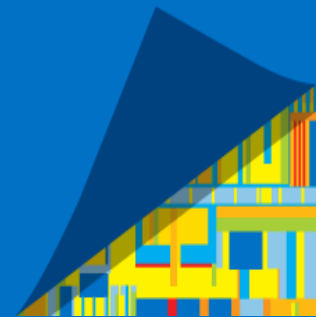
Intel® Parallel Studio XE 2016 Cluster Edition (Масштабирование MPI – производительность гибридных приложений)

Dmitry Sivkov, Michael Steyer

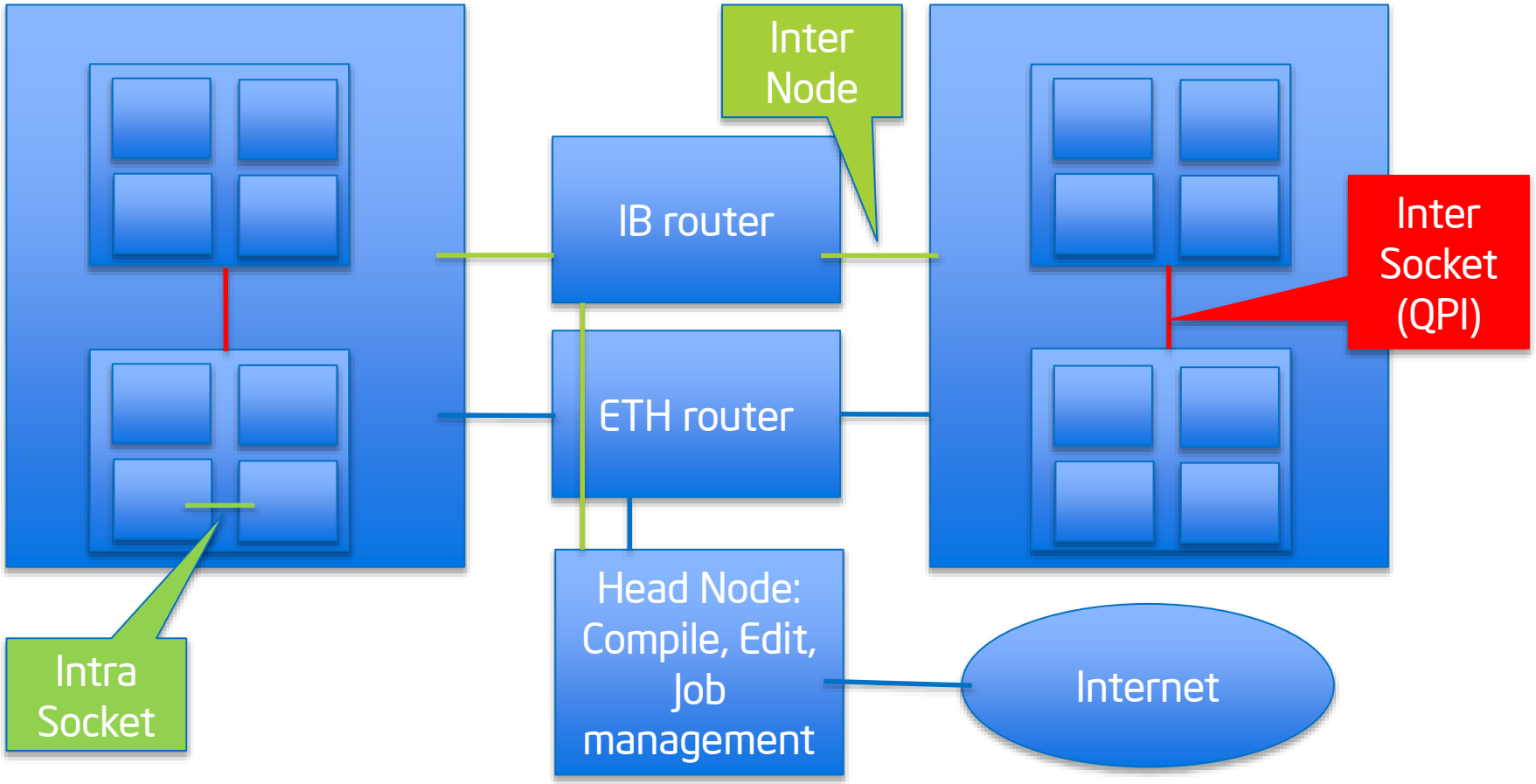
Technical Consulting Engineer

Intel



Challenges

Typical Structure of a Cluster



What Levels of Parallelism do exist?

- ILP - Instruction Level Parallelism
  - Pipelined Execution
  - Super-scalar execution
- DLP - Data Level Parallelism
  - SIMD (Single Instruction Multiple Data) vector processing
  - Implemented via vector registers and instructions
- TLP - Thread-Level Parallelism
  - Hardware support for hyper-threading
  - Multi-core architecture
  - Cache-coherent multiple sockets
- CLP - Cluster Level Parallelism
  - Multiple platforms connected via interconnection network
  - No hardware-supported cache coherence

Hybrid Computing

Combination of MPI programming model with threading model

=> Overcome limitations by adding threads

Memory gains in hybrid code

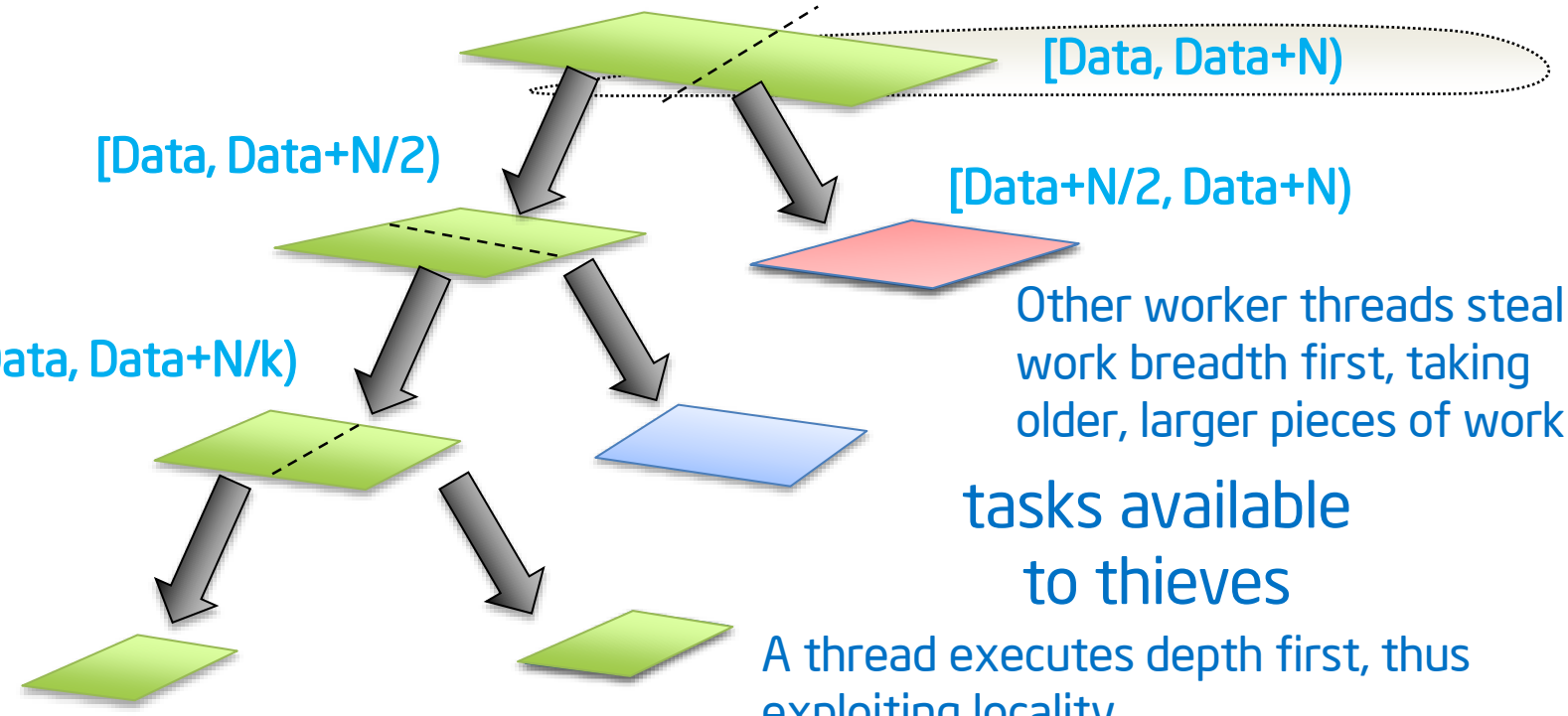
- Lower number of processes on the system
- Lower number of MPI – endpoints

Less MPI communication / more thread communication

Threading offers smart load balancing strategies

Result - maximize performance by better exploitation of available resources

Smart load balancing using Threads
Example: Intel® Threading Building Blocks (TBB) – Recursive Parallelism



Other worker threads steal work breadth first, taking older, larger pieces of work

tasks available to thieves

A thread executes depth first, thus exploiting locality

=> TBB is a tasking model - user deals with tasks rather than threads

What Threading Support does exist for MPI?

MPI_THREAD_SINGLE

Only one thread

MPI_THREAD_FUNNELED

Only the main thread executes MPI calls

MPI_THREAD_SERIALIZED

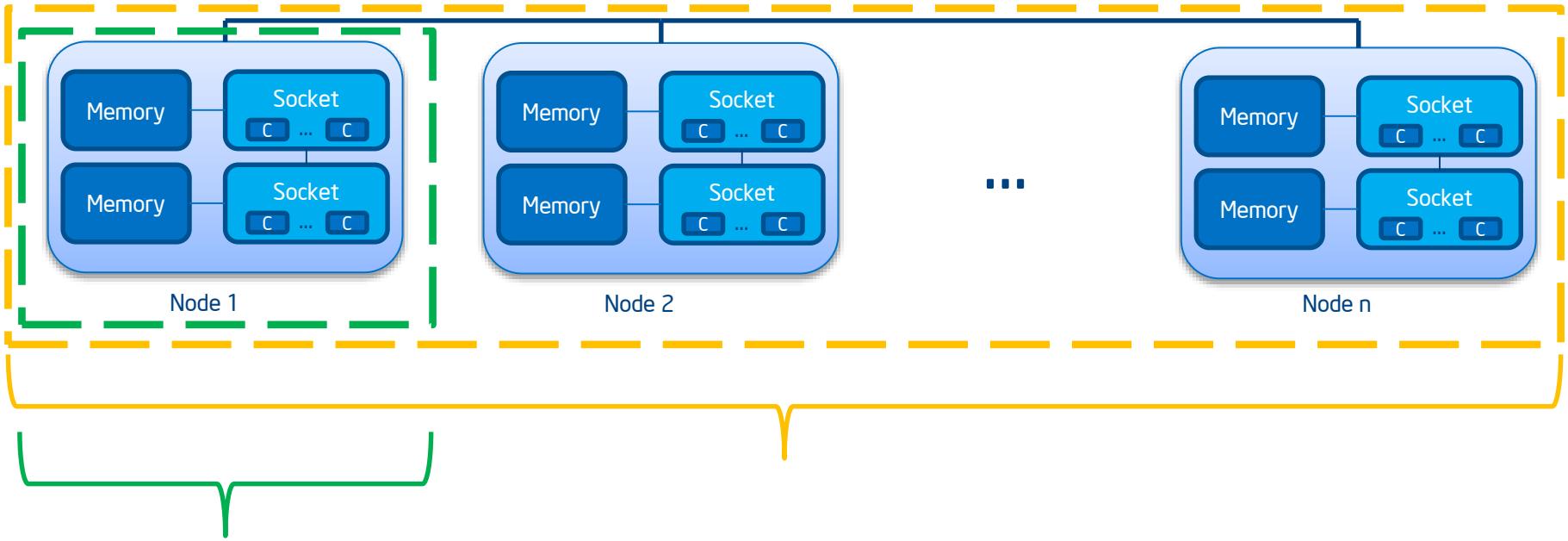
Threads exchange messages, only one at a time

MPI_THREAD_MULTIPLE

Threads exchange messages with other threads

Analyzing the Scalability of Your Code

Local- vs. Global Analysis



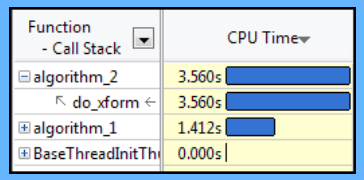
Intel® VTune™ Amplifier XE

Intel® Trace Analyzer & Collector (ITAC)
Intel MPI Performance Snapshot

Intel® VTune™ Amplifier XE
Performance Profiler

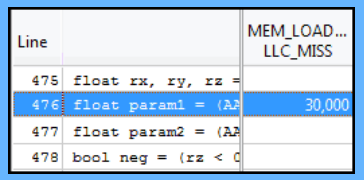
Where is my application...

Spending Time?



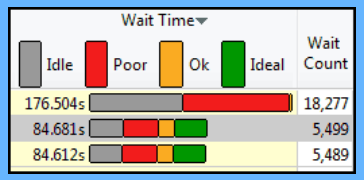
- Focus tuning on functions taking time
- See call stacks
- See time on source

Wasting Time?



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

Waiting Too Long?



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

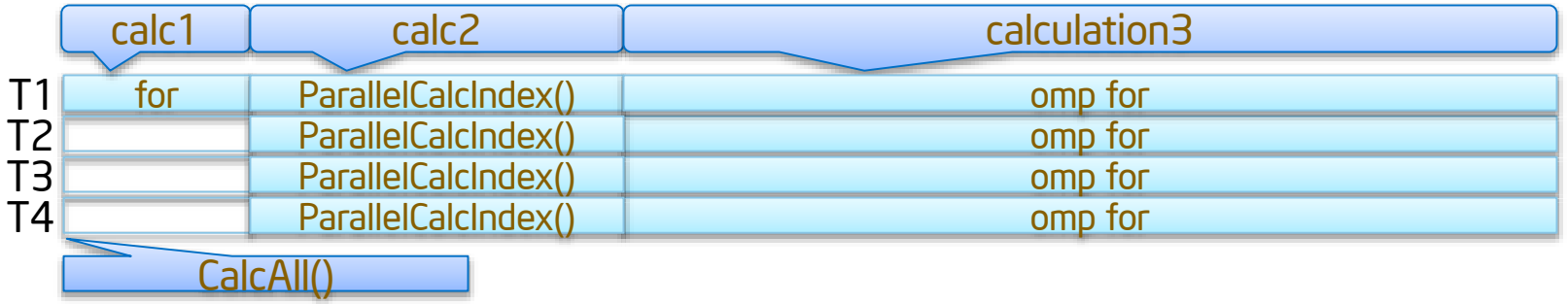
- Windows & Linux
- Low overhead
- No special recompiles

Continuously Improving Support for OpenMP

Vtune Amplifier is aware of all calls to Intel OpenMP run time which allows a detailed analysis of where the time is spend

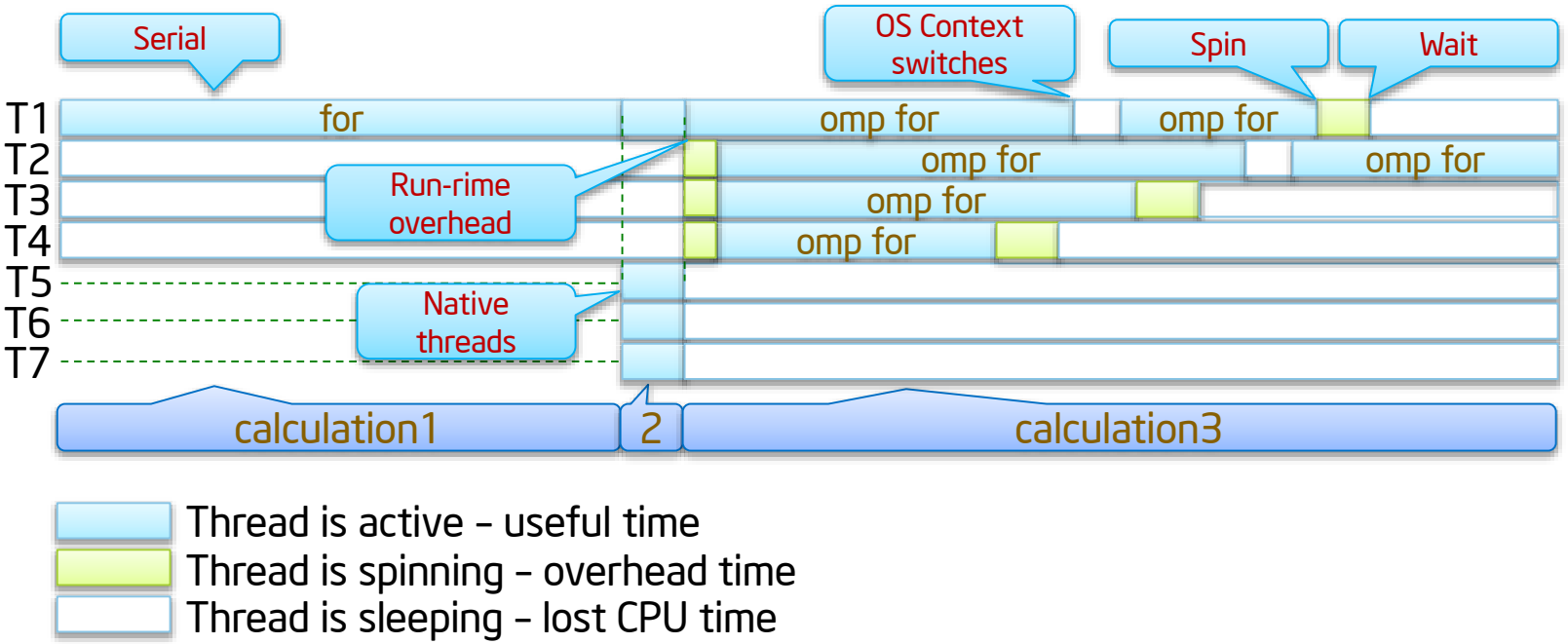
```
void CalcAll(double* p, int N)
{
  for(int i=0;i<N;i++) {…}   // calc 1
  ParallelCalcIndex(i, N);   // calc 2
  #pragma omp for
    for(i=0;  i<N; i++)      // calc 3
      {…}}
```

This is how you might believe the parallel application is working



VTune catches all OpenMP Threading Events

This is how the application might be working in reality



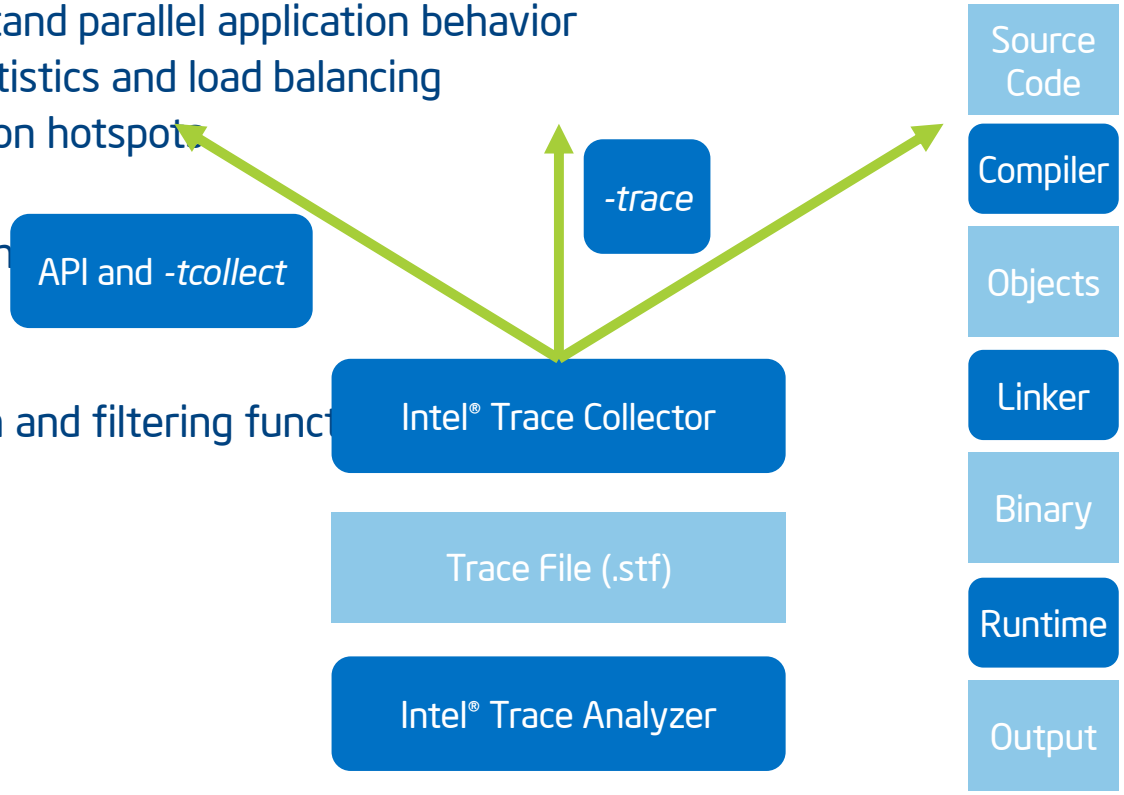
Intel® Trace Analyzer and Collector

Intel® Trace Analyzer and Collector (ITAC) helps to:

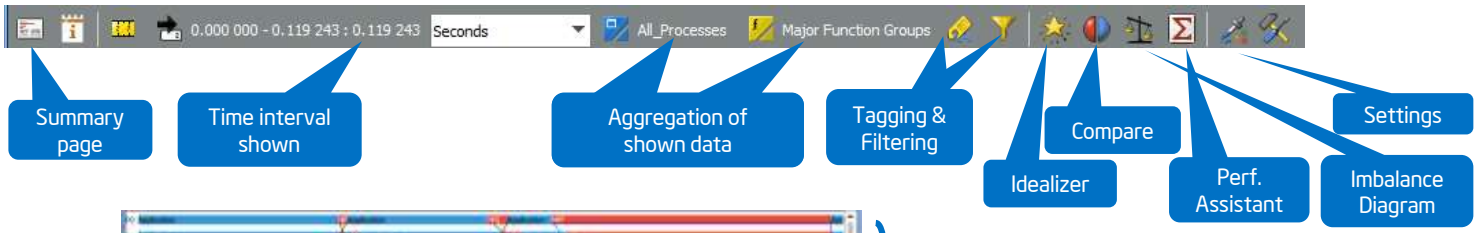
- Visualize and understand parallel application behavior
- Evaluate profiling statistics and load balancing
- Identify communication hotspots

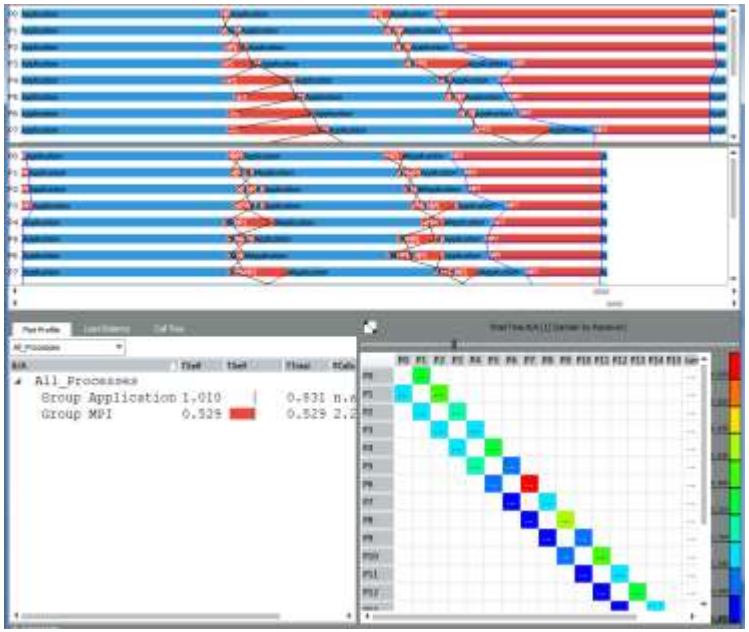
Features

- Event-based approach
- Low overhead
- Effective scalability
- Powerful aggregation and filtering funct
- Idealizer



Intel® Trace Analyzer and Collector





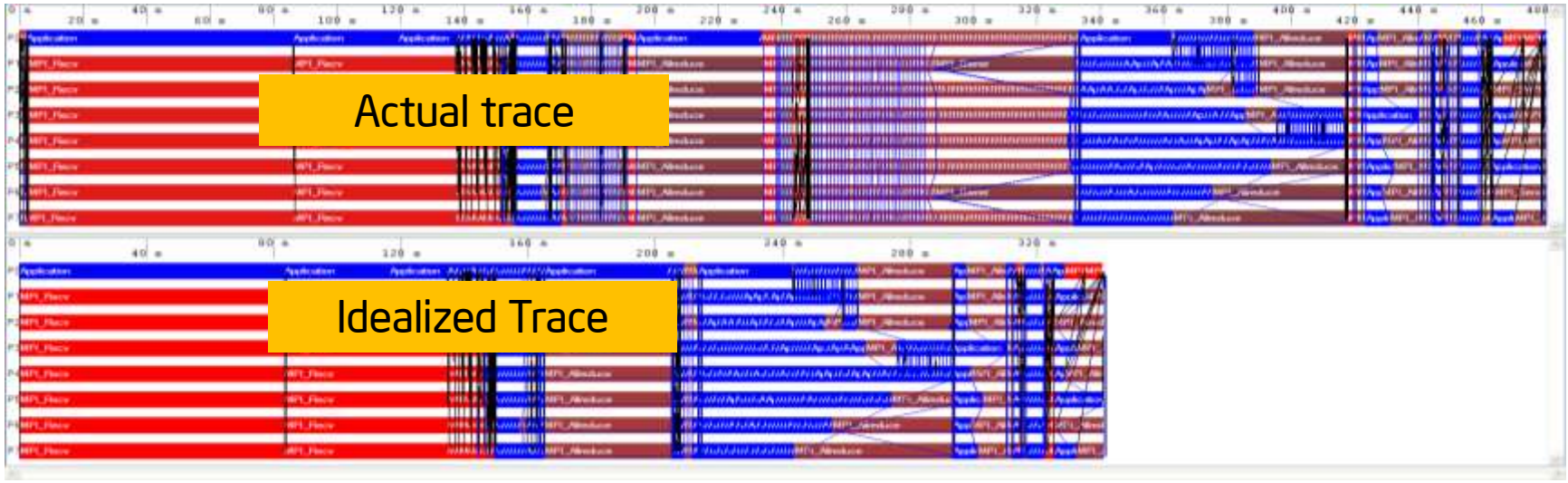
Compare the event timelines of two communication profiles

Blue = computation
Red = communication

Chart showing how the MPI processes interact

ITAC - Ideal Interconnect Simulator (Idealizer)

Is your code ready to scale?



Easy way to identify application bottlenecks

MPI Performance Snapshot (MPS)

Delivered with Intel® Trace Analyzer & Collector (ITAC)

- Intel has separated statistical analysis from ITAC event analysis
- The capability is available now via command line

New developer capability

- MPS enables the developer to quickly gather and analyze statistics on up to 32,768 ranks
- Shows data from HW counters and MPI/OpenMP imbalances
- Enables ITAC trace file targeting for deeper event based analysis.

Why MPI Performance Snapshot?

Advantage

- Get an initial profile of the application very quickly
- Performance variation at scale can be detected sooner
- Provides development recommendations to developers based on analysis
- Easy to use functionality for out of the box use

Benefits

- Difficult performance issues are easier to spot
- Application performance is improved faster
- Experienced & non-experienced developers can adopt quickly

MPI Performance Snapshot
Adding High Level Analysis

	Intel® Trace Analyzer & Collector Core Released Functionality (for Detailed Analysis)		MPI Performance Snapshot (MPS) Functionality (Beta) (for High Level Analysis)
Scalability	0 – 4K ranks		1 – 32K ranks (tested)
Trace details	High (events, source hooks)		Low (aggregation)
Trace size	Huge (~17 GB for 1K ranks)		Much less (~0,8 GB for 1K ranks; ~4.5Gb for 32K)
Event-based analysis	Yes	+	Set of HW events
Statistics analysis	Yes		Yes
Quick processing	No		Yes
Small & flexible	No		Yes
Collector	Intel® Trace Collector		Built-in & separate: IMPI Statistics + MPI/OMP imbalance

MPS Functionality Added to Intel Trace Analyzer & Collector

MPI Performance Snapshot[1]

High Capacity MPI Profiler

Lightweight Scalable analysis of MPI applications:

- Computations/communications ratio
- Compute imbalance
- Times for used MPI routines
- Message sizes distribution

```
$ source
<IMPI_installdir>/intel64/bin/mpiv
ars.sh

$ source
<ITAC_installdir>/bin/mpi_perf_sna
pshot_vars.sh


$ mpirun -mps -n 2 ./myApp

…

======== GENERAL STATISTICS
========

WallClock: 21.077 sec (All
processes)

MIN: 10.538 sec (rank 1)

MAX: 10.539 sec (rank 0)
```

[1] Available as preview in Intel® Parallel Studio XE 2015 Cluster Edition Update 1 for Linux OS

Easy to use.

MPI Performance Snapshot

High Capacity MPI Profiler – Low overhead

```
$ source
<ITAC_installdir>/bin/mpi_perf_snapshot_vars.sh
```

```
$ export
I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=disable
I_MPI_PIN_DOMAIN=socket OMP_NUM_THREADS=14
MKL_NUM_THREADS=14
```

Without MPS

```
$ time ( mpirun -n 4 -ppn 2
./xhpl_hybrid_intel64_dynamic )

…

real     4m51.676s

user     98m52.230s

sys      0m36.415s
```

With MPS

```
$ time ( mpirun -mps -n 4 -ppn 2
./xhpl_hybrid_intel64_dynamic )

…

real     4m53.572s

user     99m14.129s

sys      0m35.119s
```

Intel® MPI Performance Snapshot

High Capacity MPI Profiler - Memory & Counter Usage

New collector displays summary info immediately after end of application run. Detailed per process info can be found in output text file.

HW counters & memory usage info:

```
==================== GENERAL STATISTICS ====================
WallClock:          284.274 sec   (All processes)
      MIN:           31.998 sec   (rank 0)
      MAX:           35.534 sec   (rank 7)

================== HW COUNTERS STATISTICS ==================
GFlops:    9.563   MPI:  11.28%    NON_MPI:  88.72%

Floating-Point instructions:  45.77%
Vectorized  DP instructions:  24.69%
Memory  access instructions:  42.35%

================= MEMORY USAGE STATISTICS =================
All  processes:    256.740MB
           MIN:     30.608MB  (process      7)
           MAX:     33.136MB  (process      1)
```

Intel® MPI Performance Snapshot

High Capacity MPI Profiler - MPI & OpenMP Imbalance Analysis

```
================= MPI IMBALANCE STATISTICS =================
MPI Imbalance:          207.847 sec            73,12% (All processes)
          MIN:           23.044 sec            64,85% (rank  6)
          MAX:           30.113 sec            88,57% (rank  1)

==================== OpenMP STATISTICS ====================
OpenMP Regions:         228.631 sec           80,43%           56 region(s) (All processes)
           MIN:          25.348 sec           71,33%            7 region(s) (rank 7)
           MAX:          33.124 sec           97,42%            7 region(s) (rank 1)

OpenMP Imbalance:         103.924 sec            36,56%  (All processes)
             MIN:          11.522 sec            32,43%  (rank 3)
             MAX:          15.057 sec            44,29%  (rank 2)
```

MPI Performance Snapshot
High Capacity MPI Profiler - Diagrams

Further analysis:

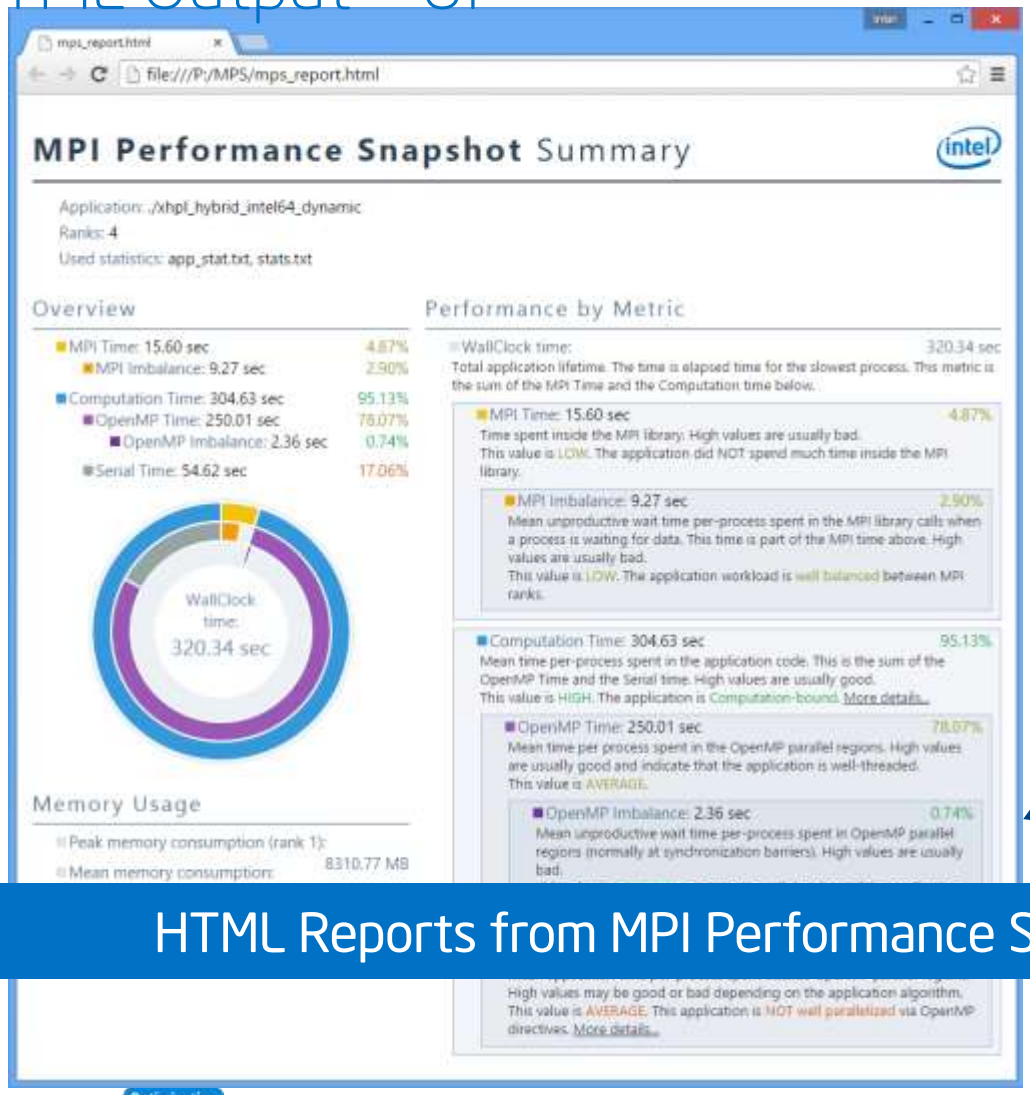
- Summary Information
  the essential information for novice users with further suggestions and guidance
- Counters and Memory usage
  general information about the application: execution time, memory and hardware resources usage
- Function summary
  the amount transferred data for each MPI function (for all ranks), and the time spent in each function
- MPI Time per Rank
  the time each rank spent in MPI functions
- Collective Operations Time
  the time each rank spent in MPI collective functions
- Message Sizes summary
  the amount of data transferred, and the number of MPI function calls for each message size used
- Data Transfers
  the transferred data volume for rank-to-rank, all ranks, MPI routines

MPI Performance Snapshot

High Capacity MPI Profiler – Diagram example

```
$ mpi_perf_snapshot -f ./stats.txt ./app_stat.txt
| Function summary for all ranks
|-------------------------------------------------------------------------------------------------------
|           Function        Time(sec)          Time(%)       Volume(MB)         Volume(%)             Calls
|-------------------------------------------------------------------------------------------------------
|               Send          25.1177           44.018          39443.6                50            130069
|              Probe          13.8178           24.2154                0                 0          12182371
|               Recv           10.337           18.1153          39443.6                50            130069
|               Wait          7.12479            12.486                0                 0            128997
|               Init          0.66177           1.15973                0                 0                 4
|             Gather       0.00261283        0.00457891      0.000366211       4.64221e-07                 4
| [skipped 2 lines]
|=======================================================================================================
| TOTAL                       57.0623               100          78887.2               100          12571530
```

HTML Output – UI



Application Analysis to Guide Development Efforts

HTML Reports from MPI Performance Snapshot

Optimization Notice

Tuning Your Code

Code- Modernization and Tuning

How to tune your code

- Algorithmic improvements
  - load balancing
  - Re- structuring of the workflow / memory
  - Alternative algorithms
- Use of non- blocking communication
- MPI 3 features
  - NBC
  - One-sided communication
  - Shared Memory
- OpenMP 4 Features
  - Threading
  - Vectorization

Tuning the MPI Library

… automatically …

MPI Tuning Motivation

"The **optimal algorithm** and the **optimal buffer** size for a given **message size** depends on a given configuration of the system including the gap values of the networks, memory models, the underlying communication layer etc. The **optimal parameters** for a system **can be best determined by conducting experiments on the system**."

[Automatically Tuned Collective Communications –

Sathish S. Vadhiyar, Graham E. Fagg, Jack Dongarra –

Computer Science Department - University of Tennessee, Knoxville]

Why automatic tuning?

- Intel® MPI library comes with wide area of configuration options - Environment variables all over
  - different algorithms for collective operations
  - different fabrics supported
  - different thresholds for library behavior (rendezvous, cache bypassing, ...)
  - different buffer sizes
  - ...
- Tuning the library itself – not the user application
- Library however, has already very good out-of-box settings ...
- MPITune is a tool that allows to explore the search space automatically

Tuning the MPI Library

… manually …

Prerequisite –
Make sure your cluster is properly configured

- Install the latest Intel® MPI Library
- Have the documentation at hand
- When benchmarking try to run on the same subset of nodes
- Understand the performance characteristics of your cluster
  - What is the lowest achievable latency
  - What is the maximum achievable bandwidth
  - What fabrics are available, how do they perform
  - Communication speeds differ
    - Intra-socket / Inter-socket
    - Intra-node / Inter-node
  - Use IMB to determine performances

Use best available communication fabric

Supported I_MPI_FABRICS	Description
shm	Shared-memory only; intra-node default
tcp	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand* (through IPoIB*)
dapl	DAPL–capable network fabrics, such as InfiniBand*, iWarp*, and XPMEM* (through DAPL*)
ofa	OFA-capable network fabric including InfiniBand* (through OFED* verbs)
tmi	TMI-capable network fabrics including Intel IB, Myrinet* (through Tag Matching Interface)

- Intel MPI will select fastest available fabric by default (shared memory within a node and InfiniBand* across nodes – shm:dapl)
- Disable fallback to ensure fabric usage

Use connectionless communication

The connectionless feature works for the ‘dapl’ and ‘tmi’ fabrics only

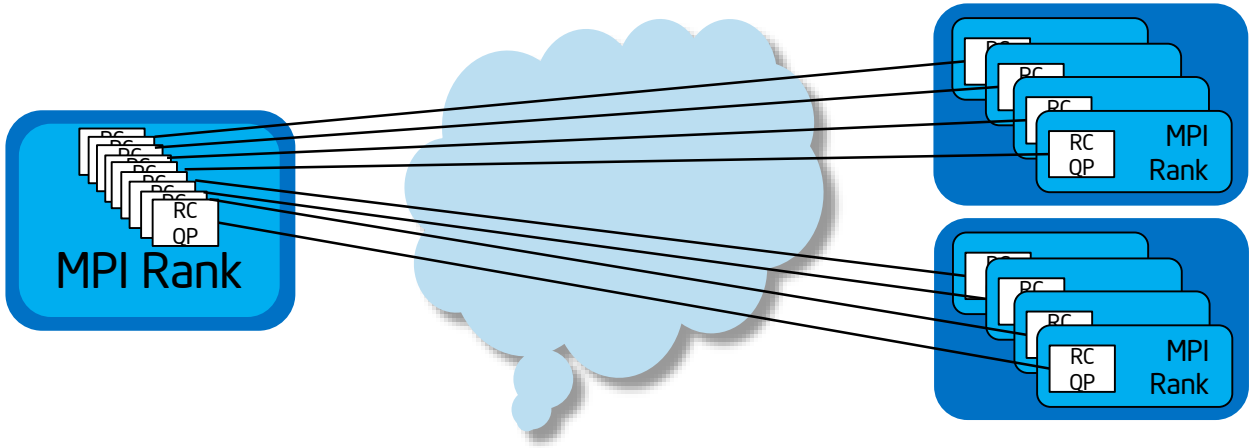
- Provides better scalability
- Significantly reduces memory requirements by reducing the number of receive queues
- Generally use for large jobs

```
$ export I_MPI_FABRICS=shm:dapl
$ export I_MPI_DAPL_UD=enable
```

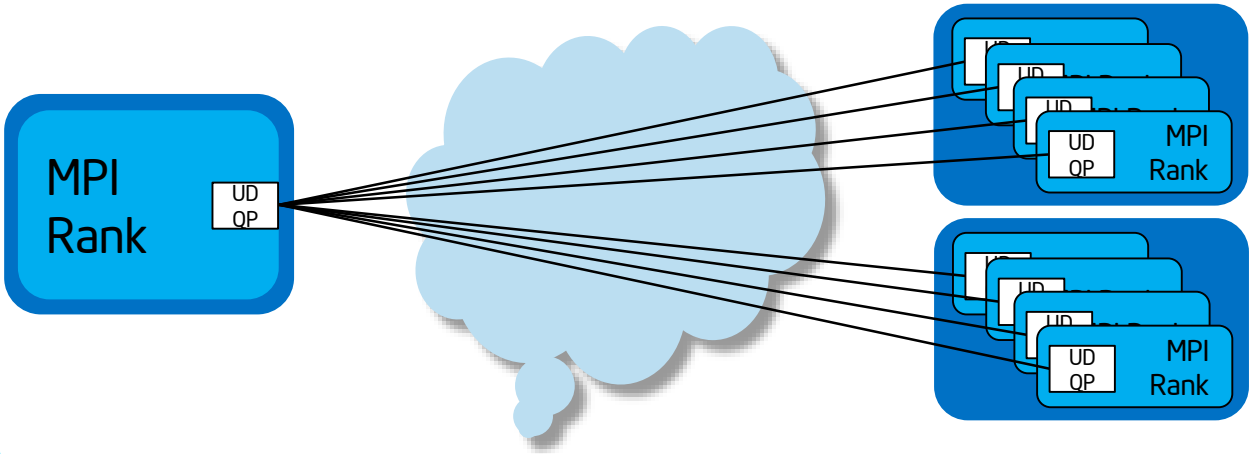
Use connectionless communication – Reliable Connection (RC) vs. User Datagram (UD)

(QP – Queue Pair)

RC



UD

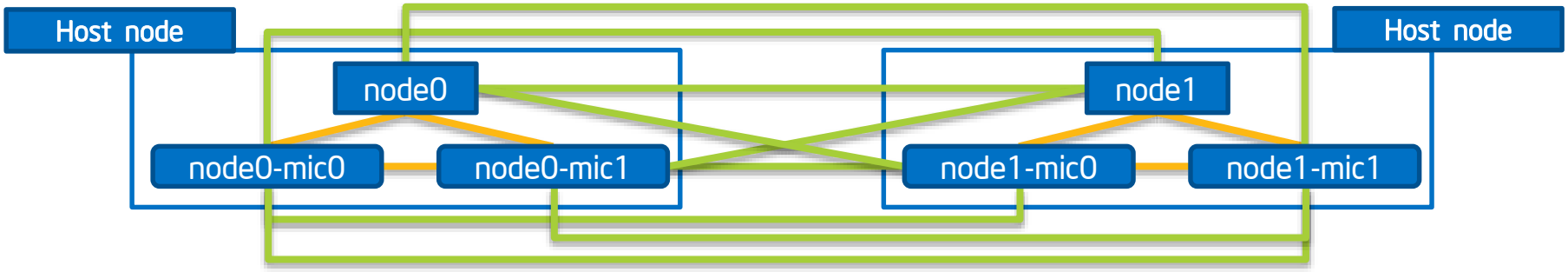


Optimization Notice

Use Multiple DAPL* Providers for Xeon Phi™

Multiple providers can be used based on message parameters

- I_MPI_DAPL_PROVIDER_LIST=<Prov1>,<Prov2>,<Prov3>
- Recommend using defaults for automatic selection
- All small messages use Prov1
- Large messages use Prov2 or Prov3
  - Prov2 if within same host node
  - Prov3 if not



Choose the best collective algorithm

- Use one of the I_MPI_ADJUST_<opname> knobs to change the algorithm
- Recommendations:
  - Focus on the most critical collective operations (see stats output)
  - Run the Intel MPI Benchmarks by selecting various algorithms to find out the right protocol switchover points for hot collective operations
  - … or … use mpitune

```
$ mpirun -genv I_MPI_ADJUST_REDUCE <algorithm #> …
```

Select proper process layout

- Default process layout is that all physical cores will be used
- If running hybrid applications, might want to reduce the number of ranks/node
- Set I_MPI_PERHOST or use the -perhost (/-ppn) option to override the default process layout:

```
$ mpirun –ppn <#processes per node> -n <#processes> …
```

- Same can be achieved using a “machinefile”
- On batch scheduler environments, the Intel MPI library respects the scheduler settings
- To overwrite the batch scheduler settings (at your own risk ☺):

```
$ export I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0
```

Use proper hybrid process pinning

- Link with thread safe library (**-openmp** / **-mt_mpi**)
- Choose MPI threading model (SINGLE / **FUNNELED** / SERIALIZED / MULTIPLE) – either using MPI_Init_thread(...) or env. var. I_MPI_THREAD_LEVEL_DEFAULT

```
$ export I_MPI_THREAD_LEVEL_DEFAULT=SINGLE
```

- Choose distribution of MPI ranks & threads – (ranks x threads = cores)

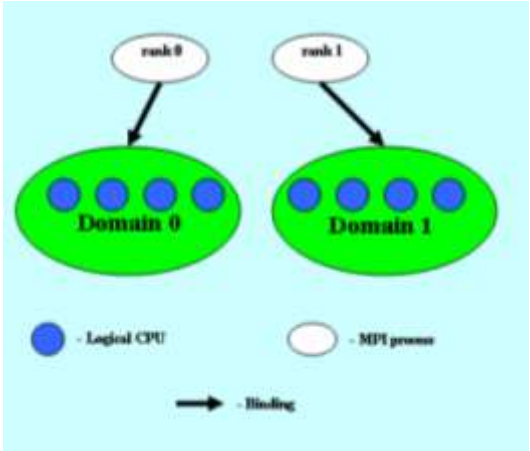
```
$ mpirun –n <#ranks> -genv OMP_NUM_THREADS <#threads>
```

- Pin MPI ranks using I_MPI_PIN_DOMAIN (e.g. „omp“ according to #OpenMP t.):

```
$ export I_MPI_PIN_DOMAIN=omp
```

- Pin threads e.g. KMP_AFFINITY

```
$ export KMP_AFFINITY=compact
```

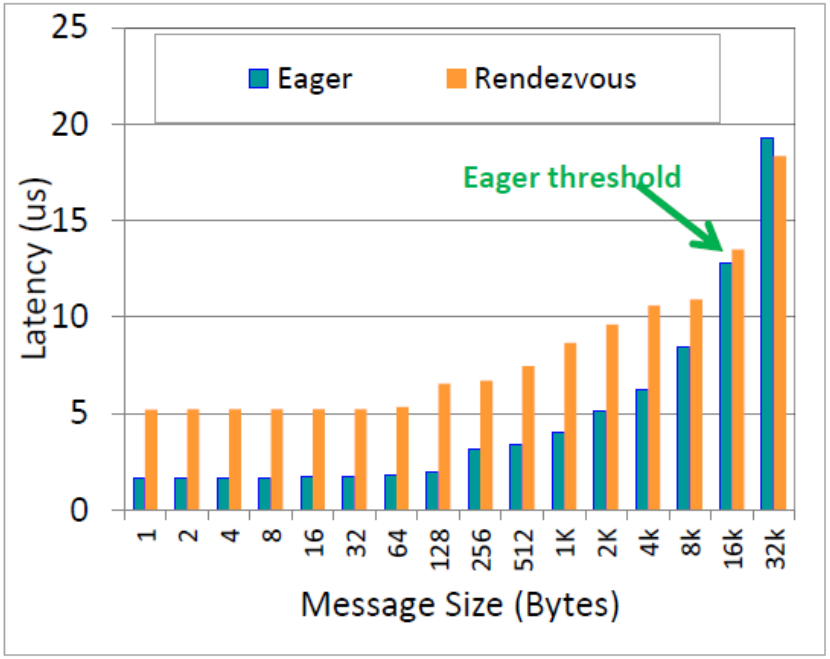


Adjust the eager / rendezvous protocol threshold

Two communication protocols:

"Eager" sends data immediately regardless of receive request availability and uses for short messages

"Rendezvous" notices receiving site on data pending and transfers when receive request is set



Eager vs Rendezvous

```
$ export I_MPI_EAGER_THRESHOLD=<#bytes>
```

Tuning for Memory Consumption

Reduce the amount of memory consumed by the DAPL- provider on large scale applications

Environment Variable	Default Value	Tuned Value
I_MPI_DAPL_UD_SEND_BUFFER_NUM	Runtime dependent	8208
I_MPI_DAPL_UD_RECV_BUFFER_NUM	Runtime dependent	8208
I_MPI_DAPL_UD_ACK_SEND_POOL_SIZE	256	8704
I_MPI_DAPL_UD_ACK_RECV_POOL_SIZE	Runtime dependent	8704
I_MPI_DAPL_UD_RNDV_EP_NUM	4	2

Intel MPI Library ships ready configurations

Located in **`$I_MPI_ROOT/intel64/etc`** you can find some configurations for common benchmarks

```
$ cat $I_MPI_ROOT/intel64/etc/hpl.conf
```

```
-genv I_MPI_EAGER_THRESHOLD 128000 # or 134000
-genv I_MPI_FALLBACK_DEVICE disable
-genv I_MPI_RDMA_RECV_QUEUE_SIZE 0
-genv OMP_NUM_THREADS 1
-genv I_MPI_FAIR_READ_SPIN_COUNT 10000
-genv RDMA_DEFAULT_MAX_WQE 500
-genv RDMA_READ_RESERVE 20
```

Summary

Summary

- Hybrid computing is key to reduce the memory footprint compared to pure MPI codes
- Analysis tools will help to know where to tune
- Code Modifications adopting newer standards can increase scalability
- … but … don't forget the low hanging fruits …
- Automatic MPI library tuning can help to find the optimal settings
- Manual MPI library tuning is still essential to some degree

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.

Materials distributed for lab sessions may redistribute source codes obtained under various Open Source licenses with additional materials supporting the lab distributed under the Intel Sample Source Code License. A copy of this latter license should be an included file within the distribution and covers this presentation along with lab source samples. The passed-through **Open Source projects might not perform as well as the originals**, since lab preparation may include the de-optimization of certain sections to provide lab examples for analysis tool exercises.

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