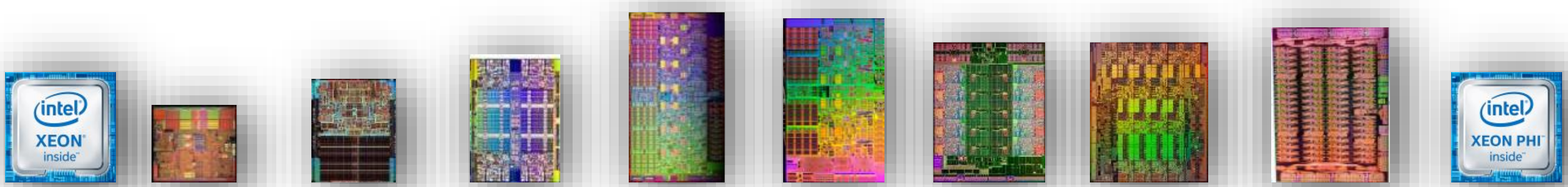
A Brief Introduction to Intel® Xeon Phi™ Architecture

Zongyan Cao 曹宗雁
SSG/PRC/DRD/SAE/HPC
Intel Corporation
January 26, 2016

Parallel is the Path Forward

Intel® Xeon® and Intel® Xeon Phi™ Product Families are both going parallel



(die sizes not to scale, for illustration only)

	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor code-named Sandy Bridge EP	Intel® Xeon® processor code-named Ivy Bridge EP	Intel® Xeon® processor code-named Haswell EP	Intel® Xeon Phi™ coprocessor code-named Knights Corner	Intel® Xeon Phi™ processor & coprocessor code-named Knights Landing[1]
Core(s)	1	2	4	6	8	12	18	61	60+
Threads	2	2	8	12	16	24	36	244	240+
SIMD Width	128	128	128	128	256	256	256	512	512

MORE CORES → MORE THREADS → WIDER VECTORS

- 1. Not launched or in planning.
- Product specification for launched and shipped products available on ark.intel.com.

A Walk Through Many Integrated Cores

- Architecture Overview
- Core & ISA
- Software & Ecosystem
- Programming Models

Knights Corner Architecture Overview

Intel® Xeon Phi™ Coprocessor PCI Express* Cards

A coprocessor card contains the coprocessor, memory, voltage regulators, system management and may contain a thermal solution

Compatible with PCI Express* 2.0 interface

Four versions available:

- Passive card (P)
- Active card (A) – has heat sink and a fan
- No Thermal Solution (X) – allows OEMs to customize the heat sink
- Dense Form Factor – No heat sink solution (DFF)

Architecture of an Intel® Xeon Phi™ Coprocessor (Recap)

Cache

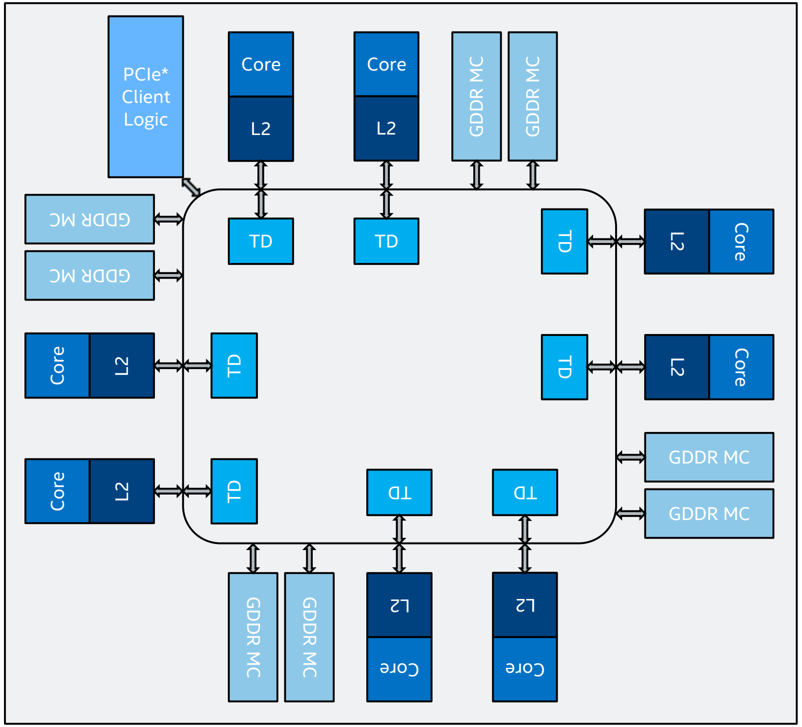
- 32 KB L1 / 512 KB L2 per core
- Fully coherent

Core Communication

- Bi-directional ring buffer
- 8/16 GB GDDR5 shared by all cores

PCIe*

- Gen2
- 16 channels



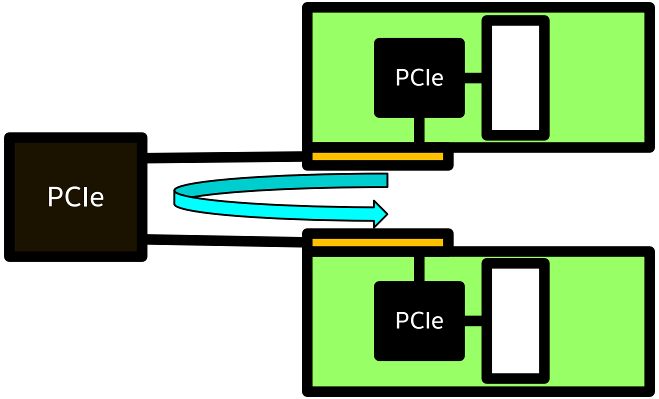
Memory Controllers

- 12 or 16 channels of GDDR5 memory (SKU dependent)
- Memory is installed double sided (clamshell)
- 8 or 16GB memory uses 32 x 256KB memory components
    - 16 devices on primary side
    - 16 more on the secondary side
- 6GB uses 24 memory chips in clamshell mode
- All buses are using ECC or other types of error correction

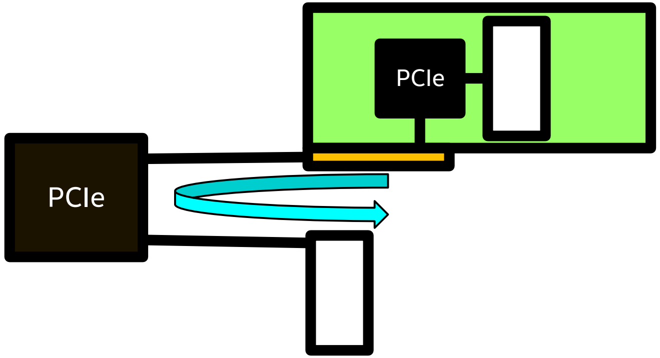
Direct Memory Access (DMA)

Coprocessor supports full peer-to-peer DMA

- Integrated DMA engine capable of ~13.5** GB/s max sustained bandwidth
- Up to 256 Bytes (4 cache lines)



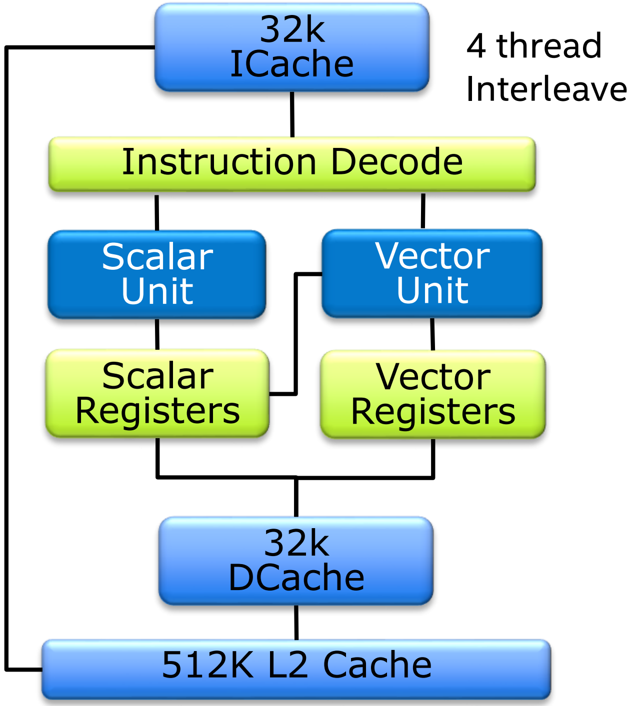
Card-to-card DMA



Card-to-system DMA
(or system-to-card)

Knights Corner Core & ISA

Core



- Pentium scalar instruction set (x87)
- Fully functional
- In order-operation
- Full 64bit addressing
- 4 HW threads/core
- Two pipelines:
  - Scalar
  - Vector/Scalar

ISA/Registers

Standard Intel64 Registers (EM64T)

- rax
- rbx
- rcx
- rdx
- rsi
- rdx
- rsp
- rbp

- r8
- r9
- r10
- r11
- r12
- r13
- r14
- r15

+ 32 512bit SIMD Registers:

- zmm0

  ...

- zmm31

+ 8 mask registers (16bit wide)

- k0 (special, don't use)

  ...

- k7

No xmm (SSE/128bit) or ymm (AVX/256bit) registers! x87 present

Xeon and Xeon Phi ISA/Registers

SSE/AVX/AVX-512

511 256	256 128	127 0
ZMM0	YMM0	XMM0
ZMM1	YMM1	XMM1
…	…	…
ZMM15	YMM15	XMM15
ZMM16	YMM16	XMM16
…	…	…
ZMM31	YMM31	XMM31

IMCI

511 0
ZMM0
ZMM1
…
ZMM31

15 0
K0
…
K7

mask registers

Standard Intel64 Registers (EM64T)

Vector Instructions

Caveat:

The following pages introduce the KNC 512bit SIMD operation on the basis of its machine language.

It is not encouraged to write code in assembly!

(Unless you find a VERY good reason and have a HIGH pain threshold!)

It is sometimes a good idea, though, to have a look at the assembly output of the compiler in order to find out why a program does or doesn’t perform so well.

Vector Instructions

Vector Instruction Format

- <u>3 operand form with explicit destination register</u>

  instruction destination, source1, source2

  → Source registers are not destroyed

  → Very compact code

- <u>(Most) MIC instructions can be masked</u>

  instruction destination {mask}, source1, source2

  → Result of masking is non-destructive, i.e. destination values are preserved

- <u>Example</u>:

  vaddps zmm1{k1},zmm2,zmm3

dest mask source1 source2

Vector Instructions

Vector Instruction Format

- Memory source reference

  The second source (or rather the last) operand may be specified as a memory reference

  instruction destination, source1, [address]

- Swizzle and permutation modifiers

  Additionally to the mask register, a swizzle or permutation modifier may be given

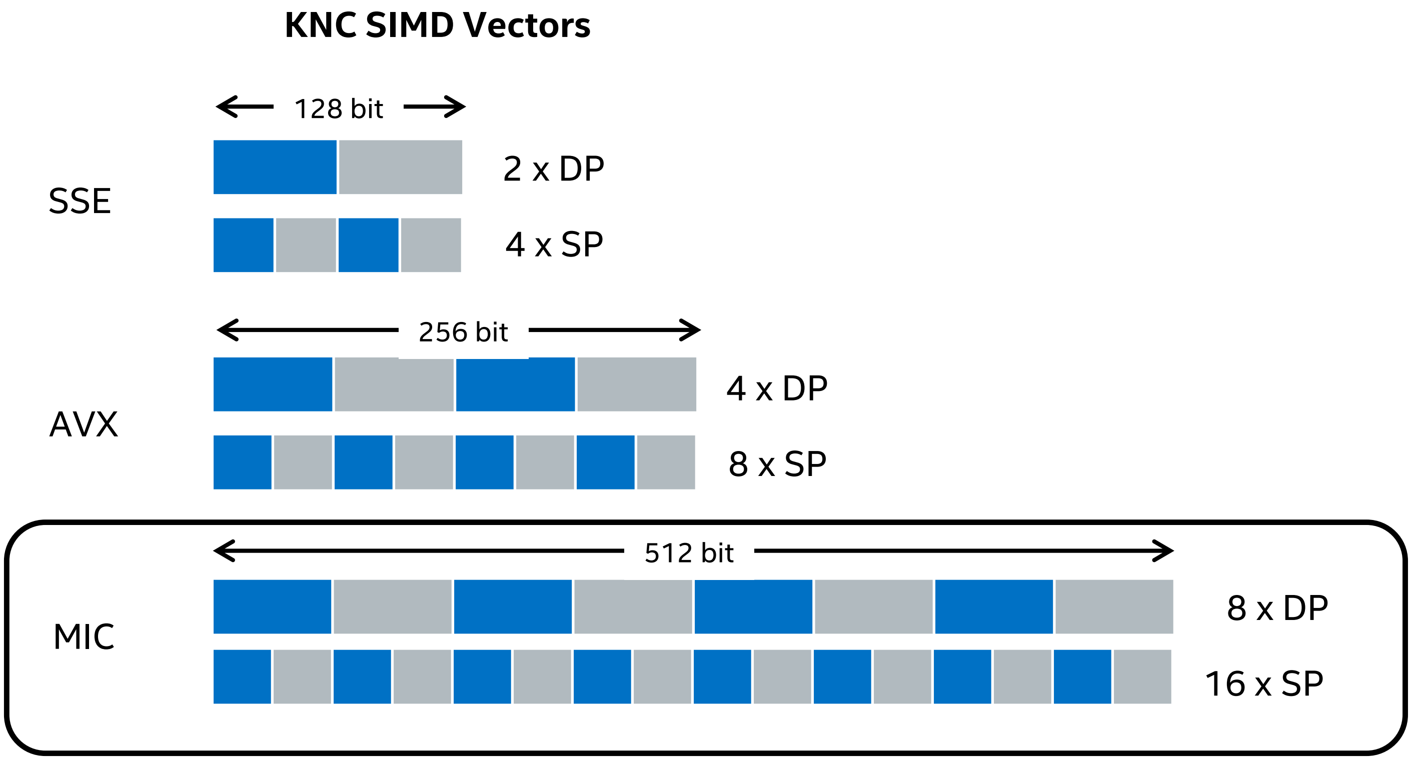
  instruction destination, mask,source1, source2, imm

or

instruction destination, mask,source1, source2{mod}

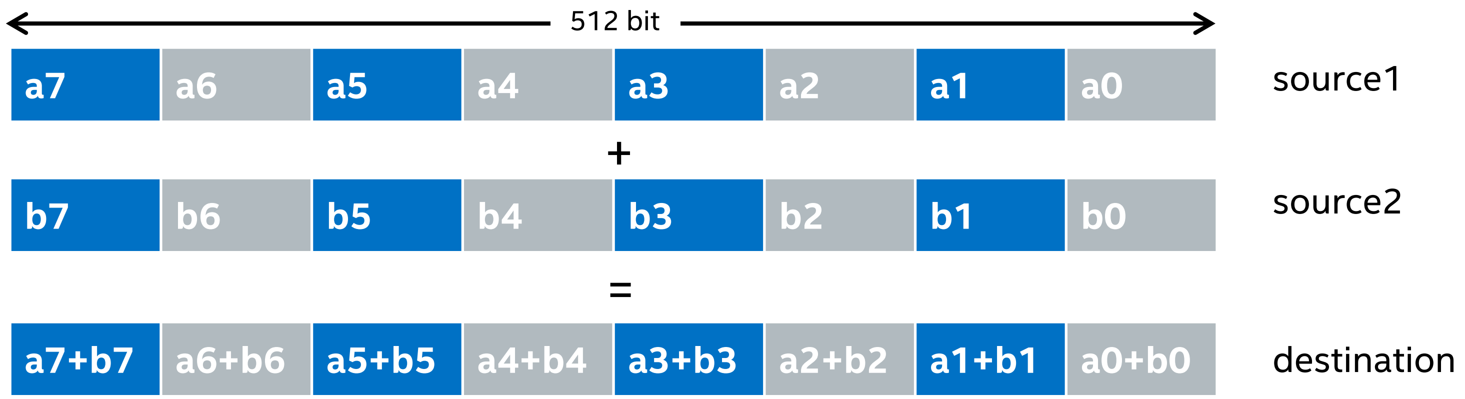
(details on swizzle and permutation below)

Vector Processing Unit and ISA



Vector Processing Unit and ISA

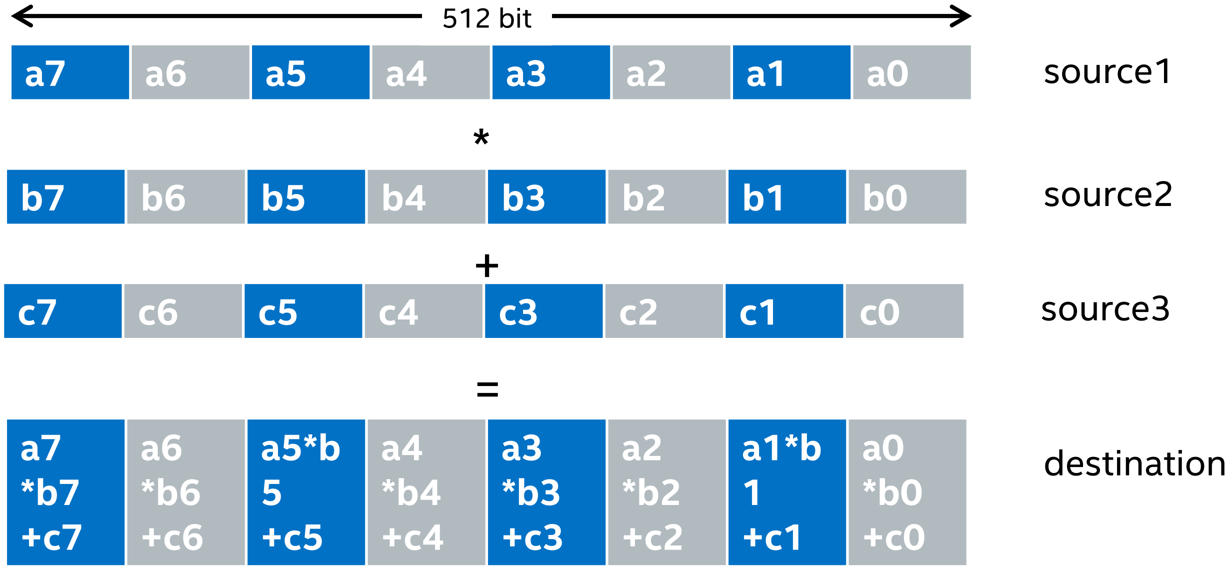
KNC SIMD Vectors Basic Arithmetic



Basic arithmetic SIMD instruction usage is trivial and identical to SSE or AVX.
vaddps, vsubps, vmulps, …
vaddpd, vsubpd, vmulpd, …

Vector Processing Unit and ISA

KNC SIMD Fused Multiply and Add/Subtract



vfmadd213ps destination,source1,source2,source3

Vector Processing Unit and ISA

FMA/FMS instructions come in different flavors

vfmadd132ps source1,source2,source3

Translates to $source1 = source1 \times source3 + source2$

vfmadd213ps source1,source2,source3

Translates to $source1 = source2 \times source1 + source3$

vfmadd231ps source1,source2,source3

Translates to $source1 = source2 \times source3 + source1$

Memory references only apply to source3!

Vector Processing Unit and ISA

KNC SIMD Vectors Masking

512 bit

a7	a6	a5	a4	a3	a2	a1	a0	source1
+								
b7	b6	b5	b4	b3	b2	b1	b0	source2
1	0	1	0	0	1	0	1	mask
=								
a7+b7	d6	a5+b5	d4	a3	a2+b2	d1	a0+b0	destination

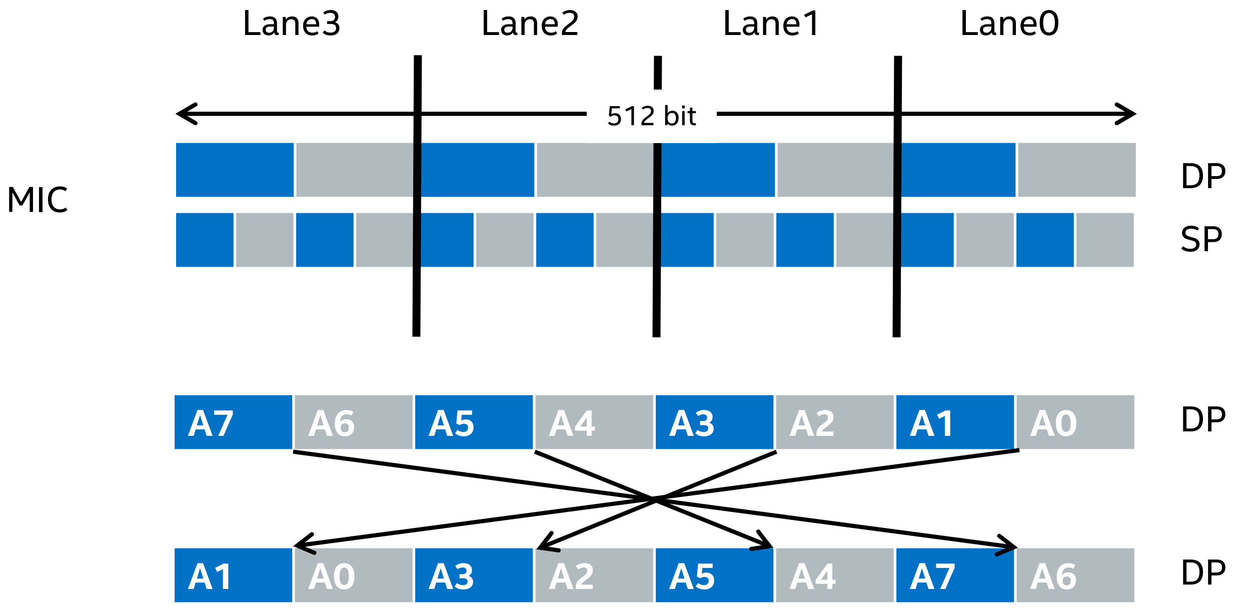
vaddps zmm0{k1}, zmm1, zmm2

Masking allows non-destructive writing to the destination (unlike AVX).

Every Knight's Corner instruction has write masking

Vector Processing Unit and ISA

KNC SIMD Vector Permutation

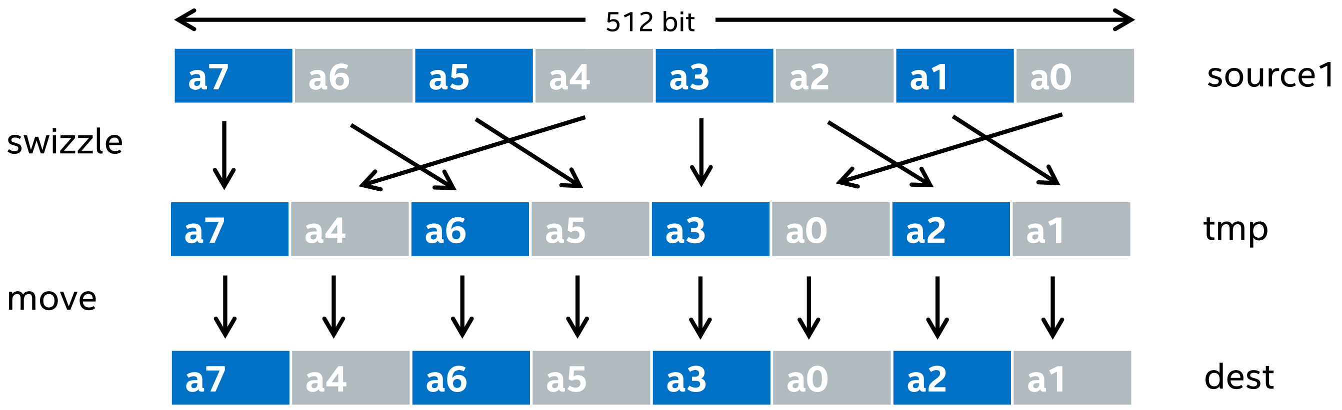


vpermf32x4 zmm1, zmm2, 27

Knights Corner Architecture Overview
Vector Processing Unit and ISA

KNC SIMD Vector Swizzling

Swizzling is the modification of the last source. One can easily envision it as creating a modified copy for the following operation.
Example: *vmovapd zmm1, zmm0{dacb}*



Knights Corner Architecture Overview
Vector Processing Unit and ISA

Swizzles can be applied to all kinds of operations (but not all) and may also be masked!

Original Array:	A B C D E F G H
_MM_SWIZ_REG_**NONE:**	A B C D E F G H
_MM_SWIZ_REG_**CDAB:**	B A D C F E H G
_MM_SWIZ_REG_**BADC:**	C D A B G H E F
_MM_SWIZ_REG_**AAAA:**	A A A A E E E E
_MM_SWIZ_REG_**BBBB:**	B B B B F F F F
_MM_SWIZ_REG_**CCCC:**	C C C C G G G G
_MM_SWIZ_REG_**DDDD:**	D D D D H H H H
_MM_SWIZ_REG_**DACB:**	B C A D F G E H

(Attention: Output is printf-order: lowest element is to the left!)

Vector Instructions – Intel® Intrinsics Guide

intel Intrinsics Guide

The Intel Intrinsics Guide is an interactive reference tool for Intel intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, and more - without the need to write assembly code. ×

Filter by ISA.

Technologies

- [ ] MMX
- [ ] SSE
- [ ] SSE2
- [ ] SSE3
- [ ] SSSE3
- [ ] SSE4.1
- [ ] SSE4.2
- [ ] AVX
- [ ] AVX2
- [ ] FMA
- [ ] AVX-512
- [x] KNC
- [ ] SVML
- [ ] Other

Filter by functionality.

Categories

- [ ] Application-Targeted
- [ ] Arithmetic
- [ ] Bit Manipulation
- [ ] Cast
- [ ] Compare
- [ ] Convert
- [ ] Cryptography
- [ ] Elementary Math Functions
- [ ] General Support

_mm_search ?

`__m512d _mm512_abs_pd (__m512d v2)` vpandq

Synopsis

```
__m512d _mm512_abs_pd (__m512d v2)
#include "zmmintrin.h"
Instruction: vpandq zmm {k}, zmm, m512
CPUID Flag : KNCNI
```

Description

Finds the absolute value of each packed double-precision (64-bit) floating-point element in v2, storing the results in dst.

Operation

```
FOR j := 0 to 7
        i := j*64
        dst[i+63:i] := ABS(v2[i+63:i])
ENDFOR
dst[MAX:512] := 0
```

Expand any intrinsic for a detailed description.

`__m512d _mm512_mask_abs_pd (__m512d src, __mmask8 k, __m512d v2)` vpandq

`__m512 _mm512_abs_ps (__m512 v2)` vpandd

`__m512 _mm512_mask_abs_ps (__m512 src, __mmask16 k, __m512 v2)` vpandd

`__m512i _mm512_adc_epi32 (__m512i v2, __mmask16 k2, __m512i v3, __mmask16 * k2_res)` vpadcd

`__m512i _mm512_mask_adc_epi32 (__m512i v2, __mmask16 k1, __mmask16 k2, __m512i v3, __mmask16 * k2_res)` vpadcd

Available at: http://software.intel.com/sites/landingpage/IntrinsicsGuide/

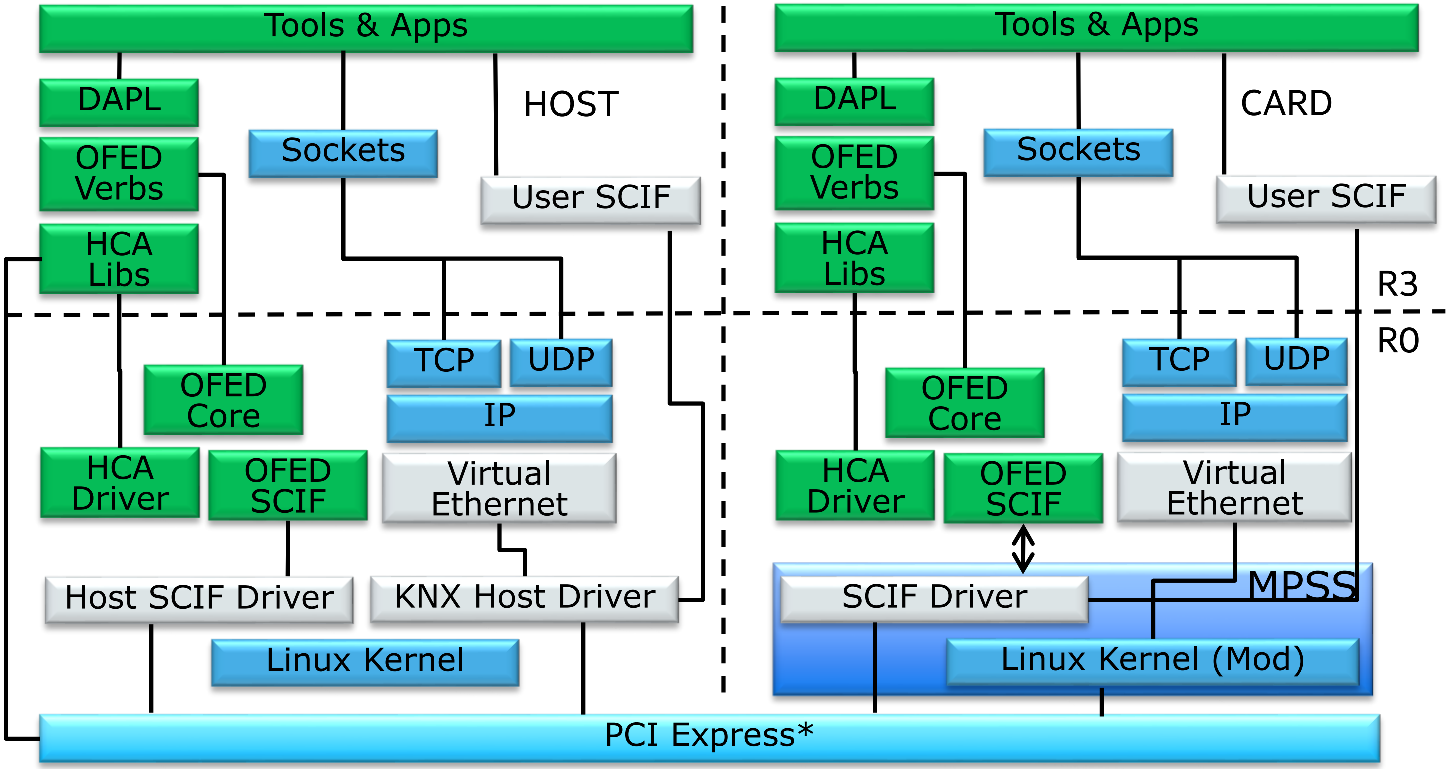
Software & Ecosystem

Software Architecture

KNC Software Architecture Components

- MIC Platform Software Stack (MPSS)
  - A Linux* micro-OS for the KNC device
  - Supports TCP/UDP IP, Sockets,…
  - Symmetric communications Inteface (SCIF)
- Development Tools
  - Intel® Fortran & C++ Compilers
    - OpenMP*, Cilk™ Plus, Theading Building Blocks
  - Intel® Debugger
  - Intel® MPI
  - Intel® Libraries (e.g. MKL)
  - Vtune Performance Analysis
  - …

Software Architecture



Software Architecture

Linux* “uOS”

- Based on standard kernel (from kernel.org)
- Minimal embedded Linux environment ported to MIC
- As few modifications as possible
- GPL’ed
- Extendable with loadable kernel modules (such as the SEP sampling collector)
- Busybox shell environment
- Linux* Standard Base(LSB) Core libraries: glibc, libstdc++, libgcc_s, libz, libcurses, libpam

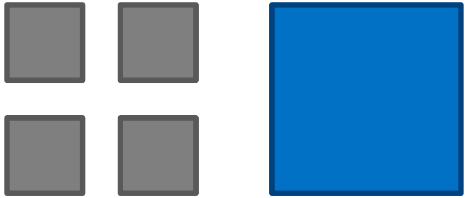
Coprocessor OS

- Fully featured Linux* kernel derived form 2.6.38
- BusyBox* toolkit (may be replaced by bash in the future)
- Drivers for virtual Ethernet
- Ethernet Bridging possible
- Local filesystem on RAM installed from host configuration file (/opt/intel/mic/filesystem/base), but NFS support available for importing a host directory for sharing
- Intel® Coprocessor Communication Link driver for InfiniBand* HCAs
- Driver for event based sampling with Intel® VTune™ Amplifier XE performance profiler
- SSH access

Programming Models

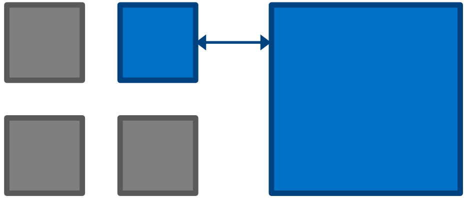
Coprocessor Programming Models (Recap)

Native



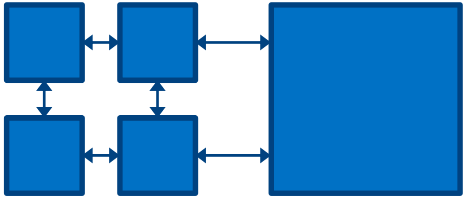
- Target Code:
  Highly parallel (threaded and vectorized) throughout.
- Potential Bottleneck:
  Serial/scalar code.

Offload



- Target Code:
  Mostly serial, but with expensive parallel regions.
- Potential Bottleneck:
  PCIe* data transfers.

Symmetric



- Target Code:
  Highly parallel and performs well on both platforms.
- Potential Bottleneck:
  Load imbalance.

Profiling Your Programs with Intel® Parallel Studio XE

Zongyan Cao 曹宗雁
SSG/PRC/DRD/SAE/HPC
Intel Corporation
January 26, 2016

Intel® Parallel Studio XE Overview



Intel® Parallel Studio XE 2016 Composer Edition	Intel® Parallel Studio XE 2016 Professional Edition	Intel® Parallel Studio XE 2016 Cluster Edition
Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*
	Intel® Advisor XE Intel® Inspector XE Intel® VTune™ Amplifier XE	Intel® Advisor XE Intel® Inspector XE Intel® VTune™ Amplifier XE Intel® MPI Library Intel® Trace Analyzer and Collector

For more information: http://intel.ly/perf-tools

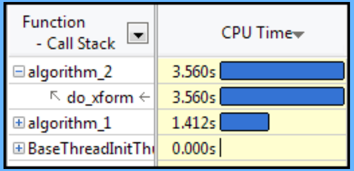
Intel® VTune™ Amplifier XE

Intel® VTune™ Amplifier XE

Performance Profiler

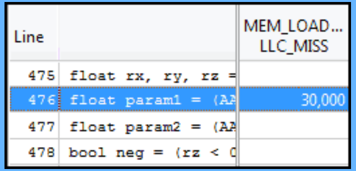
Where is my application…

Spending Time?



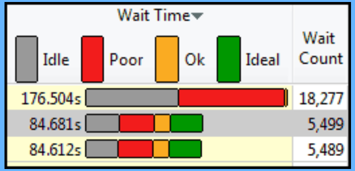
- Focus tuning on functions taking time
- See call stacks
- See time on source

Wasting Time?



- See cache misses on your source
- See functions sorted by # of cache misses

Waiting Too Long?



- See locks by wait time
- Red/Green for CPU utilization during wait

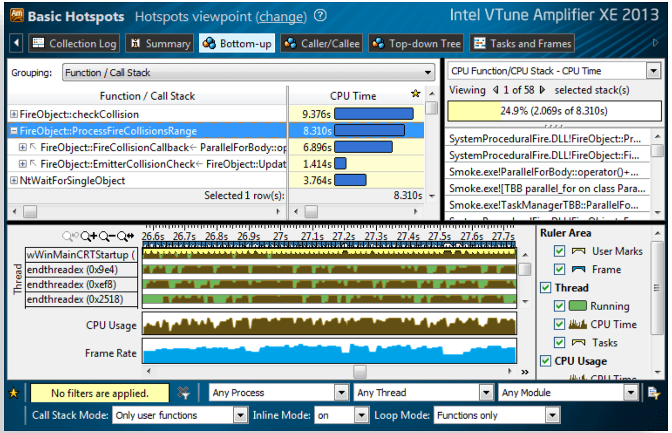
- Windows & Linux
- Low overhead
- No special recompiles

Advanced Profiling For Scalable Multicore Performance

Intel® VTune™ Amplifier XE

Tune Applications for Scalable Multicore Performance

- **Fast, Accurate Performance Profiles**
  - Hotspot (Statistical call tree)
  - Call counts (Statistical)
  - Hardware-Event Sampling
- **Thread Profiling**
  - Visualize thread interactions on timeline
  - Balance workloads
- **Easy set-up**
  - Pre-defined performance profiles
  - Use a normal production build
- **Find Answers Fast**
  - Filter extraneous data
  - View results on the source / assembly
- **Compatible**
  - Microsoft, GCC, Intel compilers
  - C/C++, Fortran, Assembly, .NET, Java
  - Latest Intel® processors and compatible processors[1]
- **Windows or Linux**
  - Visual Studio Integration (Windows)
  - Standalone user i/f and command line
  - 32 and 64-bit



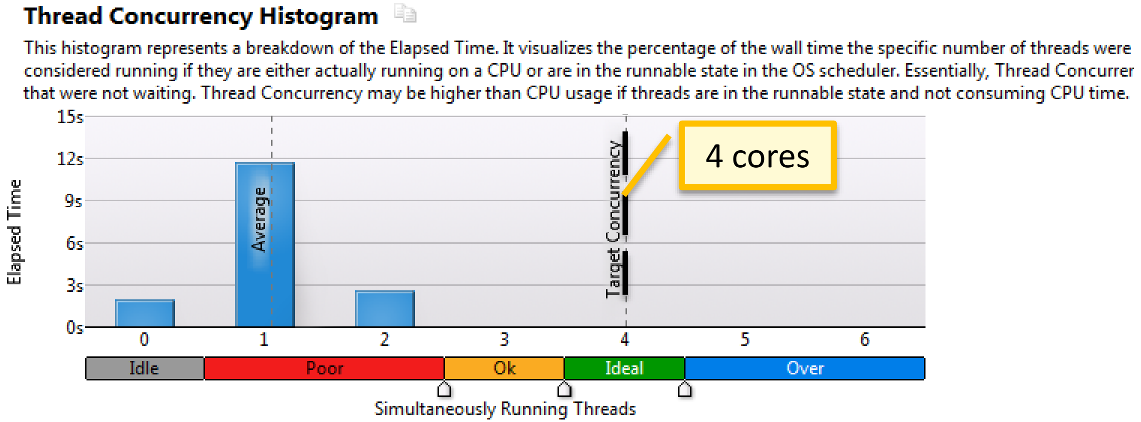
[1] IA32 and Intel® 64 architectures.
Many features work with compatible processors.
Event based sampling requires a genuine Intel® Processor.

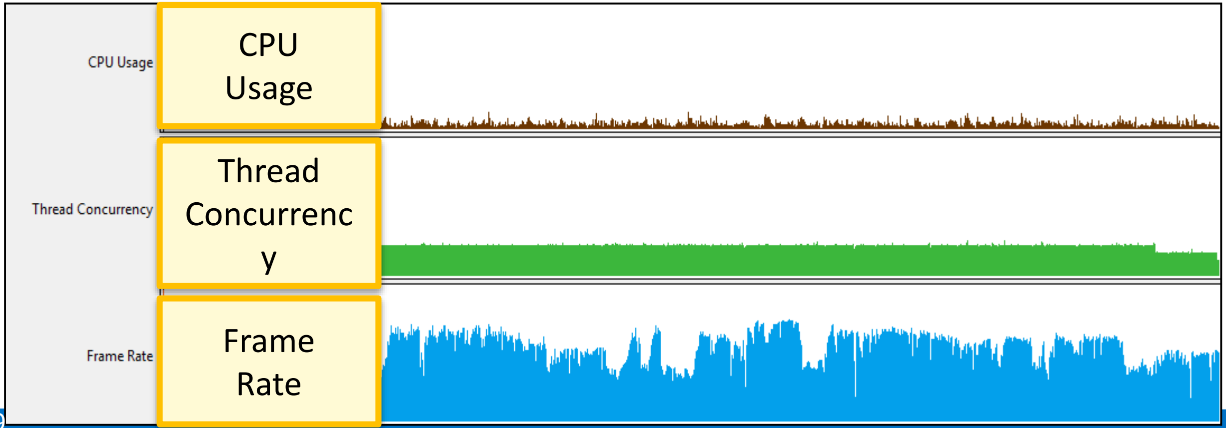
A set of instruments to identify performance problems

Quick Overview

Intel® VTune™ Amplifier XE

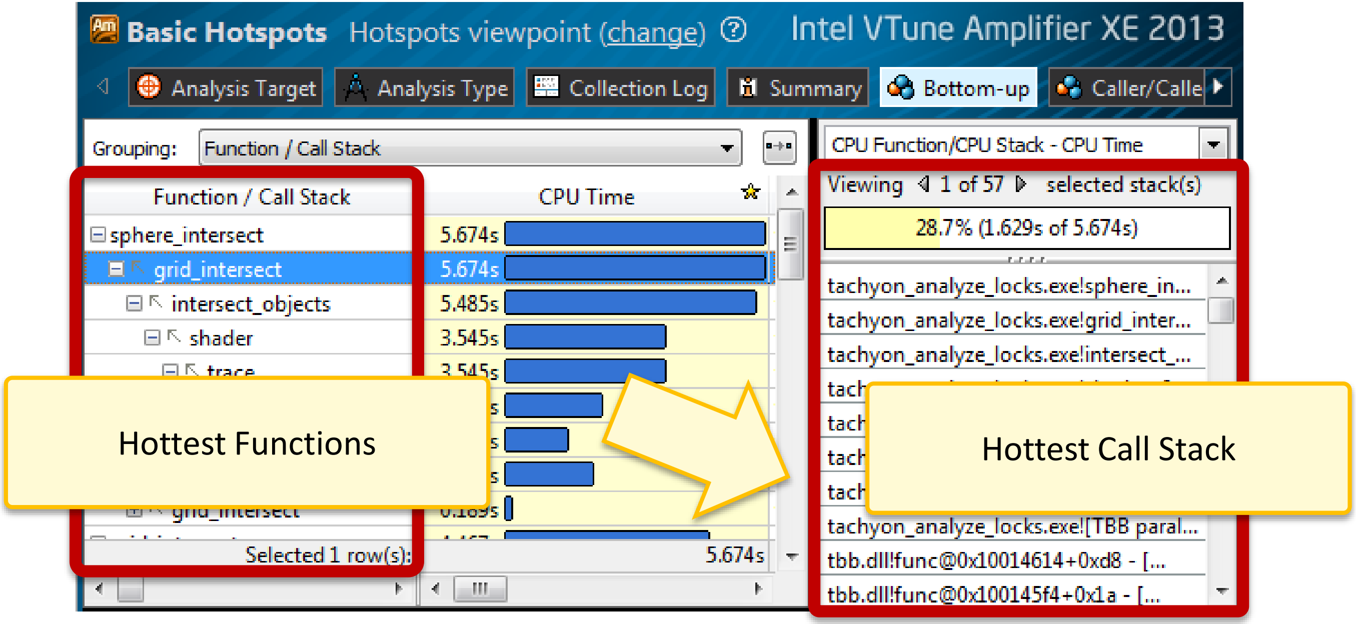
Get a quick snapshot





Intel® VTune™ Amplifier XE

Identify hotspots



Quickly identify what is important

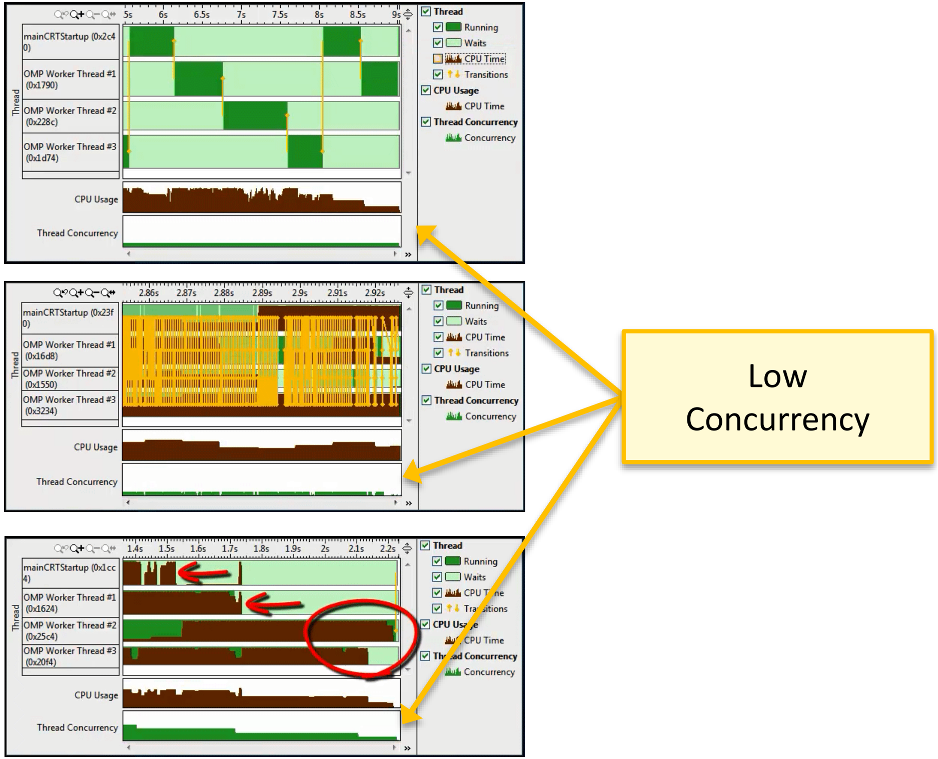
Intel® VTune™ Amplifier XE

Identify threading inefficiency

Coarse Grain Locks

High Lock Contention

Load Imbalance



Intel® VTune™ Amplifier XE

Find Answers Fast

Adjust Data Grouping

- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack

(Partial list shown)

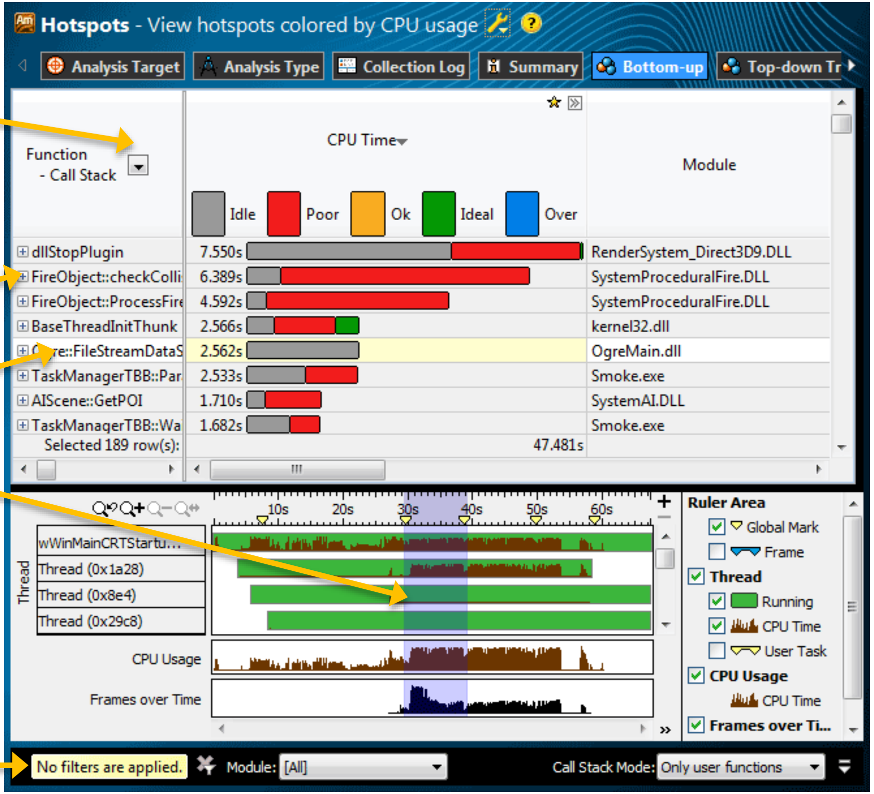
Click [+] for Call Stack

Double Click Function to View Source

Filter by Timeline Selection (or by Grid selection)

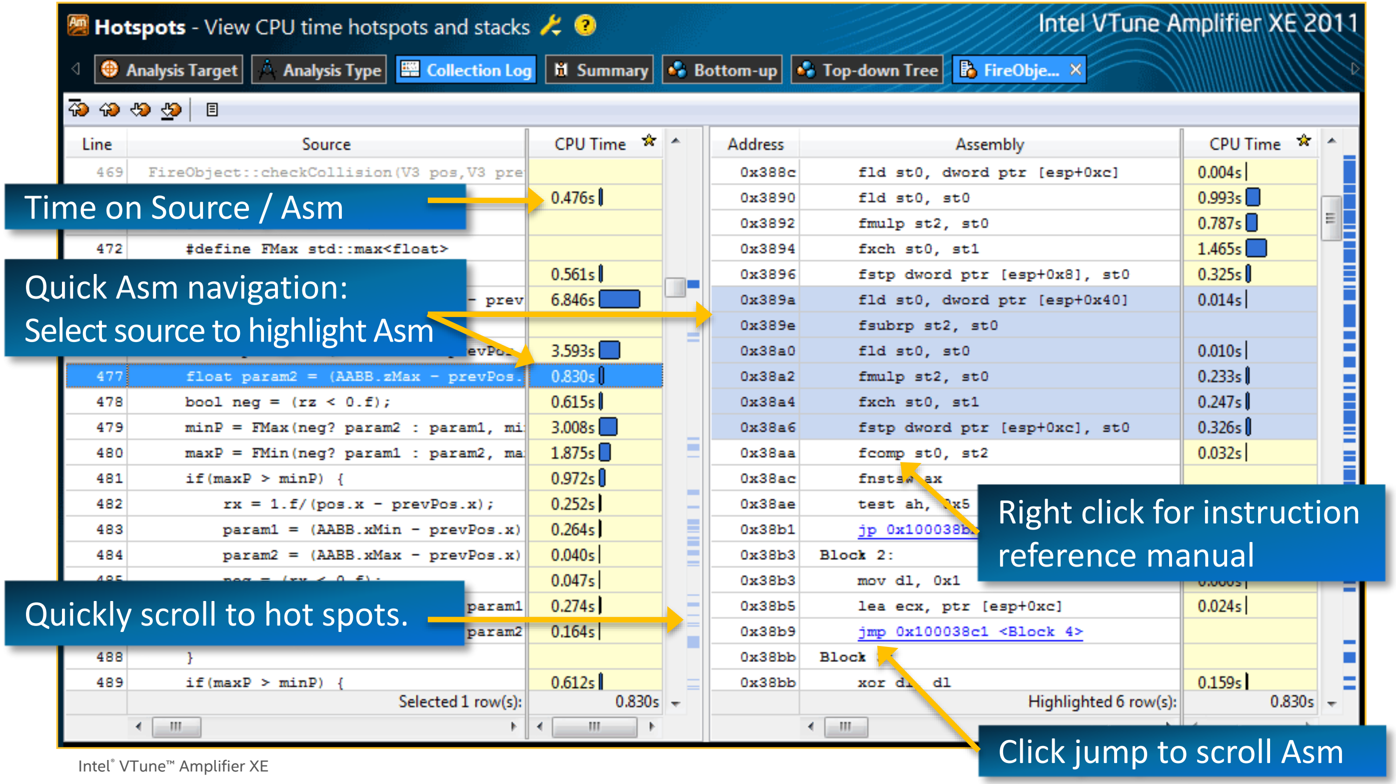
- Zoom In And Filter On Selection
- Filter In by Selection
- Remove All Filters

Filter by Module & Other Controls



Intel® VTune™ Amplifier XE

See Profile Data On Source / Asm



Intel® VTune™ Amplifier XE

Data Collectors and Analysis Types

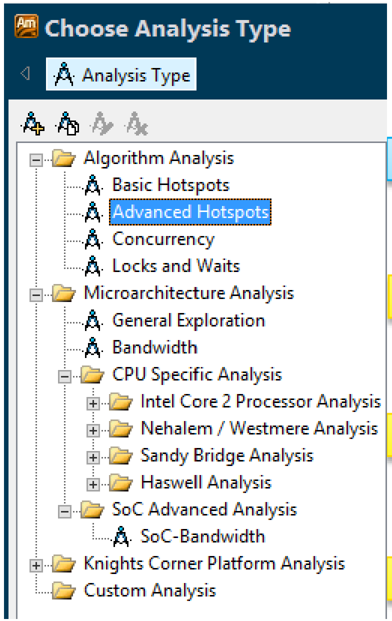
Intel® VTune™ Amplifier XE

Analysis Types (based on technology)

Software Collector Any x86 processor, any virtual, no driver	**Hardware Collector** Higher res., lower overhead, system wide
Basic Hotspots Which functions use the most time?	**Advanced Hotspots** Which functions use the most time? Where to inline? – Statistical call counts
Concurrency Tune parallelism. Colors show number of cores used.	**General Exploration** Where is the biggest opportunity? Cache misses? Branch mispredictions?
Locks and Waits Tune the #1 cause of slow threaded performance – waiting with idle cores.	**Advanced Analysis** Dig deep to tune bandwidth, cache misses, access contention, etc.

Intel® VTune™ Amplifier XE

Pre-defined Analysis Types



Advanced Hotspot analysis based on the underlying architecture

User mode sampling, Threading, IO, Signaling API instrumentation

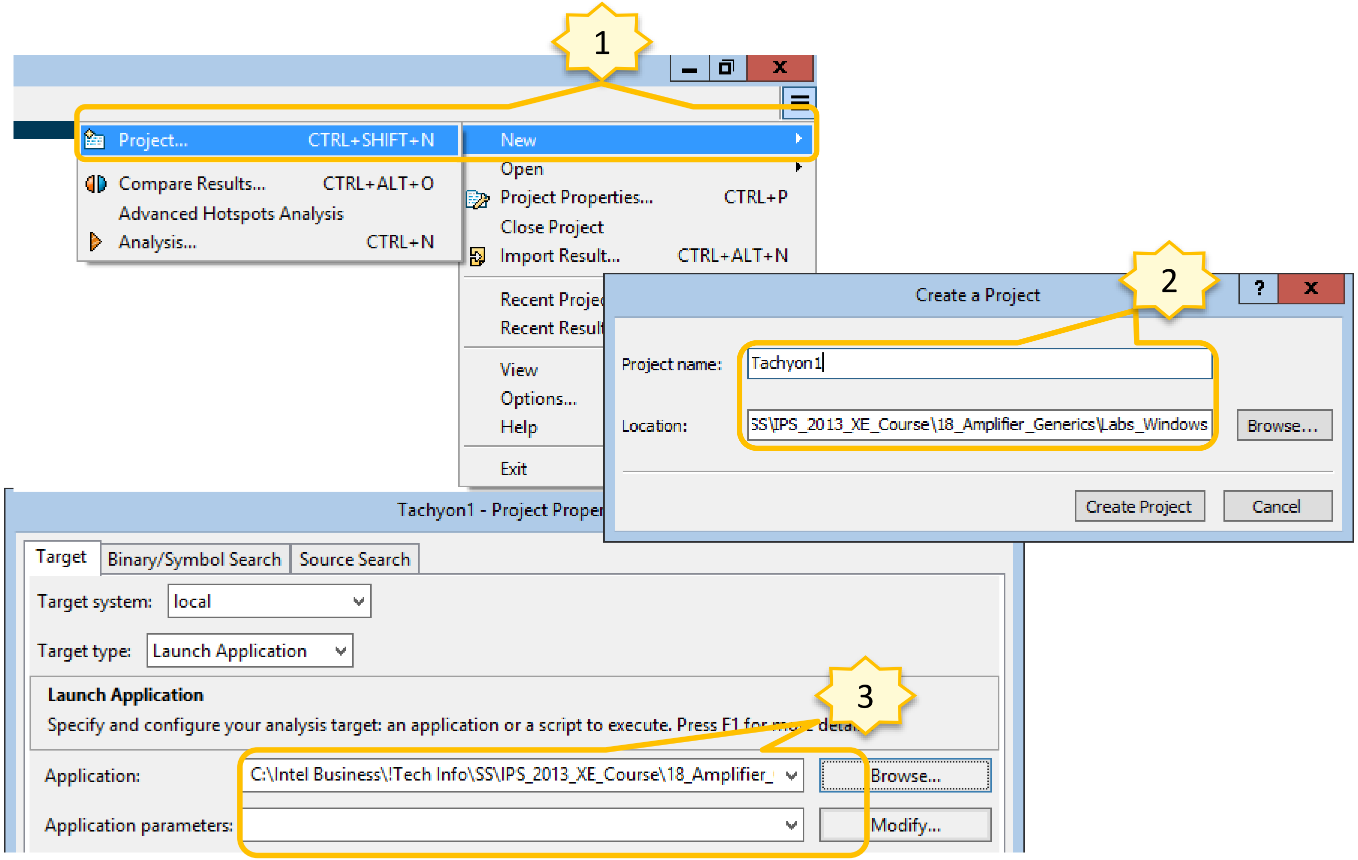
3rd Generation Core Architecture (a.k.a SandyBridge) analysis types

4th Generation Core Architecture (a.k.a Haswell) analysis types

GUI Layout

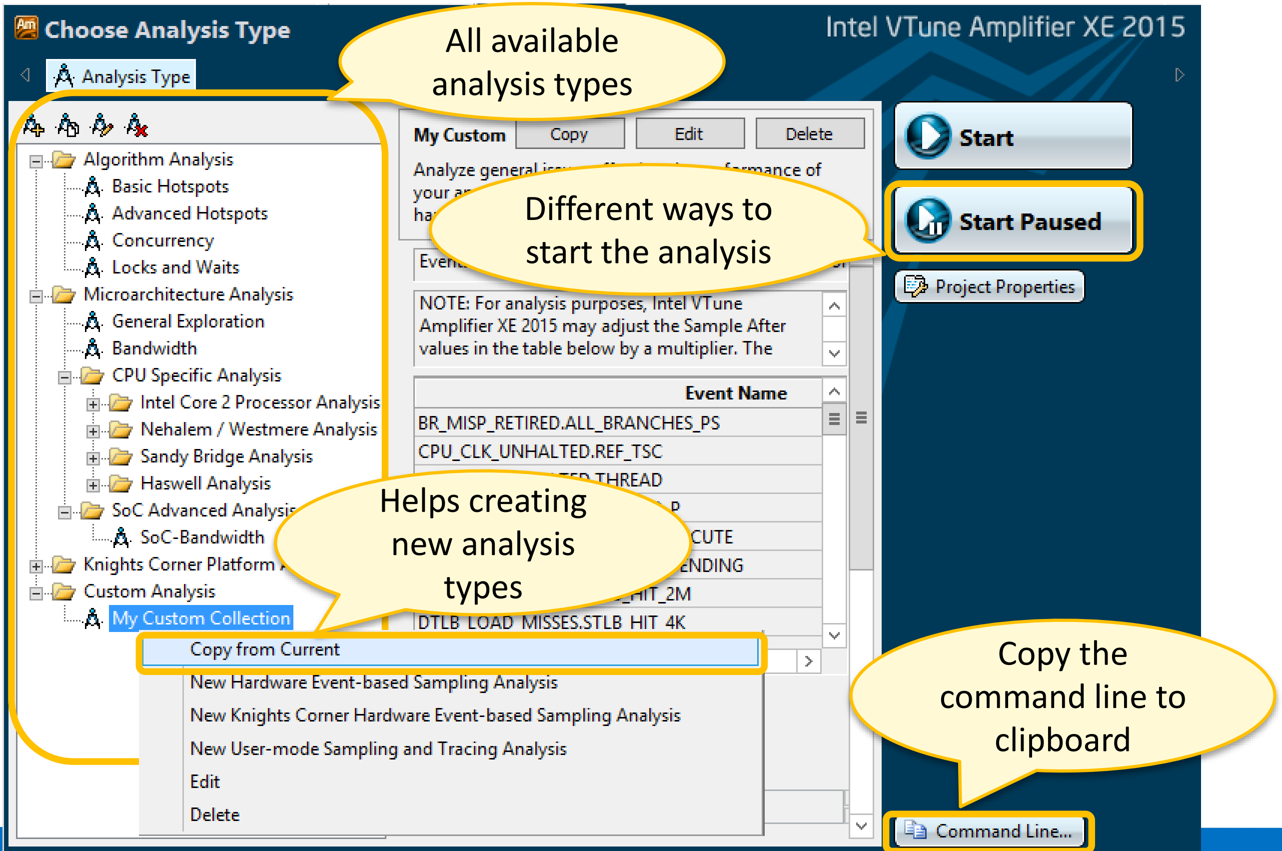
Creating a Project

GUI Layout



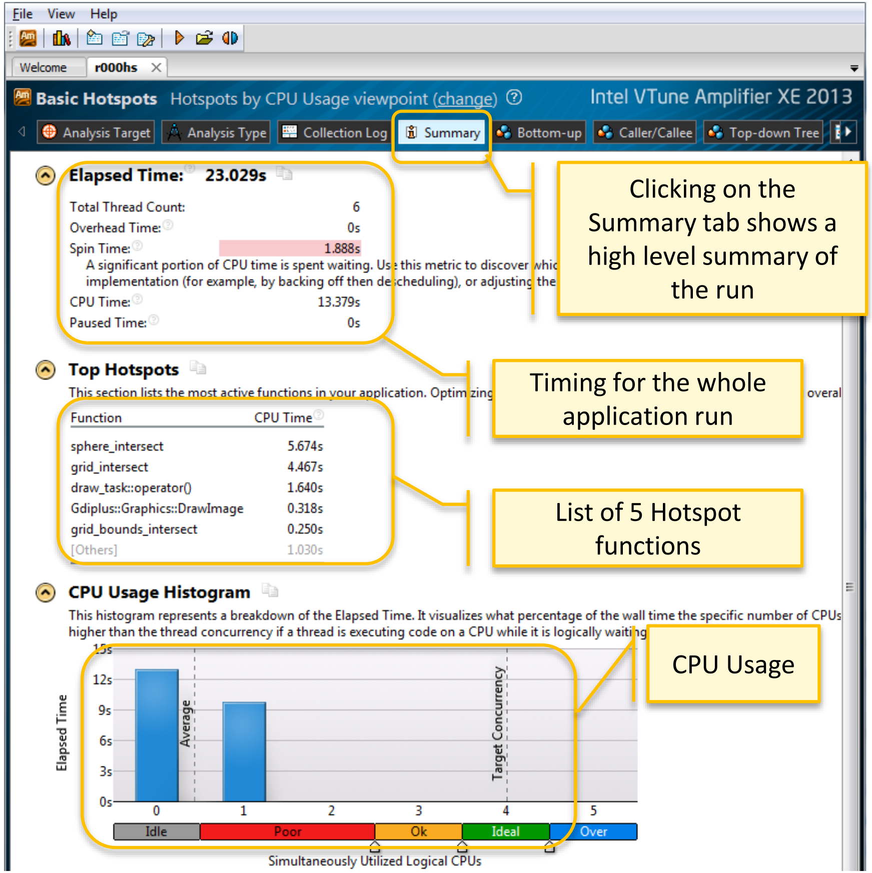
Selecting type of data collection

GUI Layout



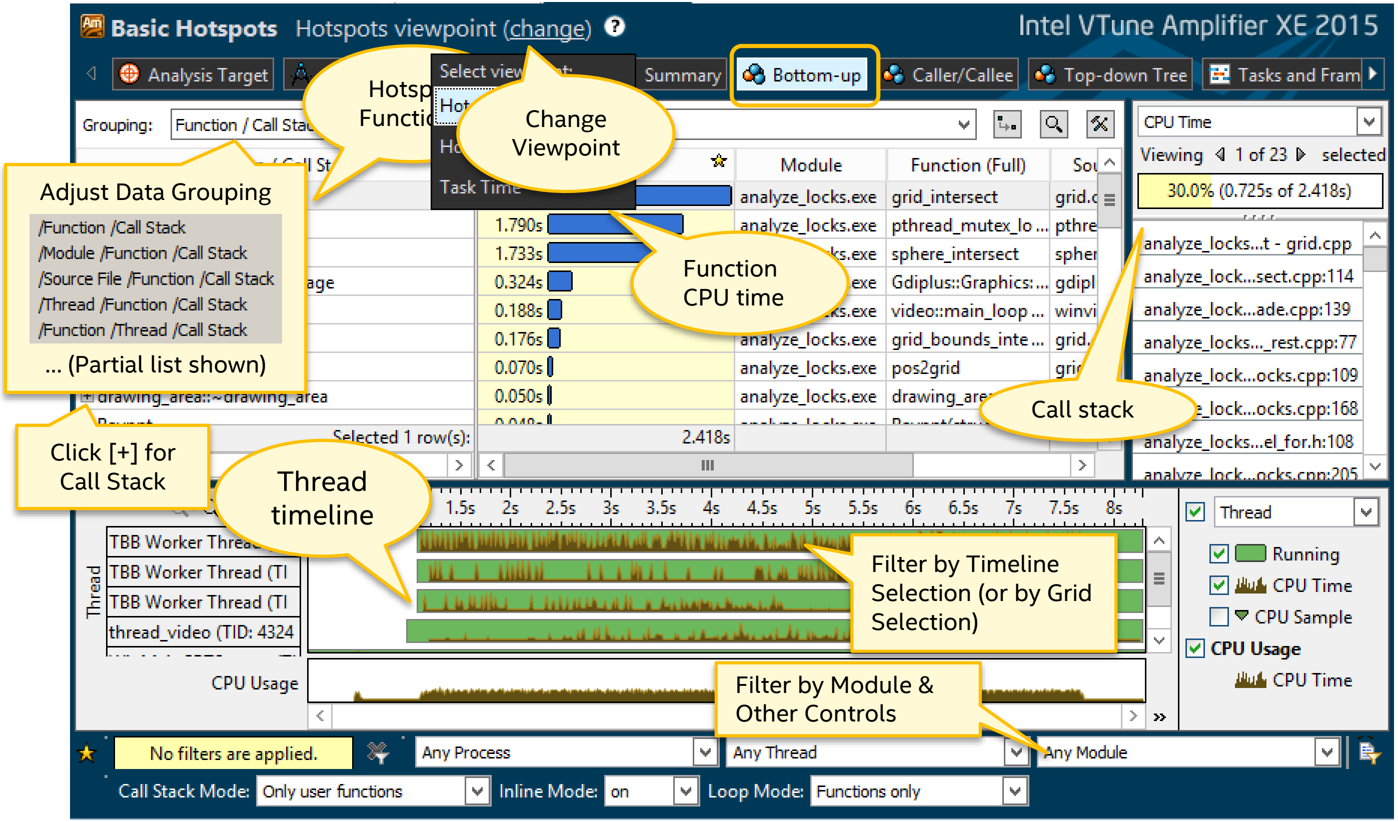
Summary View

GUI Layout



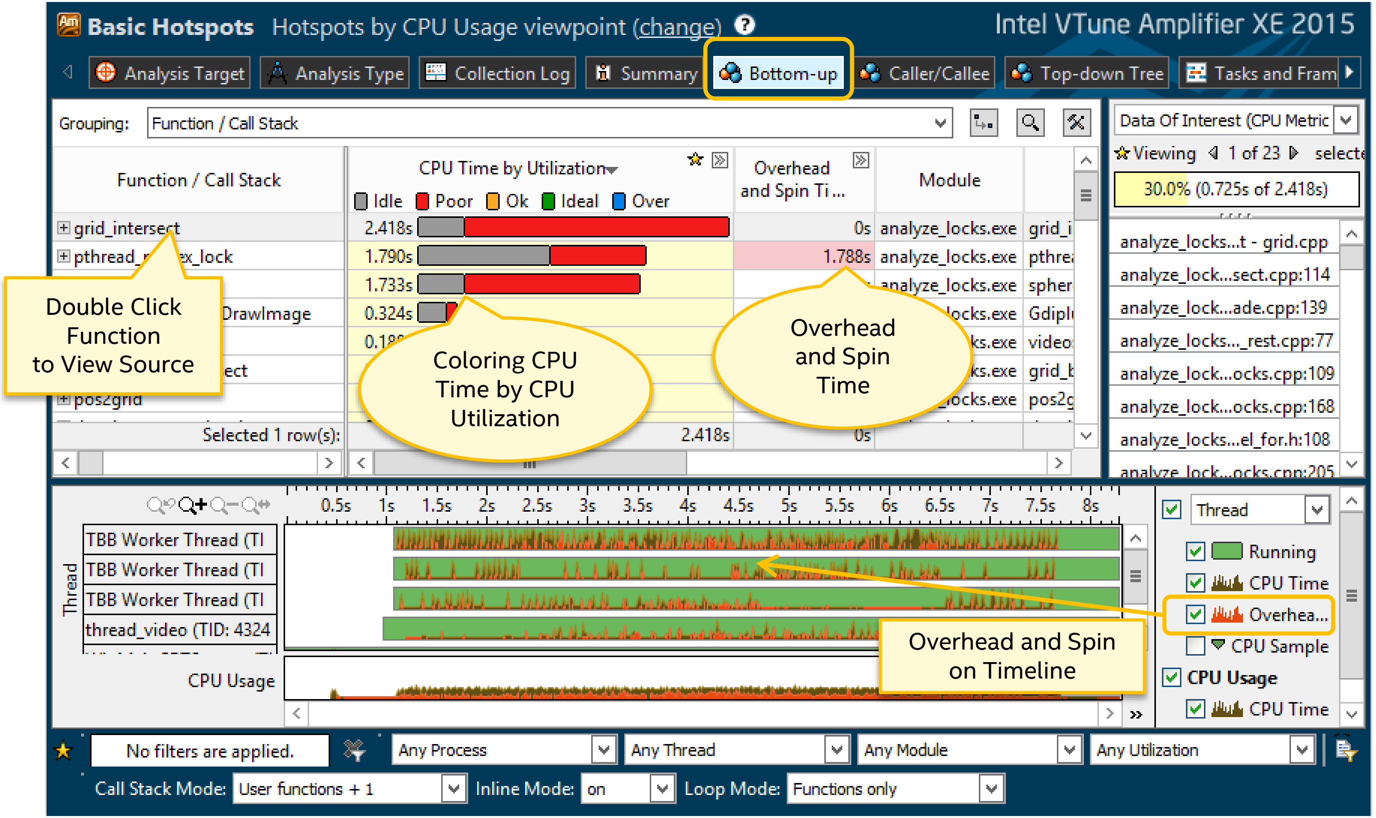
Hotspots analysis

Hotspot functions



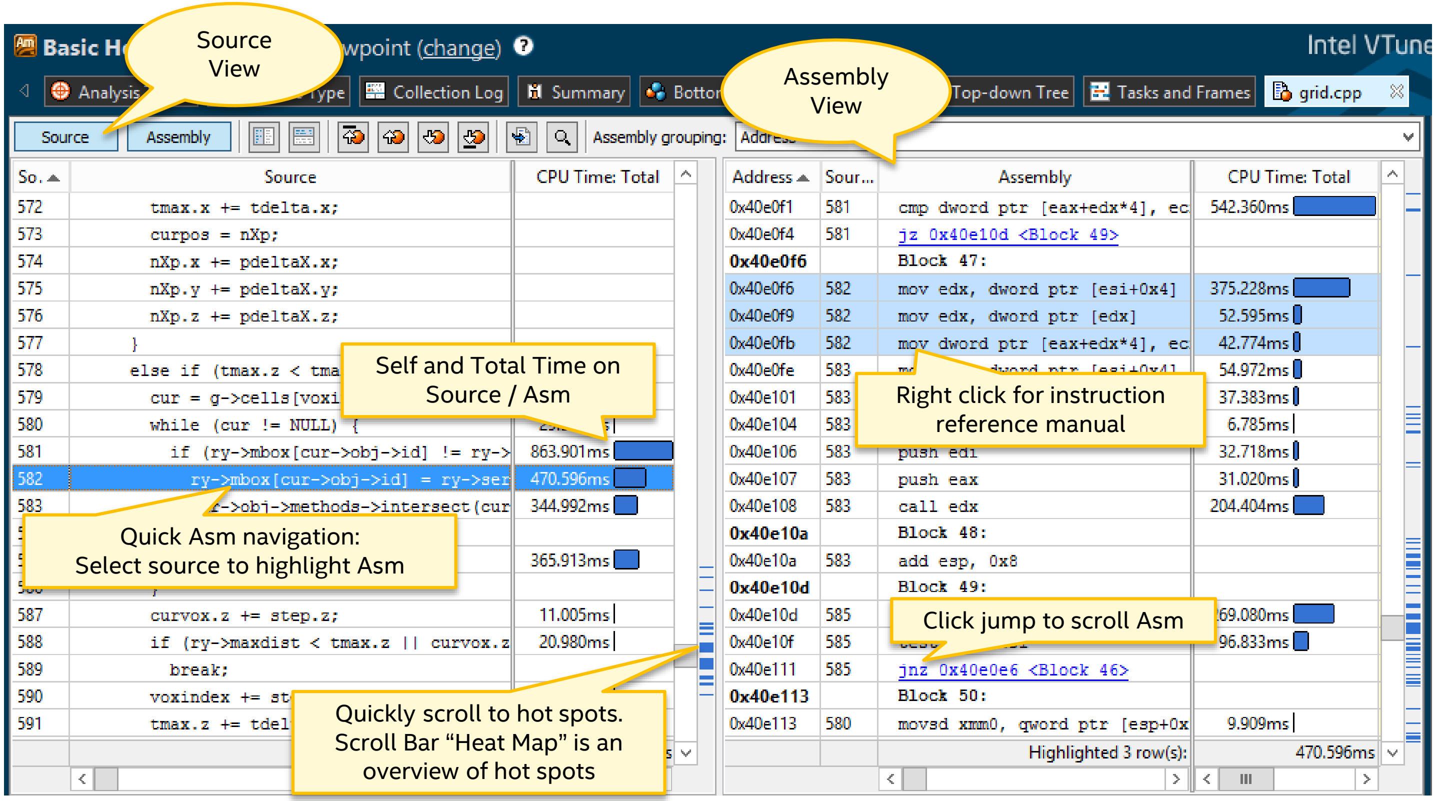
Hotspots analysis

Hotspot functions by CPU usage



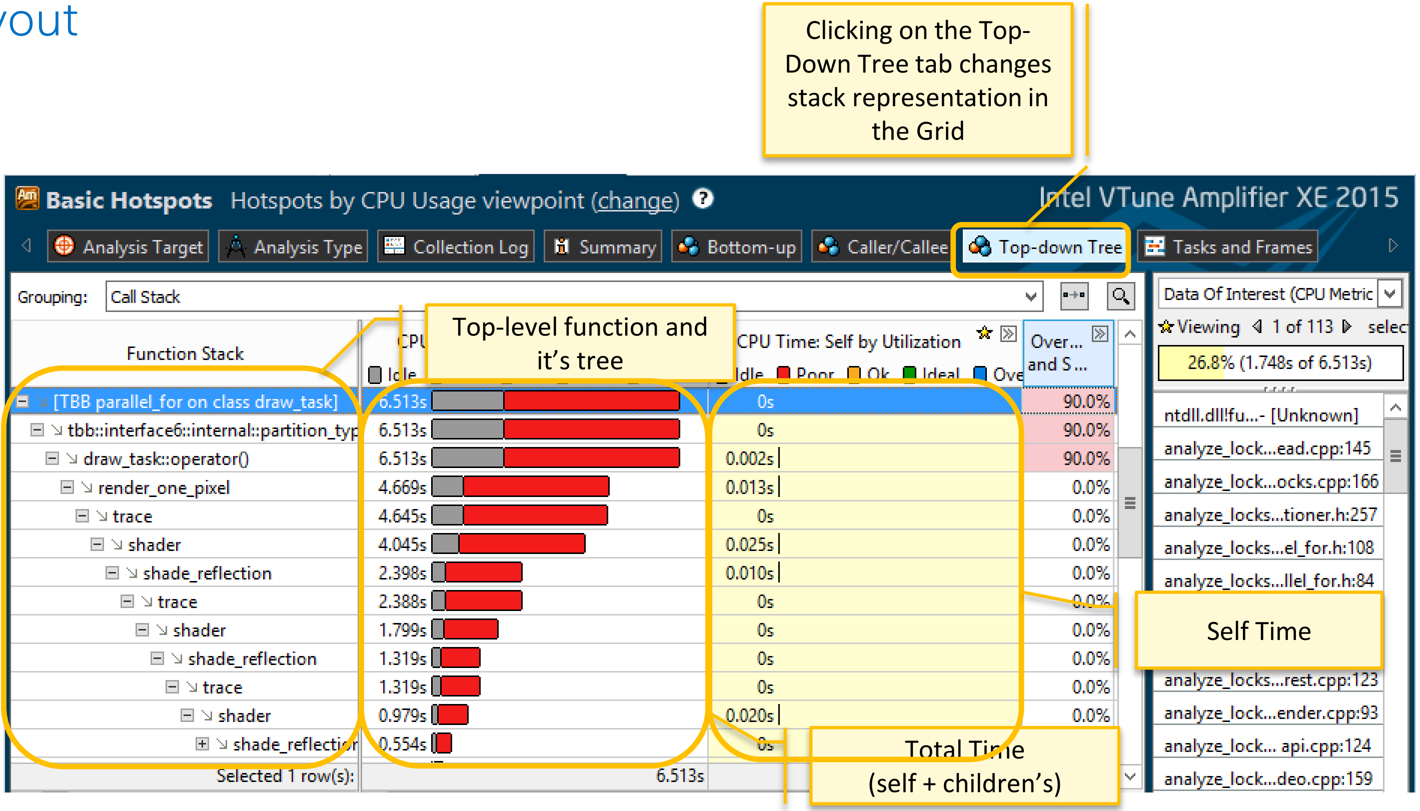
Hotspots analysis

Source View



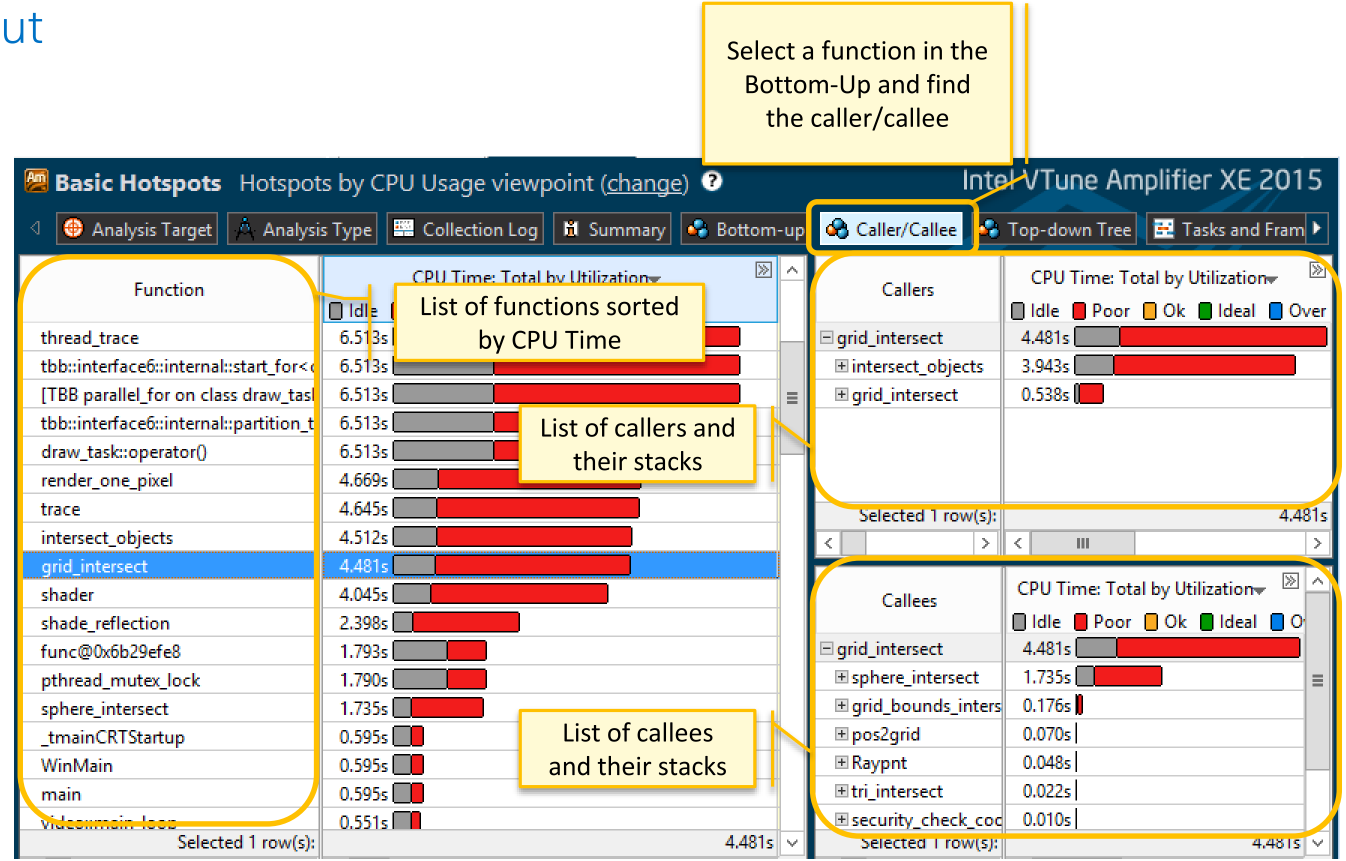
Top-Down View

GUI Layout



Caller/Callee View

GUI Layout



Command Line Interface

- Command line (CLI) versions exist on Linux* and Windows*
  - **CLI use cases:**
    - Test code changes for performance regressions
    - Automate execution of performance analyses
  - **CLI features:**
    - Fine-grained control of all analysis types and options
    - Text-based analysis reports
    - Analysis results can be opened in the graphical user interface

Command Line Interface

Examples

- Display a list of available analysis types and preset configuration levels

```
amplxe-cl –collect-list
```

- Run Hot Spot analysis on target *myApp* and store result in default-named directory, such as *r000hs*

```
amplxe-cl –c hotspots -- myApp
```

- Run the Cuncurrency analysis, store the result in directory *r001par*

```
amplxe-cl -c concurrency -result-dir r001par -- myApp
```

Command Line Interface

Reporting

```
$> amplxe-cl -report summary -r
/home/user1/examples/lab2/r003cc

Summary
-------

Average Concurrency:  9.762
Elapsed Time:         158.749
CPU Time:             561.030
Wait Time:            190.342
CPU Usage:            3.636
Executing actions 100 % done
```

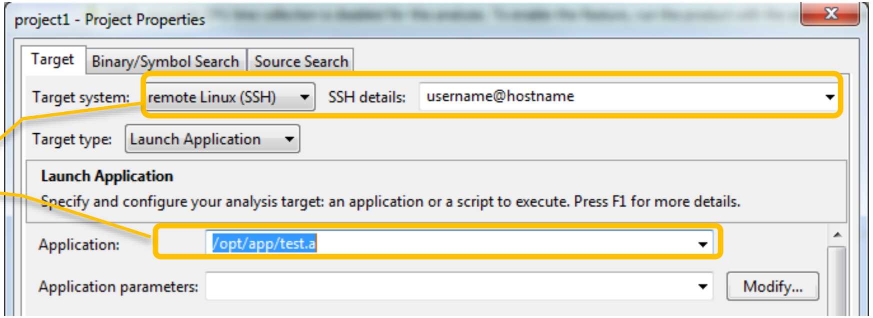
Command Line Interface

Gropof-like output

```
[levent@hlasnb AXE_lab3]$ amplxe-cl -report gprof-cc -r /home/levent/examples/cern/labs/AXE_lab3/r003cc
Using result path `/home/levent/examples/cern/labs/AXE_lab3/r003cc'
Executing actions 50 % Generating a report
Index  % Wait Time:Total  Wait Time:Self  Children  Name                                           Index
-----  -----------------  --------------  --------  ---------------------------------------------  -----
                          190.104         190.104    G4RunManager::BeamOn                          [23]
[0]    99.88              190.104         0.0       ParRunManager::DoEventLoop                     [0]

                          0.162           0.162      operator<<                                    [17]
                          0.025           0.025      G4RunManagerKernel::G4RunManagerKernel        [11]
                          0               0.001      RunAction::EndOfRunAction                     [30]
[1]    0.1                0.186           0.001     G4strstreambuf::sync                           [1]
                          0.001           0.001      G4MycoutDestination::ReceiveG4cout            [5]

                          0.033           158.141    func@0x416c28                                 [7]
[2]    83.08              0.033           158.108   main                                           [2]
                          0               158.108    G4_main                                       [18]

                          0.002           0.002      CLHEP::HepRandom::showEngineStatus            [22]
[3]    0.0                0.002           0.0       CLHEP::RanecuEngine::showStatus                [3]

                          0.001           0.001      G4_main                                       [18]
[4]    0.0                0.001           0.0       G4MycoutDestination::G4MycoutDestination       [4]

                          0.001           0.001      G4strstreambuf::sync                          [1]
[5]    0.0                0.001           0.0       G4MycoutDestination::ReceiveG4cout             [5]

                          0               0          G4UImanager::ExecuteMacroFile <cycle 1>       [28]
[6]    0.0                0.0             0.0       G4UIbatch::G4UIbatch                           [6]

[7]    83.08              0.0             158.141   func@0x416c28                                  [7]
                          0.033           158.141    main                                          [2]

                          0               190.107    G4_main                                       [18]
[8]    99.88              0.0             190.107   <cycle 1 as a whole>                           [8]
```

Remote Data Collection

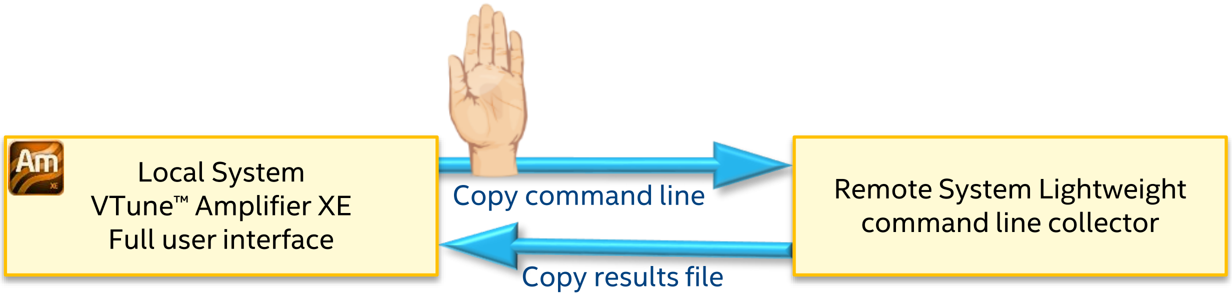
1. Setup the experiment using GUI locally
2. Configure remote target connection*
3. Specify application to run
4. Run analysis and get results copied to the Host automatically.



*Need to establish a passwordless ssh-connection

Remote Data Collection

Advanced



1. Setup the experiment using GUI locally
2. Copy command line instructions to paste buffer
3. Open remote shell on the target system
4. Paste command line, run collection
5. Copy result to your system
6. Open file using local GUI

One typical model

- Collect on Linux, analyze and display on Windows
  - The Linux machine is target
- Collect data on Linux system using command line tool
  - Doesn’t require a license
- Copy the resulting performance data files to a Windows* system
- Analyze and display results on the Windows* system
  - Requires a license

Using Intel® VTune™ Amplifier XE with MPI

- Usage depends on collection:
  - For SW collection, MPI launches VTune™ Amplifier, which launches the app

```
$mpirun <MPI args> amplxe-cl <VTune args> -- <application and args>
```

  - For hardware collection, on host, only one collection per node may run
    - Recommendation: run one system-wide collection per node; other ranks run no collections at all

```
$mpirun -n <nodes> -ppn 1 amplxe-cl –analyze-system <other opts> -- \
  <application and args> : -n <remaining ranks> <application and args>
```

    - Alternatively, can start system-wide collections outside of MPI
    - Or only run one rank per node
- When VTune Amplifier is run under MPI, each results folder has rank number appended

Intel® Xeon Phi™ Coprocessor + MPI + Intel® VTune™ Amplifier XE

- VTune™ Amplifier cannot be launched from the coprocessor
- As such, MPI cannot launch VTune Amplifier directly for coprocessor collections
- Best Known Method:
  - Run VTune Amplifier from host with coprocessor system-wide collection
    - This can even be done using a separate MPI job
  - Run real job (Offload, Native, or Symmetric)

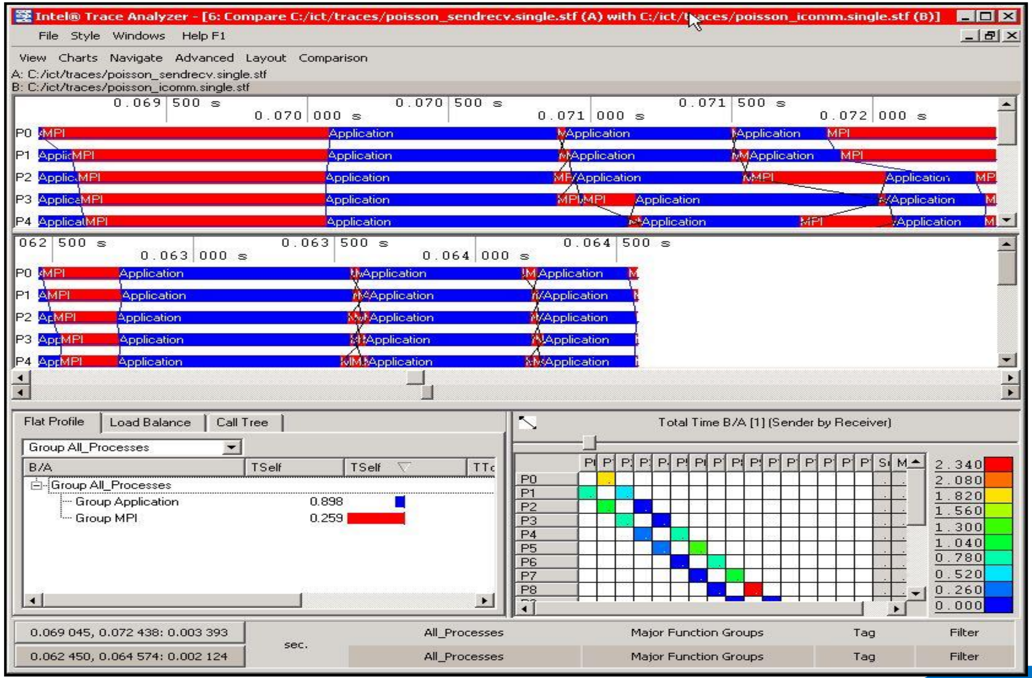
Summary for Intel® VTune Amplifier XE

- The Intel® VTune Amplifier XE can be used to find:
  - Source code for performance bottlenecks
  - Characterize the amount of parallelism in an application
  - Determine which synchronization locks or APIs are limiting the parallelism in an application
  - Understand problems limiting CPU instruction level parallelism
  - Instrument user code for better understanding of execution flow defined by threading runtimes

Intel® Trace Analyzer and Collector

Intel® Trace Analyzer and Collector

- Helps the developer:
  - Visualize and understand parallel application behavior
  - Evaluate profiling statistics and load balancing
  - Identify communication hotspots
- Features
  - Event based approach
  - Low overhead, excellent scalability
  - Comparison of multiple profiles
  - Powerful aggregation and filtering functions
  - Fail-safe MPI tracing
  - API to instrument user code
  - MPI Correctness Checker
  - Idealizer and Application Imbalance Diagram
  - New Performance Assistant



Why Tracing?

- 1:1 record of actual program execution
  - No MPI calls are missed
- Accurate timing correlated between ranks
  - Enables seeing when calls happened relative to calls in other ranks
  - Valuable for finding MPI performance bottlenecks
- Data recorded
  - Function entry/exit times
  - MPI parameters
  - Communication vs. waiting time

Multiple Methods for Data Collection

Collection Mechanism	Advantages	Disadvantages
Run with –trace or preload trace collector library.	**Automatically collects all MPI calls, requires no modification to source, compile, or link**	**No user code collection.**
Link with –trace.	Automatically collects all MPI calls	No user code collection. Must be done at link time.
Compile with –tcollect.	Automatically instruments all function entries/exits	Requires recompile of code. Significant overhead.
Add API calls to source code.	Can selectively instrument desired code sections	Requires code modification.

Data Location

- Stored in a set of stf (structured tracefile) files.
- For large runs, data can quickly grow unmanageable
  - Really depends on number of instrumented calls
  - Filters available to reduce collected data
- Files are stored by default in launching folder
  - This can be dangerous for native runs launched from the coprocessor

Example

- Here, we'll go ahead and collect a trace.
- Source the environment variable scripts (if needed) with

```
$. /opt/intel/impi/<version>/intel64/bin/mpivars.sh
$. /opt/intel/itac/<version>/intel64/bin/itacvars.sh
```

- Set up the host file and then run:

```
$cat hosts.txt
node000-mic0
node001-mic0
$mpirun -trace -n 2 -f hosts.txt -ppn 1 ../../../bin/ZSC-3D-STD-MPI
```

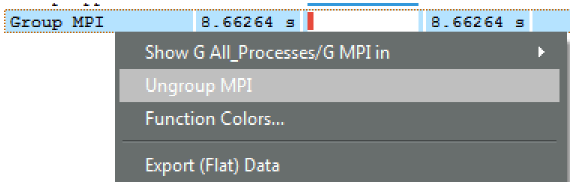
- Once this completes, you will have a set of trace files
- If you want to analyze them on a different computer, copy/move them to that computer now
  - Don't forget to move source files as well

Opening the Trace File

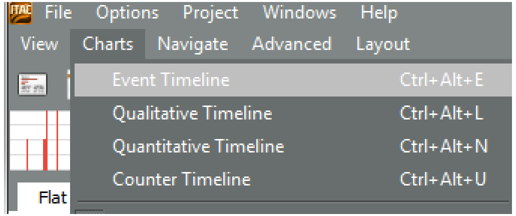
- Start Intel® Trace Analyzer
  - Linux* - run *traceanalyzer* in the console
  - Windows* - Load Intel® Trace Analyzer from the Start Menu
- Load the main trace file
  - Others are loaded as needed by the GUI
- Default view:
  - Zoomed out completely
  - Function Profile (and Performance Assistant in 9.0) visible
  - Group Application and Group MPI

Examining the Trace

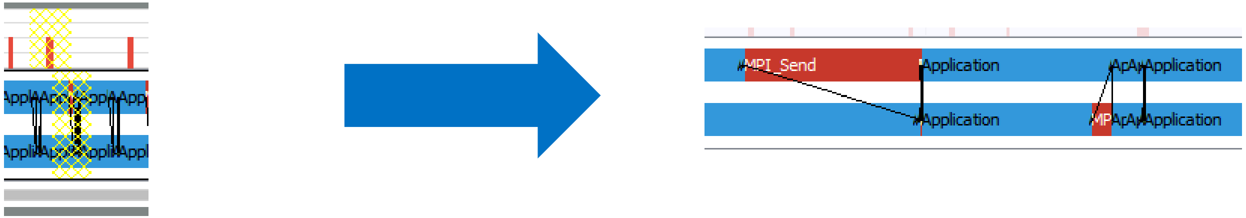
- Ungroup MPI



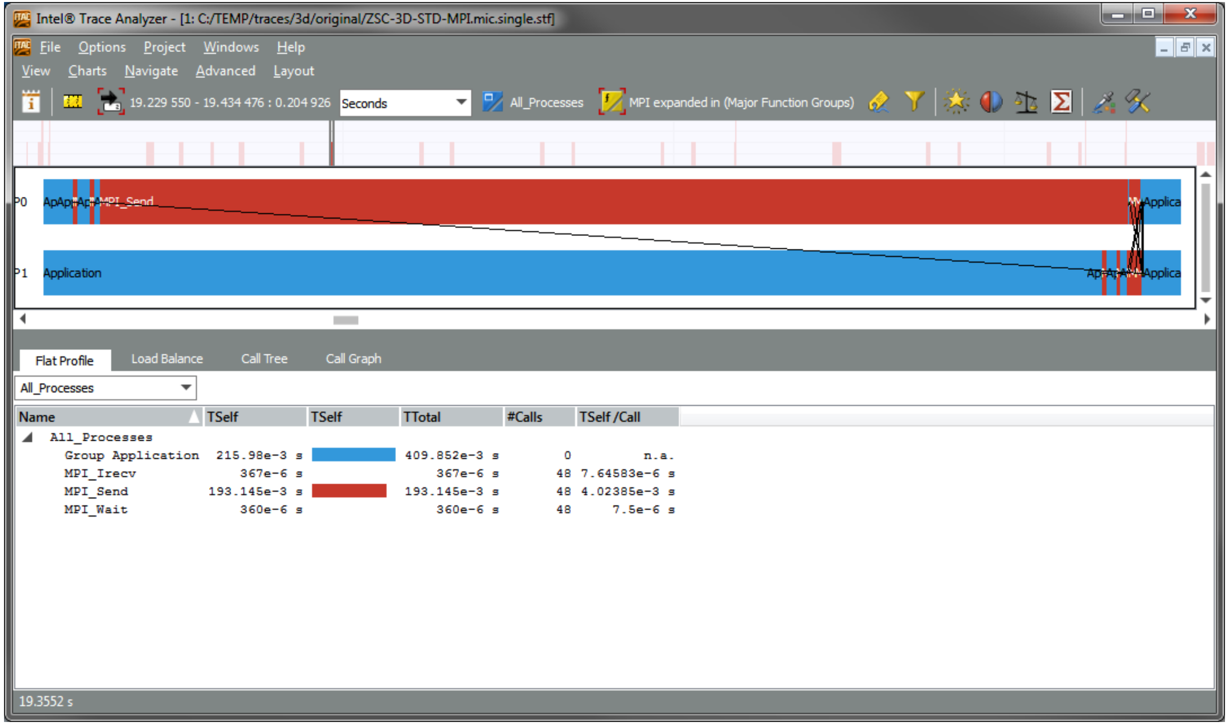
- Show Event Timeline



- Zoom to a single iteration by clicking/dragging on the Event Timeline



The Event Timeline



Getting Details on a Message

Details on Message

Show

Chart

Event Timeline Settings...

Close Chart Ctrl+Shift+K

Message

Sender	Receiver	Duration [s]	Send Time [s]	Receive Time [s]	Volume [B]	Rate [B/s]	Count	Tag	Communicator Name	Communicator ID	Sending Function	Receiving Function
P0	P1	0.213 078	17.366 236	17.579 314	45 000	211 190	1	1	CART_CREATE COMM_WORLD	1	MPI_Send	MPI_Wait

Sender	Receiver	Duration [s]	Send Time [s]	Receive Time [s]	Volume [B]	Rate [B/s]	Count	Tag	Communicator Name	Communicator ID	Sending Function	Receiving Function
P0	P1	0.001 340	17.577 998	17.579 338	45 000	33 579 007	1	3	CART_CREATE COMM_WORLD	1	MPI_Send	MPI_Wait

Code Investigation

```
! Exchange Data in each direction and each edge (only X direction shown here)
! Call MPI_Irecv for current direction
 CALL MPI_IRECV(f_east_rcv, xsize , MPI_DOUBLE_PRECISION, east, TAG1, …
 CALL MPI_IRECV(g_east_rcv, xsize5, MPI_DOUBLE_PRECISION, east, TAG2, …
 CALL MPI_IRECV(f_west_rcv, xsize , MPI_DOUBLE_PRECISION, west, TAG3, …
 CALL MPI_IRECV(g_west_rcv, xsize5, MPI_DOUBLE_PRECISION, west, TAG4, …
! Calculate values to send in nested loop
! Call MPI_Send for current direction
 CALL MPI_SEND(f_west_snd, xsize , MPI_DOUBLE_PRECISION, west, TAG1, …
 CALL MPI_SEND(g_west_snd, xsize5, MPI_DOUBLE_PRECISION, west, TAG2, …
 CALL MPI_SEND(f_east_snd, xsize , MPI_DOUBLE_PRECISION, east, TAG3, …
 CALL MPI_SEND(g_east_snd, xsize5, MPI_DOUBLE_PRECISION, east, TAG4, …
! Wait for all transfers to complete
 CALL MPI_WAIT(MPI_REQ(1), status, MPI_ERR)
 CALL MPI_WAIT(MPI_REQ(2), status, MPI_ERR)
 CALL MPI_WAIT(MPI_REQ(3), status, MPI_ERR)
 CALL MPI_WAIT(MPI_REQ(4), status, MPI_ERR)
```

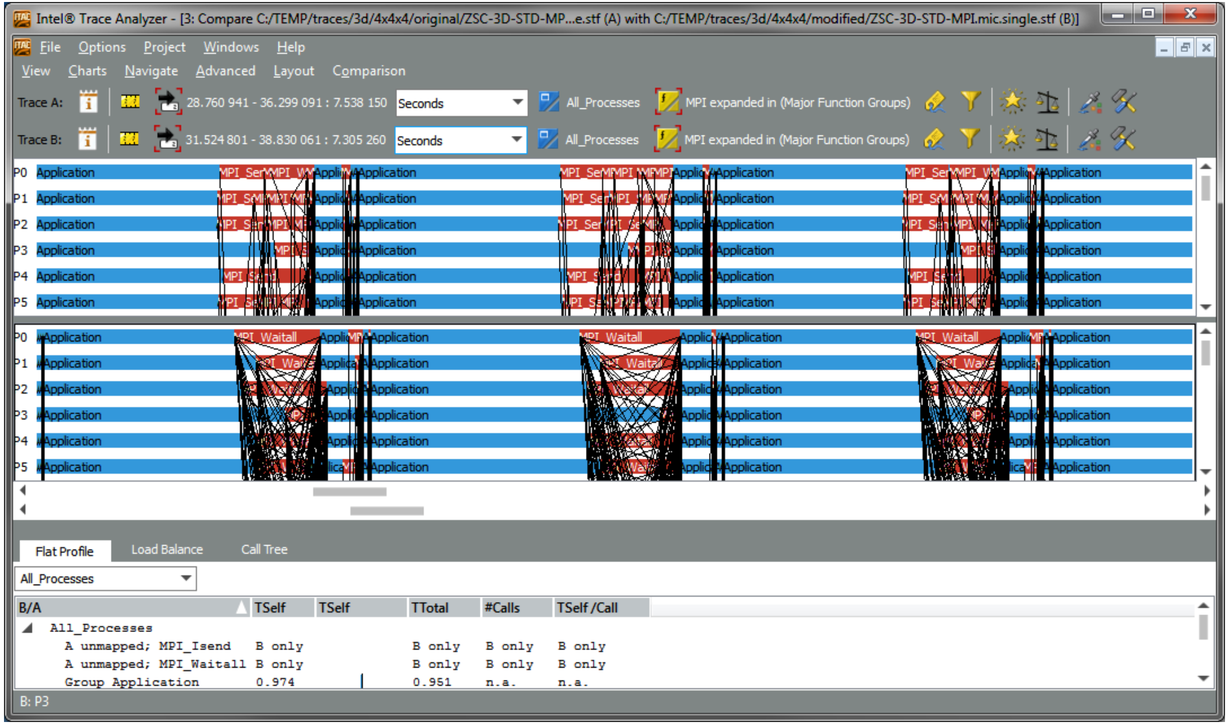
Possible Improvement

```
! Exchange data in all directions and edges
! Call MPI_Irecv for all directions and edges intiailly
 CALL MPI_IRECV(f_east_rcv, xsize , MPI_DOUBLE_PRECISION, east, TAG(1), …
 CALL MPI_IRECV(g_east_rcv, xsize5, MPI_DOUBLE_PRECISION, east, TAG(2), …
 CALL MPI_IRECV(f_west_rcv, xsize , MPI_DOUBLE_PRECISION, west, TAG(3), …
 CALL MPI_IRECV(g_west_rcv, xsize5, MPI_DOUBLE_PRECISION, west, TAG(4), …
 …
! Calculate values to send in nested loop
! Call MPI_Isend for current direction
 CALL MPI_ISEND(f_west_snd, xsize , MPI_DOUBLE_PRECISION, west, TAG(1), …
 CALL MPI_ISEND(g_west_snd, xsize5, MPI_DOUBLE_PRECISION, west, TAG(2), …
 CALL MPI_ISEND(f_east_snd, xsize , MPI_DOUBLE_PRECISION, east, TAG(3), …
 CALL MPI_ISEND(g_east_snd, xsize5, MPI_DOUBLE_PRECISION, east, TAG(4), …

! Once all directions have been calculated and all MPI_Isend calls are made
! Call MPI_Waitall to wait for all to complete
 CALL MPI_WAITALL(48, MPI_REQ, statuses, MPI_ERR)
```

Collecting the New Trace

- Ensure that the new trace doesn't overwrite the old trace
  - Rename/move/copy the previous trace files
  - Use a different named executable (src/devel/mpi_combined does this)
  - Change the base name for the trace with VT_LOGFILE_NAME
    - $export VT_LOGFILE_NAME=modified.stf
- Compile new code
- Run again with -trace

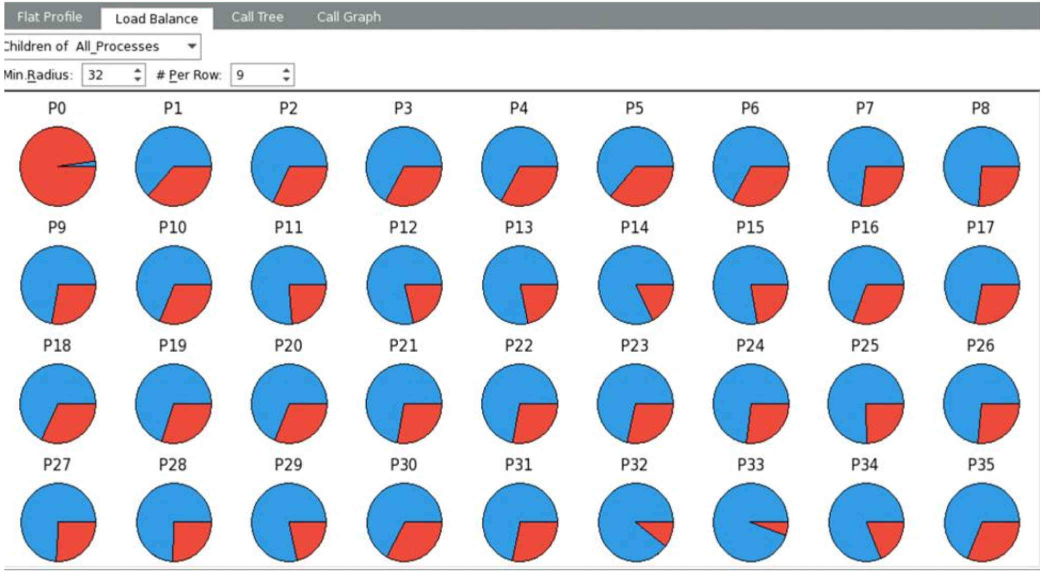
Comparing the Traces



Case: CAMx load balancing

- Profiled and summarized with ITAC
- Fixed the data distributing parameter

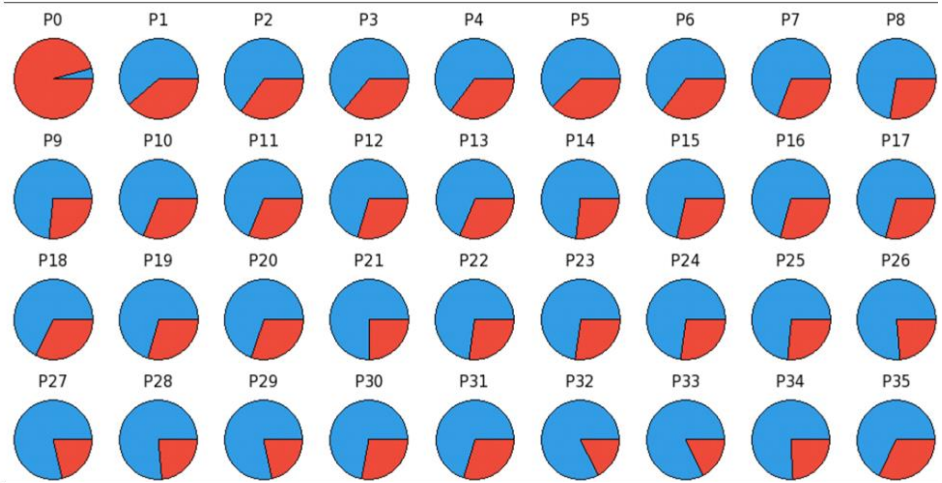
!	grid#	node#	x-beg	x-end	y-beg	y-end	#cells
!	2	1	2	19	2	16	270
!	2	2	20	37	2	16	270
!	2	3	38	55	2	16	270
!	2	4	56	73	2	16	270
!	2	5	74	91	2	16	270
!	2	6	2	19	17	31	270
!	2	7	20	37	17	31	270
!	2	8	38	55	17	31	270
!	2	9	56	73	17	31	270
!	2	10	74	91	17	31	270
!	2	11	2	19	32	49	324
!	2	12	20	37	32	49	324
!	2	13	38	55	32	49	324
!	2	14	56	73	32	49	324
!	2	15	74	91	32	49	324
!	2	16	2	19	50	64	270
!	2	17	20	37	50	64	270
!	2	18	38	55	50	64	270
!	2	19	56	73	50	64	270
!	2	20	74	91	50	64	270
!	2	21	2	19	65	79	270
!	2	22	20	37	65	79	270
!	2	23	38	55	65	79	270
!	2	24	56	73	65	79	270
!	2	25	74	91	65	79	270
!	2	26	2	19	80	94	270
!	2	27	20	37	80	94	270
!	2	28	38	55	80	94	270
!	2	29	56	73	80	94	270
!	2	30	74	91	80	94	270
!	2	31	2	19	95	112	324
!	2	32	20	37	95	112	324
!	2	33	38	55	95	112	324
!	2	34	56	73	95	112	324
!	2	35	74	91	95	112	324



Case: CAMx load balancing

- Profiled and summarized with ITAC
- Fixed the data distributing parameter

```
!   grid# node# x-beg x-end y-beg y-end #cells
!-----------------------------------------------
!     2     1     2    19     2    16   270
!     2     2    20    37     2    16   270
!     2     3    38    55     2    16   270
!     2     4    56    73     2    16   270
!     2     5    74    91     2    16   270
!     2     6     2    19    17    32   288
!     2     7    20    37    17    32   288
!     2     8    38    55    17    32   288
!     2     9    56    73    17    32   288
!     2    10    74    91    17    32   288
!     2    11     2    19    33    48   288
!     2    12    20    37    33    48   288
!     2    13    38    55    33    48   288
!     2    14    56    73    33    48   288
!     2    15    74    91    33    48   288
!     2    16     2    19    49    64   288
!     2    17    20    37    49    64   288
!     2    18    38    55    49    64   288
!     2    19    56    73    49    64   288
!     2    20    74    91    49    64   288
!     2    21     2    19    65    80   288
!     2    22    20    37    65    80   288
!     2    23    38    55    65    80   288
!     2    24    56    73    65    80   288
!     2    25    74    91    65    80   288
!     2    26     2    19    81    96   288
!     2    27    20    37    81    96   288
!     2    28    38    55    81    96   288
!     2    29    56    73    81    96   288
!     2    30    74    91    81    96   288
!     2    31     2    19    97   112   288
!     2    32    20    37    97   112   288
!     2    33    38    55    97   112   288
!     2    34    56    73    97   112   288
!     2    35    74    91    97   112   288
!-----------------------------------------------
```



Summary for Intel® Trace Analyzer and Collector

- Intel® Trace Analyzer and Collector allows accurate measurements of MPI communication patterns
- Accurate measurements can be used to analyze and improve performance
- MPI performance is not always evident at small scale

intel®

Legal Disclaimers

Legal Disclaimers

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 SSE3 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

Legal Disclaimers

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

Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more athttp://www.intel.com/go/turbo.

Estimated Results Benchmark Disclaimer:
Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

Software Source Code Disclaimer:
Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Legal Disclaimers

The above statements and any others in this document that refer to plans and expectations for the third quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions; customer acceptance of Intel’s and competitors’ products; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Uncertainty in global economic and financial conditions poses a risk that consumers and businesses may defer purchases in response to negative financial events, which could negatively affect product demand and other related matters. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the company’s most recent reports on Form 10-Q, Form 10-K and earnings release.

Rev. 7/17/13