MPI Programming on the Intel® Xeon Phi™ Coprocessor

Dr.-Ing. Michael Klemm
Software and Services Group
Intel Corporation
(michael.klemm@intel.com)

Legal Disclaimer & Optimization Notice

Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing

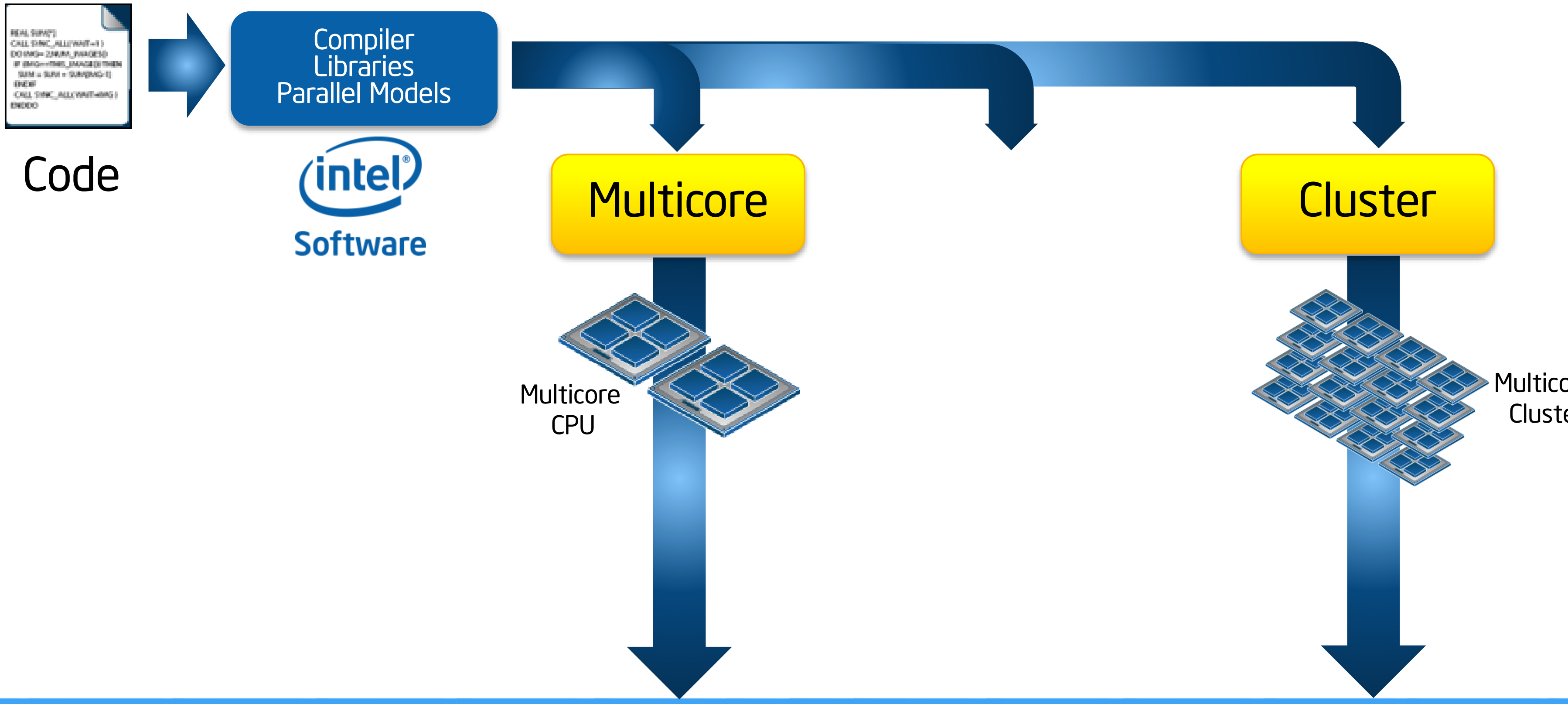
Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing

Enabling & Advancing Parallelism

High Performance Parallel Programming

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

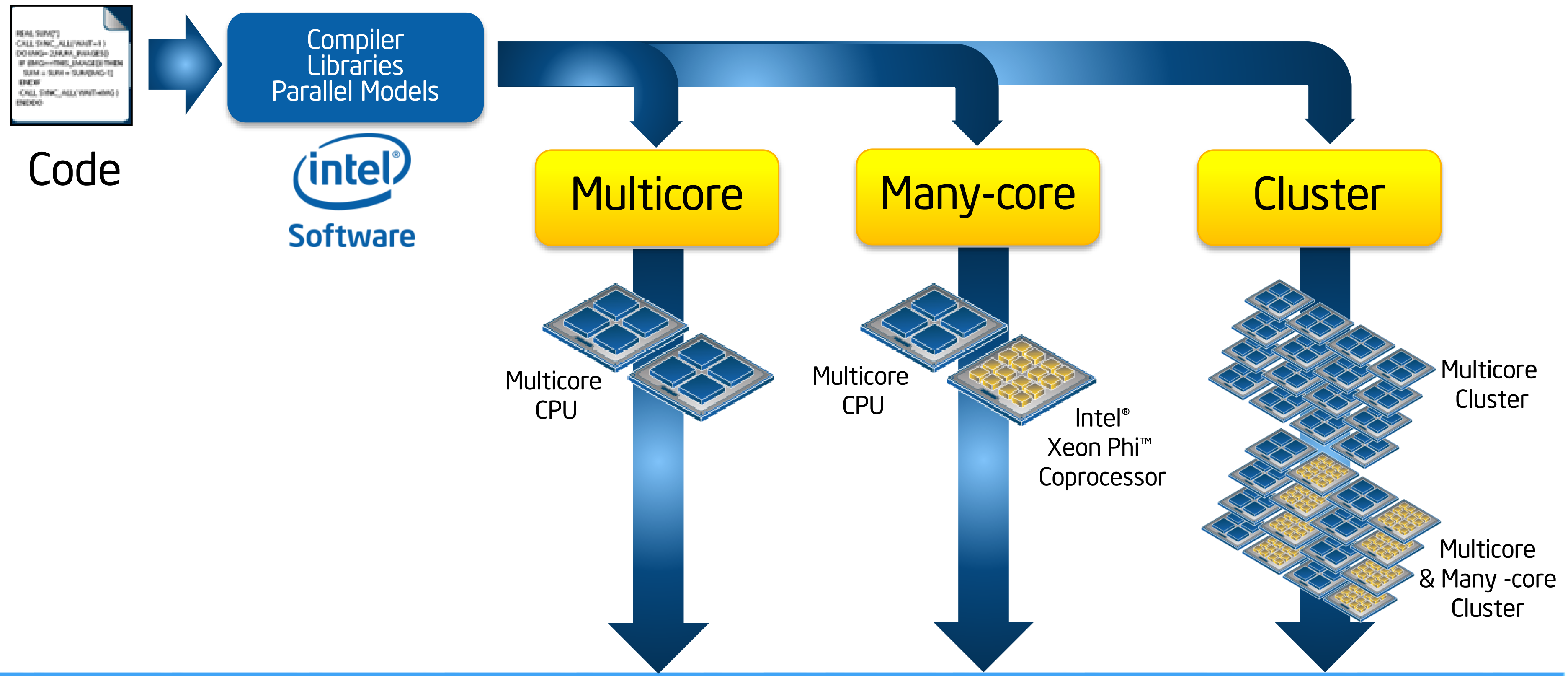


Use One Software Architecture Today. Scale Forward Tomorrow.

Enabling & Advancing Parallelism

High Performance Parallel Programming

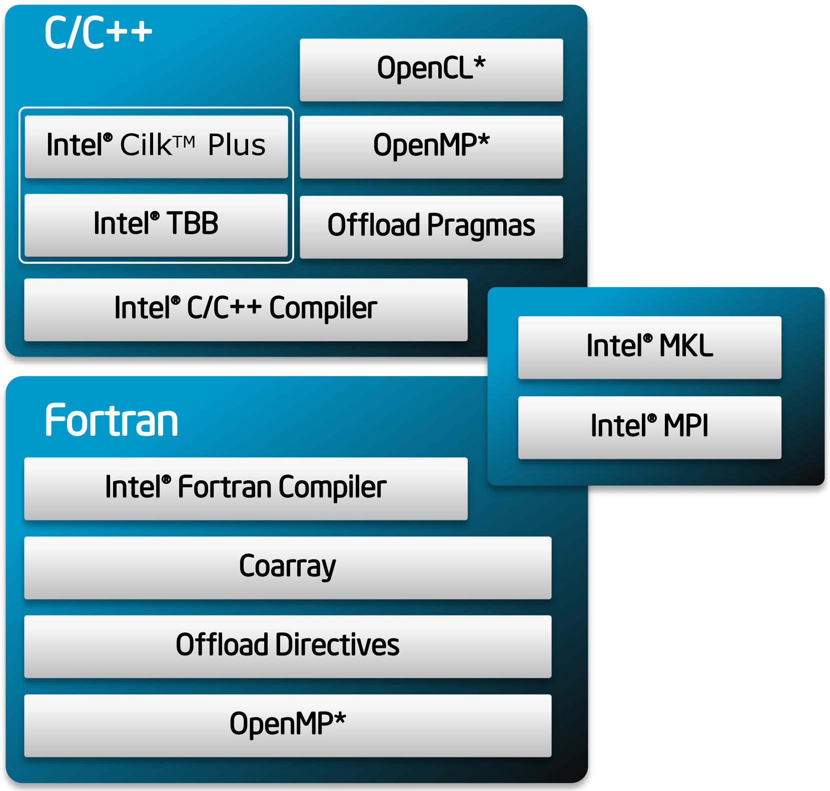
Intel tools, libraries and parallel models extend to multicore, many-core and heterogeneous computing

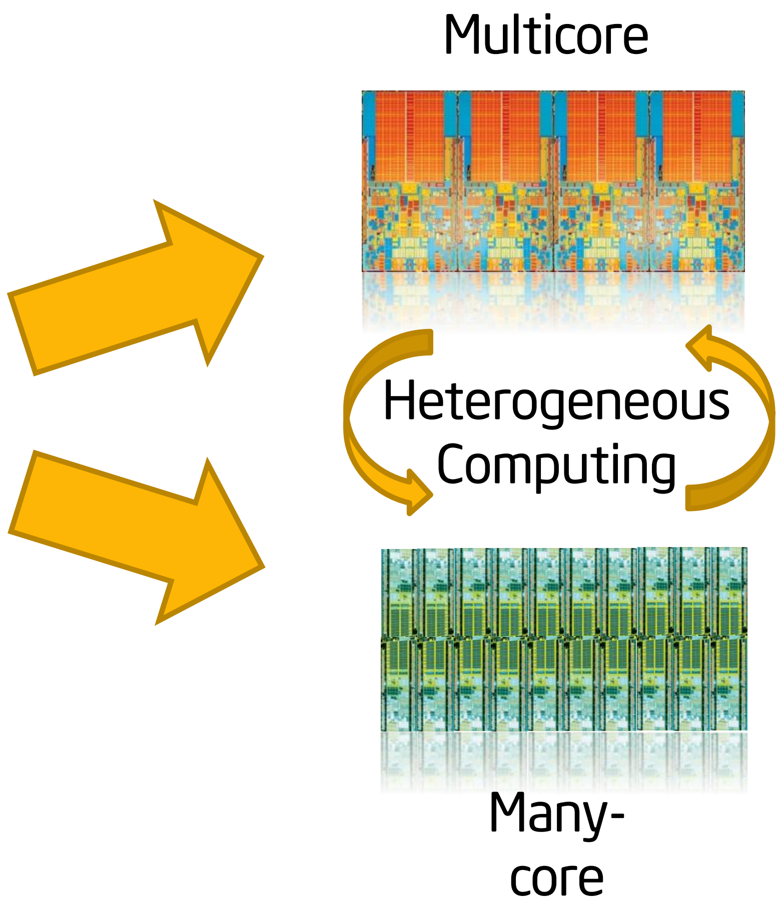


Use One Software Architecture Today. Scale Forward Tomorrow.

Preserve Your Development Investment

Common Tools and Programming Models for Parallelism

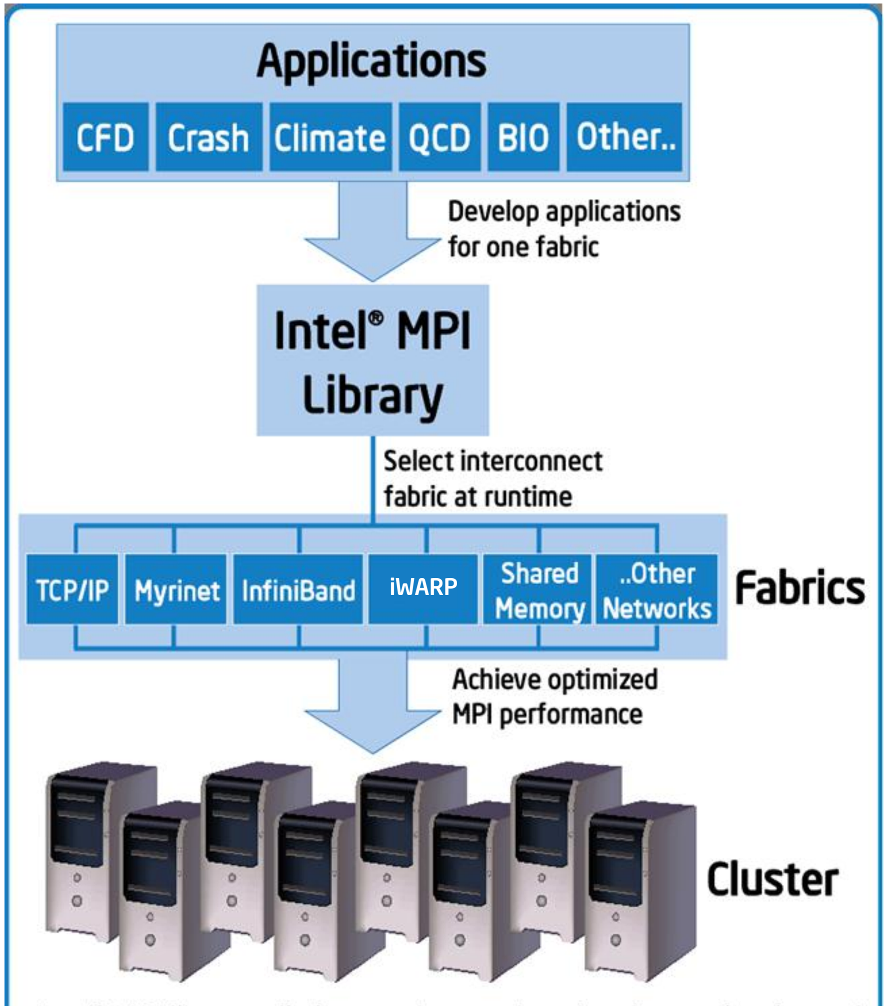




Develop Using Parallel Models that Support Heterogeneous Computing

Intel® MPI Library Overview

- Intel is a leading vendor of MPI implementations and tools
- Optimized MPI application performance
  - Application-specific tuning
  - Automatic tuning
- Lower 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

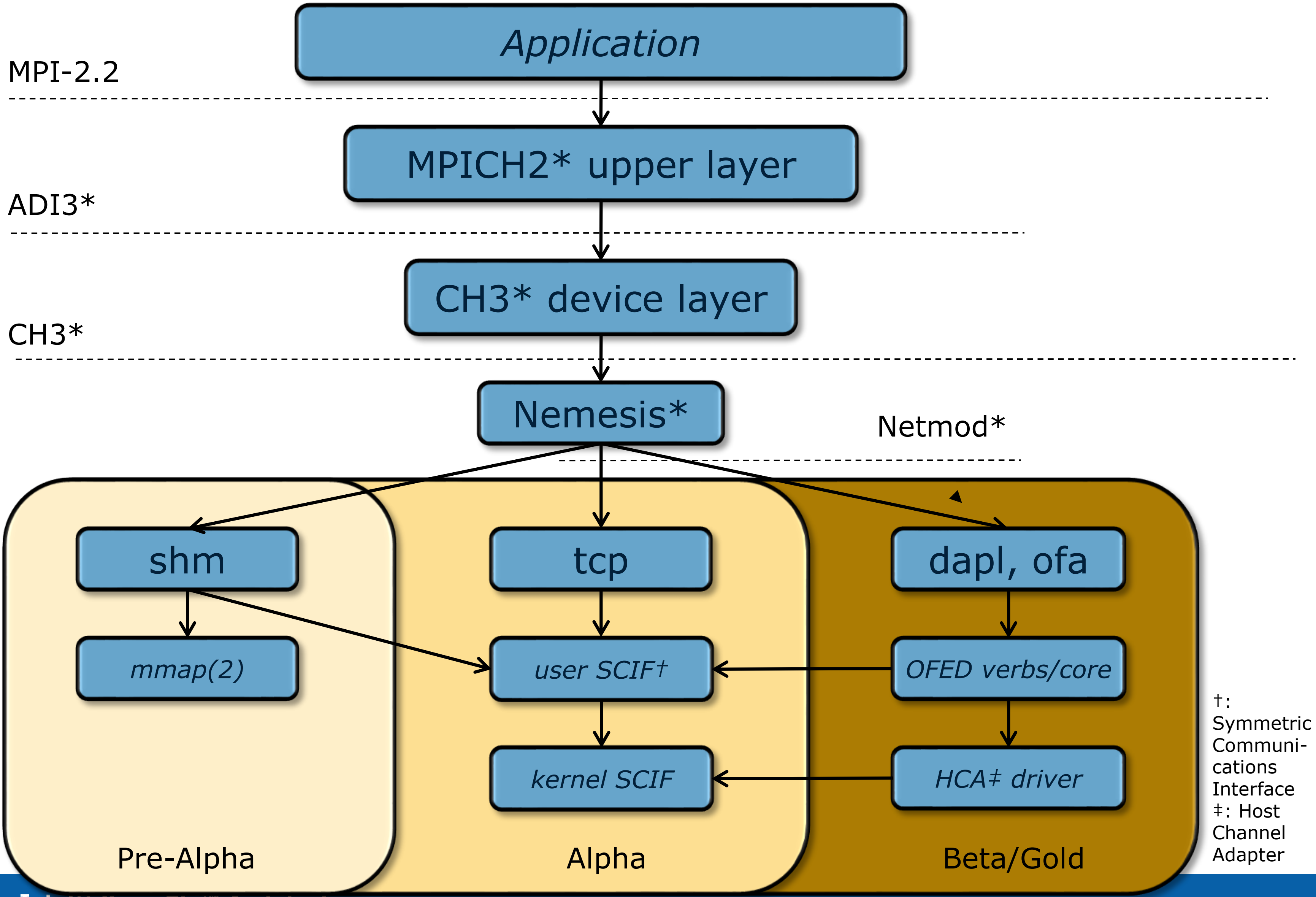


Intel® MPI library eliminates the need to develop, maintain, and test applications running on multiple fabrics.

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

Intel® MPI Library Architecture & Staging



Selecting network fabrics

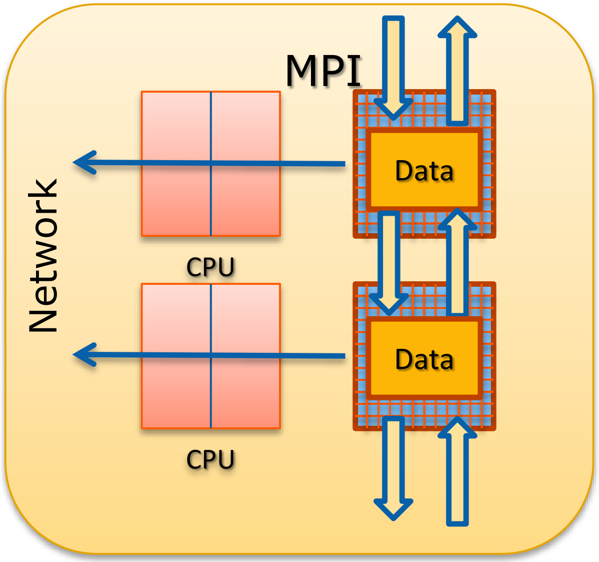
- Intel® MPI selects automatically the best available network fabric it can find.
  - Use I_MPI_FABRICS to select a different communication device explicitly
- The best fabric is usually based on Infiniband (dapl, ofa) for inter node communication and shared memory for intra node
- Available for Intel(R) Xeon Phi(TM):
  - shm, tcp, ofa, dapl
  - Availability checked in the order shm:dapl, shm:ofa, shm:tcp (intra:inter)
- Set I_MPI_SSHM_SCIF=1 to enable shm fabric between host and Intel® Xeon Phi™

Agenda

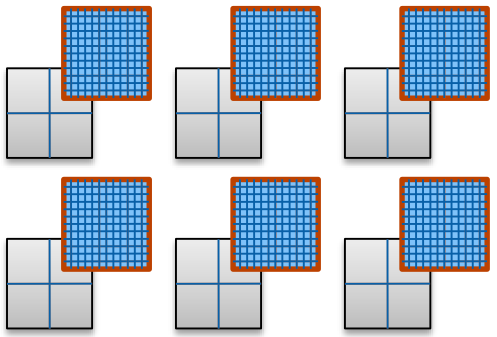
- Overview
- Programming Models
- Hybrid Computing
- Load Balancing

Coprocessor only Programming Model

- MPI ranks on Intel® Xeon Phi™ (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 Intel® compiler.

Upload the binary to the Intel® Xeon Phi™.

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

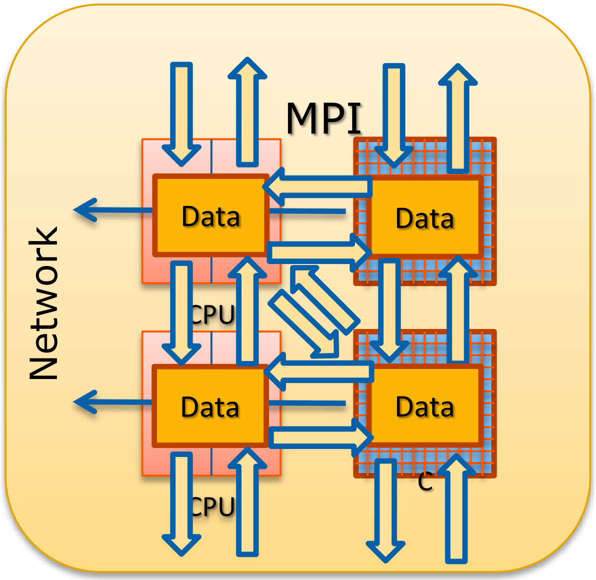
Coprocessor only Programming Model

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

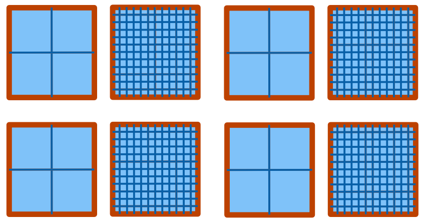
- Build the application for the Intel® Xeon Phi™ Architecture
```
# mpiicc -mmic -o test_hello.MIC test.c
```
- Upload the coprocessor executable
```
# sudo micput 172.31.1.1 ./test_hello.MIC
  /tmp/test_hello.MIC
```
  - Remark: If NFS available no explicit uploads required (just copies)!
- Launch the application on the coprocessor from host
```
# mpiexec -n 2 -wdir /tmp -host 172.31.1.1
  /tmp/test_hello.MIC
```
- Alternatively: login to the coprocessor and execute the already uploaded mpiexec.hydra there!

Symmetric Programming Model

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



Heterogeneous network of homogeneous CPUs



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

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

Run instances of the MPI application on different mixed nodes.

Symmetric Programming 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
```

- Upload the Intel® Xeon Phi™ executable

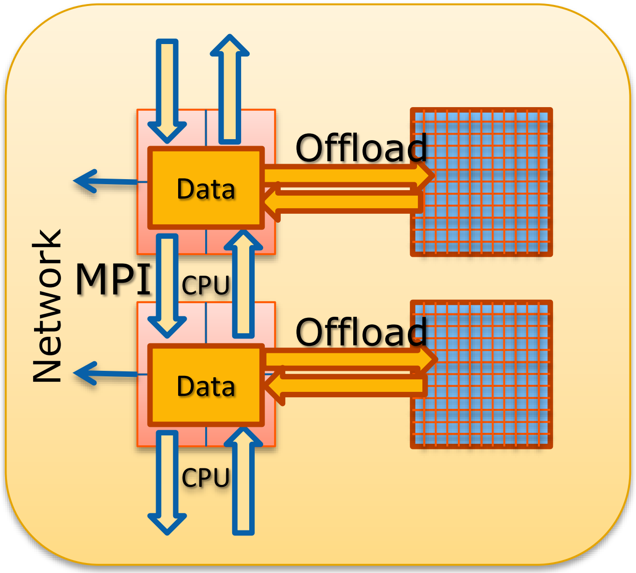
```
# sudo micput 172.31.1.1 ./test_hello.MIC
  /tmp/test_hello.MIC
```

- Launch the application on the host and the coprocessor from the host

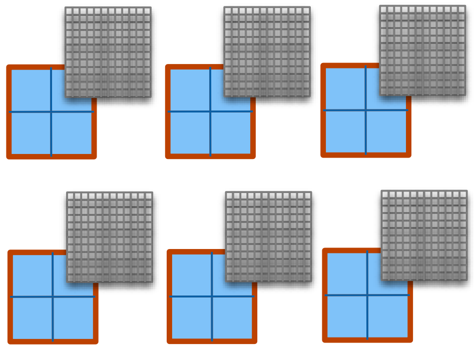
```
# mpiexec -n 2 -host <hostname> ./test_hello : -wdir
/tmp -n 2 -host 172.31.1.1 /tmp/test_hello.MIC
```

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™



Homogenous network of heterogeneous nodes



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

Run instances of the MPI application on the host, offloading code onto 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™ Architecture 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
  - Previous compilers needed `–offload-build` flag
- Execute on host(s) as usual

```
# mpiexec –n 2 ./test
```

- MPI processes will offload code for acceleration

Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing

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 (cont'd)

- 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 (including coprocessors)

Options for Thread Parallelism

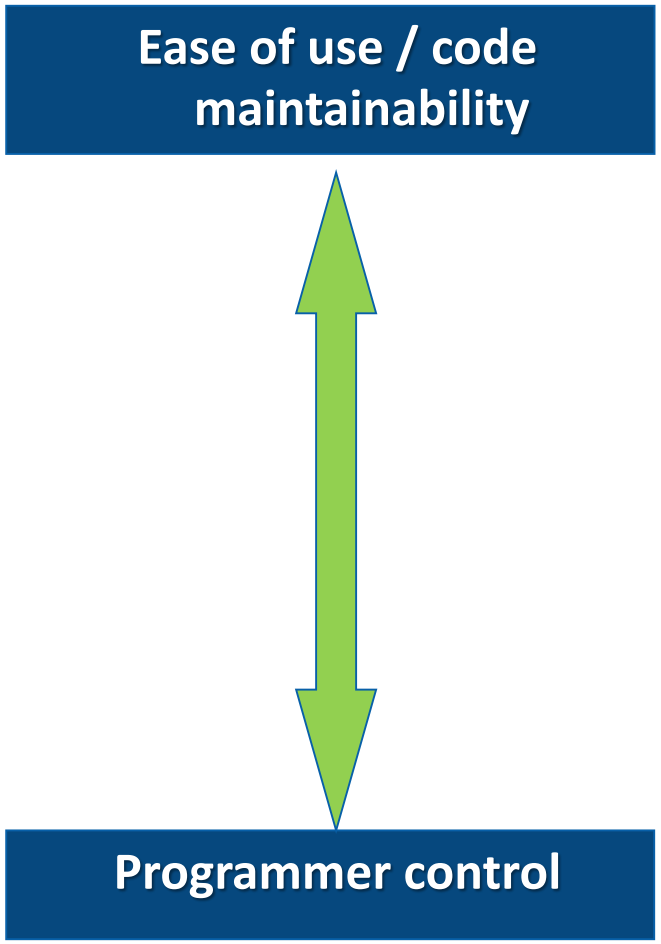
Intel® Math Kernel Library

OpenMP*

Intel® Threading Building Blocks

Intel® Cilk™ Plus

Pthreads* and other threading libraries



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

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 (takes precedence):

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

    <size> =

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

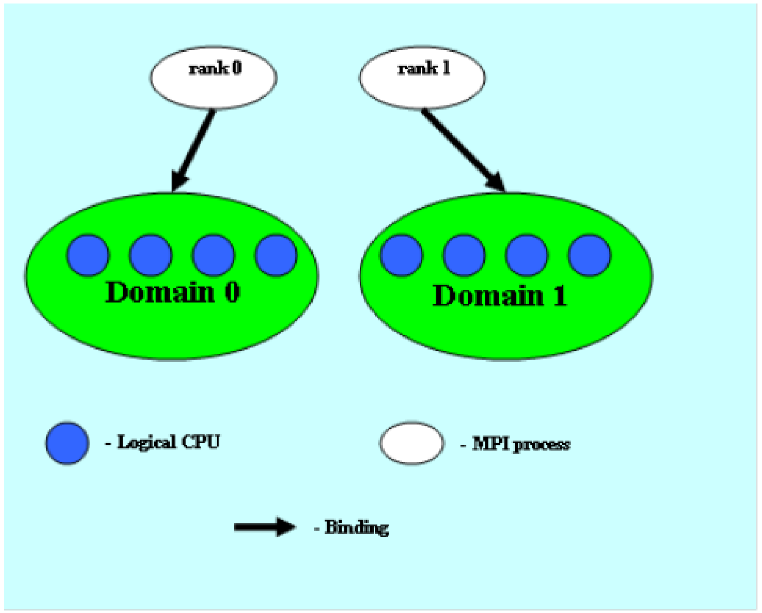
    <layout> =

`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 mpiexec of Intel® MPI reads argument sets from the command line:
  - Sections between "`:`" define an argument set (also lines in a configfile, but not yet available in Beta)
  - Host, number of nodes, but also environment can be set independently in each argument set

```
# mpiexec -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™ Plus

Co-Processor only and Symmetric Support

- Full hybrid support on Intel® Xeon from Intel ® MPI extends to Intel® Xeon Phi(TM)
- **`KMP_AFFINITY=balanced`** (only on coprocessor) in addition to **`scatter`** and **`compact`**
- Recommendations:
  - Explicitly control where MPI processes and threads run in a hybrid application
    (according to threading model and application)
  - Avoid splitting cores among MPI processes, i.e. **`I_MPI_PIN_DOMAIN`** should be a multiple of 4
  - Try different **`KMP_AFFINITY`** settings for your application

MPI+Offload Support

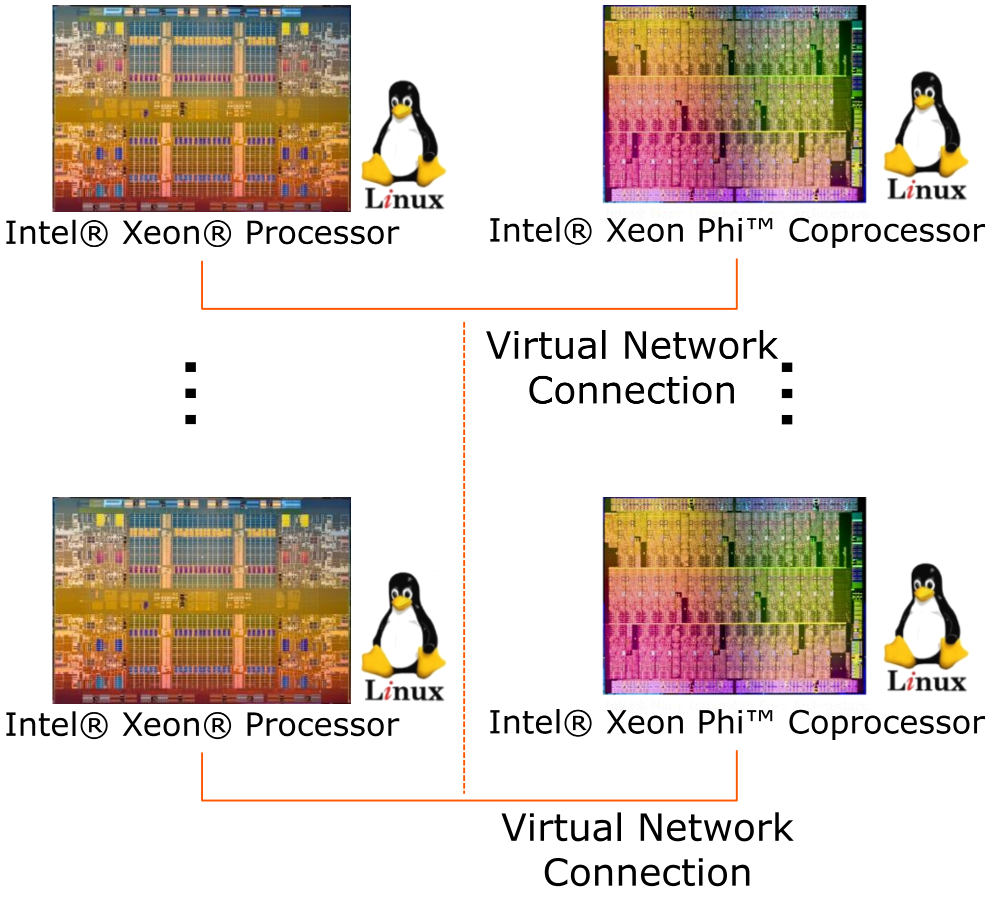
- Define thread affinity manually per single MPI process (pseudo syntax!):

```
# export OMP_NUM_THREADS=4
# mpiexec -env KMP_AFFINITY=[1-4] -n 1 -host myMIC ... :
          -env KMP_AFFINITY=[5-8] -n 1 -host myMIC ... :
          ...
```

Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing

Intel® Xeon Phi™ Coprocessor Becomes a Network Node


Linux
Intel® Xeon® Processor
Linux
Intel® Xeon Phi™ Coprocessor
Virtual Network Connection
Linux
Intel® Xeon® Processor
Linux
Intel® Xeon Phi™ Coprocessor
Virtual Network Connection

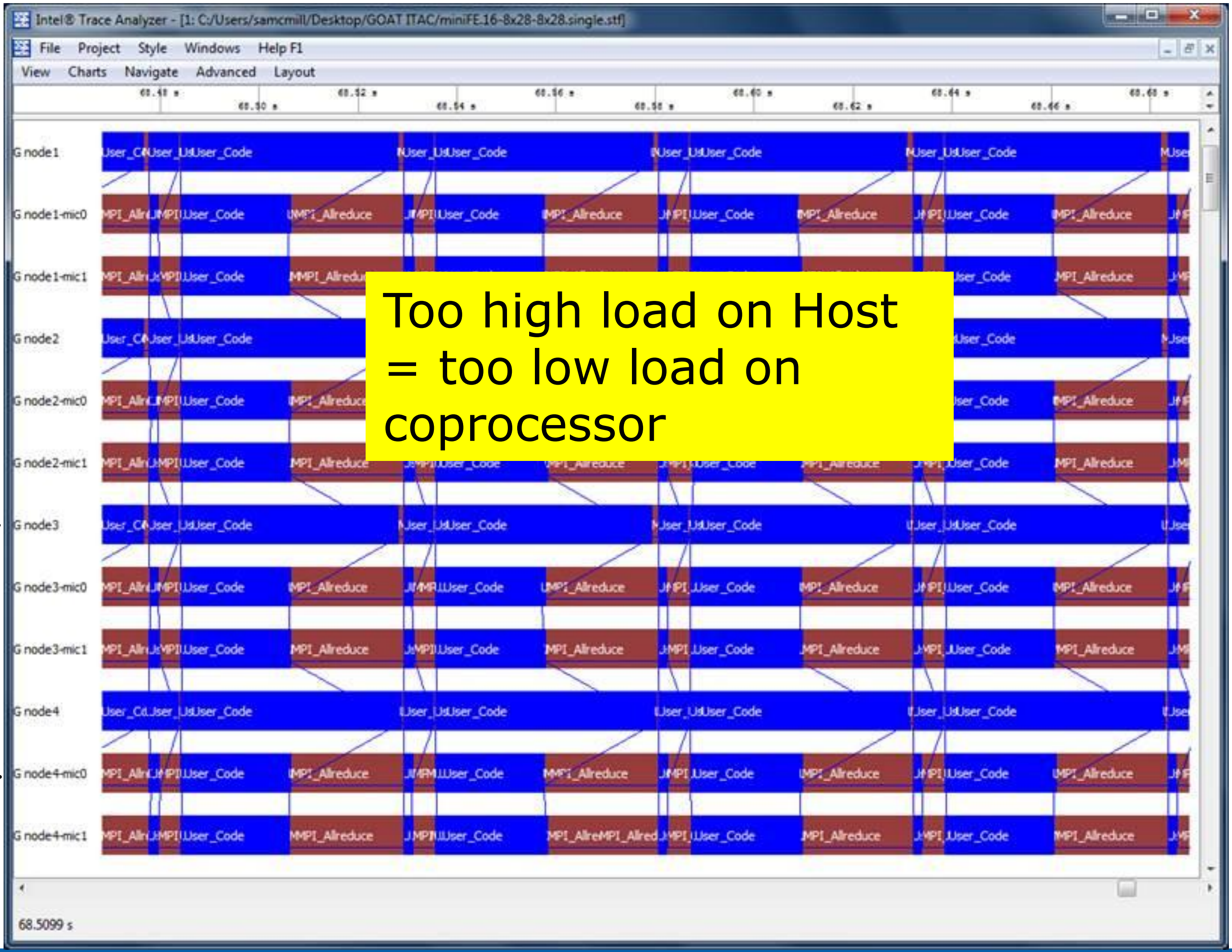
Intel® Xeon Phi™ Architecture + Linux enables IP addressability

Load Balancing

- Situation
  - Host and coprocessor computation performance are different
  - Host and coprocessor internal communication speed is different
- MPI in symmetric mode is like running on a heterogenous cluster
- Load balanced codes (on homogeneous cluser) may get imbalanced!
- Solution? No general solution!
  - Approach 1: Adapt MPI mapping of (hybrid) code to performance characteristics: #m processes per host, #n process per coprocessor(s)
  - Approach 2: Change code internal mapping of workload to MPI processes
    - Example: uneven split of calculation grid for MPI processes on host vs. coprocessor(s)
  - Approach 3: ...
- Analyze load balance of application with ITAC
  - Ideal Interconnect Simulator

Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card


Intel® Trace Analyzer - [1: C:/Users/samcmill/Desktop/GOAT ITAC/miniFE.16-8x28-8x28.single.stf]
File Project Style Windows Help F1
View Charts Navigate Advanced Layout
G node1
G node1-mic0
G node1-mic1
G node2
G node2-mic0
G node2-mic1
G node3
G node3-mic0
G node3-mic1
G node4
G node4-mic0
G node4-mic1
User_Code
MPI_Allreduce
68.5099 s

Too high load on Host = too low load on coprocessor

Host
16 MPI procs x
1 OpenMP thread

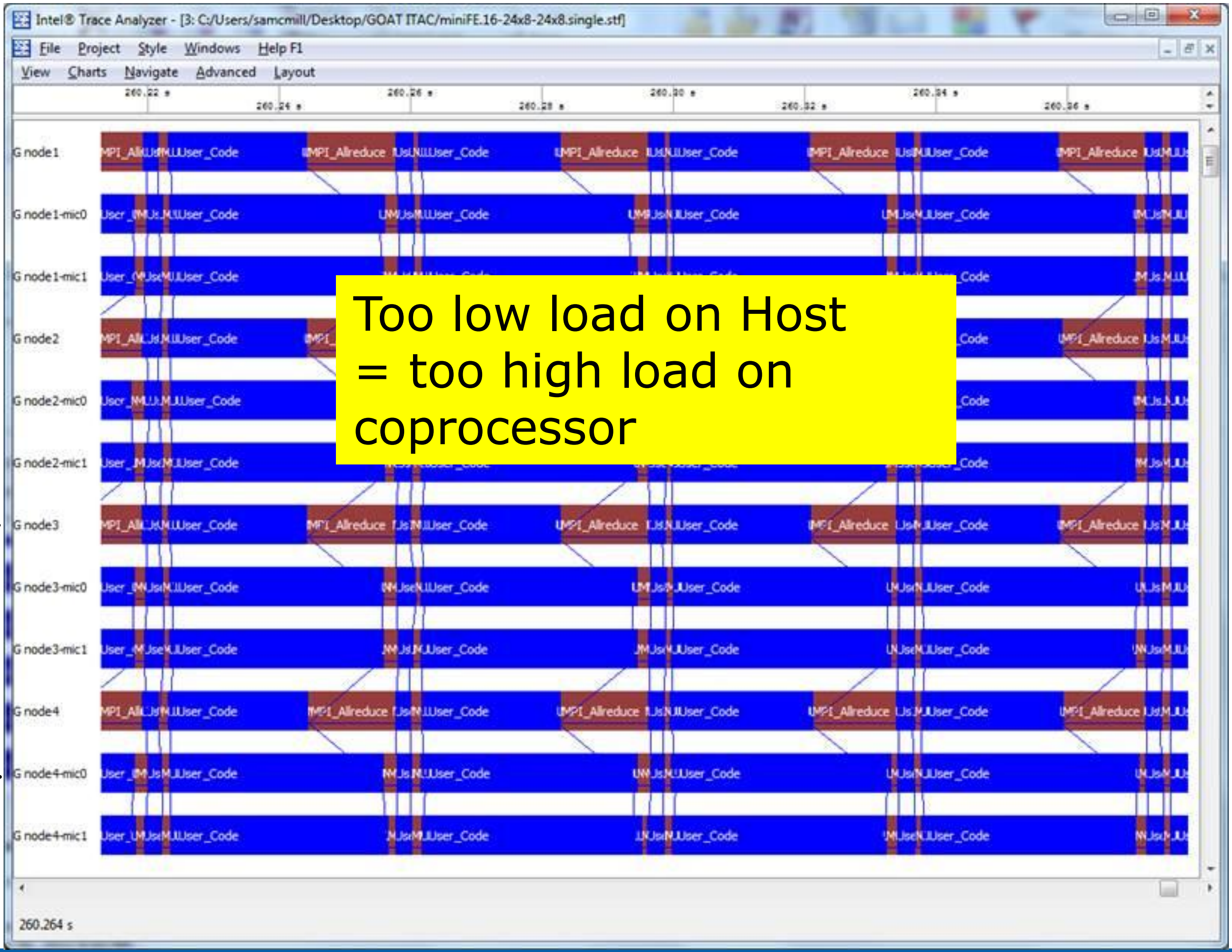
Coprocessor
8 MPI procs x
28 OpenMP threads

Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x
1 OpenMP thread

Coprocessor
24 MPI procs x
8 OpenMP threads


Intel® Trace Analyzer - [3: C:/Users/samcmill/Desktop/GOAT ITAC/miniFE.16-24x8-24x8.single.stf]
File Project Style Windows Help F1
View Charts Navigate Advanced Layout
node1
node1-mic0
node1-mic1
node2
node2-mic0
node2-mic1
node3
node3-mic0
node3-mic1
node4
node4-mic0
node4-mic1
MPI_Allreduce
User_Code
260.264 s

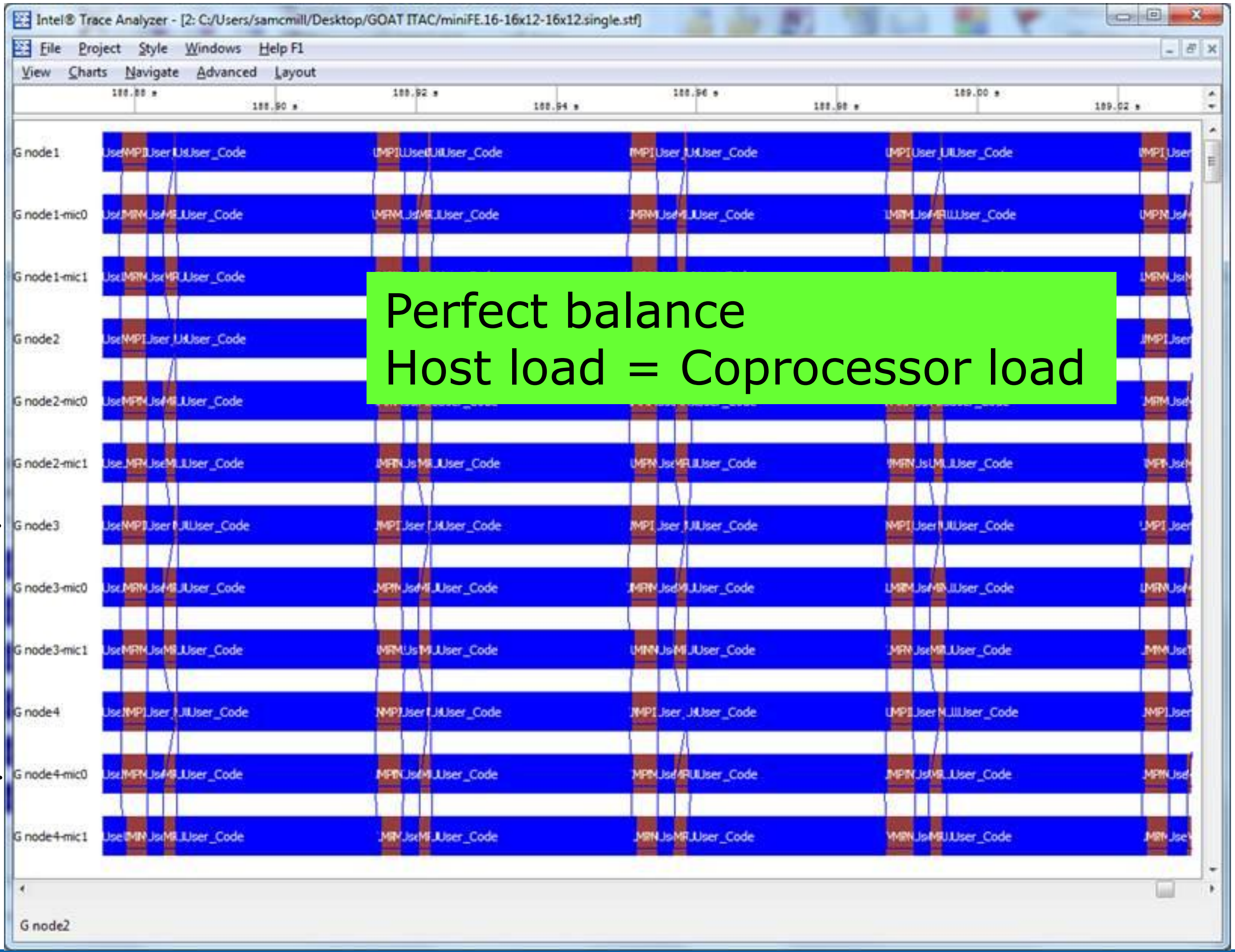
Too low load on Host = too high load on coprocessor

Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x
1 OpenMP thread

Coprocessor
16 MPI procs x
12 OpenMP thrds



Perfect balance
Host load = Coprocessor load

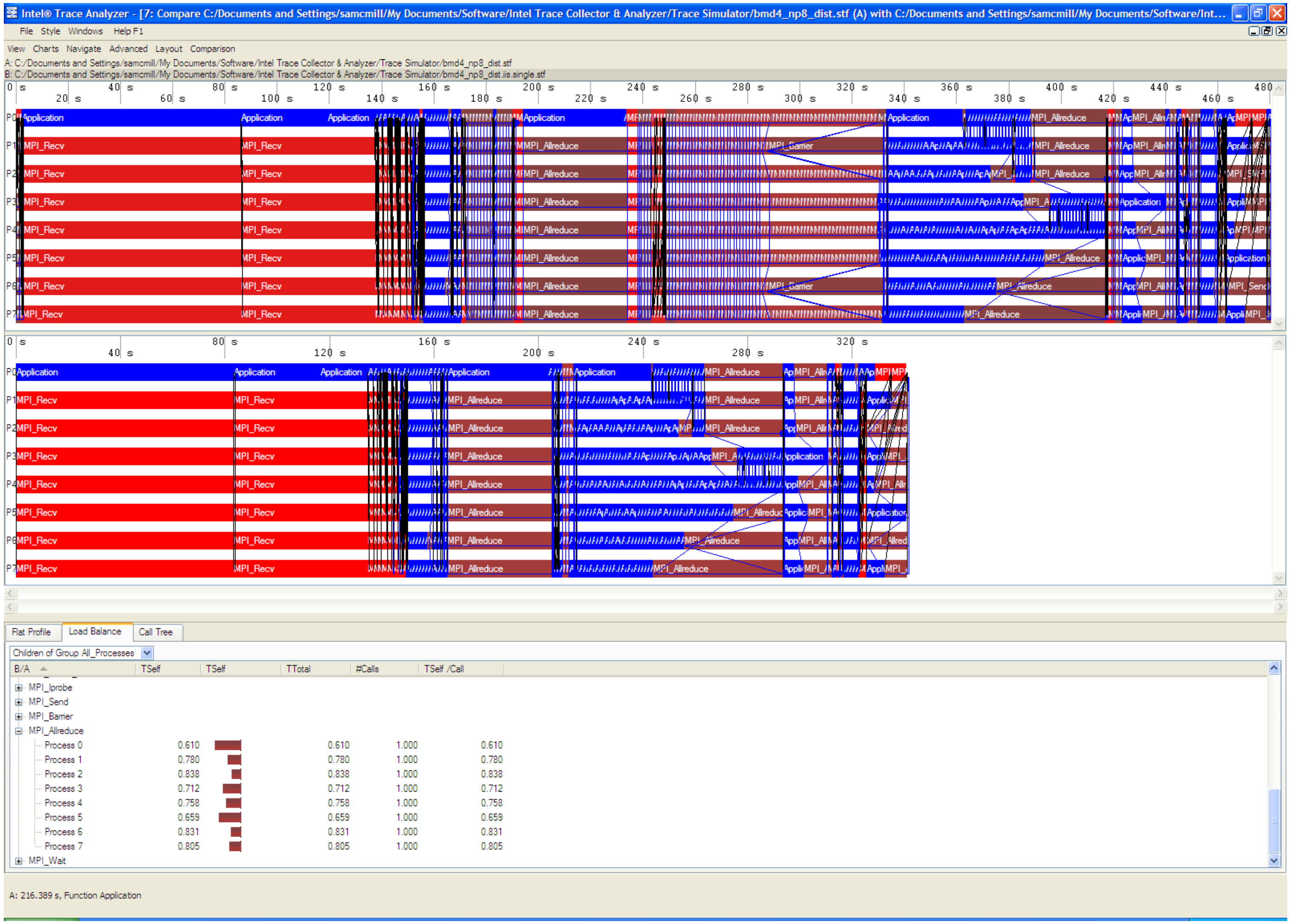
Intel® Xeon Phi™ Architecture

Software & Services Group, Developer Relations Division

Ideal Interconnect Simulator (IIS)

- What is the Ideal Interconnect Simulator (IIS)?
  - 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

Flat Profile | Load Balance | Call Tree

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					

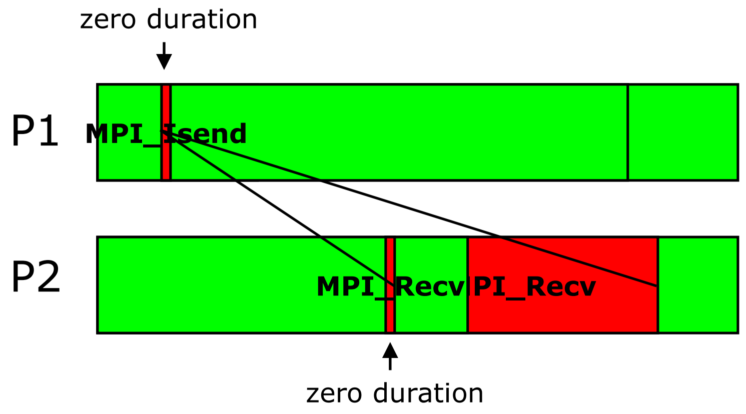
A: 216.389 s, Function Application

Actual trace

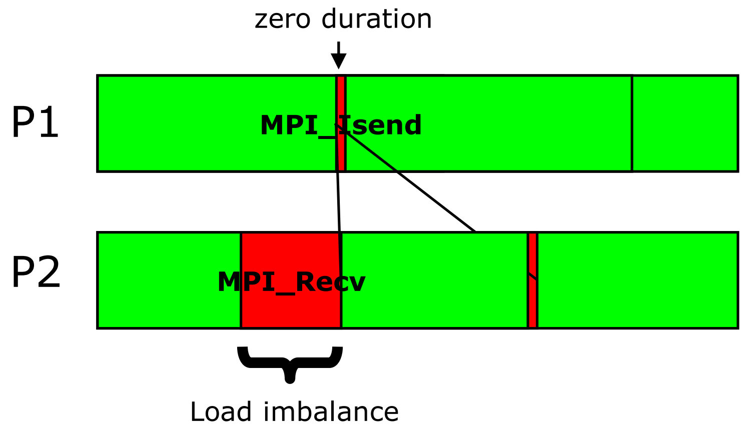
Idealized Trace

Building Blocks: Elementary Messages

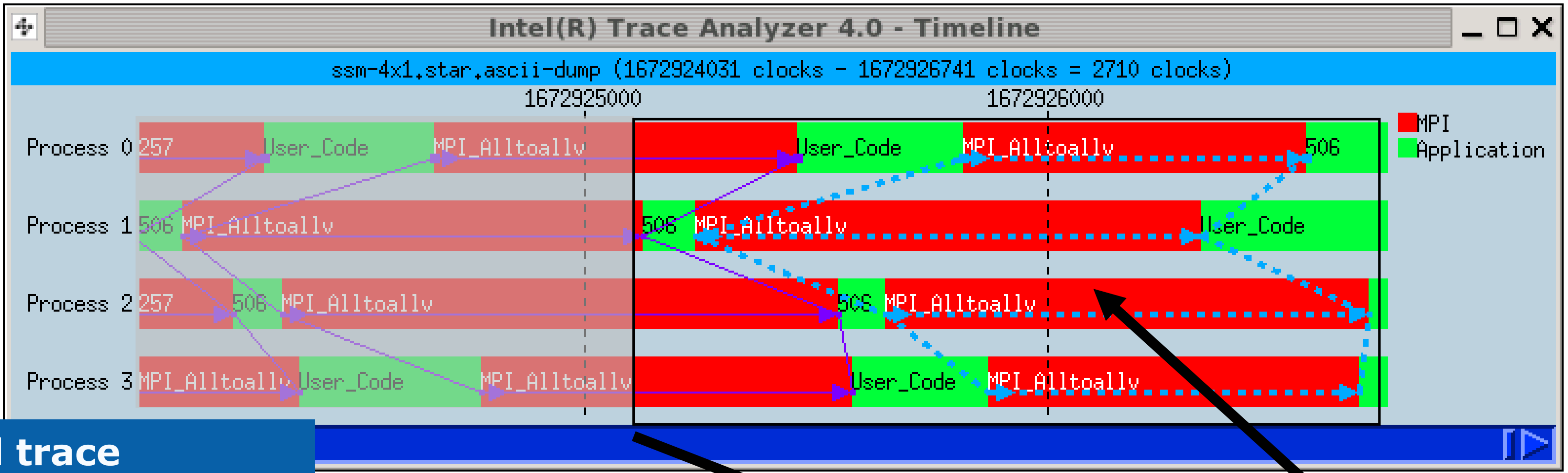
Early Send /
Late Receive



Late Send /
Early Receive

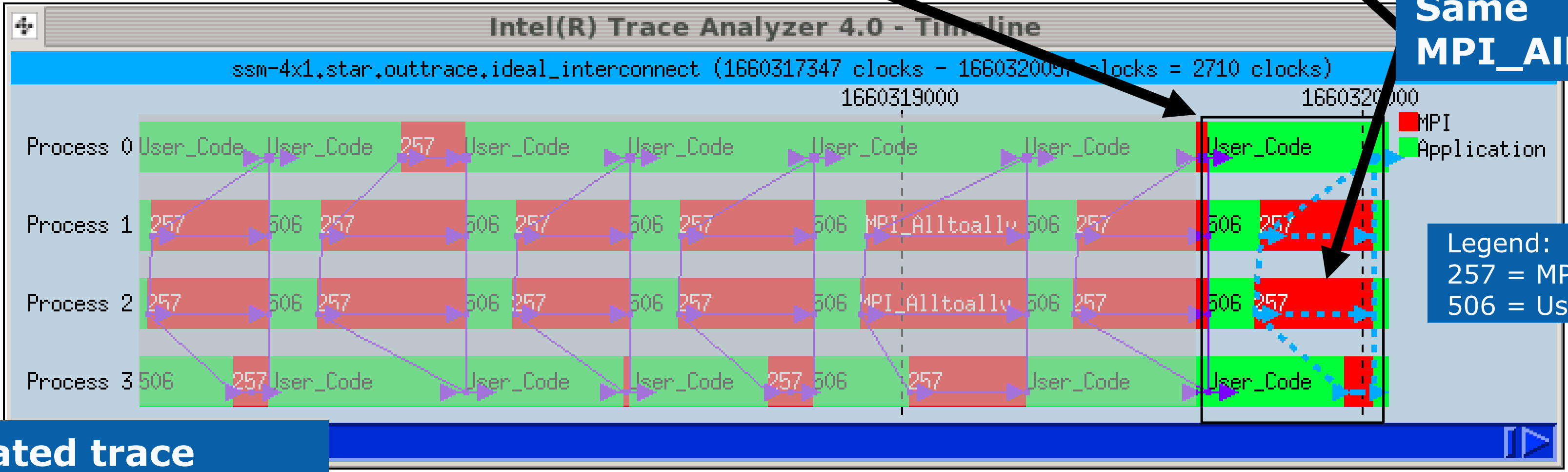


Building Blocks: Collective Operations



Intel(R) Trace Analyzer 4.0 - Timeline
ssm-4x1.star.ascii-dump (1672924031 clocks - 1672926741 clocks = 2710 clocks)
1672925000
1672926000
Process 0
Process 1
Process 2
Process 3
257
506
User_Code
MPI_Alltoallv
MPI
Application

Actual trace (Gigabit Ethernet)



Intel(R) Trace Analyzer 4.0 - Timeline
ssm-4x1.star.outtrace.ideal_interconnect (1660317347 clocks - 1660320057 clocks = 2710 clocks)
1660319000
1660320000
Process 0
Process 1
Process 2
Process 3
257
506
User_Code
MPI_Alltoallv
MPI
Application

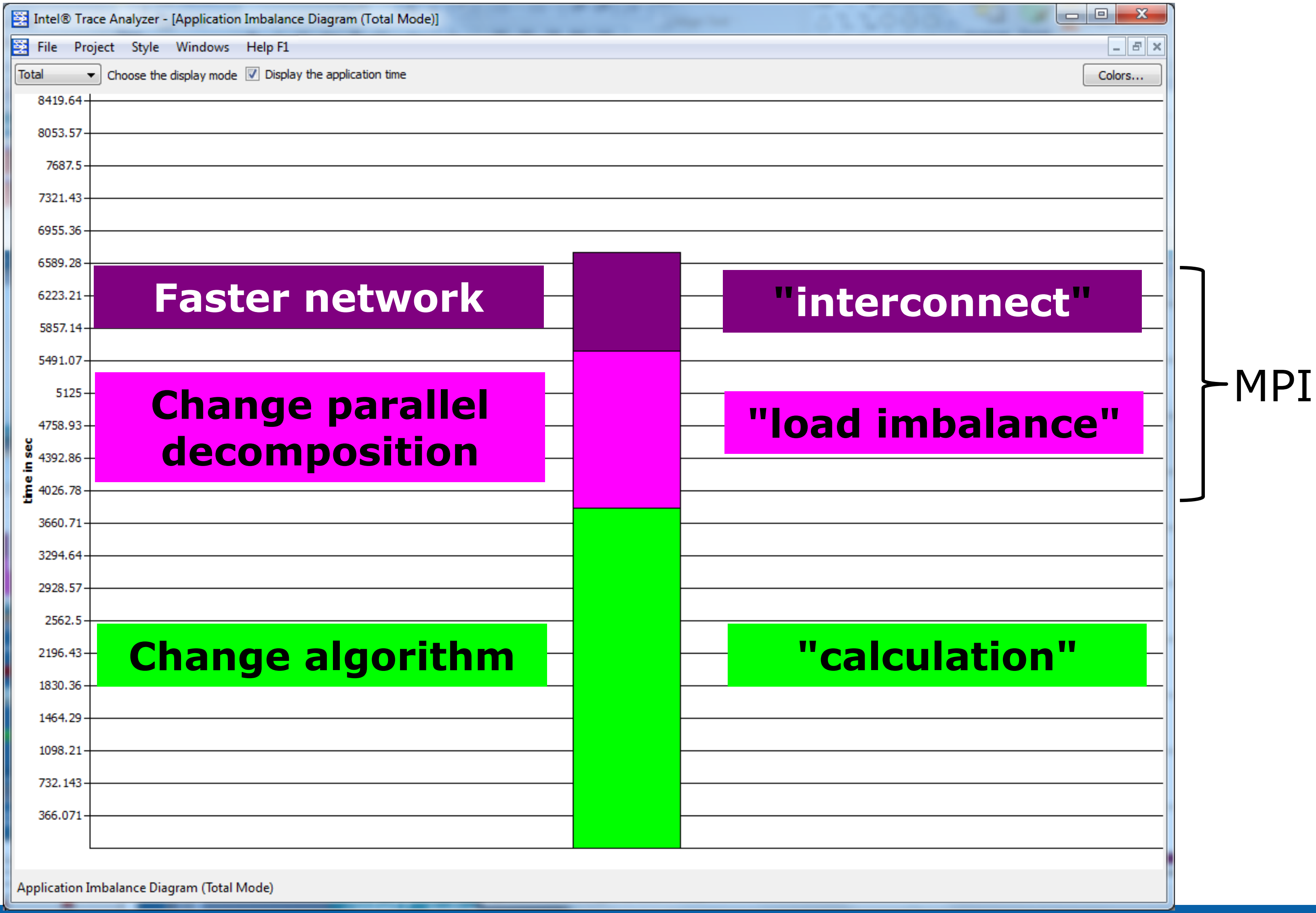
Simulated trace (Ideal interconnect)

Same MPI_Alltoallv

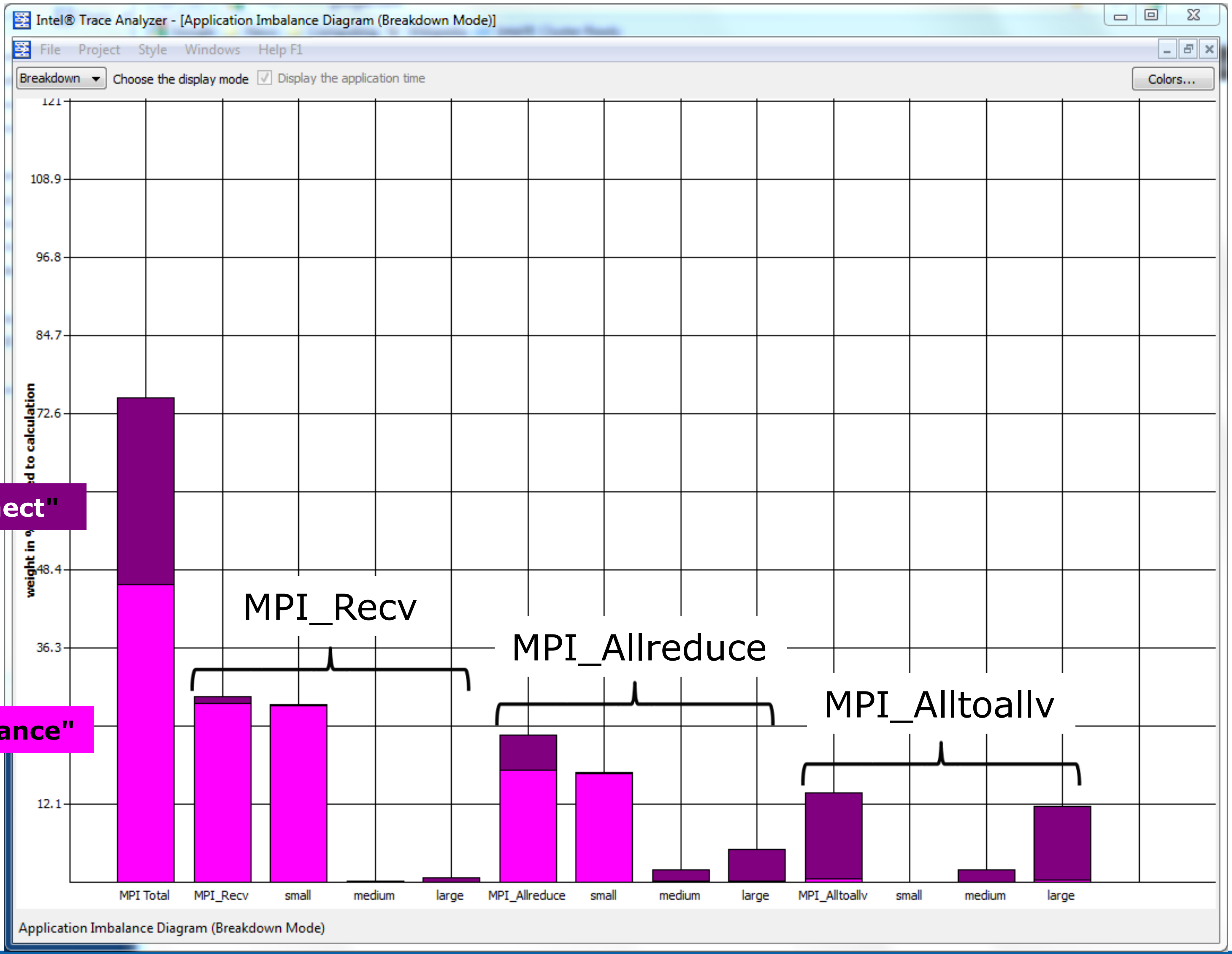
Legend:
257 = MPI_Alltoallv
506 = User_Code

Same timescale in both figures

Application Imbalance Diagram: Total



Application Imbalance Diagram: Breakdown



Questions?