2018 edition (partial list)
  - Quick metrics for shared & distributed memory apps
  - Cross-OS analysis – e.g. analyze Linux* from Windows* or macOS*
  - Profile inside containers

Learn More: software.intel.com/intel-vtune-amplifier-xe

# RICH SET OF PROFILING FEATURES FOR MULTIPLE MARKETS

## INTEL® VTUNE™ AMPLIFIER—PERFORMANCE PROFILER

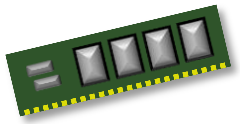
### Basic Profiling
- Hotspots

### Threading Analysis
- Concurrency, Locks & Waits
- OpenMP, Intel® Threading Building Blocks

### Micro Architecture Analysis
- Cache, branch prediction, …

### Vectorization + Intel® Advisor
- FLOPS estimates

### MPI + Intel® Trace Analyzer & Collector
- Scalability, imbalance, overhead



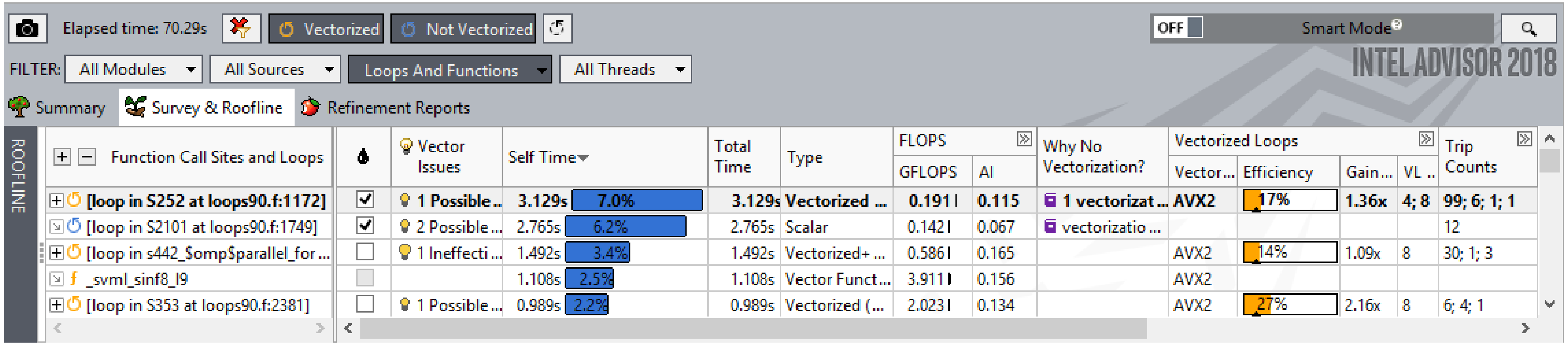
- Use Memory Efficiently
  - Tune data structures & NUMA

- Optimize for High Speed Storage
  - I/O and compute imbalance
- Intel® Media SDK Integration
  - Meaningful media stack metrics
- Low Overhead Java*, Python*, Go*
  - Managed + native code
- Containers
  - Docker*, Mesos*, LXC*

# Get Breakthrough Vectorization Performance

## Intel® Advisor—Vectorization Advisor

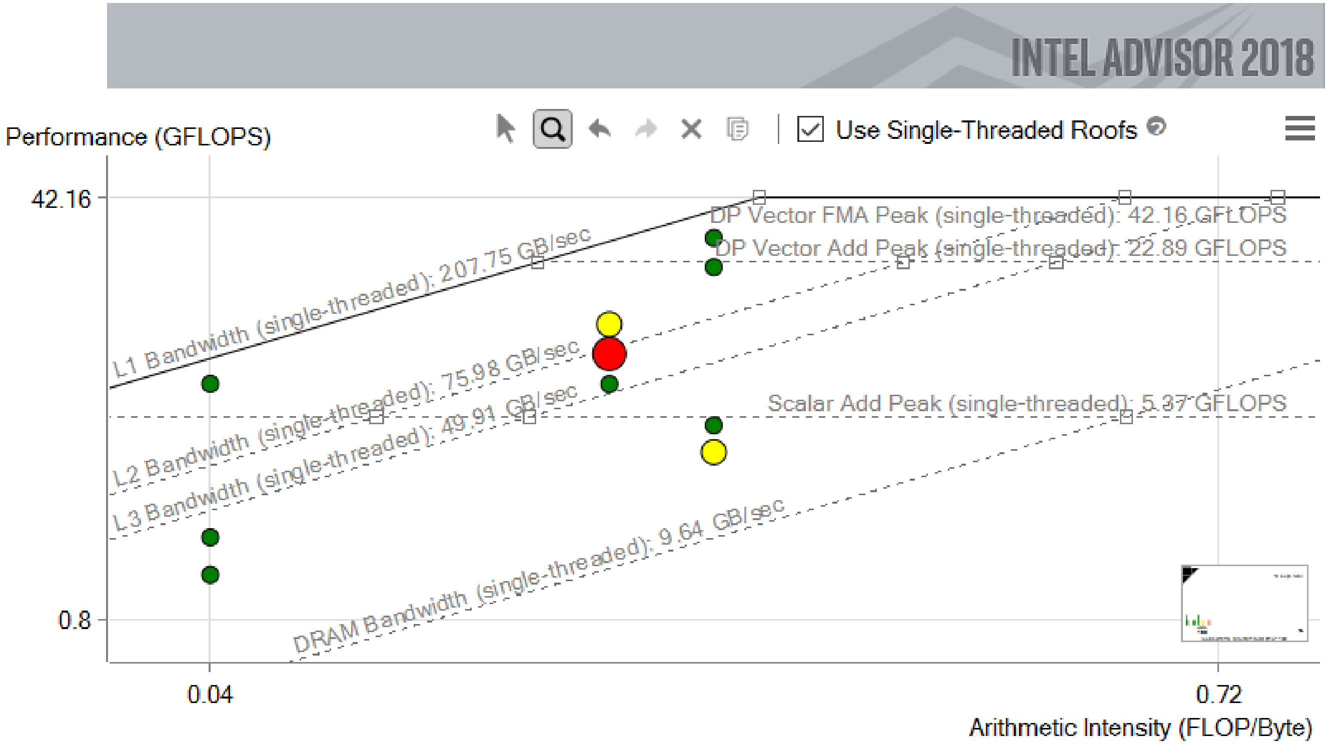
- Faster Vectorization Optimization
  - Vectorize where it will pay off most
  - Quickly ID what is blocking vectorization
  - Tips for effective vectorization
  - Safely force compiler vectorization
  - Optimize memory stride
- Data & Guidance You Need
  - Compiler diagnostics + Performance Data + SIMD efficiency
  - Detect problems & recommend fixes
  - Loop-Carried Dependency Analysis
  - Memory Access Patterns Analysis




Optimize for Intel® AVX-512 with or without access to AVX-512 hardware

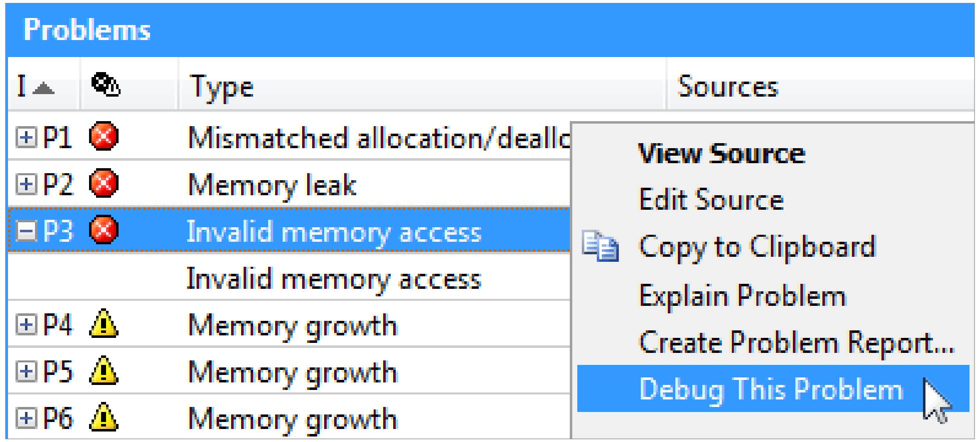
# Find Effective Optimization Strategies
## Cache-aware Roofline Analysis—Intel® Advisor

- Roofline Performance Insights
  - Highlights poor performing loops
  - Shows performance ‘headroom’ for each loop
    - Which can be improved
    - Which are worth improving
  - Shows likely causes of bottlenecks
  - Suggests next optimization steps



# Debug Memory & Threading with Intel® Inspector
## Find & Debug Memory Leaks, Corruption, Data Races, Deadlocks

### Debugger Breakpoints




Diagnose in hours instead of months

Learn More: intel.ly/inspector-xe

- Correctness Tools Increase ROI by 12%-21%[1]
  - Errors found earlier are less expensive to fix
  - Races & deadlocks not easily reproduced
  - Memory errors are hard to find without a tool
- Debugger Integration Speeds Diagnosis
  - Breakpoint set just before the problem
  - Examine variables and threads with the debugger
- What's New in 2018 edition
  - Fewer false positives
  - C++ 17 std::shared_mutex added
  - Windows SRW Locks added

[1]**Cost Factors – Square Project Analysis -** *CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab NIST: National Institute of Standards & Technology: Square Project Results*

# HOW TO START?

- Compile with minimal options and check timing
- Compile with  -xHost and –opt-report=5 and check timing
- Compile with –xHost and –no-vec
  disables vectorization. Compare with previous timing
- Use: VTune Amplifier XE: $ module load VTune/<version>
- Use: Advisor XE: $ module load   Advisor/<version>
- Google for Intel related topics → Intel Developer Zone etc.
- Book: **Optimizing HPC Applications with Intel Cluster Tools**

Software