

mpiP, TAU, Scalasca on Blue Gene

mpiP: 3.1.2

TAU Version: 2.17.1

Scalasca 1.0rc2

Author: Prashobh Balasundaram

pbalasun@ie.ibm.com

Agenda – Introduction to TAU and Scalasca

- Introduction
- mpiP – Light weight MPI profiling
- Overview of TAU features - presentation
- TAU front end tools - demo
 - ❖ Paraprof
 - ❖ Perfexplorer
 - ❖ Jumpshot
 - ❖ Automatic instrumentation using TAU
- Advantages and disadvantages of normal profiling and tracing tools
- Common bottlenecks in MPI programs
- Concepts of Scalasca
- Automatic identification of MPI bottlenecks using Scalasca
- Scalasca – demo using image processing code
 - ❖ Cube 3 Viewer usage

Introduction - Trivia

- Parallel programming using message passing model is harder than serial programming
- Communication bottlenecks can cause poor program scaling
- Normally it is difficult to analyze communication patterns
- Especially true for massively parallel processing applications
- Developers need better tools to analyze complex communication patterns.
- These tools should be
 - ❖ FAST
 - ❖ AUTOMATIC
 - ❖ USER FRIENDLY
 - ❖ DELIVER RELEVANT INFORMATION
 - ❖ Cover FORTRAN, C, C++ ---- MPI, OpenMP, OMPI

Introduction – Definitions

■ Profiling

- ❖ Recording of **summary information during execution** of a program
- ❖ **low cost performance assessment**
- ❖ Identifies computational hotspots
- ❖ Normally related to function calls, loops, user defined events etc
- ❖ Typically implemented through sampling and/or instrumentation
- ❖ Records number of calls – inclusive, exclusive time etc....

■ Tracing

- ❖ Recording of data at significant points – called **events**
- ❖ Event trace is a **time sequenced stream of event records**
- ❖ Entering, leaving functions, loops, user defined events
- ❖ Thread/process interactions like send/receive messages
- ❖ Timestamp, CPU/Process/Thread identifier etc are recorded

Introduction

■ Hardware counter profiling

- ❖ Processors expose counters in hardware which can be read to analyze performance
- ❖ Can be read using standard software like
 - Performance application programming Interface (PAPI)
 - Performance counter library (PCL)
- ❖ Many interesting counters for TLB misses, Cache misses etc
- ❖ Used to identify hot spots in code
- ❖ Blue gene HW exposes many important HW counters, and PAPI on Blue gene exposes the most important ones

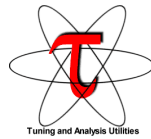
mpiP 3.1.2

- Light weight MPI profiling tool
- Very low overhead
- Usable on both BlueGene/L and Blue Gene/P
- Generates a text report with details of function calls and MPI time
- MPI time per process per function split-up will be generated
- Can trace back to the originating function

- Simple source code, can be modified to build custom applications
- Easy to use, simple
- Well suited for initial investigations

- Covered in Exercise

TAU Introduction



- Tuning and Analysis Utilities
- TAU is a **Performance System framework**
 - ❖ Automatic instrumentation
 - ❖ Profile measurement
 - ❖ Trace measurement
 - ❖ Memory tracking and memory leak detection
 - ❖ Performance experimentation tools
- Aims to be
 - ❖ Portable across machines/software stack
 - ❖ Integrated environment, yet highly configurable
 - ❖ Applicable for a general complex system
 - System with nodes, contexts and threads
- It is a **Developer's Tool**
 - ❖ Powerful, modular
 - ❖ Configure each time for the current purpose – install at your home directory

Parallel profiling using TAU

- Profile all threads/processes of a parallel program
 - ❖ Profile the communication between threads/processes
 - ❖ Visualization of profile data – pprof, Paraprof
 - ❖ Call path profiling – graphical representation, clickable call paths
 - ❖ Histograms of functions across processors
 - ❖ Profile data can include hardware counter data
- TAU is highly configurable
- Documentation available on configuration options
- For profiling use flags – Command Line – workshop exercise available
 - ❖ -PROFILE
 - ❖ -PROFILECALLPATH
 - ❖ -PROFILEPARAM
 - ❖ -PROFILESTATS

Automatic Instrumentation

- Implemented using Program Database Toolkit
- Manual modification can be done on automatic instrumented code – This is the recommended practice
- Generates profiles for each process
- Profile files can then be viewed with pprof or with paraprof

Screenshot of ParaProf – Paraprof manager

TAU: ParaProf Manager

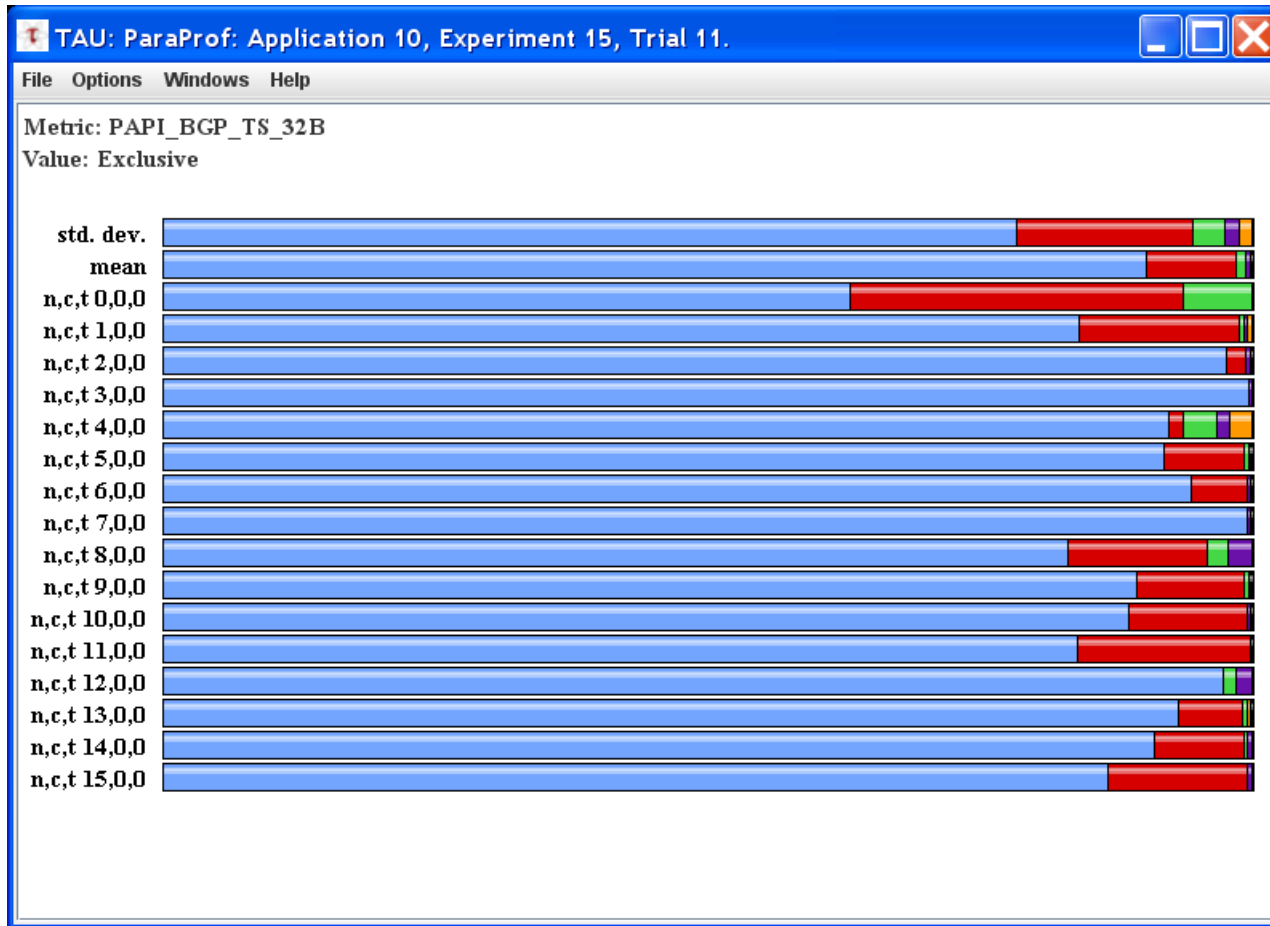
File Options Help

Applications

- Standard Applications
 - Default App
- perfdmf (jdbc:postgresql://localhost:5432/perfdmf)
 - ImageProcessing
 - 1D_16_BGL_DEF
 - C:\Documents and Settings\Prashobh\Desktop\Work\Layout Optimization
 - PAPI_BGP_TS_32B**
 - 1D_16_BGL_MAP
 - 1D_64_BGL_DEF
 - 2D_64_BGL_DEF
 - 2D_64_BGL_MAP
 - ID_64_BGL_MAP

MetricField	Value
Name	PAPI_BGP_TS_32B
Application ID	10
Experiment ID	15
Trial ID	11
Metric ID	0

Screenshot of ParaProf – Paraprof manager



Screenshot of ParaProf – Paraprof manager

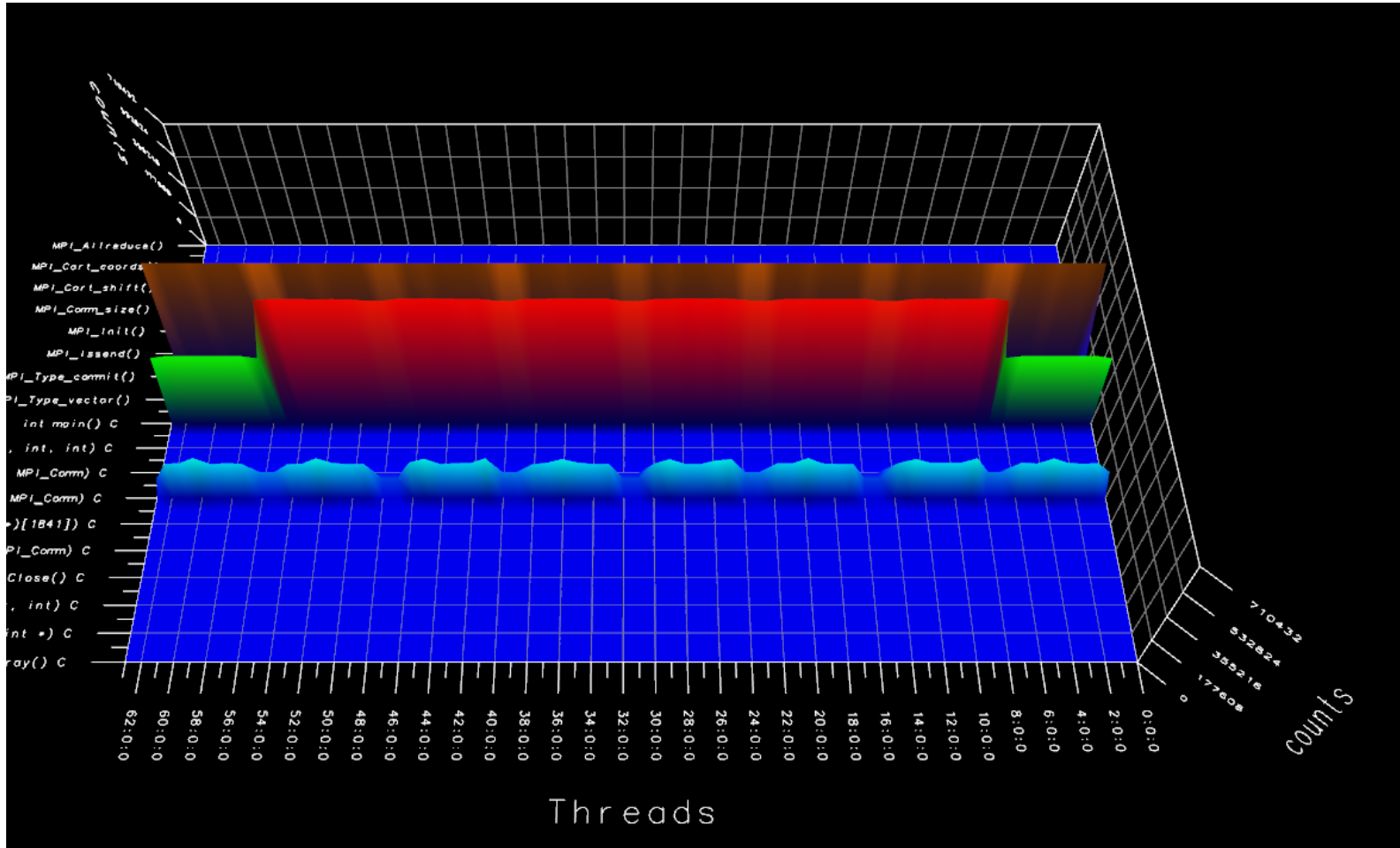
TAU: ParaProf: n,c,t, 2,0,0 - Application 10, Experiment 20, Trial 16.

File Options Windows Help

Metric: PAPI_BGL_TS_32B
Sorted By: Exclusive
Units: counts

%Total	Counts	Exclusive	Inclusive	#Calls	#Child Calls	Inclusive/Call	Name
44.5	631562	631562	631562	1	0	631562	MPI_Reduce()
25.8	366232	366232	366232	6000	0	61.039	MPI_Waitall()
18.8	267036	267036	267036	24000	0	11.126	MPI_Issend()
55.4	152964	786232	6000	78000	131.039	void image_HaloSwap(float **	
0.0	288	288	12	0	24	MPI_Allreduce()	
0.0	40	40	1	0	40	MPI_Cart_create()	
0.0	34	34	1	0	34	MPI_Init()	
100.0	0	1418156	1	12019	1418156	int main() C	
0.0	0	0	1	0	0	MPI_Bcast()	
0.0	0	0	1	0	0	MPI_Cart_coords()	
0.0	0	0	2	0	0	MPI_Cart_shift()	
0.0	0	0	2	0	0	MPI_Comm_rank()	
0.0	0	0	1	0	0	MPI_Comm_size()	
0.0	0	0	1	0	0	MPI_Finalize()	
0.0	0	0	24000	0	0	MPI_Irecv()	
0.0	0	0	12000	0	0	MPI_Type_commit()	
0.0	0	0	6000	0	0	MPI_Type_contiguous()	
0.0	0	0	6000	0	0	MPI_Type_vector()	
0.0	0	0	4	4	0	void *arralloc(size_t, int,	
0.0	0	144	6	6	24	void image_ComputeAvgPixelVa	
0.0	0	144	6	6	24	void image_ComputeDelta(floa	
0.0	0	0	1	4	0	void image_InitVars(float **	
44.5	0	631562	1	2	631562	void image_Integrate(float (
0.0	0	0	6000	0	0	void image_Reconstruct(float	
0.0	0	0	1	1	0	void image_Scatter(float (*)	
0.0	0	0	1	1	0	void parallelEnv_Close() C	
0.0	0	34	1	3	34	void parallelEnv_Init(MPI_Co	
0.0	0	0	1	0	0	void proc_ComputeGridDims(in	
0.0	0	40	1	3	40	void proc_CreateVirtualGrid(
0.0	0	0	1	2	0	void proc_FindNeighbours(MPI	

Screenshot of ParaProf – Paraprof manager

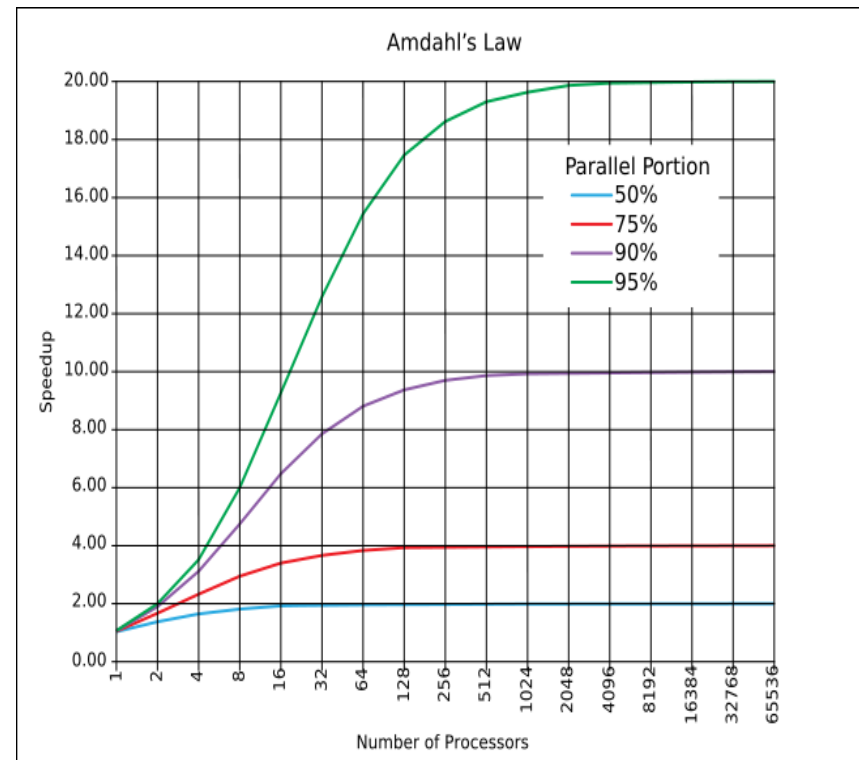


Communication Bottlenecks

- Message passing model relies on inter-process messages
- Communication bottlenecks are a major issue in parallel computing
- Issue is magnified on Massively parallel supercomputers
- Amdahl's Law
 - ❖ P is the proportion of a program that can be made parallel
 - ❖ $(1 - P)$ is the proportion that cannot be parallelized
 - ❖ then the maximum speedup =

$$\frac{1}{(1 - P) + \frac{P}{N}}$$

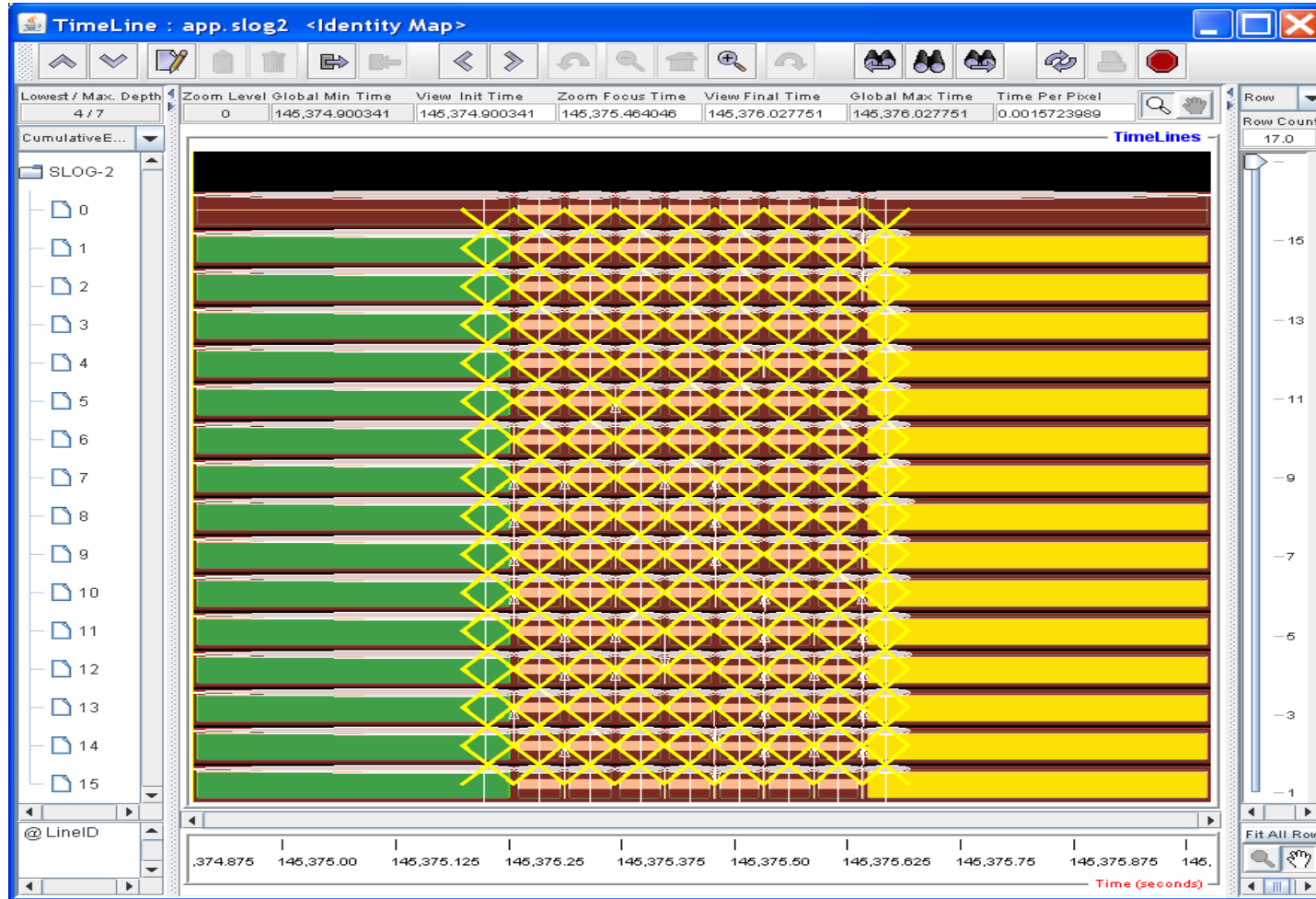
- Disaster!!! if application cannot scale to 1000's of processors



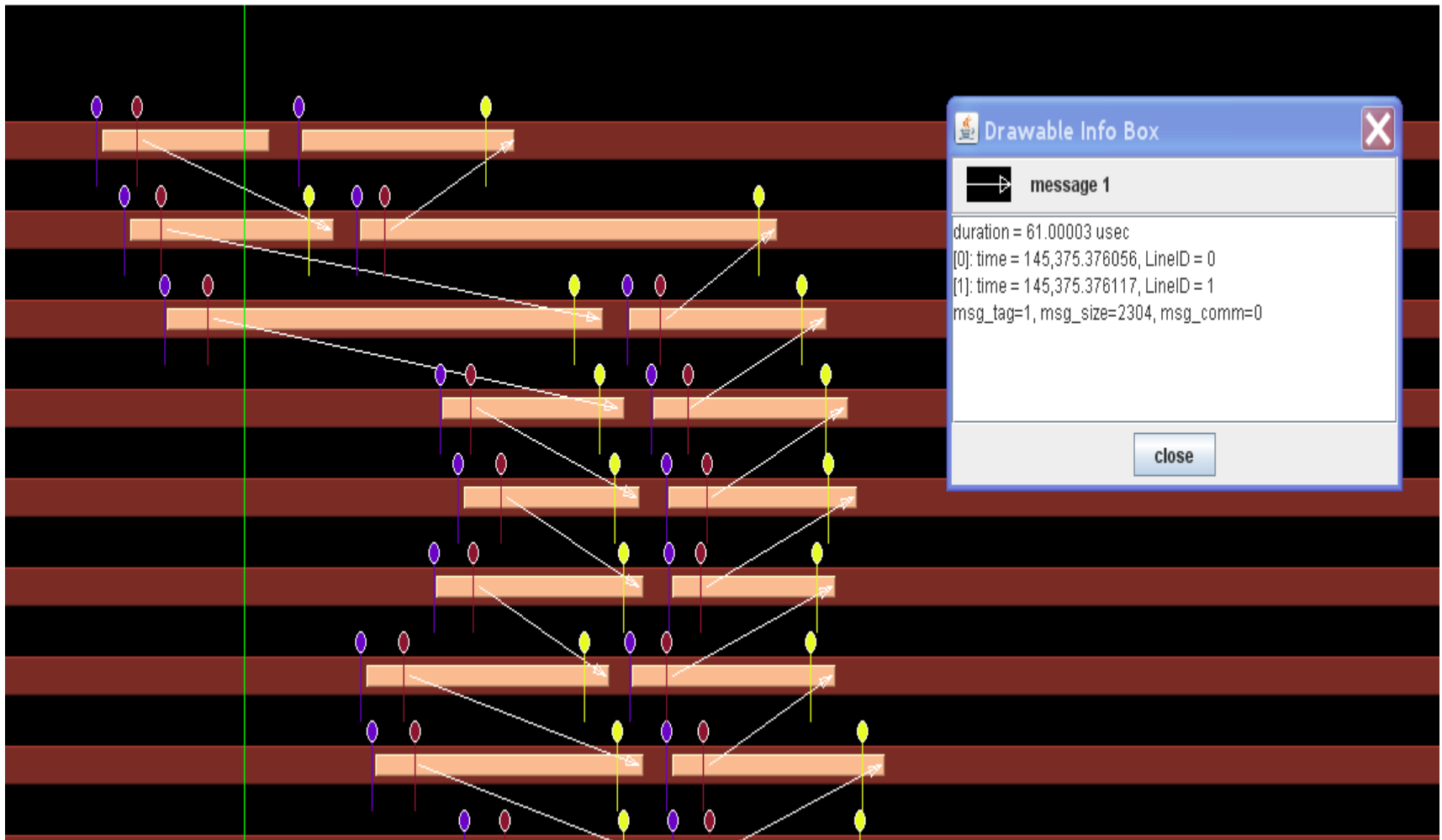
Tracing

- MPI Program tracing
- Trace Visualization using Jumpshot ~ Tracing tool integrated with TAU
- Many trace formats and conversion tools are available
 - ❖ Open Trace Format
 - ❖ Vampir Trace Format – Vampir
 - ❖ EPILOG trace format
 - ❖ Slog2 (Scalable Log Format)
- Tau trace files can be merged, converted to slog2 (tau2slog2)
- Utilities come with tau for slog2 – others can be configured.
- Jumpshot uses slog2 – scalable log format
- Lots of research on trace format files, compression etc
- Trace visualization can display high level issues with communication characteristics
- User then manually drills into the trace files to finer levels of detail

Screenshot of Jumpshot – High Level View

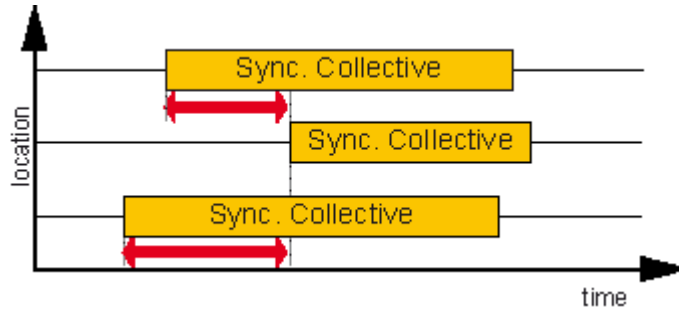


Screenshot Jump shot - drilling down the timeline

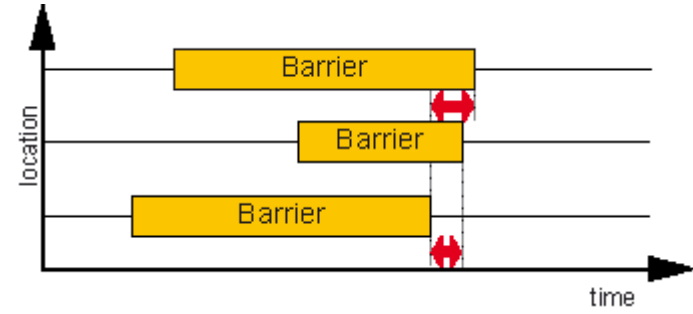


Tracing Visualization pros and cons

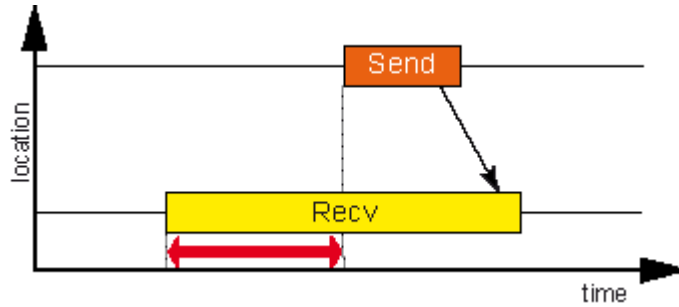
- Visualizing MPI calls is great help
- Can spot issues with MPI communication patterns visually
- User can identify message sizes, time taken
- Can also try to reorder communication after analysis
- Many communication related bottleneck patterns, parameters are well defined.
 - ❖ Wait at Barrier Time
 - ❖ Barrier Completion Time
 - ❖ Late sender Time
 - ❖ Late sender wrong order Time
 - ❖ Late receiver Time
 - ❖ Early reduce Time
 - ❖ Early scan Time
 - ❖ Late Broad cast time
 - ❖ Wait NXN Time
 - ❖ NXN completion Time



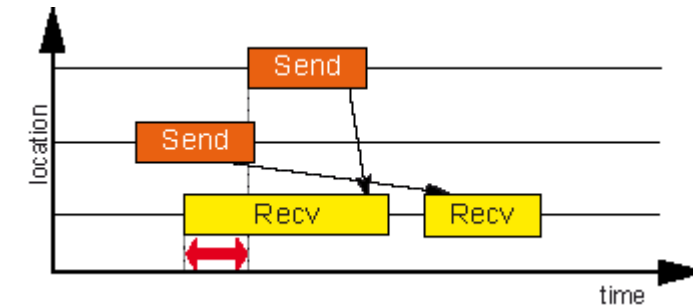
Wait at Barrier Time



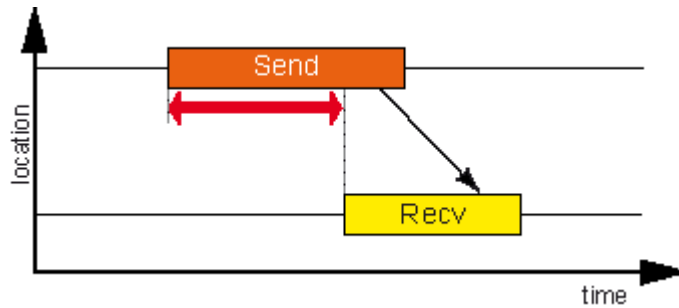
Barrier Completion Time



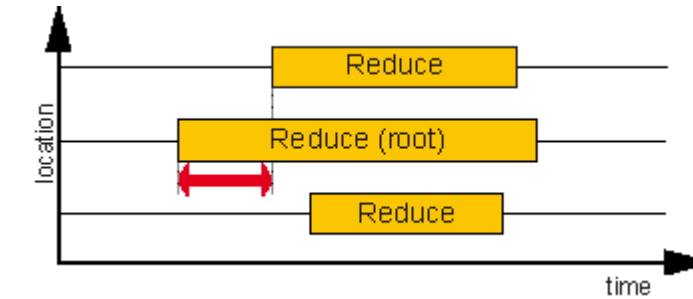
Late sender Time



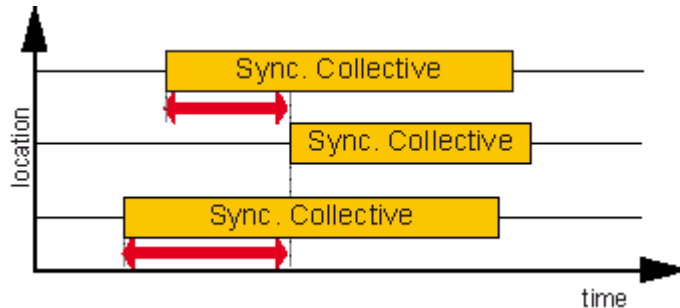
Late sender wrong order Time



Late receiver Time



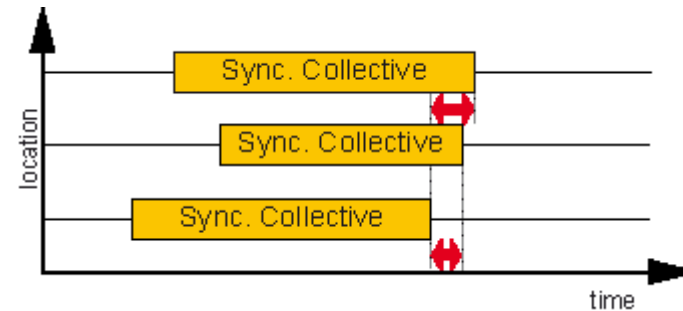
Early reduce Time



Wait at NxN

Parameters:

Total Communication Time
 Number of Visits
 Execution Time
 Overhead Time
 MPI Time
 MPI Synchronization Time
 MPI I/O Time
 MPI Collective Synchronization Time
 MPI Point to Point Communication Time



NxN Completion time

Synchronizations

Point to Point Synchronizations

Collective communications as source

Collective communications as destination

Etc

Communication patterns in MPI code and solutions to bottlenecks is a major research area.

TAU Tracing cons

- The user needs to analyze vast amounts of data visually
- Finding needle in hay stack !!!
- Large trace files need to be merged → HUGE trace files
- Slow tracefile reading, etc are common issues
- TAU **cannot identify the bottle neck communication patterns** automatically
- Great if a tool could automatically do this fast (means - In Parallel)
- This is where Scalasca makes its entry

- TAU is still useful
 - ❖ Can do profiling with hardware counters
 - ❖ Can do memory tracking, head room Analysis, memory leak detection
 - Even on FORTRAN MPI programs to a great extent
 - ❖ Traces can be easily converted to many industry standard formats
 - ❖ Can be combined with many other front end tools like vampir, paravir etc
 - ❖ Trace reader API is simple and convenient – But not as powerful as Scalasca's EARL

scalasca

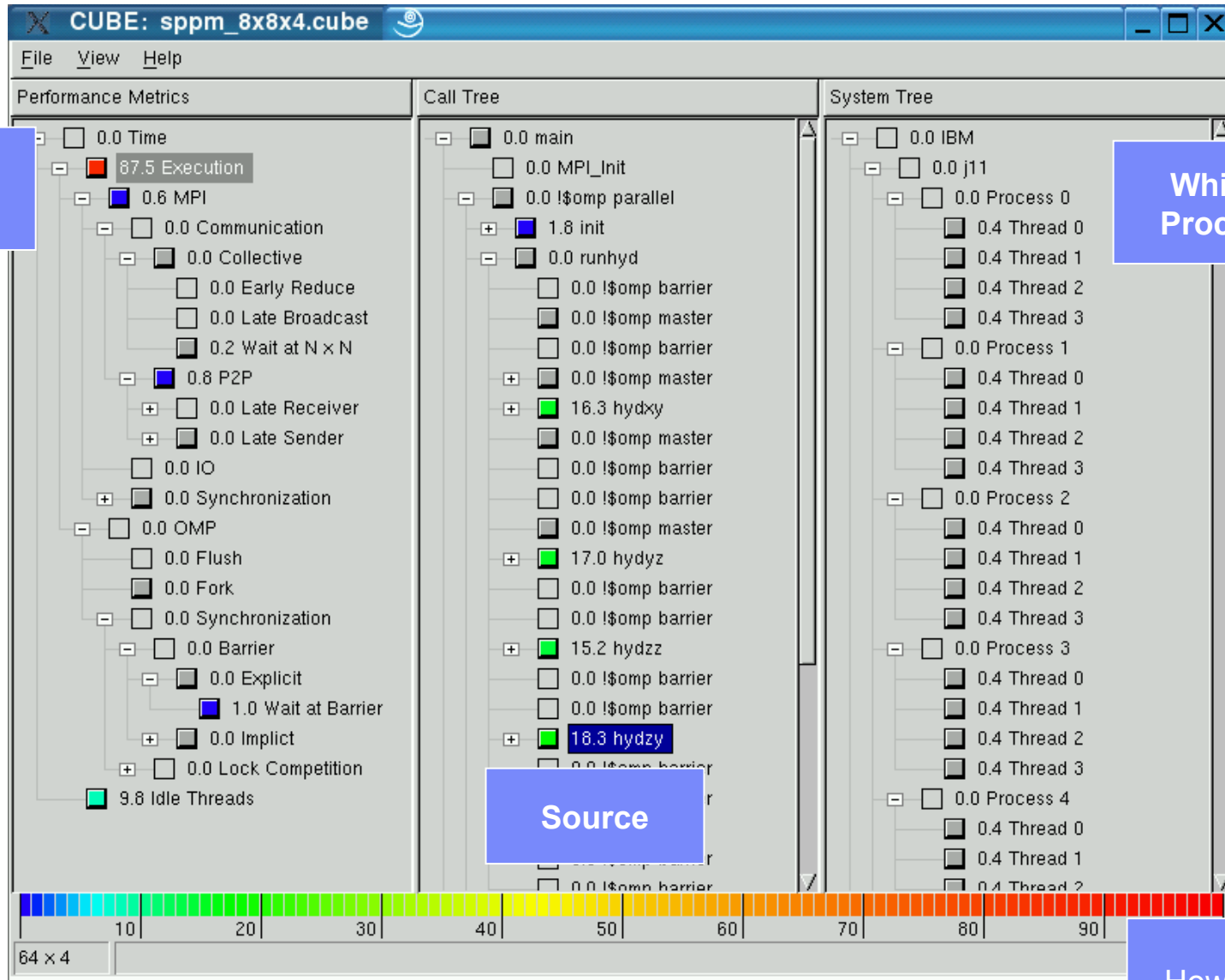
- Scalable performance analysis for large scale parallel applications
- From John Von Neumann Institute for computing, Forschungszentrum Jülich
- Successor of KOJAK
- KOJAK = Kit for **objective judgement** and **automatic knowledge** based detection of bottlenecks

- In addition to Kojak capabilities, Scalasca is a parallel analyzer by itself
- No need to merge trace files, though tools are available to do it
- Analyzes the trace files through a “REPLAY” of the events in parallel
- Analysis runs on the Blue gene backend in Parallel
- Therefore quite fast
- User friendly – Only three major Commands to remember
 - ❖ **SKIN, SCAN, SQUARE**

Skin, Scan, Square

- Skin = Scalasca Instrumentor
- Scan = Scalasca collector and analyzer
- Square = Scalasca analysis report explorer
- Scout = Parallel Automatic performance Analyzer

- Scalaca includes the following components
 - ❖ OPARI = OpenMP and User region instrumentation Tool
 - ❖ EPIK = Measurement Library for summarization and Tracing
 - ❖ PEARL = Parallel Event Trace Analysis Library
 - ❖ SCOUT = Parallel Automatic performance Analyzer, invoke from scan
 - ❖ CUBE3 = Analysis report presentation component and Utilities
- Scalasca uses EPILOG as the trace file format
- Note: TAU can also convert its traces to EPILOG format
 - ❖ But the accuracy of these traces need to be verified by repeated usage

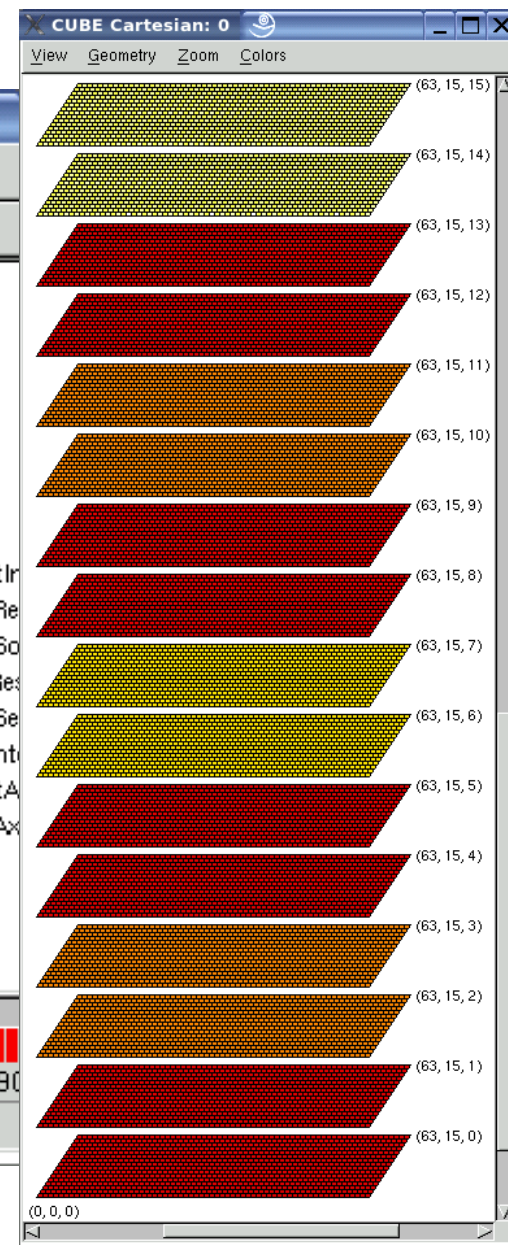
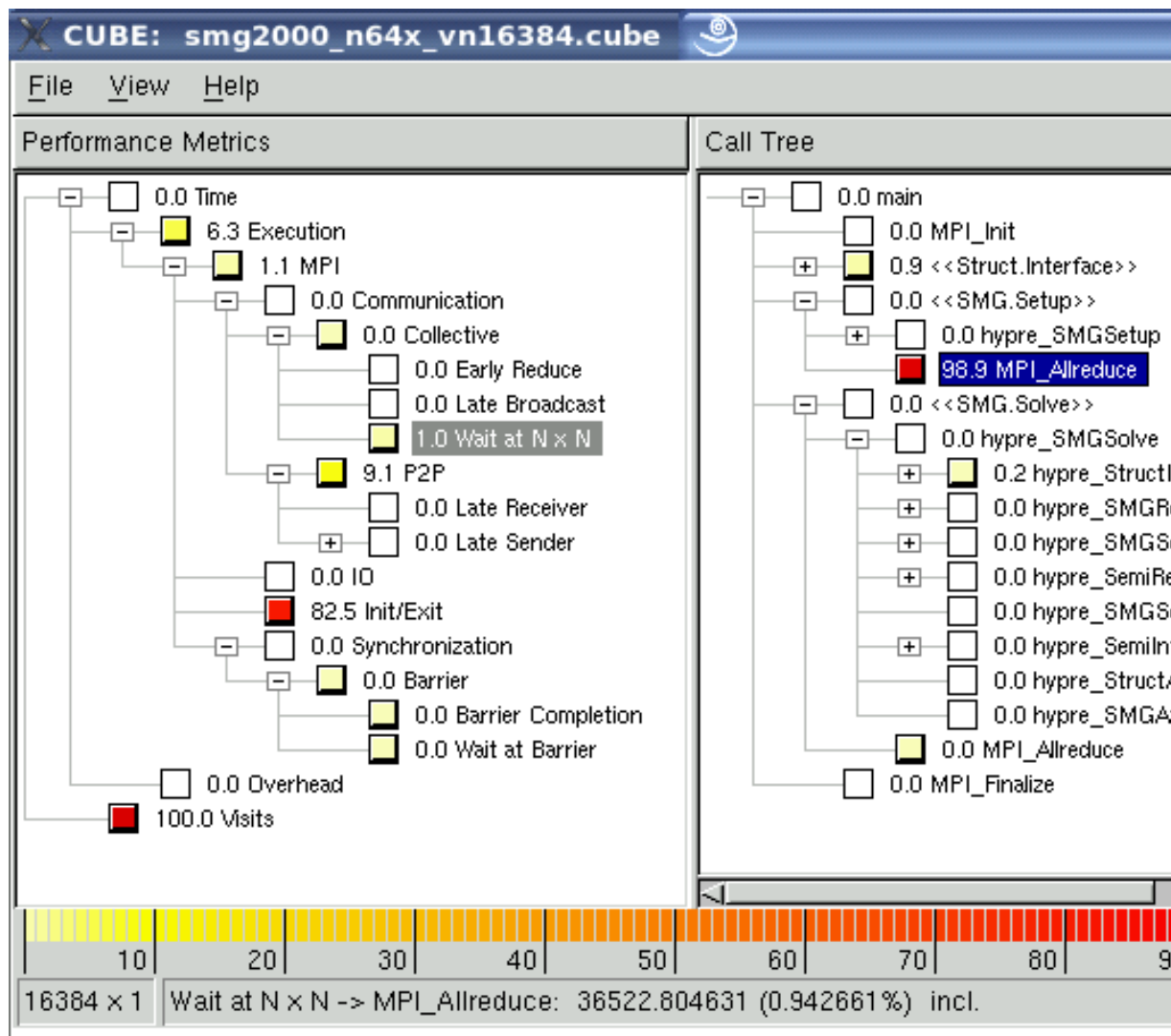


What Problem

Which Process

Source

How bad?



Workshop exercise of Scalasca on BlueGene/L

- Using a simple image processing application
- At Low processor count the application has lesser communication issues
- At higher processor count communication is an issue as there is far less work to be done.
- Tool shows these communication characteristics clearly

- In order to use Scalasca on BlueGene/P and BlueGene/L – No need to install yourself
 - ❖ On BGL : `module load scalasca-bgp`
 - ❖ On BGL : `module load scalasca-bgl`

Weblinks

- <http://www.cs.uoregon.edu/research/tau/home.php>
- <http://www.fz-juelich.de/jsc/scalasca/>

Acknowledgments

- Brendan Boulter, IBM
- Guy Robinson, IBM
- Lars Schneidenbach, IBM