Optimizing for MPI*/OpenMP* on Intel® Xeon Phi™ Coprocessors

CUG 2014, Lugano, Switzerland

Jim Jeffers, John Pennycook, Hans Pabst, Heinrich Bockhorst, Intel Corporation

Vince Betro, Paul Peltz, National Institute for Computational Sciences, UTenn

Agenda

Aims

- Focus on techniques, not syntax.
- Maximise hands-on programming experience.
- Demonstrate performance gains for real-life applications.

FAQ Session

- E-mail your questions to: john.pennycook@intel.com
- Most frequent/interesting answered at the end of the day

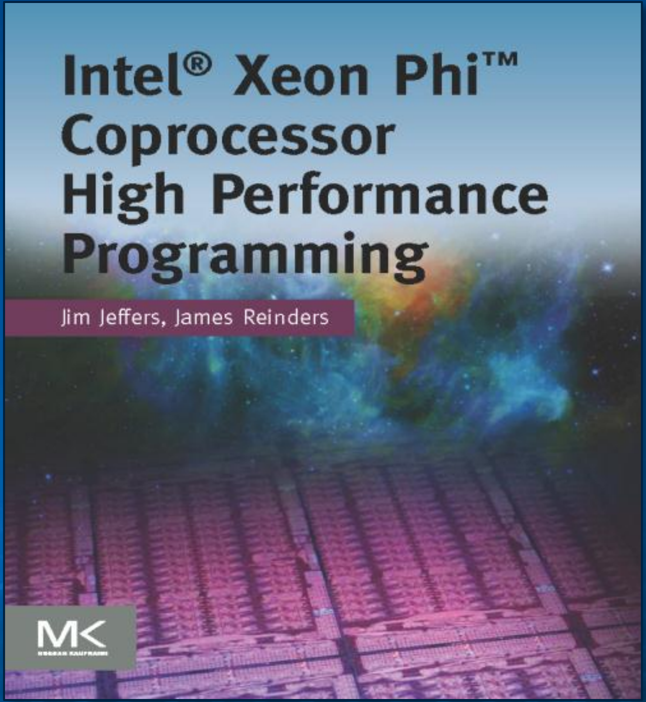
Agenda

08:30	Introduction to Intel® Xeon Phi™ Coprocessors	John Pennycook, Jim Jeffers
09:30	Cluster Administration	Paul Peltz
09:45	**Hands-on:** Logging in and Running Applications	
10:00	Break	
10:30	MPI* and OpenMP*	Vince Betro
11:00	**Hands-on:** MPI*, OpenMP* and Intel® Trace Analyzer and Collector (ITAC)	
12:00	Lunch	
13:00	Compiler-Assisted Offload	Hans Pabst
13:30	**Hands-on:** Device Extensions in OpenMP* 4.0	
14:30	Break	
14:45	Vectorization	John Pennycook
15:15	**Hands-on:** Explicit Vectorization with OpenMP* 4.0	
16:15	Frequently Asked Questions	All presenters
16:30	End	

It all comes down to PARALLEL PROGRAMMING ! (applicable to processors and Intel® Xeon Phi™ coprocessors both)

Learn more about this book:

lotsofcores.com


Intel® Xeon Phi™ Coprocessor High Performance Programming
Jim Jeffers, James Reinders
MK
MORGAN KAUFMANN

Available since mid-February 2013.

This book belongs on the bookshelf of every HPC professional. Not only does it successfully and accessibly teach us how to use and obtain high performance on the Intel MIC architecture, it is about much more than that. It takes us back to the universal fundamentals of high-performance computing including how to think and reason about the performance of algorithms mapped to modern architectures, and it puts into your hands powerful tools that will be useful for years to come.

—Robert J. Harrison
Institute for Advanced Computational Science, Stony Brook University

Intel® Xeon Phi™ Coprocessor High Performance Programming,
Jim Jeffers, James Reinders, (c) 2013, publisher: Morgan Kaufmann

Introduction to Intel® Xeon Phi™ Coprocessors

Parallelism, Parallelism and More Parallelism

Parallelism in Modern Computer Architecture

Instruction-Level Parallelism (ILP)

- Pipelining
- Multiple instruction issue
- Out-of-Order execution

Vectorization (SIMD)

- Single instructions can be applied to more than one piece of data

Threading (MIMD)

- Multiple instances of a program can run simultaneously

Parallelism in Modern Computer Architecture

Instruction-Level Parallelism (ILP)

- Pipelining
- Multiple instruction issue
- Out-of-Order execution

Vectorization (SIMD)

- Single instructions can be applied to more than one piece of data

Threading (MIMD)

- Multiple instances of a program can run simultaneously

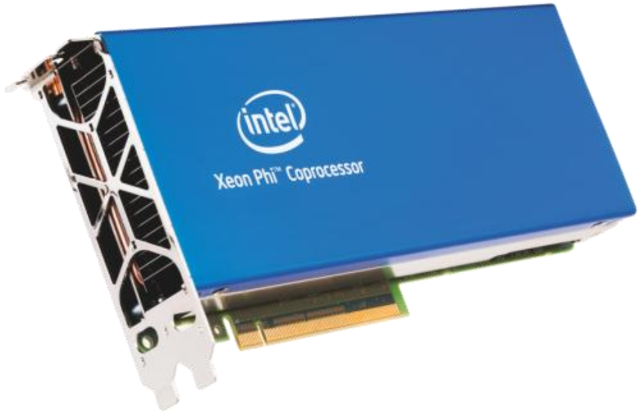
Heterogeneity in Architecture Designs

Multi-core

- High single-thread performance.
- Low number of threads.
- Suitable for any workload.

Many-core

- Low single-thread performance.
- High number of threads.
- Suitable for highly parallel workloads.



Knights Corner – the first Intel® Xeon Phi™ Coprocessor

Heterogeneity in Architecture Designs

Multi-core

- High single-thread performance.
- Low number of threads.
- Suitable for any workload.

Many-core

- Low single-thread performance.
- High number of threads.
- Suitable for highly parallel workloads.

	Multi-core	Many-core
Clock Rate	≈3 GHz	≈1 GHz
# Cores	4 – 12	≈60
Threads per Core	1 – 2	4
Bandwidth[†]	≈60 GB/s	≈350 GB/s
Thermal Design Power	≈115 W	≈225 W

[†] Peak electrical bandwidth.

Multi-core vs Many-core – Theoretical Peak Performance

Peak GFLOP/s in Single Precision

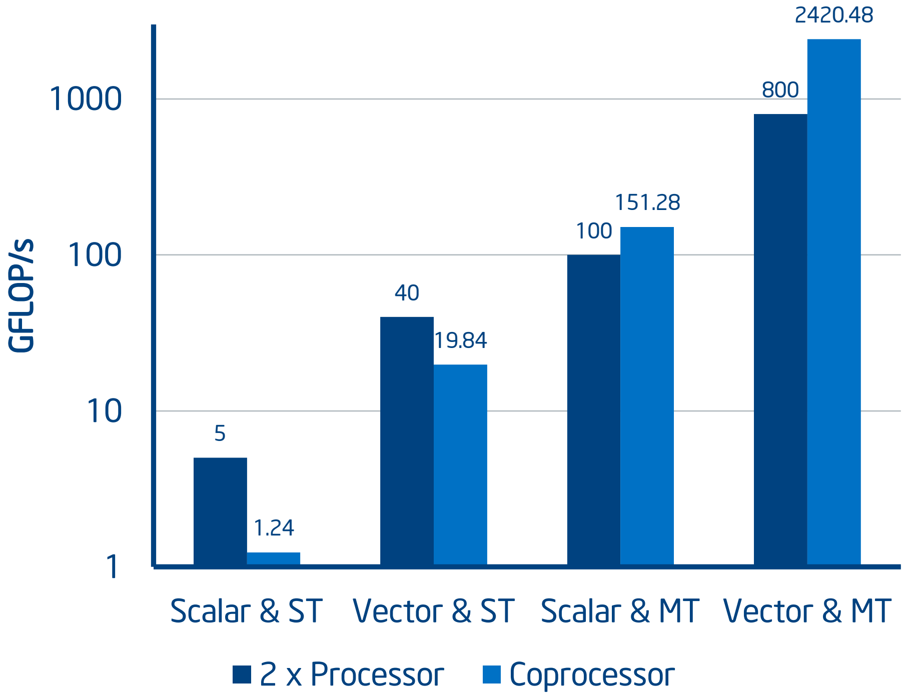
- Clock Rate x Cores x Ops/Cycle x SIMD

2 x Intel® Xeon® Processor E5-2670v2

- 2.5 GHz x 20 cores x 2 ops x 8 SIMD
  = 800 GFLOP/s

Intel® Xeon Phi™ Coprocessor 7120P

- 1.24 GHz x 61 cores x 2 ops x 16 SIMD
  = 2420.48 GFLOP/s



Note the logarithmic scale on the y-axis.
ST = Single Thread, MT = Multiple Threads

Architecture of an Intel® Xeon Phi™ Coprocessor Core

Two Pipelines

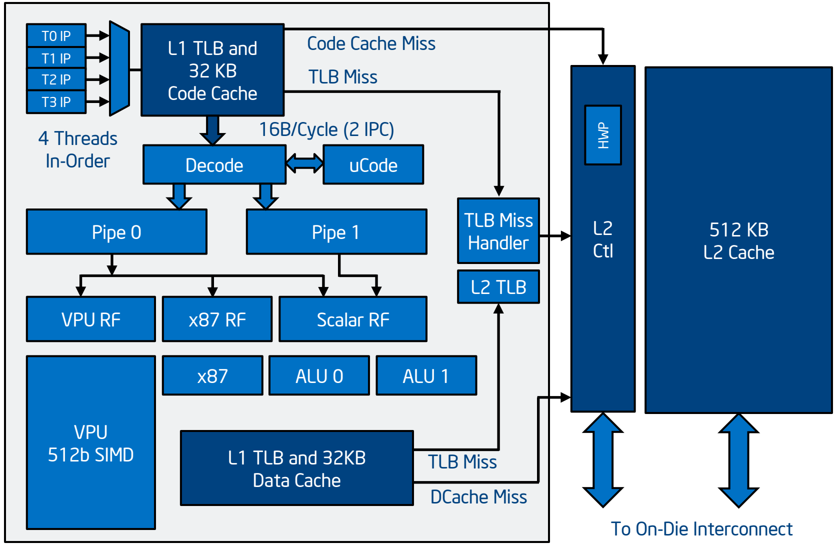
- Scalar unit based on Pentium® processors.
- Dual issue (vector + scalar)

512-bit Vector Processing Unit (VPU)

4 Hardware Threads

- Cannot issue instructions back to back from same thread
- RR scheduling to hide 4 cycle latency

512 KB L2 Cache



Architecture of an Intel® Xeon Phi™ Coprocessor

Cache

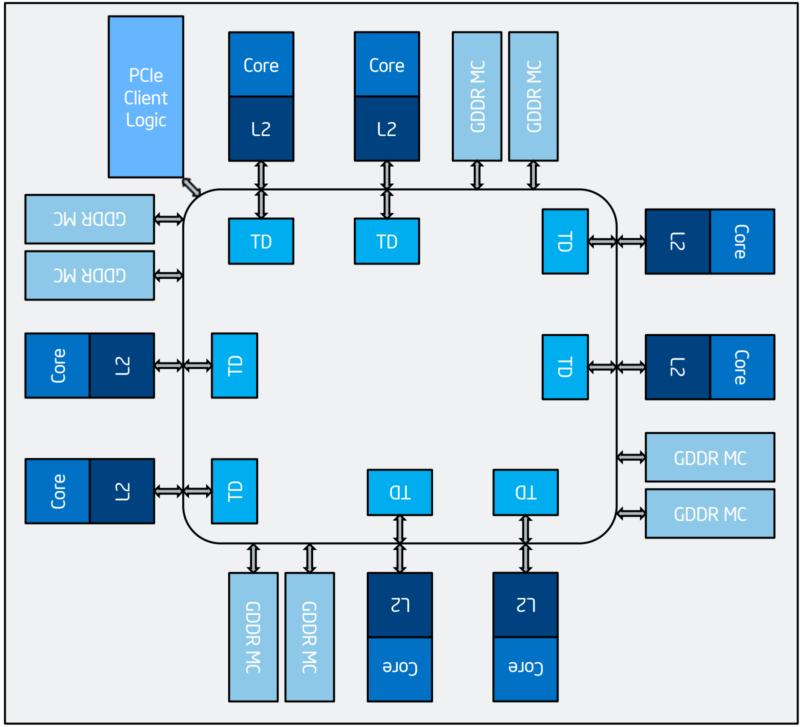
- 32 KB L1 / 512 KB L2 per core
- Fully coherent

Core Communication

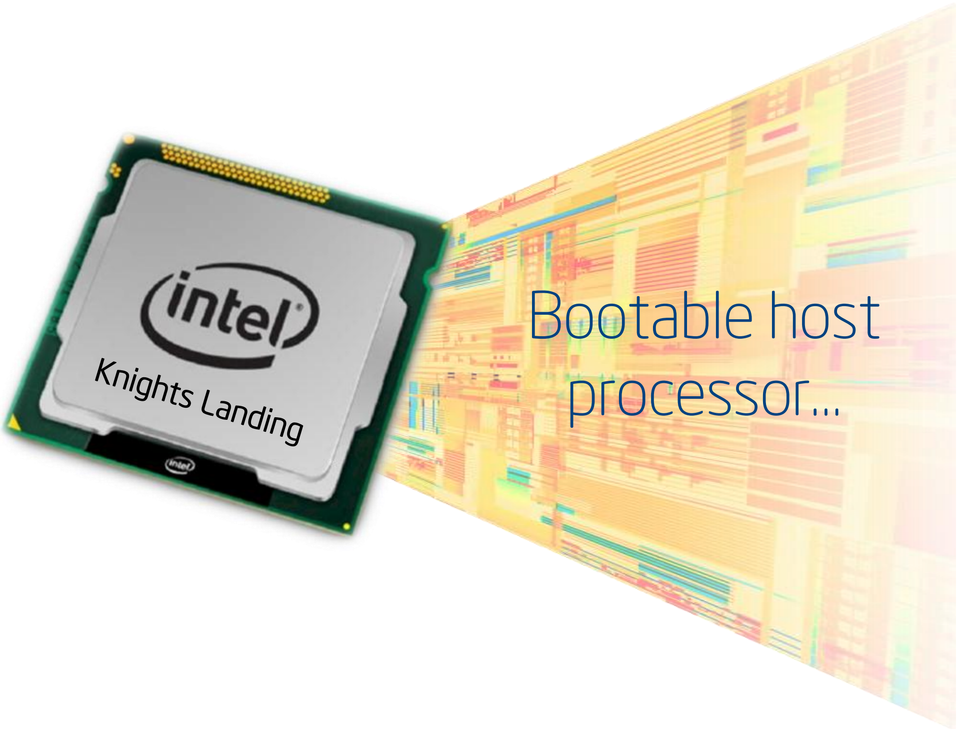
- Bi-directional ring buffer
- 8 GB GDDR5 shared by all cores

PCIe*

- Gen2
- 16 channels



Future Intel® Xeon Phi™ Processor: Knights Landing



Bootable host processor…

Unconstrained by PCIe* offload bottlenecks

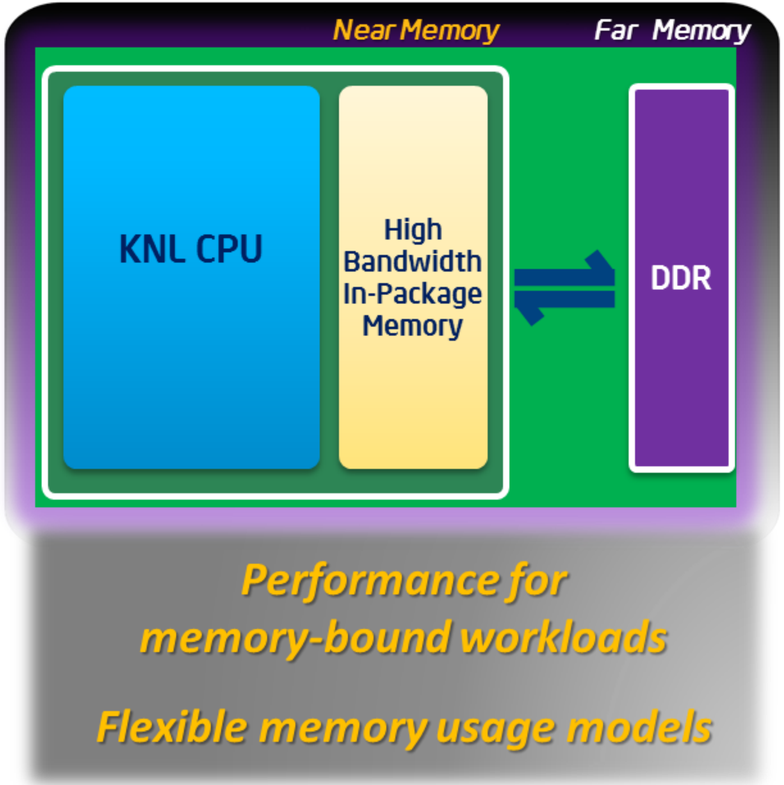
Excellent **compute density** and **power efficiency**

High **memory bandwidth** through integrated memory

Using Intel® cutting-edge **14nm transistor** technology

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Knights Landing Integrated On-Package Memory



Performance for memory-bound workloads

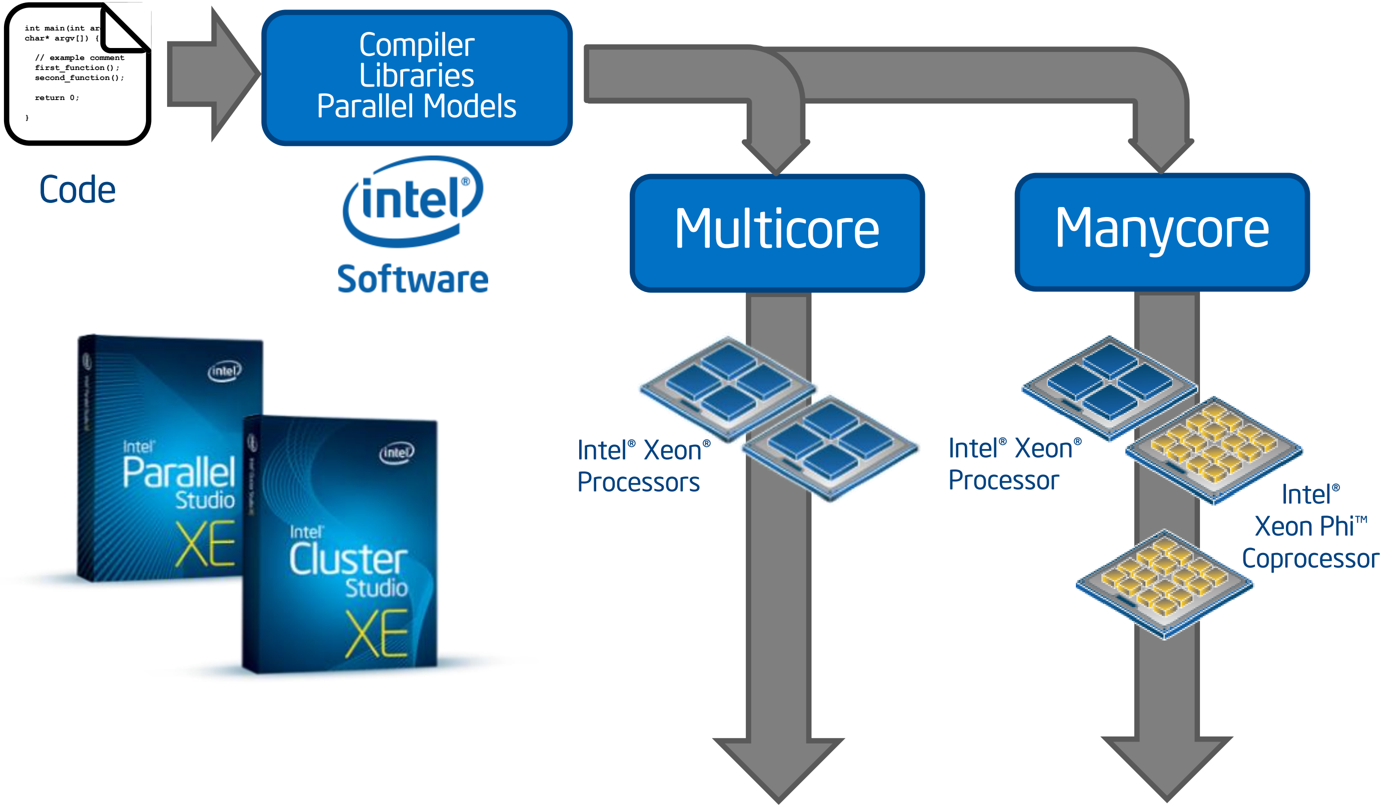
Flexible memory usage models

Cache Model Let the hardware automatically manage the integrated on-package memory as an "L3" cache between KNL CPU and external DDR

Flat Model Manually manage how your application uses the integrated on-package memory and external DDR for peak performance

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.
Diagram is for conceptual purposes only and only illustrates a CPU and memory – it is not to scale, and is not representative of actual component layout.

Consistent Tools & Programming Models



```
int main(int ar
char* argv[]) {

  // example comment
  first_function();
  second_function();

  return 0;
}
```

Code

Compiler
Libraries
Parallel Models

Software

Multicore

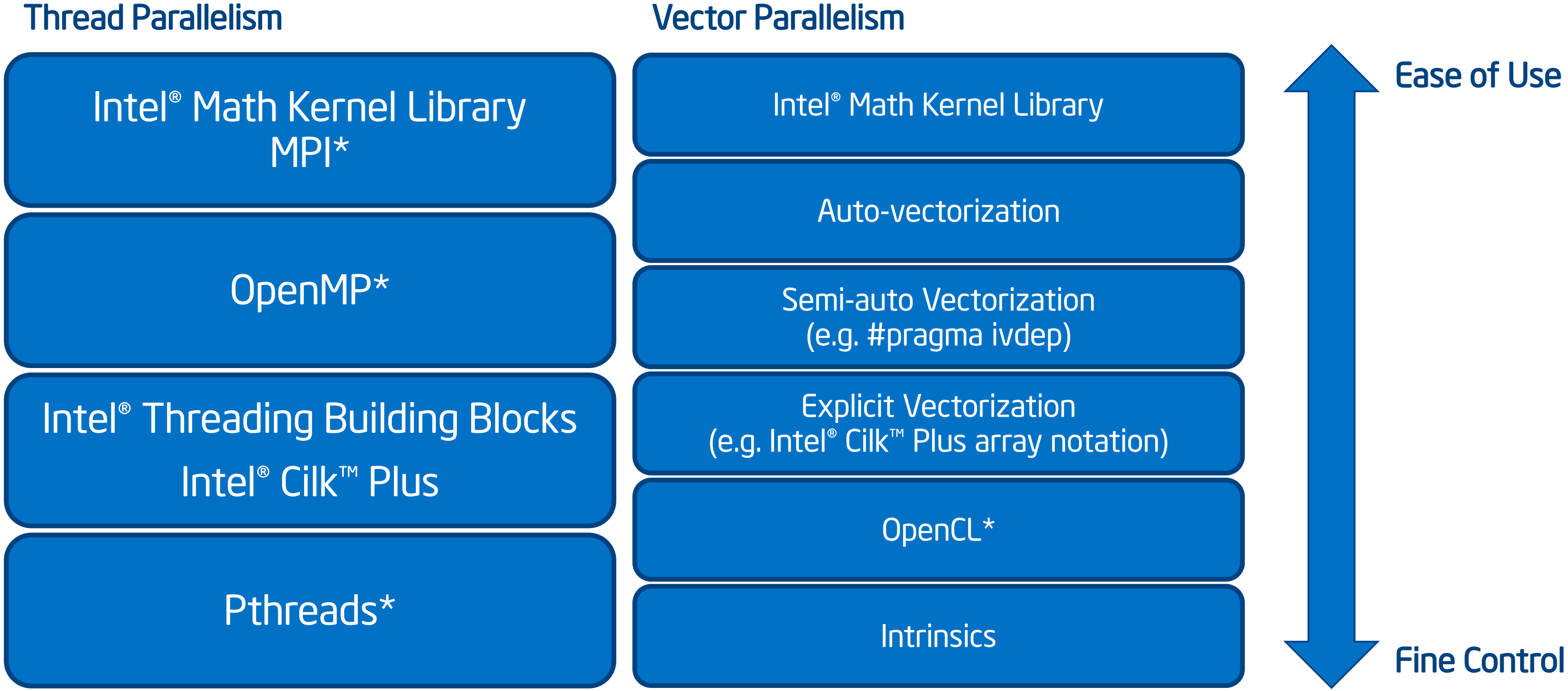
Manycore

Intel® Xeon® Processors

Intel® Xeon® Processor

Intel® Xeon Phi™ Coprocessor

Wide Range of Development Options



Thread Parallelism

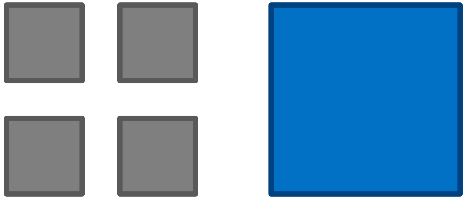
- Intel® Math Kernel Library
MPI*
- OpenMP*
- Intel® Threading Building Blocks
Intel® Cilk™ Plus
- Pthreads*

Vector Parallelism

- Intel® Math Kernel Library
- Auto-vectorization
- Semi-auto Vectorization (e.g. #pragma ivdep)
- Explicit Vectorization (e.g. Intel® Cilk™ Plus array notation)
- OpenCL*
- Intrinsics

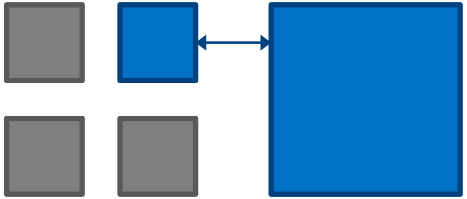
Utilizing an Intel® Xeon Phi™ Coprocessor

Native



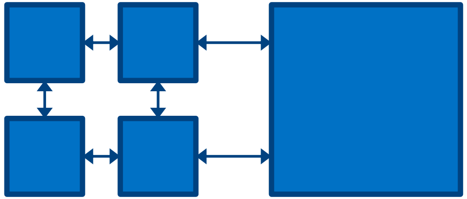
- Target Code:
  Highly parallel (threaded and vectorized) throughout.
- Potential Bottleneck:
  Serial/scalar code.

Offload



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

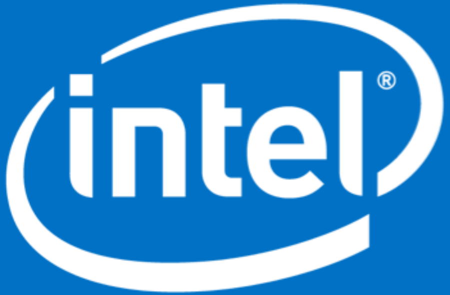
Symmetric



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

Hands-on:
Logging in and Running Applications

Using Beacon @ NICS


intel®

Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Knights Landing and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user

Intel, Look Inside, Xeon, Intel Xeon Phi, Pentium, Cilk, VTune and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

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.

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.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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

THE UNIVERSITY of TENNESSEE UT
NATIONAL LEADERSHIP IN HPC

SC13
Denver, CO | 2013

Administering the Phi in a Cluster Environment

R. Glenn Brook

Director, Application Acceleration Center of Excellence
National Institute for Computational Sciences
glenn-brook@tennessee.edu

Paul Peltz Jr

HPC Systems Administrator
National Institute for Computational Sciences
ppeltz@tennessee.edu

intel
MIC
Partner

Partnership for Extreme Scale Computing
intel
CRAY
NICS

The Beacon Project

- Funded by NSF to port and optimize scientific codes to the Intel® Xeon Phi™ coprocessor
- State-funded expansion focuses on energy efficiency, big data applications, and industry
- Example Codes: PSC, H3D, OMEN, ENZO, MADNESS, NWCHEM, Amber, MILC, and MAGMA

<table>
<tr><th colspan="2">Beacon
Cray CS300-AC™ Cluster Supercomputer
Peak Performance: 210.1 TFLOP/s</th></tr>
<tr><td>Nodes</td><td>4 service, 6 I/O, 48 compute</td></tr>
<tr><td>Interconnect</td><td>FDR IB Fat Tree</td></tr>
<tr><td>Interconnect Bandwidth</td><td>56 GB/s(total)</td></tr>
<tr><td>CPU model</td><td>Intel Xeon E5-2670</td></tr>
<tr><td>CPUs per node</td><td>2 8-core, 2.6GHz</td></tr>
<tr><td>Memory Bandwidth</td><td>128 GB/s (peak)</td></tr>
<tr><td>RAM per node</td><td>256 GB</td></tr>
<tr><td>SSD per node</td><td>2 x 480 GB (compute),16 x 300 GB (I/O)</td></tr>
<tr><td>Intel® Xeon Phi Coprocessors per node</td><td>4 x 5110P 60-core, 1.053GHz
8 GB GDDR5 RAM</td></tr>
</table>

Basic Environment

- Installation
  - Diskfull
  - CentOS 6.2
  - Static IPs
- Workload Manager
  - Torque 4.2.6/Moab 7.2
- MPSS 3.1.2
  - In the process of upgrading cluster to MPSS 3.2.x (June)
- Site wide 1.3PB Lustre File System
  - (/lustre/medusa)
- Local 17TB Lustre on ZFS based SSD file system
  - (/lustre/scratch)

MPSS Installation Procedures

- Preparing the System
  - Set up /etc/hosts to follow this format. Many scripts will depend on this later.

```
10.39.20.12         beacon001          beacon001-eth0
10.39.20.13         beacon001-mic0
10.39.20.14         beacon001-mic1
10.39.20.15         beacon002          beacon002-eth0
10.39.20.16         beacon002-mic0
10.39.20.17         beacon002-mic1
```

- Generate ssh keys
- Install OFED Software Stack

MPSS Installation Procedures

- MPSS Stack Installation
  - pdsh -w cluster[xxx-xxx] yum -y install --nogpgcheck --noplugins --disablerepo=* $MPSS_LOCATION/*.rpm
  - Install extra RPMS as necessary such as $MPSS_LOCATION/ofed/*.rpm
  - micctrl --initdefaults
    - Copy the /etc/mpss/default.conf and /etc/mpss/mic0.conf files to /etc
    - Use these two files as the template for creating a mic-create-conf.sh script to generate config files for each node
  - Create ifcfg-micbr0 and ifcfg-micX files (External Bridging)
  - micctrl --resetconfig

MPSS Upgrade Procedures

- MPSS Stack Removal
  - pdsh -w cluster[xxx-xxx] $MPSS_LOCATION/uninstall.sh
  - micctrl --initdefaults on test node
    - update template files if necessary
  - micctrl --resetconfig
  - Upgrade firmware with micflash

Torque Prologue/Epilogue

- Prologue
  - Set up users home env on MIC
    - Copy ssh keys to $TMP
    - Build .profile
    - Check health of mpssd service and ofed-mic service
    - Set up user account
    - Export/Mount $TMP, $NODE:/opt/intel, /global/opt, and /lustre/$USER
    - Verify MIC's /etc/hosts and fix if necessary
    - copy over mpiexec.hydra and pmi_proxy to $TMP

Torque Prologue/Epilogue

- Epilogue
  - Unmount all shared file systems
  - Reboot MICs
    - service ofed-mic stop
    - service mpss restart
    - micctrl -Rw
    - micctrl -rwf
    - service ofed-mic start
  - Clean up old processes

Health Checking and Monitoring

- Ganglia
  - Requires post MIC boot RPM installation currently
  - CPU Metrics
    - Intel disables these by default
    - Our testing determined no measurable effect by having them enabled
    - CPU Metrics can be disabled with “gres=noganglia”

Scripting Common Commands

- ssh
  - micssh replaces ssh
    - adds -i $ssh_key option to ssh
- mpiexec
  - micmpiexec replaces mpiexec
    - replaces ssh with micssh
    - passes the appropriate environment variables
    - allows for debuggers such as ddt and totalview

Challenges

- Driver upgrade process (2 to 3.1 and 3.1 to 3.2)
  - The ever changing default.conf and micX.conf files
- The micctrl gamble
  - Intel wants to push everything to using micctrl
  - we need a --dry-run flag
- Intel Compilers and Intel MPI
- Kernel Compatibility
  - rpmbuild --rebuild
    - sufficient for mpss driver
    - difficult for OFED 1.5.4.1 (unless you use OFED-3.5)
- SSH Keys
- OFED 1.5.4.1 or 3.5

Acknowledgements

We wish to thank the National Science Foundation for their support of the Beacon project, along with the State of Tennessee.

Contact Information

Paul Peltz Jr

HPC Systems Administrator
National Institute for Computational Sciences
ppeltz@utk.edu

A UT/ORNL PARTNERSHIP
NATIONAL INSTITUTE FOR COMPUTATIONAL SCIENCES
NICS

Using Intel MPI and OpenMP 4.0 with the Intel Xeon Phi

AACE

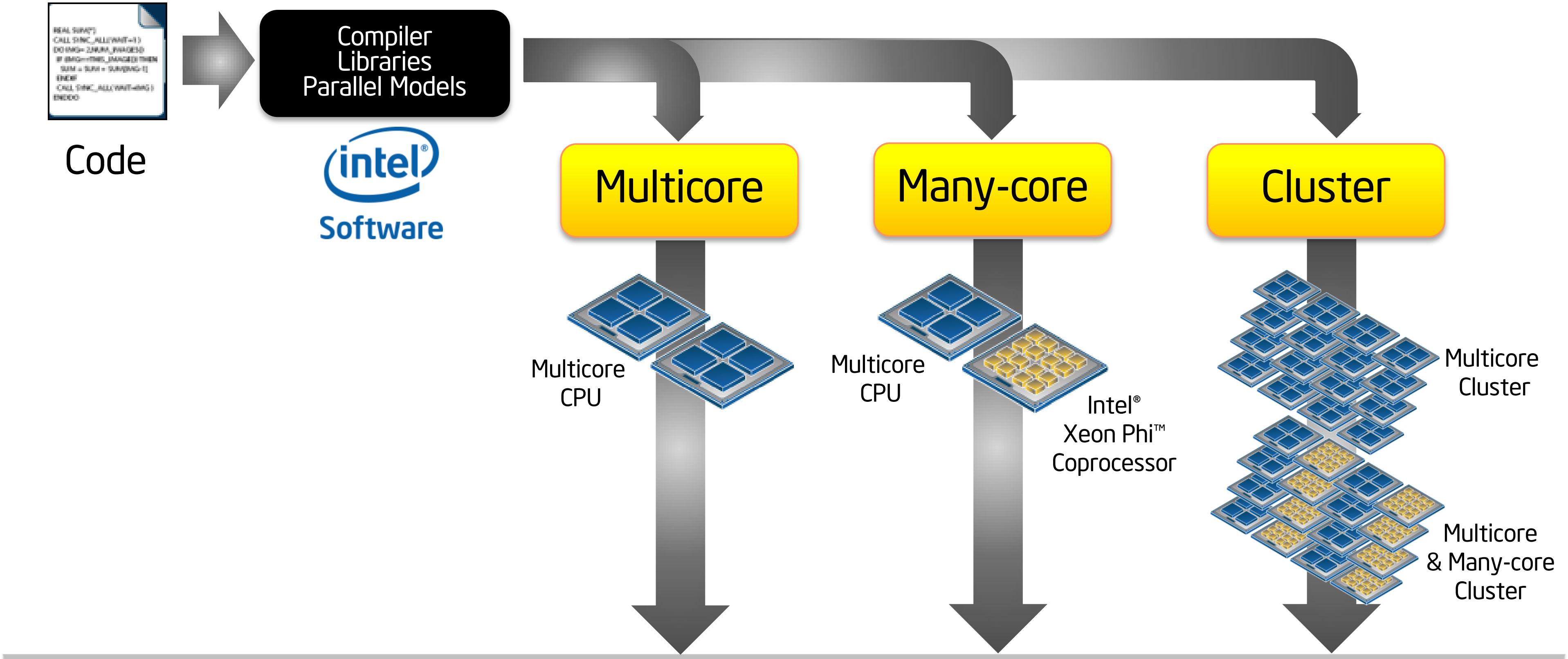
Vincent C. Betro, Ph.D.
Hans Pabst, Intel
Heinrich Blockhorst, Intel
Cray Users Group—May 5, 2014

OVERVIEW INFORMATION

Enabling & Advancing Parallelism

High Performance Parallel Programming

Intel tools, libraries and parallel models extend to multicore, many-core and heterogeneous computing



Code

Compiler
Libraries
Parallel Models

intel Software

Multicore

Many-core

Cluster

Multicore CPU

Multicore CPU

Intel® Xeon Phi™ Coprocessor

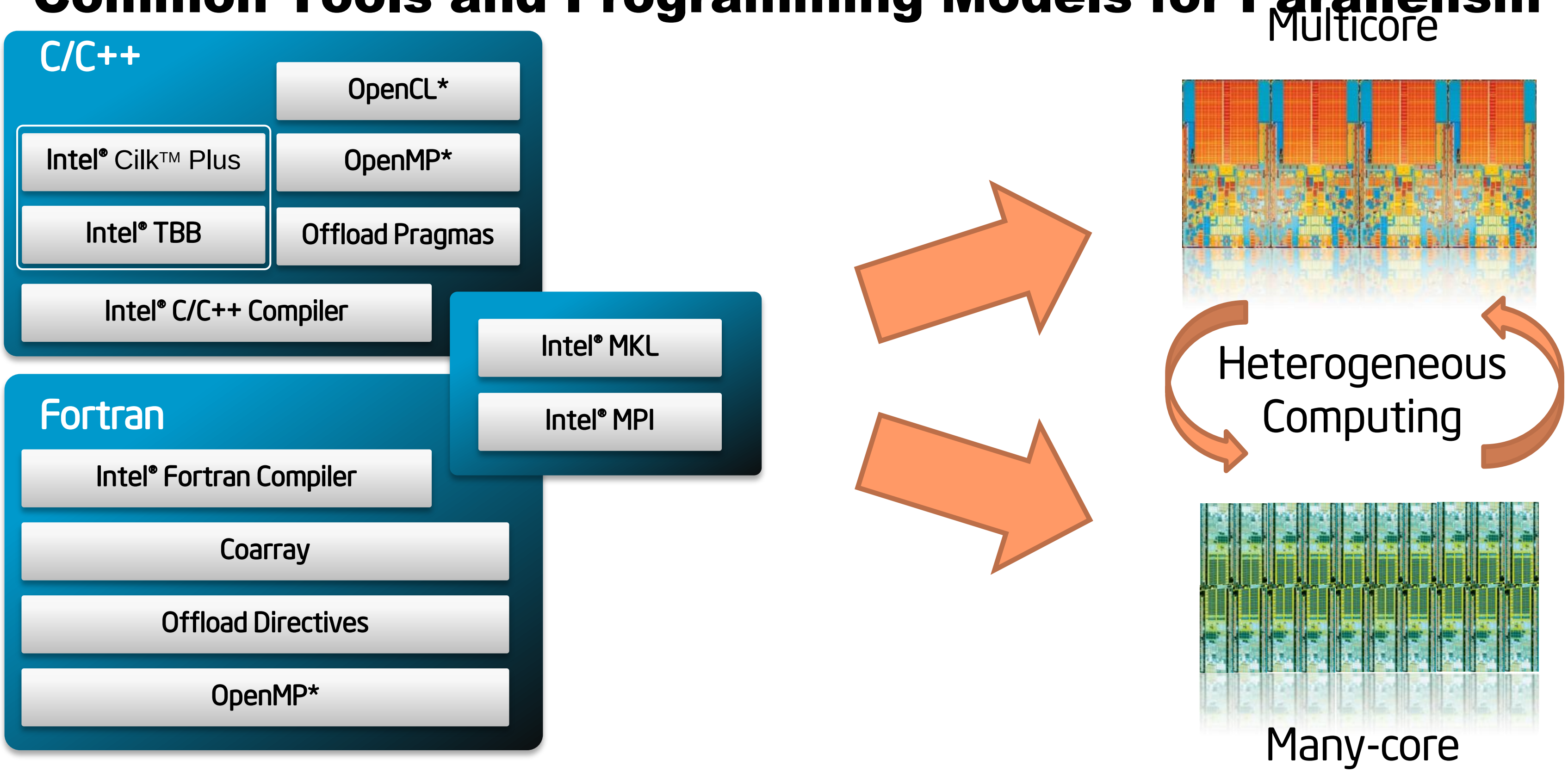
Multicore Cluster

Multicore & Many-core Cluster

Use One Software Architecture Today. Scale Forward Tomorrow.

Preserve Your Development Investment

Common Tools and Programming Models for Parallelism



C/C++

OpenCL*

Intel® Cilk™ Plus

OpenMP*

Intel® TBB

Offload Pragmas

Intel® C/C++ Compiler

Intel® MKL

Intel® MPI

Fortran

Intel® Fortran Compiler

Coarray

Offload Directives

OpenMP*

Multicore

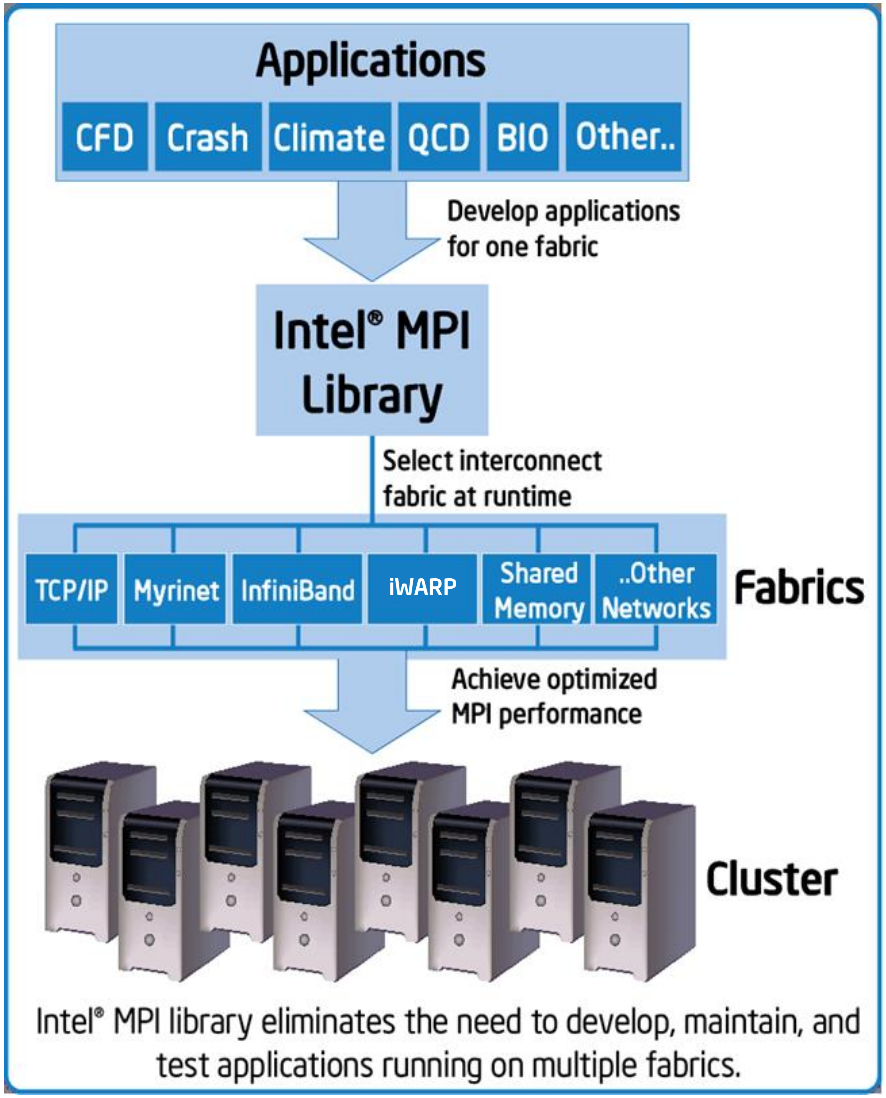
Heterogeneous Computing

Many-core

Develop Using Parallel Models that Support Heterogeneous Computing

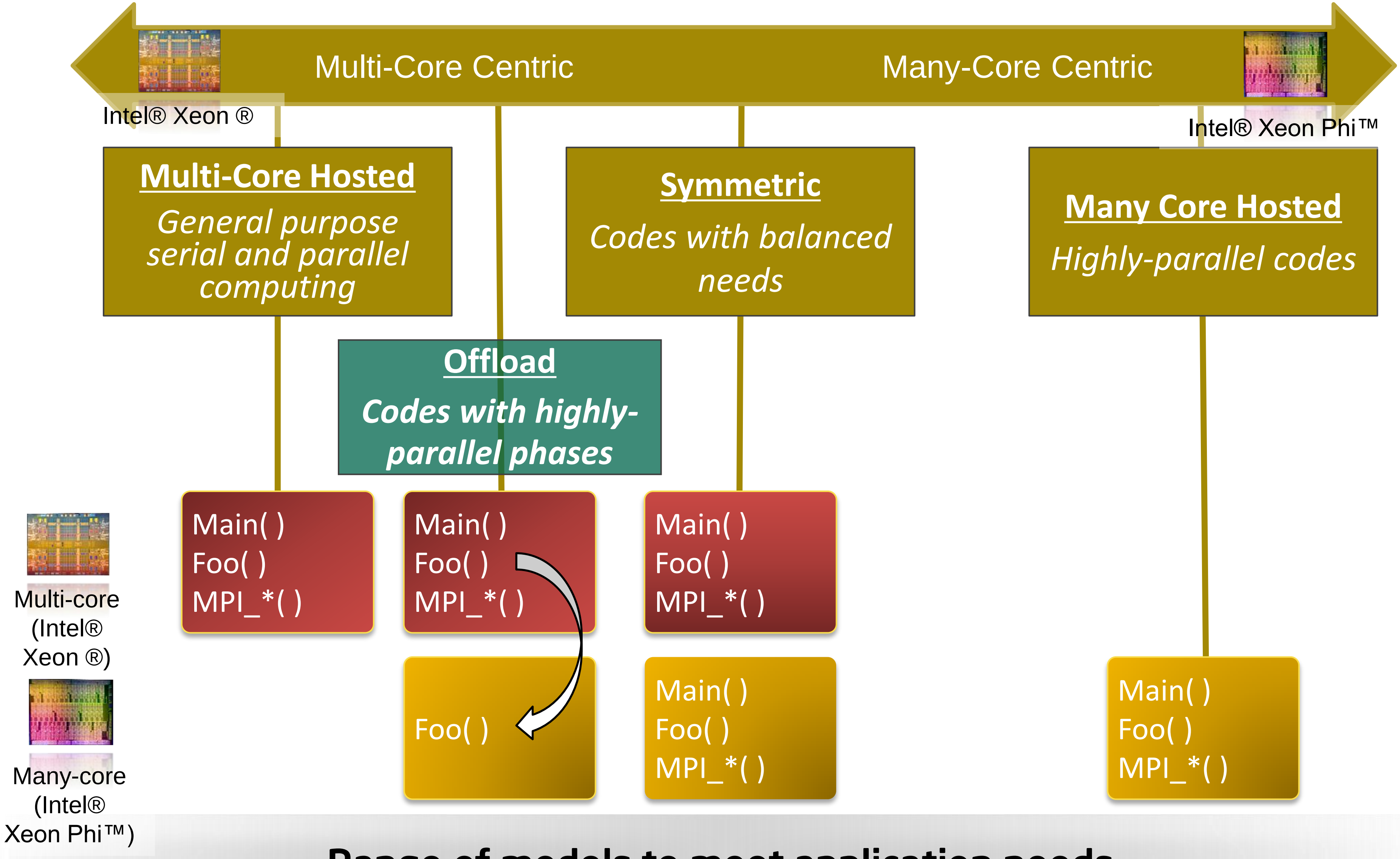
Intel® MPI Library Overview

- Intel is a leading vendor of MPI implementations and tools
- Optimized MPI application performance
  - Application-specific tuning
  - Automatic tuning
- Low latency
  - Industry leading latency
- Interconnect Independence & Runtime Selection
  - Multi-vendor interoperability
  - Performance optimized support for the latest OFED capabilities through DAPL 2.0
- More robust MPI applications
  - Seamless interoperability with Intel® Trace Analyzer and Collector



Intel® MPI library eliminates the need to develop, maintain, and test applications running on multiple fabrics.

Spectrum of Programming Models and Mindsets



Multi-Core Centric

Many-Core Centric

Intel® Xeon ®

Intel® Xeon Phi™

Multi-Core Hosted

General purpose serial and parallel computing

Offload

Codes with highly-parallel phases

Symmetric

Codes with balanced needs

Many Core Hosted

Highly-parallel codes

Main( )
Foo( )
MPI_*( )

Main( )
Foo( )
MPI_*( )

Foo( )

Main( )
Foo( )
MPI_*( )

Main( )
Foo( )
MPI_*( )

Main( )
Foo( )
MPI_*( )

Multi-core (Intel® Xeon ®)

Many-core (Intel® Xeon Phi™)

Range of models to meet application needs

Levels of communication speed

- Current clusters are not homogenous regarding communication speed:
  - Inter node (InfiniBand*, Ethernet, etc)
  - Intra node
    - Inter sockets (Quick Path Interconnect)
    - Intra socket

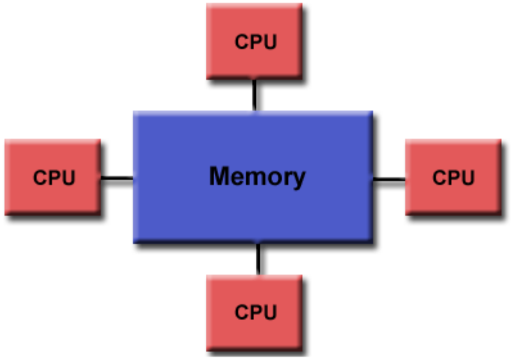
- Two additional levels to come with Intel® Xeon Phi™ coprocessor:
  - Host-coprocessor communication
  - Inter coprocessor communication

Selecting network fabrics

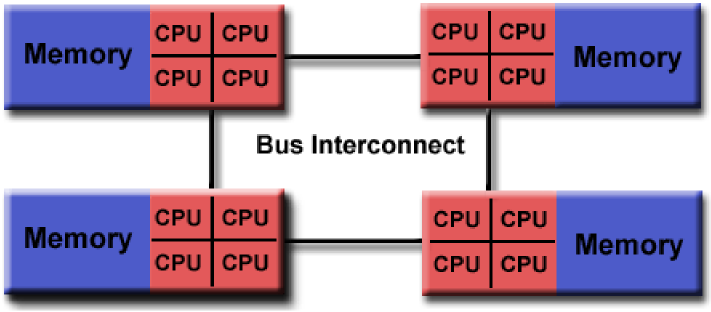
- Intel® MPI automatically selects the best available network fabric it can find.
  - Use `I_MPI_FABRICS` to select a different communication device explicitly
- The best fabric is usually based on InfiniBand* (dapl, ofa) for inter node communication and shared memory for intra node
- Available for Intel® Xeon Phi™ coprocessor:
  - `shm, tcp, ofa, dapl`
  - Availability checked in the order `shm:dapl, shm:ofa, shm:tcp` (intra:inter)

MPI vs OpenMP: What’s the Difference?

- OpenMP is used in a shared memory space
- Often referred to as threading, allows multiple tasks to occur simultaneously which are:
  - Embarrassingly parallel
  - Are working in disjoint areas of memory
  - Can have any “intersections” of threads be caught using atomic or critical sections



Shared Memory (UMA)



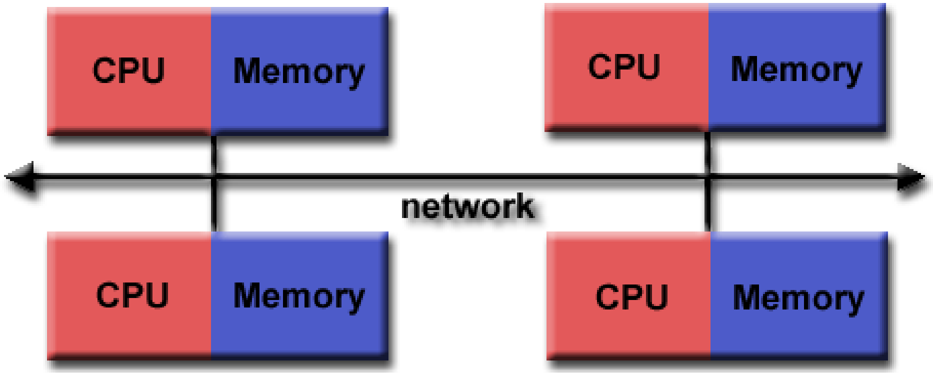
Shared Memory (NUMA)

Diagrams courtesy of Lawrence Livermore Nat’l Lab

https://computing.llnl.gov/tutorials/parallel_comp/#MemoryArch

MPI vs OpenMP: What’s the Difference?

- MPI is used in a distributed memory space
- Often referred to as message passing, allows multiple chunks of a problem’s domain to be computed on simultaneously
- Each computer functions alone with the network acting as the bridges between to update values on process boundaries between iterations

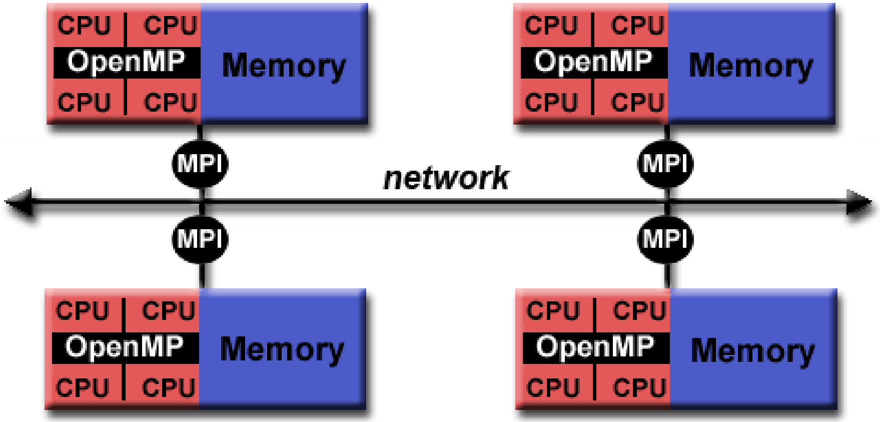


Diagrams courtesy of Lawrence Livermore Nat’l Lab

https://computing.llnl.gov/tutorials/parallel_comp/#MemoryArch

MPI vs OpenMP: What's the Difference?

- The two paradigms can be used together
- One example would be having a few MPI ranks on each Xeon Phi and then having many threads spawned by each rank
- Another would be to have MPI tasks on CPUs and offload threaded sections to Xeon Phis



Diagrams courtesy of Lawrence Livermore Nat'l Lab

https://computing.llnl.gov/tutorials/parallel_comp/#MemoryArch

Xeon Phi Programming Models

- Native Mode
  - Everything runs on the MIC
  - All libraries need to be recompiled with -mmic
- Offload Mode
  - Serial portion runs on host
  - Parallel portions are offloaded and run on the MIC

Xeon Phi Programming Models

- Native Mode
  - Everything runs on the MIC
  - All libraries need to be recompiled with –mmic

…is all we will discuss here. Offload will be covered next by Hans, including OpenMP 4.0!

However, we will briefly chat about offload in terms of having many MPI ranks do it.

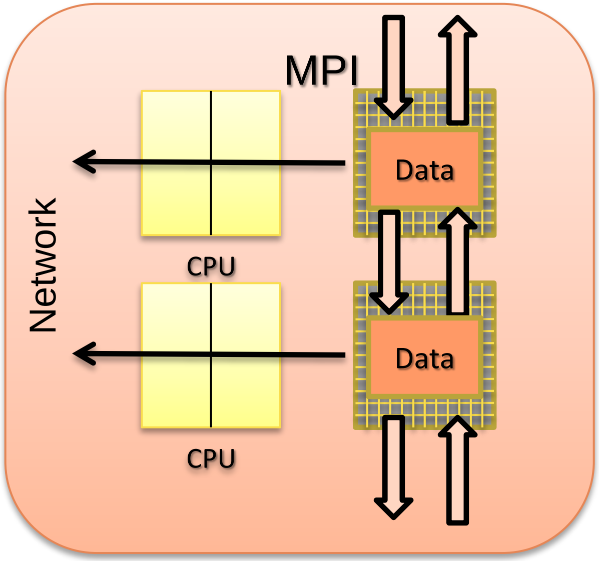
CO-PROCESSOR OR HOST MPI

Intel MPI

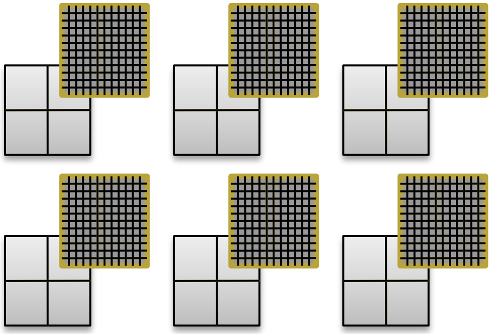
- The installed Intel MPI library implements the Message Passing Interface, version 2.2 (MPI-2.2) specifications
- 3 Programming models are supported
  - Co-processor only model
  - Symmetric model
  - MPI offload model
- Intel MPI compilers have an extra ‘i’ in their name: `mpiicc, mpiicpc, mpiifort`
- MPI applications should be launched from the host compute node using `micmpiexec`

Coprocessor-only Programming Model

- MPI ranks on Intel® Xeon Phi™ coprocessor(only)
- All messages into/out of coprocessors
- Intel® Cilk^{TM} Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads used directly within MPI processes



Homogenous network of many-core CPUs



Build Intel® Xeon Phi™ binary using the Intel® compiler.

Upload the binary to the Intel® Xeon Phi™ coprocessor.

Run instances of the MPI application on Intel® Xeon Phi™ coprocessor nodes.

Coprocessor-only Programming Model

- **MPI ranks on the Intel® Xeon Phi™ coprocessor(s) only**
- **MPI messages into/out of the coprocessor(s)**
- **Threading possible**

- Build the application for the Intel® Xeon Phi™ coprocessor

```
# mpiicc -mmic -o test_hello.MIC test.c
```

- Launch the application on the coprocessor from host

```
# export I_MPI_MIC=enable
# mpirun -n 2 -host host-mic0 ./test_hello.MIC
```

- Alternatively: login to the Intel® Xeon Phi™ coprocessor and start the MPI run there!
- No NFS: Upload the Intel® Xeon Phi™ executable and add working directory flag

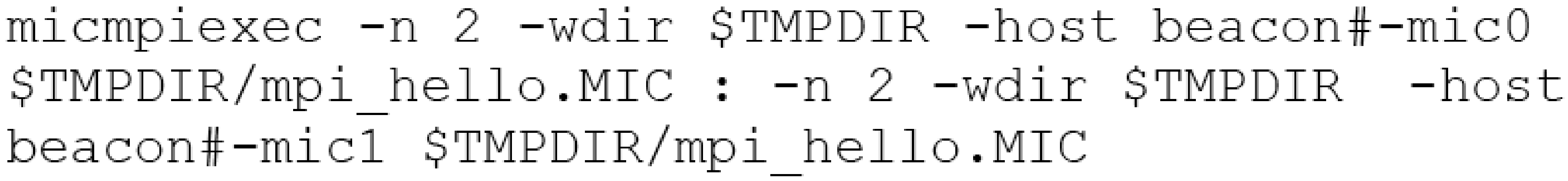
```
# scp ./test_hello.MIC host-mic0:/my_mic_dir/
# mpirun … –wdir /my_mic_dir/ …
```

Using Intel MPI: MIC 2 MIC

- If one MIC card is not sufficient for your domain decomposition, you may use all MIC cards on the node or even multiple MIC cards on multiple nodes.

- If you need to use MICs on multiple nodes, you must request multiple nodes with qsub and check which ones they are with “cat $PBS_NODEFILE”

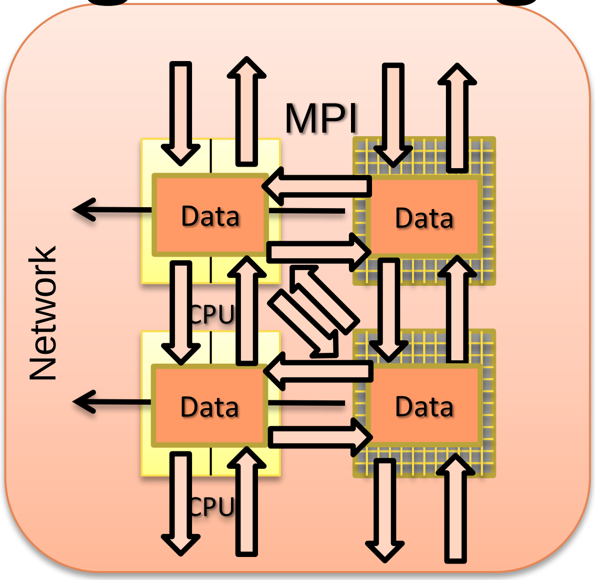
- Note that the MIC requires the –wdir argument



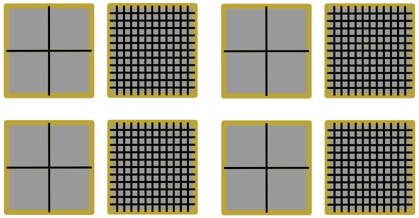
```
micmpiexec -n 2 -wdir $TMPDIR -host beacon#-mic0
$TMPDIR/mpi_hello.MIC : -n 2 -wdir $TMPDIR  -host
beacon#-mic1 $TMPDIR/mpi_hello.MIC
```

Symmetric Programming Model

- MPI ranks on Intel® Xeon Phi™ Architecture and host CPUs
- Messages to/from any core
- Intel® Cilk^TM Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* used directly within MPI processes



Heterogeneous network of homogeneous CPUs



Build binaries by using the resp. compilers targeting Intel® 64 and Intel® Xeon Phi™ Architecture.

Upload the binary to the Intel® Xeon Phi™ coprocessor.

Run instances of the MPI application on different mixed nodes.

Symmetric model

- **MPI ranks on the coprocessor(s) and host CPU(s)**
- **MPI messages into/out of the coprocessor(s) and host CPU(s)**
- **Threading possible**

- Build the application for Intel®64 and the Intel® Xeon Phi™ Architecture separately

```
# mpiicc -o test_hello test.c
# mpiicc –mmic -o test_hello.MIC test.c
```

- Launch the application on the host and the coprocessor

```
# export I_MPI_MIC=enable
# mpirun -n 2 -host <hostname> ./test_hello :
         -n 2 -host host-mic0 ./test_hello.MIC
```

- No NFS: Upload the Intel® Xeon Phi™ executable and add flag

```
# scp ./test_hello.MIC host-mic0:/my_mic_dir/
# mpirun … : –wdir /my_mic_dir/ …
```

Utilize the POSTFIX env variable

Support for NFS-shared cards

- **Assumption: The current working directory is available with identical path on the coprocessor (e.g. mounted)**
- **Specify the suffix of the coprocessor binary**

```
# export I_MPI_MIC_POSTFIX=.MIC
```

- **Specify the node names in a file**

```
# cat mpi_hosts
  host
  host-mic0
```

- **Execute in symmetric mode on the host and the coprocessor**

```
# export I_MPI_MIC=enable

# mpirun -f mpi_hosts -n 4 ./test_hello
```

- **The binary** `./test_hello${I_MPI_MIC_POSTFIX}` **will be used by mpirun on the coprocessor**

Utilize the PREFIX env variable

Support for NFS-shared cards

- Assumption: The current working directory is available with identical path on the coprocessor (e.g. mounted)
- Place the coprocessor binary in a separate directory, but with identical basename of the host

  ```
  # mpiicc –mmic -o ./MIC/test_hello test.c
  ```
- Specify the prefix of the coprocessor binary

  ```
  # export I_MPI_MIC_PREFIX=./MIC/
  ```
- Execute in symmetric mode on the host and the coprocessor

  ```
  # export I_MPI_MIC=enable
  ```

  ```
  # mpirun –f mpi_hosts –n 4 ./test_hello
  ```
- The binary `${I_MPI_MIC_PREFIX}./test_hello` will be used by mpirun on the coprocessor

Launching an MPI Application with Manually Specified Hosts

- Launch an MPI application on mic0 of the current compute node with

  -
    ```
    micmpiexec -n 1 -wdir $TMPDIR -host beacon#-
    mic0 $TMPDIR/application.MIC
    ```
- Launch an MPI application on both mic0 and mic1 of the current compute node with

  -
    ```
    micmpiexec -n 1 -wdir $TMPDIR -host beacon#-
    mic0 $TMPDIR/application.MIC : -n 1 -wdir
    $TMPDIR -host beacon#-mic1
    $TMPIDR/application.MIC
    ```
- Launch an MPI application on both MICs and the compute node with

  -
    ```
    micmpiexec -n 1 -wdir $TMPDIR -host beacon#-
    mic0 $TMPDIR/application.MIC : -n 1 -wdir
    $TMPDIR -host beacon#-mic1
    $TMPIDR/application.MIC : -n 1 -host beacon#
    ./application
    ```

Launching an MPI Application with a Machine File

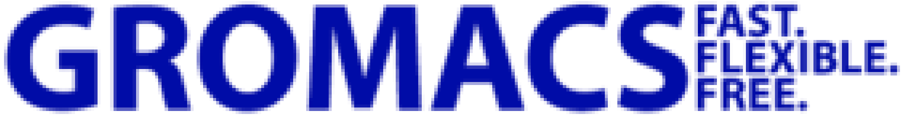
- The machine file needs to be of the form <host>:<number of ranks>
- Sample machine file named hosts_file:
  - `beacon11:8`
  - `beacon12:8`
  - `beacon11-mic0:2`
  - `beacon11-mic1:2`
  - `beacon12-mic0:2`
  - `beacon12-mic1:2`
- Launch the MPI application using
  - `micmpiexec -machinefile hosts_file -n 16 ./application : -n 8 -wdir $TMPDIR -genv LD_LIBRARY_PATH $TMPDIR/lib:/lib64:/lib $TMPDIR/application.MIC`

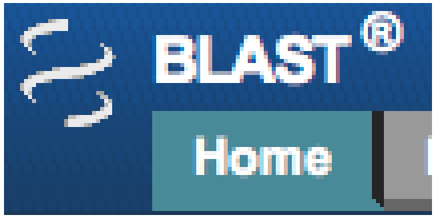
Launching an MPI Application with Process Pinning

- export I_MPI_PIN=1
- export I_MPI_PIN_PROCESSOR_LIST="0-6,8-14"
- export I_MPI_DEBUG=4 or 5

- Allows you to pin MPI processes to sockets on the host.

Example of Native Mode MPI application experiences





Both GROMACS and BLAST are bioinformatics packages used to look at proteins, lipids, and nucleic acids, often for the purpose of Genomics research or drug docking research.

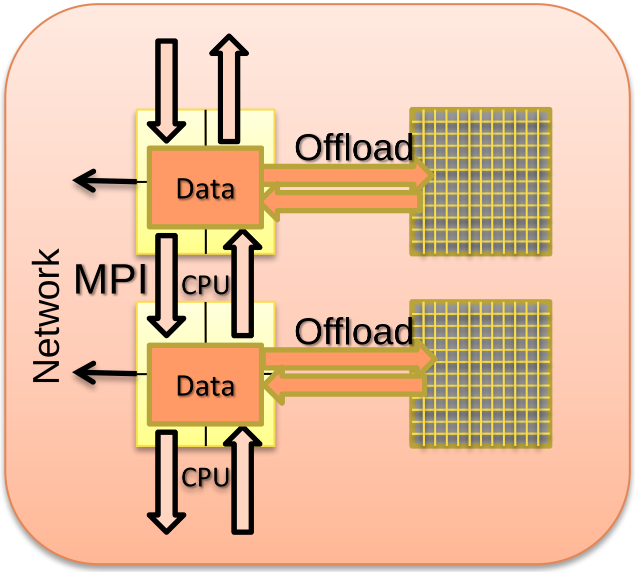
Both use the MPI-OpenMP hybrid model of execution, where the problem is decomposed in an embarrassingly parallel fashion over several MPI ranks and then the consequent divisions are threaded in execution.

This approach allows the developers to take advantage of the large number of threads available on the Intel Xeon Phi along with allowing them to decompose the problem well so it fits on a given coprocessor.

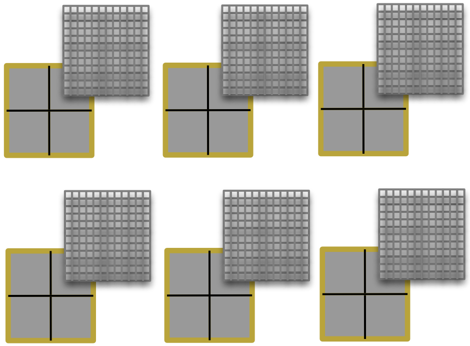
MPI on HOST + OFFLOAD on PHI

MPI+Offload Programming Model

- MPI ranks on Intel® Xeon® processors (only)
- All messages into/out of host CPUs
- Offload models used to accelerate MPI ranks
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* within Intel® Xeon Phi™ coprocessors



Homogenous network of heterogeneous nodes



Build Intel® 64 executable with included offload by using the Intel® compiler.

Run instances of the MPI application on the host, offloading code onto the coprocessor.

Advantages of more cores and wider SIMD for certain applications

MPI+Offload Programming Model

- **MPI ranks on the host CPUs only**
- **MPI messages into/out of the host CPUs**
- **Intel® Xeon Phi™ coprocessor as an accelerator**

- Compile for MPI and internal offload

```
# mpiicc -o test test.c
```

- Latest compiler compiles by default for offloading if offload construct is detected!
  - Switch off by `-no-offload` flag
- Execute on host(s) as usual

```
# mpirun -n 2 ./test
```

- MPI processes will offload code for acceleration

Launching an MPI Application with Each Rank Offloading to a Different Card

- When specifying the target for offload by a given rank, one can use mic:N, where N=0,1,2,3… for as many Xeon Phis as are on the system.
- Despite the fact that the pragmas are pre-processed, one can use N as a variable to allow it to depend on which MPI rank is calling the offload at runtime.

MPI+Offload Support

- How to control mapping of threads on the coprocessor?
  - How do I avoid that offload of first MPI process interferes with offload of second MPI process, i.e. by using identical cores/threads on the coprocessor?
  - Default: No special support (now). Offloads from MPI processes handled by system like offloads from independent processes (or users).
- Define thread affinity manually per single MPI process:

```
# export OMP_NUM_THREADS=8
# mpirun -env KMP_AFFINITY=proclist=[1-8],explicit -n 1
         -host myHost ./test_mpioffload :
         -env KMP_AFFINITY=proclist=[9-16],explicit -n 1
         -host myHost ./test_mpioffload : ...
           ...
```

MPI+Offload Support (ctd.)

- **Alternative: Use `KMP_PLACE_THREADS` and Intel® MPI rank number `PMI_RANK` in a wrapper script:**

```
# cat ./wrapoffload.sh
  cores=$(( (OMP_NUM_THREADS+3)/4 ))
  offset=$(( cores*PMI_RANK ))
  export KMP_PLACE_THREADS=${cores}Cx4T,${offset}O
  ./test_mpioffload
```

```
# export OMP_NUM_THREADS=8

# mpirun -n 4 -host myHost ./wrapoffload.sh
```

- **The mapping will be:**

```
MPI rank 0: KMP_PLACE_THREADS=2Cx4T,0O == [1-8]

MPI rank 1: KMP_PLACE_THREADS=2Cx4T,2O == [9-16]
```

OFFLOADING TO MULTIPLE CARDS FROM ONE OR MORE HOST PROCESSES

Simulataneous Computing using OpenMP

- Any OMP call blocks until the statement completes, unless the “nowait” modifier is used
- To use both the host and MIC simultaneously, multiple threads need to be executed on host
  - One or more threads that contain an offload call
  - Other threads have the host do some work
- With OpenMP, this achieved using OpenMP task calls

Offloading to multiple cards from one host process

- **Unoptimized fashion: http://software.intel.com/en-us/forums/topic/393649**

```
//Allocate memory on each of N mics
for(N=0; N<=1; N++){
#pragma offload_transfer target(mic:N) nocopy(pi: alloc_if(1) free_if(0)) signal(&test1[N])
}
for(N=0; N<=1; N++){
#pragma offload target(mic:N) in(N,num_steps,step,sum) inout(pi[N]) signal(&test[N]) wait(&test1[N])
 {
 #pragma omp parallel for reduction(+:sum)
 for (i=0;i<iter; i++)
 <parallel loop body>
 }
}
//Free memory on mic
for(N=0; N<=1; N++){
#pragma offload_transfer target(mic:N) nocopy(pi: alloc_if(0) free_if(1)) wait(&test[N])
}
```

Offloading to multiple cards from one host process

- **According to OpenMP 4.0 (and in my opinion to be pedantic and make sure it behaves):**

```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
  #pragma omp target(mic) OR #pragma offload target(mic)
  {
      <various serial code>
      #pragma omp parallel for
      for (int i=0; i<limit; i++)
         <parallel loop body>
  }
#pragma omp task
  {<host code or another offload>}
}
```

Example of Offload Mode OpenMP application experiences

- **NAMD** (**NA**noscale **M**olecular **D**ynamics program)

  - freeware molecular dynamics simulation package
  - uses Charm++ parallel programming model
  - simulates large systems (millions of atoms)

  Charm++ has been ported to the Intel Xeon Phi, and NAMD is being run across several Intel Xeon Phis in offload mode on Beacon and other resources.

Useful Environment Variables

- **The following applies only to offload mode execution**
- **All environment variables defined on the host are replicated on the MIC in offload mode**
- **To modify specific MIC vaules, `MIC_ENV_PREFIX` must be defined**

```
OMP_NUM_THREADS=8
OMP_STACKSIZE=16M
MIC_ENV_PREFIX=MIC_
```

```
MIC_OMP_NUM_THREADS=96
MIC_OMP_STACKSIZE=4M
```

- **For `csh: setenv ENV_VARIABLE VALUE`**
- **For `sh: export ENV_VARIABLE=VALUE`**

Useful Environment Variables part 2

- `OFFLOAD_REPORT` can be useful when trying to debug code that offloads
  - `OFFLOAD_REPORT=1`
    - Gives basic information (e.g. CPU time) about whether code blocks marked for offload are running on the host or coprocessor
  - `OFFLOAD_REPORT=2`
    - Gives detailed information (e.g. CPU time and data transfer) about the offload process
- Use `MIC_HOST_LOG` to output traces to a file
  - `MIC_HOST_LOG=~/app/mic.log`

MPI+OPENMP ON CO-PROCESSOR (OR HOST)

Running Hybrid Code in Native Mode

- One major advantage of running on the Xeon Phi is that despite its lower clock speed per core, there are an egregious number of threads that can be spawned from LOCAL MPI ranks on a card
- Here, we will discuss a paradigm for doing this as well as how to determine the optimal combination of MPI ranks and threads such that your domain fits on the cards and the threads remain saturated throughout computation

Traditional Cluster Computing

- MPI is »the« portable cluster solution
- Parallel programs use MPI over cores inside the nodes
  - Homogeneous programming model
  - "Easily" portable to different sizes of clusters
  - No threading issues like »False Sharing« (common cache line)
  - Maintenance costs only for one parallelization model

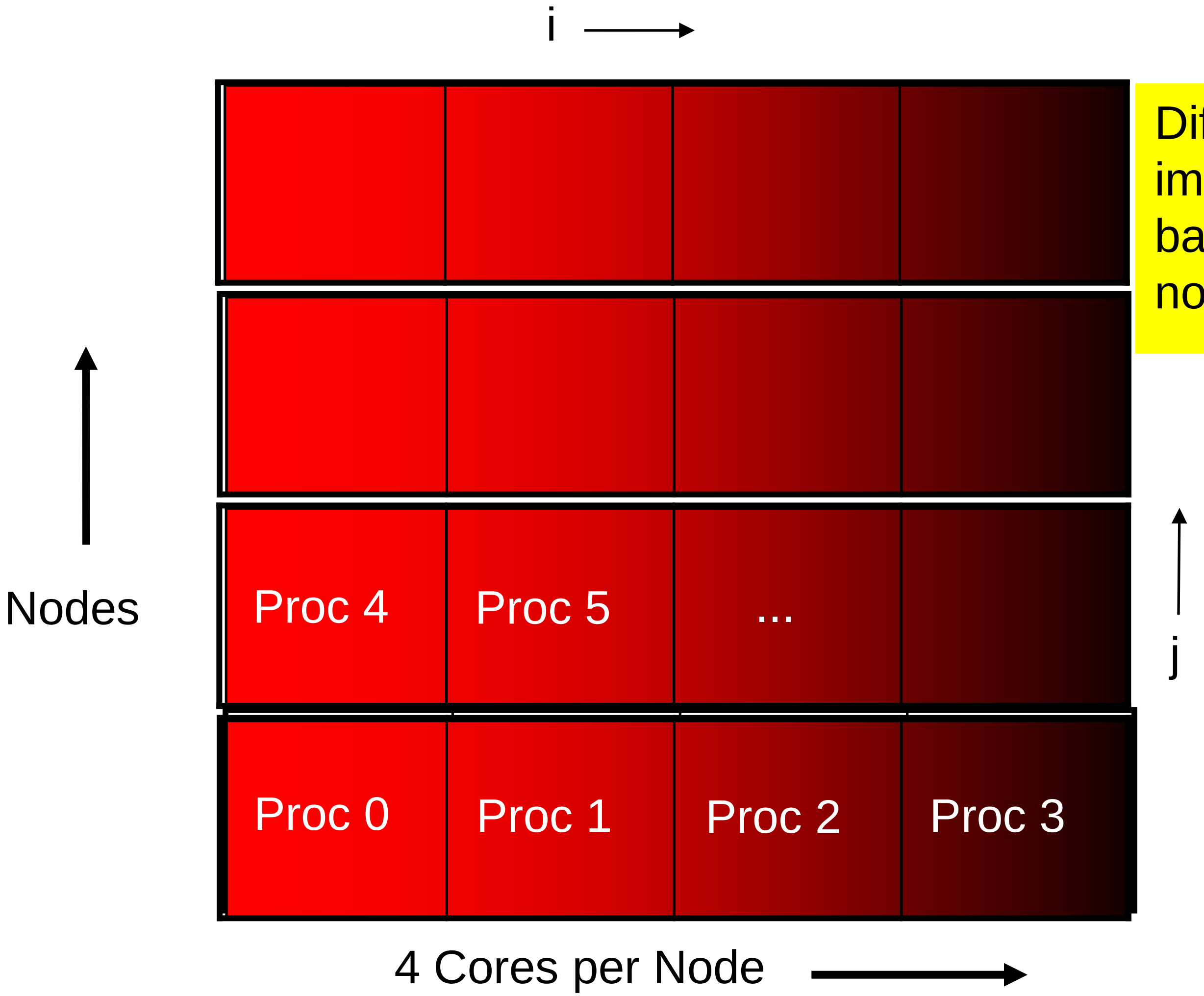
Traditional Cluster Computing (contd.)

- Hardware trends
  - Increasing number of cores per node - plus cores on co-processors
  - Increasing number of nodes per cluster
- Consequence: Increasing number of MPI processes per application
- Potential MPI limitations
  - Memory consumption per MPI process, sum exceeds the node memory
  - Limited scalability due to exhausted interconnects (e.g. MPI collectives)
  - Load balancing is often challenging in MPI

Hybrid Computing

- Combine MPI programming model with threading model
- Overcome MPI limitations by adding threading:
  - Potential memory gains in threaded code
  - Better scalability (e.g. less MPI communication)
  - Threading offers smart load balancing strategies
- Result: Maximize performance by exploitation of hardware (incl. co-processors)

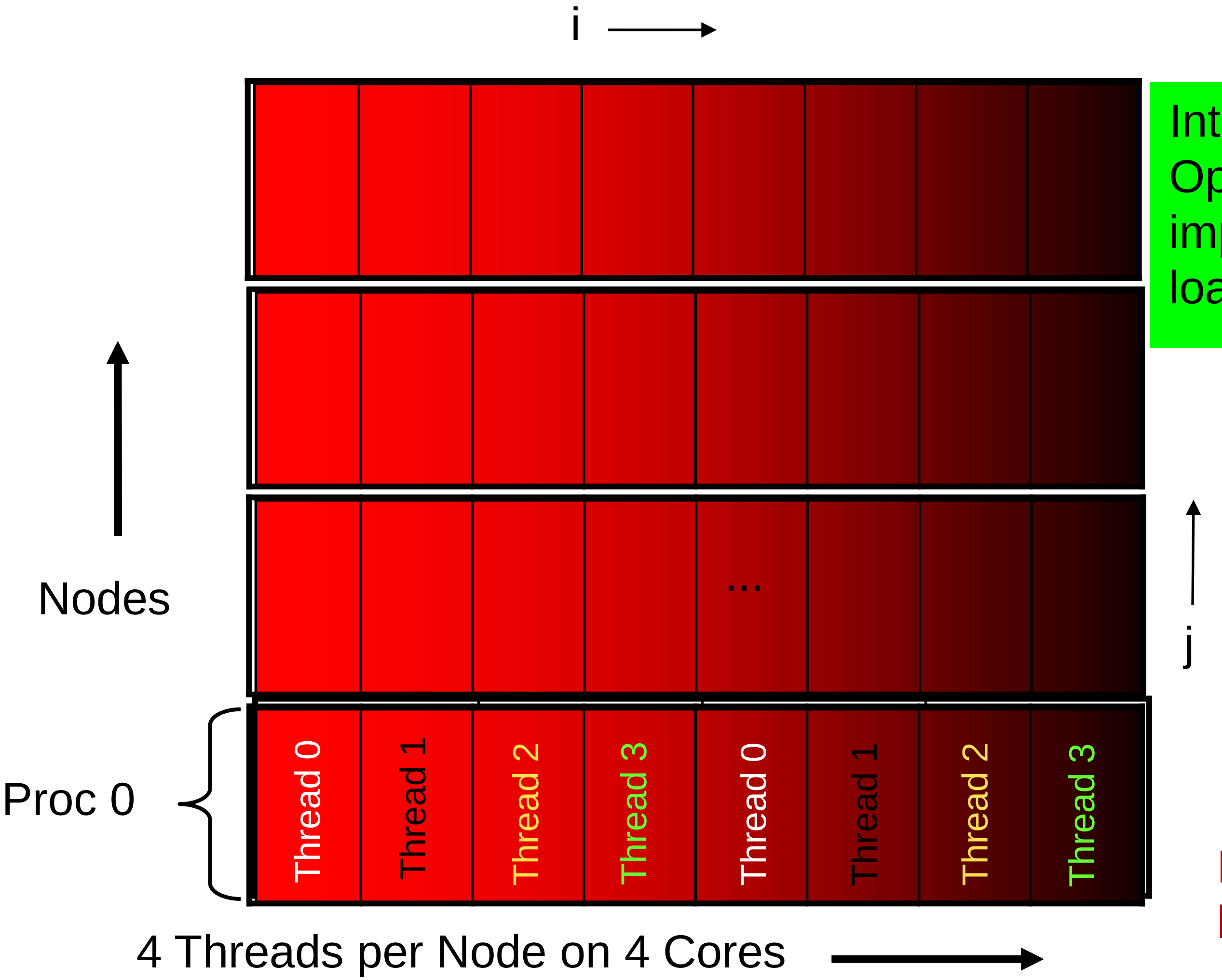
Example: MPI Load Imbalance



Difficult to implement load balancing in nodes with MPI

Dark red = high load

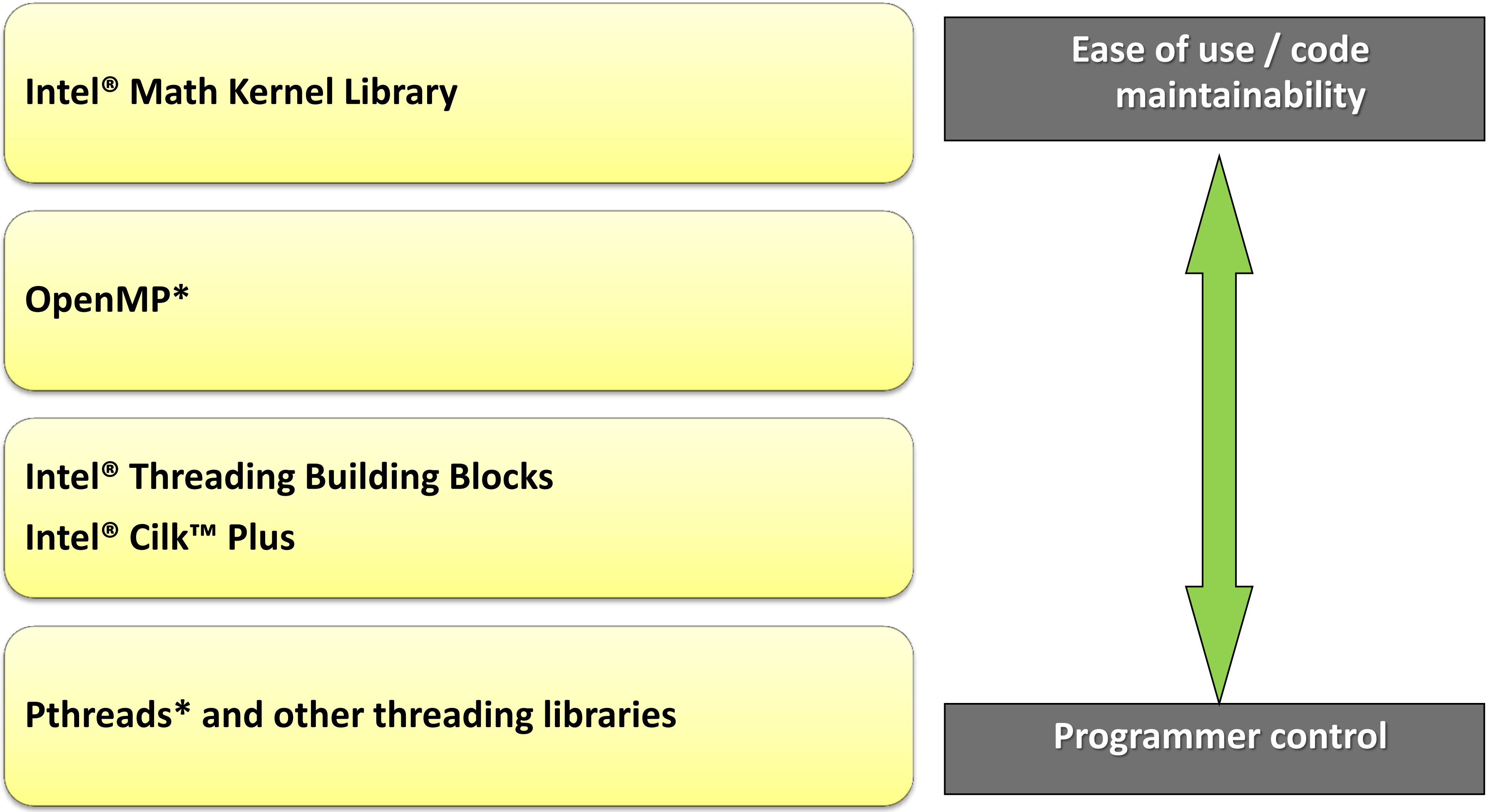
Example: Hybrid Load Balance



Interleaved OpenMP threads improve total load balancing

Dark red = high load

Options for Thread Parallelism



Intel® Math Kernel Library

OpenMP*

Intel® Threading Building Blocks

Intel® Cilk™ Plus

Pthreads* and other threading libraries

Ease of use / code maintainability

Programmer control

Choice of unified programming to target Intel® Xeon and Intel® Xeon Phi™!

Intel® MPI Support of Hybrid Codes

- Intel® MPI is strong in mapping control
- Sophisticated default or user controlled
  - `I_MPI_PIN_PROCESSOR_LIST` for pure MPI
  - For hybrid codes (default, takes precedence):

    `I_MPI_PIN_DOMAIN` =<*size*>*[:<layout>]*

    <size> =

`omp`	Adjust to OMP_NUM_THREADS
`auto`	#CPUs/#MPIprocs (default)
`<n>`	Number

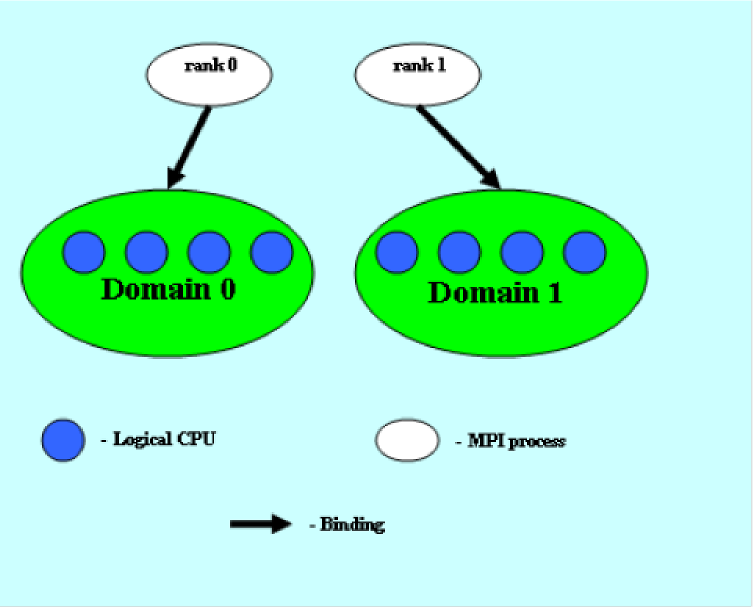
    <layout> =

`platform`	According to BIOS numbering
`compact`	Close to each other
`scatter`	Far away from each other

- Naturally extends to hybrid codes on Intel® Xeon Phi™

Intel® MPI Support of Hybrid Codes

- **Define `I_MPI_PIN_DOMAIN` to split logical processors into non-overlapping subsets**
- **Mapping rule: 1 MPI process per 1 domain**



Pin OpenMP threads inside the domain with `KMP_AFFINITY` (or in the code)

Intel® MPI Environment Support

- The execution command mpirun of Intel® MPI reads argument sets from the command line:
  - Sections between „`:`“ define an argument set (alternatively a line in a configfile specifies a set)
  - Host, number of nodes, but also environment can be set independently in each argument set

```
# mpirun -env I_MPI_PIN_DOMAIN 4 -host myXEON ...
        : -env I_MPI_PIN_DOMAIN 16 -host myMIC
```

- Adapt the important environment variables to the architecture
  - `OMP_NUM_THREADS`, `KMP_AFFINITY` for OpenMP
  - `CILK_NWORKERS` for Intel® Cilk$^{\text{TM}}$ Plus

Coprocessor-only and Symmetric Support

- **Full hybrid support on Intel® Xeon from Intel ® MPI extends to the Intel ® Xeon Phi™ coprocessor**
- `KMP_AFFINITY=balanced` **(only on the coprocessor) in addition to** `scatter` **and** `compact`
- `KMP_PLACE_THREADS=<n>Cx<m>T,<o>O` **(**`<n>`**-Cores times** `<m>`**-Threads with** `<o>`**-cores Offset, only on coprocessor) in addition to** `KMP_AFFINITY` **for exact but still generic thread placement**
- **Recommendations:**
  - **Explicitly control where MPI processes and threads run in a hybrid application (according to threading model)**
  - **Avoid splitting cores among MPI processes, i.e.** `I_MPI_PIN_DOMAIN` **should be a multiple of 4**
  - **Try different** `KMP_AFFINITY` **and/or** `KMP_PLACE_THREADS` **settings for your application**

OS Thread Affinity Mapping

- The Intel® Xeon Phi™ coprocessor has N cores, each with 4 hardware thread contexts, for a total of M=4*N threads
- The OS maps “procs” to the M hardware threads:

MIC core	0				1		…	(N-2)	(N-1)			
MIC HW thread	0	1	2	3	0	1	…	3	0	1	2	3
OS “proc”	1	2	3	4	5	6	…	(M-4)	0	(M-3)	(M-2)	(M-1)

- The OS runs on proc 0, which lives on core (N-1)!
  - Rule of thumb: Avoid using OS procs 0, (M-3), (M-2), and (M-1) to avoid contention with the OS
    - Only less than 2% resources unused (1/#cores)
  - Especially important when using the offload model due to data transfer activity!
  - But: Non-offload applications may slightly benefit from running on core (N-1)

OS Thread Affinity Mapping (ctd.)

- OpenMP library maps to the OS “procs”
- Examples (for non-offload apps which benefit from core N-1):
  - `KMP_AFFINITY=compact,granularity=thread,compact`

MIC core	0				1		…	(N-2)	(N-1)			
MIC HW thread	0	1	2	3	0	1	…	3	0	1	2	3
OS “proc”	1	2	3	4	5	6	…	(M-4)	0	(M-3)	(M-2)	(M-1)
OpenMP thread	0	1	2	3	4	5	…	(M-5)	(M-4)	(M-3)	(M-2)	(M-1)

  - `KMP_AFFINITY=balanced,granularity=thread`
    `OMP_NUM_THREADS=n=M/2`

MIC core	0				1		…	(N-2)	(N-1)			
MIC HW thread	0	1	2	3	0	1	…	3	0	1	2	3
OS “proc”	1	2	3	4	5	6	…	(M-4)	0	(M-3)	(M-2)	(M-1)
OpenMP thread	0	1			3	4	…		(n-2)	(n-1)		

OS Thread Affinity Mapping (ctd.)

- **Use `balanced` affinity to minimize False Sharing!**
  - `KMP_PLACE_THREADS=2Cx2T,0O`
  - **but still with implicit default mapping `scatter,granularity=thread`**

MIC core	0				1		…	(N-2)	(N-1)			
MIC HW thread	0	1	2	3	0	1	…	3	0	1	2	3
OS "proc"	1	2	3	4	5	6	…	(M-4)	0	(M-3)	(M-2)	(M-1)
OpenMP thread	0	2			1	3						

  - `KMP_PLACE_THREADS=2Cx2T,0O` **and** `KMP_AFFINITY=balanced`

MIC core	0				1		…	(N-2)	(N-1)			
MIC HW thread	0	1	2	3	0	1	…	3	0	1	2	3
OS "proc"	1	2	3	4	5	6	…	(M-4)	0	(M-3)	(M-2)	(M-1)
OpenMP thread	0	1			2	3						

Additional Resources

- Other documentation, presentations, and even a community forum can be found at
  - http://software.intel.com/mic-developer
  - http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

Let’s do a Trace Analyzer Lab!

http://software.intel.com/en-us/articles/intelr-xeon-phitm-advanced-workshop-labs

Contact

Vincent Betro
vbetro@tennessee.edu

Help
help@nics.utk.edu

OpenMP* 4.0 for HPC in a Nutshell

CUG 2014 · Lugano

Hans Pabst, May 5th 2014

[Slides by Dr. M. Klemm, Intel Corp.]

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS 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.

Optimization Notice

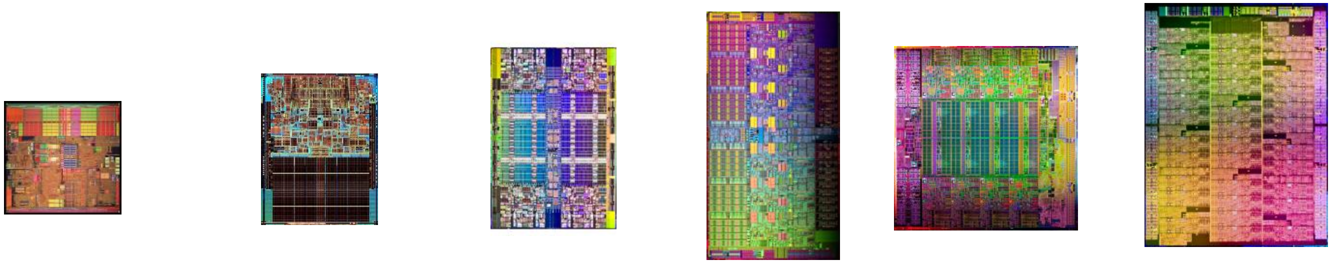
Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, 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

OpenMP API

- De-facto standard, OpenMP 4.0 out since July 2013
- API for C/C++ and Fortran for shared-memory parallel programming
- Based on directives (pragmas in C/C++)
- Portable across vendors and platforms
- Supports various types of parallelism

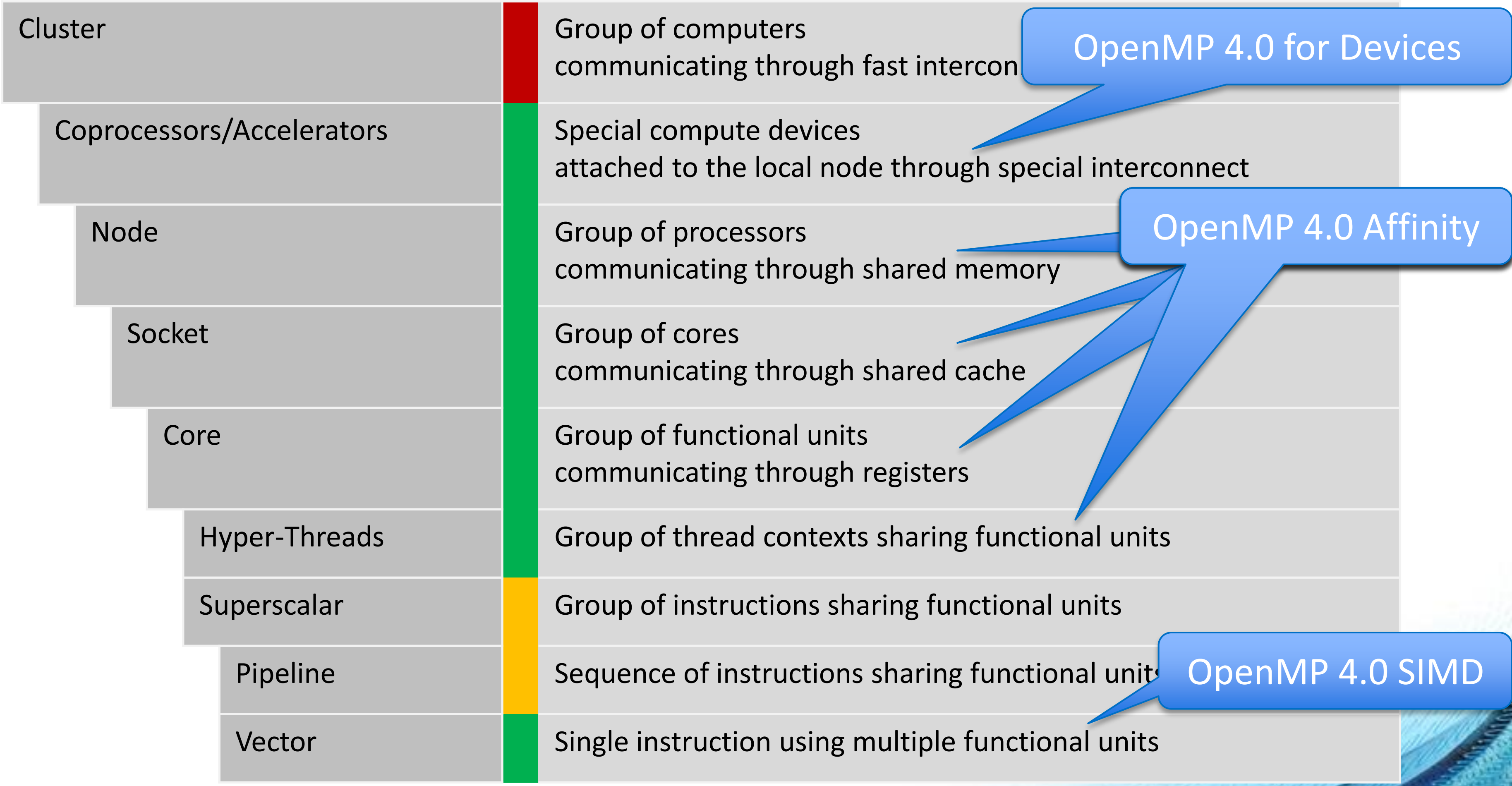
Evolution of Hardware (at Intel)



Images not intended to reflect actual die sizes

	64-bit Intel® Xeon® processor	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor E5-2600v2 series	Intel® Xeon Phi™ Co-processor 7120P
Frequency	3.6GHz	3.0GHz	3.2GHz	3.3GHz	2.7GHz	1.238MHz
Core(s)	1	2	4	6	12	61
Thread(s)	2	2	8	12	24	244
SIMD width	128 (2 clock)	128 (1 clock)	128 (1 clock)	128 (1 clock)	256 (1 clock)	512 (1 clock)

Levels of Parallelism in OpenMP 4.0

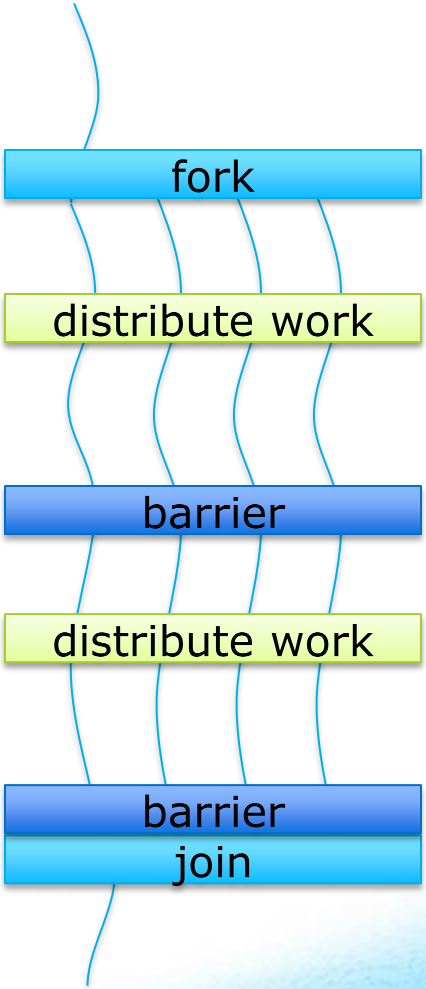


Cluster	🟥	Group of computers communicating through fast intercon
Coprocessors/Accelerators	🟩	Special compute devices attached to the local node through special interconnect
Node	🟩	Group of processors communicating through shared memory
Socket	🟩	Group of cores communicating through shared cache
Core	🟩	Group of functional units communicating through registers
Hyper-Threads	🟩	Group of thread contexts sharing functional units
Superscalar	🟨	Group of instructions sharing functional units
Pipeline	🟨	Sequence of instructions sharing functional unit
Vector	🟩	Single instruction using multiple functional units

SOFTWARE AND SERVICES

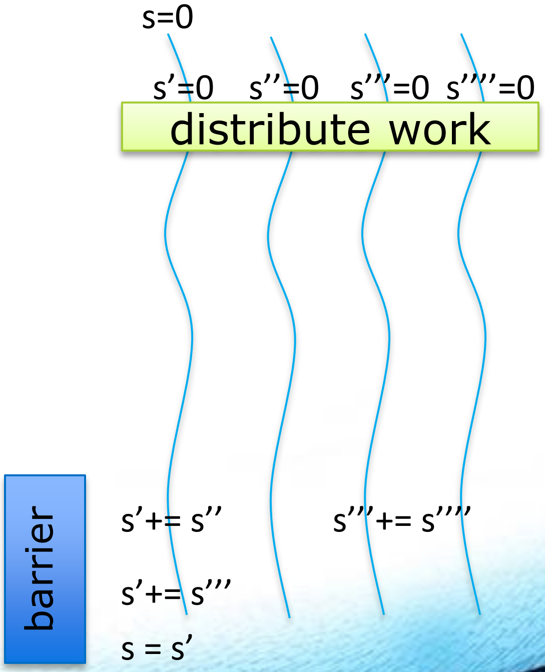
OpenMP Intro in Three Slides (1)

```
#pragma omp parallel
{
   #pragma omp for
   for (i = 0; i<N; i++)
   {…}

   #pragma omp for
   for (i = 0; i< N; i++)
   {…}
}
```



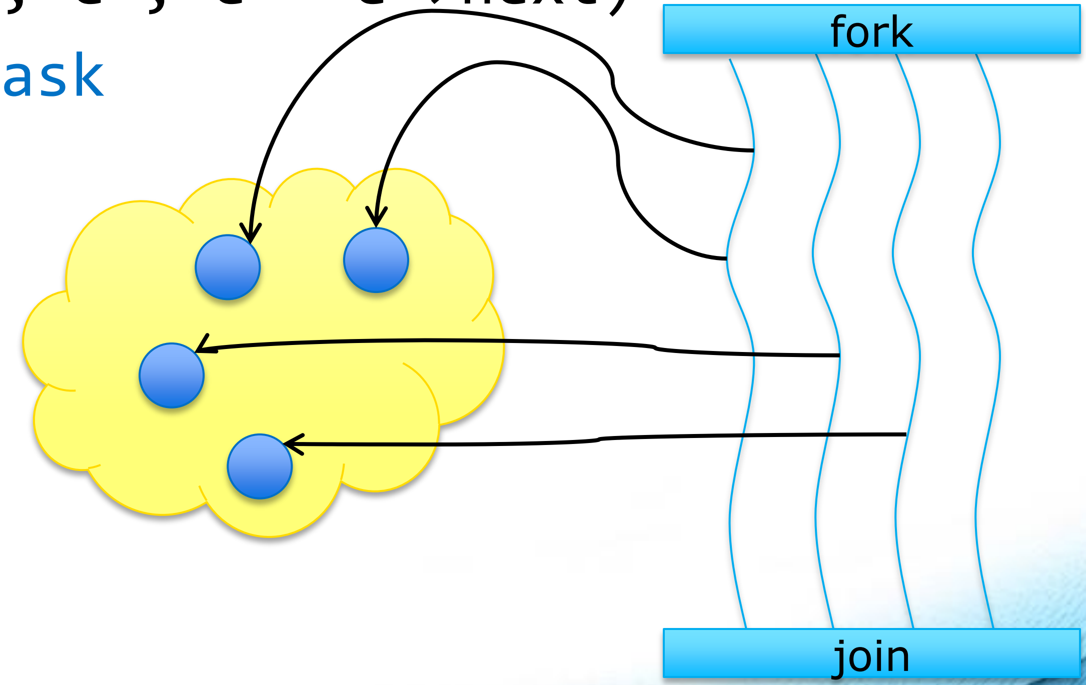
OpenMP Intro in Three Slides (2)

```
double a[N];
double l,s = 0;
#pragma omp parallel for reduction(+:s) private(l) \
                         schedule(static,4)


for (i = 0; i<N; i++)
{
   l = log(a[i]);
   s += l;
}
```



OpenMP Intro in Three Slides (3)

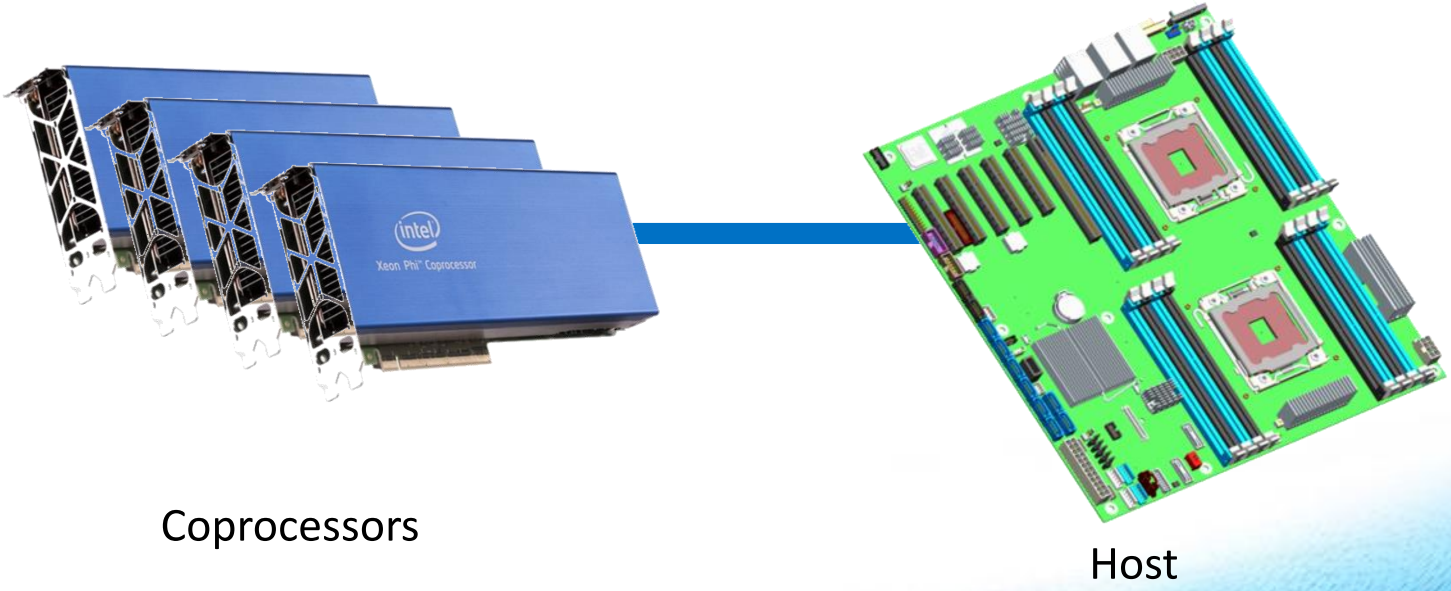
```
#pragma omp parallel
#pragma omp single
for(e = l->first; e ; e = e->next)
   #pragma omp task
     process(e);
```



OpenMP 4.0 for Devices

Device Model

- OpenMP 4.0 supports accelerators/coprocessors
- Device model:
  - One host
  - Multiple accelerators/coprocessors of the same kind



Coprocessors

Host

OpenMP 4.0 for Devices - Constructs

- Transfer control [and data] from the host to the device

- Syntax (C/C++)

```
#pragma omp target [data] [clause[[,] clause],…]
structured-block
```

- Syntax (Fortran)

```
!$omp target [data] [clause[[,] clause],…]
structured-block
!$omp end target [data]
```

- Clauses

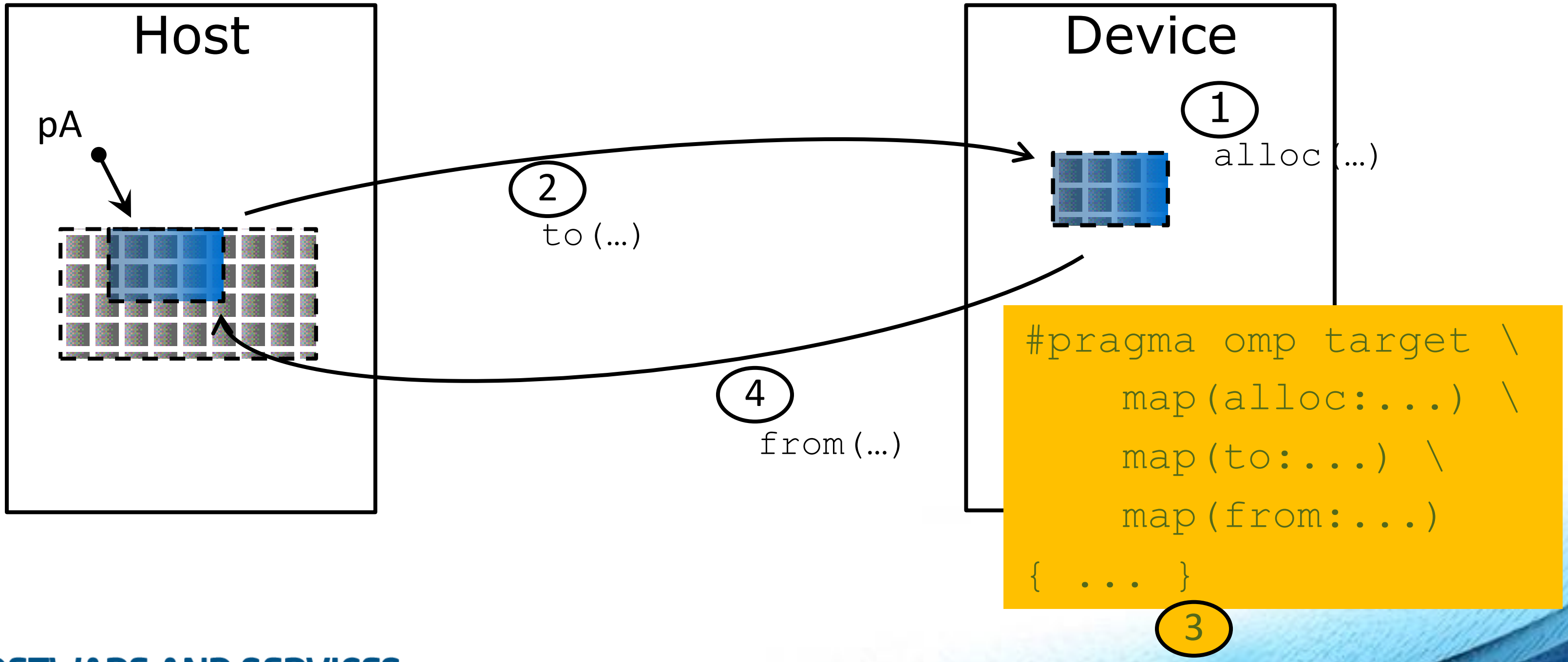
```
device(scalar-integer-expression)
map(alloc | to | from | tofrom: list)
if(scalar-expr)
```

Execution Model

- The `target construct` transfers the control flow to the target device
  - Transfer of control is sequential and synchronous
  - The transfer clauses control direction of data flow
  - Array notation is used to describe array length
- The `target data` construct creates a scoped device data environment
  - Does not include a transfer of control
  - The transfer clauses control direction of data flow
  - The device data environment is valid through the lifetime of the target data region
- Use `target update` to request data transfers from within a target data region

Execution Model

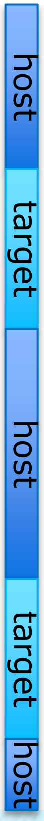
- Data environment is lexically scoped
  - Data environment is destroyed at closing curly brace
  - Allocated buffers/data are automatically released



```
#pragma omp target \
    map(alloc:...) \
    map(to:...) \
    map(from:...)
{ ... }
```

Example

```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N)) map(from:res)
 {
#pragma omp target device(0)
#pragma omp parallel for
  for (i=0; i<N; i++)
   tmp[i] = some_computation(input[i], i);

  update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])

#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
  for (i=0; i<N; i++)
   res += final_computation(input[i], tmp[i], i)
 }
```



`teams` Construct

- Support multi-level parallel devices

- Syntax (C/C++):

```
#pragma omp teams [clause[[,] clause],…]
structured-block
```

- Syntax (Fortran):

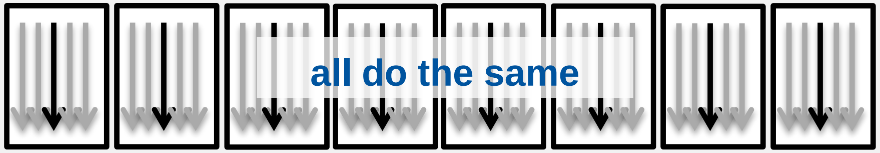
```
!$omp teams [clause[[,] clause],…]
structured-block
```

- Clauses

```
num_teams(integer-expression)
num_threads(integer-expression)
default(shared | none)
private(list), firstprivate(list)
shared(list), reduction(operator : list)
```

Offloading SAXPY to a Coprocessor

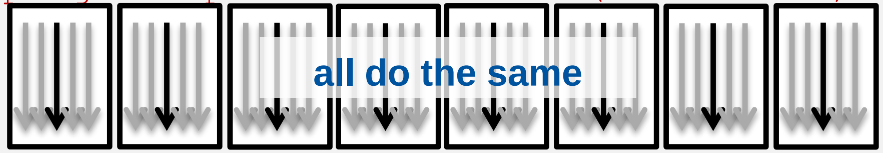
```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
  {
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(nthreads)
```



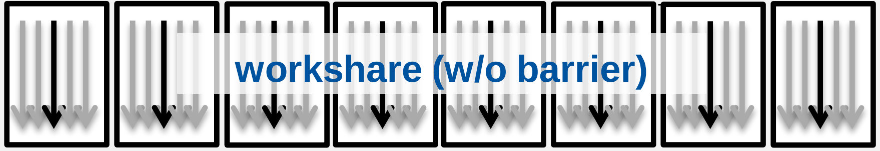
```
  for (int i = 0; i < n; i += num_blocks){
    for (int j = i; j < i + num_blocks; j++) {
        y[j] = a*x[j] + y[j];
  } }
  }
  free(x); free(y); return 0;
}
```

Offloading SAXPY to a Coprocessor

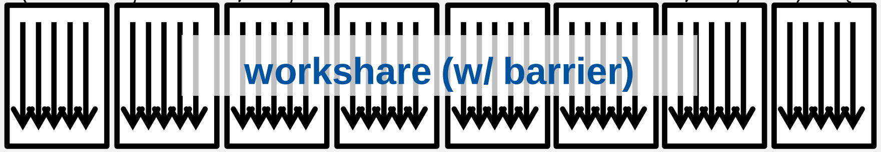
```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
```



```
#pragma omp distribute
  for (int i = 0; i < n; i += num_blocks){
```



```
#pragma omp parallel for
    for (int j = i; j < i + num_blocks; j++) {
```



```
        y[j] = a*x[j] + y[j];
  } }
} free(x); free(y); return 0; }
```

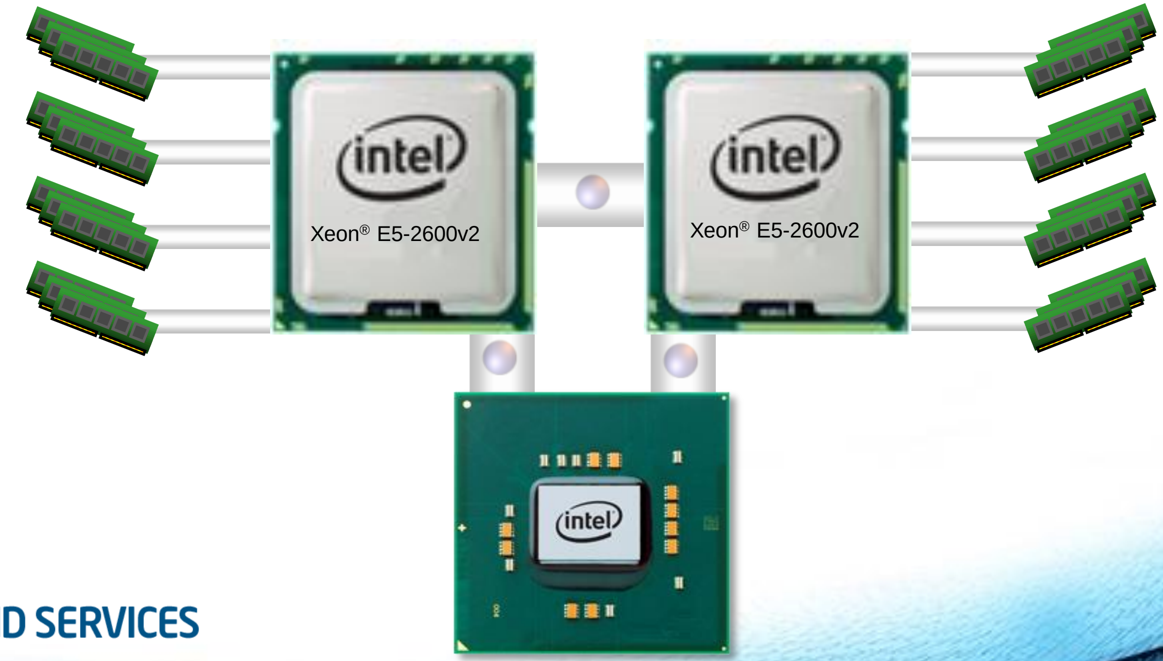
Offloading SAXPY to a Coprocessor

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target map(to:x[0:n]) map(tofrom:y)
  {
#pragma omp teams distribute parallel for \
       num_teams(num_blocks) num_threads(bsize)
    for (int i = 0; i < n; ++i){
      y[i] = a*x[i] + y[i];
    }
  }

  free(x); free(y); return 0;
}
```

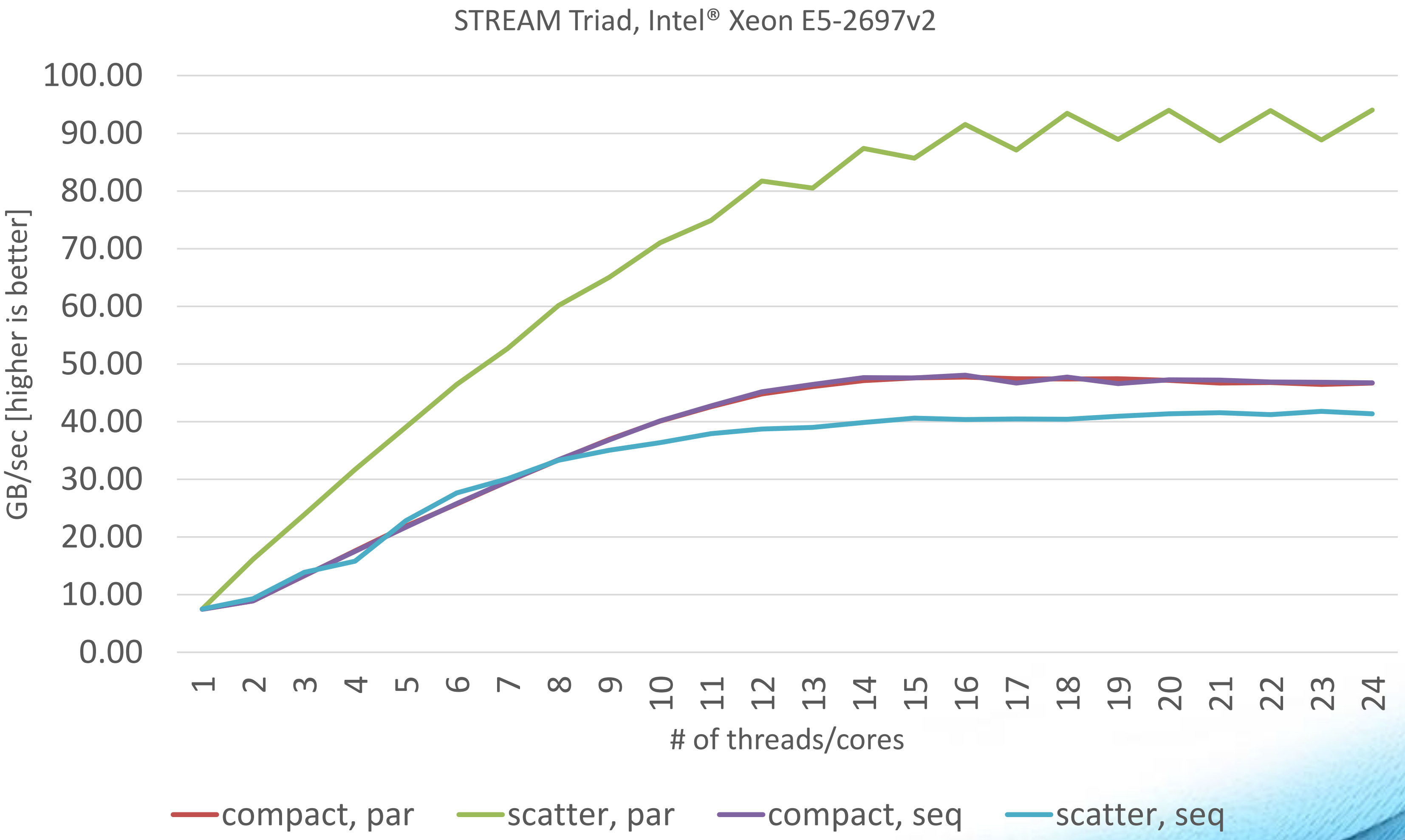
OpenMP 4.0 Affinity

NUMA is here to Stay…

- (Almost) all multi-socket compute servers are NUMA systems
    - Different access latencies for different memory locations
    - Different bandwidth observed for different memory locations
- Example: Intel® Xeon E5-2600v2 Series processor



Thread Affinity – Why It Matters?



SOFTWARE AND SERVICES

Thread Affinity – Processor Binding

Binding strategies depends on machine and the app

- Putting threads far, i.e. on different packages
  - (May) improve the aggregated memory bandwidth
  - (May) improve the combined cache size
  - (May) decrease performance of synchronization constructs
- Putting threads close together, i.e. on two adjacent cores which possible share the cache
  - (May) improve performance of synchronization constructs
  - (May) decrease the available memory bandwidth and cache size (per thread)

Thread Affinity in OpenMP* 4.0

- OpenMP 4.0 introduces the concept of places…
  - set of threads running on one or more processors
  - can be defined by the user
  - pre-defined places available:
    - threads one place per hyper-thread
    - cores one place exists per physical core
    - sockets one place per processor package

… and affinity policies…

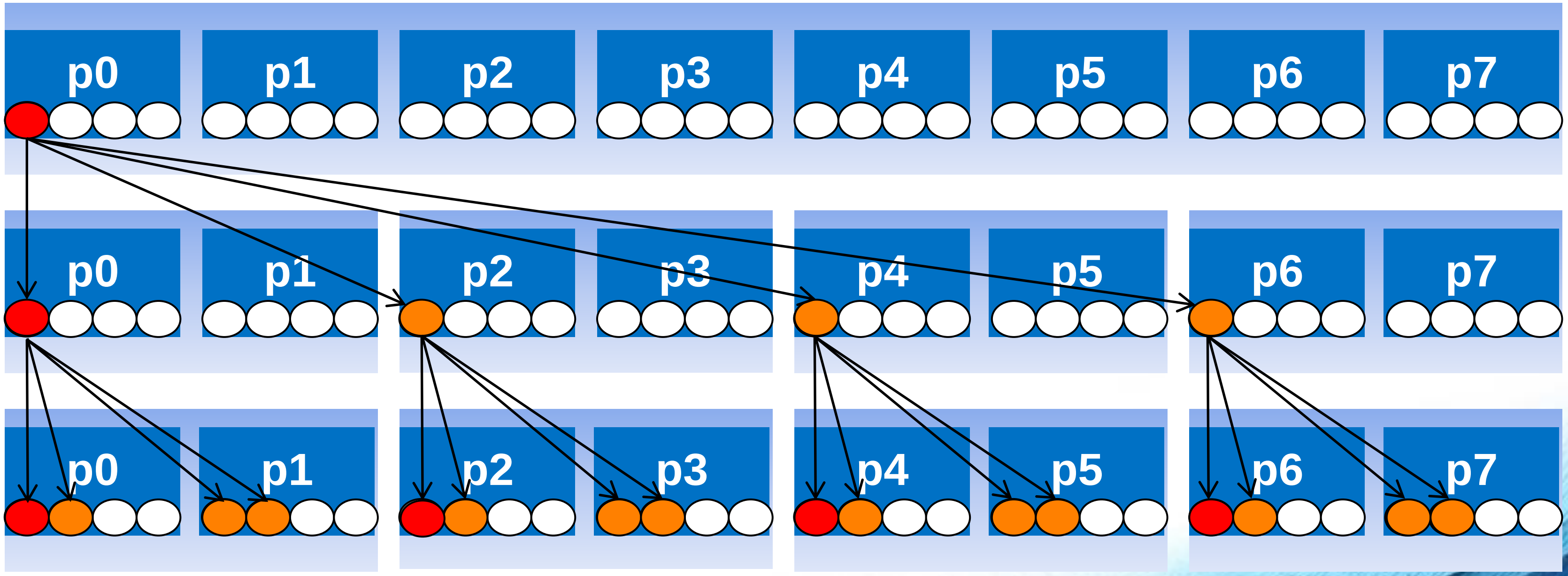
  - spread spread OpenMP threads evenly among the places
  - close pack OpenMP threads near master thread
  - master collocate OpenMP thread with master thread

- … and means to control these settings
  - Environment variables OMP_PLACES and OMP_PROC_BIND
  - clause `proc_bind` for parallel regions

Thread Affinity Example

- Example (Intel® Xeon Phi™ Coprocessor):
  Distribute outer region, keep inner regions close

```
OMP_PLACES=cores(8); OMP_NUM_THREADS=4,4

#pragma omp parallel proc_bind(spread)
   #pragma omp parallel proc_bind(close)
```

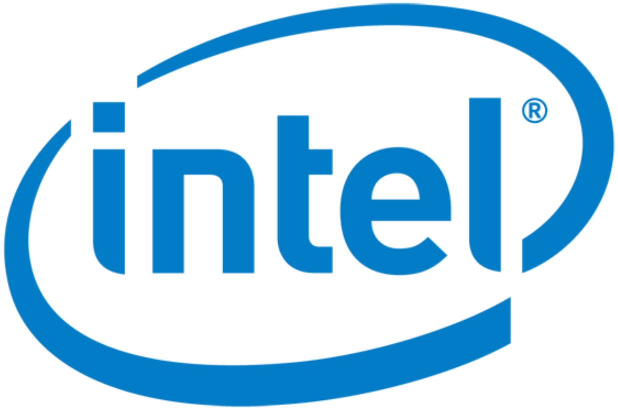


We’re Almost Through

- OpenMP 4.0 is a major leap for OpenMP
  - New kind of parallelism has been introduced
  - Support for heterogeneous systems with coprocessor devices
- OpenMP 4.0 has more to offer!
  - Improved Fortran 2003 support
  - User-defined reductions
  - Task dependencies
  - Cancellation
- Video series including this content
  - http://software.intel.com/en-us/videos/part-1-of-5-openmp-40-for-simd-and-affinity-features-with-intel-xeon-processors-and-intel

The last Slide…

- OpenMP 4.0 support in Intel® Compiler
  - Introduced Intel® Composer XE 2013 SP1
    - SIMD Constructs (except combined constructs)
    - OpenMP for devices (except combined constructs)
    - OpenMP Affinity
  - Feature-complete in Intel® Composer XE 2015
- Intel® Software Development Tools 2015 Beta
  - Try out: http://bit.ly/sw-dev-tools-2015-beta


intel®

Optimization Techniques for Implicit and Explicit Vectorization

CUG 2014, Lugano, Switzerland

What is SIMD?

Scalar Code

- Executes code one element at a time.

Vector Code

- Executes code multiple elements at a time.
- Single Instruction Multiple Data.

[Scalar] 1 elem at a time

```
addss xmm1, xmm2
```

[SSE] 4 elems at a time

```
addps xmm1, xmm2
```

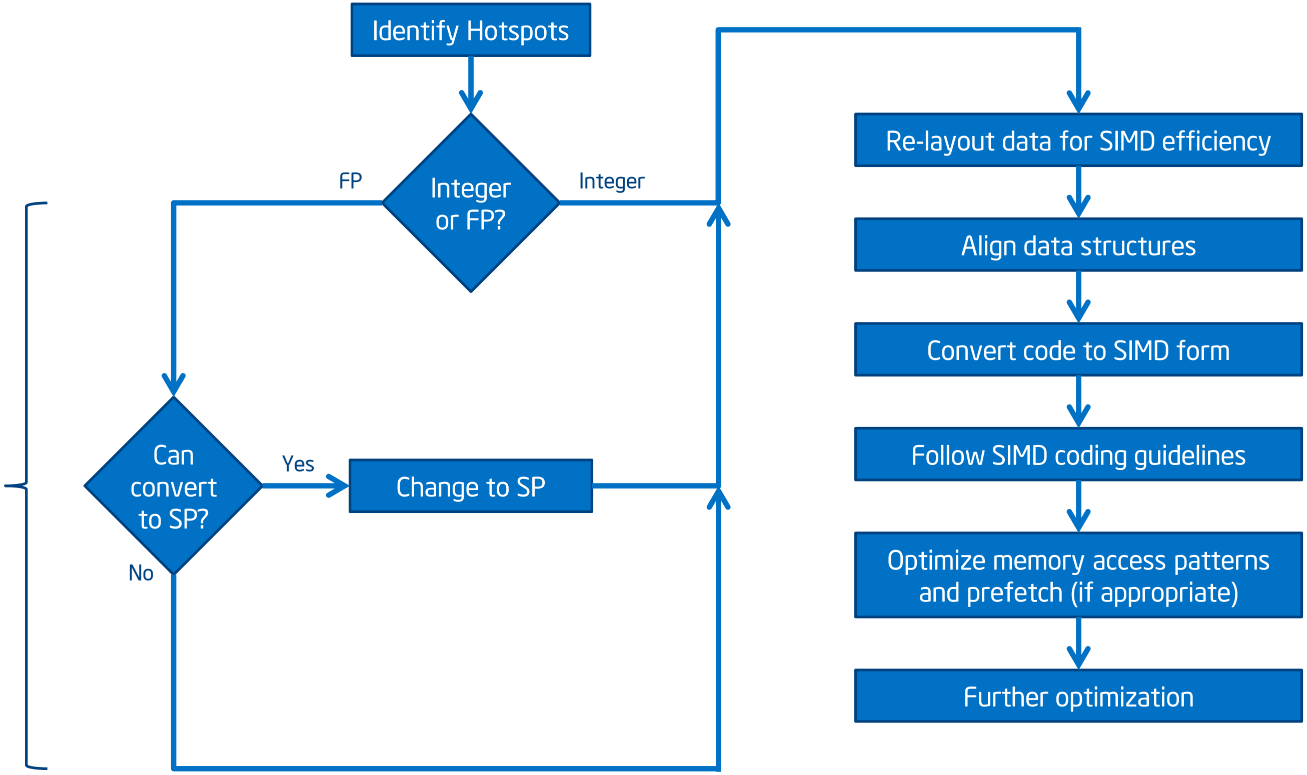
[AVX] 8 elems at a time

```
vaddps ymm1, ymm2, ymm3
```

[MIC / AVX-512] 16 elems at a time

```
vaddps zmm1, zmm2, zmm3
```

Preparing Code for SIMD



Identify Hotspots

Integer or FP?

FP

Integer

Can convert to SP?

Yes

No

Change to SP

Re-layout data for SIMD efficiency

Align data structures

Convert code to SIMD form

Follow SIMD coding guidelines

Optimize memory access patterns and prefetch (if appropriate)

Further optimization

Precision is important: impacts the SIMD width.

Session Plan

Hands-off

- Data Layout and Alignment
- Implicit Vectorization
- Explicit Vectorization
- SIMD Intrinsics

Hands-on

- Explicit Vectorization with OpenMP* 4.0

Session Plan

Hands-off

- Data Layout and Alignment
- Implicit Vectorization
- Explicit Vectorization
- SIMD Intrinsics

Hands-on

- Explicit Vectorization with OpenMP* 4.0

Data Layout – Why It's Important

Instruction-Level

- Hardware is optimized for contiguous loads/stores.
- Support for non-contiguous accesses differs with hardware. (e.g. AVX2/KNC gather)

Memory-Level

- Contiguous memory accesses are cache-friendly.
- Number of memory streams can place pressure on prefetchers.

Data Layout – Common Layouts

Array-of-Structs (AoS)

x	y	z	x	y	z
x	y	z	x	y	z
x	y	z	x	y	z

- Pros:
  Good locality of {x, y, z}.
  1 memory stream.
- Cons:
  Potential for gather/scatter.

Struct-of-Arrays (SoA)

x	x	x	x	x	x
y	y	y	y	y	y
z	z	z	z	z	z

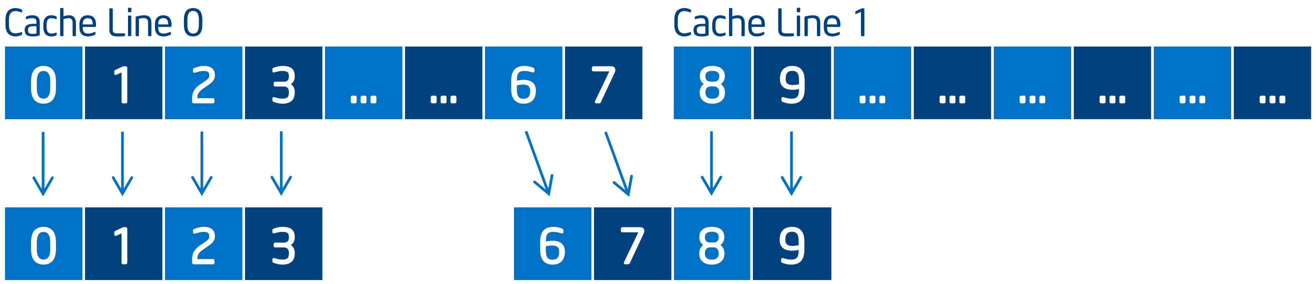
- Pros:
  Contiguous load/store.
- Cons:
  Poor locality of {x, y, z}.
  3 memory streams.

Hybrid (AoSoA)

x	x	y	y	z	z
x	x	y	y	z	z
x	x	y	y	z	z

- Pros:
  Contiguous load/store.
  1 memory stream.
- Cons:
  Not a “normal” layout.

Data Alignment – Why It's Important



Aligned Load

- Address is aligned.
- One cache line.
- One instruction.

Unaligned Load

- Address is not aligned.
- Potentially multiple cache lines.
- Potentially multiple instructions.

Data Alignment – Why It's Important

Cache Associativity

- L1 and L2 on Knights Corner are 8-way associative.
  - L1: 8 cache lines that are 32KB/8 = 4KB apart.
  - L2: 8 cache lines that are 512KB/8 = 64KB apart.

Set Conflicts

- Occur when references are a multiple of 4K (L1) or 64K (L2) apart.
- Look for high cache miss rate, even though working set < cache capacity.
- Solution is to pad arrays appropriately.

Data Alignment – Sample Applications

1) Align Memory

- _mm_malloc(bytes, 64) / !dir$ attributes align:64

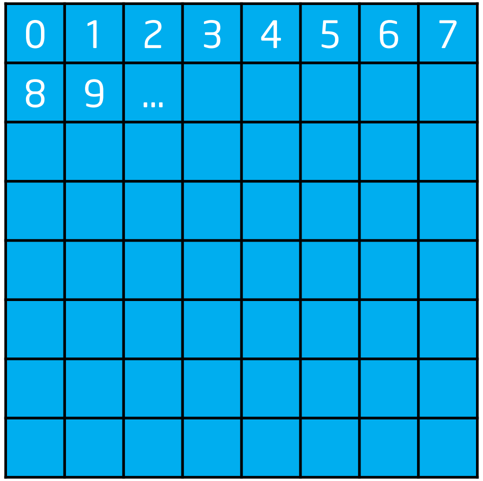
2) Access Memory in an Aligned Way

- for (i = 0; i < N; i++) { array[i] … }

3) Tell the Compiler

- #pragma vector aligned / !dir$ vector aligned
- __assume_aligned(p, 16) / !dir$ assume_aligned (p, 16)
- __assume(i % 16 == 0) / !dir$ assume (mod(i, 16) .eq. 0)

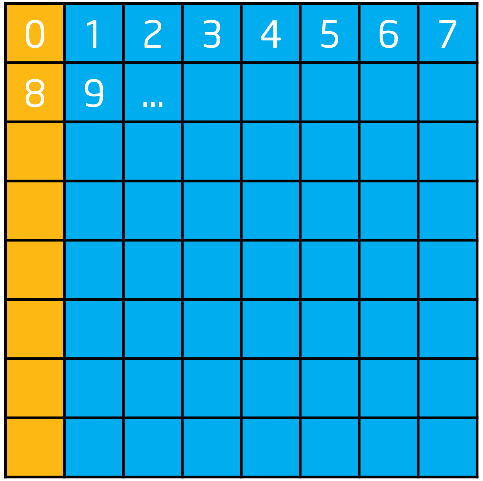
Data Alignment – Real-life Applications



0	1	2	3	4	5	6	7
8	9	…					

🟦 Data

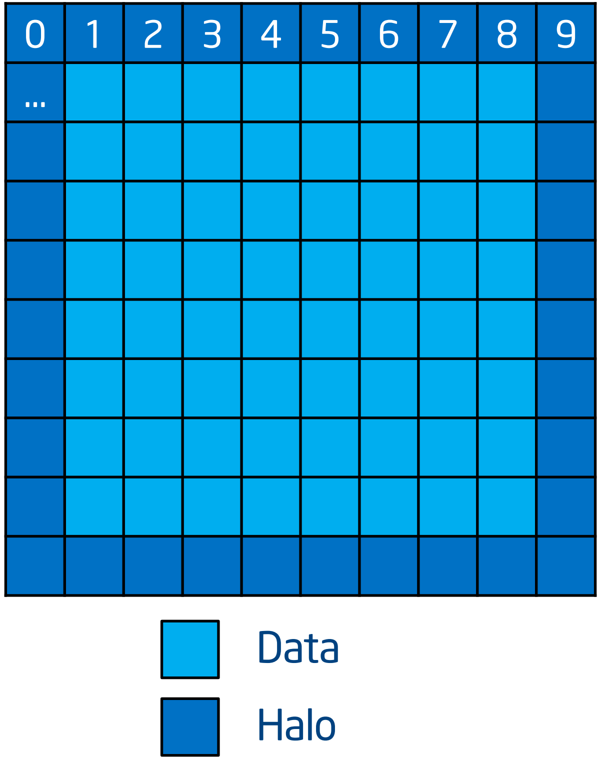
Data Alignment – Real-life Applications



0	1	2	3	4	5	6	7
8	9	…					

🟦 Data

Data Alignment – Real-life Applications

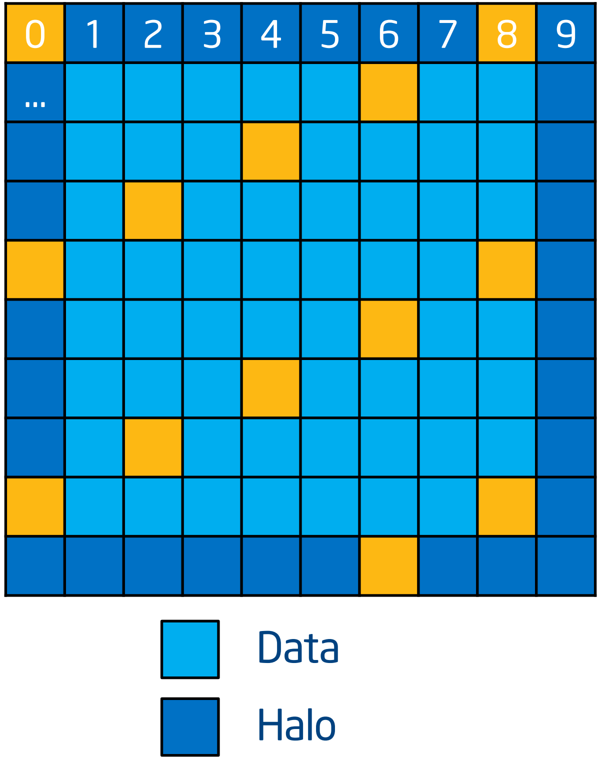


0	1	2	3	4	5	6	7	8	9
... ■	□	□	□	□	□	□	□	□	■
■	□	□	□	□	□	□	□	□	■
■	□	□	□	□	□	□	□	□	■
■	□	□	□	□	□	□	□	□	■
■	□	□	□	□	□	□	□	□	■
■	□	□	□	□	□	□	□	□	■
■	□	□	□	□	□	□	□	□	■
■	□	□	□	□	□	□	□	□	■
■	■	■	■	■	■	■	■	■	■

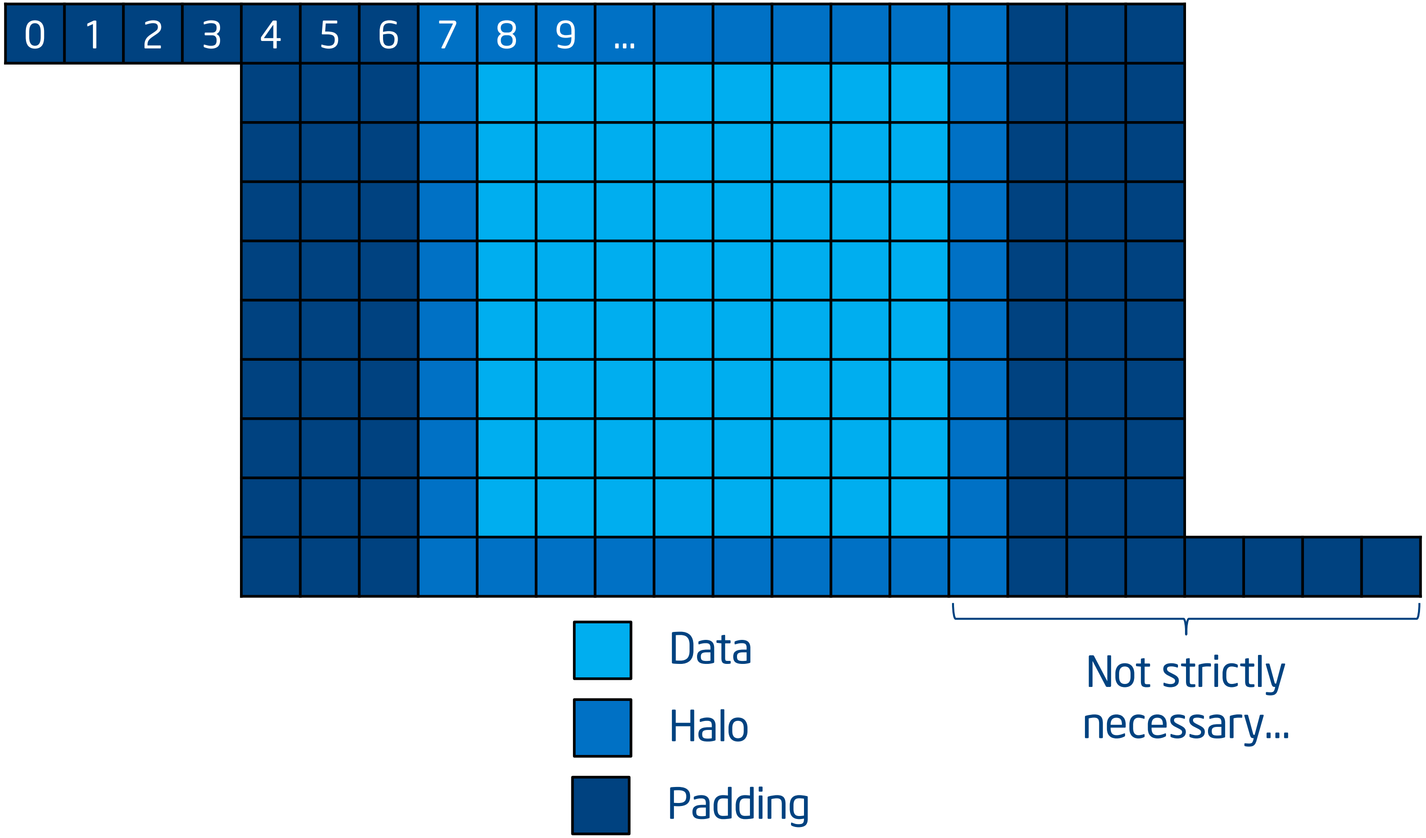
□ Data

■ Halo

Data Alignment – Real-life Applications



Data Alignment – Real-life Applications



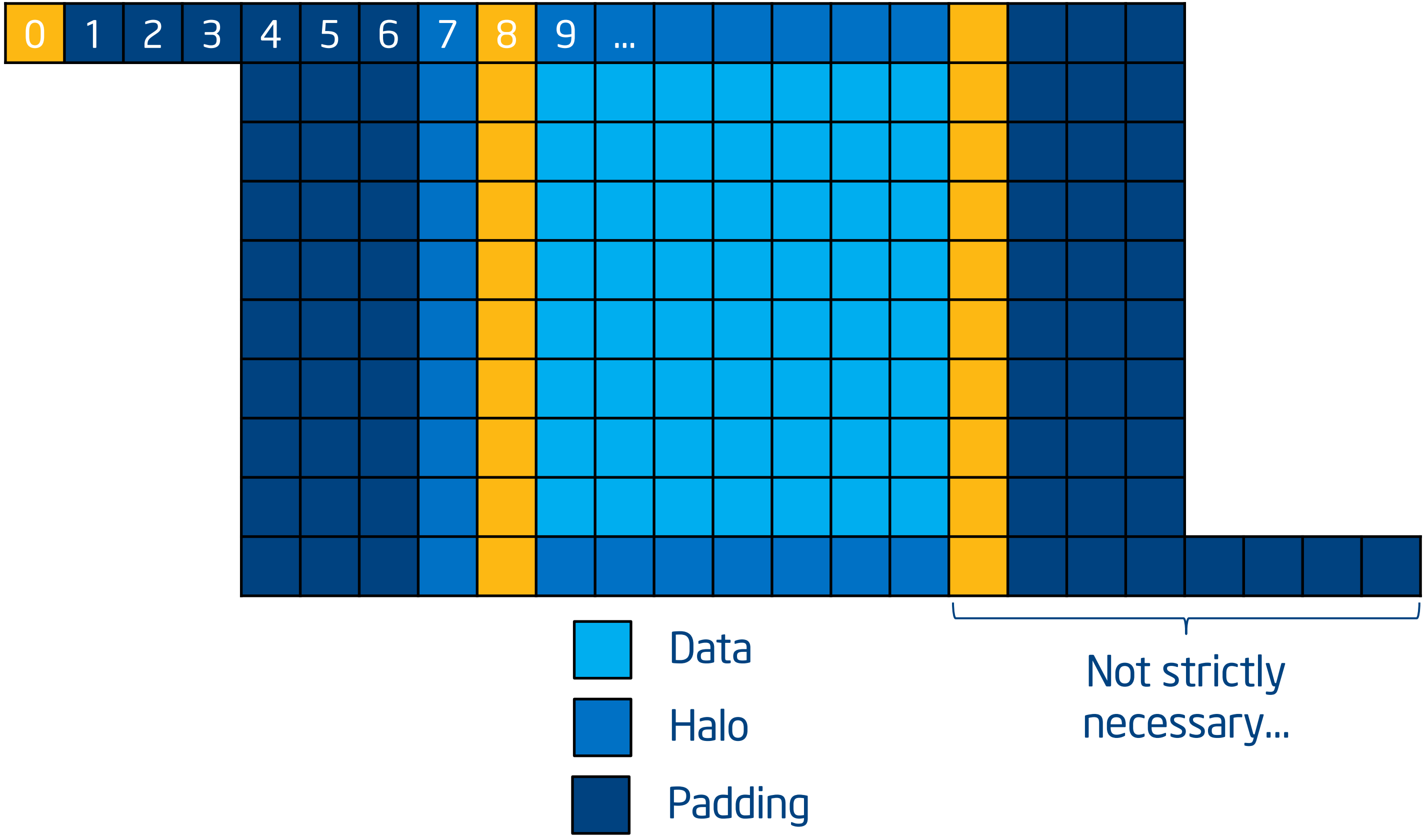
Data

Halo

Padding

Not strictly necessary…

Data Alignment – Real-life Applications



Not strictly necessary…

Session Plan

Hands-off

- Data Layout and Alignment
- Implicit Vectorization
- Explicit Vectorization
- SIMD Intrinsics

Hands-on

- Explicit Vectorization with OpenMP* 4.0

Implicit Vectorization

- Very powerful, but a compiler cannot make unsafe assumptions.

```
int* g_size;

void not_vectorizable
(float* a, float* b, float* c, int* ind) {
      for (int i = 0; i < *g_size; i++) {
            int j = ind[i];
            c[j] += a[i] + b[i];
      }
}
```

- Unsafe Assumptions:
  - a, b and c point to different arrays.
  - Value of global g_size is loop-invariant.
  - ind[i] is a one-to-one mapping.

Implicit Vectorization

- Very powerful, but a compiler cannot make unsafe assumptions.

```
int* g_size;

void vectorizable
(float* restrict a, float* restrict b, float* restrict c, int* restrict ind) {
      int size = *g_size;
      #pragma ivdep
      for (int i = 0; i < size; i++) {
            int j = ind[i];
            c[j] += a[i] + b[i];
      }
}
```

- Safe Assumptions:
  - a, b and c point to different arrays. (restrict)
  - Value of global g_size is loop-invariant. (pointer dereference outside loop)
  - ind[i] is a one-to-one mapping. (#pragma ivdep)

Implicit Vectorization – Improving Performance

Getting code to vectorize is only half the battle

- "LOOP WAS VECTORIZED" != "the code is optimal"
- Vectorized code can be slower than the scalar equivalent.

Compiler will <u>always</u> choose correctness over performance

- "Hints" and pragmas can't possibly cover all the situations…
- … but we can usually rewrite loop bodies to assist the compiler.

Implicit Vectorization – Common Code Transformations

Pattern:	Error checking within a vectorizable loop.
Issue:	Number of loop iterations unknown at compile time.
Original:	if (error condition) { exit }
Transform:	if (error condition) { error = true; } … if (error) exit

Pattern:	Memory access guarded by conditional.
Issue:	Compiler cannot assume memory is safe to access.
Original:	if (condition) a[i] += result;
Transform:	a[i] += (condition) ? result : 0;

Implicit Vectorization – Common Code Transformations

Pattern: Loading neighbour values, with a branch for boundary conditions.

Issue(s): Compiler may generate a gather; array[position-1] may be unsafe.

Original: double left = (column > 0) ? array[position -1] : boundary

Transform: Pad array with appropriate halo data: avoids branch and ensures load is safe.

Pattern: Loop contains OpenMP atomics, intrinsics, inline assembly

Issue: Compiler cannot vectorize these things.

Original:

```
for (…) {
  // vectorizable
  // non-vectorizable
}
```

Transform: Separate vectorizable and non-vectorizable code into two loops.

Session Plan

Hands-off

- Data Layout and Alignment
- Implicit Vectorization
- Explicit Vectorization
- SIMD Intrinsics

Hands-on

- Explicit Vectorization with OpenMP* 4.0

Explicit Vectorization

Compiler Responsibilities

- Allow programmer to declare that code can and should be run in SIMD.
- Generate the code the programmer asked for.

Programmer Responsibilities

- Correctness (e.g. no dependencies, no invalid memory accesses).
- Efficiency (e.g. alignment, loop order, masking).

Explicit Vectorization – Motivating Example 1

```
float sum = 0.0f;
float *p = a;
int step = 4;

#pragma omp simd reduction(+:sum) linear(p:step)
for (int i = 0; i < N; ++i) {
        sum += *p;
        p += step;
}
```

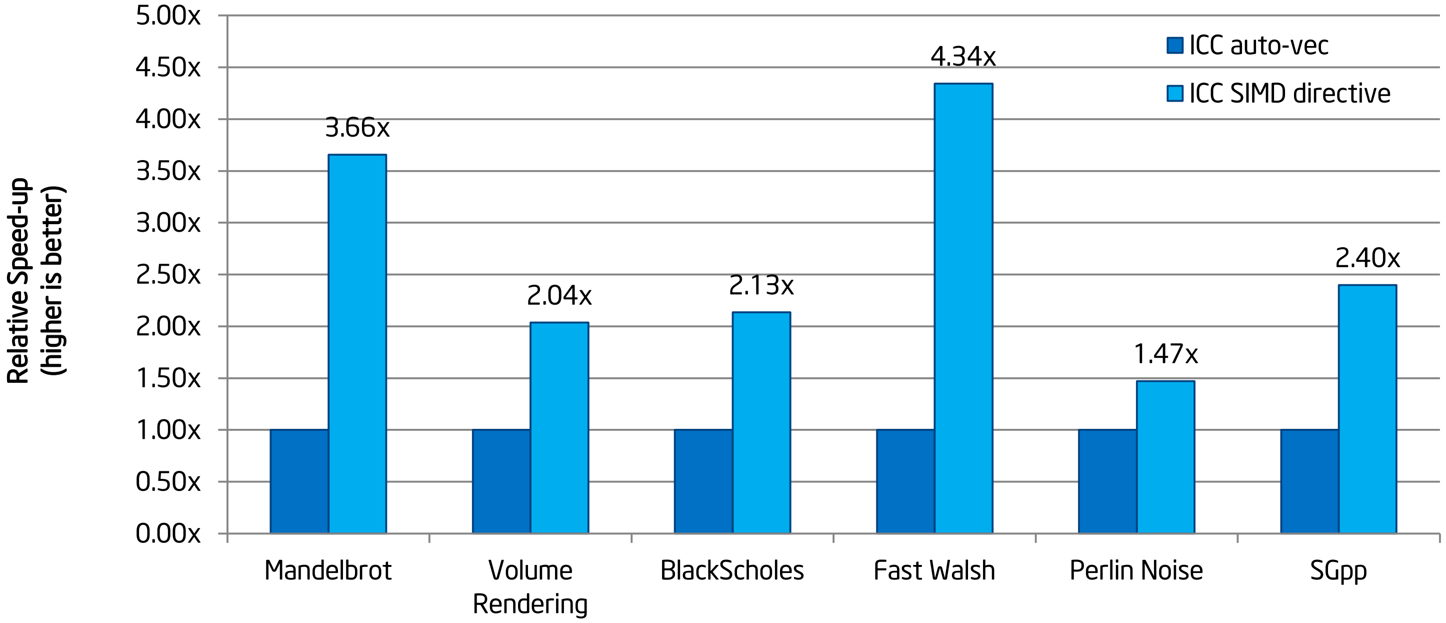
- The two += operators have different meaning from each other.
- The programmer should be able to express those differently.
- The compiler has to generate different code.
- The variables *i*, *p* and *step* have different “meaning” from each other.

Explicit Vectorization – Motivating Example 2

```
#pragma omp declare simd simdlen(16)
uint32_t mandel(fcomplex c)
{
    uint32_t count = 1; fcomplex z = c;
    for (int32_t i = 0; i < max_iter; i += 1) {
        z = z * z + c;
        int t = cabsf(z) < 2.0f;
        count += t;
        if (!t) { break;}
    }
    return count;
}
```

- mandel() function is called from a loop over X/Y points.
- We would like to vectorize that outer loop.
- Compiler creates a vectorized function that acts on a vector of 16 c values.

Explicit Vectorization – Performance Impact



M. Klemm, A. Duran, X. Tian, H. Saito, D. Caballero, and X. Martorell, “Extending OpenMP with Vector Constructs for Modern Multicore SIMD Architectures. In Proc. of the Intl. Workshop on OpenMP”, pages 59-72, Rome, Italy, June 2012. LNCS 7312.

Explicit Vectorization – Array Notation

"Long Form"

```
C[0:N] = A[0:N] + B[0:N];
D[0:N] = C[0:N] * C[0:N];
```

"Short Form"

```
for (i = 0; i < N; i += V) {
    C[i:V] = A[i:V] + B[i:V];
    D[i:V] = C[i:V] * C[i:V];
}
```

- Long form is more elegant, but short form is better for performance.
- Imagine each array assignment as a for loop:
  - For large N, long form will not keep C[0:N] in cache.
  - For appropriate V, short form keeps C[i:V] in registers.
- Some differences between Intel® Cilk™ Plus notation (C/C++) and Fortran 90.

Explicit Vectorization – OpenMP* SIMD Loops

#pragma omp simd / !$omp simd => for/do loop is a SIMD loop.

safelen (*length*)	Maximum distance between two iterations executed concurrently by a SIMD instruction.
linear (*list[:linear-step]*)	List items are private and have a linear relationship with respect to the iteration space.
aligned (*list[:alignment]*)	List items are aligned to a platform-dependent value (or the value of the optional parameter).

private (*list*), lastprivate (*list*), reduction (*reduction-identifier:list*) and collapse (*n*) are also supported, with functionality matching that of omp for.

See: OpenMP 4.0 Specification http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

Explicit Vectorization – OpenMP* SIMD Functions

#pragma omp declare simd / !$omp declare simd => function will be called from a SIMD loop.

simdlen (*length*)	Maximum number of concurrent arguments to the function (i.e. maximum SIMD width).
uniform (*argument-list*)	List items have the same value for all SIMD lanes, and can therefore be broadcast.
inbranch notinbranch	Function always called inside a conditional. Function never called inside a conditional.

linear (*argument-list[:linear-step]*) and aligned (*argument-list[:alignment]*) are also supported, with functionality matching that of omp simd.

See: OpenMP 4.0 Specification http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

Implicit vs Explicit Vectorization – Important Differences

Implicit

- Automatic dependency analysis.
  (e.g. recognises SIMD reductions)
- Recognizes idioms with data dependencies.
  (e.g. array[i++] = x; -> vcompress)
- Non-inline functions will be scalarized.
- Limited support for outer-loop vectorization (only with –O3).

Explicit

- No dependency analysis.
  (e.g. SIMD reductions must be declared)
- Recognizes idioms without data dependencies.
- Non-inline functions can be vectorized.
- Outer loops can be vectorized.

Session Plan

Hands-off

- Data Layout and Alignment
- Implicit Vectorization
- Explicit Vectorization
- SIMD Intrinsics

Hands-on

- Explicit Vectorization with OpenMP* 4.0

SIMD Intrinsics – An Introduction

Intrinsic Functions

- Substituted by compiler for specific instruction(s).
- Higher level of abstraction than assembly.

SIMD Intrinsics

- Explicit vectorization at an instruction level.
- Direct manipulation of SIMD and mask registers.

```
// Scalar code for vector add.
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}

// SIMD intrinsics for vector add.
for (int i = 0; i < N; i += 8) {
    __m512 ai = _mm512_load_pd(&a[i]);
    __m512 bi = _mm512_load_pd(&b[i]);
    __m512 ci = _mm512_add_pd(ai, bi);
    _mm512_store_pd(&c[i], ci);
}
```

SIMD Intrinsics – When and How?

When to Use Intrinsics

- You think you can beat the compiler (after asm inspection).
- Your code absolutely needs to run as fast as possible.
- Nothing else works -- intrinsics should be your last resort!

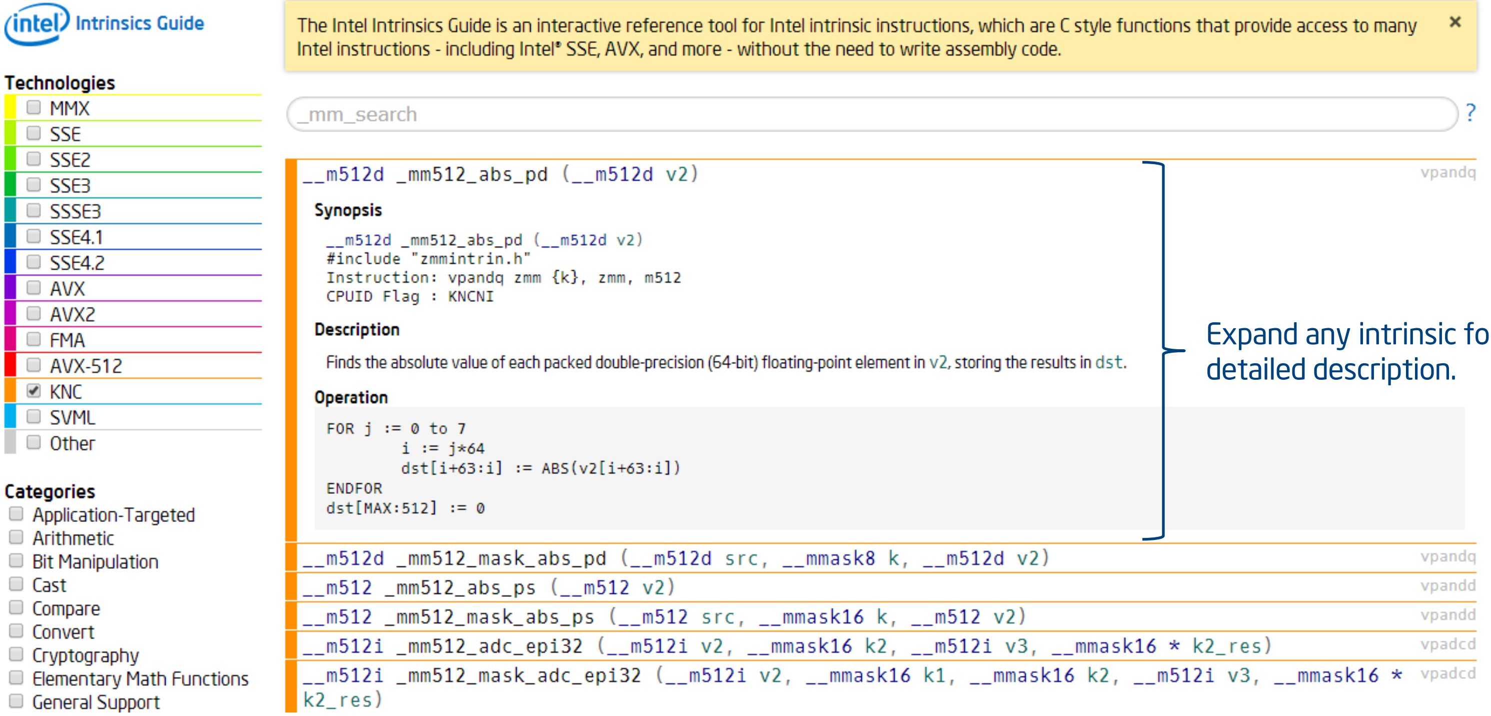
Common Use-cases

- Utilizing particular instructions (e.g. bit-manipulation, cryptography).
- "Horizontal" vectorization (e.g. computing dot products of SIMD registers).
- Exploiting memory layout knowledge (e.g. AoS to SoA transpose).

SIMD Intrinsics – Intel® Intrinsics Guide

Filter by ISA.

Filter by functionality.



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. ×

_mm_search ?

Technologies

- ☐ MMX
- ☐ SSE
- ☐ SSE2
- ☐ SSE3
- ☐ SSSE3
- ☐ SSE4.1
- ☐ SSE4.2
- ☐ AVX
- ☐ AVX2
- ☐ FMA
- ☐ AVX-512
- ☑ KNC
- ☐ SVML
- ☐ Other

Categories

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

`__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
```

`__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

Expand any intrinsic for a detailed description.

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

SIMD Intrinsics – Performance Impact

“Ninja” Performance Gap

- The gap between naïve code and the best-optimized code.
- Average speed-up of 24x on Sandy Bridge and Knights Ferry.

Closing the Gap

- Apply well-known algorithmic techniques (e.g. cache blocking).
- Use a recent compiler, and explicit vectorization features.
- Average speed-up of only ~1.3x from intrinsics.

N. Satish, C. Kim, J. Chhugani, H. Saito, R. Krishnaiyer, M. Smelyanskiy, M. Girkar and P. Dubey, “Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications?”, in Proceedings of the International Symposium on Computer Architecture (ISCA), Portland, OR, 2012

Hands-on:
Explicit Vectorization with OpenMP* 4.0

Accelerating N-Body Codes with SIMD Instructions

Optimization Techniques for Implicit and Explicit Vectorization

Summary

- Intel® Composer XE provides many alternative methods for vectorization.
- Generating vector code is not complicated:
  - Compiler reports / tools help with auto-vec.
  - Explicit vectorization an industry standard (OpenMP* 4.0).

- Extensions to existing languages allow you to target vector architectures while keeping your source code readable, maintainable, and familiar.

Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Knights Landing and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user

Intel, Look Inside, Xeon, Intel Xeon Phi, Pentium, Cilk, VTune and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

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.

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

Optimizing for MPI*/OpenMP* on Intel® Xeon Phi™ Coprocessors

CUG 2014, Lugano, Switzerland

Frequently Asked Questions

Levels of Parallelism

Cluster	Group of computers communicating through fast interconnect
Coprocessors/Accelerators	Special compute devices attached to the local node through special interconnect
Node	Group of processors communicating through shared memory
Socket	Group of cores communicating through shared cache
Core	Group of functional units communicating through registers
Hyper-Threads	Group of thread contexts sharing functional units
Superscalar	Group of instructions sharing functional units
Pipeline	Sequence of instructions sharing functional units
Vector	Single instruction using multiple functional units

Levels of Parallelism in OpenMP* 4.0

Cluster	Group of computers communicating through fast interconnect
Coprocessors/Accelerators	Special compute devices attached to the local node through special interconnect
Node	Group of processors communicating through shared memory
Socket	Group of cores communicating through shared cache
Core	Group of functional units communicating through registers
Hyper-Threads	Group of thread contexts sharing functional units
Superscalar	Group of instructions sharing functional units
Pipeline	Sequence of instructions sharing functional units
Vector	Single instruction using multiple functional units

Levels of Parallelism in MPI* + OpenMP* 4.0

Cluster	Group of computers communicating through fast interconnect
Coprocessors/Accelerators	Special compute devices attached to the local node through special interconnect
Node	Group of processors communicating through shared memory
Socket	Group of cores communicating through shared cache
Core	Group of functional units communicating through registers
Hyper-Threads	Group of thread contexts sharing functional units
Superscalar	Group of instructions sharing functional units
Pipeline	Sequence of instructions sharing functional units
Vector	Single instruction using multiple functional units

Summary

Compilation

- Single toolchain for CPU and MIC; can often just add “-mmic”.
- Support for industry standards (including MPI* and OpenMP*).

Optimization

- Biggest benefits from tuning memory and vector behaviour.
- Straightforward parallel tuning techniques still apply.
- Intel tools can help to identify optimization opportunities (and are always improving).
- “Dual-Transforming-Tuning Advantage”; optimize for processor and coprocessor.

Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Knights Landing and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user

Intel, Look Inside, Xeon, Intel Xeon Phi, Pentium, Cilk, VTune and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

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.

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

Intel® Xeon Phi™ Product Family

Intel ® ITAC

Heinrich Bockhorst, May 5th 2014

Software and Services Group
Intel Corporation

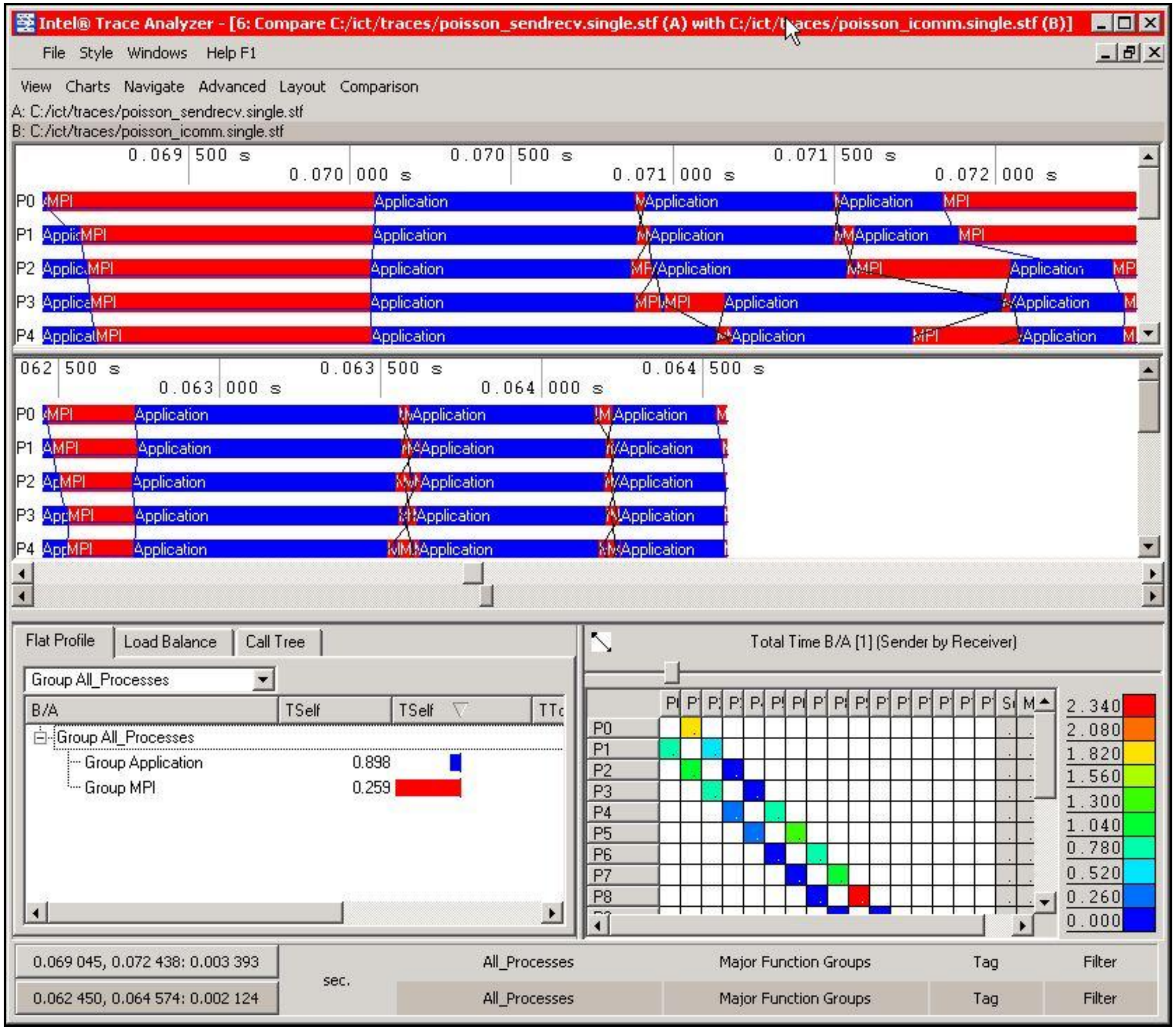
Introduction – What is Tracing?

- Record program execution
  - Program events such as function enter/exit, communication
- 1:1 protocol of the actual program execution
  - Sampling gathers statistical information
- Accurate data
- Easily get loads of data

Event based approach

- Event = time stamp + thread ID + description
  - Function entry/exit
  - Messages
  - Collective operations
  - Counter samples
- Strengths:
  - Predict detailed program behavior
  - Record exact sequence of program states – keep timing consistent
  - Collect information about exchange of messages: at what times and in which order
  - Detect temporal dependencies

Intel® Trace Analyzer and Collector


Intel® Trace Analyzer - [6: Compare C:/ict/traces/poisson_sendrecv.single.stf (A) with C:/ict/traces/poisson_icomm.single.stf (B)]

A: C:/ict/traces/poisson_sendrecv.single.stf
B: C:/ict/traces/poisson_icomm.single.stf

Group All_Processes

B/A	TSelf	TSelf
Group All_Processes		
Group Application		0.898
Group MPI		0.259

Total Time B/A [1] (Sender by Receiver)

0.069 045, 0.072 438: 0.003 393	sec.	All_Processes	Major Function Groups	Tag	Filter
0.062 450, 0.064 574: 0.002 124		All_Processes	Major Function Groups	Tag	Filter

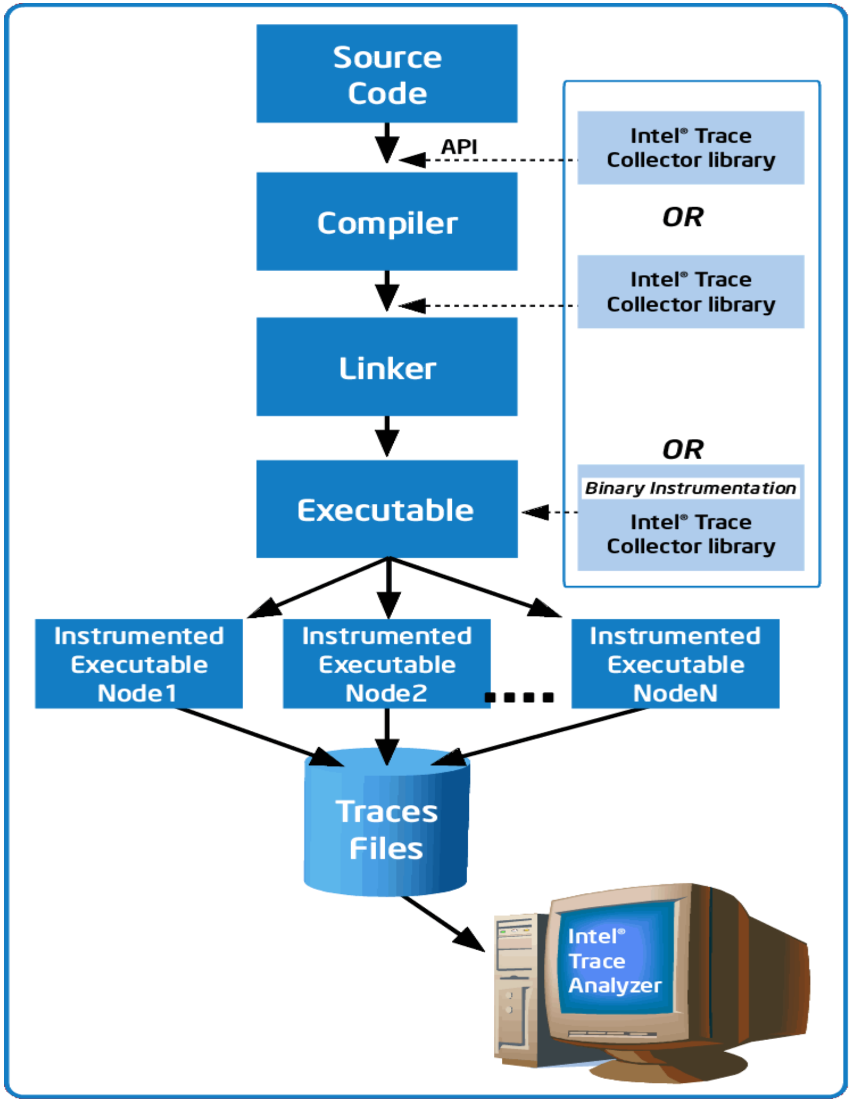
Compare the event timelines of two communication profiles

Blue = computation
Red = communication

Chart showing how the MPI processes interact

Intel® Trace Analyzer and Collector Overview

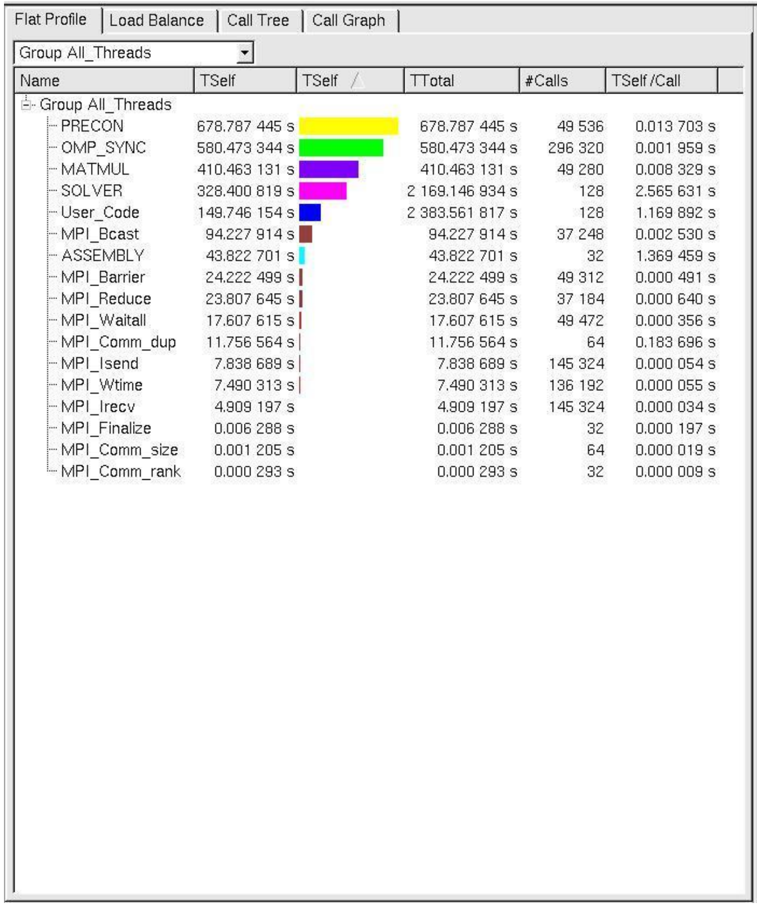
- **Intel® Trace Analyzer and Collector helps the developer:**
  - Visualize and understand parallel application behavior
  - Evaluate profiling statistics and load balancing
  - Identify communication hotspots
- **Features**
  - Event-based approach
  - Low overhead
  - Excellent scalability
  - Comparison of multiple profiles
  - Powerful aggregation and filtering functions
  - Fail-safe MPI tracing
  - Provides API to instrument user code
  - MPI correctness checking
  - Idealizer


Source Code
API
Intel® Trace Collector library
Compiler
OR
Intel® Trace Collector library
Linker
OR
Binary Instrumentation
Intel® Trace Collector library
Executable
Instrumented Executable Node1
Instrumented Executable Node2
....
Instrumented Executable NodeN
Traces Files
Intel® Trace Analyzer

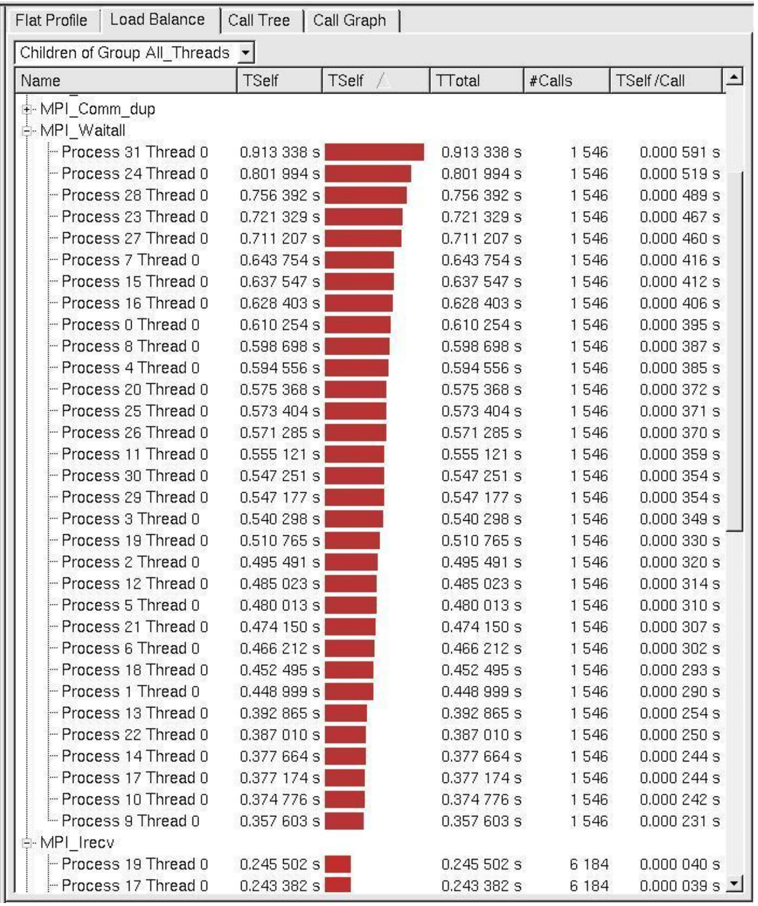
Optimization Notice

Profiles: Flat Function Profile

- Statistics about functions



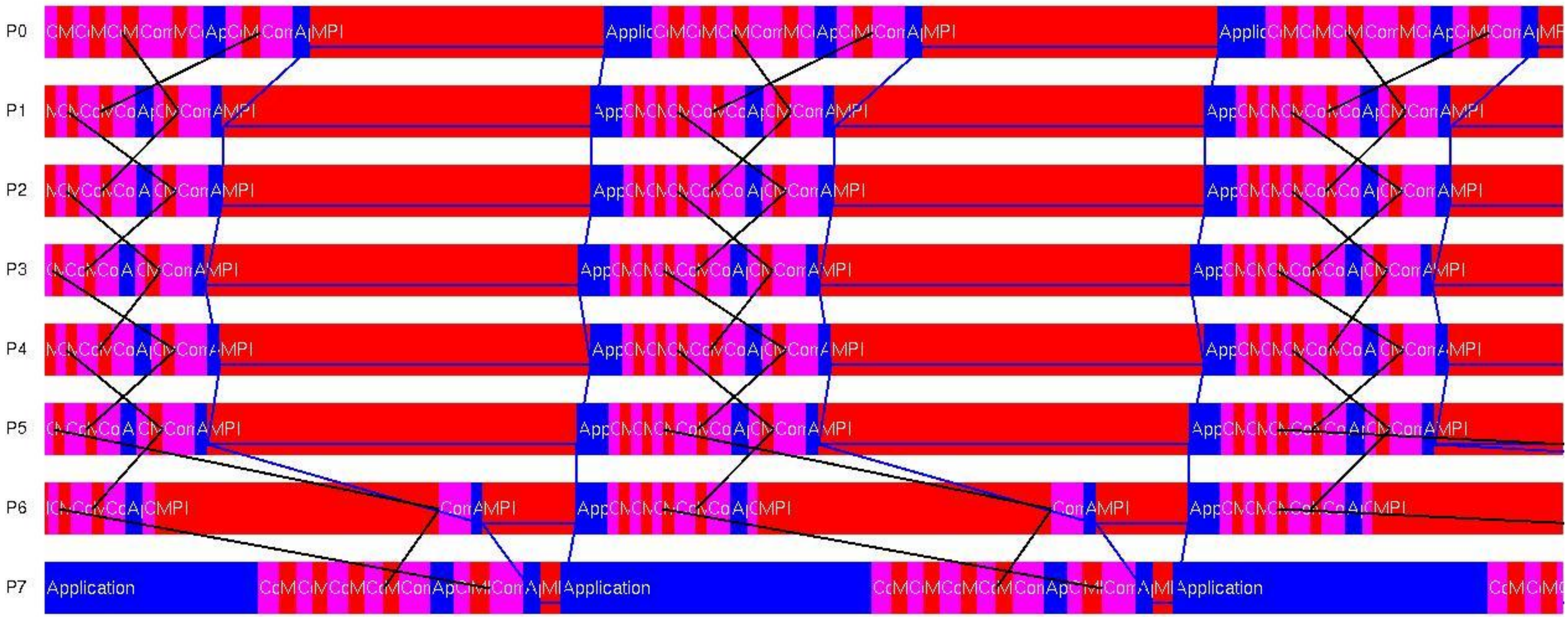
Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Threads					
PRECON	678.787 445 s		678.787 445 s	49 536	0.013 703 s
OMP_SYNC	580.473 344 s		580.473 344 s	296 320	0.001 959 s
MATMUL	410.463 131 s		410.463 131 s	49 280	0.008 329 s
SOLVER	328.400 819 s		2 169.146 934 s	128	2.565 631 s
User_Code	149.746 154 s		2 383.561 817 s	128	1.169 892 s
MPI_Bcast	94.227 914 s		94.227 914 s	37 248	0.002 530 s
ASSEMBLY	43.822 701 s		43.822 701 s	32	1.369 459 s
MPI_Barrier	24.222 499 s		24.222 499 s	49 312	0.000 491 s
MPI_Reduce	23.807 645 s		23.807 645 s	37 184	0.000 640 s
MPI_Waitall	17.607 615 s		17.607 615 s	49 472	0.000 356 s
MPI_Comm_dup	11.756 564 s		11.756 564 s	64	0.183 696 s
MPI_Isend	7.838 689 s		7.838 689 s	145 324	0.000 054 s
MPI_Wtime	7.490 313 s		7.490 313 s	136 192	0.000 055 s
MPI_Irecv	4.909 197 s		4.909 197 s	145 324	0.000 034 s
MPI_Finalize	0.006 288 s		0.006 288 s	32	0.000 197 s
MPI_Comm_size	0.001 205 s		0.001 205 s	64	0.000 019 s
MPI_Comm_rank	0.000 293 s		0.000 293 s	32	0.000 009 s



Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
MPI_Comm_dup					
MPI_Waitall					
Process 31 Thread 0	0.913 338 s		0.913 338 s	1 546	0.000 591 s
Process 24 Thread 0	0.801 994 s		0.801 994 s	1 546	0.000 519 s
Process 28 Thread 0	0.756 392 s		0.756 392 s	1 546	0.000 489 s
Process 23 Thread 0	0.721 329 s		0.721 329 s	1 546	0.000 467 s
Process 27 Thread 0	0.711 207 s		0.711 207 s	1 546	0.000 460 s
Process 7 Thread 0	0.643 754 s		0.643 754 s	1 546	0.000 416 s
Process 15 Thread 0	0.637 547 s		0.637 547 s	1 546	0.000 412 s
Process 16 Thread 0	0.628 403 s		0.628 403 s	1 546	0.000 406 s
Process 0 Thread 0	0.610 254 s		0.610 254 s	1 546	0.000 395 s
Process 8 Thread 0	0.598 698 s		0.598 698 s	1 546	0.000 387 s
Process 4 Thread 0	0.594 556 s		0.594 556 s	1 546	0.000 385 s
Process 20 Thread 0	0.575 368 s		0.575 368 s	1 546	0.000 372 s
Process 25 Thread 0	0.573 404 s		0.573 404 s	1 546	0.000 371 s
Process 26 Thread 0	0.571 285 s		0.571 285 s	1 546	0.000 370 s
Process 11 Thread 0	0.555 121 s		0.555 121 s	1 546	0.000 359 s
Process 30 Thread 0	0.547 251 s		0.547 251 s	1 546	0.000 354 s
Process 29 Thread 0	0.547 177 s		0.547 177 s	1 546	0.000 354 s
Process 3 Thread 0	0.540 298 s		0.540 298 s	1 546	0.000 349 s
Process 19 Thread 0	0.510 765 s		0.510 765 s	1 546	0.000 330 s
Process 2 Thread 0	0.495 491 s		0.495 491 s	1 546	0.000 320 s
Process 12 Thread 0	0.485 023 s		0.485 023 s	1 546	0.000 314 s
Process 5 Thread 0	0.480 013 s		0.480 013 s	1 546	0.000 310 s
Process 21 Thread 0	0.474 150 s		0.474 150 s	1 546	0.000 307 s
Process 6 Thread 0	0.466 212 s		0.466 212 s	1 546	0.000 302 s
Process 18 Thread 0	0.452 495 s		0.452 495 s	1 546	0.000 293 s
Process 1 Thread 0	0.448 999 s		0.448 999 s	1 546	0.000 290 s
Process 13 Thread 0	0.392 865 s		0.392 865 s	1 546	0.000 254 s
Process 22 Thread 0	0.387 010 s		0.387 010 s	1 546	0.000 250 s
Process 14 Thread 0	0.377 664 s		0.377 664 s	1 546	0.000 244 s
Process 17 Thread 0	0.377 174 s		0.377 174 s	1 546	0.000 244 s
Process 10 Thread 0	0.374 776 s		0.374 776 s	1 546	0.000 242 s
Process 9 Thread 0	0.357 603 s		0.357 603 s	1 546	0.000 231 s
MPI_Irecv					
Process 19 Thread 0	0.245 502 s		0.245 502 s	6 184	0.000 040 s
Process 17 Thread 0	0.243 382 s		0.243 382 s	6 184	0.000 039 s

Timelines: Event Timeline

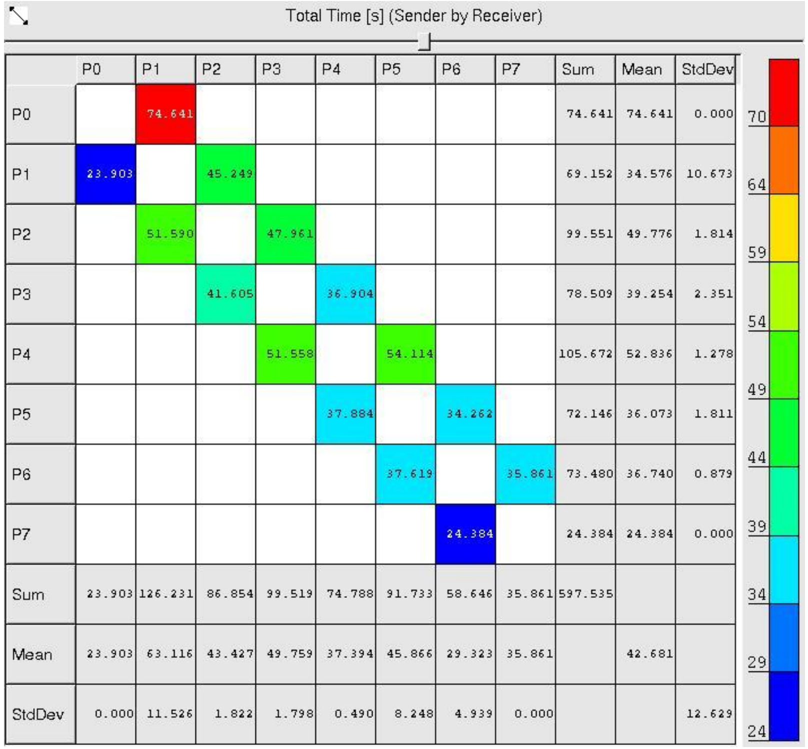
- Get impression of program structure
- Display functions, messages and collective operations for each process/thread along time-axis
- Retrieval of detailed event information



Optimization Notice

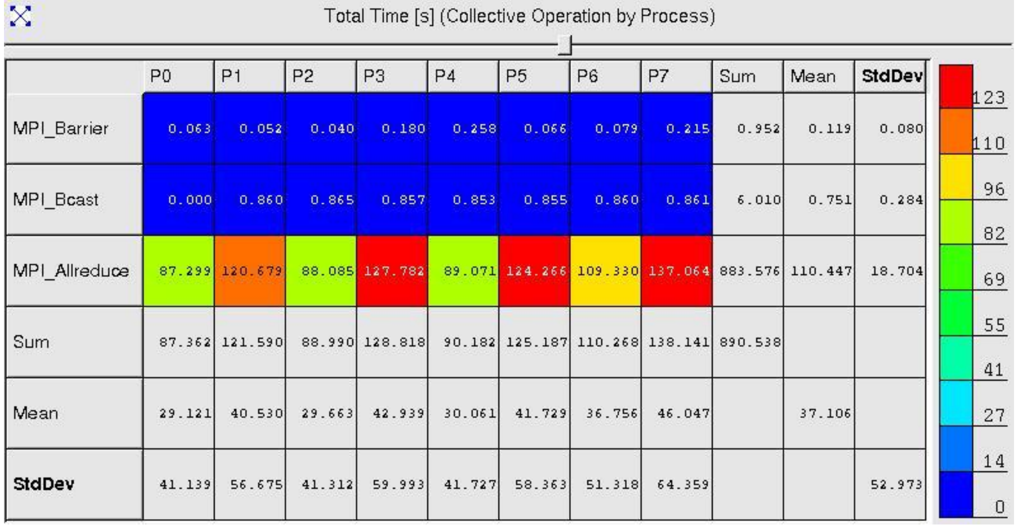
Communication Profiles

- Statistics about point-to-point or collective communication
- Generic matrix supports grouping by several attributes in each dimension
  Sender, Receiver, Data volume per msg, Tag, Communicator, Type
- Available attributes: Count, Bytes transferred, Time, Transfer rate



Total Time [s] (Sender by Receiver)

	P0	P1	P2	P3	P4	P5	P6	P7	Sum	Mean	StdDev
P0		74.641							74.641	74.641	0.000
P1	23.903		45.249						69.152	34.576	10.673
P2		51.590		47.961					99.551	49.776	1.814
P3			41.605		36.904				78.509	39.254	2.351
P4				51.558		54.114			105.672	52.836	1.278
P5					37.884		34.262		72.146	36.073	1.811
P6						37.619		35.861	73.480	36.740	0.879
P7							24.384		24.384	24.384	0.000
Sum	23.903	126.231	86.854	99.519	74.788	91.733	58.646	35.861	597.535		
Mean	23.903	63.116	43.427	49.759	37.394	45.866	29.323	35.861		42.681	
StdDev	0.000	11.526	1.822	1.798	0.490	8.248	4.939	0.000			12.629

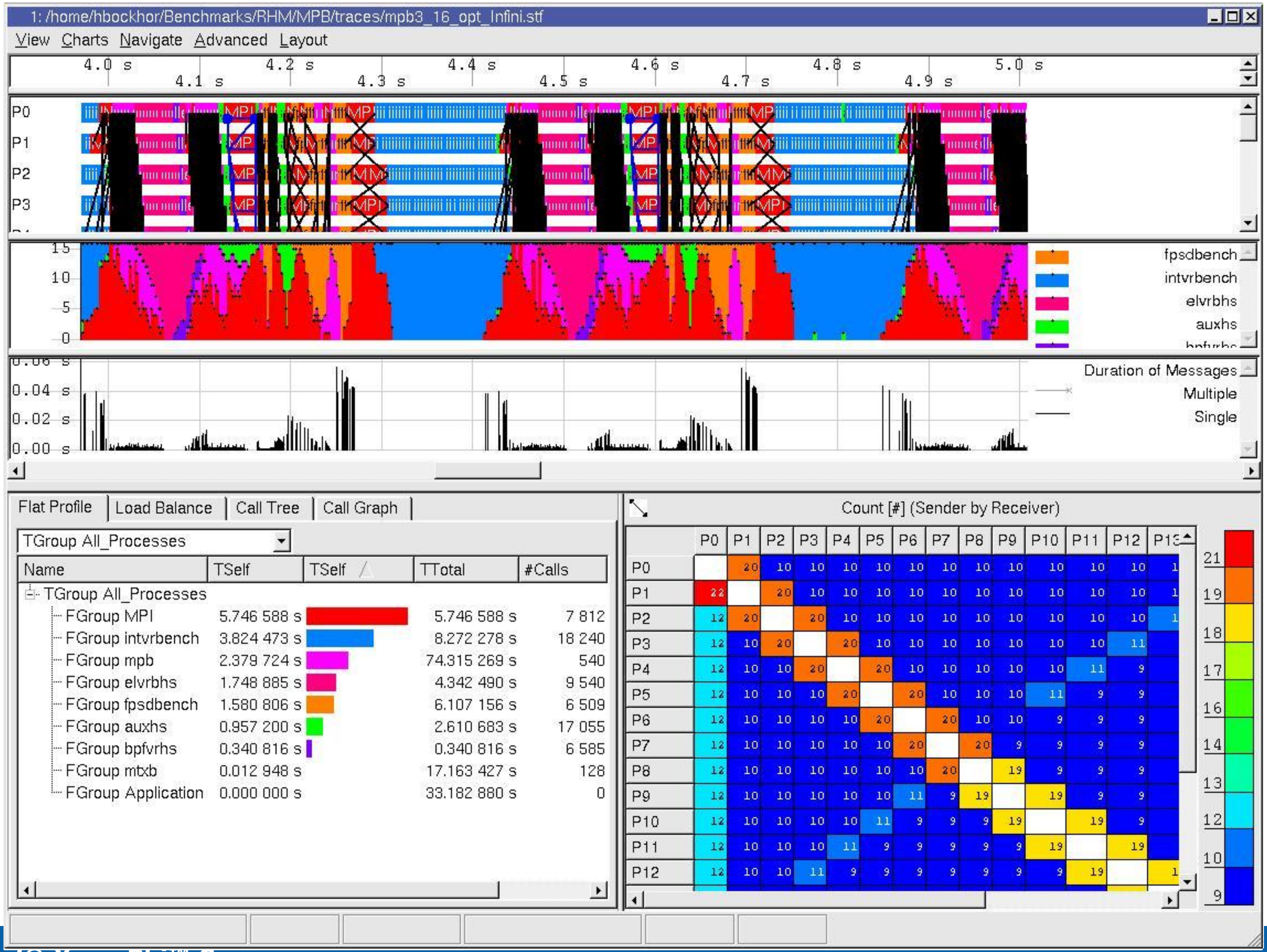


Total Time [s] (Collective Operation by Process)

	P0	P1	P2	P3	P4	P5	P6	P7	Sum	Mean	**StdDev**
MPI_Barrier	0.063	0.052	0.040	0.180	0.258	0.066	0.079	0.215	0.952	0.119	0.080
MPI_Bcast	0.000	0.860	0.865	0.857	0.853	0.855	0.860	0.861	6.010	0.751	0.284
MPI_Allreduce	87.299	120.679	88.085	127.782	89.071	124.266	109.330	137.064	883.576	110.447	18.704
Sum	87.362	121.590	88.990	128.818	90.182	125.187	110.268	138.141	890.538		
Mean	29.121	40.530	29.663	42.939	30.061	41.729	36.756	46.047		37.106	
StdDev	41.139	56.675	41.312	59.993	41.727	58.363	51.318	64.359			52.973

Optimization Notice

View - zooming



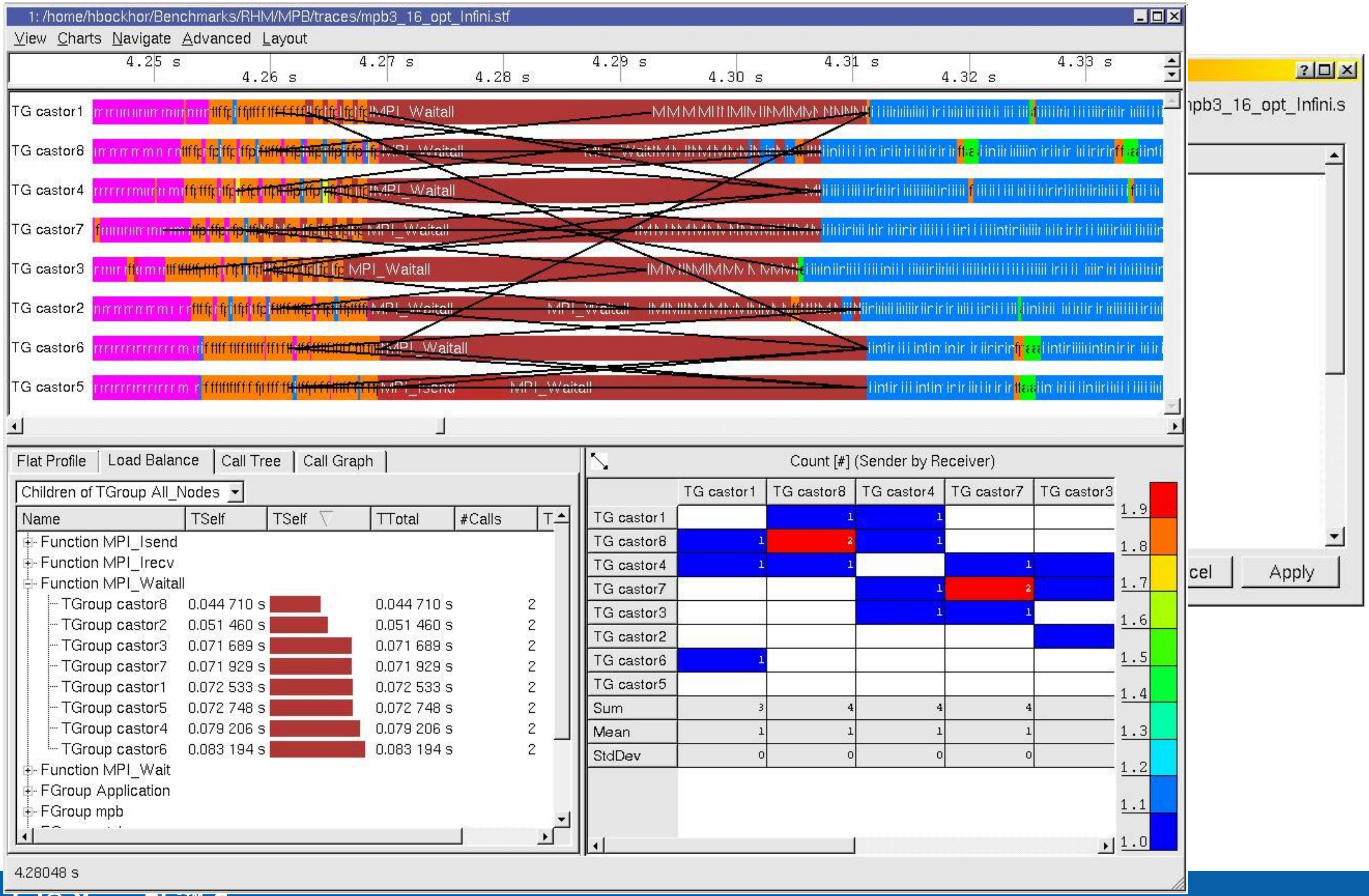
Name	TSelf	TSelf	TTotal	#Calls
TGroup All_Processes				
FGroup MPI	5.746 588 s		5.746 588 s	7 812
FGroup intvrbench	3.824 473 s		8.272 278 s	18 240
FGroup mpb	2.379 724 s		74.315 269 s	540
FGroup elvrbhs	1.748 885 s		4.342 490 s	9 540
FGroup fpsdbench	1.580 806 s		6.107 156 s	6 509
FGroup auxhs	0.957 200 s		2.610 683 s	17 055
FGroup bpfvrhs	0.340 816 s		0.340 816 s	6 585
FGroup mtxb	0.012 948 s		17.163 427 s	128
FGroup Application	0.000 000 s		33.182 880 s	0

Count [#] (Sender by Receiver)

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
P0		20	10	10	10	10	10	10	10	10	10	10	10	1
P1	22		20	10	10	10	10	10	10	10	10	10	10	1
P2	12	20		20	10	10	10	10	10	10	10	10	10	1
P3	12	10	20		20	10	10	10	10	10	10	10	11	
P4	12	10	10	20		20	10	10	10	10	10	11	9	
P5	12	10	10	10	20		20	10	10	10	11	9	9	
P6	12	10	10	10	10	20		20	10	10	9	9	9	
P7	12	10	10	10	10	10	20		20	9	9	9	9	
P8	12	10	10	10	10	10	10	20		19	9	9	9	
P9	12	10	10	10	10	10	11	9	19		19	9	9	
P10	12	10	10	10	10	11	9	9	9	19		19	9	
P11	12	10	10	10	11	9	9	9	9	9	19		19	
P12	12	10	10	11	9	9	9	9	9	9	9	19		1

Optimization Notice

Aggregation Example



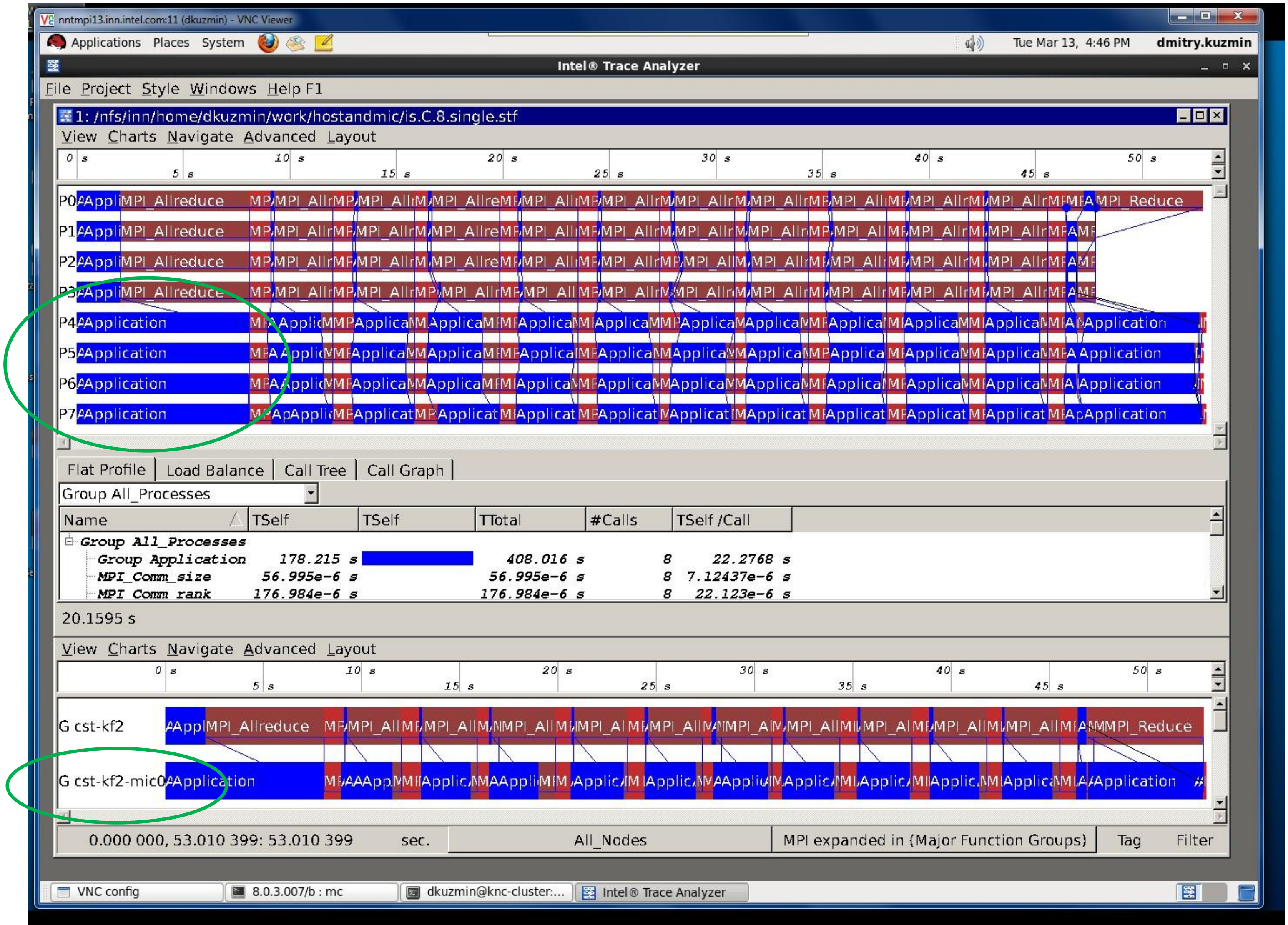
Name	TSelf	TSelf	TTotal	#Calls	T
Function MPI_Isend					
Function MPI_Irecv					
Function MPI_Waitall					
TGroup castor8	0.044 710 s		0.044 710 s	2	
TGroup castor2	0.051 460 s		0.051 460 s	2	
TGroup castor3	0.071 689 s		0.071 689 s	2	
TGroup castor7	0.071 929 s		0.071 929 s	2	
TGroup castor1	0.072 533 s		0.072 533 s	2	
TGroup castor5	0.072 748 s		0.072 748 s	2	
TGroup castor4	0.079 206 s		0.079 206 s	2	
TGroup castor6	0.083 194 s		0.083 194 s	2	
Function MPI_Wait					
FGroup Application					
FGroup mpb					

Count [#] (Sender by Receiver)

	TG castor1	TG castor8	TG castor4	TG castor7	TG castor3
TG castor1		1	1		
TG castor8	1	2	1		
TG castor4	1	1		1	
TG castor7			1	2	
TG castor3			1	1	
TG castor2					
TG castor6	1				
TG castor5					
Sum	3	4	4	4	
Mean	1	1	1	1	
StdDev	0	0	0	0	

Optimization Notice

Full ITAC Functionality on Intel® Xeon Phi™



Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
⊟ *Group All_Processes*					
Group Application	*178.215 s*		*408.016 s*	*8*	*22.2768 s*
MPI_Comm_size	*56.995e-6 s*		*56.995e-6 s*	*8*	*7.12437e-6 s*
MPI_Comm_rank	*176.984e-6 s*		*176.984e-6 s*	*8*	*22.123e-6 s*

20.1595 s

0.000 000, 53.010 399: 53.010 399 sec. All_Nodes MPI expanded in (Major Function Groups) Tag Filter

ITAC Prerequisites

- Set ITAC environment (per user)

  ```
  # source /opt/intel/itac/8.1.2.033/intel64/bin/itacvars.sh impi4
  ```

  - Identical for host and the coprocessor

- No NFS: Upload ITAC library manually

  ```
  # sudo scp /opt/intel/itac/8.1.2.033/mic/slib/libVT.so host-
    mic0:/lib64/
  ```

ITAC Usage with Intel® Xeon Phi™ Coprocessor

- Run with –trace flag (without linkage) to create a trace file
  - MPI+Offload

    ```
    # mpirun –trace -n 2 ./test
    ```

  - Coprocessor only

    ```
    # mpirun –trace -n 2 -wdir /tmp
      -host host-mic0 ./test_hello.MIC
    ```

  - Symmetric

    ```
    # mpirun –trace -n 2 -host michost./test_hello :
      -wdir /tmp -n 2 -host host-mic0
      ./test_hello.MIC
    ```

- Flag „-trace“ will implicitly pre-load libVT.so (which finally calls libmpi.so to execute the MPI call)
- Set `VT_LOGFILE_FORMAT=stfsingle` to create a single trace

Optimization Notice

ITAC Usage with Intel® Xeon Phi™: Compilation Support

- Compile and link with „–trace“ flag

```
# mpiicc -trace -o test_hello test.c
# mpiicc –trace –mmic -o test_hello.MIC test.c
```

  - Linkage of libVT library
- Compile with –tcollect flag

```
# mpiicc –tcollect -o test_hello test.c
# mpiicc –tcollect –mmic -o test_hello.MIC test.c
```

  - Linkage of libVT library
  - Will do a full instrumentation of your code, i.e. All user functions will be visible in the trace file
  - Maximal insight, but also maximal overhead
- Use the VT API of ITAC to manually instrument your code.
- Run Intel® MPI program as usual without „-trace“ flag

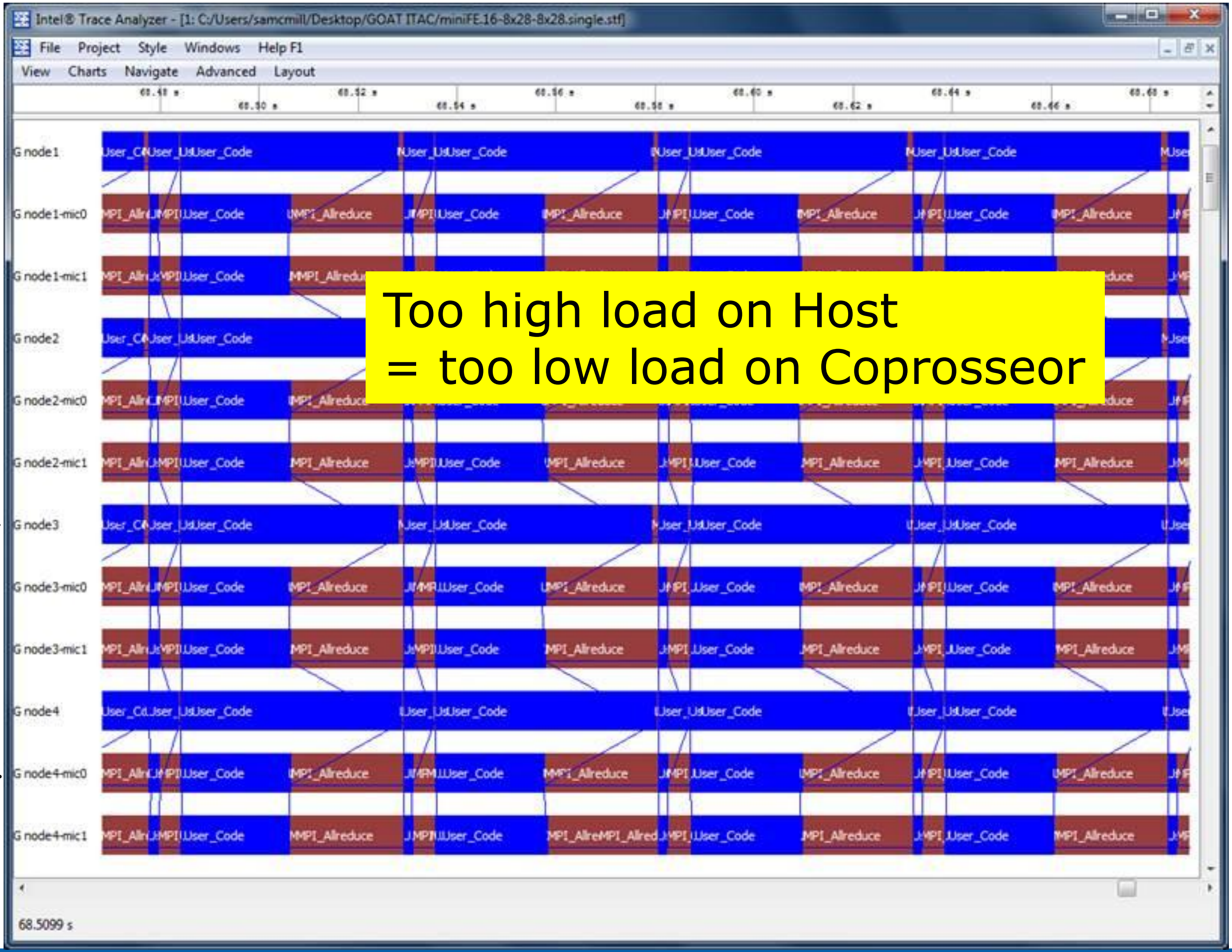
```
# mpirun ...
```

Improving Load Balance: Real World Case

Collapsed data
per node and
coprocessor

Host
16 MPI procs x
1 OpenMP thread

Coprocessor
8 MPI procs x
28 OpenMP threads



Intel® Trace Analyzer - [1: C:/Users/samcmill/Desktop/GOAT ITAC/miniFE.16-8x28-8x28.single.stf]

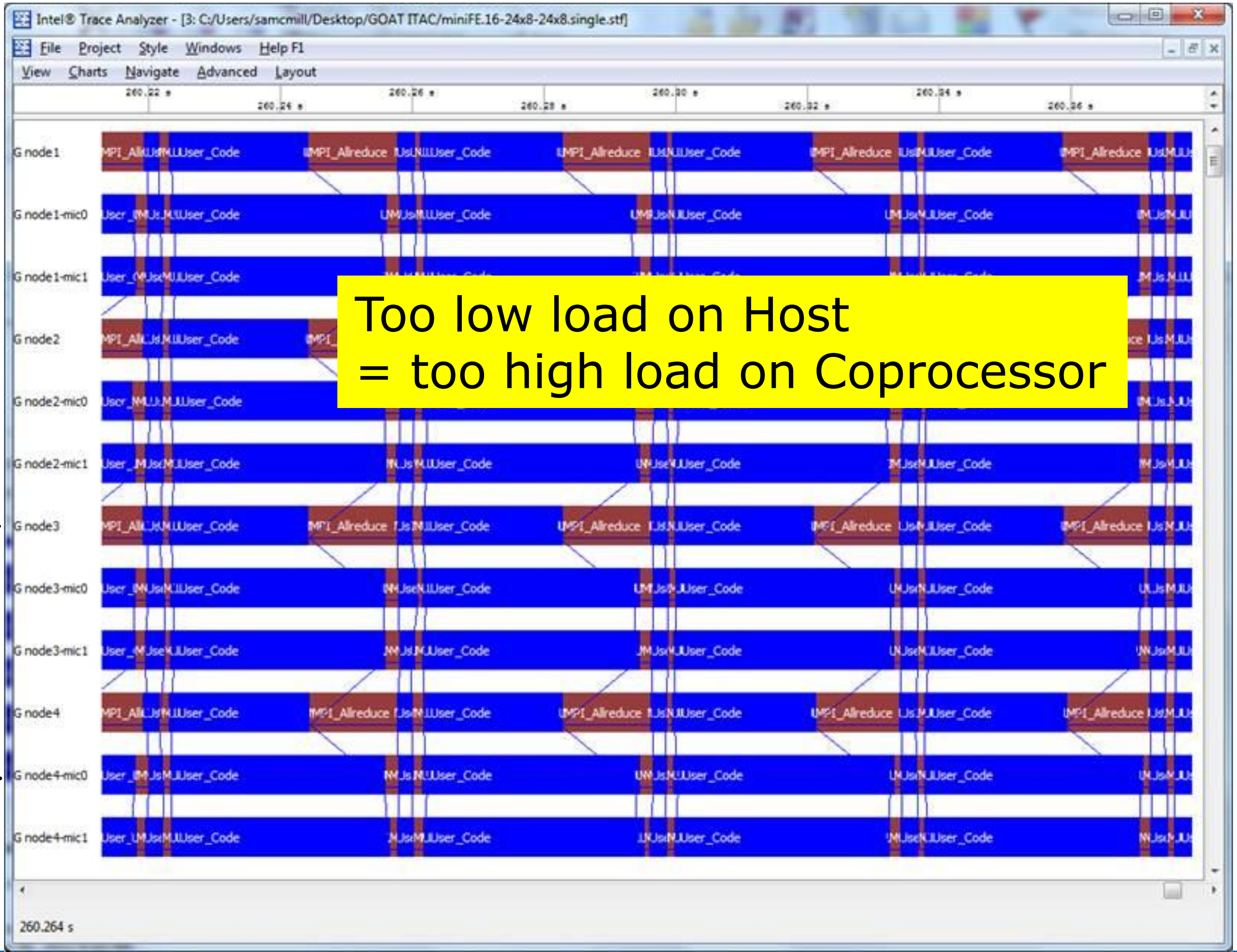
Too high load on Host
= too low load on Coprosseor

Improving Load Balance: Real World Case

Collapsed data per node and coprocessor

Host
16 MPI procs x
1 OpenMP thread

Coprocessor
24 MPI procs x
8 OpenMP threads



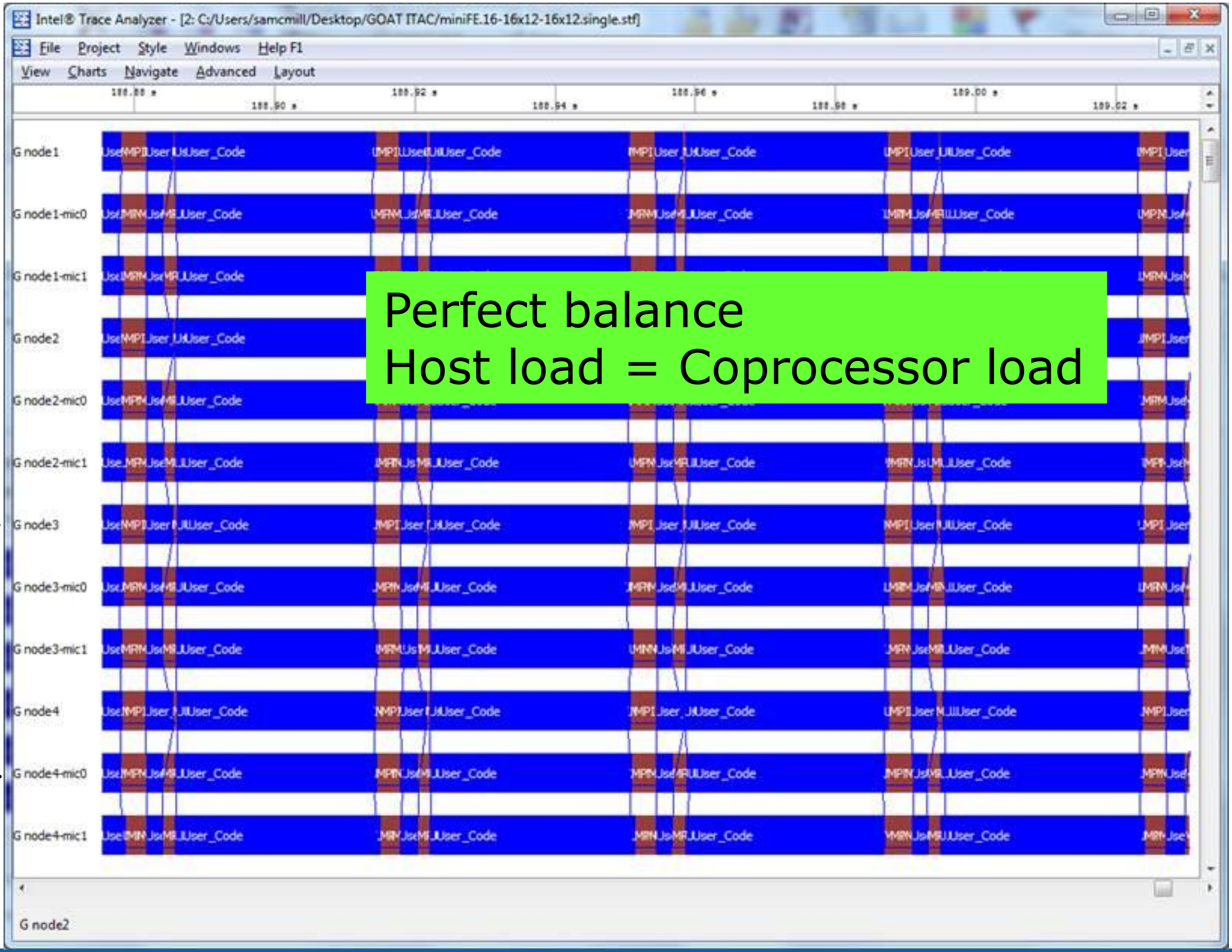
Too low load on Host
= too high load on Coprocessor

Improving Load Balance: Real World Case

Collapsed data per node and coprocessor

Host
16 MPI procs x
1 OpenMP thread

Coprocessor
16 MPI procs x
12 OpenMP thrds

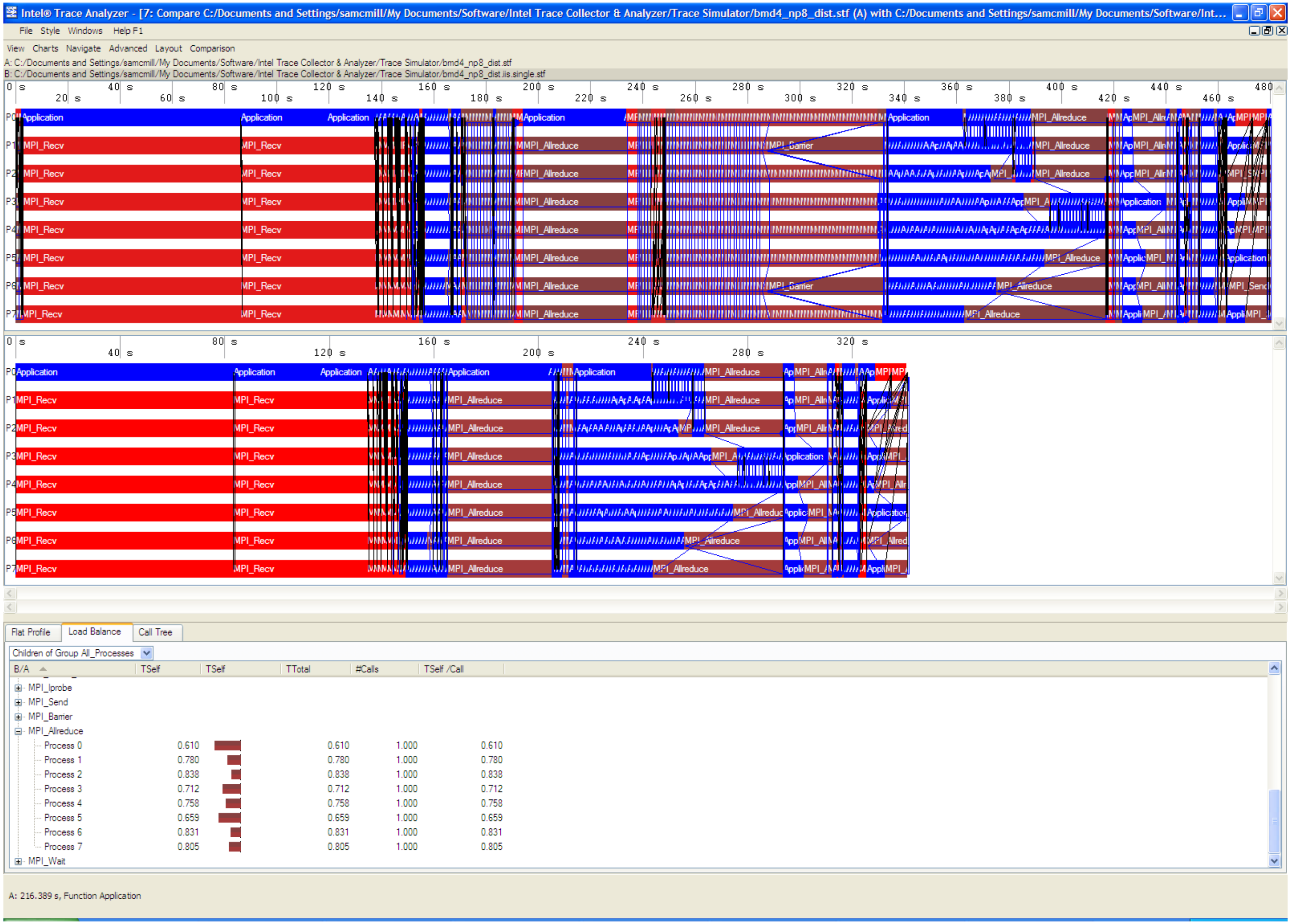


Perfect balance
Host load = Coprocessor load

Ideal Interconnect Simulator (Idealizer)

- What is the Ideal Interconnect Simulator?
  - Using a ITAC trace of an MPI application, simulate it under ideal conditions
    - Zero network latency
    - Infinite network bandwidth
    - Zero MPI buffer copy time
    - Infinite MPI buffer size
  - Only limiting factors are concurrency rules, e.g.,
    - A message can not be received before it is sent
    - An All-to-All collective may end only when the last thread starts

Ideal Interconnect Simulator (Idealizer)



Intel® Trace Analyzer - [7: Compare C:/Documents and Settings/samcmill/My Documents/Software/Intel Trace Collector & Analyzer/Trace Simulator/bmd4_np8_dist.stf (A) with C:/Documents and Settings/samcmill/My Documents/Software/Int...

File Style Windows Help F1

View Charts Navigate Advanced Layout Comparison

A: C:/Documents and Settings/samcmill/My Documents/Software/Intel Trace Collector & Analyzer/Trace Simulator/bmd4_np8_dist.stf
B: C:/Documents and Settings/samcmill/My Documents/Software/Intel Trace Collector & Analyzer/Trace Simulator/bmd4_np8_dist.iis.single.stf

Children of Group All_Processes

B/A	TSelf	TSelf	TTotal	#Calls	TSelf /Call
⊞ MPI_Iprobe					
⊞ MPI_Send					
⊞ MPI_Barrier					
⊟ MPI_Allreduce					
Process 0	0.610		0.610	1.000	0.610
Process 1	0.780		0.780	1.000	0.780
Process 2	0.838		0.838	1.000	0.838
Process 3	0.712		0.712	1.000	0.712
Process 4	0.758		0.758	1.000	0.758
Process 5	0.659		0.659	1.000	0.659
Process 6	0.831		0.831	1.000	0.831
Process 7	0.805		0.805	1.000	0.805
⊞ MPI_Wait					

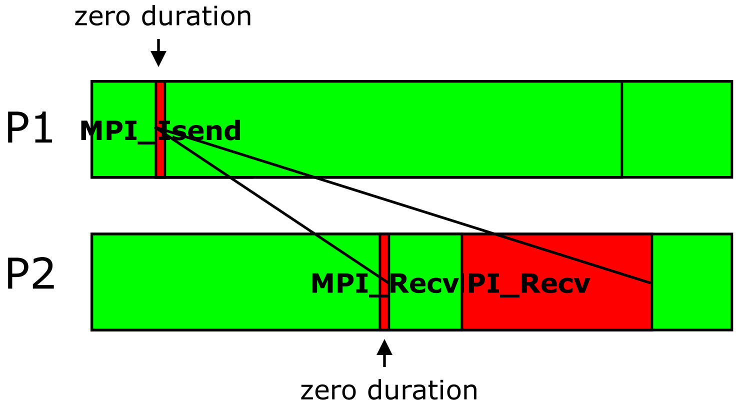
A: 216.389 s, Function Application

Actual trace

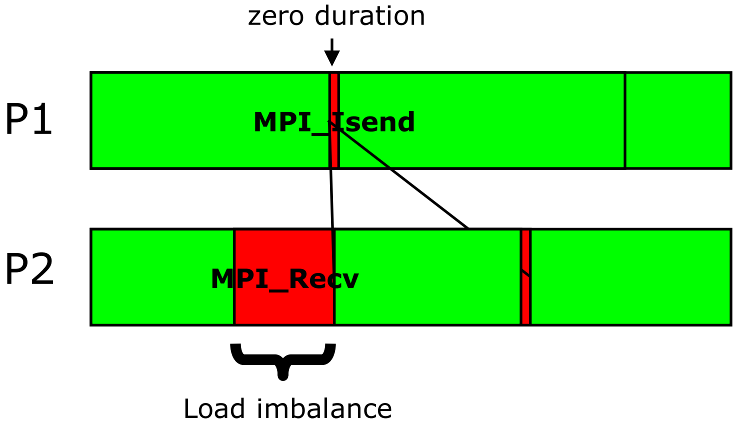
Idealized Trace

Building Blocks: Elementary Messages

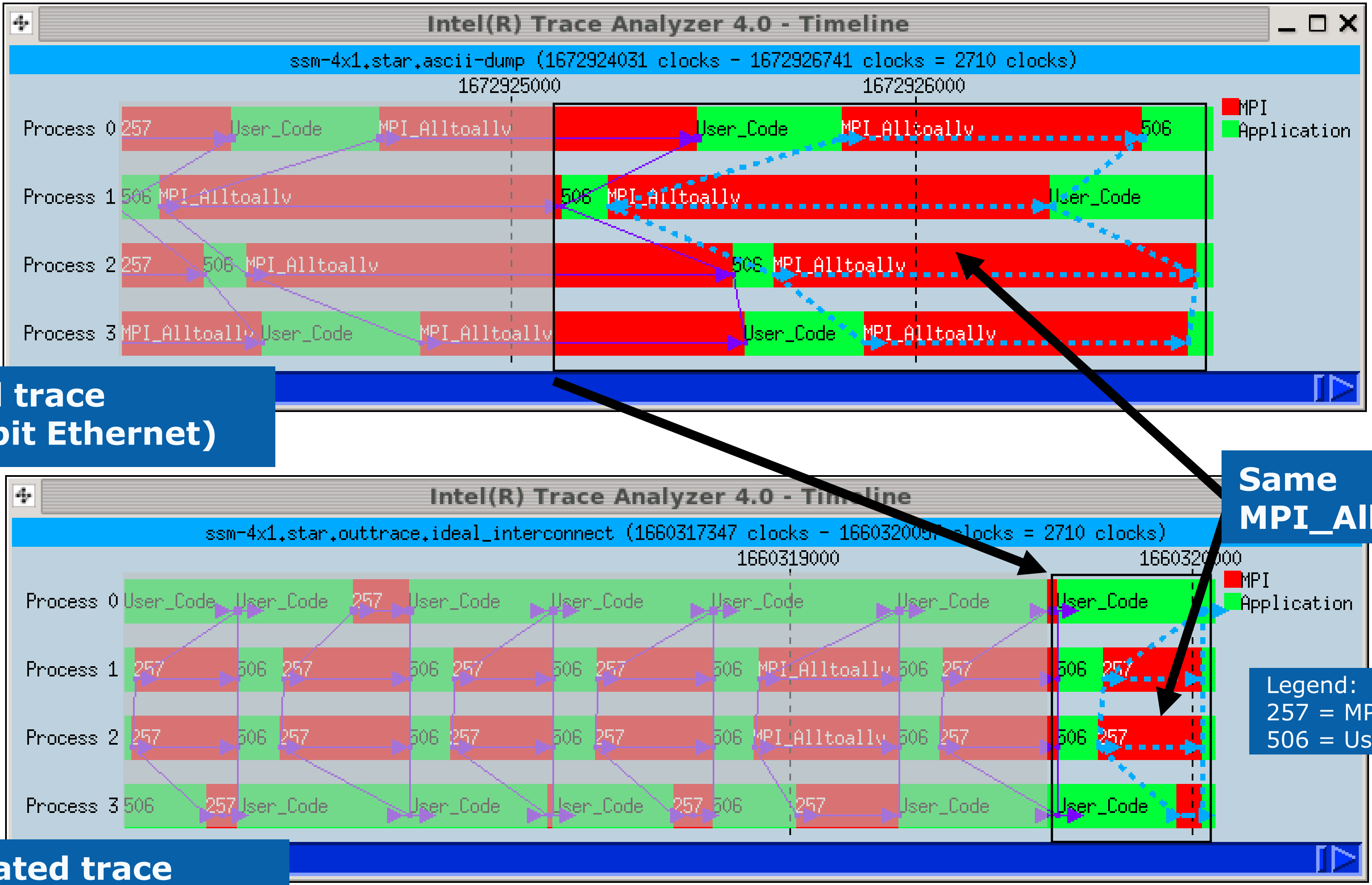
Early Send / Late Receive



Late Send / Early Receive



Building Blocks: Collective Operations



Intel(R) Trace Analyzer 4.0 - Timeline

ssm-4x1.star.ascii-dump (1672924031 clocks - 1672926741 clocks = 2710 clocks)

Actual trace (Gigabit Ethernet)

Intel(R) Trace Analyzer 4.0 - Timeline

ssm-4x1.star.outtrace.ideal_interconnect (1660317347 clocks - 1660320057 clocks = 2710 clocks)

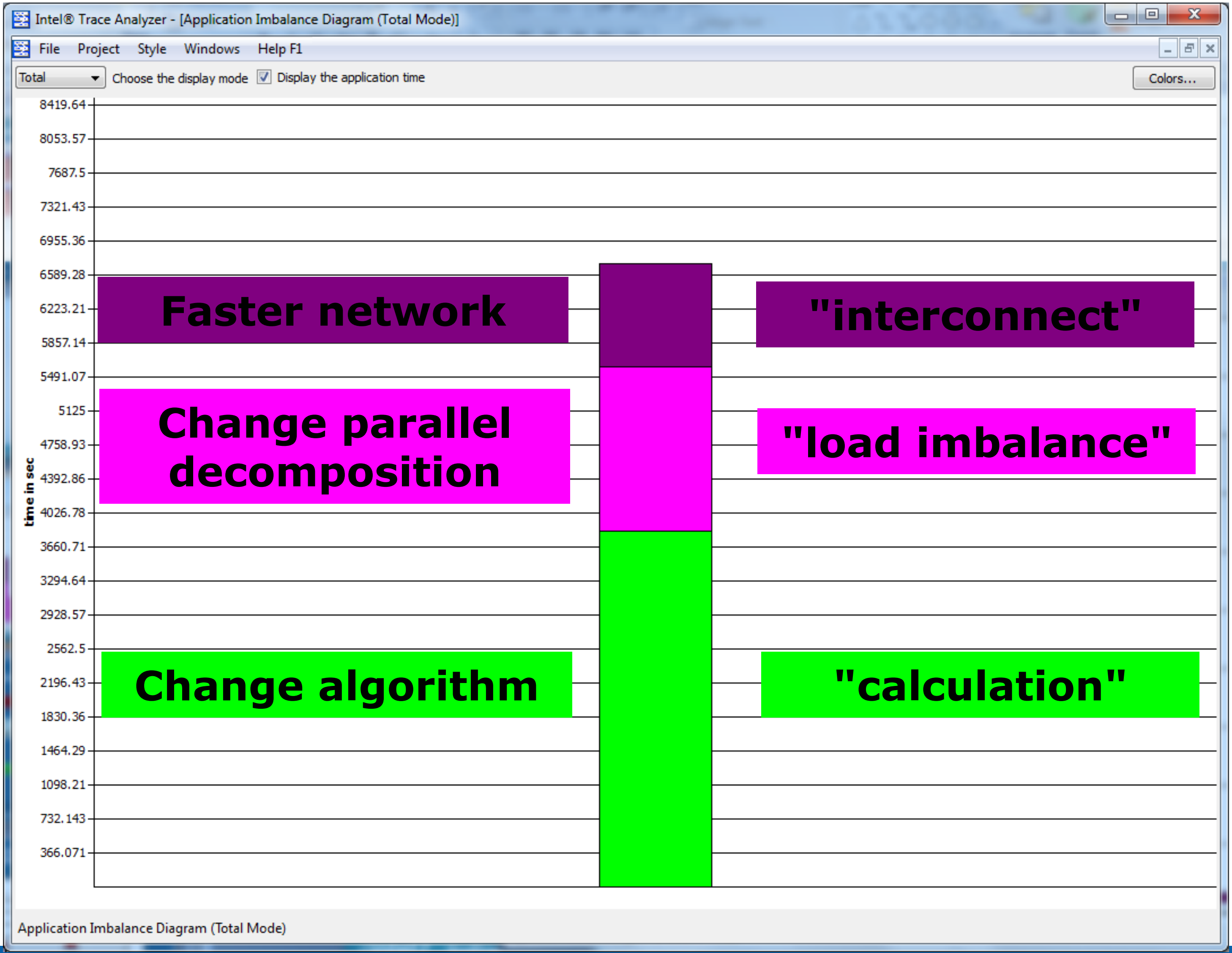
Same MPI_Alltoallv

Legend:
257 = MPI_Alltoallv
506 = User_Code

Simulated trace (Ideal interconnect)

Same timescale in both figures

Application Imbalance Diagram: Total



Intel® Trace Analyzer - [Application Imbalance Diagram (Total Mode)]

Faster network

"interconnect"

Change parallel decomposition

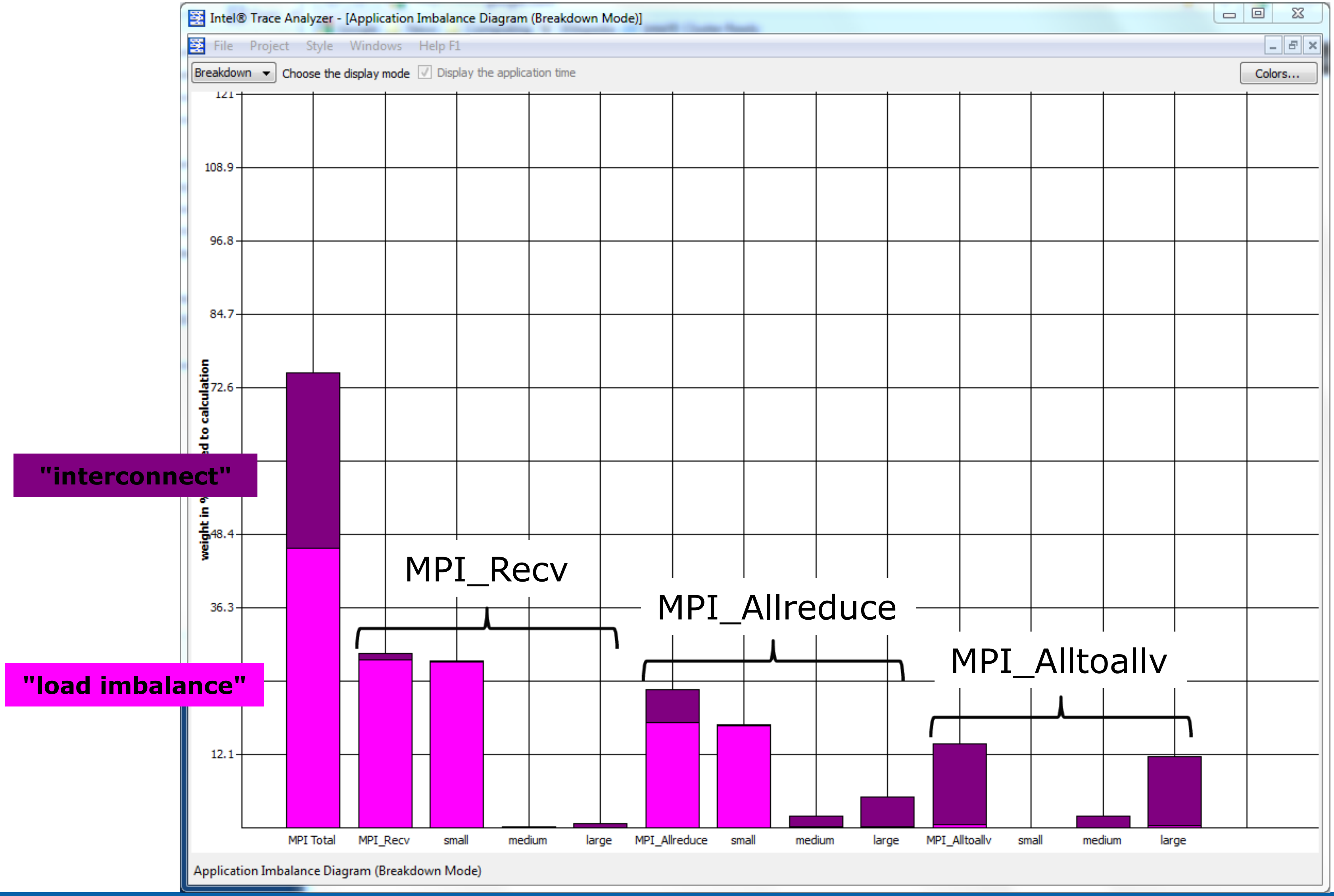
"load imbalance"

Change algorithm

"calculation"

MPI

Application Imbalance Diagram: Breakdown



Online Resources

- Intel® MPI Library product page
  www.intel.com/go/mpi
- Intel® Trace Analyzer and Collector product page
  www.intel.com/go/traceanalyzer
- Intel® Clusters and HPC Technology forums
  http://software.intel.com/en-us/forums/intel-clusters-and-hpc-technology/
- Intel® Xeon Phi™ Coprocessor Developer Community
  http://software.intel.com/en-us/mic-developer

Summary

- The ease of use of Intel® MPI and related tools like the Intel Trace Analyzer and Collector extends from the Intel Xeon architecture to the Intel® Xeon Phi™ coprocessor.

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 © 2014, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, 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

VTune for Intel® Xeon Phi™ Coprocessors

Heinrich Bockhorst, May 5th 2014

Software and Services Group
Intel Corporation

Agenda

Start tuning on host

Overview of Intel® VTune™ Amplifier XE

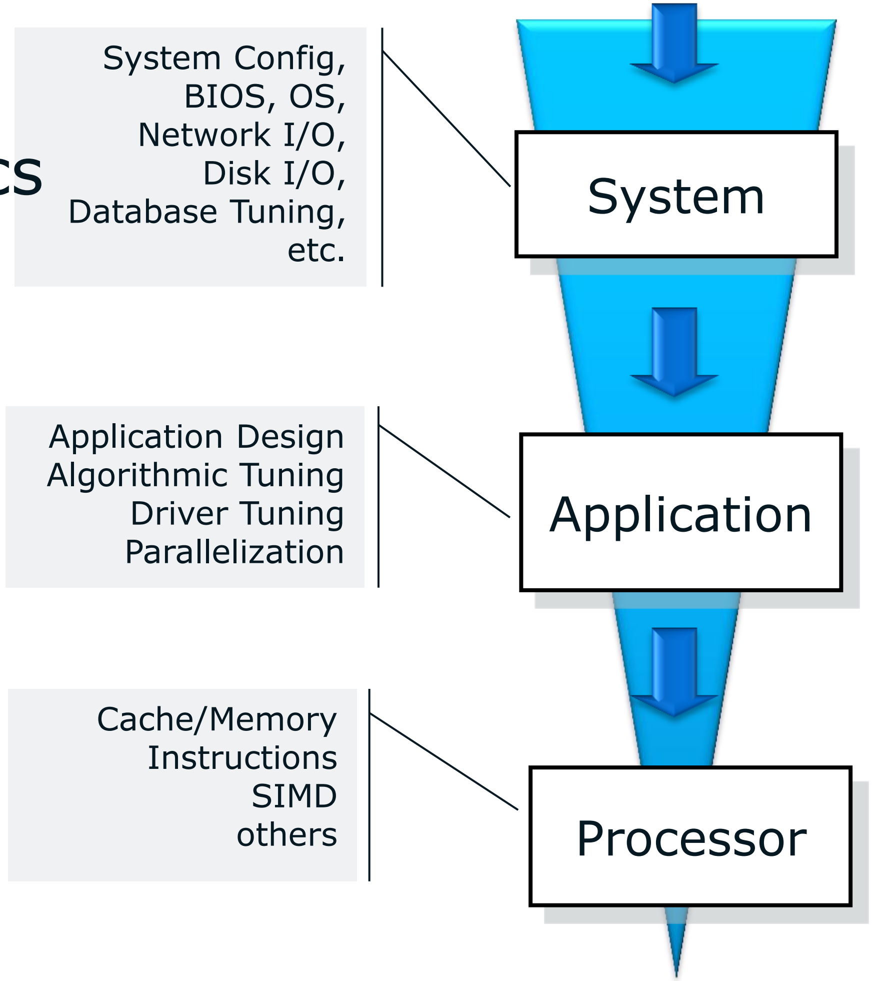
Efficiency metrics

Problem areas

Performance Analysis Methodology

Optimization: A Top-down Approach

- Use top down approach
- Understand application and system characteristics
  - Use appropriate tools at each level



System Config, BIOS, OS, Network I/O, Disk I/O, Database Tuning, etc.

System

Application Design
Algorithmic Tuning
Driver Tuning
Parallelization

Application

Cache/Memory
Instructions
SIMD
others

Processor

Optimization Notice

Performance Analysis Methodology

Optimization: A Top-down Approach

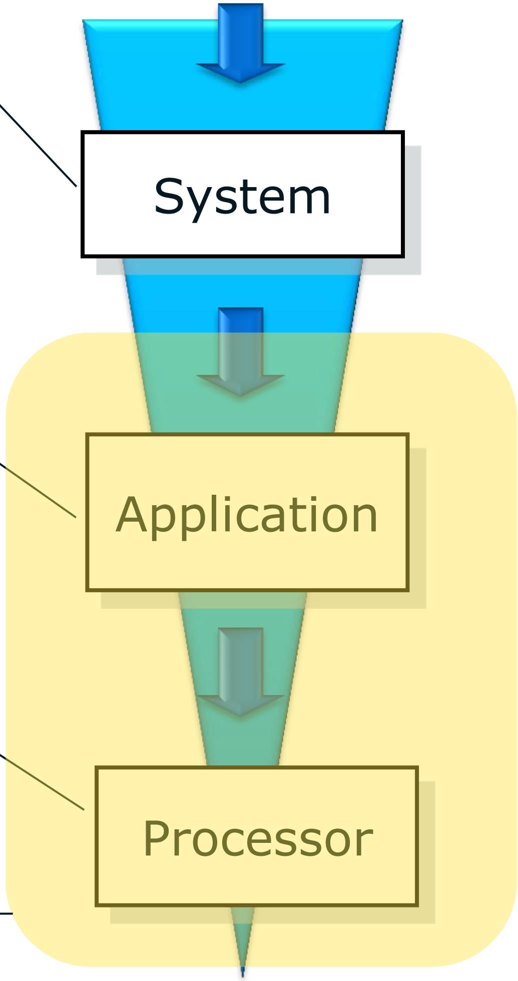
- Use top down approach
- Understand application and system characteristics
  - Use appropriate tools at each level

System Config,
BIOS, OS,
Network I/O,
Disk I/O,
Database Tuning,
etc.

Application Design
Algorithmic Tuning
Driver Tuning
Parallelization

Cache/Memory
Instructions
SIMD
others

VTune™ Amplifier
XE can help here



Optimization Notice

Start with host-based profiling to identify vectorization/ parallelism/ offload candidates

Start with representative/reasonable workloads!

Use Intel® VTune™ Amplifier XE to gather hot spot data

- Tells what functions account for most of the run time
- Often, this is enough
  - But it does not tell you much about program structure

Start with host-based profiling to identify vectorization/ parallelism/ offload candidates

Start with representative/reasonable workloads!

Use Intel® VTune™ Amplifier XE to gather hot spot data

- Tells what functions account for most of the run time
- Often, this is enough
  - But it does not tell you much about program structure

Alternately, profile functions & loops using Intel® Composer XE

- Build with options

```
  -profile-functions -profile-loops=all -profile-loops-
report=2
```

- Run the code (which may run slower) to collect profile data
- Look at the resulting `dump` files, or open the `xml` file with the data viewer `loopprofileviewer.sh` located in the compiler `./bin` directory
- Tells you

  which loops and functions account for the most run time
  how many times each loop executes (min, max and average)

Correctness/Performance Analysis of Parallel code

Intel® Inspector XE and thread-reports in VTune™ Amplifier XE are not available on the Intel® Xeon Phi™ coprocessor

So…

Correctness/Performance Analysis of Parallel code

Intel® Inspector XE and thread-reports in VTune™ Amplifier XE are not available on the Intel® Xeon Phi™ coprocessor

So…

- Use Intel Inspector XE on your code with **offload disabled** (on host) to identify correctness errors (e.g., deadlocks, races)
  - Once fixed, then enable offload and continue debugging on the coprocessor

Correctness/Performance Analysis of Parallel code

Intel® Inspector XE and thread-reports in VTune™ Amplifier XE are not available on the Intel® Xeon Phi™ coprocessor

So…

- Use Intel Inspector XE on your code with **offload disabled** (on host) to identify correctness errors (e.g., deadlocks, races)
  - Once fixed, then enable offload and continue debugging on the coprocessor
- Use VTune Amplifier XE's parallel performance analysis tools to find issues on the host by running your program with **offload disabled**
  - Fix everything you can
  - Then study scaling on the coprocessor using lessons from host tuning to further optimize parallel performance
    - Be wary of synchronization across more than a handful of threads
    - Pay attention to load balance.

Agenda

Start tuning on host

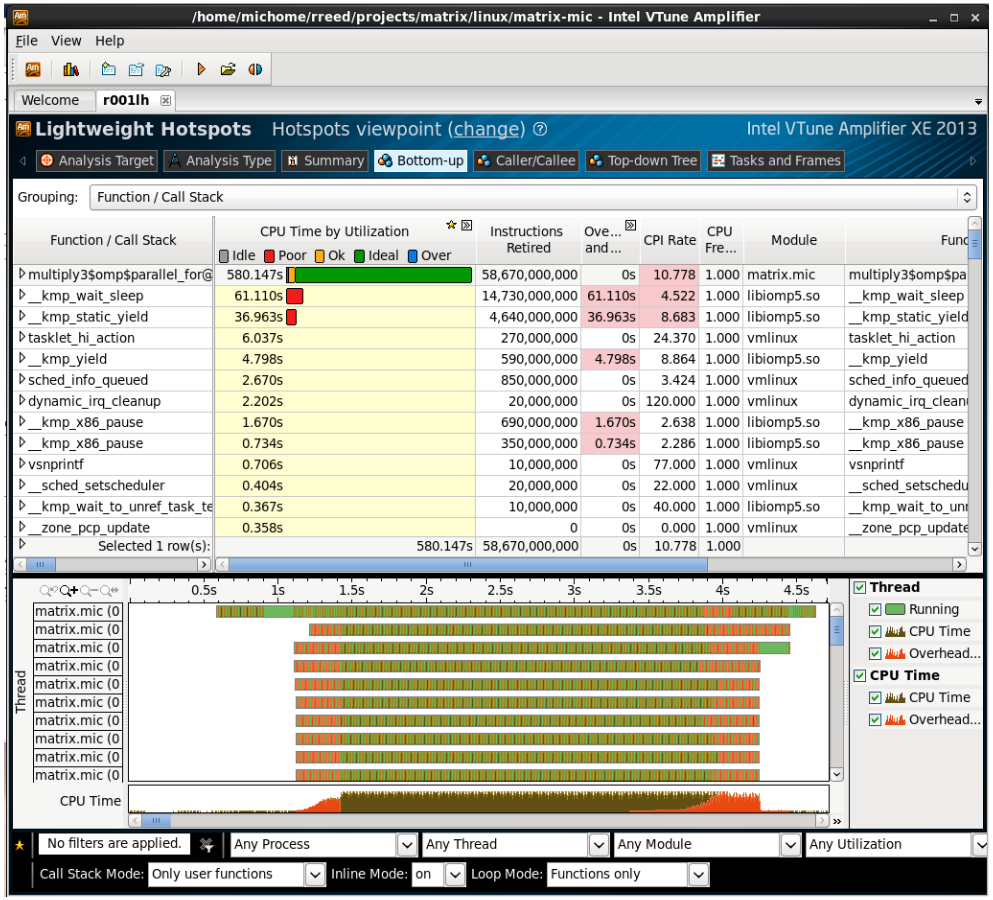
Overview of Intel® VTune™ Amplifier XE

Efficiency metrics

Problem areas

Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

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



Grouping: Function / Call Stack

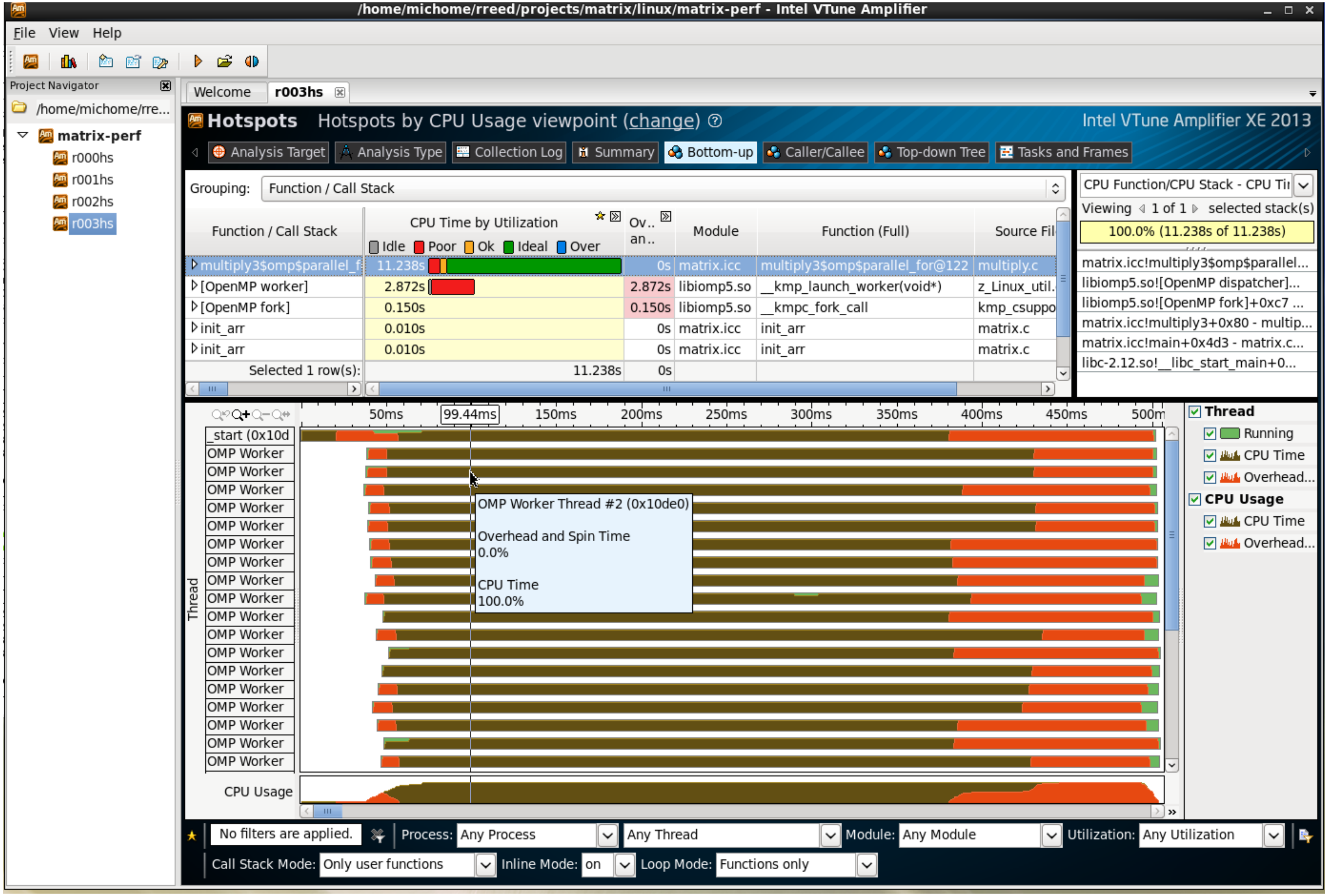
Function / Call Stack	CPU Time by Utilization ▢ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Instructions Retired	Ove... and...	CPI Rate	CPU Fre...	Module	Func
▷ multiply3$omp$parallel_for@	580.147s	58,670,000,000	0s	10.778	1.000	matrix.mic	multiply3$omp$pa
▷ __kmp_wait_sleep	61.110s	14,730,000,000	61.110s	4.522	1.000	libiomp5.so	__kmp_wait_sleep
▷ __kmp_static_yield	36.963s	4,640,000,000	36.963s	8.683	1.000	libiomp5.so	__kmp_static_yield
▷ tasklet_hi_action	6.037s	270,000,000	0s	24.370	1.000	vmlinux	tasklet_hi_action
▷ __kmp_yield	4.798s	590,000,000	4.798s	8.864	1.000	libiomp5.so	__kmp_yield
▷ sched_info_queued	2.670s	850,000,000	0s	3.424	1.000	vmlinux	sched_info_queued
▷ dynamic_irq_cleanup	2.202s	20,000,000	0s	120.000	1.000	vmlinux	dynamic_irq_clean
▷ __kmp_x86_pause	1.670s	690,000,000	1.670s	2.638	1.000	libiomp5.so	__kmp_x86_pause
▷ __kmp_x86_pause	0.734s	350,000,000	0.734s	2.286	1.000	libiomp5.so	__kmp_x86_pause
▷ vsnprintf	0.706s	10,000,000	0s	77.000	1.000	vmlinux	vsnprintf
▷ __sched_setscheduler	0.404s	20,000,000	0s	22.000	1.000	vmlinux	__sched_setschedu
▷ __kmp_wait_to_unref_task_te	0.367s	10,000,000	0s	40.000	1.000	libiomp5.so	__kmp_wait_to_unr
▷ __zone_pcp_update	0.358s	0	0s	0.000	1.000	vmlinux	__zone_pcp_update
▷ Selected 1 row(s):	580.147s	58,670,000,000	0s	10.778	1.000		

No filters are applied. Any Process Any Thread Any Module Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

[1] IA-32 and Intel® 64 architectures.
Many features work with compatible processors.
Event based sampling requires a genuine Intel Processor.

VTune™ Amplifier XE visualizes performance



Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	Module	Function (Full)	Source Fil
multiply3$omp$parallel_f	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

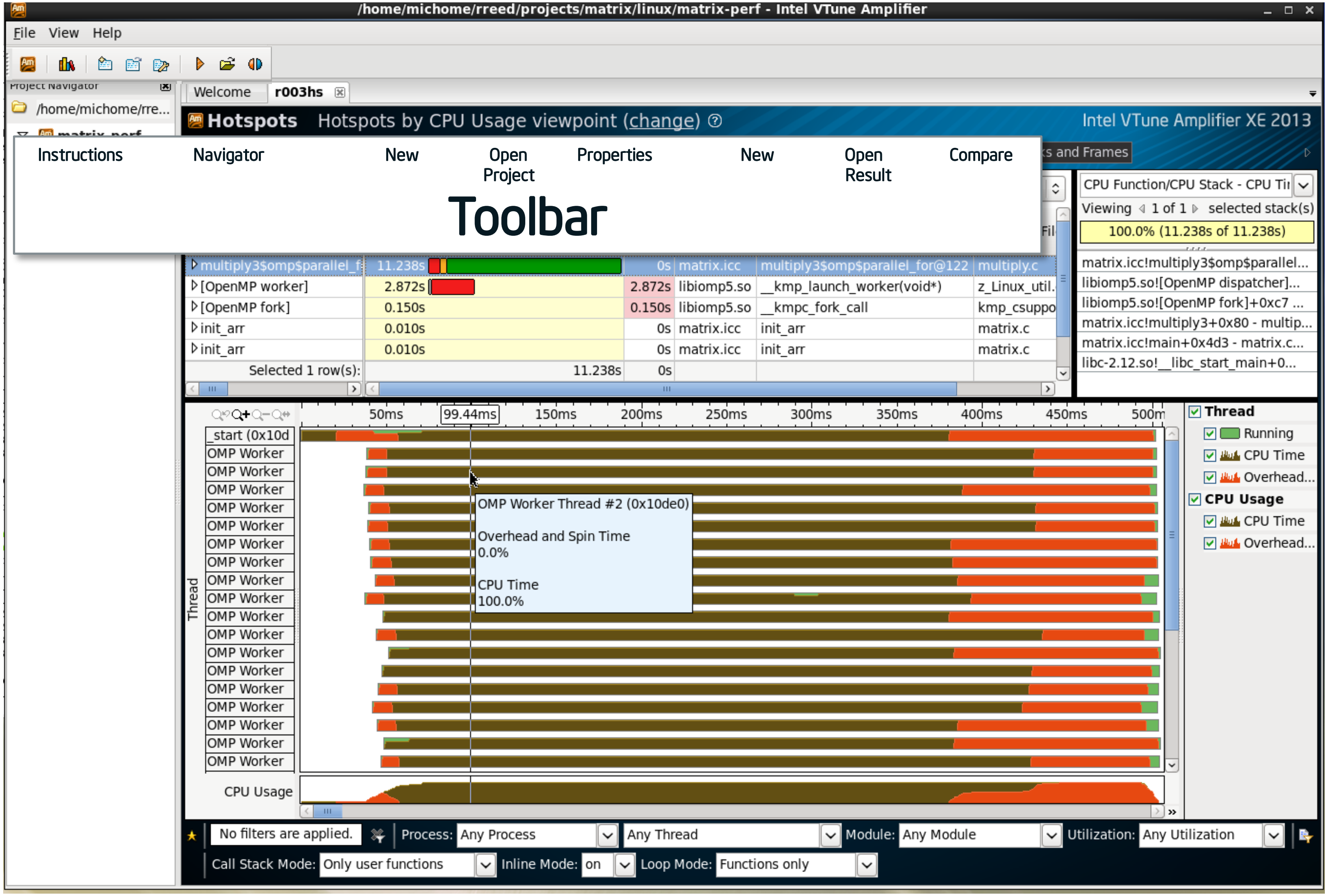
CPU Time
100.0%

No filters are applied. Process: Any Process; Any Thread; Module: Any Module; Utilization: Any Utilization

Call Stack Mode: Only user functions; Inline Mode: on; Loop Mode: Functions only

Optimization Notice

VTune™ Amplifier XE visualizes performance


/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier
File View Help
Project Navigator
/home/michome/rre...
matrix_perf
Welcome
r003hs
Hotspots Hotspots by CPU Usage viewpoint (change)
Intel VTune Amplifier XE 2013
s and Frames
CPU Function/CPU Stack - CPU Ti
Viewing 1 of 1 selected stack(s)
Thread
CPU Usage

Instructions Navigator New Open Project Properties New Open Result Compare

Toolbar

multiply3$omp$parallel_f	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

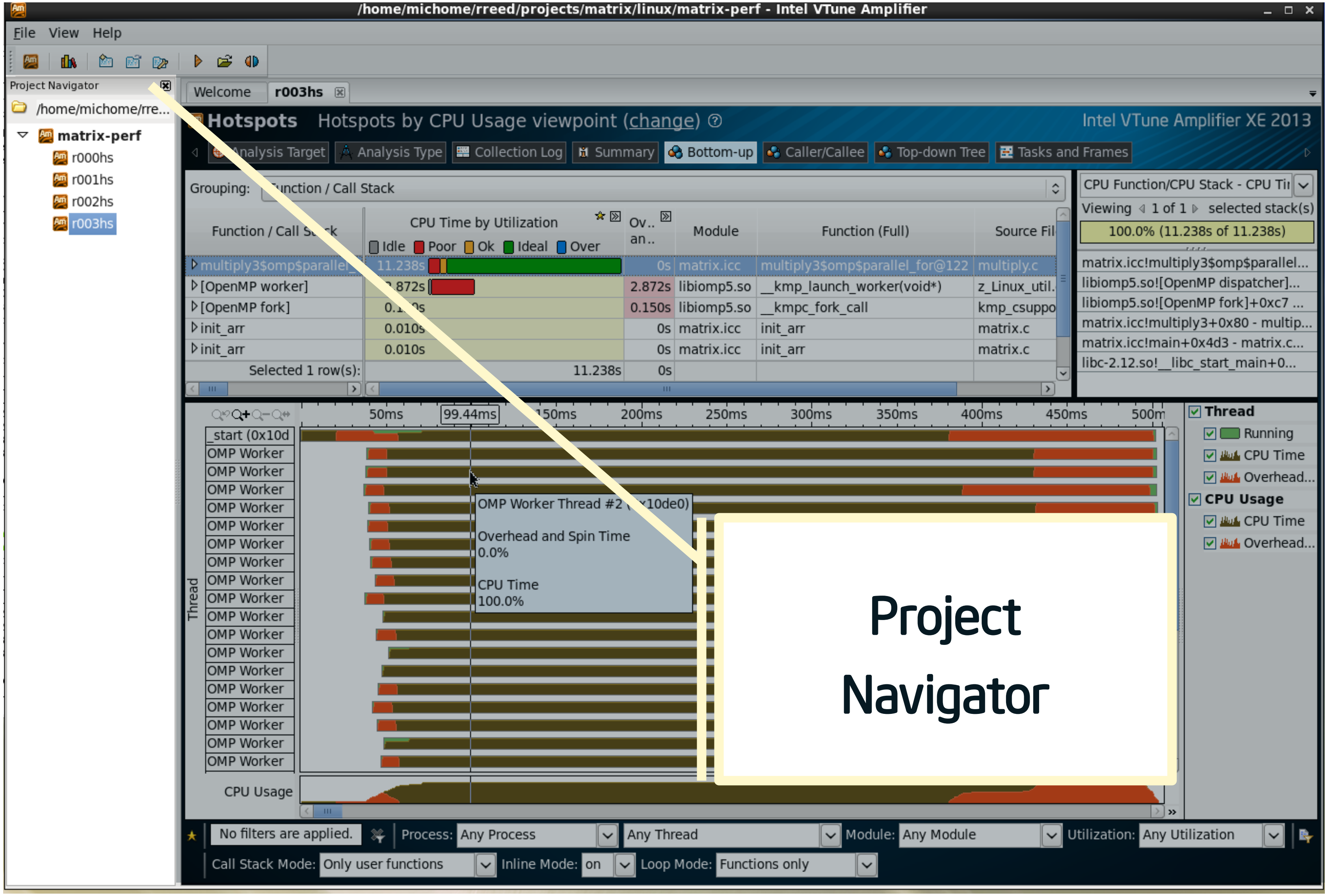
CPU Time
100.0%

No filters are applied. Process: Any Process Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Optimization Notice

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier

File View Help

Project Navigator

/home/michome/rre...

- matrix-perf
  - r000hs
  - r001hs
  - r002hs
  - r003hs

Welcome | r003hs

Hotspots Hotspots by CPU Usage viewpoint (change) Intel VTune Amplifier XE 2013

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames

Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	Module	Function (Full)	Source Fil
multiply3$omp$parallel	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

CPU Time
100.0%

☑ Thread
☑ Running
☑ CPU Time
☑ Overhead...
☑ CPU Usage
☑ CPU Time
☑ Overhead...

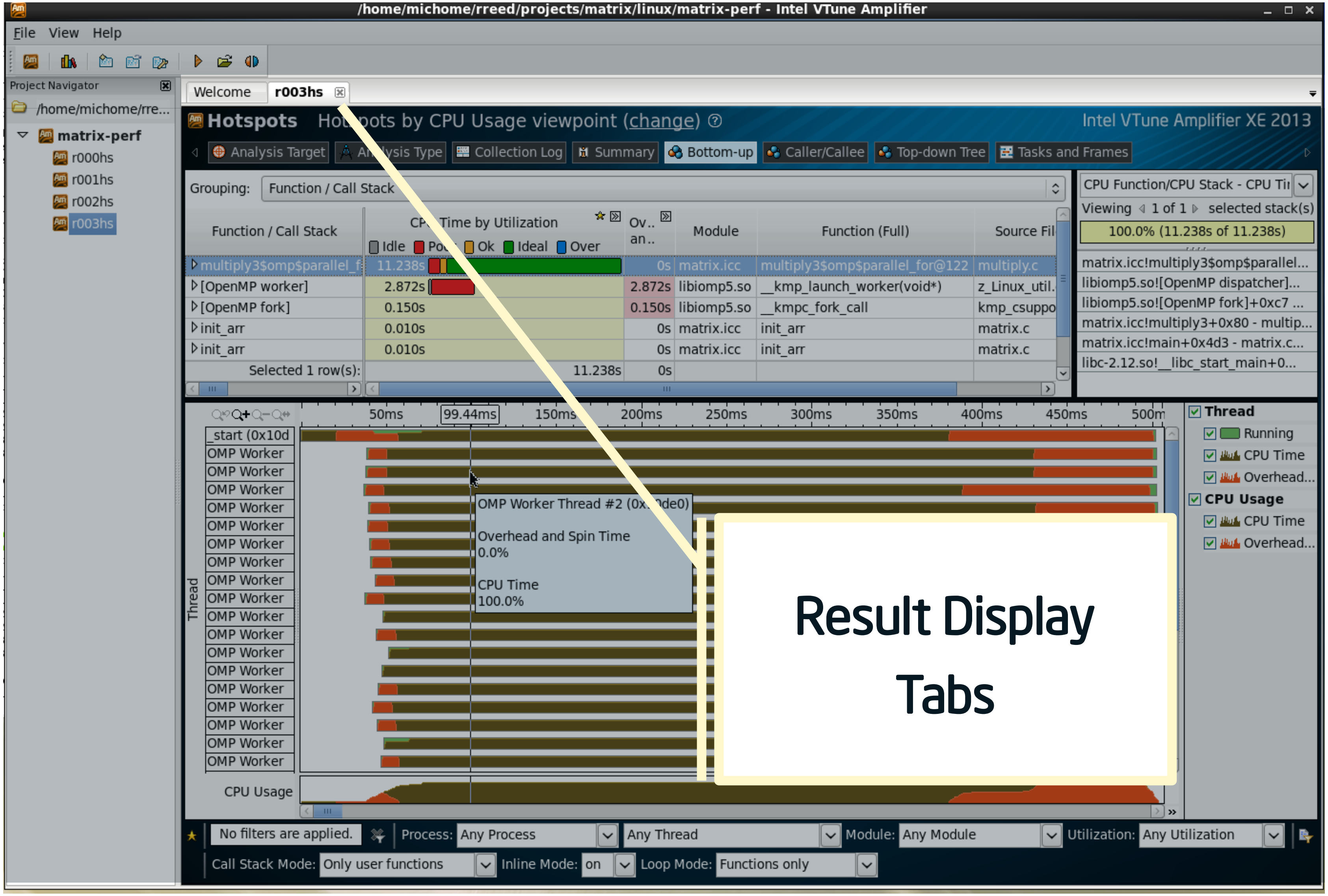
No filters are applied. Process: Any Process | Any Thread | Module: Any Module | Utilization: Any Utilization

Call Stack Mode: Only user functions | Inline Mode: on | Loop Mode: Functions only

Project Navigator

Optimization Notice

VTune™ Amplifier XE visualizes performance



Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization Idle Poor Ok Ideal Over	Ov.. an..	Module	Function (Full)	Source File
multiply3$omp$parallel_f	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

_start (0x10d
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker
OMP Worker

OMP Worker Thread #2 (0x..de0)

Overhead and Spin Time
0.0%

CPU Time
100.0%

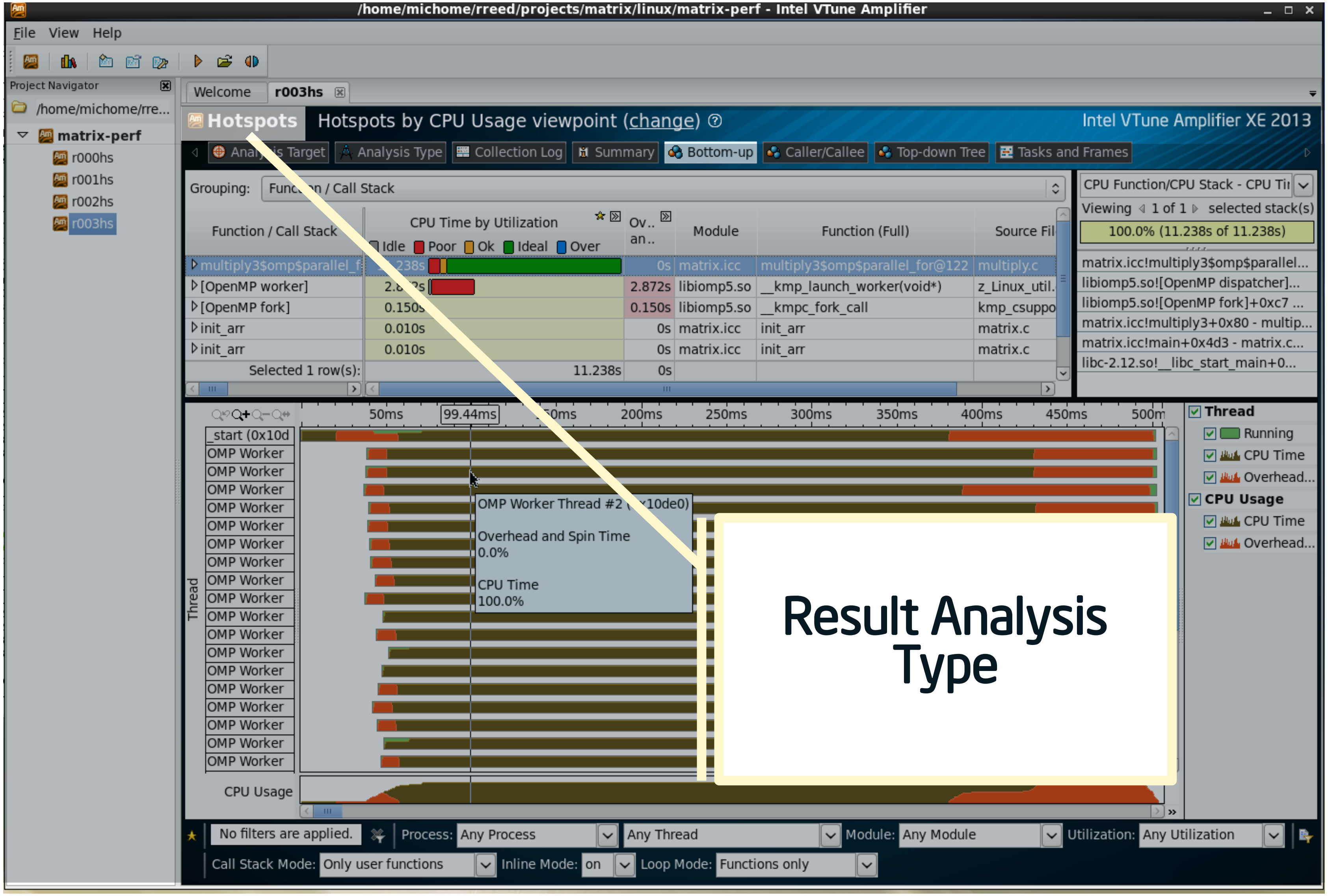
No filters are applied. Process: Any Process Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Result Display Tabs

Optimization Notice

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier

Hotspots by CPU Usage viewpoint (change) ⓘ

Intel VTune Amplifier XE 2013

Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	Module	Function (Full)	Source Fil
multiply3$omp$parallel_f	[illegible].238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.8[illegible]2s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Tir

Viewing ◁ 1 of 1 ▷ selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

CPU Time
100.0%

No filters are applied.

Process: Any Process

Any Thread

Module: Any Module

Utilization: Any Utilization

Call Stack Mode: Only user functions

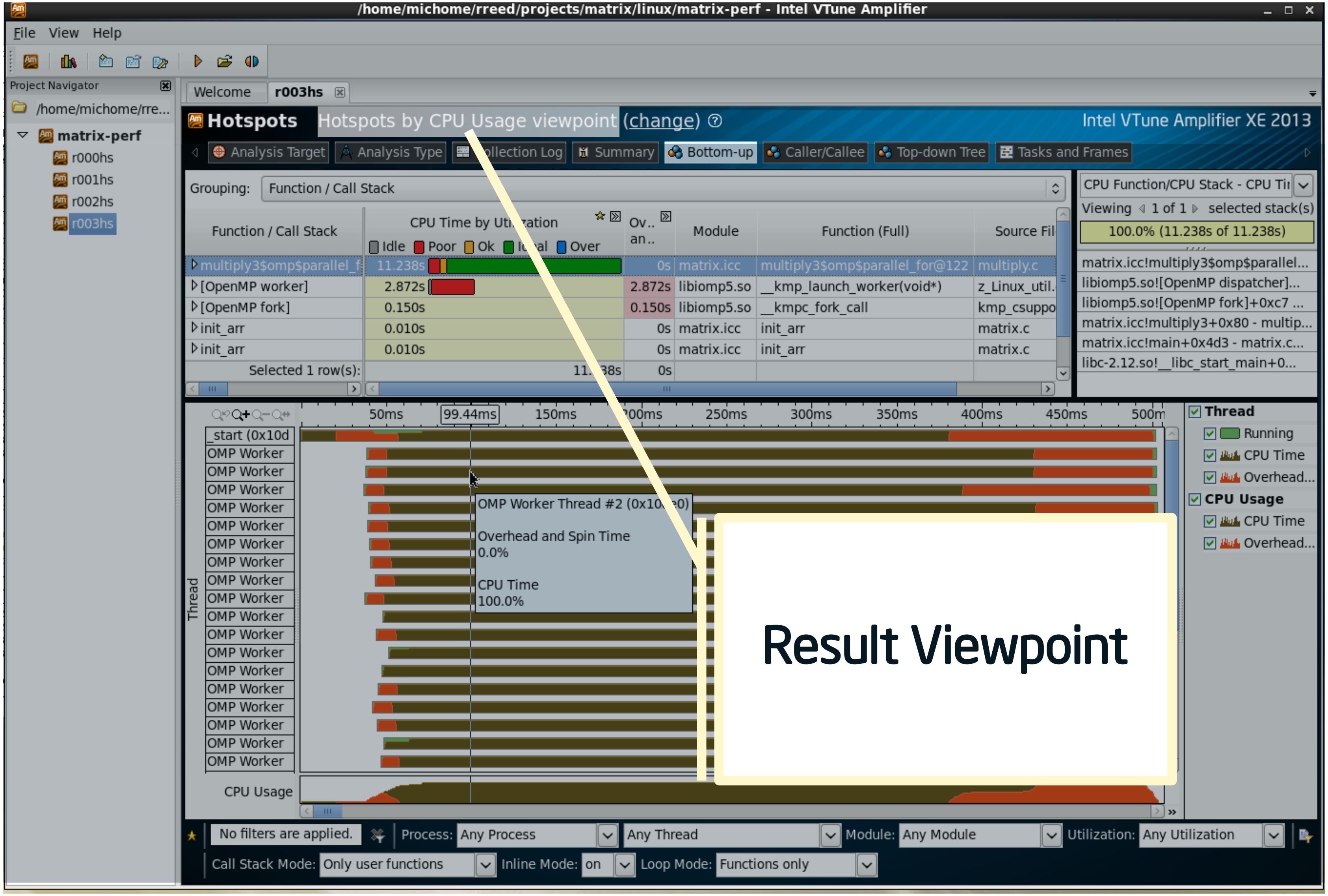
Inline Mode: on

Loop Mode: Functions only

Result Analysis Type

Optimization Notice

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier

Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	Module	Function (Full)	Source Fil
multiply3$omp$parallel_f	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10?0)

Overhead and Spin Time
0.0%

CPU Time
100.0%

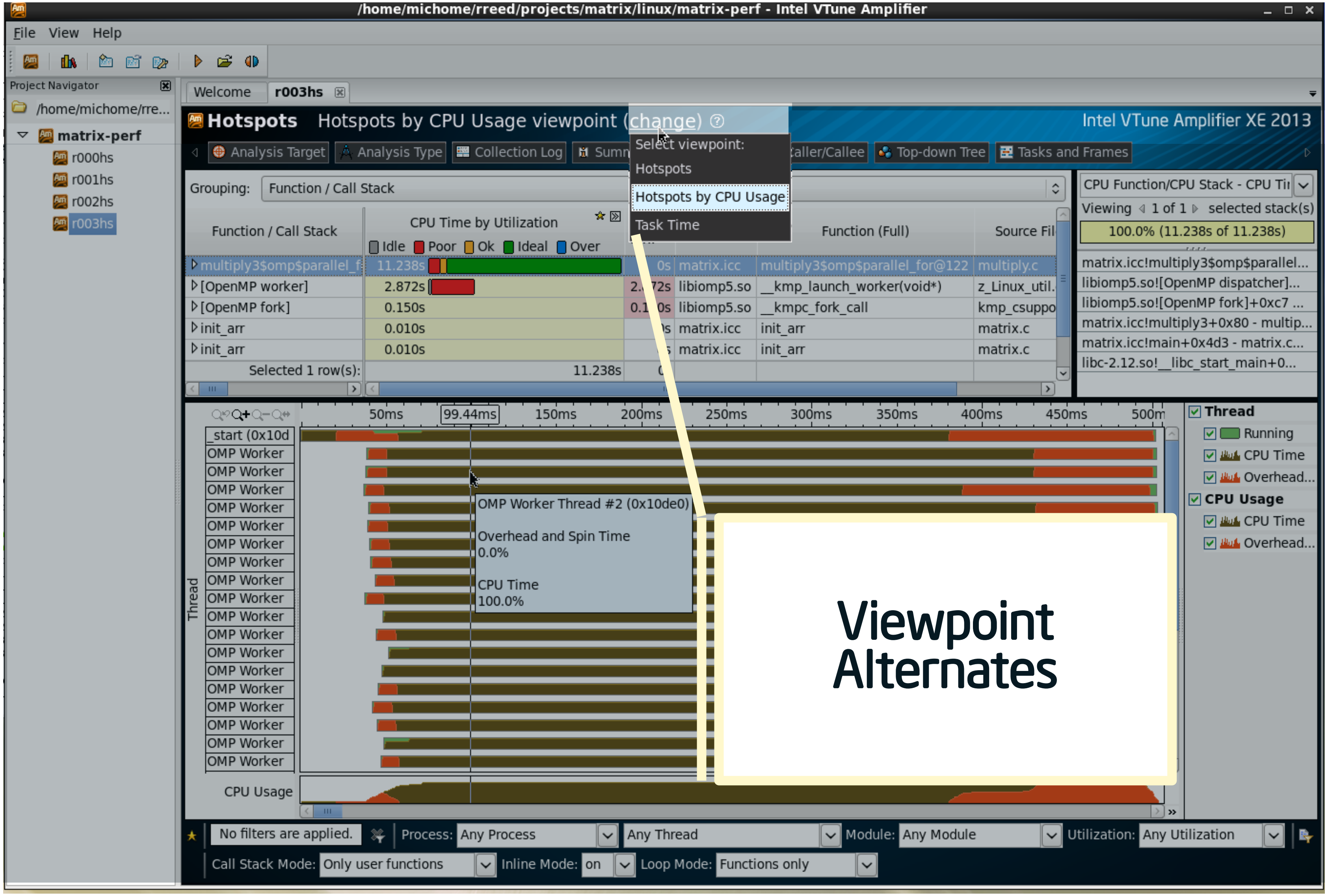
No filters are applied. Process: Any Process Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Result Viewpoint

Optimization Notice

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier

Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over			Function (Full)	Source Fil
multiply3$omp$parallel_f	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2. 72s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.1 0s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0			

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

CPU Time
100.0%

No filters are applied.

Process: Any Process

Any Thread

Module: Any Module

Utilization: Any Utilization

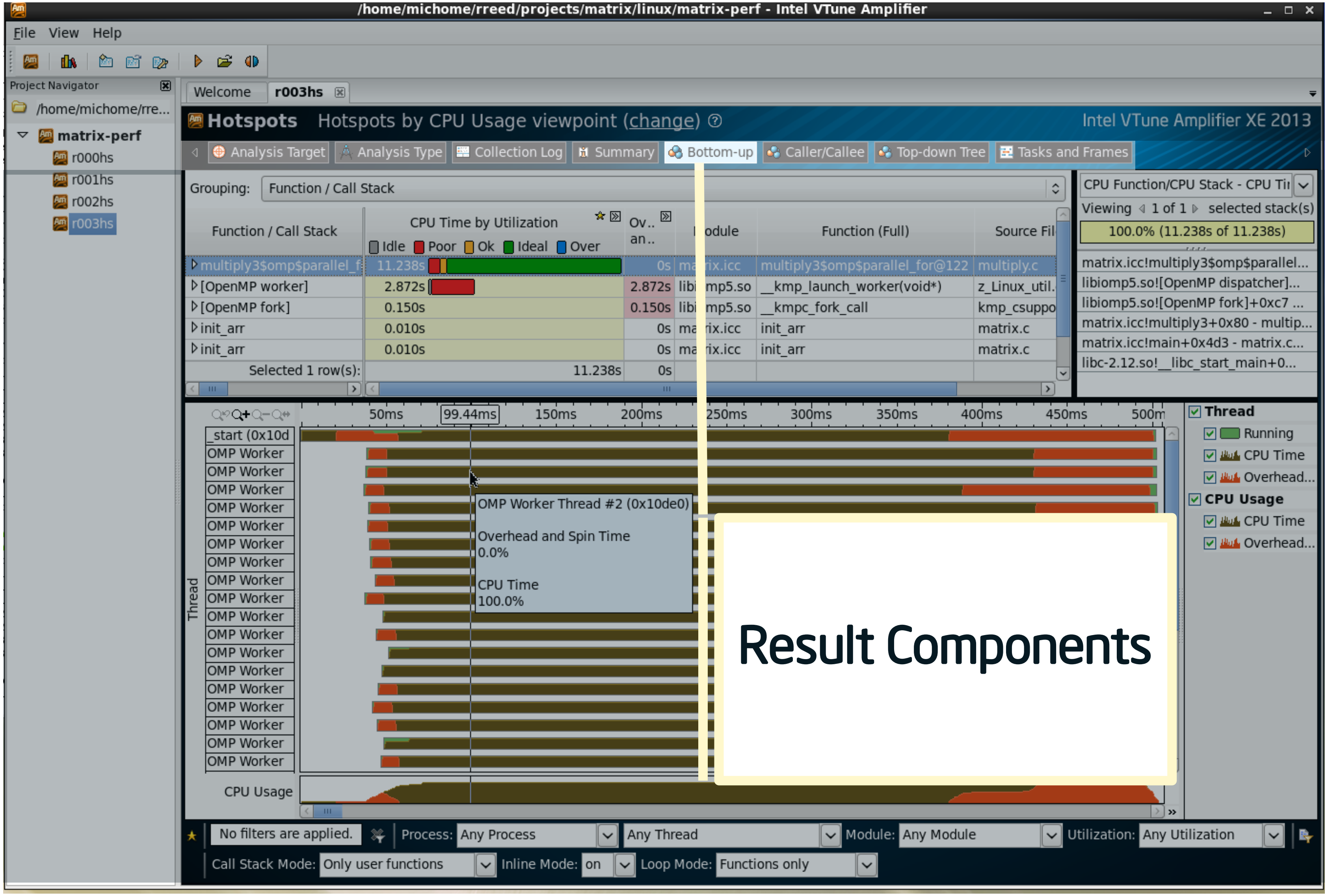
Call Stack Mode: Only user functions

Inline Mode: on

Loop Mode: Functions only

Viewpoint Alternates

VTune™ Amplifier XE visualizes performance



Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	odule	Function (Full)	Source Fil
multiply3$omp$parallel_f	11.238s	0s	ma ix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2.872s	libi mp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libi mp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	ma ix.icc	init_arr	matrix.c
init_arr	0.010s	0s	ma ix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

- matrix.icc!multiply3$omp$parallel...
- libiomp5.so![OpenMP dispatcher]...
- libiomp5.so![OpenMP fork]+0xc7 ...
- matrix.icc!multiply3+0x80 - multip...
- matrix.icc!main+0x4d3 - matrix.c...
- libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

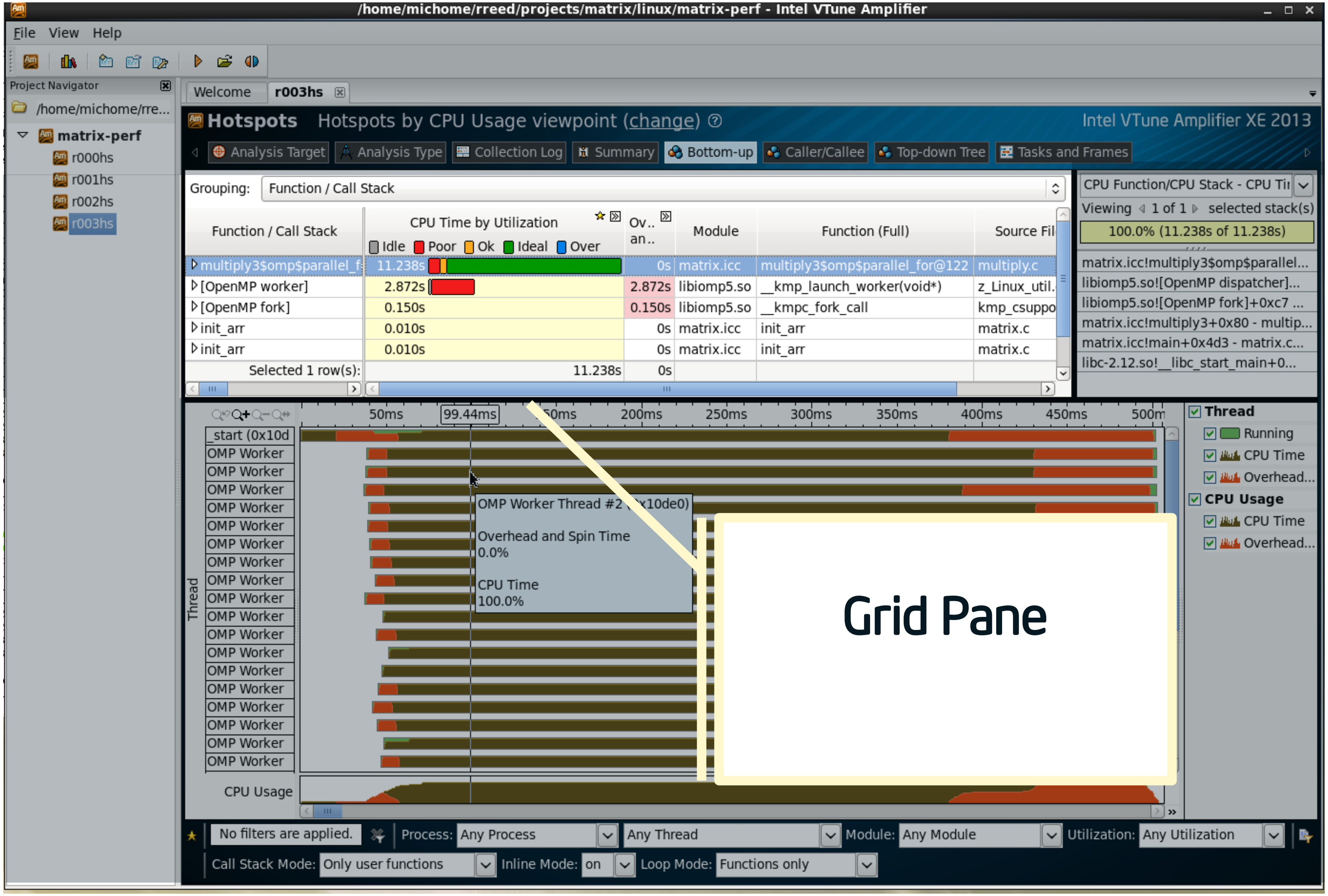
CPU Time
100.0%

Result Components

No filters are applied. Process: Any Process | Any Thread | Module: Any Module | Utilization: Any Utilization

Call Stack Mode: Only user functions | Inline Mode: on | Loop Mode: Functions only

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier

Hotspots by CPU Usage viewpoint (change)

Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	Module	Function (Full)	Source Fil
multiply3$omp$parallel_f	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
[OpenMP worker]	2.872s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
[OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

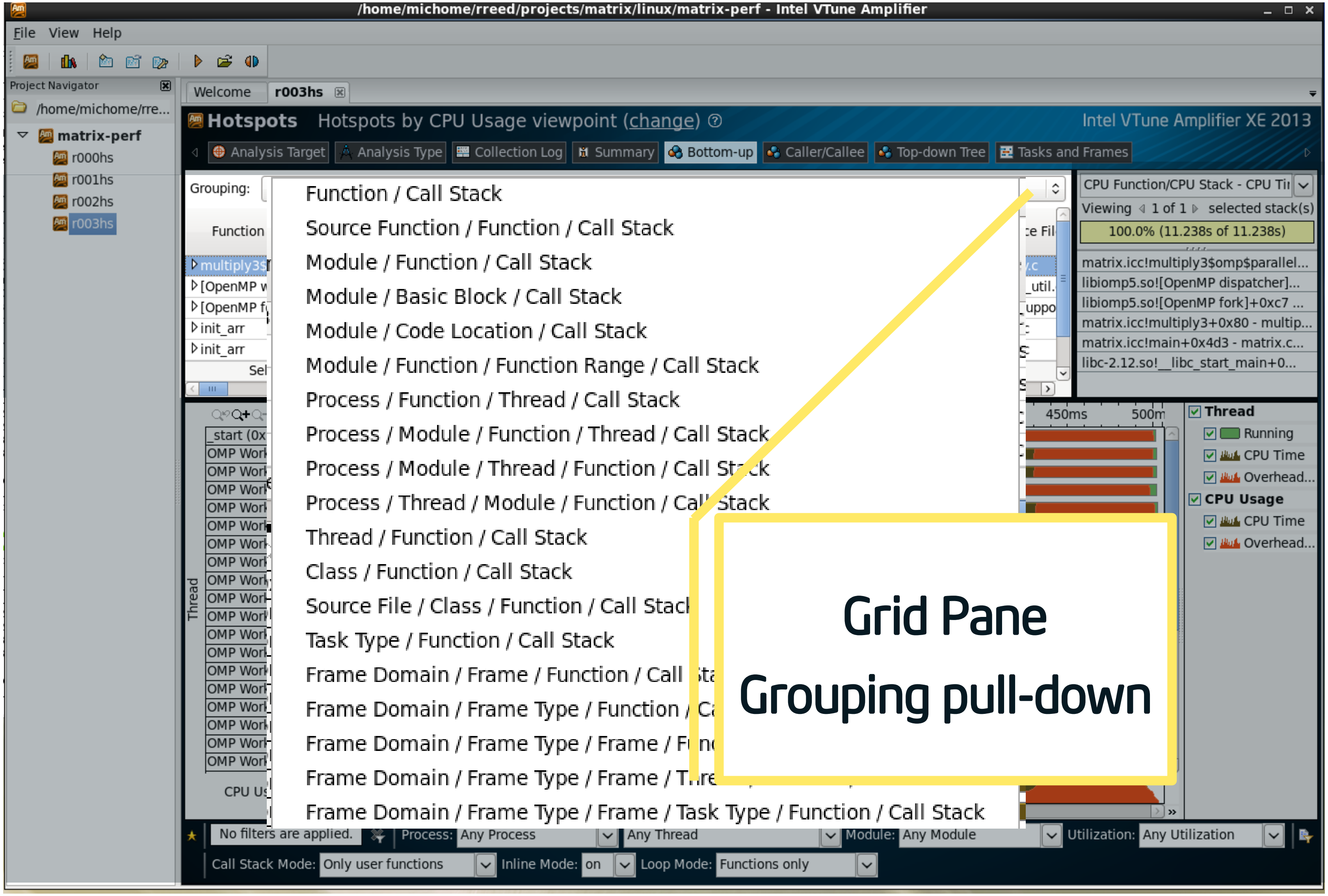
CPU Time
100.0%

No filters are applied. Process: Any Process Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Grid Pane

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier

Hotspots Hotspots by CPU Usage viewpoint (change) Intel VTune Amplifier XE 2013

Grouping:

- Function / Call Stack
- Source Function / Function / Call Stack
- Module / Function / Call Stack
- Module / Basic Block / Call Stack
- Module / Code Location / Call Stack
- Module / Function / Function Range / Call Stack
- Process / Function / Thread / Call Stack
- Process / Module / Function / Thread / Call Stack
- Process / Module / Thread / Function / Call Stack
- Process / Thread / Module / Function / Call Stack
- Thread / Function / Call Stack
- Class / Function / Call Stack
- Source File / Class / Function / Call Stack
- Task Type / Function / Call Stack
- Frame Domain / Frame / Function / Call Sta
- Frame Domain / Frame Type / Function / Ca
- Frame Domain / Frame Type / Frame / Fun
- Frame Domain / Frame Type / Frame / Thre
- Frame Domain / Frame Type / Frame / Task Type / Function / Call Stack

CPU Function/CPU Stack - CPU Ti

Viewing 1 of 1 selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

No filters are applied. Process: Any Process Any Thread Module: Any Module Utilization: Any Utilization

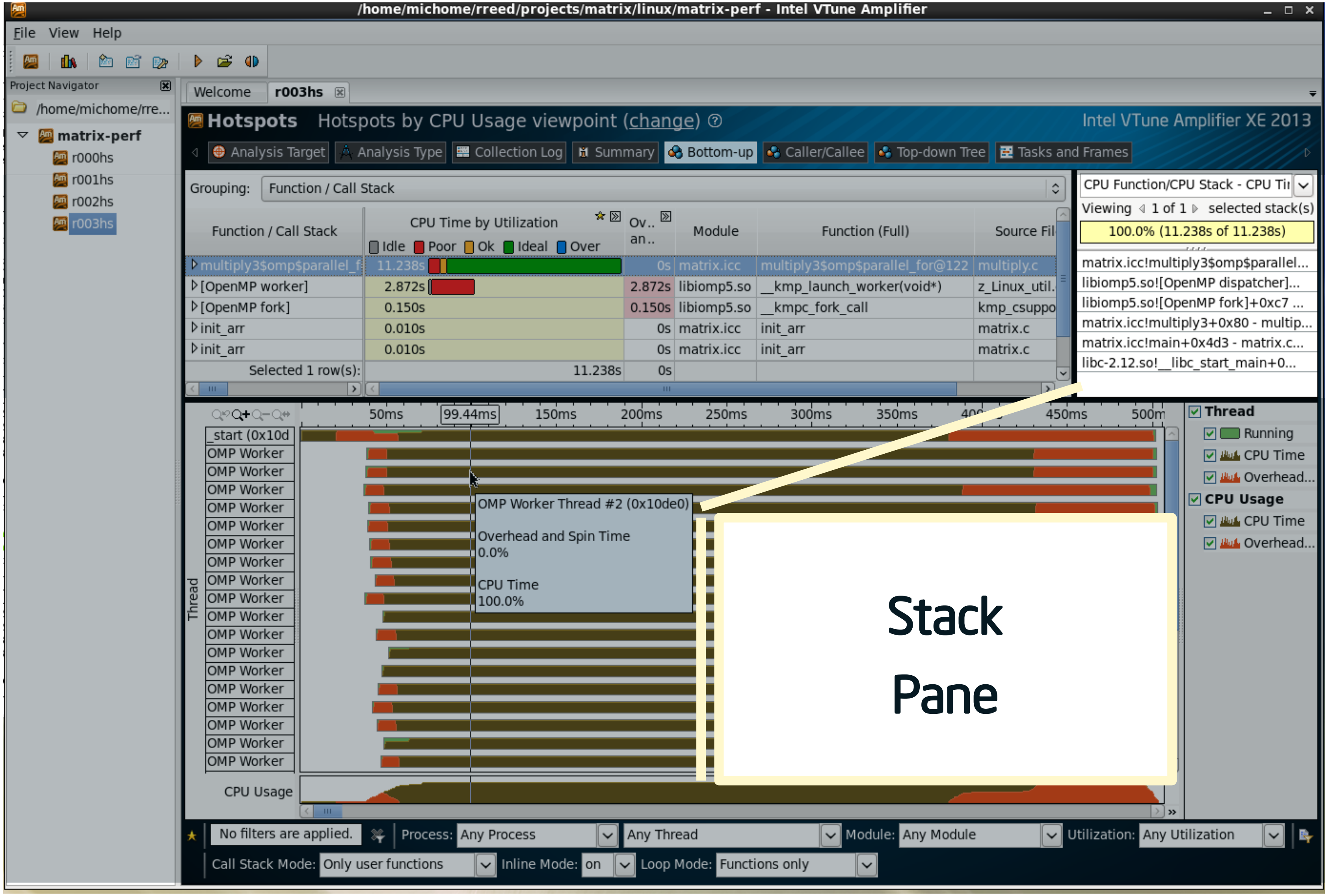
Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Grid Pane

Grouping pull-down

Optimization Notice

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-perf - Intel VTune Amplifier

Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization ▢ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	Module	Function (Full)	Source Fil
▷ multiply3$omp$parallel_f	11.238s	0s	matrix.icc	multiply3$omp$parallel_for@122	multiply.c
▷ [OpenMP worker]	2.872s	2.872s	libiomp5.so	__kmp_launch_worker(void*)	z_Linux_util.
▷ [OpenMP fork]	0.150s	0.150s	libiomp5.so	__kmpc_fork_call	kmp_csuppo
▷ init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
▷ init_arr	0.010s	0s	matrix.icc	init_arr	matrix.c
Selected 1 row(s):	11.238s	0s			

CPU Function/CPU Stack - CPU Ti

Viewing ◁ 1 of 1 ▷ selected stack(s)

100.0% (11.238s of 11.238s)

matrix.icc!multiply3$omp$parallel...
libiomp5.so![OpenMP dispatcher]...
libiomp5.so![OpenMP fork]+0xc7 ...
matrix.icc!multiply3+0x80 - multip...
matrix.icc!main+0x4d3 - matrix.c...
libc-2.12.so!__libc_start_main+0...

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

CPU Time
100.0%

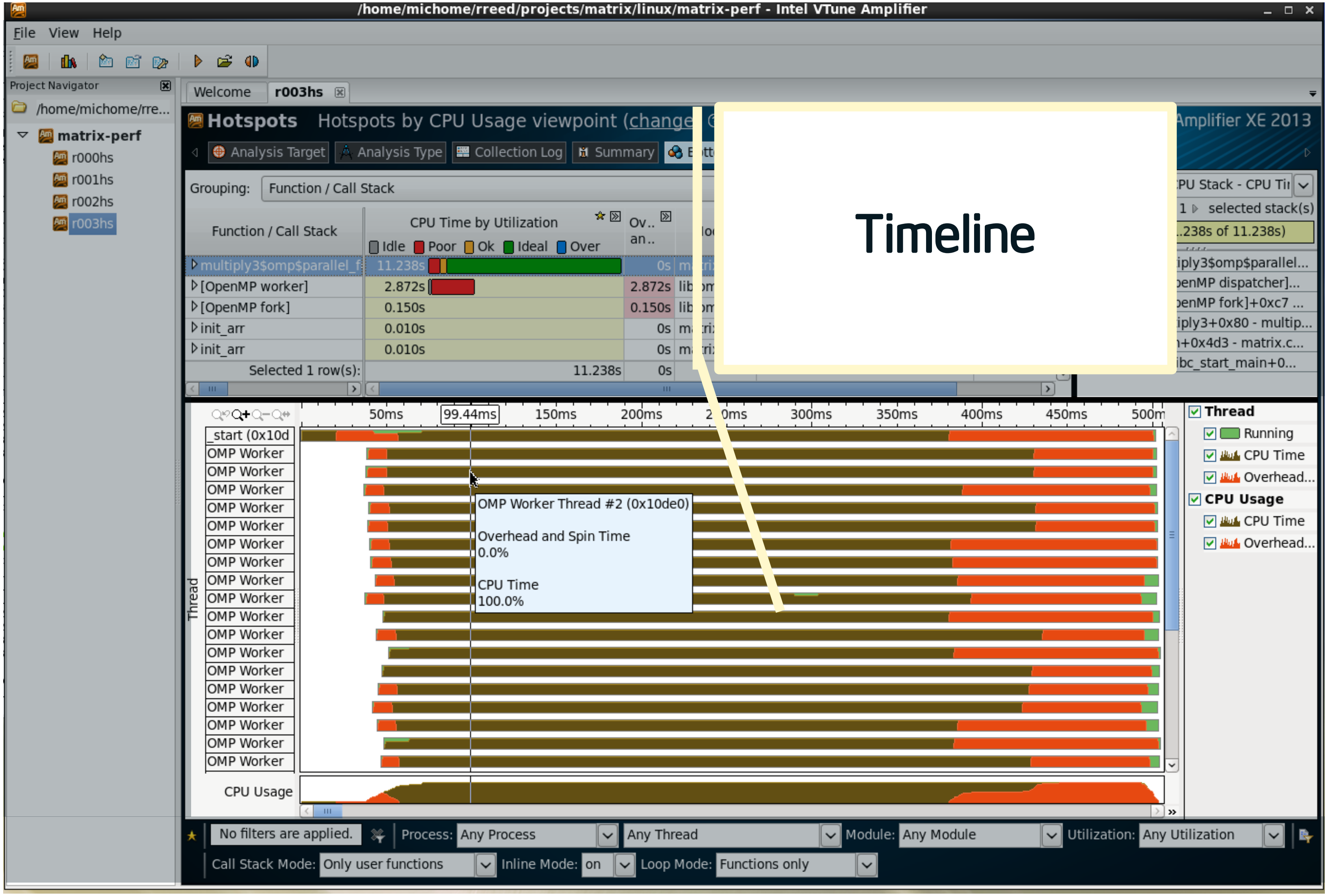
Stack Pane

No filters are applied. Process: Any Process Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Optimization Notice

VTune™ Amplifier XE visualizes performance



Timeline

Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	
multiply3$omp$parallel_f	11.238s	0s	m trix
[OpenMP worker]	2.872s	2.872s	lib m
[OpenMP fork]	0.150s	0.150s	lib m
init_arr	0.010s	0s	m trix
init_arr	0.010s	0s	m trix
Selected 1 row(s):	11.238s	0s	

OMP Worker Thread #2 (0x10de0)

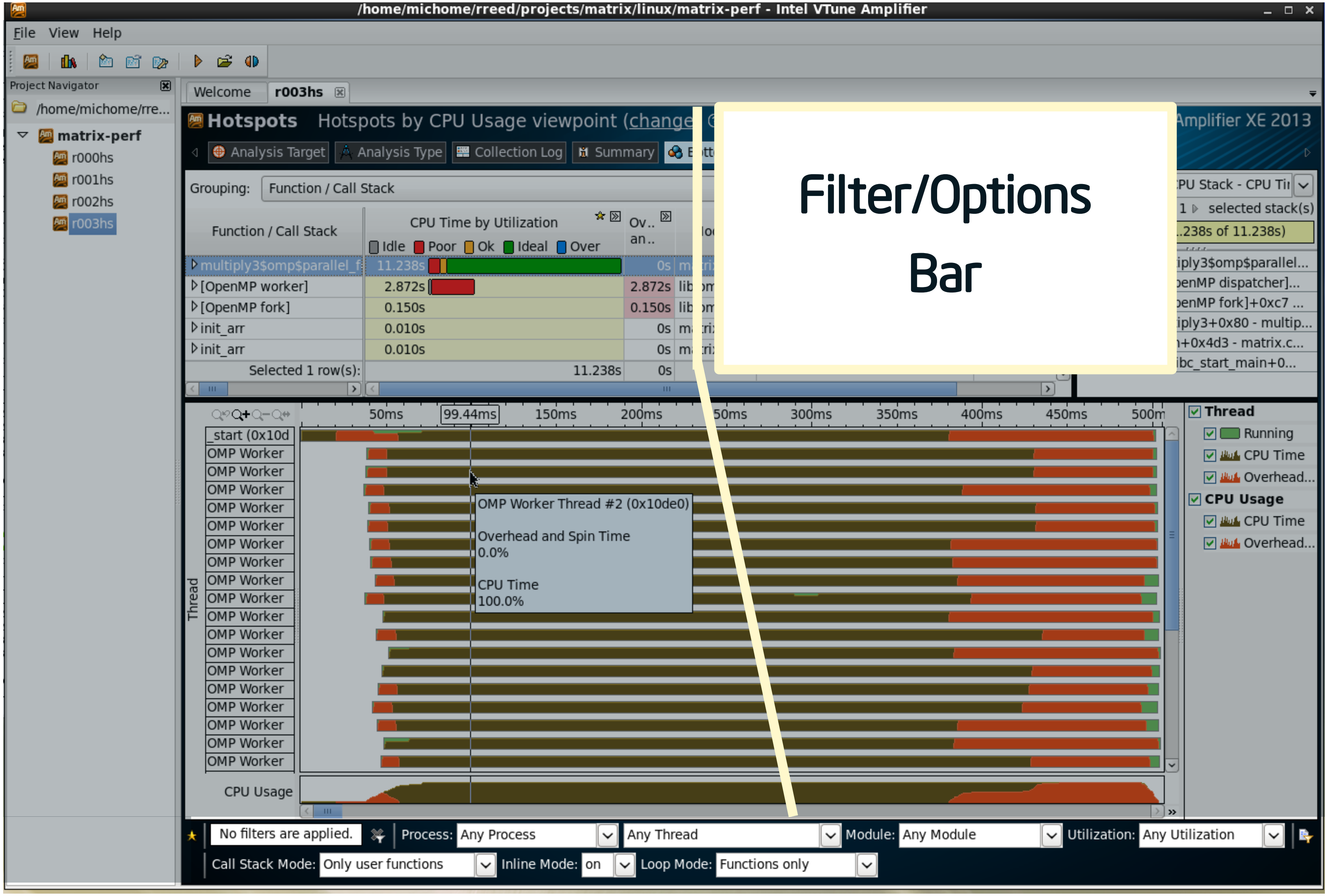
Overhead and Spin Time
0.0%

CPU Time
100.0%

No filters are applied. Process: Any Process | Any Thread | Module: Any Module | Utilization: Any Utilization

Call Stack Mode: Only user functions | Inline Mode: on | Loop Mode: Functions only

VTune™ Amplifier XE visualizes performance



Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization □ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov.. an..	lo
▹ multiply3$omp$parallel_f	11.238s	0s	matrix
▹ [OpenMP worker]	2.872s	2.872s	lib om
▹ [OpenMP fork]	0.150s	0.150s	lib om
▹ init_arr	0.010s	0s	ma rix
▹ init_arr	0.010s	0s	ma rix
Selected 1 row(s):	11.238s	0s	

OMP Worker Thread #2 (0x10de0)

Overhead and Spin Time
0.0%

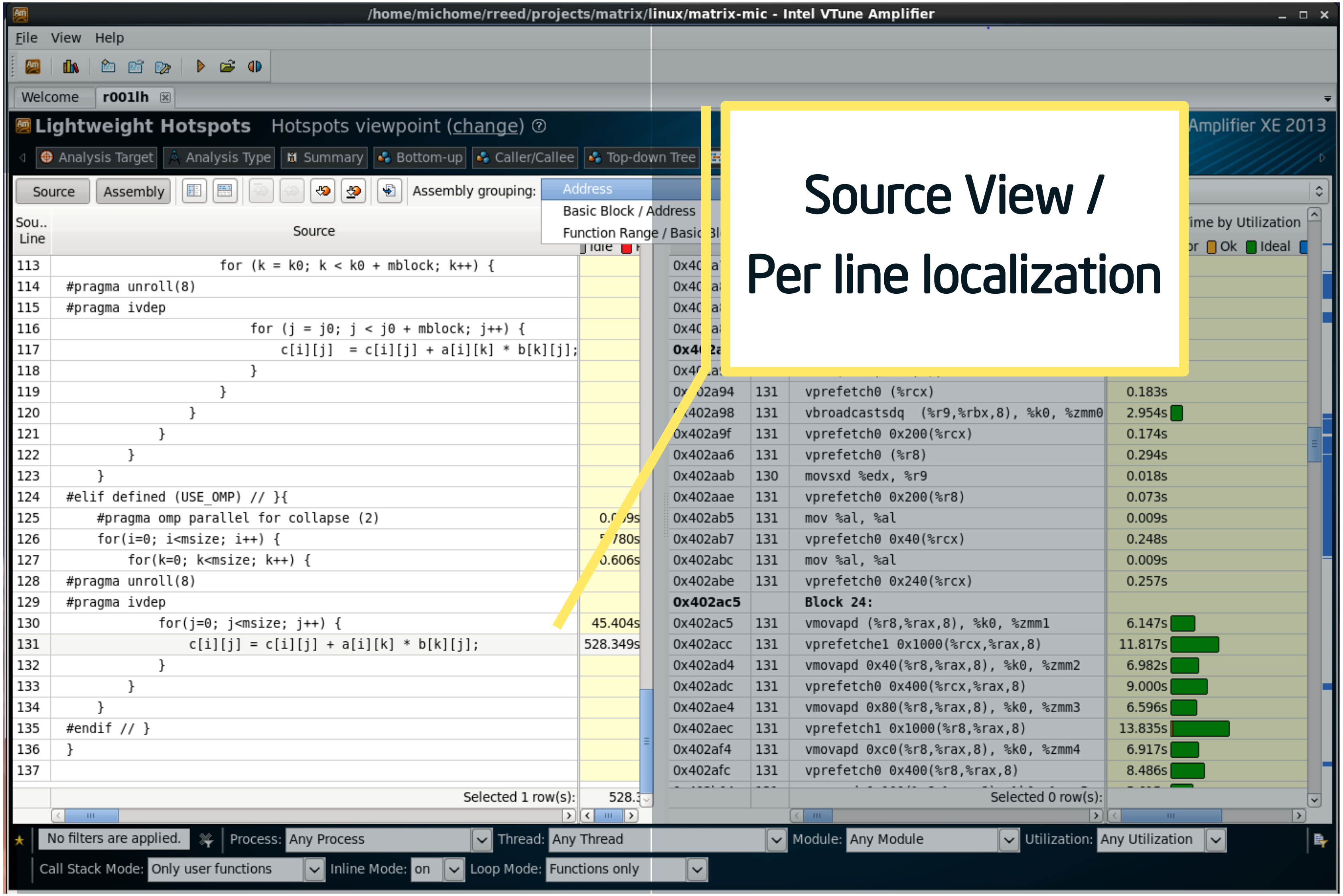
CPU Time
100.0%

No filters are applied. Process: Any Process Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Filter/Options Bar

VTune™ Amplifier XE visualizes performance



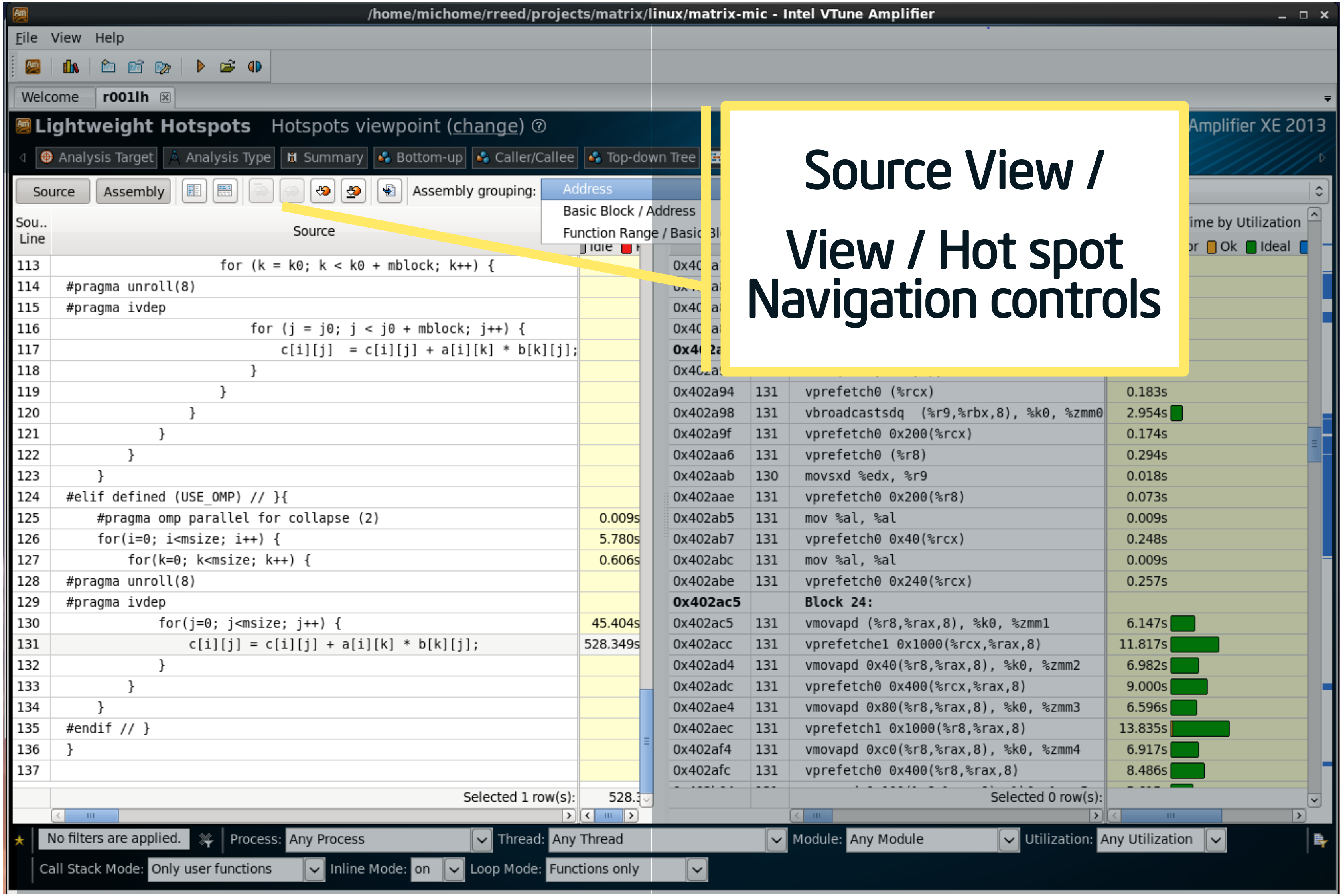
Sou.. Line	Source	Idle
113	`for (k = k0; k < k0 + mblock; k++) {`	
114	`#pragma unroll(8)`	
115	`#pragma ivdep`	
116	`for (j = j0; j < j0 + mblock; j++) {`	
117	`c[i][j]  = c[i][j] + a[i][k] * b[k][j];`	
118	`}`	
119	`}`	
120	`}`	
121	`}`	
122	`}`	
123	`}`	
124	`#elif defined (USE_OMP) // }{`	
125	`#pragma omp parallel for collapse (2)`	0.0 9s
126	`for(i=0; i<msize; i++) {`	780s
127	`for(k=0; k<msize; k++) {`	0.606s
128	`#pragma unroll(8)`	
129	`#pragma ivdep`	
130	`for(j=0; j<msize; j++) {`	45.404s
131	`c[i][j] = c[i][j] + a[i][k] * b[k][j];`	528.349s
132	`}`	
133	`}`	
134	`}`	
135	`#endif // }`	
136	`}`	
137		
	Selected 1 row(s):	528.3

			ime by Utilization
0x40 a			
0x40 a			
0x40 a			
0x40 a			
0x4(2a			
0x4(a			
0x 02a94	131	`vprefetch0 (%rcx)`	0.183s
402a98	131	`vbroadcastsdq  (%r9,%rbx,8), %k0, %zmm0`	2.954s 🟩
0x402a9f	131	`vprefetch0 0x200(%rcx)`	0.174s
0x402aa6	131	`vprefetch0 (%r8)`	0.294s
0x402aab	130	`movsxd %edx, %r9`	0.018s
0x402aae	131	`vprefetch0 0x200(%r8)`	0.073s
0x402ab5	131	`mov %al, %al`	0.009s
0x402ab7	131	`vprefetch0 0x40(%rcx)`	0.248s
0x402abc	131	`mov %al, %al`	0.009s
0x402abe	131	`vprefetch0 0x240(%rcx)`	0.257s
0x402ac5		**Block 24:**	
0x402ac5	131	`vmovapd (%r8,%rax,8), %k0, %zmm1`	6.147s 🟩
0x402acc	131	`vprefetche1 0x1000(%rcx,%rax,8)`	11.817s 🟩
0x402ad4	131	`vmovapd 0x40(%r8,%rax,8), %k0, %zmm2`	6.982s 🟩
0x402adc	131	`vprefetch0 0x400(%rcx,%rax,8)`	9.000s 🟩
0x402ae4	131	`vmovapd 0x80(%r8,%rax,8), %k0, %zmm3`	6.596s 🟩
0x402aec	131	`vprefetch1 0x1000(%r8,%rax,8)`	13.835s 🟩
0x402af4	131	`vmovapd 0xc0(%r8,%rax,8), %k0, %zmm4`	6.917s 🟩
0x402afc	131	`vprefetch0 0x400(%r8,%rax,8)`	8.486s 🟩
		Selected 0 row(s):	

Source View /
Per line localization

Optimization Notice

VTune™ Amplifier XE visualizes performance



/home/michome/rreed/projects/matrix/linux/matrix-mic - Intel VTune Amplifier

Assembly grouping: Address

Sou.. Line	Source	
113	for (k = k0; k < k0 + mblock; k++) {	
114	#pragma unroll(8)	
115	#pragma ivdep	
116	for (j = j0; j < j0 + mblock; j++) {	
117	c[i][j] = c[i][j] + a[i][k] * b[k][j];	
118	}	
119	}	
120	}	
121	}	
122	}	
123	}	
124	#elif defined (USE_OMP) // }{	
125	#pragma omp parallel for collapse (2)	0.009s
126	for(i=0; i<msize; i++) {	5.780s
127	for(k=0; k<msize; k++) {	0.606s
128	#pragma unroll(8)	
129	#pragma ivdep	
130	for(j=0; j<msize; j++) {	45.404s
131	c[i][j] = c[i][j] + a[i][k] * b[k][j];	528.349s
132	}	
133	}	
134	}	
135	#endif // }	
136	}	
137		
	Selected 1 row(s):	528.3

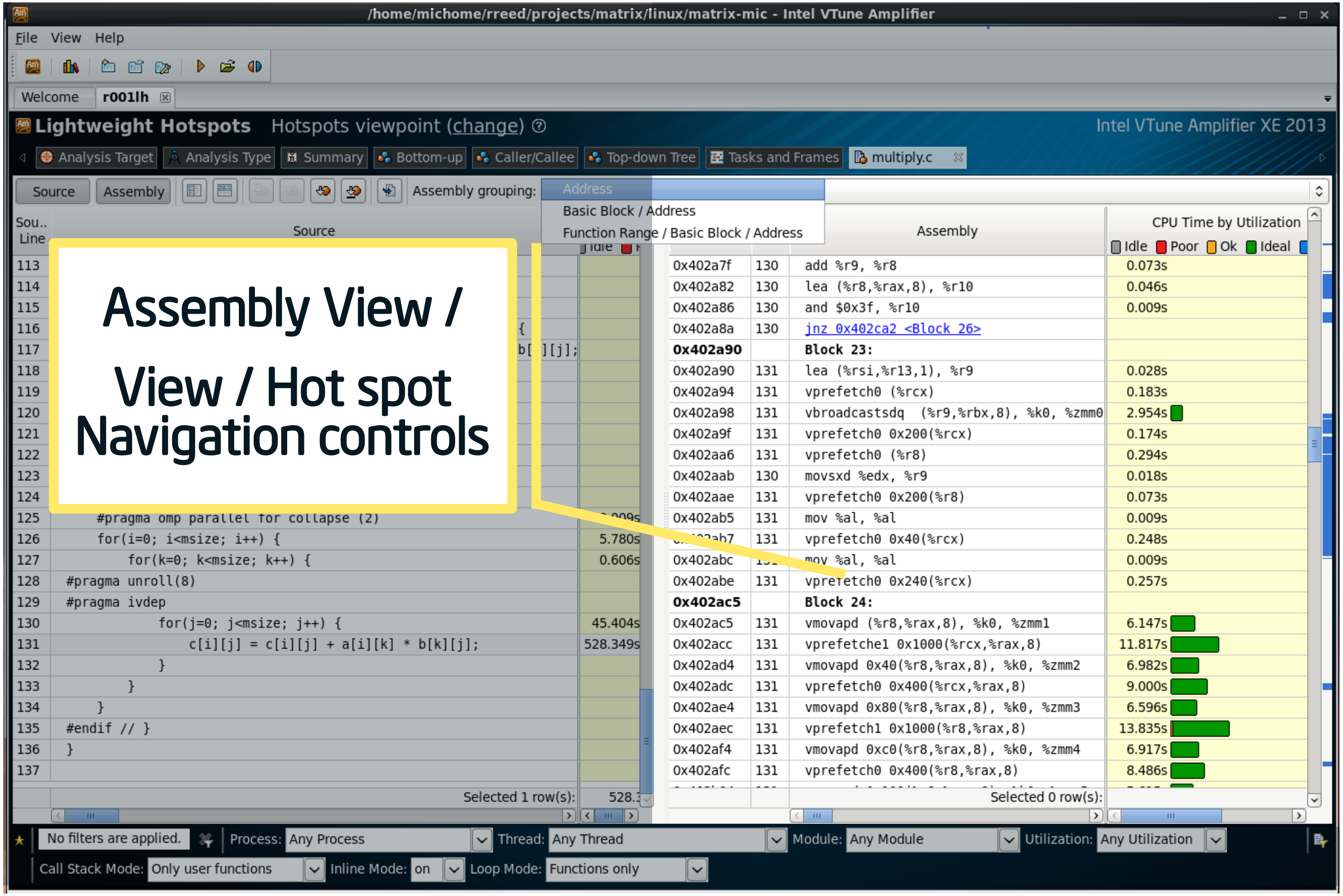
0x402a94	131	vprefetch0 (%rcx)	0.183s
0x402a98	131	vbroadcastsdq (%r9,%rbx,8), %k0, %zmm0	2.954s
0x402a9f	131	vprefetch0 0x200(%rcx)	0.174s
0x402aa6	131	vprefetch0 (%r8)	0.294s
0x402aab	130	movsxd %edx, %r9	0.018s
0x402aae	131	vprefetch0 0x200(%r8)	0.073s
0x402ab5	131	mov %al, %al	0.009s
0x402ab7	131	vprefetch0 0x40(%rcx)	0.248s
0x402abc	131	mov %al, %al	0.009s
0x402abe	131	vprefetch0 0x240(%rcx)	0.257s
0x402ac5		**Block 24:**	
0x402ac5	131	vmovapd (%r8,%rax,8), %k0, %zmm1	6.147s
0x402acc	131	vprefetche1 0x1000(%rcx,%rax,8)	11.817s
0x402ad4	131	vmovapd 0x40(%r8,%rax,8), %k0, %zmm2	6.982s
0x402adc	131	vprefetch0 0x400(%rcx,%rax,8)	9.000s
0x402ae4	131	vmovapd 0x80(%r8,%rax,8), %k0, %zmm3	6.596s
0x402aec	131	vprefetch1 0x1000(%r8,%rax,8)	13.835s
0x402af4	131	vmovapd 0xc0(%r8,%rax,8), %k0, %zmm4	6.917s
0x402afc	131	vprefetch0 0x400(%r8,%rax,8)	8.486s
		Selected 0 row(s):	

No filters are applied. Process: Any Process Thread: Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Source View / View / Hot spot Navigation controls

VTune™ Amplifier XE visualizes performance



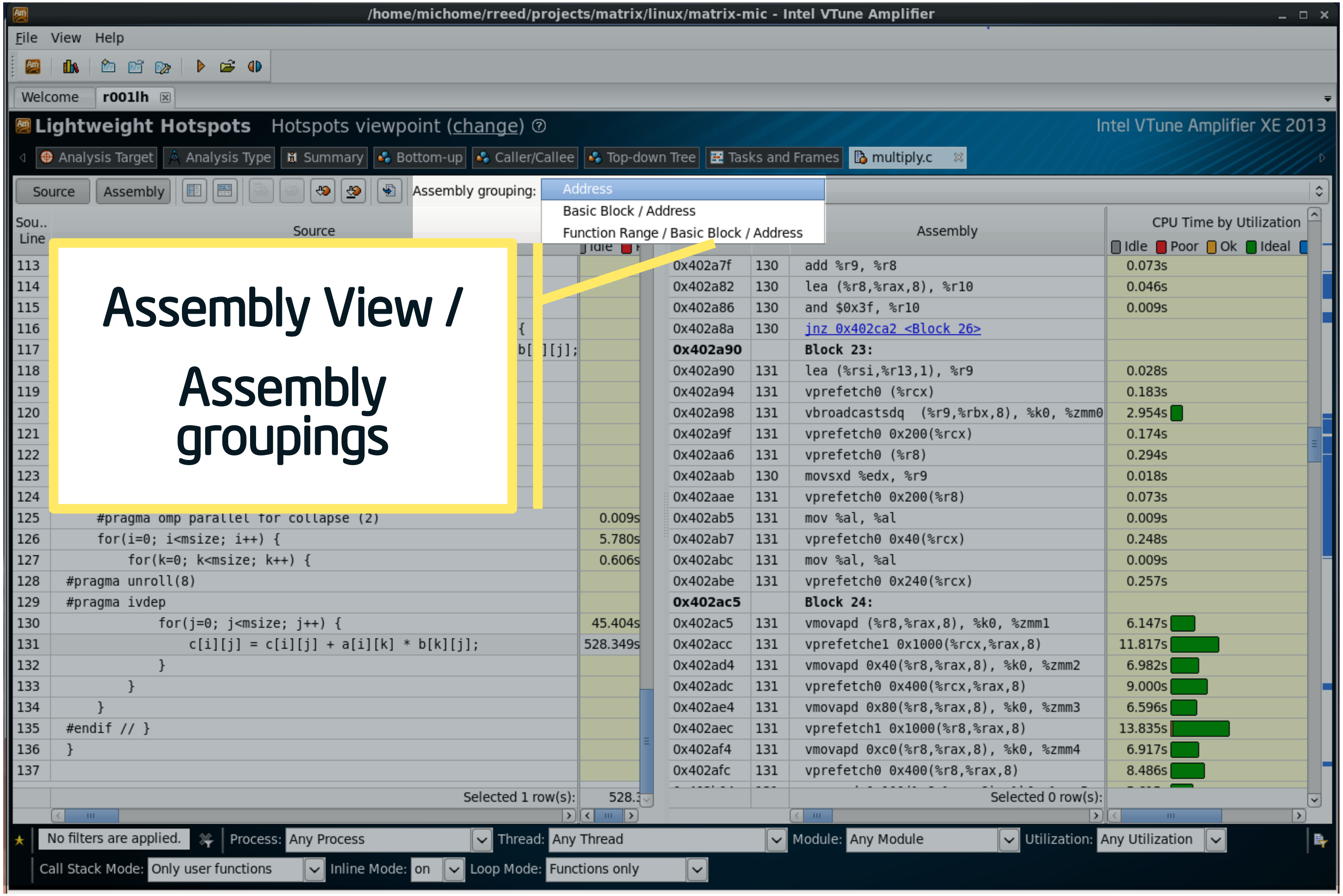
Assembly View / View / Hot spot Navigation controls

```
113
114
115
116                                                  {
117                                                  b[ ][j];
118
119
120
121
122
123
124
125       #pragma omp parallel for collapse (2)      0.009s
126       for(i=0; i<msize; i++) {                   5.780s
127           for(k=0; k<msize; k++) {               0.606s
128    #pragma unroll(8)
129    #pragma ivdep
130               for(j=0; j<msize; j++) {          45.404s
131                   c[i][j] = c[i][j] + a[i][k] * b[k][j];   528.349s
132               }
133           }
134       }
135    #endif // }
136    }
137
                                    Selected 1 row(s):   528.3
```

		Assembly	CPU Time by Utilization ▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal
0x402a7f	130	add %r9, %r8	0.073s
0x402a82	130	lea (%r8,%rax,8), %r10	0.046s
0x402a86	130	and $0x3f, %r10	0.009s
0x402a8a	130	jnz 0x402ca2 <Block 26>	
0x402a90		**Block 23:**	
0x402a90	131	lea (%rsi,%r13,1), %r9	0.028s
0x402a94	131	vprefetch0 (%rcx)	0.183s
0x402a98	131	vbroadcastsdq (%r9,%rbx,8), %k0, %zmm0	2.954s
0x402a9f	131	vprefetch0 0x200(%rcx)	0.174s
0x402aa6	131	vprefetch0 (%r8)	0.294s
0x402aab	130	movsxd %edx, %r9	0.018s
0x402aae	131	vprefetch0 0x200(%r8)	0.073s
0x402ab5	131	mov %al, %al	0.009s
0x402ab7	131	vprefetch0 0x40(%rcx)	0.248s
0x402abc	131	mov %al, %al	0.009s
0x402abe	131	vprefetch0 0x240(%rcx)	0.257s
0x402ac5		**Block 24:**	
0x402ac5	131	vmovapd (%r8,%rax,8), %k0, %zmm1	6.147s
0x402acc	131	vprefetche1 0x1000(%rcx,%rax,8)	11.817s
0x402ad4	131	vmovapd 0x40(%r8,%rax,8), %k0, %zmm2	6.982s
0x402adc	131	vprefetch0 0x400(%rcx,%rax,8)	9.000s
0x402ae4	131	vmovapd 0x80(%r8,%rax,8), %k0, %zmm3	6.596s
0x402aec	131	vprefetch1 0x1000(%r8,%rax,8)	13.835s
0x402af4	131	vmovapd 0xc0(%r8,%rax,8), %k0, %zmm4	6.917s
0x402afc	131	vprefetch0 0x400(%r8,%rax,8)	8.486s
		Selected 0 row(s):	

No filters are applied.
Process: Any Process
Thread: Any Thread
Module: Any Module
Utilization: Any Utilization
Call Stack Mode: Only user functions
Inline Mode: on
Loop Mode: Functions only

VTune™ Amplifier XE visualizes performance



Assembly grouping: Address

Sou.. Line	Source	
113		
114		
115		
116	{	
117	b[ ][j];	
118		
119		
120		
121		
122		
123		
124		
125	`#pragma omp parallel for collapse (2)`	0.009s
126	`for(i=0; i<msize; i++) {`	5.780s
127	`for(k=0; k<msize; k++) {`	0.606s
128	`#pragma unroll(8)`	
129	`#pragma ivdep`	
130	`for(j=0; j<msize; j++) {`	45.404s
131	`c[i][j] = c[i][j] + a[i][k] * b[k][j];`	528.349s
132	`}`	
133	`}`	
134	`}`	
135	`#endif // }`	
136	`}`	
137		

Selected 1 row(s): 528.3

Address	Sou.. Line	Assembly	CPU Time by Utilization
0x402a7f	130	`add %r9, %r8`	0.073s
0x402a82	130	`lea (%r8,%rax,8), %r10`	0.046s
0x402a86	130	`and $0x3f, %r10`	0.009s
0x402a8a	130	`jnz 0x402ca2 <Block 26>`	
0x402a90		**Block 23:**	
0x402a90	131	`lea (%rsi,%r13,1), %r9`	0.028s
0x402a94	131	`vprefetch0 (%rcx)`	0.183s
0x402a98	131	`vbroadcastsdq (%r9,%rbx,8), %k0, %zmm0`	2.954s
0x402a9f	131	`vprefetch0 0x200(%rcx)`	0.174s
0x402aa6	131	`vprefetch0 (%r8)`	0.294s
0x402aab	130	`movsxd %edx, %r9`	0.018s
0x402aae	131	`vprefetch0 0x200(%r8)`	0.073s
0x402ab5	131	`mov %al, %al`	0.009s
0x402ab7	131	`vprefetch0 0x40(%rcx)`	0.248s
0x402abc	131	`mov %al, %al`	0.009s
0x402abe	131	`vprefetch0 0x240(%rcx)`	0.257s
0x402ac5		**Block 24:**	
0x402ac5	131	`vmovapd (%r8,%rax,8), %k0, %zmm1`	6.147s
0x402acc	131	`vprefetche1 0x1000(%rcx,%rax,8)`	11.817s
0x402ad4	131	`vmovapd 0x40(%r8,%rax,8), %k0, %zmm2`	6.982s
0x402adc	131	`vprefetch0 0x400(%rcx,%rax,8)`	9.000s
0x402ae4	131	`vmovapd 0x80(%r8,%rax,8), %k0, %zmm3`	6.596s
0x402aec	131	`vprefetch1 0x1000(%r8,%rax,8)`	13.835s
0x402af4	131	`vmovapd 0xc0(%r8,%rax,8), %k0, %zmm4`	6.917s
0x402afc	131	`vprefetch0 0x400(%r8,%rax,8)`	8.486s

Selected 0 row(s):

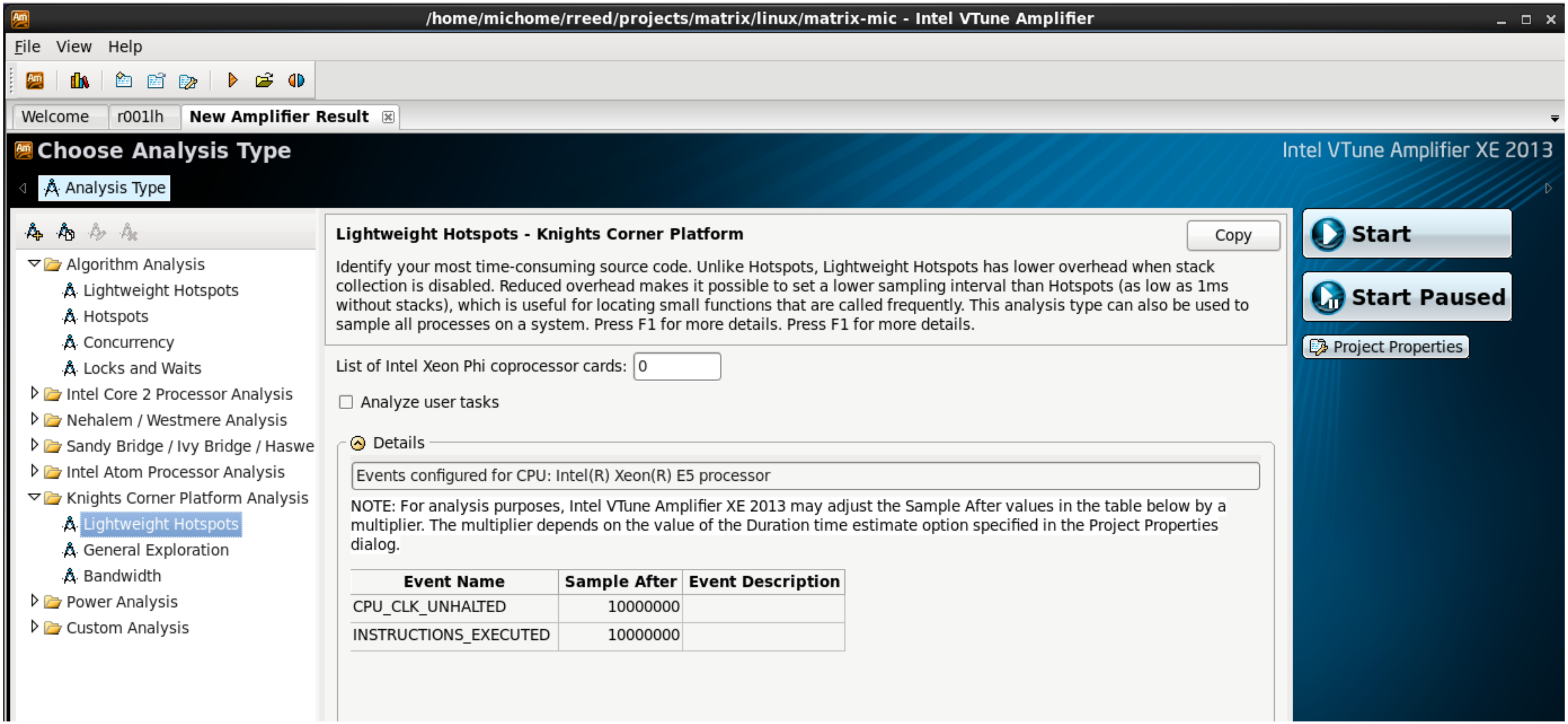
No filters are applied. Process: Any Process Thread: Any Thread Module: Any Module Utilization: Any Utilization

Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Assembly View /

Assembly groupings

For event collection the coprocessor is treated as a special HW architecture



Lightweight Hotspots - Knights Corner Platform

Identify your most time-consuming source code. Unlike Hotspots, Lightweight Hotspots has lower overhead when stack collection is disabled. Reduced overhead makes it possible to set a lower sampling interval than Hotspots (as low as 1ms without stacks), which is useful for locating small functions that are called frequently. This analysis type can also be used to sample all processes on a system. Press F1 for more details. Press F1 for more details.

List of Intel Xeon Phi coprocessor cards: 0

□ Analyze user tasks

Details

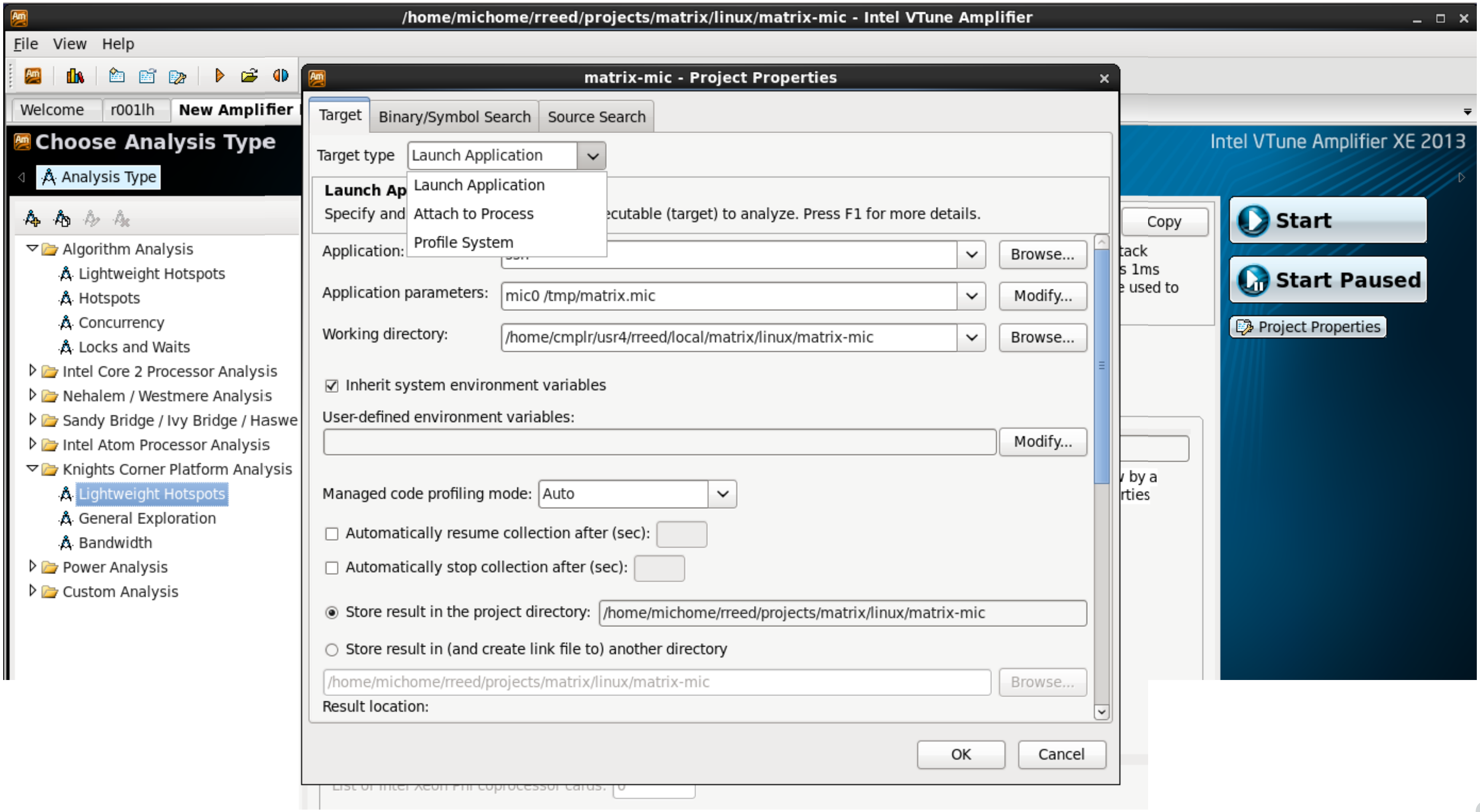
Events configured for CPU: Intel(R) Xeon(R) E5 processor

NOTE: For analysis purposes, Intel VTune Amplifier XE 2013 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the Project Properties dialog.

Event Name	Sample After	Event Description
CPU_CLK_UNHALTED	10000000	
INSTRUCTIONS_EXECUTED	10000000	

Optimization Notice

Project properties provides the means to invoke data collection by target type



Target type: Launch Application

Launch Ap

Specify and ... ecutable (target) to analyze. Press F1 for more details.

Application: ssh

Application parameters: mic0 /tmp/matrix.mic

Working directory: /home/cmplr/usr4/rreed/local/matrix/linux/matrix-mic

☑ Inherit system environment variables

User-defined environment variables:

Managed code profiling mode: Auto

☐ Automatically resume collection after (sec):

☐ Automatically stop collection after (sec):

● Store result in the project directory: /home/michome/rreed/projects/matrix/linux/matrix-mic

○ Store result in (and create link file to) another directory

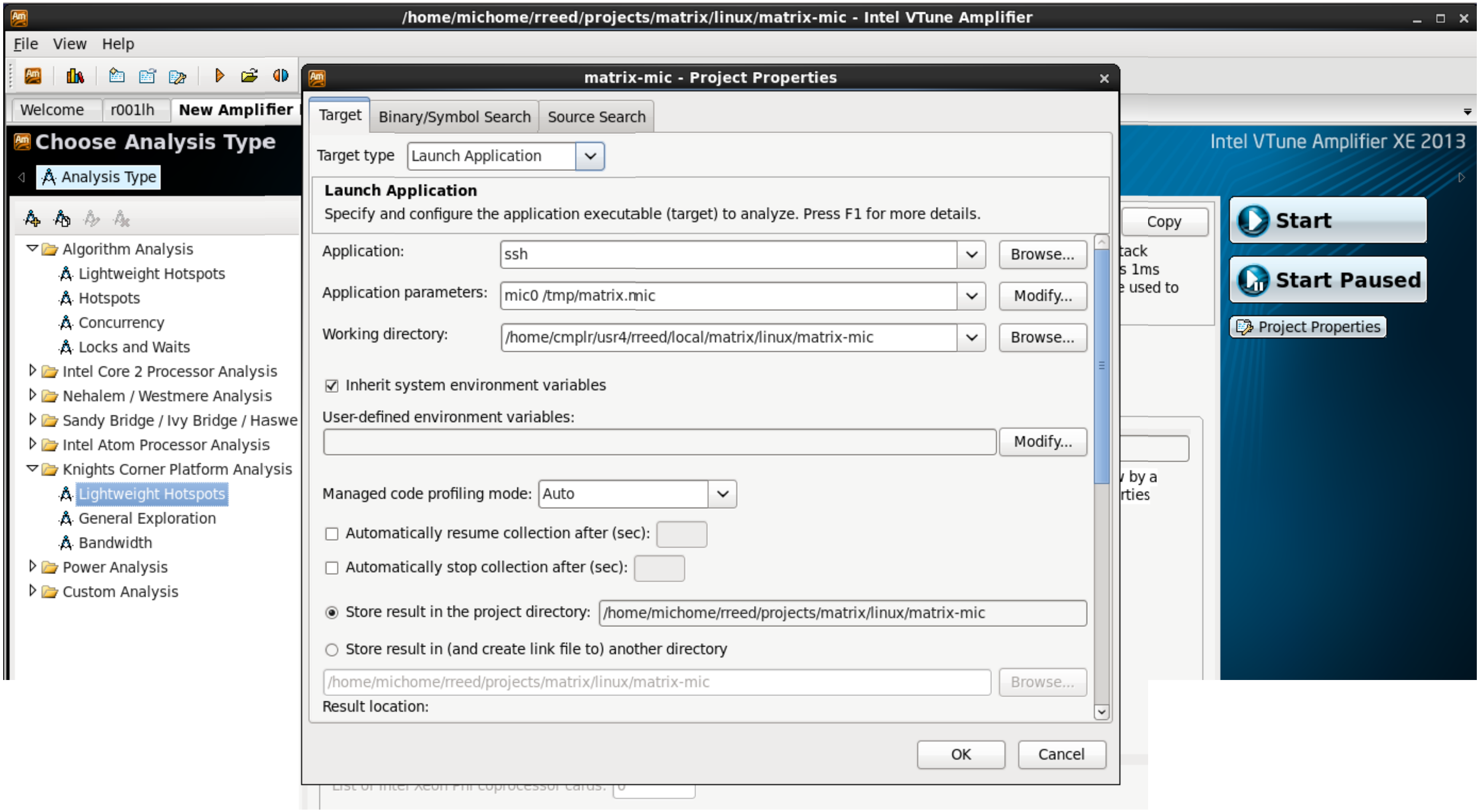
/home/michome/rreed/projects/matrix/linux/matrix-mic

Result location:

List of Intel Xeon Phi coprocessor cards: 0

Optimization Notice

Launch Application serves many uses, from host/offload to native execution



Target type: Launch Application

Launch Application

Specify and configure the application executable (target) to analyze. Press F1 for more details.

Application: ssh

Application parameters: mic0 /tmp/matrix.mic

Working directory: /home/cmplr/usr4/rreed/local/matrix/linux/matrix-mic

☑ Inherit system environment variables

User-defined environment variables:

Managed code profiling mode: Auto

□ Automatically resume collection after (sec):

□ Automatically stop collection after (sec):

● Store result in the project directory: /home/michome/rreed/projects/matrix/linux/matrix-mic

○ Store result in (and create link file to) another directory

/home/michome/rreed/projects/matrix/linux/matrix-mic

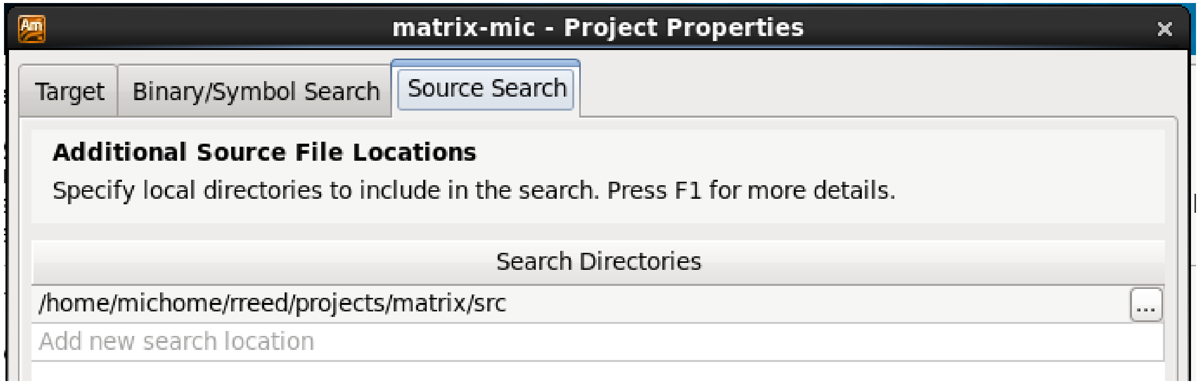
Result location:

List of Intel Xeon Phi coprocessor cards: 0

Optimization Notice

Search directories have been reorganized to speed symbol resolution during finalization

- Enumerate source directories under this tab



Additional Source File Locations

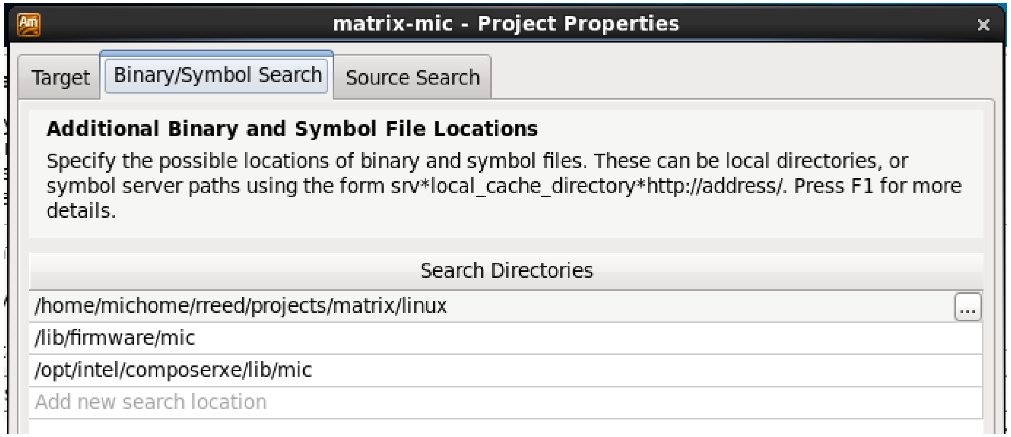
Specify local directories to include in the search. Press F1 for more details.

Search Directories

/home/michome/rreed/projects/matrix/src

Add new search location

- Put library paths here



Additional Binary and Symbol File Locations

Specify the possible locations of binary and symbol files. These can be local directories, or symbol server paths using the form srv*local_cache_directory*http://address/. Press F1 for more details.

Search Directories

/home/michome/rreed/projects/matrix/linux

/lib/firmware/mic

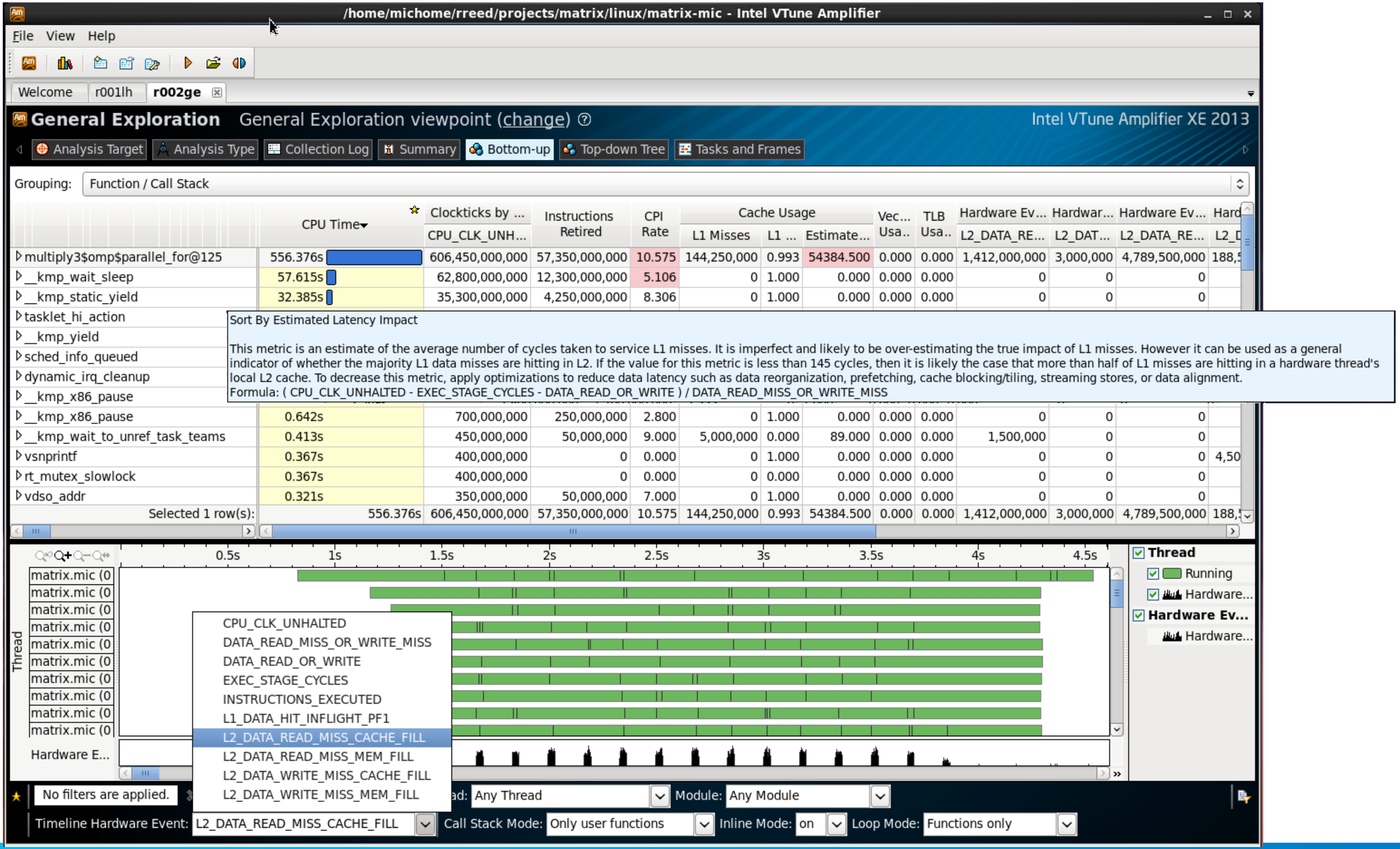
/opt/intel/composerxe/lib/mic

Add new search location

Notable coprocessor library paths:

/lib/firmware/mic
/usr/linux-k1om-4.7/linux-k1om/lib64
/opt/intel/composerxe/lib/mic
/opt/intel/composerxe/tbb/lib/mic
/opt/intel/composerxe/mkl/lib/mic
/opt/intel/mpi-rt/4.1.0/mic

General Exploration runs a set of events to drive top-down analysis



Grouping: Function / Call Stack

	CPU Time	Clockticks by ...	Instructions Retired	CPI Rate	Cache Usage			Vec... Usa..	TLB Usa..	Hardware Ev...	Hardwar...	Hardware Ev...	Hard
		CPU_CLK_UNH...			L1 Misses	L1 ...	Estimate...			L2_DATA_RE...	L2_DAT...	L2_DATA_RE...	L2_D
multiply3$omp$parallel_for@125	556.376s	606,450,000,000	57,350,000,000	10.575	144,250,000	0.993	54384.500	0.000	0.000	1,412,000,000	3,000,000	4,789,500,000	188,5
__kmp_wait_sleep	57.615s	62,800,000,000	12,300,000,000	5.106	0	1.000	0.000	0.000	0.000	0	0	0	
__kmp_static_yield	32.385s	35,300,000,000	4,250,000,000	8.306	0	1.000	0.000	0.000	0.000	0	0	0	
tasklet_hi_action													
__kmp_yield													
sched_info_queued													
dynamic_irq_cleanup													
__kmp_x86_pause													
__kmp_x86_pause	0.642s	700,000,000	250,000,000	2.800	0	1.000	0.000	0.000	0.000	0	0	0	
__kmp_wait_to_unref_task_teams	0.413s	450,000,000	50,000,000	9.000	5,000,000	0.000	89.000	0.000	0.000	1,500,000	0	0	
vsnprintf	0.367s	400,000,000	0	0.000	0	1.000	0.000	0.000	0.000	0	0	0	4,50
rt_mutex_slowlock	0.367s	400,000,000	0	0.000	0	0.000	0.000	0.000	0.000	0	0	0	
vdso_addr	0.321s	350,000,000	50,000,000	7.000	0	1.000	0.000	0.000	0.000	0	0	0	
Selected 1 row(s):	556.376s	606,450,000,000	57,350,000,000	10.575	144,250,000	0.993	54384.500	0.000	0.000	1,412,000,000	3,000,000	4,789,500,000	188,5

Sort By Estimated Latency Impact

This metric is an estimate of the average number of cycles taken to service L1 misses. It is imperfect and likely to be over-estimating the true impact of L1 misses. However it can be used as a general indicator of whether the majority L1 data misses are hitting in L2. If the value for this metric is less than 145 cycles, then it is likely the case that more than half of L1 misses are hitting in a hardware thread's local L2 cache. To decrease this metric, apply optimizations to reduce data latency such as data reorganization, prefetching, cache blocking/tiling, streaming stores, or data alignment.
Formula: (CPU_CLK_UNHALTED - EXEC_STAGE_CYCLES - DATA_READ_OR_WRITE) / DATA_READ_MISS_OR_WRITE_MISS

- CPU_CLK_UNHALTED
- DATA_READ_MISS_OR_WRITE_MISS
- DATA_READ_OR_WRITE
- EXEC_STAGE_CYCLES
- INSTRUCTIONS_EXECUTED
- L1_DATA_HIT_INFLIGHT_PF1
- L2_DATA_READ_MISS_CACHE_FILL
- L2_DATA_READ_MISS_MEM_FILL
- L2_DATA_WRITE_MISS_CACHE_FILL
- L2_DATA_WRITE_MISS_MEM_FILL

No filters are applied. ad: Any Thread Module: Any Module

Timeline Hardware Event: L2_DATA_READ_MISS_CACHE_FILL Call Stack Mode: Only user functions Inline Mode: on Loop Mode: Functions only

Agenda

Start tuning on host

Overview of Intel® VTune™ Amplifier XE

Efficiency metrics

Problem areas

Cycles Per Instruction (CPI), a standard measure, has some special kinks

- Threads on each Intel® Xeon™ Phi core share a clock
  - If all 4 HW threads are active, each gets ¼ total cycles
- Multi-stage instruction decode requires two threads to utilize the whole core – one thread only gets half
- With two ops/per cycle (U-V-pipe dual issue):

Threads per Core	Best CPI per Core	Best CPI per Thread
1 x	1.0	= 1.0
2 x	0.5	= 1.0
3 x	0.5	= 1.5
4 x	0.5	= 2.0

- To get thread CPI, multiply by the active threads

Optimization Notice

As an efficiency metric, CPI must be considered carefully: it IS a ratio

- Changes in CPI absent major code changes can indicate general latency gains/losses

Metric	Formula	Investigate if
CPI per Thread	CPU_CLK_UNHALTED/ INSTRUCTIONS_EXECUTED	> 4.0, or increasing
CPI per Core	(CPI per Thread) / Number of hardware threads used	> 1.0, or increasing

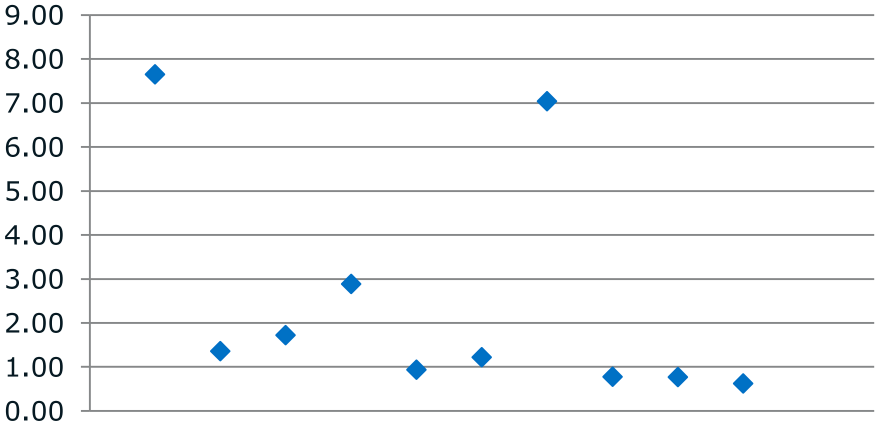
- Note the effect on CPI from applied optimizations
- Reduce high CPI through optimizations that target latency
  - Better prefetch
  - Increase data reuse through better blocking

Two more examples why absolute CPI value is less important than changes

- Scaling data from a typical lab workload:

Metric	1 hardware thread / core	2 hardware threads / core	3 hardware threads / core	4 hardware threads / core
CPI per Thread	5.24	8.80	11.18	13.74
CPI per Core	5.24	4.40	3.73	3.43

- Observed CPIs from several tuned workloads:



Efficiency Metric: Compute to Data Access Ratio

- Measures an application’s computational density, and suitability for Intel® Xeon Phi™ coprocessors

Metric	Formula	Investigate if
Vectorization Intensity	VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED	
L1 Compute to Data Access Ratio	VPU_ELEMENTS_ACTIVE / DATA_READ_OR_WRITE	< Vectorization Intensity
L2 Compute to Data Access Ratio	VPU_ELEMENTS_ACTIVE / DATA_READ_MISS_OR_WRITE_MISS	< 100x L1 Compute to Data Access Ratio

- Increase computational density through vectorization and reducing data access (see cache issues, also, DATA ALIGNMENT!)

Agenda

Start tuning on host

Overview of Intel® VTune™ Amplifier XE

Efficiency metrics

Problem areas*

*tuning suggestions requiring deeper understanding of architectural tradeoffs and application data handling details are highlighted with this “ninja” notation

Problem Area: L1 Cache Usage

- Significantly affects data access latency and therefore application performance

Metric	Formula	Investigate if
L1 Misses	DATA_READ_MISS_OR_WRITE_MISS + L1_DATA_HIT_INFLIGHT_PF1	
L1 Hit Rate	(DATA_READ_OR_WRITE – L1 Misses) / DATA_READ_OR_WRITE	< 95%

- Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores
  - If using 4K access stride, may be experiencing conflict misses
  - Examine Compiler prefetching (Compiler-generated L1 prefetches should not miss)

Problem Area: Data Access Latency

- Significantly affects application performance

Metric	Formula	Investigate if
Estimated Latency Impact	(CPU_CLK_UNHALTED – EXEC_STAGE_CYCLES – DATA_READ_OR_WRITE) / DATA_READ_OR_WRITE_MISS	>145

- Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores
  - Check cache locality – turn off prefetching and use CACHE_FILL events - reduce sharing if needed/possible
  - If using 64K access stride, may be experiencing conflict misses

Problem Area: TLB Usage

- Also affects data access latency and therefore application performance

Metric	Formula	Investi-gate if:
L1 TLB miss ratio	DATA_PAGE_WALK/DATA_READ_OR_WRITE	> 1%
L2 TLB miss ratio	LONG_DATA_PAGE_WALK / DATA_READ_OR_WRITE	> .1%
L1 TLB misses per L2 TLB miss	DATA_PAGE_WALK / LONG_DATA_PAGE_WALK	> 100x

- Tuning Suggestions:
  - Improve cache usage & data access latency
  - If L1 TLB miss/L2 TLB miss is high, try using large pages
  - For loops with multiple streams, try splitting into multiple loops
  - If data access stride is a large power of 2, consider padding between arrays by one 4 KB page

Optimization Notice

Problem Area: VPU Usage

- Indicates whether an application is vectorized successfully and efficiently

Metric	Formula	Investigate if
Vectorization Intensity	VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED	<8 (DP), <16(SP)

- Tuning Suggestions:
  - Use the Compiler vectorization report!
  - For data dependencies preventing vectorization, try using Intel® Cilk™ Plus #pragma SIMD (if safe!)
  - Align data and tell the Compiler!
  - Restructure code if possible: Array notations, AOS->SOA

Problem Area: Memory Bandwidth

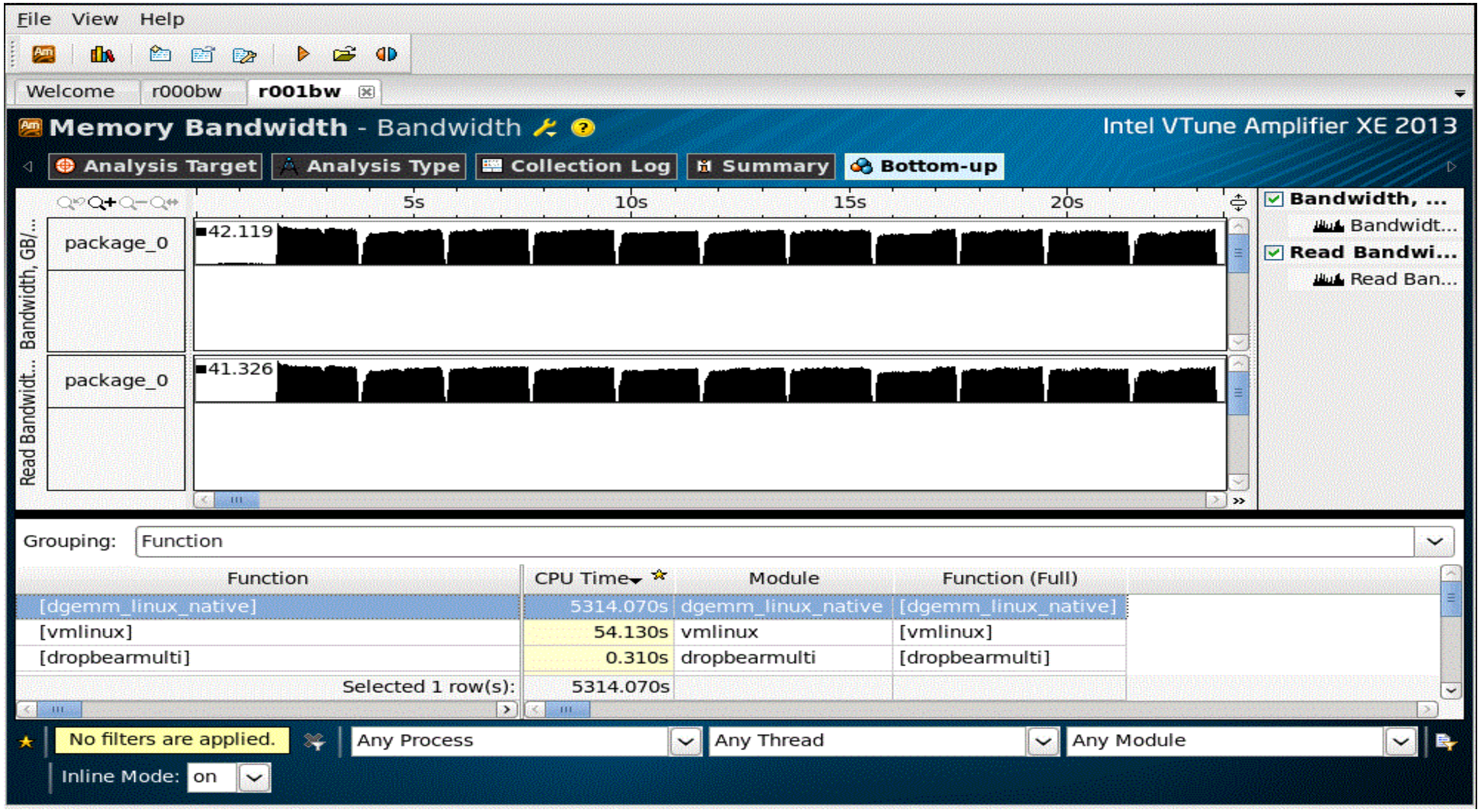
- Can increase data latency in the system or become a performance bottleneck

Metric	Formula	Investigate if
Memory Bandwidth	(UNC_F_CH0_NORMAL_READ + UNC_F_CH0_NORMAL_WRITE+ UNC_F_CH1_NORMAL_READ + UNC_F_CH1_NORMAL_WRITE) X 64/time	< 80GB/sec (practical peak 140GB/sec) (with 8 memory controllers)

- Tuning Suggestions:
  - Improve locality in caches
  - Use streaming stores
  - Improve software prefetching

Final caution: coprocessor collections can generate dense volumes of data

Example: DGEMM on 60+ cores



Grouping: Function

Function	CPU Time	Module	Function (Full)
[dgemm_linux_native]	5314.070s	dgemm_linux_native	[dgemm_linux_native]
[vmlinux]	54.130s	vmlinux	[vmlinux]
[dropbearmulti]	0.310s	dropbearmulti	[dropbearmulti]
Selected 1 row(s):	5314.070s		

No filters are applied. Any Process Any Thread Any Module

Inline Mode: on

Tip: Use a CPU Mask to reduce data volume while maintaining equivalent accuracy.

Summary

- Vectorization, Parallelism, and Data locality are critical to good performance for the Intel® Xeon Phi™ Coprocessor
- Event names can be misleading – we recommend using the metrics given in this presentation or our tuning guide at http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding
- Intel® VTune™ Amplifier XE supports collecting all of the above metrics, as well as providing special analysis types like General Exploration and Memory Bandwidth

Please return your evaluation forms!

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.

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®