

# **MPI** Analysis





#### 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<sup>®</sup> Xeon<sup>®</sup> E5-2697 v2 processors Ivy Bridge (IVB) with 12 cores. Frequency: 2.7 GHz

2 processors per node (  $\rightarrow$  24 cores per node)

Mellanox MT4099 QDR Infiniband

**Operating system: RedHat EL 6.5** 

Intel® MPI 5.0.0.28

## 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



# Speedup for 2D and 1D process grids



inte

# Measuring MPI times with ITAC

Intel® Trace Analyzer - [1: C:/Users/hbockhor/Projects/Groh_XE_ONLY_ITAC_01/N-1_H24T1_PME-PPN0/mdrun_mpi.single.stf]       Image: State S							
				1			
Flat Profile Load Balance	e Call Tree Cal	ll Graph					
All_Processes 💌							
Name	TSelf	TSelf TTotal	#Calls	TSelf /Call	TSelf /Proc	TTotal /Proc	
All Processes							
Group Application	1.50587e+3 s	1.77979e+3	s 24	62.7445 s	62.7445 s	74.1581 s	
MPI Bcast	91.0121 s	91.0121	s 104844	868.072e-6 s	3.79217 s	3.79217 s	
MPI_Sendrecv	77.8351 s	77.8351	s 159743	487.252e-6 s	3.24313 s	3.24313 s	
MPI_Alltoall	73.2621 s	73.2621	s 26976	2.71583e-3 s	3.05259 s	3.05259 s	
MPI_Waitall	19.3109 s	19.3109	s 33831	570.803e-6 s	804.619e-3 s	804.619e-3 s	
MPI_Comm_split	5.22549 s	5.22549	s 408	12.8076e-3 s	217.729e-3 s	217.729e-3 s	
MPI_Scatter	2.51137 s	2.51137	s 24	104.64e-3 s	104.64e-3 s	104.64e-3 s	
MPI_Scatterv	1.68849 s	1.68849	s 72	23.4512e-3 s	70.3537e-3 s	70.3537e-3 s	
MPI_Allreduce	1.27823 s	1.27823	s 1608	794.919e-6 s	53.2595e-3 s	53.2595e-3 s	
MPI_Recv	1.11112 s	1.11112	s 3403	326.511e-6 s	46.2966e-3 s	46.2966e-3 s	
MPI_Isend	373.166e-3 s	373.166e-3	s 65322	5.71272e-6 s	15.5486e-3 s	15.5486e-3 s	
MPI_Irecv	186.462e-3 s	186.462e-3	s 65322	2.85451e-6 s	7.76925e-3 s	7.76925e-3 s	
MPI_Gather	48.955e-3 s	48.955e-3	s 252	194.266e-6 s	2.03979e-3 s	2.03979e-3 s	
MPI_Barrier	35.71e-3 s	35.71e-3	s 48	743.958e-6 s	1.48792e-3 s	1.48792e-3 s	
MPI_Finalize	22.003e-3 s	22.003e-3	s 24	916.792e-6 s	916.792e-6	916.792e-6 s	
MPI_Send	21.998e-3 s	21.998e-3	s 3403	6.4643e-6 s	916.583	916.583e-6 s	
MPI_Comm_rank	1.293e-3 s	1.293e-3	s 384	3.36719e-6 s	53.8	53.875e-6 s	
MPI_Comm_size	1.268e-3 s	1.268e-3	s 360	3.52222e-6 s	52	52.8333e-6 s	
MPI_Comm_free	1.24e-3 s	1.24e-3	s 48	25.8333e-6 s		51.6667e-6 s	
MPI_Type_commit	74e-6 s	74e-6	s 24	3.08333e-6	i i i i i i i i i i i i i i i i i i i	3.08333e-6 s	
MPI_Get_processo	_name 53e-6 s	53e-6	s 48	1.10417e	8	2.20833e-6 s	
MPI_Type_contigue	ous 50e-6 s	50e-6	s 24	2.0837	s	2.08333e-6 s	
MPI_Initialized	33e-6 s	33e-6	s 72	458	5	1.375e-6 s	

Two additonal columns showing timings per process: right click → Function Profile Settings 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





- 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 Bandwidth BW
- = transfer time for 0 byte message
- 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  $BW = \infty$  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

Idealization         To simulate the application behavior in an ideal comminication environment create an ideal tracefile by converting the original one - set up parameters below and click "Start". Press F1 for more details.	Name for idealized trace file gets additional
Ideal trace file name	"ideal"
_ITAC/2_D/poissonITC.x_16_PPN24_P384_V0.ideal.single.stf Browse	
Time range for conversion	
0.000000 start time, sec 0.438758 end time, sec	Press start to
✓ Open after creation ✓ Save as a single-STF file	continue
Conversion status: 0%	
Start Cancel	

# Waiting time due to dependencies



inte

## 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



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  $\rightarrow$  Application Imbalance Diagram



## Imbalance diagram – 16 Nodes 24x16

행 Imbalance Diagram	
Total Mode  Choose the display mode  Display the application time	Colors
12.4294	Interconnect time
11.6526	(T_transfer,acc)
10.0989	move mouse over bar:
9.32207	3.89 [sec]
8.54523	
7.76839	
<b>6</b> ,99155	
e 6.21471	Imbalance (T. wait acc):
5.43787	
4.66103	0.714021[Sec]
3.10736	
2.33052	
1.55368	Application(T compace):
0.776839	Application(1_comp,acc).
	/.5545 [sec]
Total Mode	contains some artificial
K	startup time

inte

# 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



inte

## 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 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

Intel® Trace Analyzer - [1: C:/Users/hbockhor/Projects/Cluster	ToolsTraininson_results/TrainingPXT/02_ITAC/1_D_nx1/pois	ssonITC.x_16_PPN24_P384_V1.stf]				
🥦 File Options Project Windows Help						
View Charts Navinate Advanced Lavout						
🔚 👔 🛄 🔛 0.030 399 - 0.056 909 : 0.026 510 Seconds	🔻 🖳 All_Processes 🛛 🗾 Major Function Groups	🖉 🍸 🔆 🕕 🏝 🗵 🔏 🛠				
Flat Profile Load Balance Call Tree Call Graph						
		Show Pies				
Name TSelf TSelf	TTotal #Calls TSelf /Call					
Process 121 11.83e-3 s	11.83e-3 s 1300 9.1e-6 s					
Process 122 11.738e-3 s	11.738e-3 s 1300 9.02923e-6 s					
Process 123 11.712e-3 s	11.712e-3 s 1300 9.00923e-6 s					
Process 124 11.568e-3 s	11.568e-3 s 1300 8.89846e-6 s					
Process 125 11.521e-3 s	11.521e-3 s 1300 8.86231e-6 s					
Process 126 11.5/9e-3 s	11.5/9e-3 s 1300 8.90693e-6 s					
Process 127 11.514e-3 s	11.514e-3 s 1300 8.85693e-6 s	Ranks U-143: 10x3600 boints				
Process 128 11.7/6e-3 s	11.7/6e-3 S 1300 9.05846e-6 S					
Process 129 11.98/e-3 5	11.98/e-3 s 1300 9.220//e-6 s	Danks 111 202, 0x2600 points				
Process 130 11.988e-3 5	11.966e-3 5 1300 9.22154e-6 5	Ranks 144-303. 983000 points				
Process 131 11.555e-3 5	11.555e-3 5 1300 5.23e-6 5					
Process 132 11.756-3 5	11.756E-3 5 1300 9.07335E-6 5	(10*144+9*240) = 3600				
Process 133 11.534e-3 5	11.534E-3 5 1300 5.15E-6 5	(10 1 + 15 2 + 0) = 5000				
Process 125 11 7590-2 c	11.744E-3 5 1300 9.045385E-6 5					
Process 135 11.7592 3 5	11.7572 3 3 1300 9.043572 6 3					
Process 137 11 474e-3 s	11 474e-3 e 1300 8 82616e-6 e	-				
Process 138 11.625e-3 s	11.625e-3 s 1300 8.94231e-6 s					
Process 139 11.633e-3 s	11.633e-3 s 1300 8.9484					
Process 140 11.844e-3 s	11.844e-3 s 1300					
Process 141 11.916e-3 s	11.916e-3 s					
Process 142 12.051e-3 s	12.051e-3 s 1300 9.27e-6 s					
Process 143 11.9e-3 s	11.9e-3 s 1300 9.15385e-6 s					
Process 144 12.778e-3 s	12.778e-3 s 1300 9.82923e-6 s					
Process 145 13.268e-3 s	13.268e-3 s 1300 10.2062e-6 s	Minimal MPI can be found by				
Process 146 13.027e-3 s	13.027e-3 s 1300 10.0208e-6 s	Minimat Mint Carrise Tourie by				
Process 147 13.156e-3 s	13.156e-3 s 1300 10.12e-6 s					
Process 148 12.92e-3 s	12.92e-3 s 1300 9.93846e-6 s	clicking on LSelf (Column				
Process 149 13.05e-3 s	13.05e-3 s 1300 10.0385e-6 s					
Process 150 13.03e-3 s	13.03e-3 s 1300 10.0231e-6 s	booder) - cort				
Process 151 13.009e-3 s	13.009e-3 s 1300 10.0069e-6 s	neader) - Sort				
Process 152 13.122e-3 s	13.122e-3 s 1300 10.0938e-6 s					
Process 153 13.358e-3 s	13.358e-3 s 1300 10.2754e-6 s					
Process 154 13.357e-3 s	13.357e-3 s 1300 10.2746e-6 s					
Process 155 13.272e-3 s	13.272e-3 s 1300 10.2092e-6 s					
Process 156 13.048e-3 s	13.048e-3 s 1300 10.0369e-6 s					
Process 157 13.305e-3 s	13.305e-3 s 1300 10.2346e-6 s					
Process 158 13.104e-3 s	13.104e-3 s 1300 10.08e-6 s	_				
Process 159 13.173e-3 s	13.173e-3 s 1300 10.1331e-6 s					

inte

# 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**



### 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 $\rightarrow$ 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 $\rightarrow$ Attribute to show

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

# Message Passing Profile: 24x16 grid



# 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  $\rightarrow$  Process Aggregation

This will pop up a new window: check All\_Nodes and apply

## Total Volume: 2D vs. 1D distribution



33

## Average Rate: 2D vs. 1D distribution



**Optimization Notice** 

34

# 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

0	1	2		7	8		14	15		Node #0	24 ranks per			
16	17	18		23	24		30	31		Node #1	node! One			
					R					Node #2	rank per core			
32	33	34		39	40	~	46	47		Node #3				
48	49	50		55	56		62	63						
64	65	66		71	72		78	79	$\searrow$	Additional horizontal				
20	01	07		07	00		04	05		exchange. 16 or 17 bounda lines between two nodes				
80	01	02		07	00		94	95						
96														
		[ pr	Defir oces bet	ning a ss gri tter -	a 16x d ma - whv	24 y be ?								
					vity	•					(intel)			

Copyright © 2014, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others.

38

# **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					

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

inte

# Impact of Optimized Mapping



40

## 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  $\rightarrow$  Quantitative Timeline

## Quantitative Timeline for 16 nodes



43

# 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  $\rightarrow$  Event Timeline

# Single Iteration Poisson

View Charts Navigate Advanced Layout	
🖬 👔 🏛 👩 0.045 289 - 0.045 532 : 0.000 243 Seconds 🔹 💆 All_Processes 🗾 MPI expanded in (Major Function Groups) 🔗 Y 🔅 🚺 🗓 🚺	Same configuration
	Same configuration
P0 MApplication AMPI MApplication MPI RecWPApplication MPI Recv MApplication MPI_Alkeduce	as used before in
P1 MEApplication APPLINA application MPI_Re WApplication MPI_Recv AMPI Application MPI_Allreduce	
P2 MPApplication AMPI AMMApplication MPI R AMMApplication MPI Recv MAMApplication MPI_Allreduce	the mapping
P3 MPApplication MPI Allreduce MPI Recy Mapplication MPI Recy Mapplication MPI Recy Mapplication MPI Allreduce	
P4 Meappication APP enceptication MP1 Recoveration MP1 Re	Section: 24X16
PS de-uplication MD My Moningtion MDI 0/// W NDI Decy Manington MDI All decy Monington MDI All advice	Apple
P7 MPIApplication MI MIMPI RecApplication VIZURecy AMPI Recy CApplication / WAR Recy MPI Recy	Applik
P8 MPApplication APP Lazy X-MRApplication VPI. Recv X-Application MPI Recv XXApplication MPI_Alreduce	Applic
P9 MPApplication MPL Strue VAXApplication MPL Alreduce	Applic
P10 MPApplication MPL Ray Availation MPL Mixed Availation MPL All reduce	Applic
P11 MPIApplication MT Reven WARApplication MPI acce WARApplication MPI Allreduce	-
P12 MPIApplication MPI_Alreduce MPI_Alreduce MPI_Alreduce	
P13 MPIApplication I TRANSMAR Application MPIARCON Application MPIARCON Application MPIARCON Application MPIARCON	I his is the default
P14 MPIApplication MPI Ref. 104 PI Application MPI Auto A 44 Application WPI Ref. WAAdopication MPI_Allreduce	
P15 MPI Application P1 Res / 1 APM Application / MPI P15 MPI Are Application MPI Alreduce	- mapping:
P16 MPLAppication MPLTV-PLAPPication VPPL Recv TV-Puppington MPL Recv TV-Puppington MPLAPPication	
P17 VEX.ppication PPPT-Representation P21. Rev Transportation VPT Rev Paperson approximation VPT All advantation VPT All advan	logi
P19 MPI Application MPI Application MPI Reck III WAM Application MPI Provident MPI Application MPI All reduce	
P20 MPIApplication CMPI ARPAApplication CMPI Recy IN CMPApplication MPI Recy III All reduce	Appli
P21 MPI Application VPI Recy MANApplication VPI Recy MANAPplication MPI Fact Manapplication MPI Allreduce	Appi
P22 MPT application VMPT PT TO MA Appletion MPT Rety MASS Avaluation MPT Rety Control PT All reduce	Appli
P23 MPI Application VI Application MPI_Allreduce	Appli
P24 MPL Application MPL Algorithm Application MPL Algorithm Application MPL Algorithm Application MPL Algorithm MP	Api
P25 MPI_Application MARAPPlication Application MPI_Allreduce	At
P26 MPI_Application ////////Application // Caller Application // C	Ap.
P27 MPI_Application MPI_Algoritation MPI_Algoritation MPI_Algoritation MPI_Algoritation MPI_Algoritation MPI_Algoritation	
·	· · · · · · · · · · · · · · · · · · ·

inte

## 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

Compare poissonITC.x_16_PPN24_P384_V0.ideal.single.stf	J						
Select tracefile to compare the original one with, and dick "OK". Resulting Comparison View will calculate the exact differences and speedups. Press F1 for more details.							
Compare poissonITC.x_16_PPN24_P384_V0.ideal.single.stf with <ul> <li>1: C:/Users/hbockhor/Projects/ClusterToolsTraining/Poisson_results/TrainingPXT/02_ITAC/2_D/poissonITC.x_16_PPN24_P384_V0.ideal.single.stf</li> <li>2: C:/Users/hbockhor/Projects/ClusterToolsTraining/Poisson_results/TrainingPXT/06_EXCHANGE1_ITAC/2_D/poissonITC.x_16_PPN24_P384_V0.ic</li> <li>C:/Users/hbockhor/Projects/ClusterToolsTraining/Poisson_results/TrainingPXT/06_EXCHANGE1_ITAC/2_D/poissonITC.x_16_PPN24_P384_V0.ic</li> </ul>							
OK Cancel							

Open another file for a comparison

## Comparison: Boundary Exchange

View Charts Navigate Advanced Layout Companion	Trace A. Cond/Deck
	Trace A. Senu/Recv
	Tuese Dules and /line and /late the ll
1 Application Application 2 provide Applicat	I race B: Isend/Irecv/waitall
P2 Application Application Application	
P3 Application Application Application Application	
P4 Application Application Application Application	
PS Application Application Application Application	
P6 Application /// Application / Y21 Application	
P7 Application Application /// P21 Application	
P8 Application Application / Application Application	Wait Lime on P6 has
P3 Application Application Application	
PUD Application upperform production poly	vonichadl
22 Andication Protection Andication	vanisneu!
PI3 Application Application Application	
P14 Application Application	
P15 Application Application Application	
P16 Application Application Application	
217 Application KopicaRon / / Application	·
P0 Application Application Application	
P1 Application Application Application	
P2 Application Application	
P3 Application Application 32 Application	
Registratori Appendiori Appendior	
Angleston Action	
P7 Application Application Application	MPI Wait Time for this
P3 Application Application Application	
P10 Application Application Application	poundary exchange
P11 Application Application Application	<b>J</b>
P12 Application Application Application Application	shrunk by a factor
23 Application Application Application Application	SHITIN DY a factor
1 <sup>21</sup> Replanding	0.000
Pis Application	0.689
P17 Application	erees
	7
Flat Profile Load Balance Call Tree	
AL_Processes	
B/A A TSelf TSelf TTotal #C-# rself/Call	
Group Application 1.033 0.994 n.a. n.a.	
Group MPI 0.689 1.127 0.612	

'inte

#### 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

View Charts Navigat	te Advanced Layout 0.096 846 - 0.097 588 : 0.000 :	742 Seconds	🕶 🌄 All_Processes 🔣 MP:	expanded in () 🔬 🍸 🗱 🕕 찶	2 🔏 🛠
P0 copy	MP	red	MPMplack	MPIMPIresiduum	MPI_Alireduce
P1 copy		Ared	MMPholack	AP MPT scesiduum	MPT Allreduce
	Medici	Ared	MC BW black	All and recide up	MOT Allreduce
		ve		STREAS PESICIUM	BP1_AMPHICE
РЗ сору		PI_Nred	MPLeM black	0 ev residuum	MPI_Allreduce
P4 copy		Milered	NET	Mare Stresiduum	MPI_Allreduce_o
P5 copy	100	f <mark>e</mark> (P)red	i Rolack	Millerresiduum	MPI_Allreduce
Рб сору	//\ e	ni <mark>Ve</mark> Mired	Vi V black	M2NN residuum	MPI_Allreduce
P7 copy	MP/ R	e o <mark>Ve</mark> Mred	NR Mblack	Mail residuum	MPI_Allreduce
P8 copy		o viered	MP W hack	MET 7/2 MET cesiduum	MPL Allredu
P9 copy			MP C 12 DIACK	E FIT SHAPPESIDUM	MELANPOIRE
P10copy	MPINE	Alt Refred	12 AM black	A Construction of the cons	MPI_Allreduce
P11 <mark>copy</mark>		1 Sevented	(1 TM/black	A 1214 MEresiduum	MPI_Allreduce
P12 <mark>copy</mark>	A.	Mill MPred	MPblack	MIE WIE residuum	MPI_Allreduce
P13 copy		4 Mired	VFI black	A residuum	MPI_Allreduce
P14 copy		VAN Wed	N Ablack	Musicesiduum	MPI Allreduce
010					
Piscopy		n ne e ned	NO DIACK	MET 1 SIGUUM	
P16 copy		192 Wred	MET No black	Mai Afresiduum	MPI_Aliceduci
P17 <mark>v</mark> copy		W avred	M // INtholack	12112 residuum	MPI_Allreduce
P18 <mark>copy</mark>		C Rired	NEO Weblack	and the state of t	MPI_Allreduce
P19 <mark>V</mark> copy		Mared	MH Nyblack	Mail V MPS residuum	MPI_AL
P20 copy	(1)	V P. Vevred	M2. Wolack	Me row residuum	MPT AII
R21 conv			MG Simplack		MDI Allendure
- 2 <u>iii copy</u>	Ĩ		U U UM		
P22 <mark>v</mark> copy	11	2 Kanvred	MP2 15 black	NE2 Coresiduum	MPL_Allreduce_
P23 <mark>v</mark> copy	(MP	1 F/ Mred	MPI / Volack	MPM Piresiduum	MPI_Aliredu
P24 conv	11	Control of the second	UNIE THUS Jack	Still in seich um	MDT Allendure
Flat Profile Load B	ialance Call Tree C	all Graph			
All_Processes	-				
Name	TSelf TSelf	TTotal #	Calls TSelf /Call		
All_Processes					
copy	16.7224e-3 s	16.7224e-3 s	139 120.305e-6 s		
residuum	15.9378e-3 s	23.3142e-3 s	96 166.019e-6 s		
red	11.407e-3 s	11.407e-3 s	96 118.823e-6 s 96 111 187e-6 c		
MPI Allreduce	7.37639e-3 s	7.37639e-3 s	96 76.8374e-6 s		
MPI_Recv	6.25e-3 s	6.25e-3 s	1080 5,7870		
exchange0	1.279e-3 s	8.782e-3 s	200 4.44097e-6 s		

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<sup>®</sup> VTune<sup>™</sup> 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<sup>™</sup> 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<sup>™</sup> 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:

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



**Optimization Notice** 

59

## Hotspots: Bottom-Up

Seno current project> - Intel VTune Amplifier										
11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1										
Basic Hotspots Hotspots by CPU Usage viewpoint (change) ? Intel VTune Amplifier XE 2015										
Analysis Target Analysis	Type Collection Log	🕅 Summary 🔗 Bottom	-up 🚱 Caller/Ca	llee 🔗 Top-down Tree	📰 Tasks and	Frames				
Grouping: Function / Call Stack				▼ 4.	9 9 12	Data Of Interest	(CPU Metrics)			
Function / Call Stack	CPU Time by Utiliza	ation <del>√</del> ★ ≫ Ov. ≫ deal Over an.	Module	Function (Full)	Source File	☆ Viewing ∢ 1 o 40.0% (	of 3 ▶ selected stack(s) 0.020s of 0.050s)			
PMPI_Allreduce	119.988ms	0ms	libmpi.so.12.0	PMPI_Allreduce	allreduce.c	liberei es 12.00	MDT Maitall anaitallia			
⊕ red	60.002ms	Oms	poissonAXE.x	red	compute.c	IIDmp1.so.12.0:PI	IVIPI_Waitali - Waitali.c			
	50.000ms	0ms	libmpi.so.12.0	PMPI_Waitall	waitall.c	poissonAXE.x!ex	chxb/d - comm.c:526			
	45.655ms	0ms	poissonAXE.x	intel_ssse3_rep_memcpy		poissonAXE.x!ex	:ch1960 - comm.c:307			
⊞ black	30.000ms	0ms	poissonAXE.x	black	compute.c	poissonAXE.x!m	ain+0xeb - poisson.c:209			
⊞ residuum	29.998ms	0ms	poissonAXE.x	residuum	compute.c	libc-2.12.so!_lib	ocUnknown]:[Unknown]			
🗆 exchange1	10.000ms	0ms	poissonAXE.x	exchange1	comm.c		I			
K exchange ← main ←libc_start_	10.000ms	Oms	poissonAXE.x	exchange	comm.c					
PMPI_Isend	9.999ms	Ums	libmpi.so.12.0	PMPI_Isend	isend.c					
Selected 1 row(s):		50.000ms 0ms								
۲ ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) (					F					
Q© <b>Q+</b> Q−Q⇔ 0.1	s 0.2s 0.5	0.5s 0.6s 0.7s	0.8s 0.9s 1	ls 1.1s 1.2s 1.3s	1.4s 1	.5s 1.6s	✓ Thread			
poissonAXE.x (TID: 3941 prg.sh (TID: 39388) prg.sh (TID: 39402)	(	Click on					Running     Multi CPU Time			
uname (TID: 39402) prg.sh (TID: 39414)	fund	ction nar	ne			≡ paused	CPU Sample			
⊨ dapli_thread_init (TID: 3	+	o rovoal					CPU Usage			
dapli_thread_init (TID: 3	L L	U leveal					CPU Time			
dapli_thread_init (TID: 3	dapli_thread_init (TID: 3						🕑 🚧 Overhead an			
poissonAXE.x (TID: 3944	SO	urce viev	N							
CPU Usage		paused				paused				
٠.						+ »				
📩 No filters are applied. 🛛 🎇	Any Process	<ul> <li>Any Thread</li> </ul>		<ul> <li>Any Module</li> </ul>	-	Utilization: Any U	tilization 💌 📴			
Call Stack Mode: User functions +	Call Stack Mode: User functions +1 Inline Mode: on Loop Mode: Functions only									

3 different stacks for MPI\_Waitall. Source line of call to exchange in poisson.c is shown. Exchange is called 3 times!

inte

## Hotspots Analysis: Source

쪒 <no< td=""><td>current project&gt; - Intel VTune Amplifier</td><td></td><td></td><td></td><td></td></no<>	current project> - Intel VTune Amplifier				
<b>1</b> 1 🖄	🛾 🎥 🕨 🗗 🕼 🚔 🕜 🛛 Welcome 🛛 hotsp	ots_38 ×		=	
💹 Ba	sic Hotspots Hotspots by CPU Usage viewp	point ( <u>change</u> ) 🕐	Ir	tel VTune Amplifier XE 2015	Click on first
⊲ [⊕	Analysis Target 🙏 Analysis Type 📮 Collection Log	🖬 Summary 🦂 Bottom-up 🚱 Caller/Callee	🚱 Top-down Tree 🗮 Tasks and	Frames 🖡 comm.c	Click off first
					stack and
Sou	Assembly	Assembly grouping: Address	<b>•</b>	Data Of Interest (CPU Metrics)	
So. Li. ▲	Source	CPU Time: Total by Utilization 🖄	CPU Time: Self by Utili	40.0% (0.020s of 0.050s)	poisson.c line
205 206	#ifdef USE_MPI			libmpi.so.12.0!PMPI_Waitall - waitall.c	shows first cal
207	// boundary exchange			poissonAXE.xlexchxb/d - comm.c:526	to ovebange a
208	exchange (cn. cgn) :	20.001 mc	Ome	poissonAXExtexch1900 - comm.cisor	to exchange a
210	<pre>#endif</pre>	20.001115	UIIS	libc-212 sol_libc_Unknown]:[[]nknown]	line 200
211					tine 209
212					
213	// RED update				
214	red(sp. com);				
215	IEU(«þ, «gi),				
217	#ifdef USE_MPI				Second call at
218	// boundary exchange				
219					line 220 shows
220	exchange(sp,sgr);				
221	#endlI				up hy selecting
223	// BLACK update		<b>—</b>		ap by secceding
224	-		E		another stack
225	<pre>black(sp, sgr);</pre>				another stack
226					
227	<pre>#ifdef USE_MPI</pre>				
228	// boundary exchange	-			
230	exchange(sp,sgr);				
231	#endif				
- 222	Selected 1 row(s):	20.001ms			
	< +	< III	•		

'inte

#### 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



63

#### Advanced Hotspots: Bottom-up

Intel VTune Ample Amp	plifier					
	advanced-h ×				lotol V/Tupo Am	olifior VE 2015
Advanced Hotspots Hotsp	lots viewpoint ( <u>change</u> )					
🕘 🕀 Analysis Target 🕺 Analysis Typ	pe 🔛 Collection Log 🚹 Summary 🐼 B	ottom-up 😽 Caller	/Callee 🏼 🚭 Top	-down Tree 🛛 🔀 Tasks and	Frames 🔤 kibmpifort.s	₽
Grouping: Function / Call Stack						/
Function / Call Stack	CPU Time by Utilization → 🖈 🗵 II Idle III Poor III Ok III Ideal III Over	Instructio Ov. ≫ Retired an .	CPI CPU Freq Rate Rati	o Module	Function (Full)	Sam
⊕ residuum	57.917ms	132,300,000 0ms	1.224	1.038 poissonAXE.x	residuum	
MPIDI_CH3I_Progress	54.947ms	321,300,000 0ms	0.504	1.095 libmpi.so.12.0		
⊕ red	53.462ms	234,900,000 0ms	0.586	0.950		
❀intel_ssse3_rep_memcpy	47.522ms	102,600,000 0ms	1.055			
⊞ black	46.779ms	270,000,000 0005	0.400	1.029 poissonAXE.x	black	compute.c
	25.246ms	121,500,000 0ms	0.756	1.350 libmpi.so.12.0	MPID_nem_mpich_test_recv	mpid_nem_inl
■ MPID_nem_dapl_rc_poll_dyn_opt_20	11.138ms	32,400,000 0ms	1.167	1.260 libmpi.so.12.0	MPID_nem_dapl_rc_poll_dyn_opt_20	dapl_poll_rc.c
MPID_nem_network_poll	9.653ms	0 0ms	0.000	0.935 libmpi.so.12.0	MPID_nem_network_poll	mpid_nem_ne
MPIDI_CH3_iSendv	8.168ms	5,400,000 0ms	3.000	0.736 libmpi.so.12.0	MPIDI_CH3_iSendv	ch3_isendv.c
MPIDU_Sched_progress	4.455ms	29,700,000 0ms	0.000	0.000 libmpi.so.12.0	MPIDU_Sched_progress	mpid_sched.c
MPIDU_Sched_are_pending	3.713ms	48,600,000 0ms	0.111	0.540 libmpi.so.12.0	MPIDU_Sched_are_pending	mpid_sched.c
	2.970ms	5,400,000 0ms	2.500	1.687 libmpifort.so.12.0	_I_MPIintel_ssse3_rep_memcpy	
	2.970ms	0 0ms	0.000	0.675	[Outside any known module]	
⊕ exchange1	1.485ms	10,800,000 0ms	0.500	1.350 poissonAXE.x	exchange1	comm.c
	1.485ms	2,700,000 0ms	2.000	1.350 libmlx4-rdmav2.so	func@0x3032	
MPIDI_CH3_iStartMsg	1.485ms	0 0ms	0.000	1.350 libmpi.so.12.0	MPIDI_CH3_iStartMsg	ch3_istartmsq.c
Selected 1 row(s)	1: 25.246ms	121,500,000 0ms	0.756	1.350		· ·
· •	•		III			•
Q© <b>Q+</b> Q−Q+ 0.1:	s 0.2s 0.3s 0.4s 0.5s	0.6s 0.7	/s 0.8s	0.9s 1s 1.	ls 1.2s 1.3s	Thread
poissonAXE.x (TID: 7554	paused				paused v	✓     ✓       ✓ </td
CPU Time	paused				paused	
No filterr are applied	Process: Any Process	d: Any Thread		Module: Any Module	▼ Utilization April	tilization
No filters are applied.	Threa	any Inread		Module Any Module	Othization: Any U	
Call Stack Mode: User functions + 1	<ul> <li>Inline Mode: on Loop Mod</li> </ul>	de: Functions only	-			

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

(intel) 64

### Source and Assembly

💯 <no< th=""><th colspan="11"></th></no<>											
	Madvanced Hotenete Hotenete Visupeint (chapse)										
Ad 🗠											
⊲ 🐣	🖉 🖶 Analysis Target 🗛 Analysis Type 🚆 Collection Log 🕅 Summary 🚭 Bottom-up 🚭 Caller/Callee 🚭 Top-down Tree 🛃 Tasks and Frames 💱 libmpifort.s 🚯 compute.c 🖄 🖊 👘										
Sour	Source Assembly 📳 🗒 🐼 🥸 😰 🔍 Assembly grouping: Address 🔹										
So. Li.	Source	CPU Time by Utilizatio	^	Address 🔺	Sour Line	Assembly	CPU Time by Utilization 🛛 🖈 🗵 🔲 Idle 📕 Poor 📋 Ok 📱 Ideal 📮 Over	Instruct Retired	Ov. [ ^ an .		
228	<pre>int n = gr-&gt;lrow;</pre>			0x4058d8	244	addpd %xmm5, %					
229	<pre>int m = gr-&gt;lcol;</pre>			0x4058dc	244	addpd %xmm6, %					
230				0x4058e0	244	addpd %xmm7, %					
231	double** x_new = gr->x_new;			0x4058e4	241	add \$0x8, %r12					
232	double** x_cur = gr->x_cur;			0x4058e8	241	cmp %rbp, %r12					
233				0x4058eb	241	jb 0x405870 <b< td=""><td></td><td></td><td></td></b<>					
234	<pre>double resid = 0.0,resid_tmp;</pre>			0x4058ed		Block 17:			_		
235				0x4058ed	241	jmp 0x40594d <			_		
236	<pre>#pragma omp parallel for private(i,j) reduction</pre>			0x4058ef		Block 18:					
237	for(i=1; i< n+1; i++)			0x4058ef	241	nop					
238	#ifdef SIMD			0x4058f0		Block 19:			_		
239	<pre>#pragma simd reduction(+:resid)</pre>			0x4058f0	243	movapsx 0x8(%	0ms	2,700,000	0n		
240	#endif			0x4058f6	243	movapsx 0x18	2.970ms	8,100,000	0n		
241	for(j=1; j <m+1; j++)<="" td=""><td>3.713ms</td><td></td><td>0x4058fc</td><td>243</td><td>movapsx 0x28</td><td>3.713ms</td><td>5,400,000</td><td>0n</td></m+1;>	3.713ms		0x4058fc	243	movapsx 0x28	3.713ms	5,400,000	0n		
242	{			0x405902	243	movapsx 0x38	2.970ms	10,800,	0n ≘		
243	<pre>double diff = x_new[i][j] - x_cur[i][j];</pre>	33.414ms		0x405908	243	subpdx 0x8(%)	6.683ms	10,800,	0n		
244	resid += diff*diff;	20.791ms		0x40590f	243	subpdx 0x18(%	6.683ms	10,800,	0n		
245	}			0x405916	243	subpdx 0x28(%	4.455ms	10,800,	0n		
246				0x40591d	243	subpdx 0x38(%	5.940ms	8,100,000	0n		
247	#ifdef USE_MPI			0x405924	244	mulpd %xmm4, %	11.880ms	18,900,	<u>0n</u>		
248	resid_tmp = resid;			0x405928	244	mulpd %xmm5, %	1.485ms	2,700,000	On		
249	MPI_Allreduce(&resid_tmp,&resid,1,MPI_DOUB			0x40592c	244	mulpd %xmm6, %	2.970ms	8,100,000	0n		
250	#endif			0x405930	244	mulpd %xmm7, %	1.485ms	2,700,000	0n		
251				0x405934	244	addpd %xmm4, %	2.228ms	8,100,000	0n		
252	#ifdef TIMING_SYS			0x405938	244	addpd %xmm5, %	Oms	5,400,000	0n		
253	timer_end(4);			0x40593c	244	addpd %xmm6, %	0 742	5,400,000	0n		
254	#enali			0x405940	244	addpd %xmm7, %	0./43ms	2,700,000	0n 0n		
255				0:405944	241	add suxs, %r12	5./15ms	2,700,000	Un		
250	return resia;			0:405948	241	cmp srpp, srl2			_		
257	1		Ξ	0x40594b	241	JD UX4058IO <e< td=""><td></td><td></td><td></td></e<>					
208	}			0:405044	244	DIOCK 20;			_		
	Selected 1 row(r)		-	0x40594d	244	Highlighted 43 ro	22 /11/mc	70 200	0.0 -		
	4 III k	4 III b				A b	35.414ms	70,200,	b b		
	·	· · · · · · · · · · · · · · · · · · ·			_		`				



### 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<sup>®</sup> VTune<sup>™</sup> 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



67

#### Efficiency vs. Bandwidth on first node



**Optimization Notice** 

Copyright © 2014, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others.

68

inte

## **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





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




## Backup – Optimized Mapping 24x16 Message Passing – total Volume

Intel® Trace Analyzer - [1: C:/Users/hbockhor/Projects/ClusterToolsresults/TrainingPXT/02_ITAC/2_D/poissonITC.x_16_PPI     Ele Options Project Windows Help	N24_P384_V0.stf]	Intel® Trace Analyzer - [1: C/Users/hbockhor/Projects/ClusterToolsTraining/Poisson_results/TrainingPXT/04a_PE_Place/2_D/_V1.stf] Intel® Trace Analyzer - [1: C/Users/hbockhor/Poisson_results/TrainingPXT/04a_PE_Place/2_D/_V1.stf] Intel® Trace Analyzer - [1: C/Users/hbockhor/Poisson_results/TrainingPXT/04a_PE_Place/2_D/_V1.stf] Intel® Trace Analyzer - [1: C/Users/hbockhor/Poisson_results/TrainingPXT/04a_PE_Place/2_D/_V1.stf] Intel® Trace Analyzer - [1: C/Users/hbockhor/Poisson_results/TrainingPXT/Pa_Place/2_D/_V1.stf] Intel® Trace Analyzer - [1: C/Users/hbockhor/Poisson_results/TrainingPXT/Pa_Place/2_D/_V1.stf] Intel® Trace Analyzer - [1: C/Users/hbockhor/Pa_Place/2_D/_V1.stf]
View Charts Navigate Advanced Layout		View Charts Navigate Advanced Layout
🔚 🛐 🎹 💩 0.000 000 - 0.438 758 : 0.438 758 Seconds 📑 🌅 All_Nodes 😾 Major Function Groups 🎸	2 🍸 🔆 🏶 🌒 🖄 🗵 🛛 »	Total Data Volume (B) (Sender by Receiver)
Total Data Volume (B) (Sender by Receiver)		
esq24 esq16 esq16 esq19 esq19 esq19 esq19 esq19 esq19 esq17 esq17 esq17 esq17 esq17 esq17 esq17 esq17 esq17 esq	17 Sum Mean StdDev	jesq02( jesq02) jesq02( jesq02) jesq02( jesq02) jesq02( jesq02( jesq02) jesq03( jesq03) jesq03( jesq03) jesq04( jesq0
G esg241 24.8 M 9.08 M	33.8 M 16.9 M 7.84 M	Gesgozo 39.5 M 2.13 M 2.10 M 2.10 M
G esg167 9.08 M 24.8 M 8.72 M	42.6 M 14.2 M 7.48 M	G esg021 2.19M 34.9M 2.19M 2.18M 41.5M 10.4M 14.2M
Gesg168 8.72M 24.8M 9.08M	42.6 M 14.2 M 7.48 M	Gesg022 2.19M 349H 2.19M 2.18M 0ntimized
Gesg190 9.08M 24.6M 6.72M Original	42.6 M 14.2 M 7.48 M	G esg023 2.19M 34.9M 2.18M ODUITITZEC 15.4M
G esg191 8.72M 24.8M 9.08M	42.6 M 14.2 M 7.48 M	6 esg024 2.13M 2.13M 2.13M Mapping 14.2M
G esg192 9.08 M 24.8 M 8.72 M	42.6 M 14.2 M 7.48 M	G esg026 2.18M 2.19M 34.9M 2.19M 2.18M 13.1M
G esg195 8.72 M 24.8 M 9.08 M	42.6 M 14.2 M 7.48 M	G esg027 2.18M 2.19M 34.9M 2.19M 2.18M 43.7M 8.73M 13.1M
G esg196 9.08 9.08 8.72 M	42.6 M 14.2 M 7.48 M	G esg028 2.18M 2.19M 34.9M 2.18M 41.5M 10.4M 14.2M
G esg201 8.72 4.8 9.08 M	42.6 M 14.2 M 7.48 M	G esg031 2.18M 34.9M 2.19M 2.18M 44.5 M 10.4 M 14.2 M
G esg173 9.08 M 24.8 M 8.72 M	42.6 M 14.2 M 7.48 M	Gesg032 2.18M 2.19M 34.9M 2.18M 43.7M 8.73M 13.1M
G esg174 8.72 M 24.8 M 9.08 M	42.6 M 14.2 M 7.48 M	Gesg033 2.19M 2.19M 2.19M 2.19M 2.18M 43.7M 8.73M 13.1M
G esg175 9.08 8.72 M	42.6 M 14.2 M 7.48 M	G esg034 2.18M 2.19M 34.9M 2.18M 41.5M 10.4M 14.2M
G esg176 8.72 M 24.8 M 9.08 M	42.6 M 14.2 M 7.48 M	2.18M         34.9M         2.19M         39.3M         13.1M         15.4M
Gesg177 9.08 24.8 8.72 M	42.6 M 14.2 M 7.48 M	Gesg042 2.19M 2.19M 2.19M 2.19M 41.5M 10.4M 14.2M
G esg178 8.72 M 24.8 M 9.0	8 M 42.6 M 14.2 M 7.48 M	Gesg043 2.19M 2.19M 34.9M 2.19M 41.5M 10.4M 14.2M
G esg179 9.08 M 244	33.8 M 16.9 M 7.84 M	Gesg050 2.19M 24.9M 34.9M 39.3M 13.1M 15.4M
Sum 33.8 M 42.6 M 33.1	3 M 664 M 9.7 M	Sum 29.3M 41.5M 41.5M 39.3M 41.5M 43.7M 43.7M 43.7M 41.5M 41.5M 43.7M 43.7M 41.5M 39.3M 41.5M 41.5M 39.3M 664M
Mean 16.9M 14.2M 1	9 M 14.4 M	Mean         13.1M         10.4M         13.1M         10.4M         8.73M         10.4M         8.73M         8.73M         10.4M         13.1M         10.4M         13.1M         10.4M
StdDev 7.84 M 7.48 M	ам 7.55 M	StdDev 15.4M 14.2M 14.2M 15.4M 14.2M 13.1M 13.1M 14.2M 14.2M 13.1M 14.2M