MPI Analysis

Agenda

Performing a scaling analysis supported by ITAC

Simulation of run time using an ideal network

Splitting run time into components (compute, wait,...)

Analysis of message passing structure

Detailed Visualization of MPI programs

Analysis of program structure (non MPI) with Intel® VTune™ Amplifier XE

Summary

Simple Scaling analysis

First step may be to just run the program for various number of processes [p] and record timings: T[p]

Speedup S is defined as: S[p] = T[1]/T[p]

Efficiency E is defined as: E[p] = S[p]/p

An ideal parallel program will show:
S[p] = p and E[p] = 1

Benchmark Cluster

Intel® Xeon® E5-2697 v2 processors Ivy Bridge (IVB) with 12 cores. Frequency: 2.7 GHz

2 processors per node (→ 24 cores per node)

Mellanox MT4099 QDR Infiniband

Operating system: RedHat EL 6.5

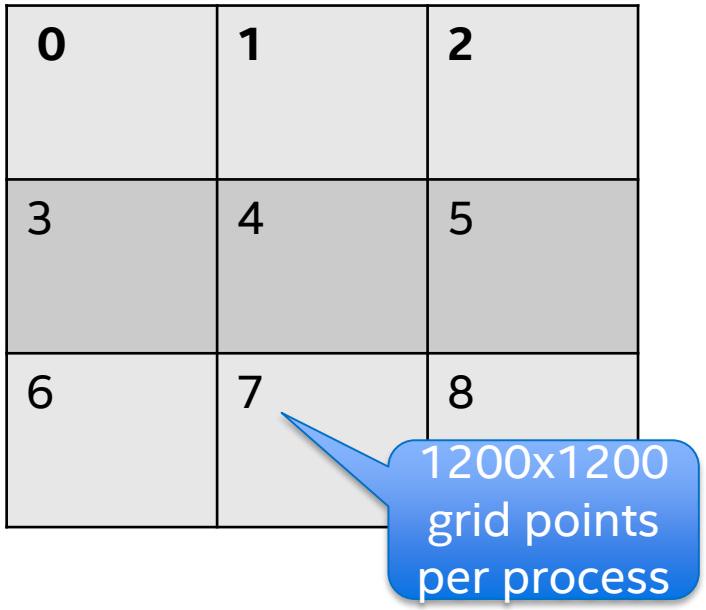
Intel® MPI 5.0.0.028

Test Application: Poisson Solver

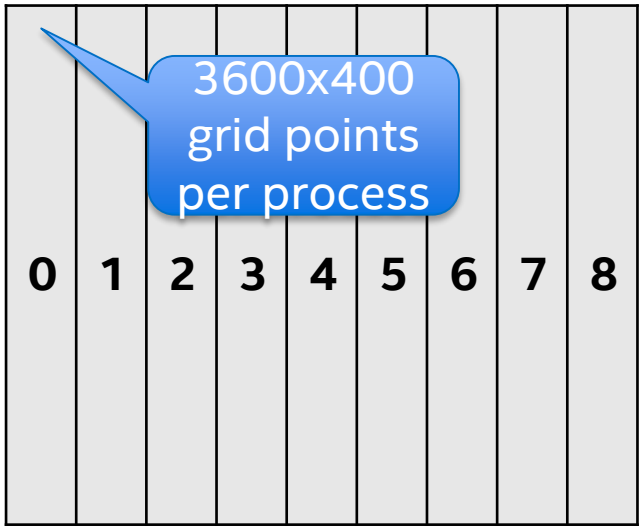
- Very simple implementation of Poisson solver: e.g. heat equation
- We will investigate a square 3600x3600 computational grid. It will be large enough to run into bandwidth limitations
- Grid points will be distributed to MPI ranks on a 2D process grid: e.g. 9 ranks = 3 rows x 3 columns (see next slide). The Cartesian Process grid is a feature of this Poisson solver. Other programs can have different data distributions. This example is discussed in the classical MPI book: *Using MPI* by Gropp,Lusk and Skjellum in Chapter #4

Choice of process grid

Which choice of process grid is optimal? Total grid: 3600x3600

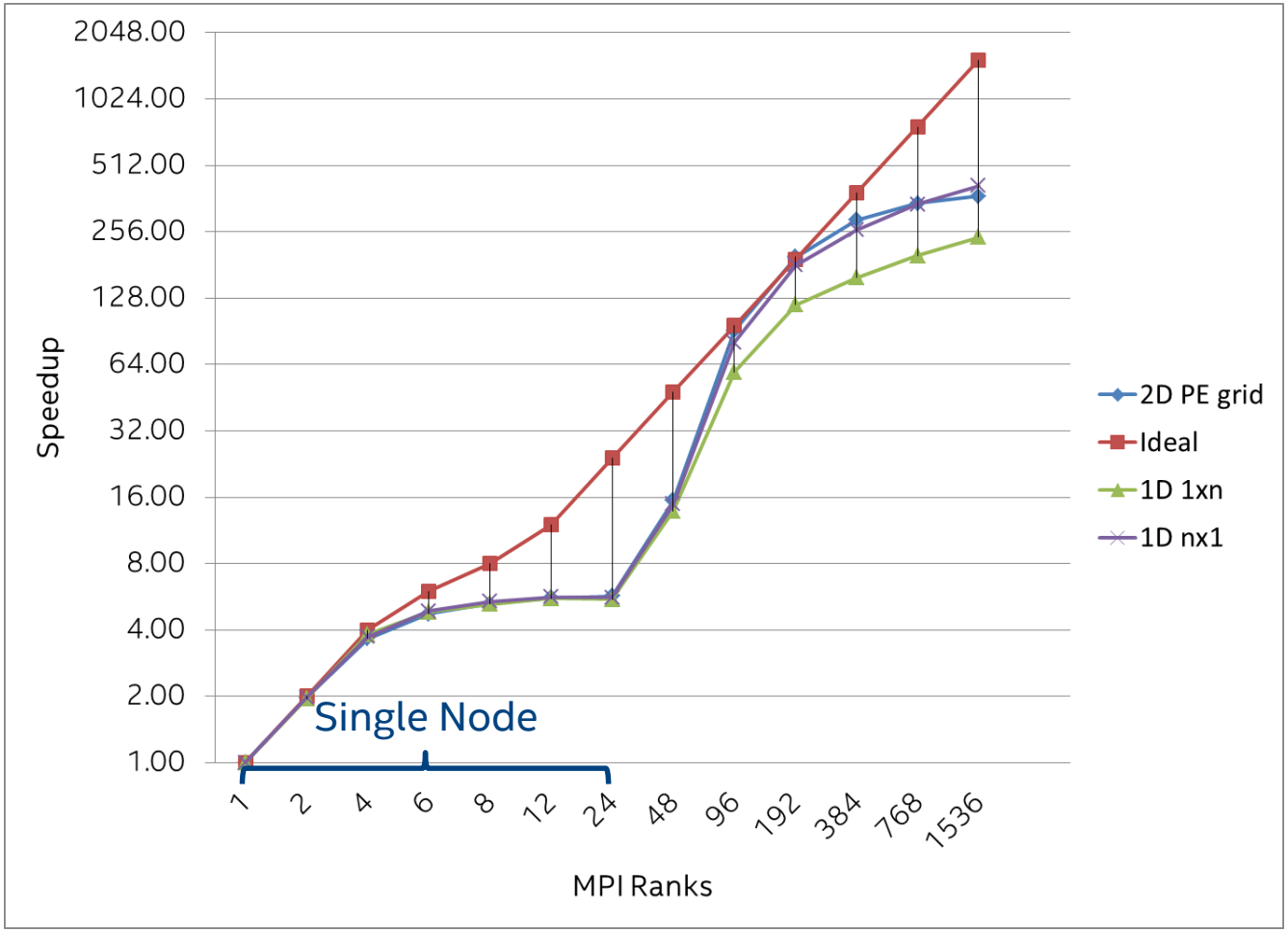


3x3 2D process grid

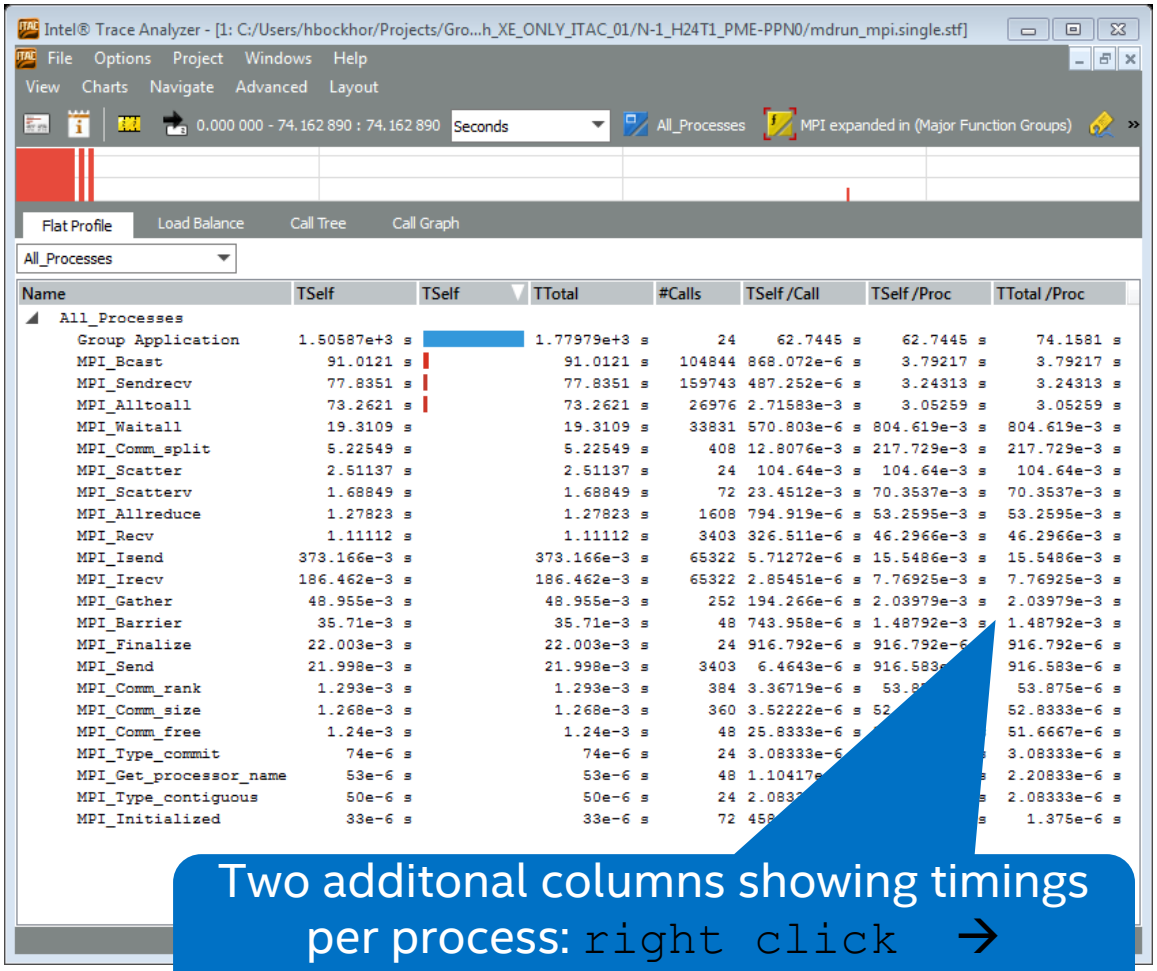


1x9 1D process grid

Speedup for 2D and 1D process grids



Measuring MPI times with ITAC



This Chart shows up automatically after clicking `Continue` on the start screen:

`right click -> Ungroup MPI`

shows all MPI functions and the Application time == non MPI run time.

Times are accumulated over all ranks

Measuring MPI times with ITAC

Plain ITAC provides accumulated timings for all MPI routines T_mpi,acc and the computation T_comp,acc (named: Group Application). For the analysis we need average times:

T_comp[p] = T_comp,acc[p]/p

T_mpi[p] = T_mpi,acc[p]/p

The averages can be directly shown by ITAC using the `Function Profile Settings` and checking `TSelf/process`

In the following all accumulated timings get the “acc” suffix. All other timing are averages or single process timings!

Measuring MPI times with ITAC

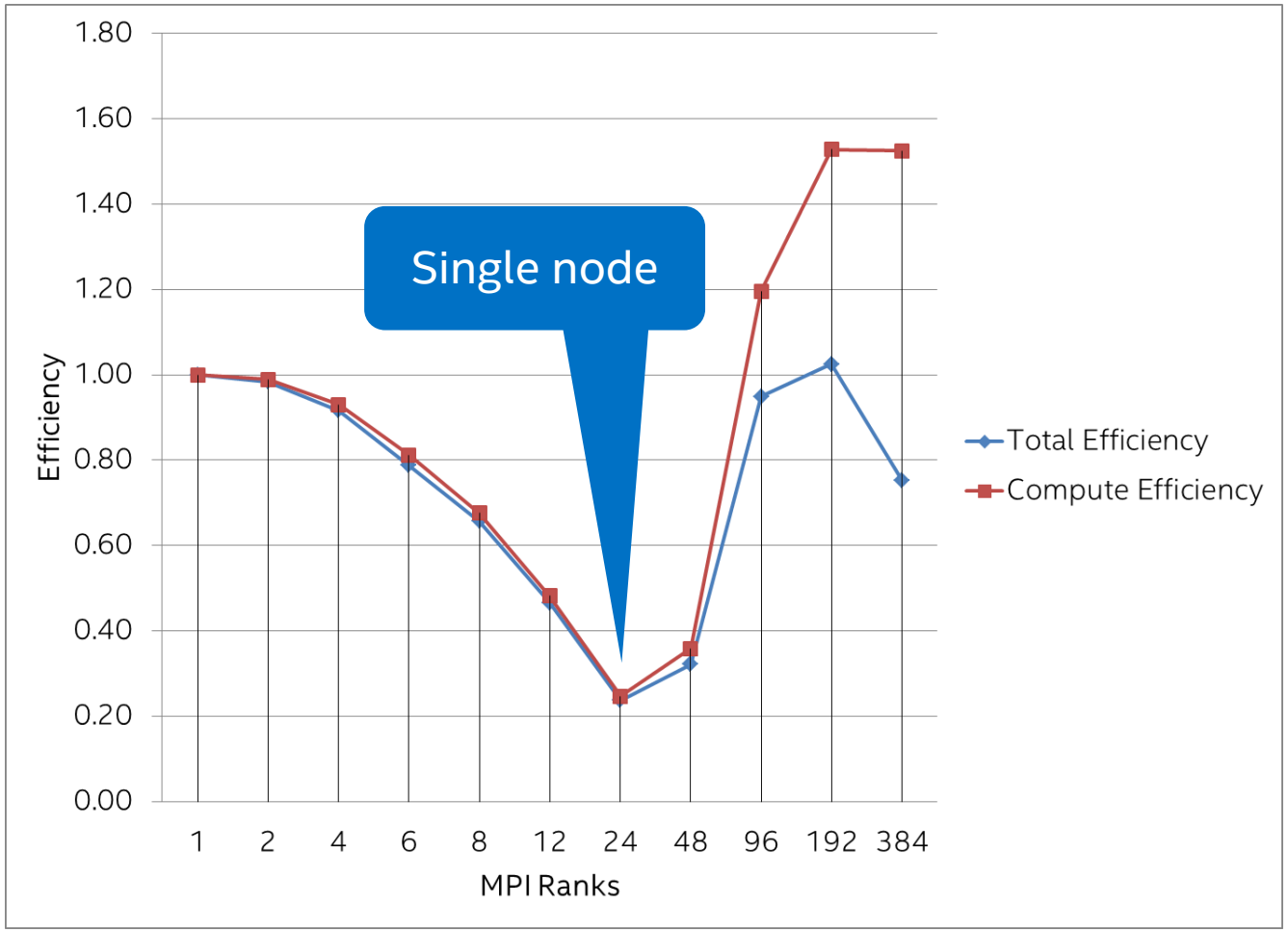
Using conventions from last slide we define the first step of splitting the wall clock run time T[p]:

$T[p] = T\_comp[p] + T\_mpi[p]$

Speedup and Efficiency can now be calculated for the compute time separately:

$S\_comp[p] = T\_comp[1]/T\_comp[p] = T[1]/T\_comp[p]$

Compute Efficiency vs. Total Efficiency Poisson example



Agenda

Performing a scaling analysis supported by ITAC

Simulation of run time using an ideal network

Splitting run time into components (compute, wait,...)

Analysis of message passing structure

Detailed Visualization of MPI programs

Analysis of program structure (non MPI) with Intel® VTune™ Amplifier XE

Summary

Algorithm and Network evaluation

ITAC shows timing of all MPI routines used by a program

The timing of MPI routines may be due to network transfer times caused by interconnect bandwidth limitations

The other possibility are waiting times caused by the algorithm: load imbalance or dependencies

A simple Network Model

The most simple network model defines :

Latency L	=	transfer time for 0 byte message
Bandwidth BW	=	transfer rate for (asymptotically) large messages
Message Volume V	=	data amount sent

The transfer time is:

$$T\_trans[V] = L + (1/BW)*V$$

ITAC: Ideal Network Simulator

It is extremely complicated to simulate a realistic network!

An extreme case – the ideal network – may be simulated by setting all transfer times to 0. This would mean L = 0 and BW = ∞ for the simple model

ITAC offers an ideal network simulation with transfer times set to zero. Compute times (non MPI) will stay the same

An existing real trace file is used as basis for the simulation

ITAC: Ideal Network Simulator

With a perfectly balanced algorithm the total MPI time will be vanishing in the ideal case

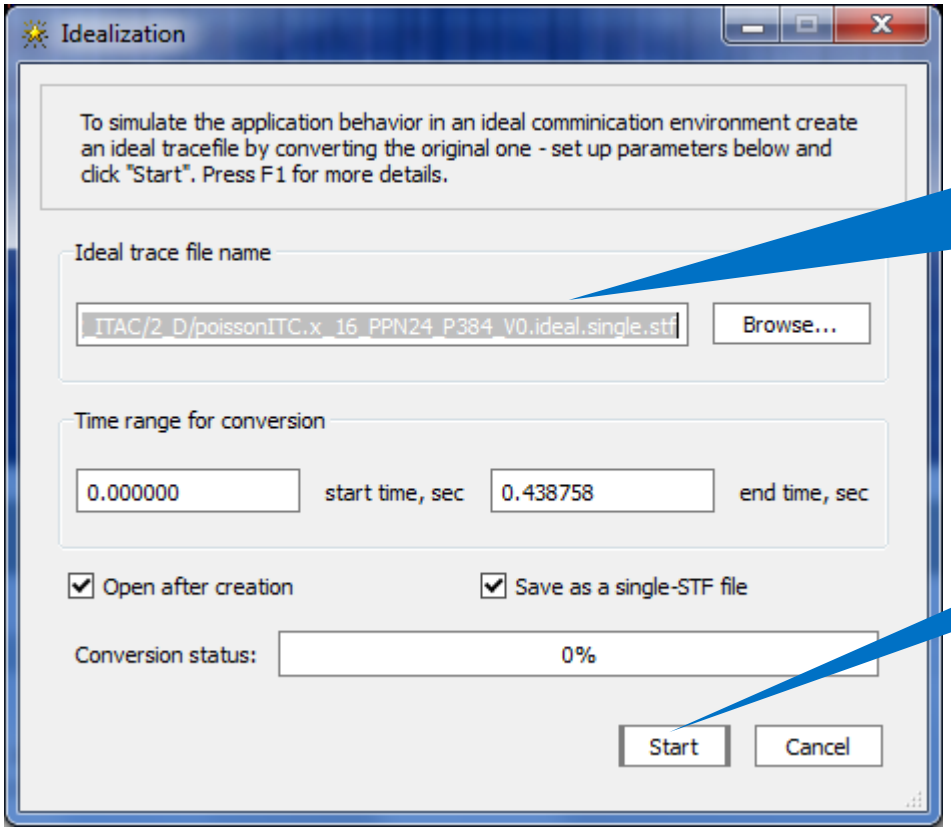
In most real cases the MPI time will just shrink but not vanish

The remaining part is due to waiting time e.g. when the receiver is starting to receive before the sender is ready to send

Start simulator with: `Advanced→Idealization`

Simulation details

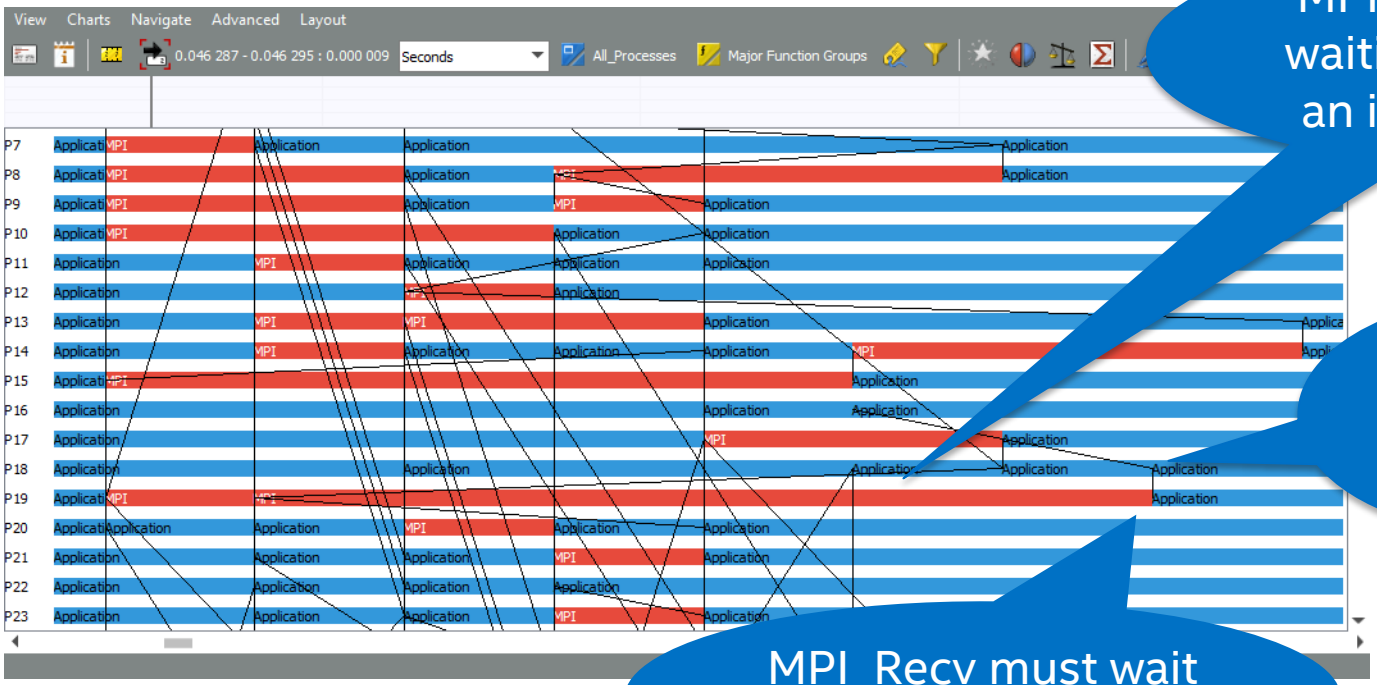
Test cases for simulation are the 16 nodes configurations: 24x16,1x384, 384x1



Name for idealized trace file gets additional “ideal”

Press start to continue

Waiting time due to dependencies



MPI _Recv is pure waiting time inside an ideal trace file

MPI_Send time shrinks to 0

MPI_Recv must wait on MPI_Send call

Agenda

Performing a scaling analysis supported by ITAC

Simulation of run time using an ideal network

Splitting run time into components (compute, wait,…)

Analysis of message passing structure

Detailed Visualization of MPI programs

Analysis of program structure (non MPI) with Intel® VTune™ Amplifier XE

Summary

MPI time

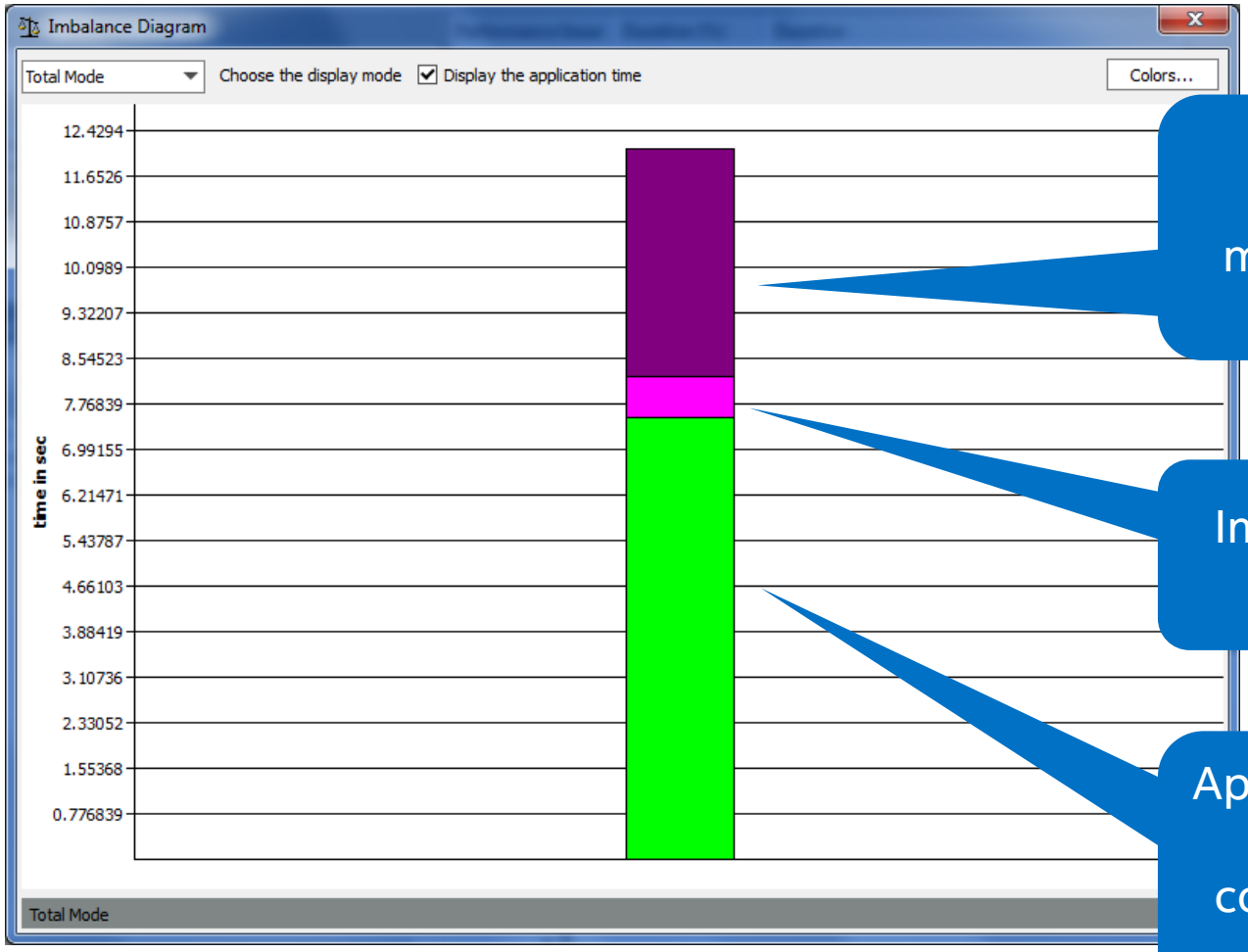
The simulated MPI time for the ideal network may be regarded as the waiting time `T_wait` due to imbalance and dependencies:

```
T_mpi = T_transfer + T_wait
```

After generation of an ideal trace file the result can be displayed in the Imbalance Diagram:

`Advanced → Application Imbalance Diagram`

Imbalance diagram – 16 Nodes 24x16



Interconnect time (T_transfer,acc) move mouse over bar: 3.89 [sec]

Imbalance (T_wait,acc): 0.714621 [sec]

Application(T_comp,acc): 7.5545 [sec] contains some artificial startup time

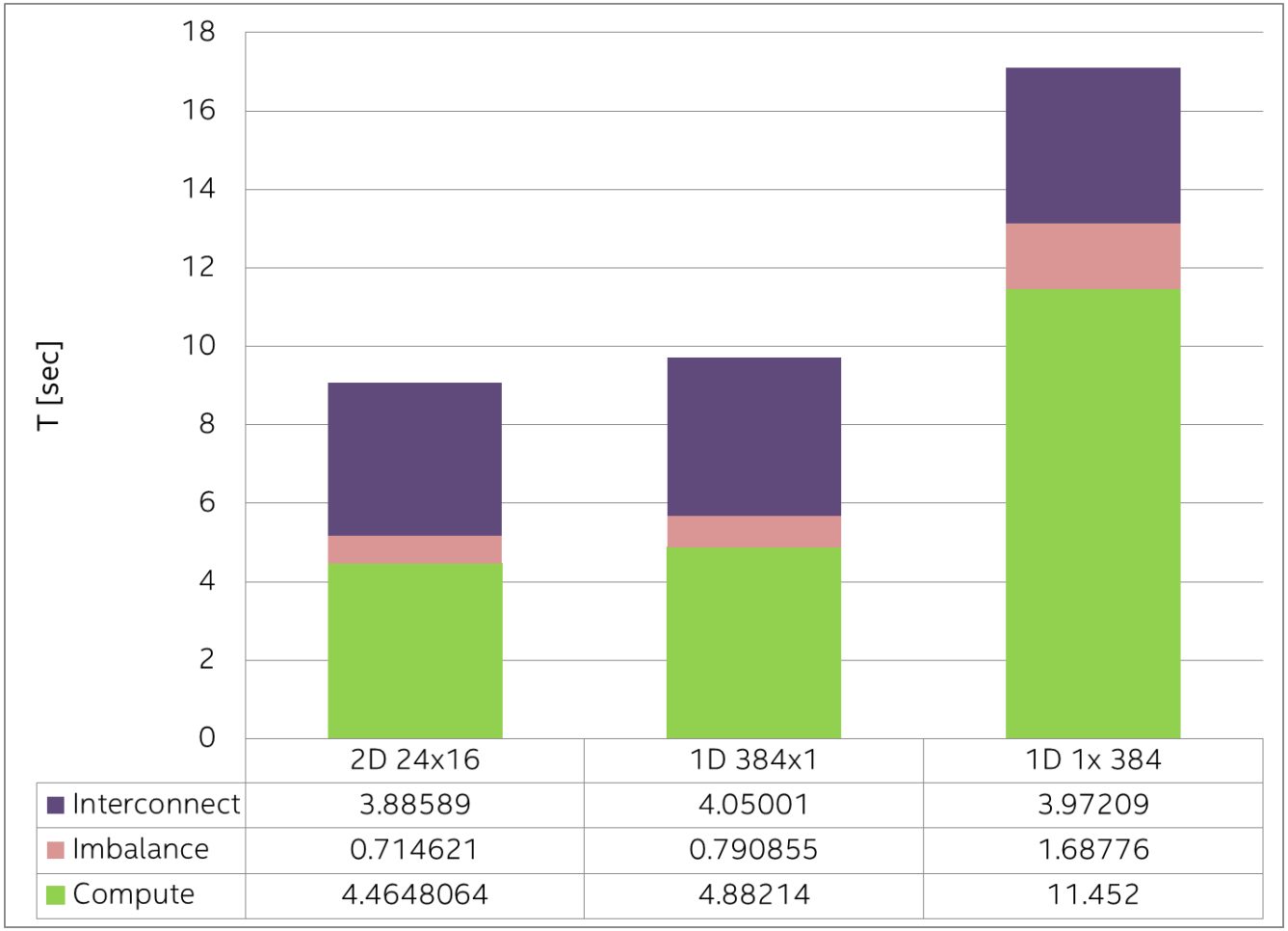
Imbalance Diagram = Tuning Start Point

The imbalance diagram displays the relation of transfer to wait time. Due to the result we can decide how to proceed with tuning:

- Transfer time (Interconnect) dominates: the algorithm is balanced but we have to improve the network performance by e.g. different process placement or new network hardware

- Waiting time (Imbalance) dominates: the algorithm has to be revisited e.g. better load balancing. New network hardware or better process placement will not help!

Imbalance diagram – 16 Compute Nodes



	2D 24x16	1D 384x1	1D 1x 384
Interconnect	3.88589	4.05001	3.97209
Imbalance	0.714621	0.790855	1.68776
Compute	4.4648064	4.88214	11.452

Imbalance diagram – 16 Compute Nodes

Compute time almost equal for 2D 24x16 and 1D 384x1 process grid. Row vectors are long enough. 3600/16 = 225 for 24x16 process grid and 3600 grid points for 384x1 process grid

Compute time for 1x384 is almost 3X longer probably because of short vector length 3600/384 < 10

Imbalance time best for 2D because process grid fits perfectly: local grid = (3600/24) x (3600/16) = 150 x 225 grid points

Imbalance time for 384x1 slightly worse because number of local grid point rows will vary between 10 and 9 (3600/384 = 9.375). See next slide(s) for a discussion about the measurement of imperfect data distribution

Imbalance time for 1x384 is even larger because of longer compute time. The imbalance stretches with compute time

Global Load Imbalance

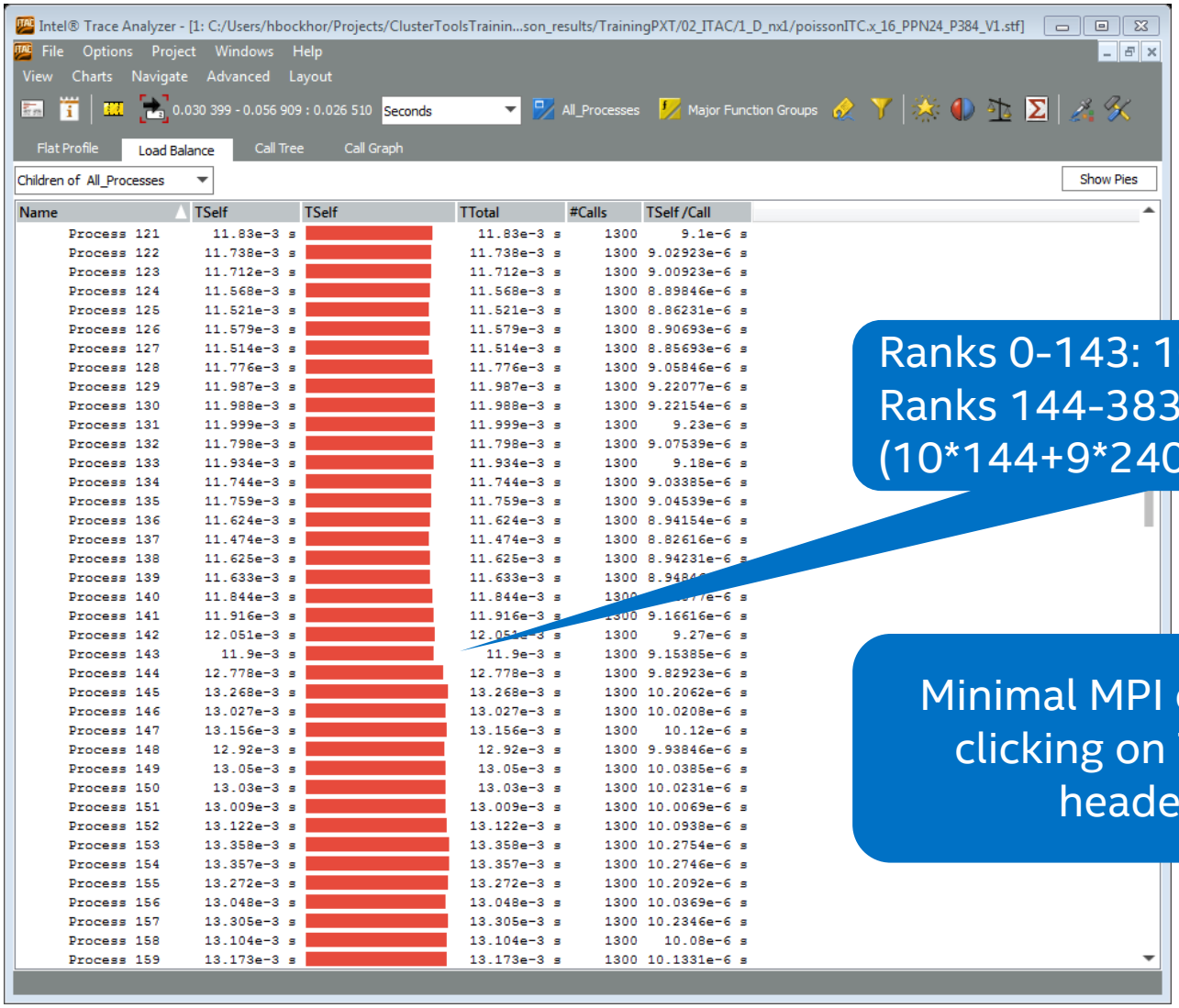
A portion of the waiting time is normally due to Global Load Imbalance.

The Global Load Imbalance is measured by determining the maximum per rank and average compute time over all processes:

```
T_load   = T_compute_max  - T_compute
         = T_mpi          - T_mpi_min
```

`T_load` is the time we may win by achieving a perfect load balance. It should be lower than the previously calculated MPI time for an ideal network (= T_wait = Imbalance/p)

Load Imbalance: MPI for 1D 384x1



Ranks 0-143: 10x3600 points
Ranks 144-383: 9x3600 points
(10*144+9*240) = 3600

Minimal MPI can be found by clicking on TSelf (Column header) ➔ sort

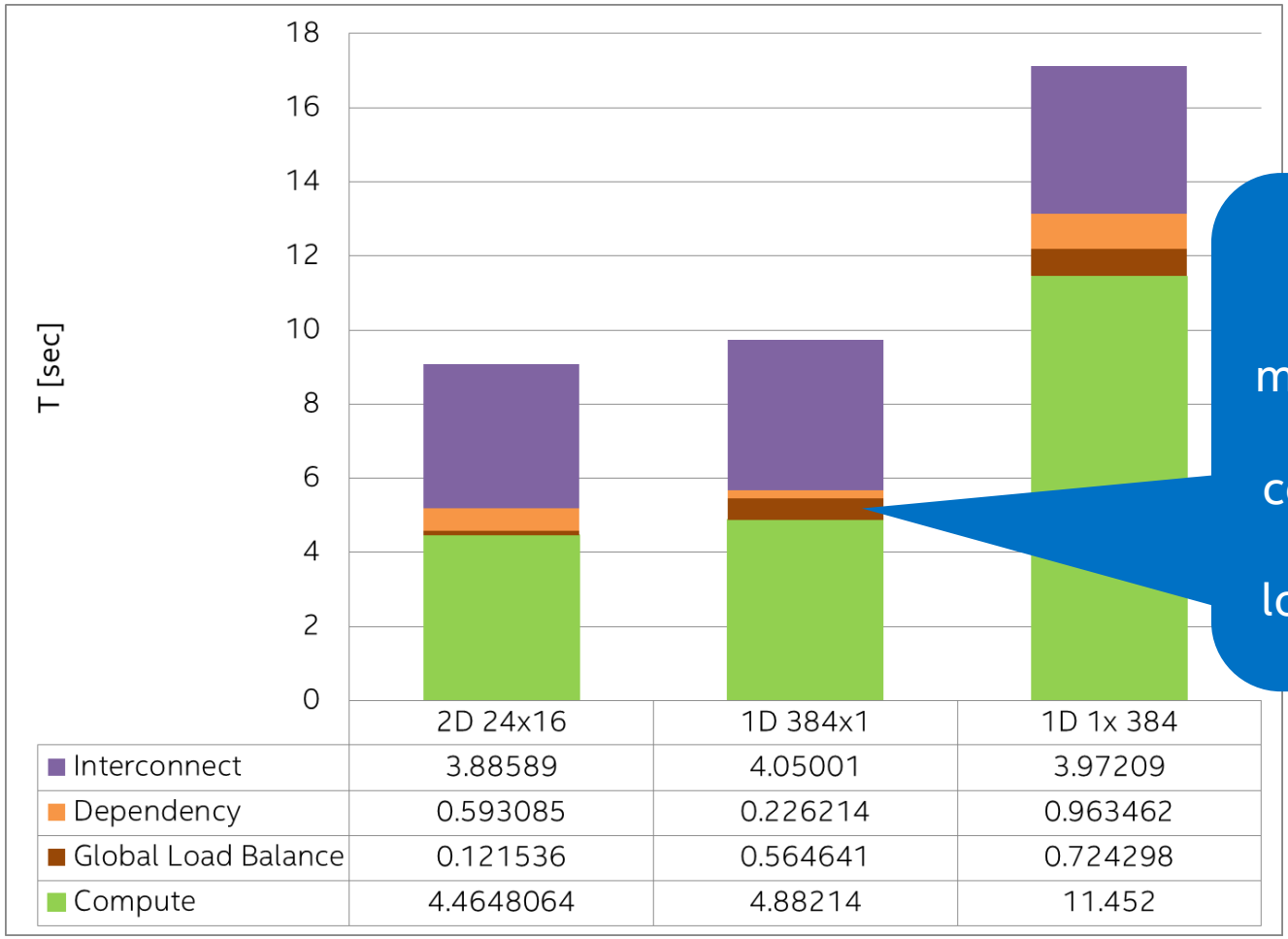
Split of timings

We have now all components of our split of timings:

```
T  = T_compute   + T_mpi
   = T_compute   + T_trans  +  T_wait
   = T_compute   + T_trans  +  T_load   + T_depend
```

The Imbalance diagram shows only the second line but we might additionally compute `T_load` and `T_depend` for a deeper analysis. `T_depend` is called Dependency time. This is just the rest of the imbalance time `T_wait` that is not due to the Global Load Imbalance.

Refined Imbalance Diagram



	2D 24x16	1D 384x1	1D 1x 384
Interconnect	3.88589	4.05001	3.97209
Dependency	0.593085	0.226214	0.963462
Global Load Balance	0.121536	0.564641	0.724298
Compute	4.4648064	4.88214	11.452

Global Load Imbalance is more severe for 1D. This is consistent with the different local grid sizes!

Agenda

Performing a scaling analysis supported by ITAC

Simulation of run time using an ideal network

Splitting run time into components (compute, wait,...)

Analysis of message passing structure

Detailed Visualization of MPI programs

Analysis of program structure (non MPI) with Intel® VTune™ Amplifier XE

Summary

Message Passing Profile

Message passing profile displays various characteristics of message passing in a sender/receiver Matrix

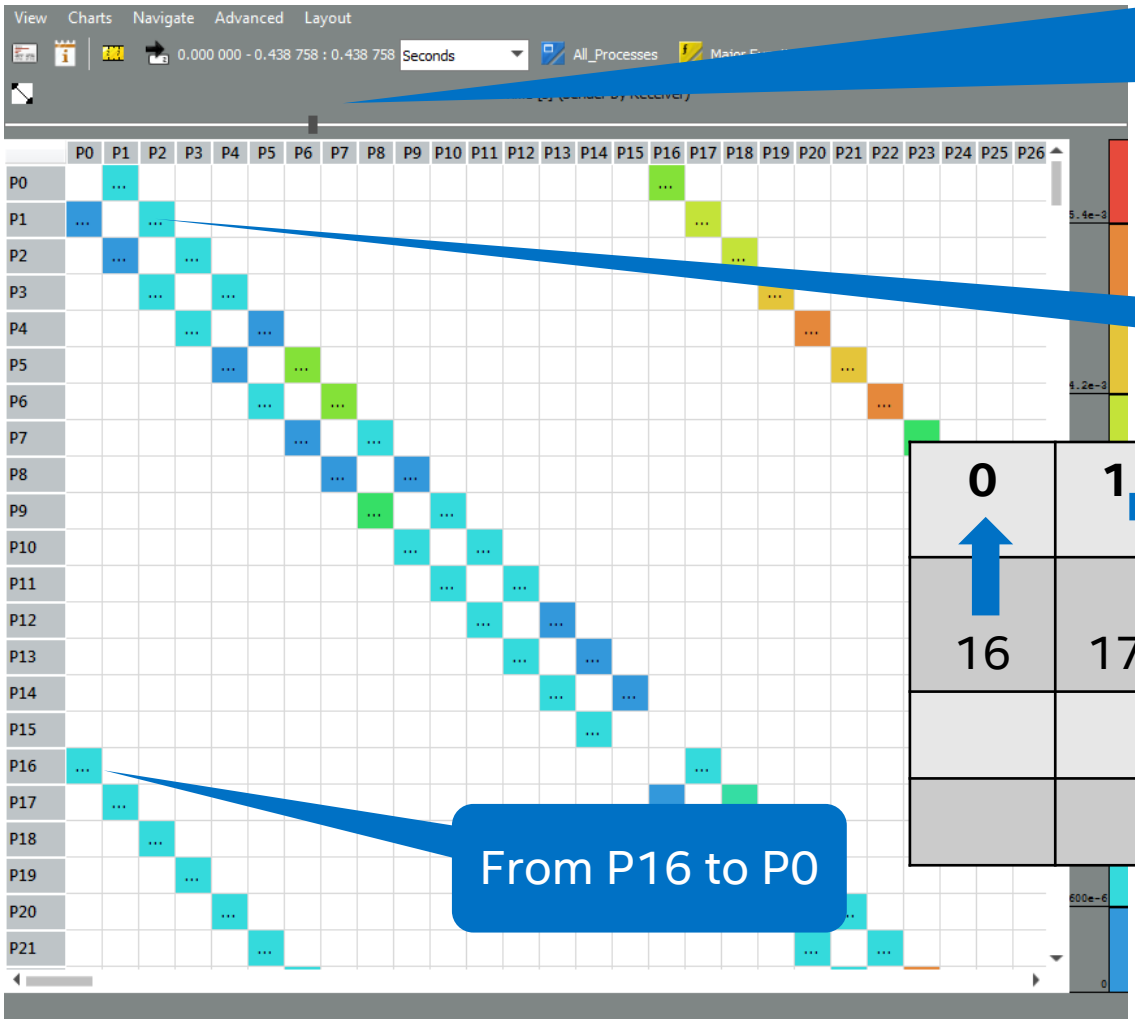
`Charts → Message Profile`

The Matrix element N,M corresponds to the message passing characteristics from rank N to rank M. Change these attributes by:

`Right Click → Attribute to show`

Characteristics are: total message volume, message passing time, max, min, average rate and count

Message Passing Profile: 24x16 grid



Use slider for changing the size of cells or: `Message Profile Settings → Automatic Cell Size`

Messages from P1 to P2

From P16 to P0

0	1	2	...	14	15
16	17	18	...	30	31
			...		

Message Passing Profile:16 Nodes

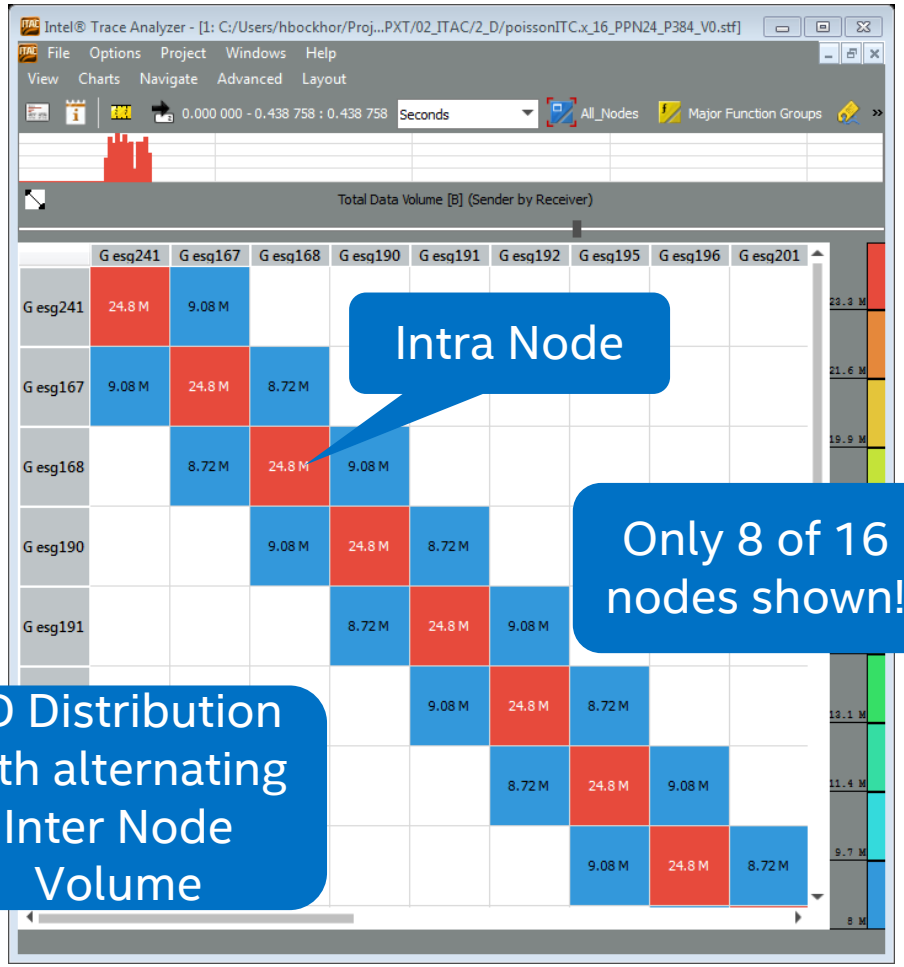
For 16 nodes (384 ranks on IVB) the total Message Passing profile is not very handy

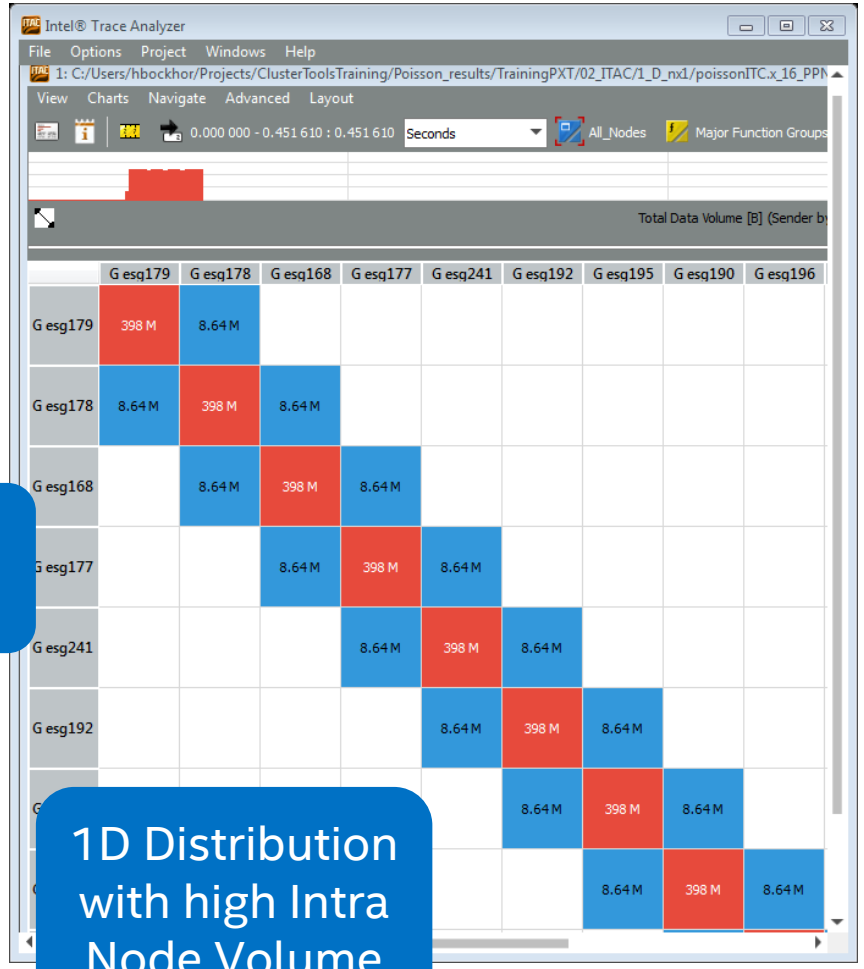
We may fuse the communication to compute node level. In this case 384 ranks are fused to 16 compute nodes:

`Advanced → Process Aggregation`

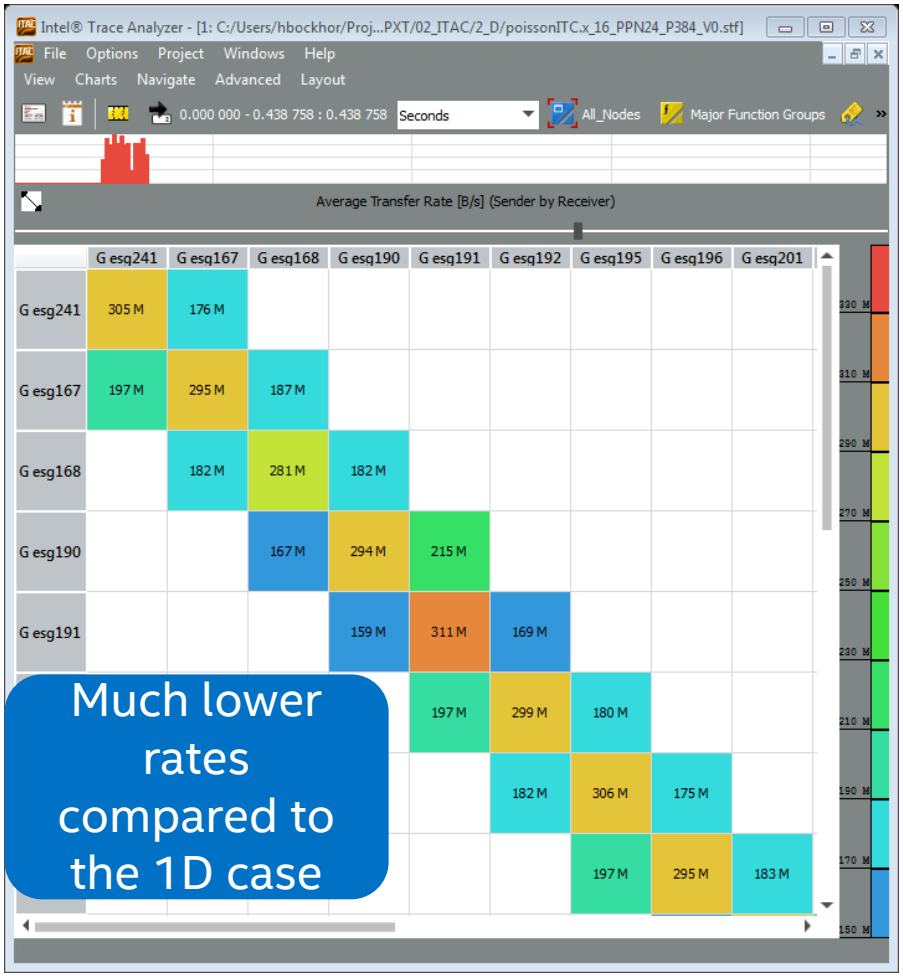
This will pop up a new window: check `All_Nodes` and apply

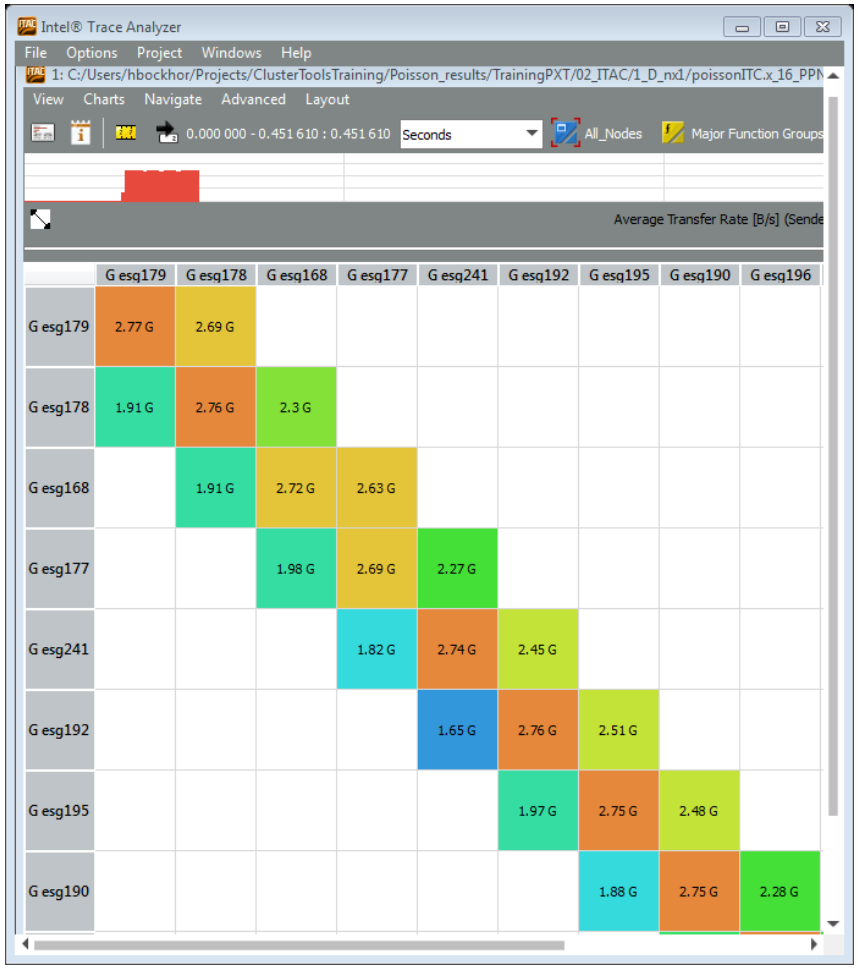
Total Volume: 2D vs. 1D distribution



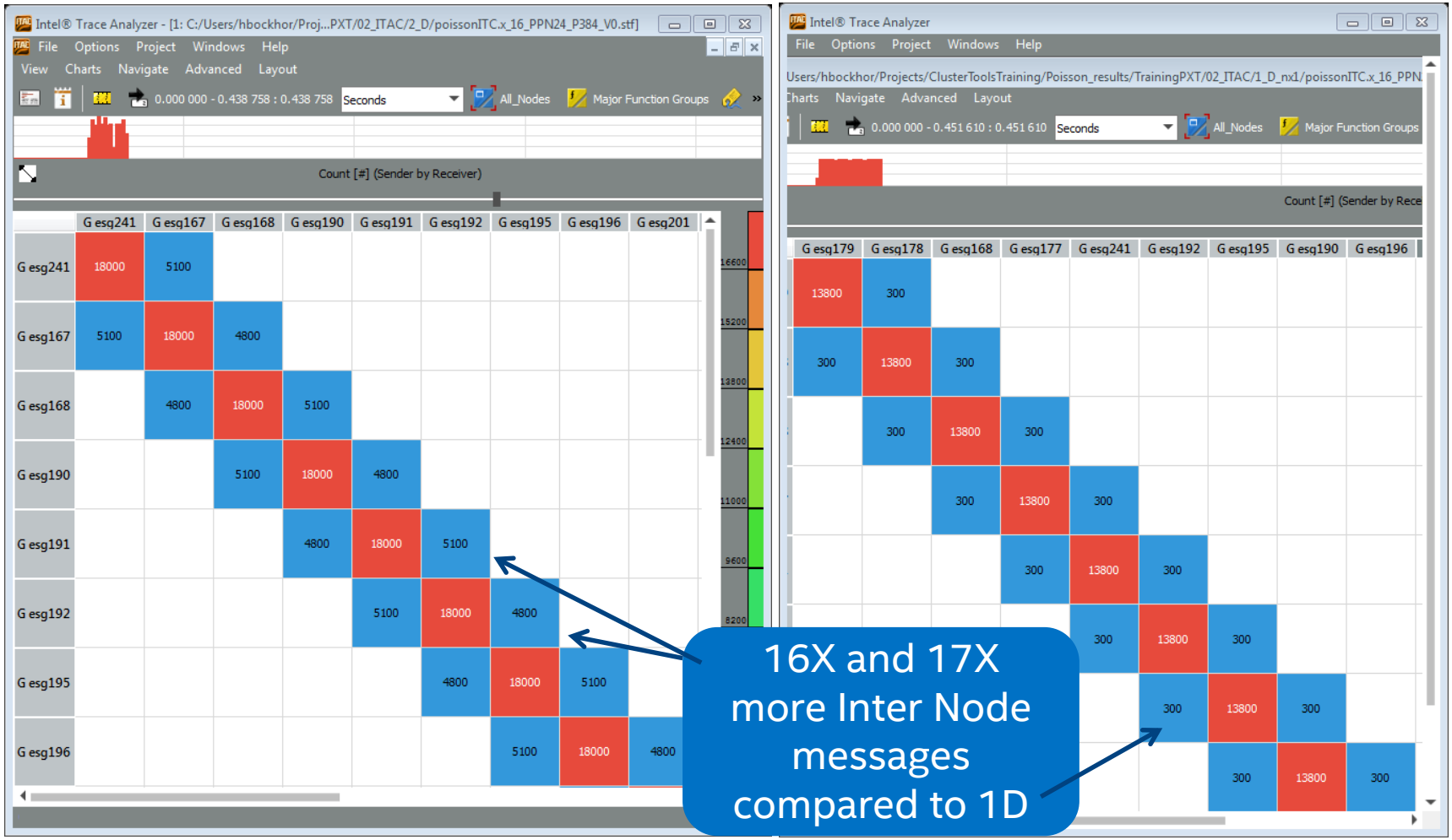


Average Rate: 2D vs. 1D distribution





Number of mesg.: 2D vs. 1D distribution



Message Profile: Observations

Inter node communication has about the same volume in the 2D case but 16x more messages are sent

There is just a single inter node message per boundary exchange in the 1D case (3 exchanges per iteration times 100 iterations == 300 messages)

Communication rate drops so much in the quadratic 2D case that the total transfer time (Imbalance diagram) is almost equal for both configurations

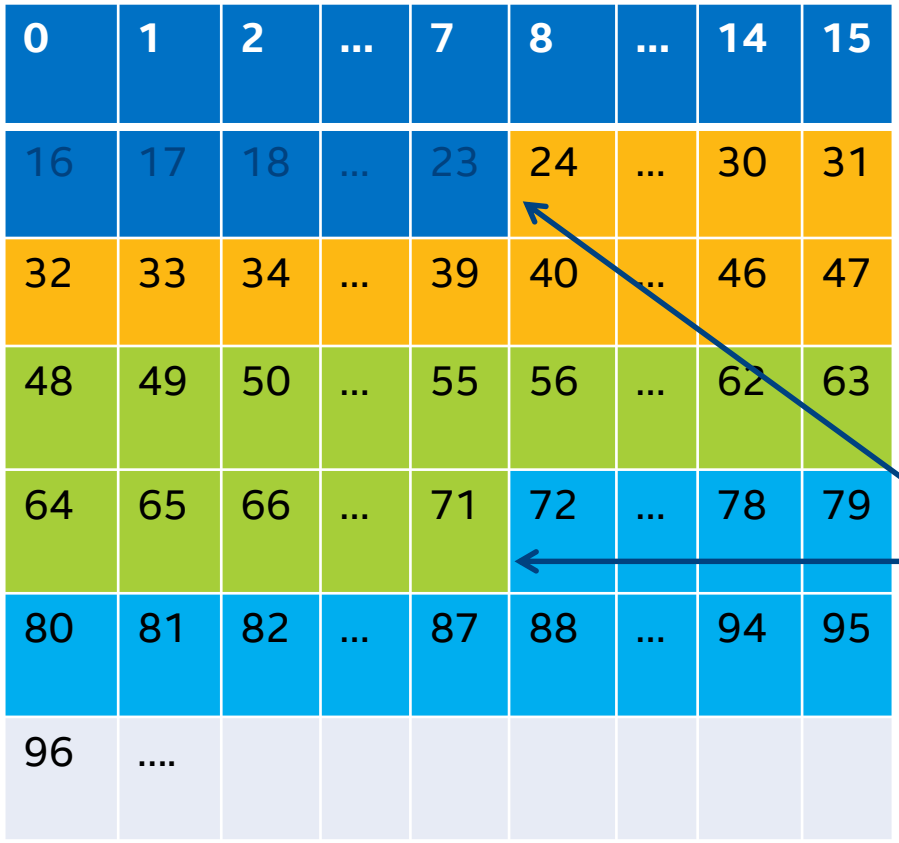
Optimization ideas

A compromise between quadratic and 1D processor grid may be more appropriate here like 48x8 or 96x4. This will reduce the number of inter node messages and raise the bandwidth for each message

The default rank to node mapping is just linear. This leads to alternating communication patterns (see following slides)

A better mapping can be achieved by putting all ranks of a rectangular sub process grid onto a single node. The following slides explain the ranks to node mapping

Default Mapping for 24x16 process grid



Node #0
Node #1
Node #2
Node #3

24 ranks per node! One rank per core

Additional horizontal exchange. 16 or 17 boundary lines between two nodes

Defining a 16x24 process grid may be better – why?

Optimized Mapping

0	…	3	4	…	7	8	…	11	12	…	15
16	…	19	20	…	23	24	…	27	28	…	31
32	…	35	36	…	39	40	…	43	44		47
48	…	51	52	…	55	56	…	59	60		48
64	…	67	68	…	71	72	…	75	76		79
80	…	83	84	…	87	88	…	91	92		95
96	….	99	100	…							

Node #0

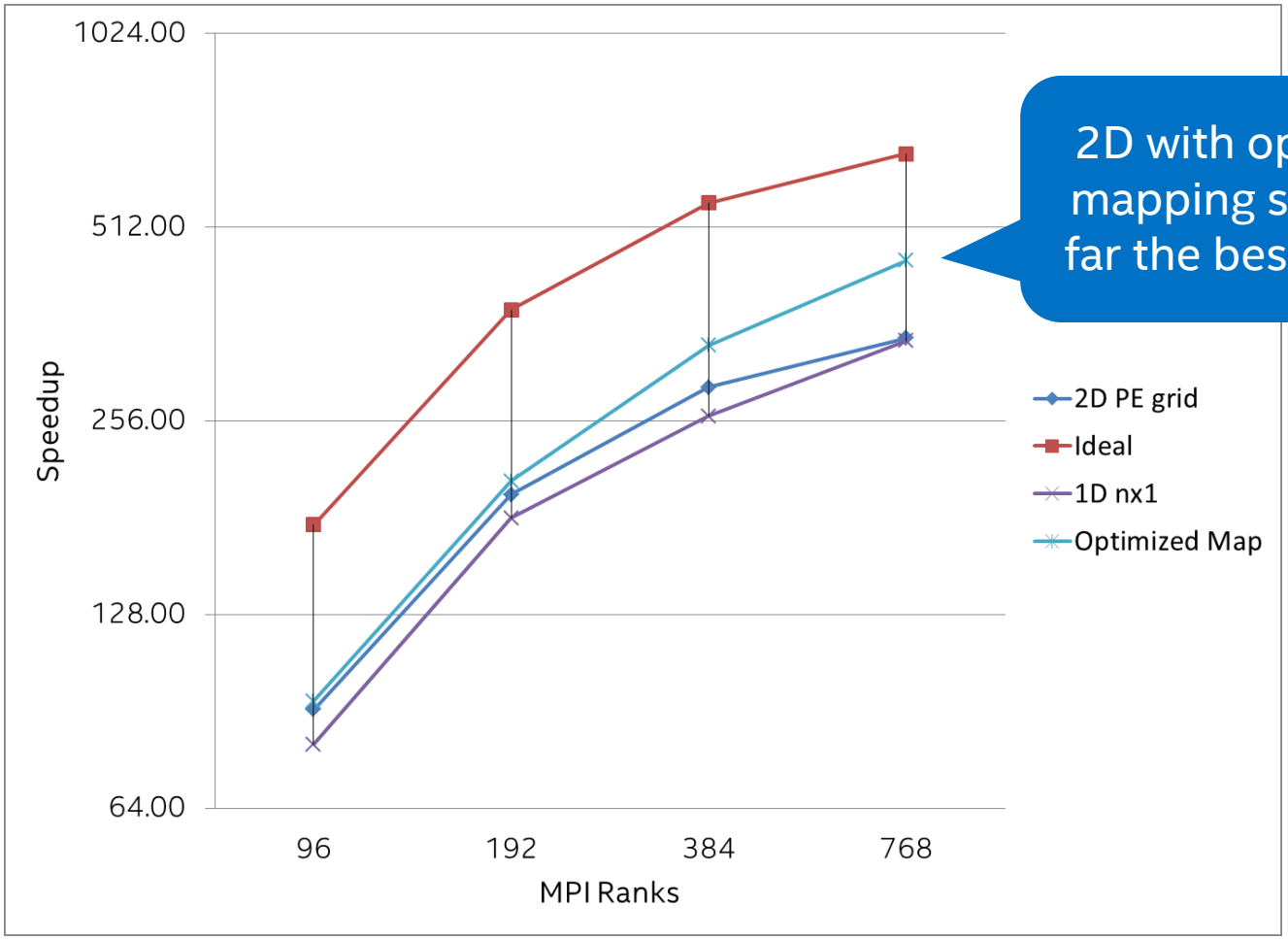
Node #1

Node #2

Node #3

This 6x4 pattern can be repeated for all nodes. The number of processor boundary lines between nodes are: 4 (vertical) and 6 (horizontal)

Impact of Optimized Mapping



Agenda

Performing a scaling analysis supported by ITAC

Simulation of run time using an ideal network

Splitting run time into components (compute, wait,...)

Analysis of message passing structure

Detailed Visualization of MPI programs

Analysis of program structure (non MPI) with Intel® VTune™ Amplifier XE

Summary

Detailed Visualization of MPI programs

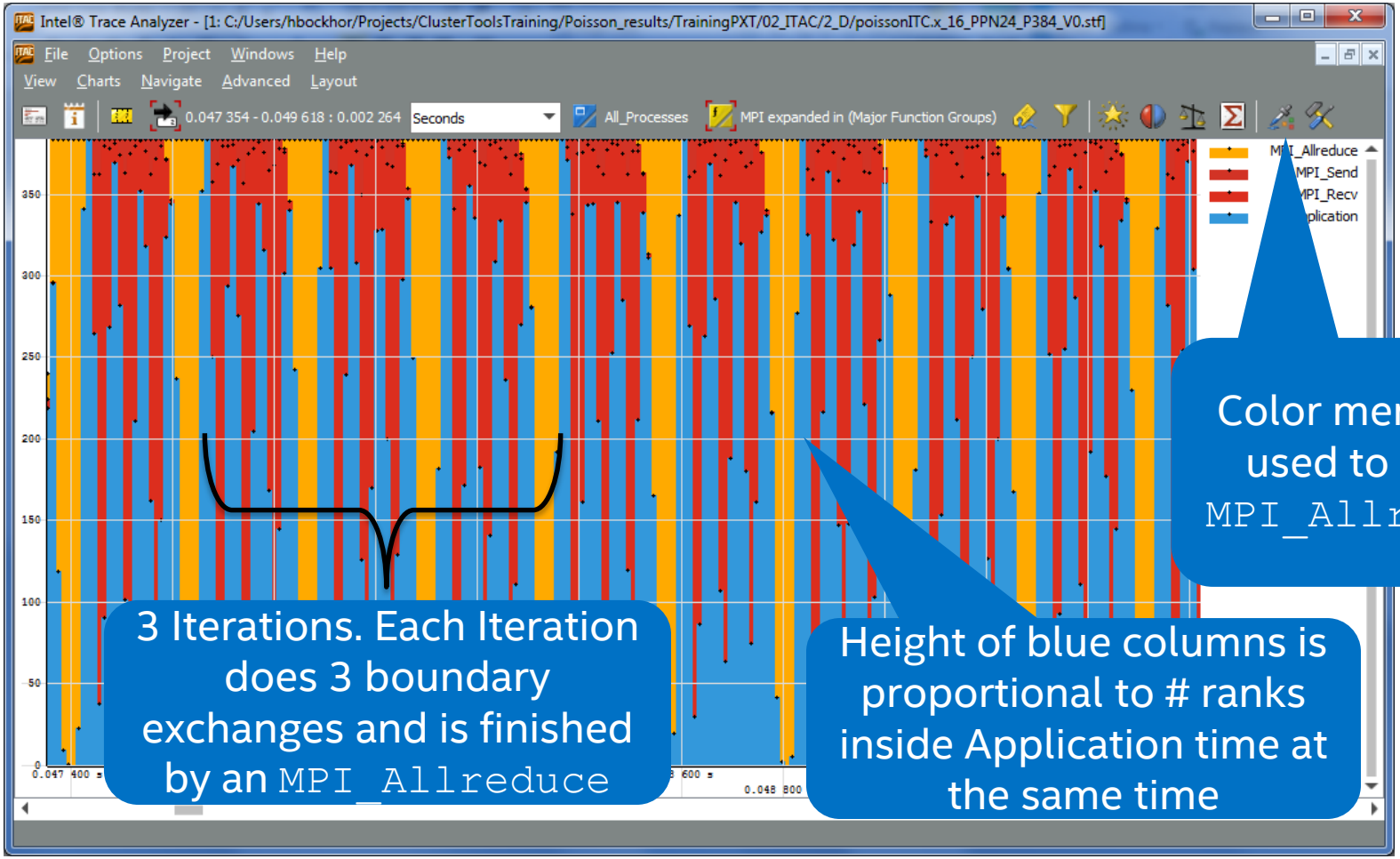
After some global evaluations we may dive now into the MPI algorithm by showing the temporal evolution with ITAC

Most programs consist of recurring patterns like iterations or different phases: initialization, computation and I/O

Quantitative timeline shows nicely coarse patterns:

`Charts → Quantitative Timeline`

Quantitative Timeline for 16 nodes



Single iteration – Event Timeline

After identification of basic patterns we may now change to the more detailed Event Timeline

Event timeline is the most important Chart in ITAC

Temporal development reveals root causes of dependencies due to suboptimal implementations

`Charts → Event Timeline`

Single Iteration Poisson

Same configuration as used before in the mapping section: 24x16

This is the default mapping!

Boundary Exchange in Ideal Network

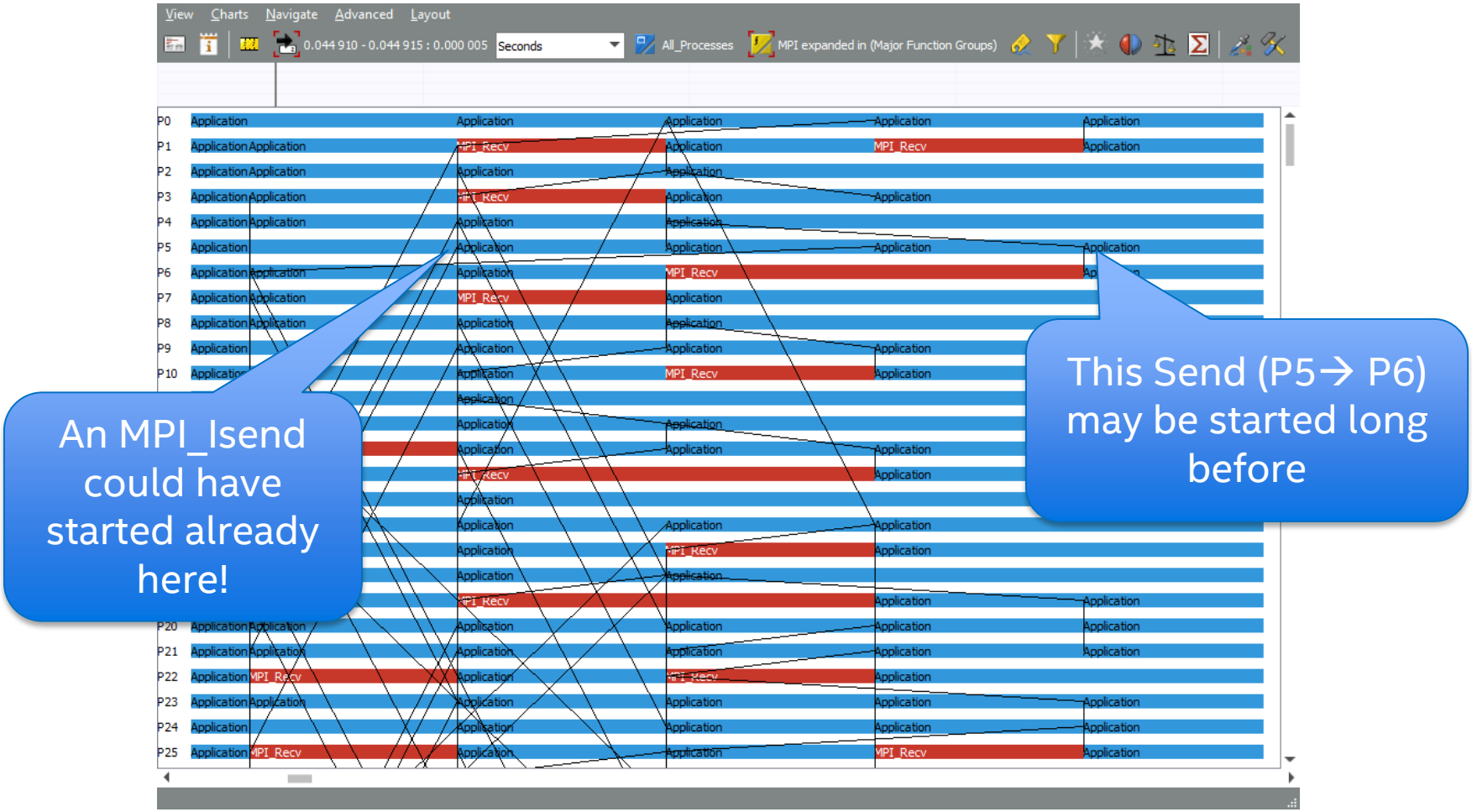
MPI times in the ideal network case are due to global load imbalances and dependencies

Dependencies are e.g. due to order of blocking sends and receives

The current naive implementation of the boundary exchange uses blocking sends and receives: `MPI_Send, MPI_Recv`

The Ideal network simulation helps to clearly identify dependencies

Boundary Exchange in Ideal Network



Optimization idea

Some of the dependencies may be resolved by using `MPI_Isend` and `MPI_Irecv` with an `MPI_Waitall()` in the end

In a first step we may just exchange the blocking Sends/Recvs by the immediate routines and place a `MPI_Waitall()` at the end. Data copies of boundary arrays have to be done after the wait routine

In a second step we may optimize the order of MPI routines and data copies. Some requests may be ended by a separate `MPI_Wait()`

Comparing ITAC traces

Compare before and after optimization e.g. compare boundary exchange with blocking Send/Receive to non blocking Send/Receive

Further potential comparison scenarios:

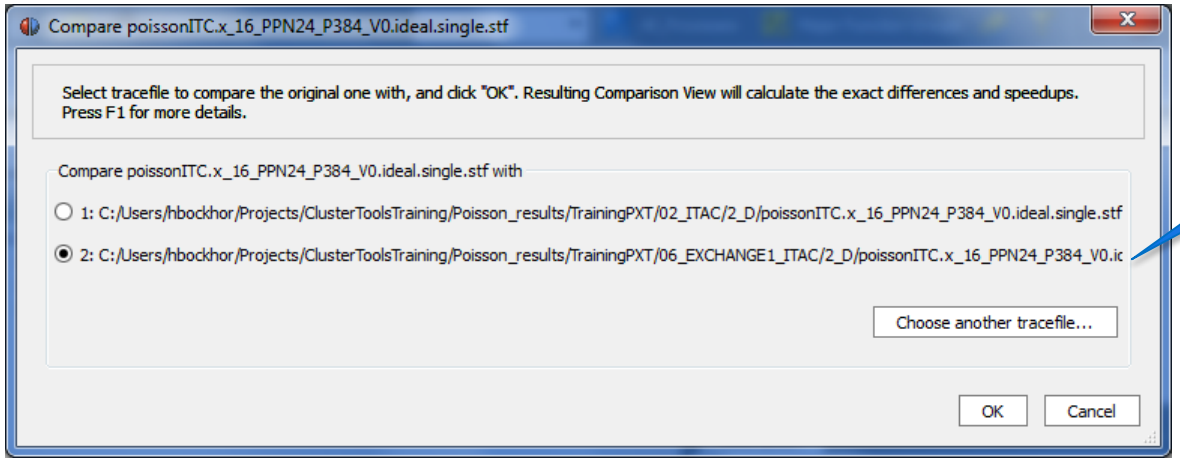
Compare ideal to real trace

Compare different number of ranks

Compare different mappings

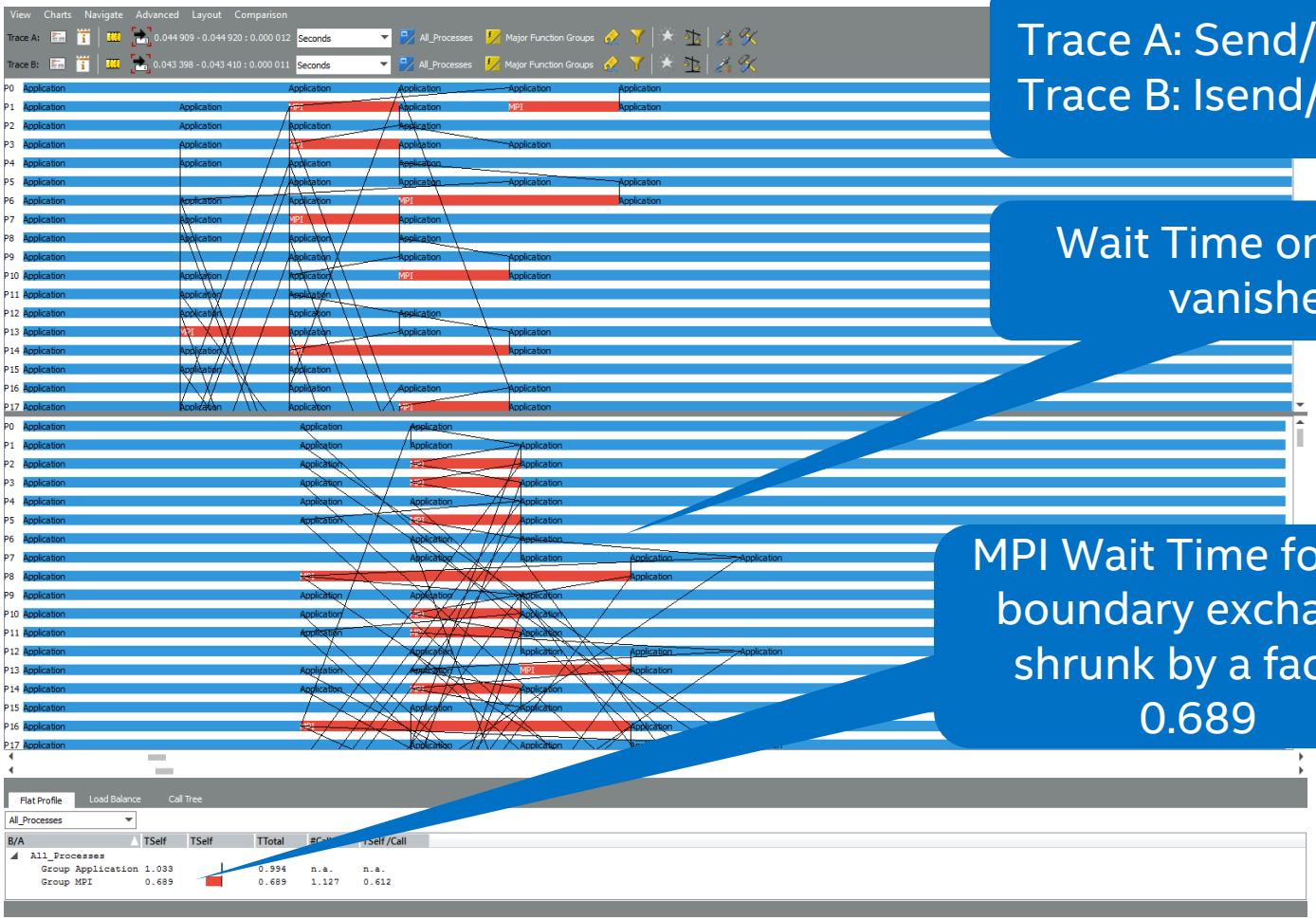
Comparing ITAC traces - HowTo

Open tab: `View` → `Compare`



Open another file for a comparison

Comparison: Boundary Exchange



Trace A: Send/Recv
Trace B: Isend/Irecv/Waitall

Wait Time on P6 has vanished!

MPI Wait Time for this boundary exchange shrunk by a factor 0.689

Instrumentation of User Functions

So far, we only see MPI routines and Application time inside ITAC traces

Navigation becomes far more easy when adding user functions

For evaluation of the impact of optimization we may want to see the timing of the boundary exchange including all its MPI calls

ITAC Compiler Instrumentation

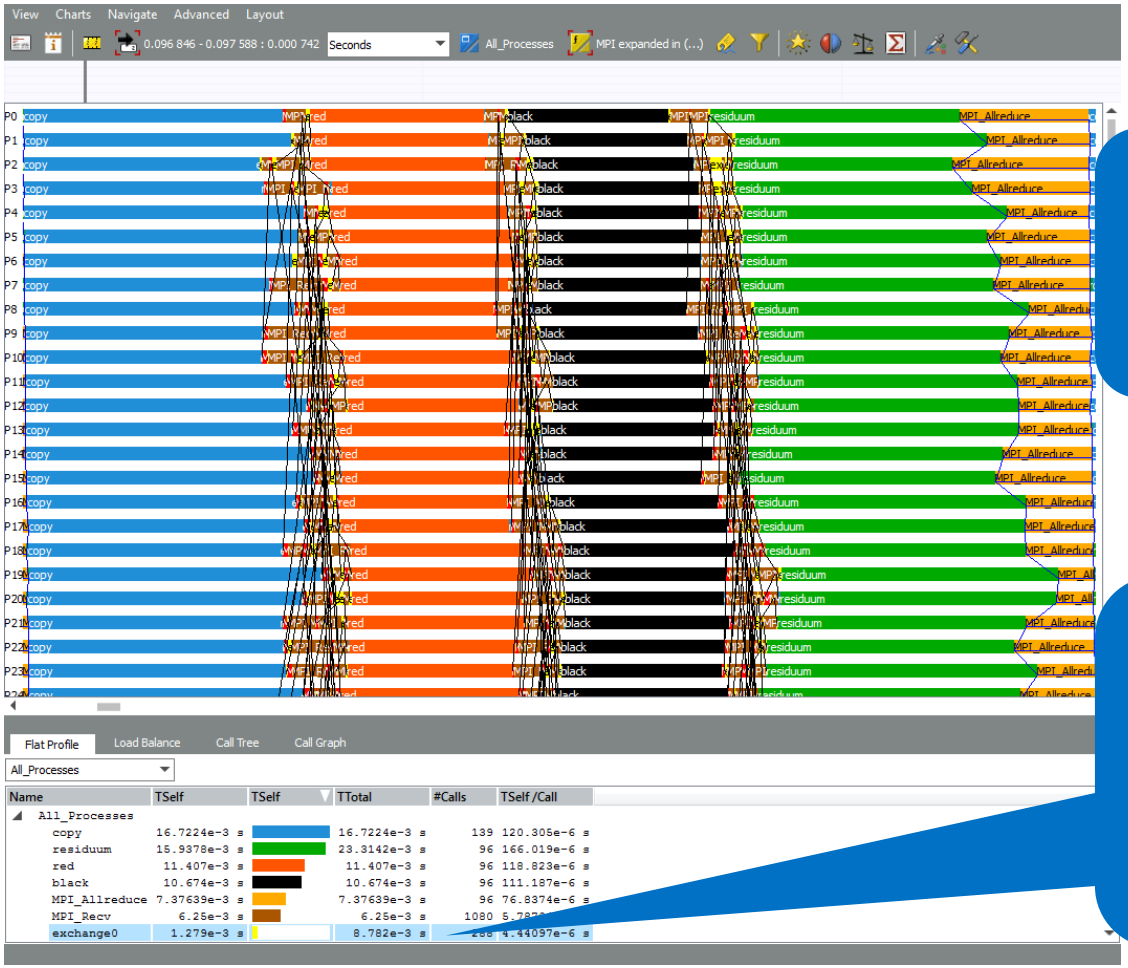
All source files or just the files of interest may be compiled with the

`-tcollect`

flag (Intel compiler only)
The executable has to be linked using this flag, as well

As an alternative (different compiler or code blocks that are not a function) we might consider to use the ITAC API functions for instrumentation. This is discussed in ITAC Advanced presentation

Iteration with User Functions



Instrumented user functions like exchange0 can improve analysis of the MPI algorithm

Exchange routine is on bottom of the list. But the total time TTotal also contains all MPI functions. This time exceeds the Allreduce time

Agenda

Performing a scaling analysis supported by ITAC

Simulation of run time using an ideal network

Splitting run time into components (compute, wait,...)

Analysis of message passing structure

Detailed Visualization of MPI programs

Analysis of program structure (non MPI) with Intel® VTune™ Amplifier XE

Summary

Intel® VTune™ Amplifier XE

We used ITAC for the analysis of the message passing algorithm

We already saw that computation performance saturated on a single node

With this tool we may have a closer look to the processor performance and program structure

VTune™ Amplifier XE based analysis can be started and performed by its GUI. Together with MPI on a Cluster which probably prefers batch usage, we will use the command line interface

Hot Spot Analysis

This is the most basic analysis type to start an investigation

The analysis will present hotspots of the calculation for a chosen MPI rank. Timings go down to source lines or assembly code

The Call Stack provides information about how the function is called and how much time is due to this branch

Vtune™ Amplifier XE on 16 nodes

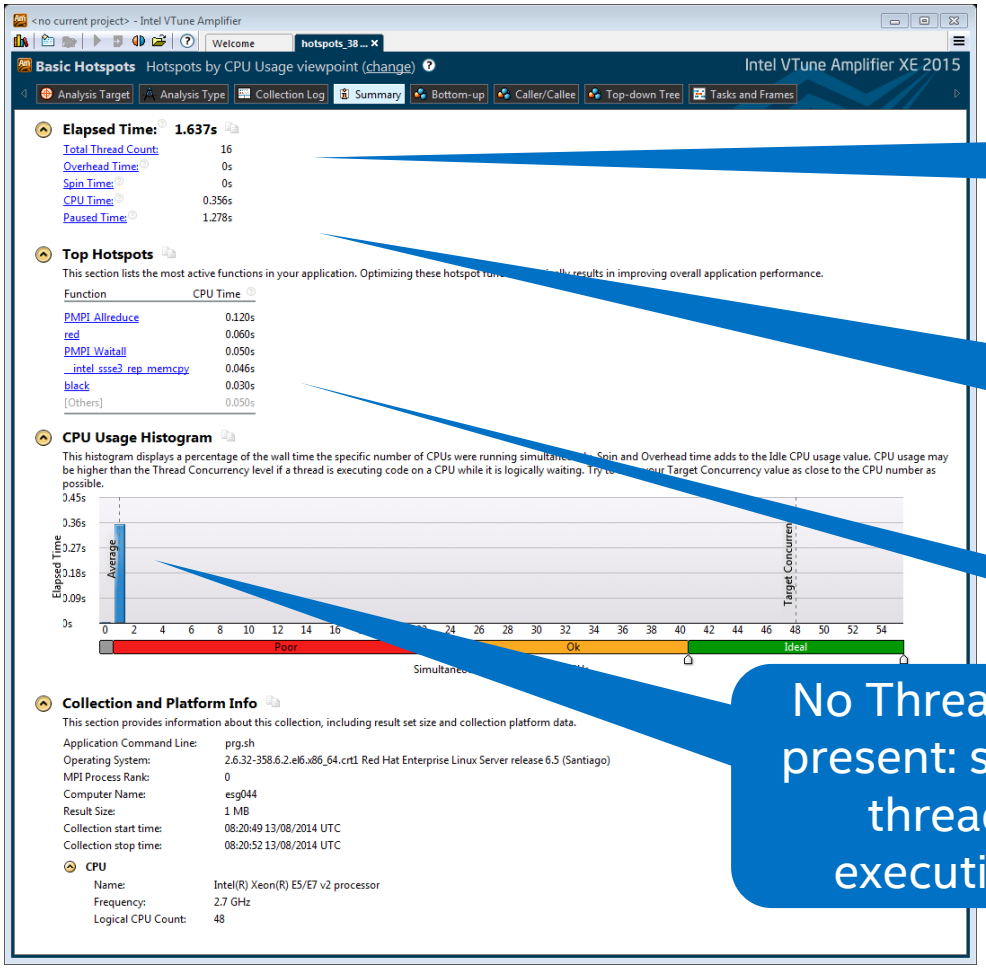
This analysis may be conducted for each of all 384 ranks but probably we may concentrate on a single rank first:

```
mpirun   -n  1 amplxe-cl --result-dir hotspots \
                         --collect hotspots -- poisson.x  :\
         -n 383 poisson.x
```

Hotspot analysis is performed on rank 0 and results are stored in directory hotspots.0. All other ranks run poisson.x without analysis

More complex selection of ranks are possible building groups of ranks doing analysis or not

Hotspots Analysis: Summary



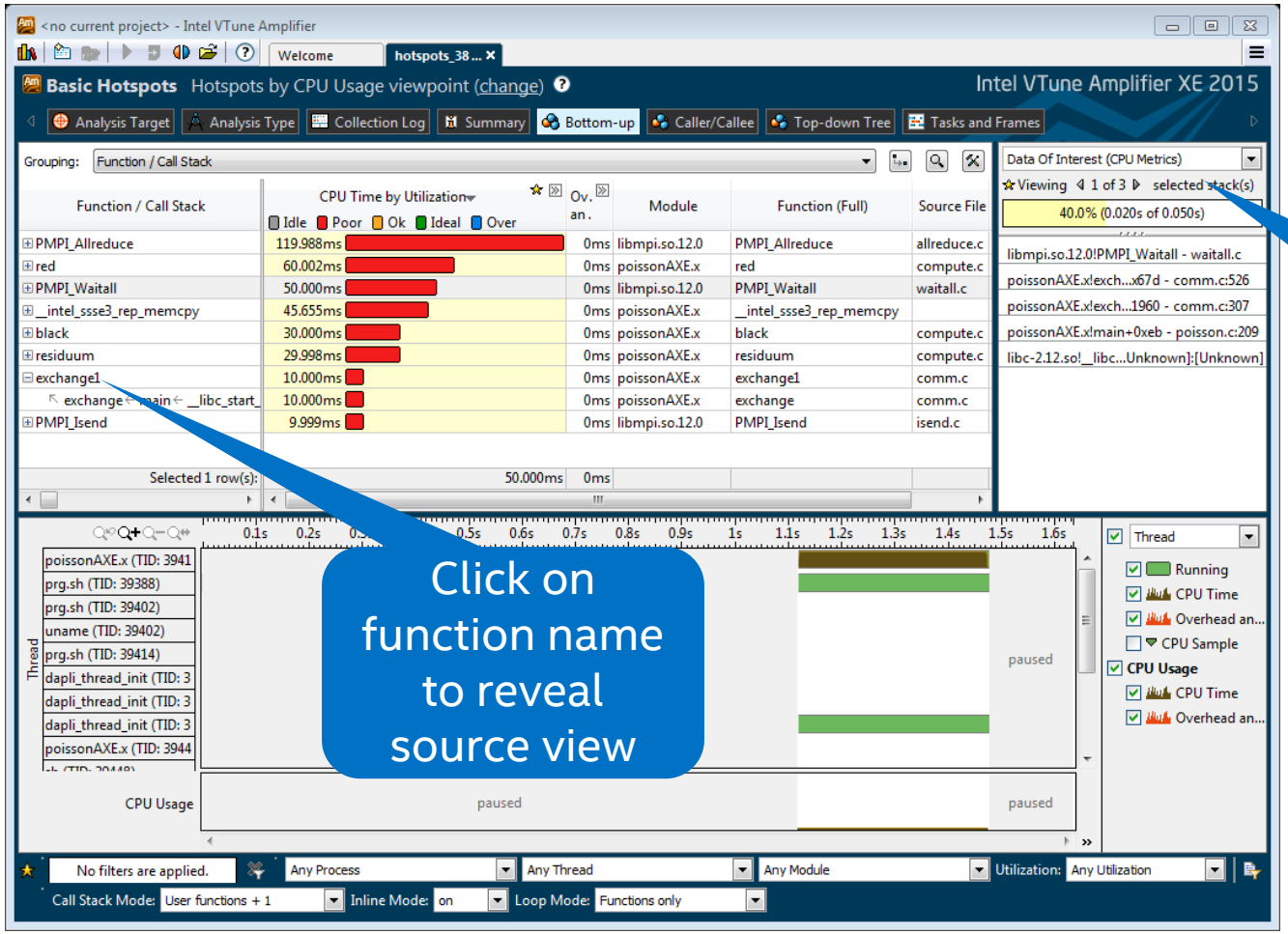
Ignore Thread count. It is just a single MPI process

Analysis only for Iterations. Rest of app. Is paused by using API functions

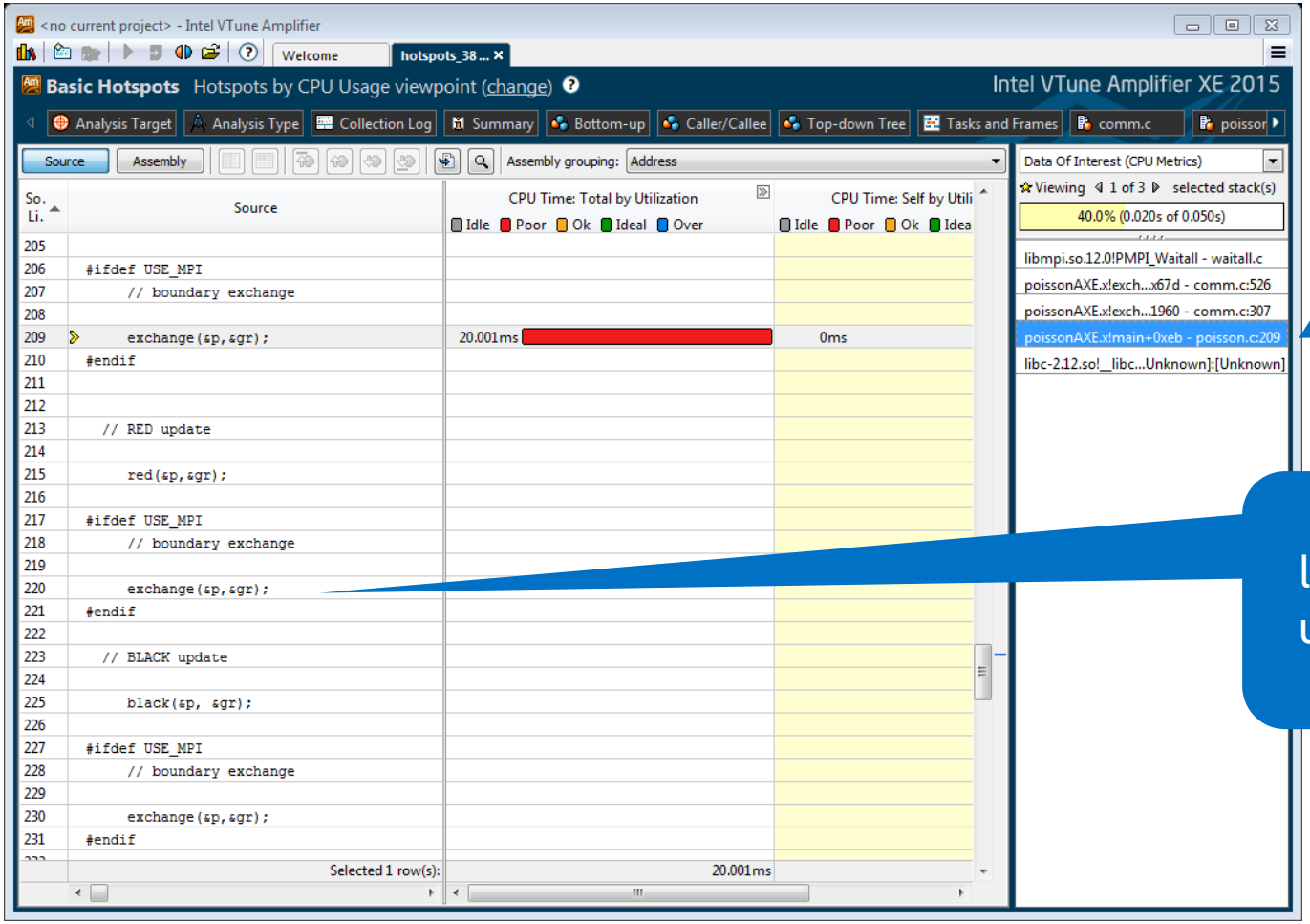
No Threading present: single thread execution

memcpy is called by copy routine

Hotspots: Bottom-Up



Hotspots Analysis: Source



Click on first stack and poisson.c line: shows first call to exchange at line 209

Second call at line 220 shows up by selecting another stack

Advanced Hotspots

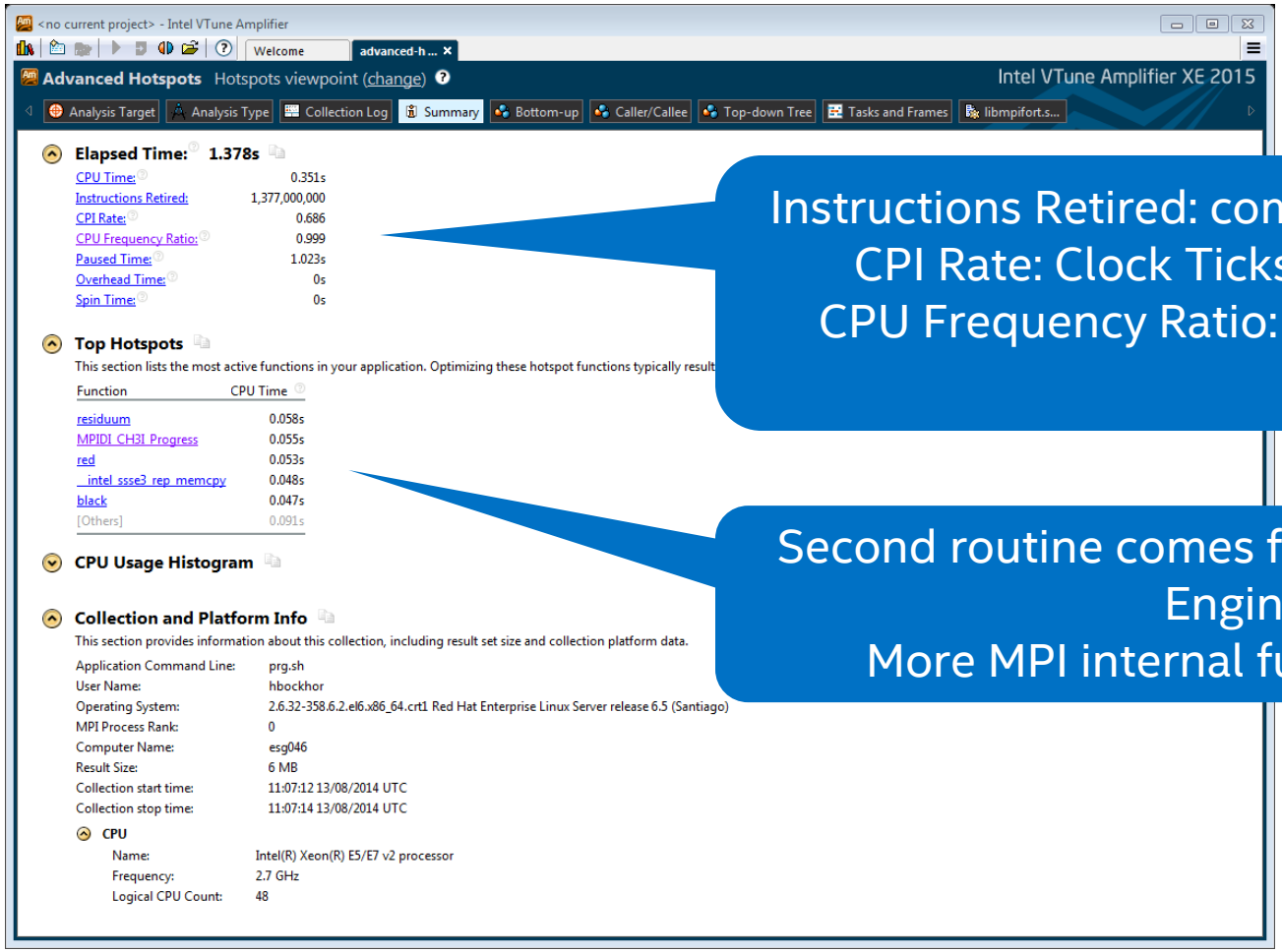
Hotspot identification using directly the Performance monitoring Unit (PMU) . Needs special drivers realized by kernel modules (root rights necessary for installation)

Exchange **`hotspots`** by **`advanced-hotspots`** in previous command line

Instructions retired is the basic indicator for processor utilization. Maximum is 4 simultaneous instructions per clock-tick.

The output shows CPI: clock-ticks per Instruction.
4 simultaneous instructions mean (CPI=0.25)

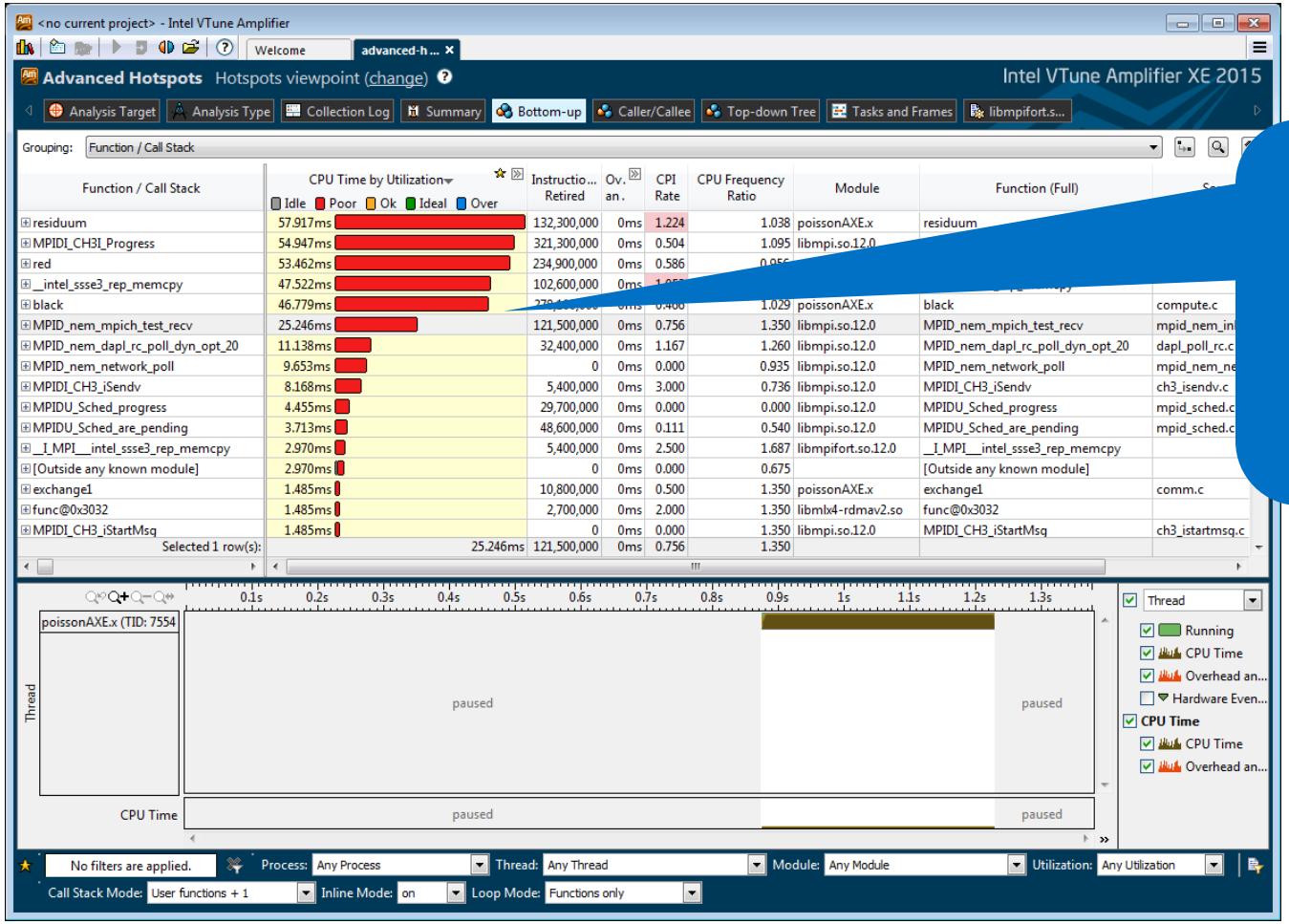
Advanced Hotspots: Summary



Instructions Retired: completed Instructions
CPI Rate: Clock Ticks per Instruction
CPU Frequency Ratio: >1 : Turbo boost!

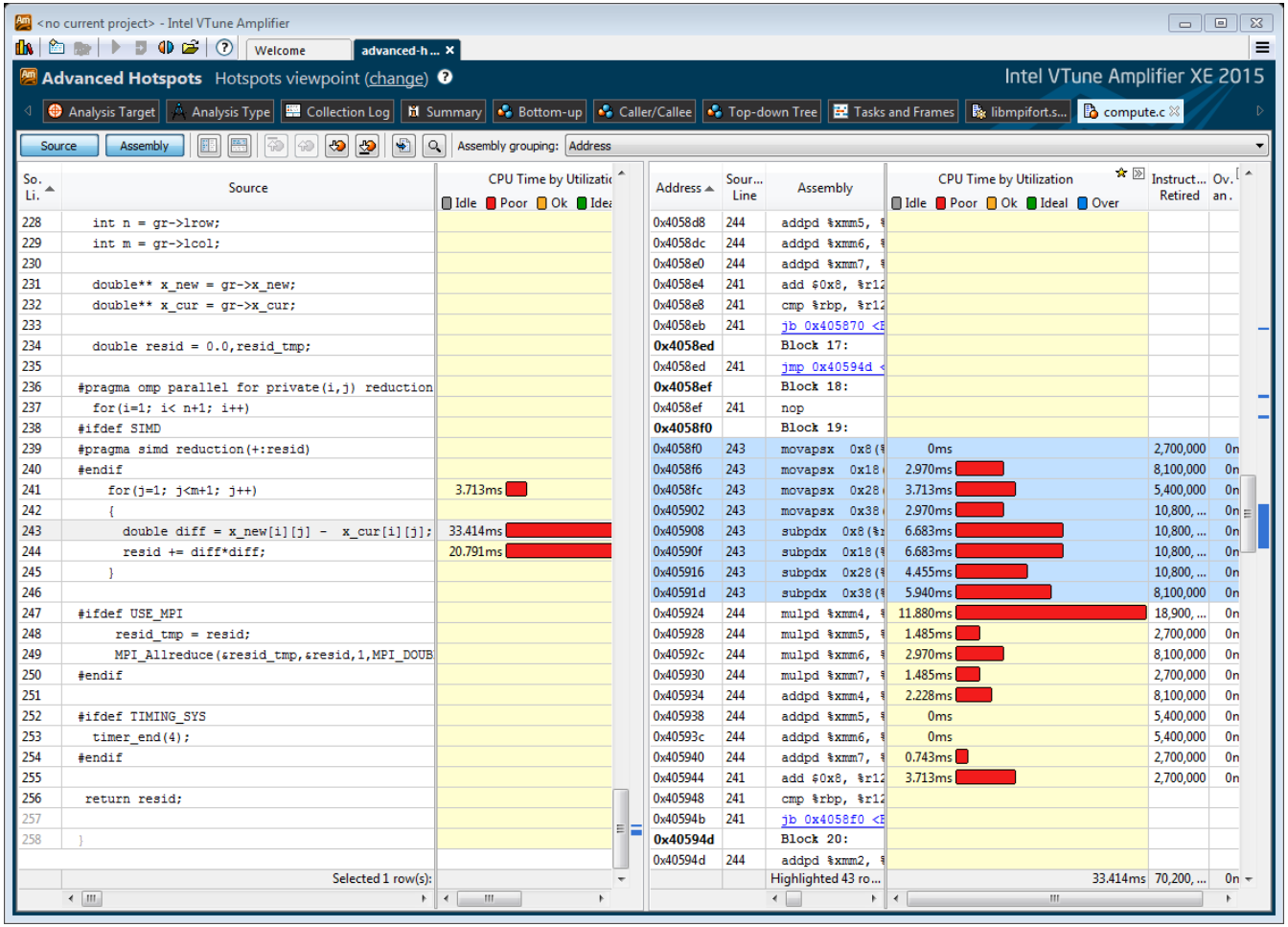
Second routine comes from MPI – Progress Engine
More MPI internal functions shown

Advanced Hotspots: Bottom-up



Order of Hotspot functions changes due to: 1. better time resolution 2. Internal MPI functions are displayed

Source and Assembly



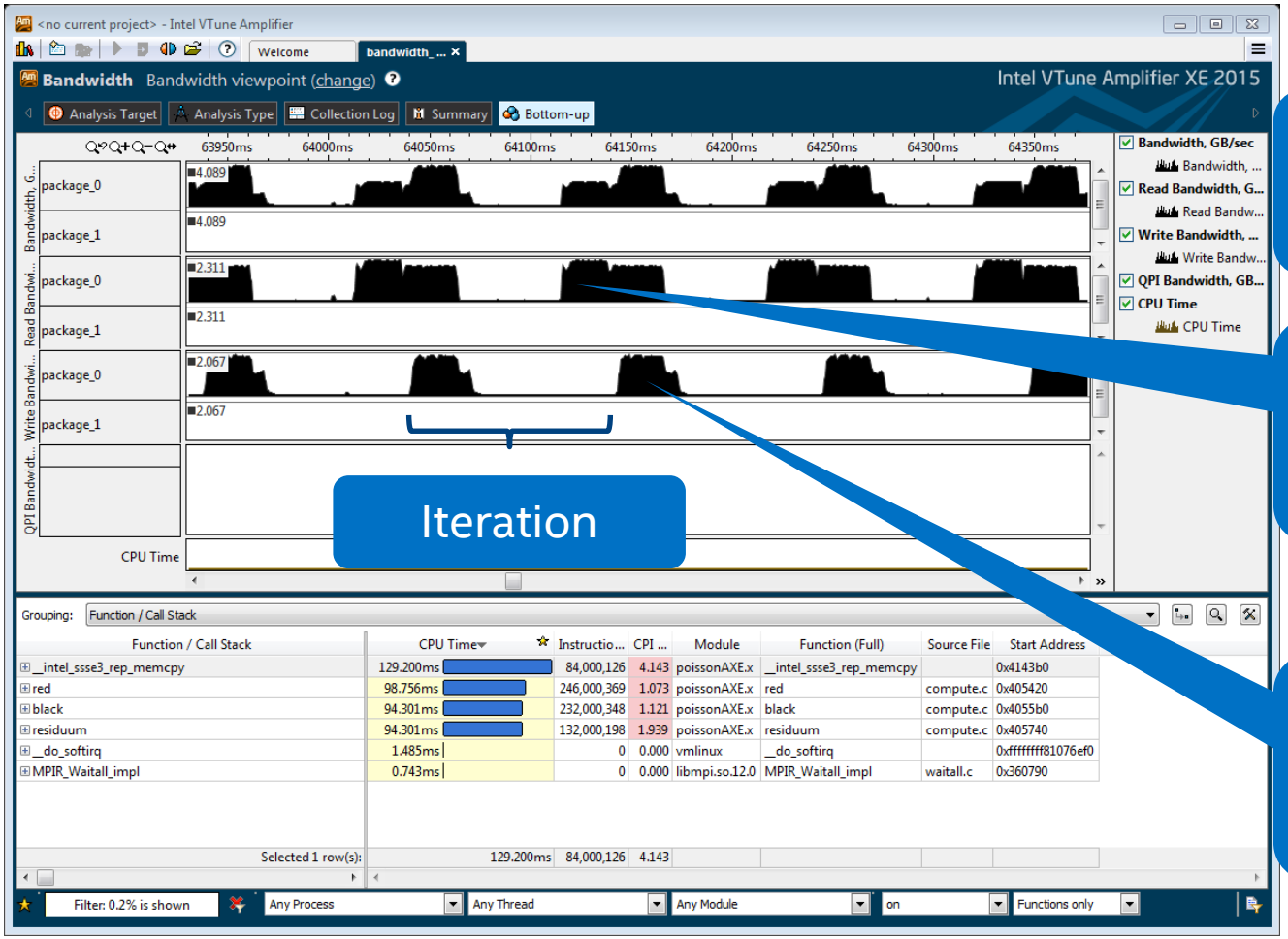
Bandwidth Analysis

The speedup curve for a single node shows saturation for more than 12 ranks per node (24 cores per node in total)

Intel® VTune™ Amplifier XE provides a Bandwidth analysis for proving this assumption

We concentrate on total bandwidth which can be related to the bandwidth that is delivered by the STREAM benchmark (~ 80GB/s on IVB dual Socket)

Bandwidth: Bottom-Up – Sequential Run

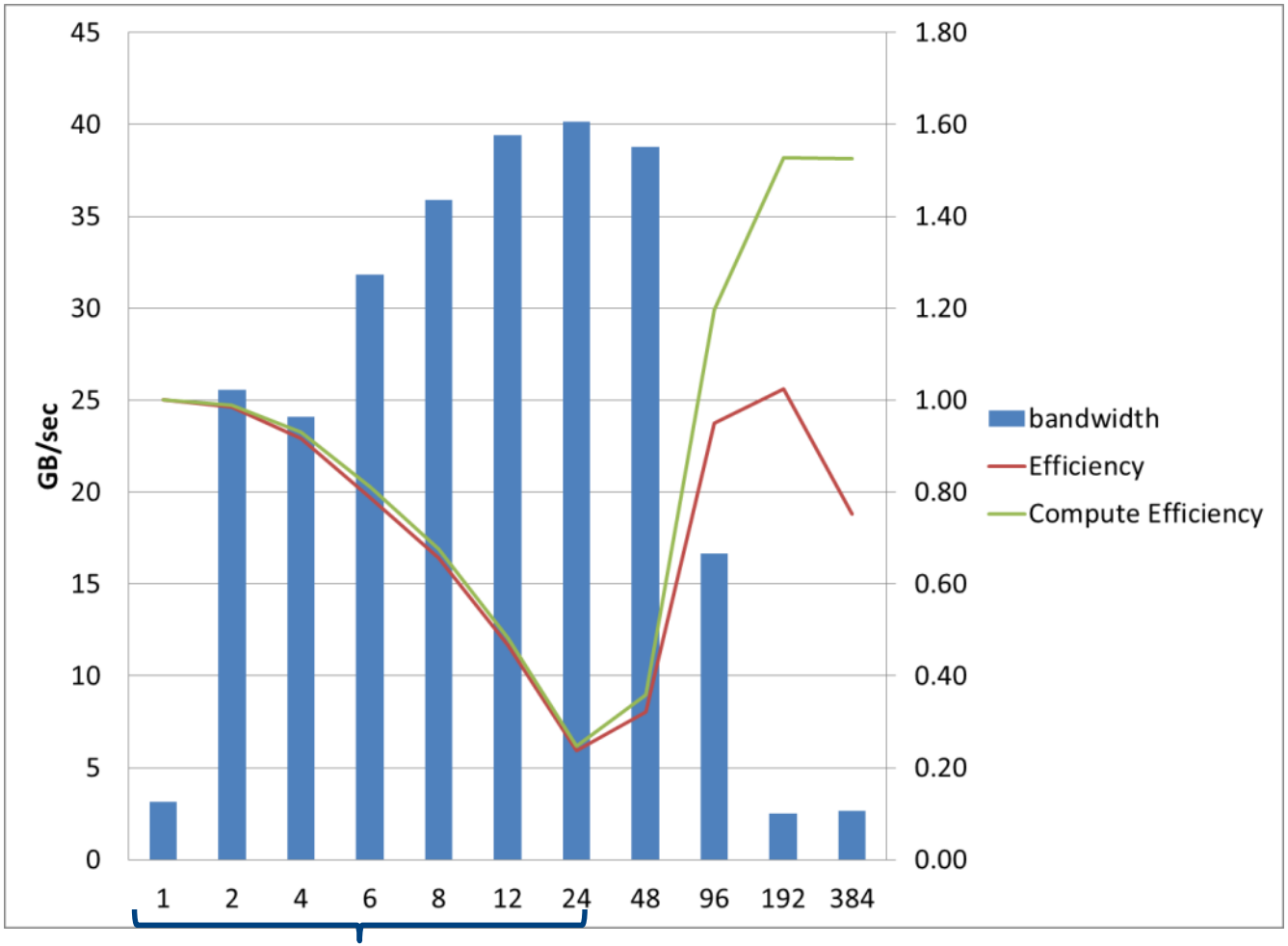


Zoomed until iterations are visible

Only significant read bandwidth for residuum

Copy routine generates read- and write bw.

Efficiency vs. Bandwidth on first node



Optimization Ideas

Bandwidth can be reduced by combining copy and residuum routine. This is possible because residuum is at the end and copy at the beginning of a new iteration

Bandwidth reduction may only have and impact in the bandwidth limited regime that we observe for this grid size only for less than 4 nodes

Prefetching of data may also improve performance in the copy and reduction loop

A blocked loop structure for the iteration loop may also improve data reuse

Optimization Summary

Two optimizations were successfully applied.

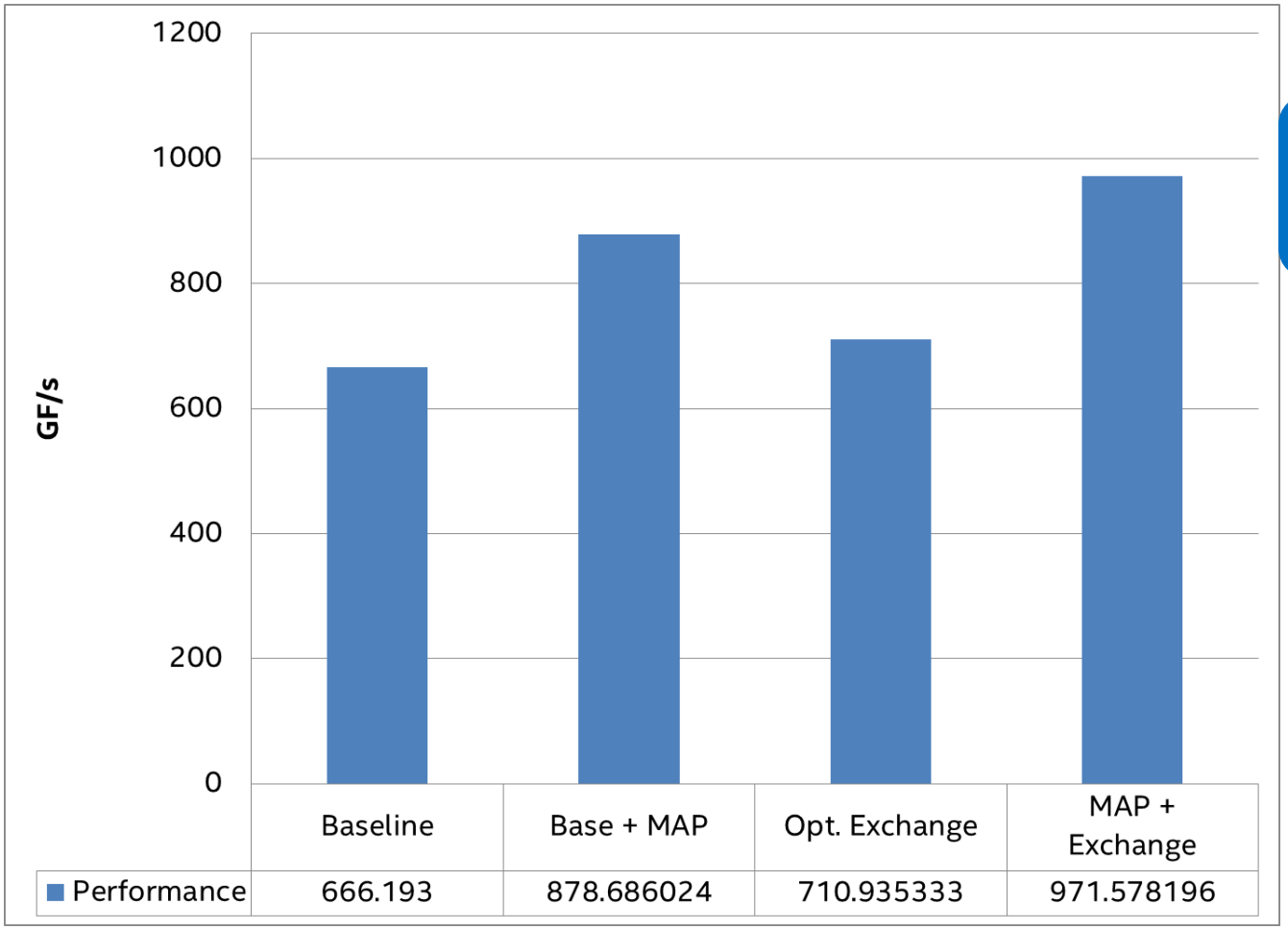
The first was the different Rank to Node mapping. This can be done by a special machine file. No code changes are necessary. This optimization will become more important for larger number of cores per node

The second optimization was to use immediate Sends/Recvs instead the blocking ones. The success can be evaluated with ITAC

Single process optimizations following the VTune™ Amplifier XE analysis has not been tried, so far

Following slide shows the impact of optimization for the largest available number of nodes (32)

Performance Improvement for 32 nodes



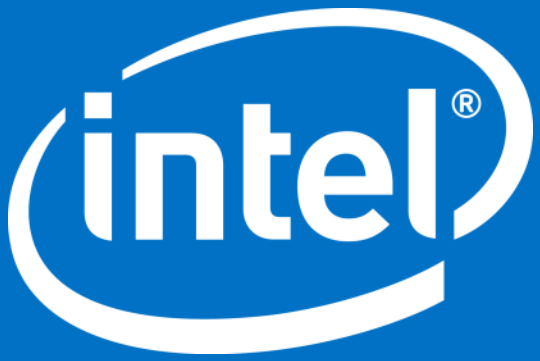
All 2D 32x24 process grids

Summary

Some methodologies were presented for performing a MPI analysis

ITAC offers interesting new features like simulation of ideal traces and the computation of transfer and waiting time

Intel® VTune™ Amplifier XE analyzes the compute part of the application. Bandwidth analysis is useful for many HPC applications


intel®

Backup – Optimized Mapping 24x16 Message Passing – total Volume