

Performance tools on BlueGene

Brian J. N. Wylie

John von Neumann Institute for Computing,
Forschungszentrum Jülich



Disclaimer

- I'm a (long-time) developer of performance tools
 - and I use performance tools frequently in my work
- I'm very familiar with some tools, much less with others
 - faults may be due to my misunderstanding
- Performance analysis & tuning is often an art
 - but basic techniques & use of the right tools can be learned
- Most application developers aren't aware of available tools
 - which unfortunately often differ from system to system
 - and are sometimes complicated to learn to use (effectively)
- Your mileage may vary!

Overview

- Basic questions
 - How is my application performing?
 - Why isn't it performing as it should?
 - What are the relevant performance metrics?
 - What are appropriate performance analysis techniques?
- Selected tools for BlueGene/L
 - IBM HPC Toolkit
 - mpiP
 - TAU
 - Scalasca
- Analysis & tuning tips

Basics

- Performance metrics
 - Visits/Calls, Time, Comm/Synch
 - Heap/Memory, I/O, Hardware counters
- Performance sources
 - MPI library functions, User functions/regions/phases
 - callsite/callpath context
- Performance locations
 - MPI processes (threads?)
- Profiling/summarization vs. event tracing
 - aggregated/averaged vs. time-dependent information

IBM HPC Toolkit*

- ACTC research prototype
- Commercial product from IBM?
 - annual subscription fee based on number of nodes
 - ~~US\$20,000~~ list price for 512+ nodes: US\$5,000 for BG/L only
- Unified package for IBM AIX & Linux/Power systems
- Suite of tools (most don't support BlueGene yet)
 - User, CPU/HWC, MPI, ~~OpenMP~~, Memory, I/O metrics
 - recompile/relink ~~or binary analysis & instrumentation~~
 - profiling & tracing options
 - measurement summary & trace viewers
 - ~~Eclipse PTP IDE plug-in~~
- *Not RiceU HPCToolkit (not available for BlueGene)

JUBL module hpct/2.0

- User code callgraph profiling
 - compile & link with `-g -pg` flags
 - `gmon.out` + `$(MPIRANK)` files produced at finalisation
 - text report produced with: ***gprof*** `<EXE> gmon.out`
 - view results with: ***Xprofiler*** `<EXE> gmon.out.*`
- MPI profiling & tracing
 - compile with `-g` flag and link with `-g -L$(HPCT_LIBRTS) -lmpitrace -llicense`
 - view profiles with: ***peekperf*** `*.viz`
 - view trace with: ***peekview*** `single_trace`
- HWC profiling [HPM]
 - requires manual source code instrumentation

HPCT/Xprofiler

Xprofiler <@jubl>
_ □ ×

File
View
Filter
Report
Utility

Call Graph Profile <@jubl>
_ □ ×

File
Help

index	%time	self	descendants	called/total called+self called/total	parents name index children
[1]	31.4	0.85	548.40	128+450048	<cycle 2 as a whole> [1]
		0.19	420.92	194048	hypr_SMGRelax <cycle 2> [2]
		0.08	85.12	10240	hypr_SMGsolve <cycle 2> [10]
		0.01	23.12	9152	hypr_SMGRelaxSetupASol <cycle 2> [12]
		0.01	14.92	1024	hypr_SMGSetup <cycle 2> [16]
		0.42	2.67	8128	hypr_SMGSetupInterp0p <cycle 2> [43]
		0.14	1.65	227584	hypr_SMGRelaxSetup <cycle 2> [60]

				16256	hypr_SMGSetupInterp0p <cycle 2> [43]
				177792	hypr_SMGsolve <cycle 2> [10]
[2]	24.0	0.19	420.92	194048	hypr_SMGRelax <cycle 2> [2]
		403.30	18.65	351168/351168	hypr_CyclicReduction [6]
		414.65	1.58	361344/444992	hypr_SMGResidual [5]
		3.62	0.03	83648/83648	hypr_SMGSetStructVectorConstantValues [59]
				194048	hypr_SMGRelaxSetup <cycle 2> [60]
				10176	hypr_SMGsolve <cycle 2> [10]

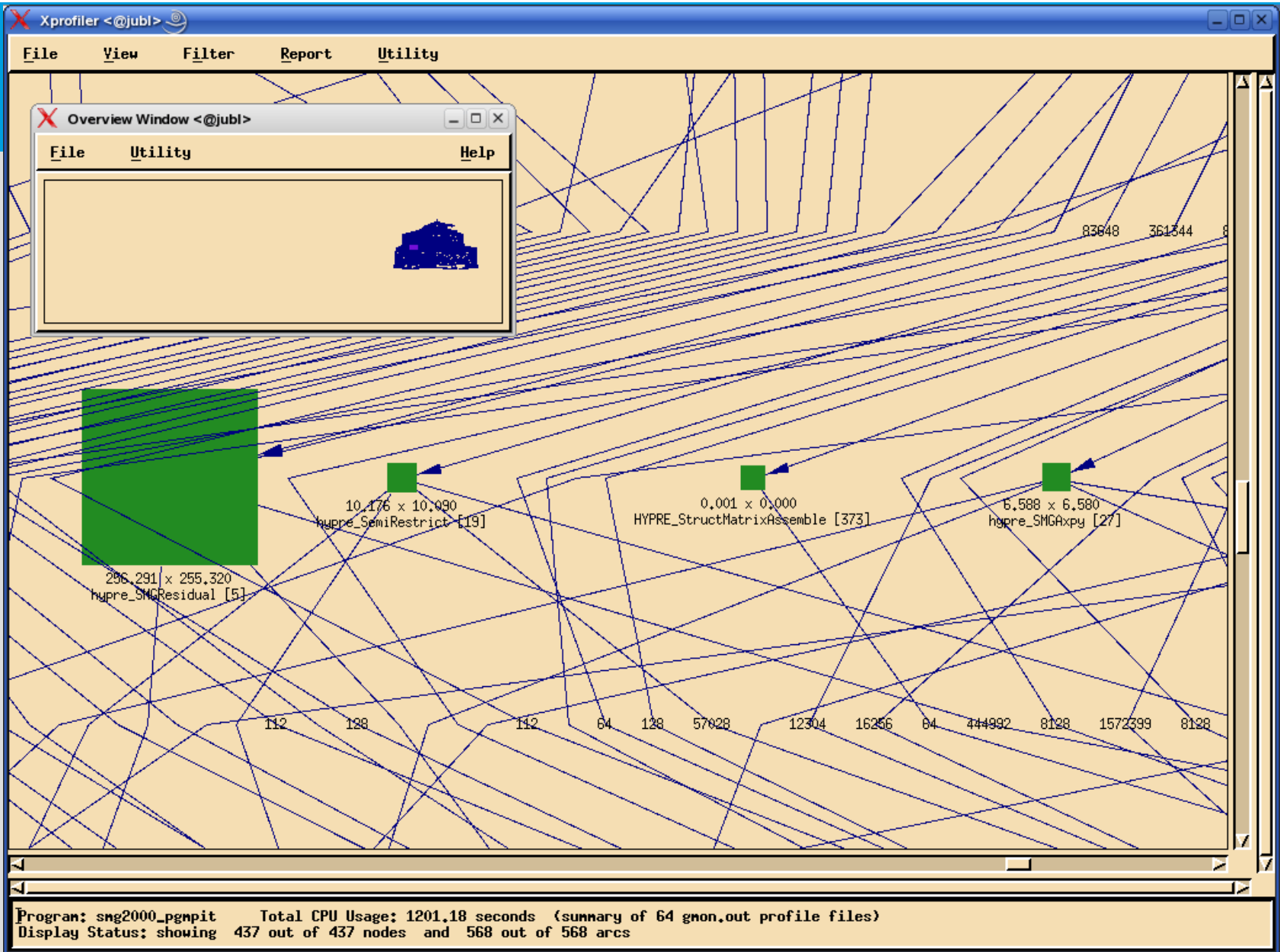
		0.71	550.75	64/64	_start_blrts [4]
[3]	31.5	0.71	550.75	64	main [3]
		0.00	274.63	64/64	HYPRE_StructSMGSetup [7]
		0.00	274.63	64/64	HYPRE_StructSMGsolve [8]
		0.00	1.27	112/112	HYPRE_StructMatrixSetBoxValues [96]
		0.00	0.22	128/128	HYPRE_StructVectorSetBoxValues [214]
		0.00	0.01	64/64	HYPRE_StructSMGDestroy [367]

Search Engine: (regular expressions supported)

Program: smg2000_pgmplt
Total CPU Usage: 1201.18 seconds (summary of 64 gnou.out profile files)

Display Status: showing 437 out of 437 nodes and 568 out of 568 arcs

7 | 2007.09.03 | b.wylie@fz-juelich.de



Main Window <@jubl>

File Manual Automatic Windows Tool

DATA VISUALIZATION WINDOW

mpidata

Label	Count	WallClock	Transferred Bytes
coarsen.c	5657	0.341085	5.09069e+06
hypre_StructCoarsen(coarsen.c)	5657	0.341085	5.09069e+06
MPI_Comm_rank_230	474	0.000182	0
MPI_Irecv_469	5657	0.003729	22628
MPI_Irecv_509	5652	0.00315	3.54707e+06
MPI_Isend_483	5657	0.010871	22628
MPI_Isend_534	5631	0.009128	5.09069e+06
MPI_Waitall_491	474	0.341085	0
MPI_Waitall_497	474	0.003336	0
MPI_Waitall_542	474	0.222408	0
MPI_Waitall_551	474	0.004619	0
communication.c	3892	0.696205	7.63512e+06
hypre_CommPkgCreateInfo(communica...	3892	0.00074	0
hypre_FinalizeCommunication(commu...	2461	0.696205	0
MPI_Waitall_667	2461	0.696205	0
hypre_InitializeCommunication(commu...	2046	0.063248	7.63512e+06
MPI_Irecv_485	2008	0.004634	7.63211e+06
MPI_Isend_492	2046	0.063248	7.63512e+06
smg2000.c	1	1.6e-05	0
main(smg2000.c)	1	1.6e-05	0
struct_grid.c	1	3.3e-05	28
SUMMARY	183510	5.9853	1.33699e+08
MPI_Allgather	1	2.2e-05	4
MPI_Allgatherv	1	3.3e-05	28
MPI_Allreduce	12	0.412137	96
MPI_Barrier	1	1.6e-05	0
MPI_Comm_rank	18384	0.003422	0
MPI_Comm_size	16086	0.003104	0
MPI_Irecv	183415	0.198877	1.22258e+08
MPI_Isend	183510	1.12246	1.33699e+08
MPI_Reduce	2	0.072587	8
MPI_Waitall	78773	5.9853	0
timing.c	1	0.018301	8

SOURCE CODE WINDOW

smg2000.c coarsen.c communication.c

```

647
648 }
649 h
650 h
651 h
652 h
653 h
654
655 re
656 }
657
658 #el
659
660 int
661 hypre_FinalizeCommunication( hypre_CommHandle *comm_handle )
662 {
663     int ierr = 0;
664
665     if (hypre_CommHandleNumRequests(comm_handle))
666     {
667         MPI_Waitall(hypre_CommHandleNumRequests(comm_handle),
668                   hypre_CommHandleRequests(comm_handle),
669                   hypre_CommHandleStatus(comm_handle));
670     }
671
672     /*-----
673     * Free up communication handle
674     *-----*/
675
676     hypre_TFree(hypre_CommHandleRequests(comm_handle));
677     hypre_TFree(hypre_CommHandleStatus(comm_handle));
678     hypre_TFree(comm_handle);
679
680     return ierr;
681 }
682
683 #endif

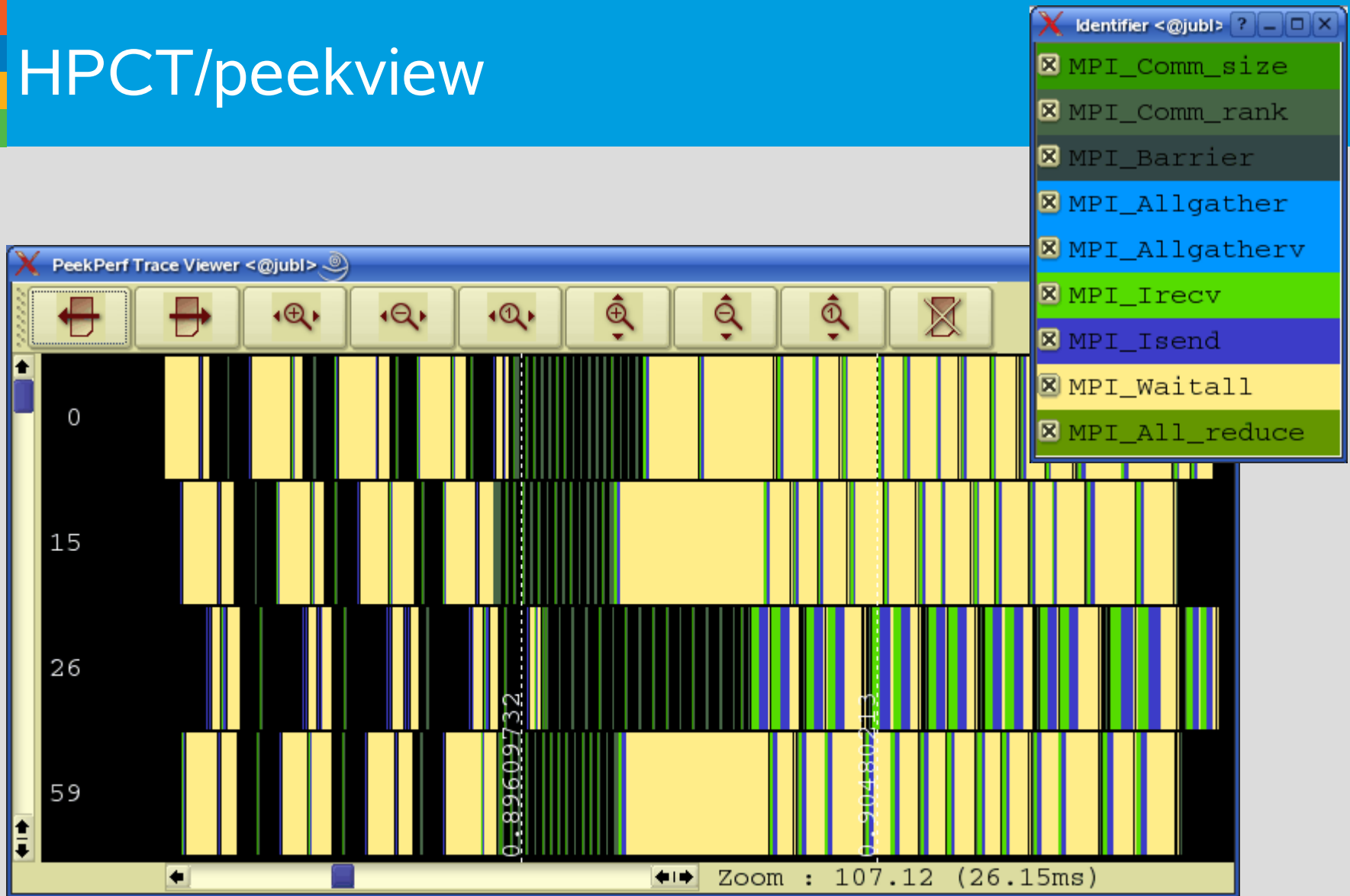
```

Metric Browser: MPI_Waitall <@jubl>

Close Metric Options Precision

Task	Count	WallClock	Transferred Bytes
0	74731	5.56967	0
15	59653	5.9853	0
26	78773	3.84792	0
59	72259	5.33363	0

HPCT/peekview



HPCT notes

- Most of the available documentation is inaccurate!
- pg
 - one gmon.out file per rank dumped in current directory
 - silently overwrites existing files
 - **Xprofiler** runs out of memory with more than 256 files
- mpitrace
 - single_trace & mpi_profile files dumped in current directory
 - silently overwrites existing files
 - only 4 ranks are traced irrespective of TRACE_ALL_RANKS
 - 0 & ranks with minimum/median/maximum communication time
 - traces are truncated when trace buffer is full
 - no way to adjust trace buffer size
 - **peekview** trace viewer doesn't show MPI messages

JUBL module mpiP/3.1

- MPI profiling library
 - open source from LLNL: [//sourceforge.net/projects/mpip](http://sourceforge.net/projects/mpip)
 - supports a wide range of MPI libraries/systems
- Usage
 - relink application executable: `-L$ (MPIP_LIBRTS) -lmpiP`
 - compile/link with `-g` flag for source-level information
 - measurement configurable with MPIP environment variable
 - spaces in BG/L environment variables are problematic!
 - plain text report written on finalisation to `<EXE>.<NPROCS>.<PID>.<SEQ>.mpiP`
 - postprocess report for source-level information
 - ***mpip-insert-src*** `<EXE> <EXE>.<NPROCS>.0.1.mpiP`
 - reports can be imported by TAU ***paraprof*** GUI?

mpiP textual report extract

@mpiP

@--- MPI Time (seconds) -----

Task	AppTime	MPITime	MPI%
0	20	9.87	49.26
1	20	9.34	46.61
*	1.28e+03	596	46.48

@--- Callsites: 867 -----

ID	Lev	File/Address	Line	Parent_Funct	MPI_Call
457	0	communication.c	667	hypr_FinalizeCommunication	Waitall
457	1	computation.c	402	hypr_FinalizeIndtComputations	
457	2	semi_interp.c	250	hypr_SemiInterp	
457	3	smg_solve.c	263	hypr_SMGSolve	
457	4	HYPRE_struct_smg.c	64	HYPRE_StructSMGSolve	
457	5	smg2000.c	575	main	
457	6	start.c	107	_start_blrts	

@--- Aggregate Time (top twenty, descending, milliseconds) -----

Call	Site	Time	App%	MPI%	COV
Waitall	157	3.53e+04	2.76	5.93	0.18
Waitall	457	3.34e+04	2.60	5.59	0.85
Waitall	467	3.04e+04	2.37	5.10	0.33
Allreduce	40	2.06e+04	1.60	3.45	0.79

@--- Aggregate Sent Message Size (top twenty, descending, bytes) -----

Call	Site	Count	Total	Avrg	Sent%
Isend	530	1219176	8.18e+08	671	15.09

mpiP notes

- Report includes every MPI process
 - and aggregates (*) for all together
- Default records only caller of MPI routines
 - often insufficient to determine attribution
 - communication wrapper routines obscure attribution
- -kN records N call ancestors of MPI routines
- Creation of final report can fail due to insufficient memory
 - even with “low memory” option (default on JUBL)
 - depends on complexity of application (and -k depth)
- **paraprof** doesn't parse mpiP reports properly

JUBL module tau/2.16

- Portable profiling & tracing toolkit
 - open source from U. Oregon: [//cs.uoregon.edu/research/tau/](http://cs.uoregon.edu/research/tau/)
 - highly configurable capabilities (not just for MPI)
 - supports a wide range of systems
 - most functionality (including tracing) untested on JUBL
- Basic usage
 - Modify Makefile to: `include $(TAU_MAKEFILE)`
 - set `$TAU_MAKEFILE` according to desired measurement
 - Prefix compile/link commands with `$(TAU_COMPILER)` to build instrumented executable(s)
 - View results with ***paraprof*** GUI
 - also partially works for viewing mpiP profiles
 - and might also work with gprof & cube files

Advanced TAU configuration

- TAU_MAKEFILE specifies instrumentation configuration
 - chose from pre-installed configs in \$TAU_MF_DIR
 - JUBL default includes MPI+callgraph profiling
- TAU_OPTIONS for TAU_COMPILER configuration
 - -optVerbose, -optKeepFiles, -optPreProcess,...
- -multiplecounters for hardware counter metrics
 - COUNTER1=GET_TIME_OF_DAY
 - COUNTER2=PAPI_NATIVE_PM_CYC
 - COUNTER3=PAPI_NATIVE_PM_INST_CMPL
- TAU_THROTTLE=1 for measurement throttling
 - alternatively create selective instrumentation file
 - rebuild with -optTauSelectFile=throttle.tau

TAU/ParaProf

About ParaProf <@jubl>



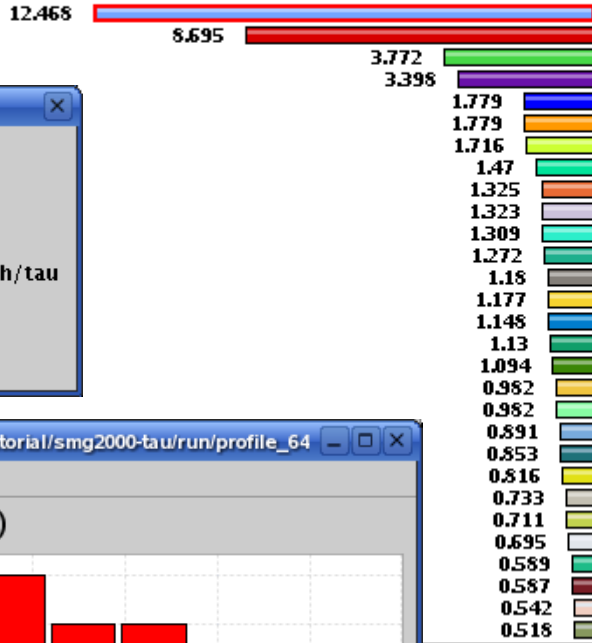
ParaProf 3
 Fri Nov 10 14:03:51 PST 2006
 JVM Heap Size: 118626kb
<http://www.cs.uoregon.edu/research/tau>

OK

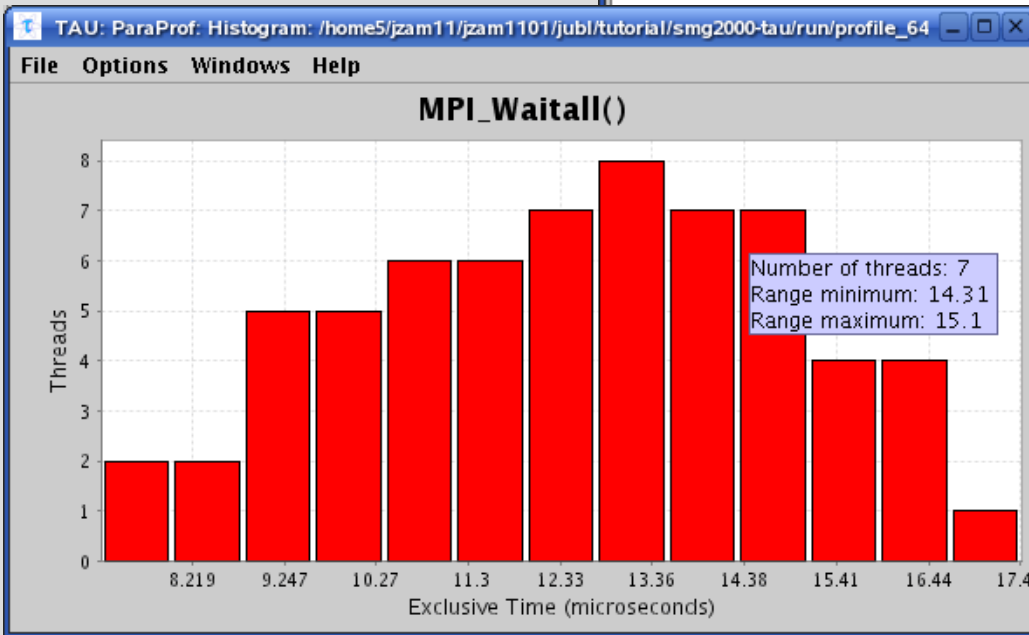
TAU: ParaProf: Mean Data - /home5/jzam11/jzam1101/jubl/tutorial/smg2000-tau/run/profile_64

File Options Windows Help

Metric: Time
 Value: Exclusive
 Units: microseconds



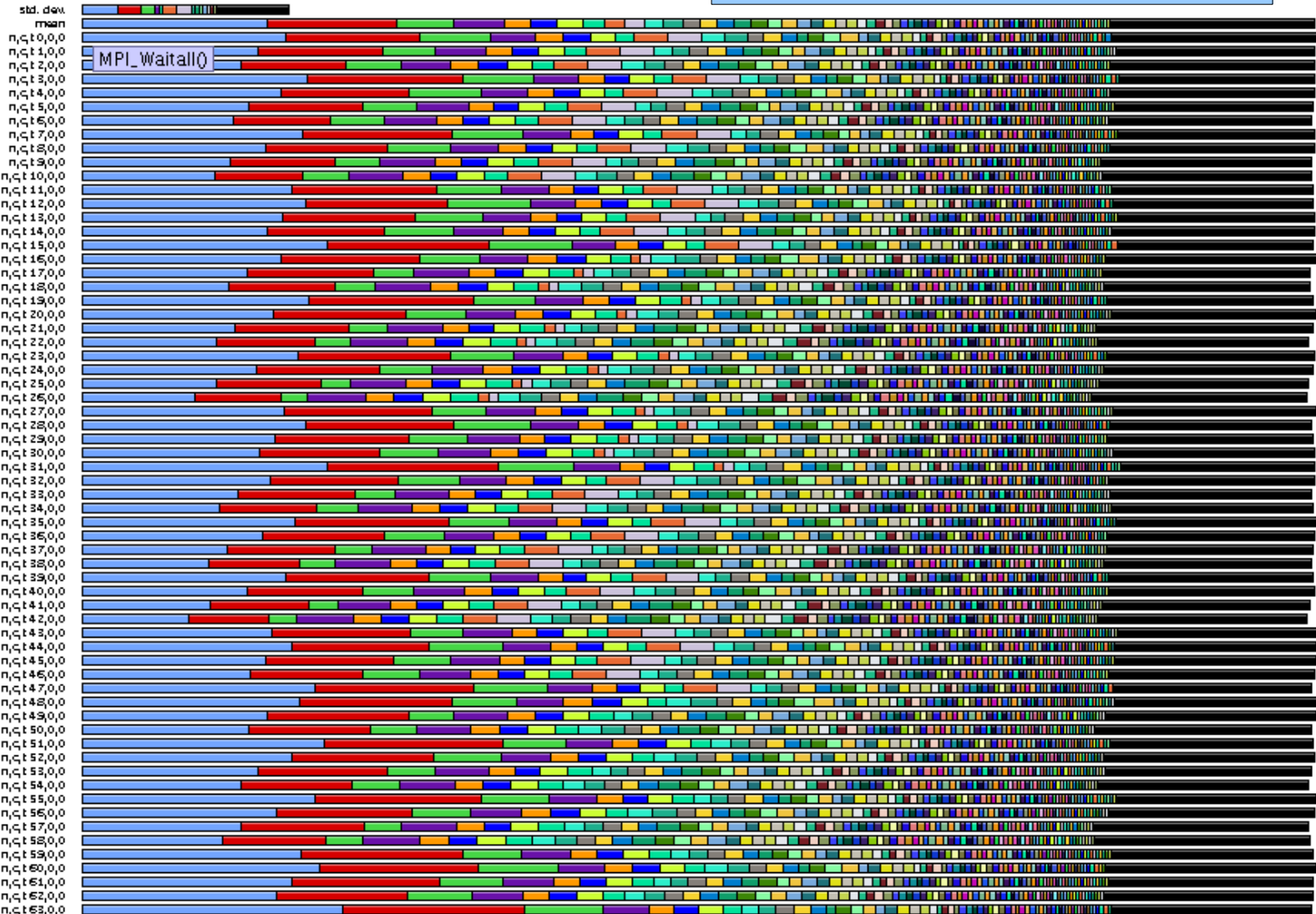
Value (microseconds)	Function
12.468	MPI_Waitall()
8.695	int hypr_FinalizeCommunication@hypr_CommHandle ^
3.772	int hypr_StructCoarsen@hypr_StructGrid ^, int ^, int ^, i
3.398	void hypr_Free(char ^) C [{memory.c} {173,1}-{186,1}]
1.779	int hypr_CyclicReduction(void ^, hypr_StructMatrix ^, h
1.779	int hypr_SMGRelax(void ^, hypr_StructMatrix ^, hypr_S
1.716	char ^hypr_CAlloc(int, int) C [{memory.c} {101,1}-{131,1}
1.47	char ^hypr_MAlloc(int) C [{memory.c} {67,1}-{95,1}]
1.325	MPI_Allreduce()
1.323	int hypr_PrintTiming(char ^, MPI_Comm) C [{timing.c} {
1.309	double ^hypr_StructMatrixExtractPointerByIndex@hypr.
1.272	int hypr_CyclicReduction(void ^, hypr_StructMatrix ^, h
1.18	int hypr_FinalizeCommunication@hypr_CommHandle ^
1.177	int hypr_SMGResidual(void ^, hypr_StructMatrix ^, hypr
1.148	int hypr_InitializeCommunication@hypr_CommPkg ^, d
1.13	MPI_Isend()
1.094	int hypr_InitializeIndtComputations@hypr_ComputePk
0.982	int hypr_StructStencilElementRank@hypr_StructStencil
0.982	double ^hypr_StructMatrixExtractPointerByIndex@hypr.
0.891	int hypr_InitializeCommunication@hypr_CommPkg ^, d
0.853	int hypr_FinalizeCommunication@hypr_CommHandle ^
0.816	int hypr_FinalizeIndtComputations@hypr_CommHandle
0.733	int hypr_InitializeCommunication@hypr_CommPkg ^, d
0.711	int hypr_SMGRelax(void ^, hypr_StructMatrix ^, hypr_S
0.695	int hypr_StructCoarsen@hypr_StructGrid ^, int ^, int ^, i
0.589	int hypr_CyclicReductionSetup(void ^, hypr_StructMatr
0.587	int hypr_StructMapFineToCoarse(int ^, int ^, int ^)
0.542	int hypr_BoxGetStrideSize@hypr_Box ^, int ^, int ^) C [{b
0.518	MPI_Irecv()

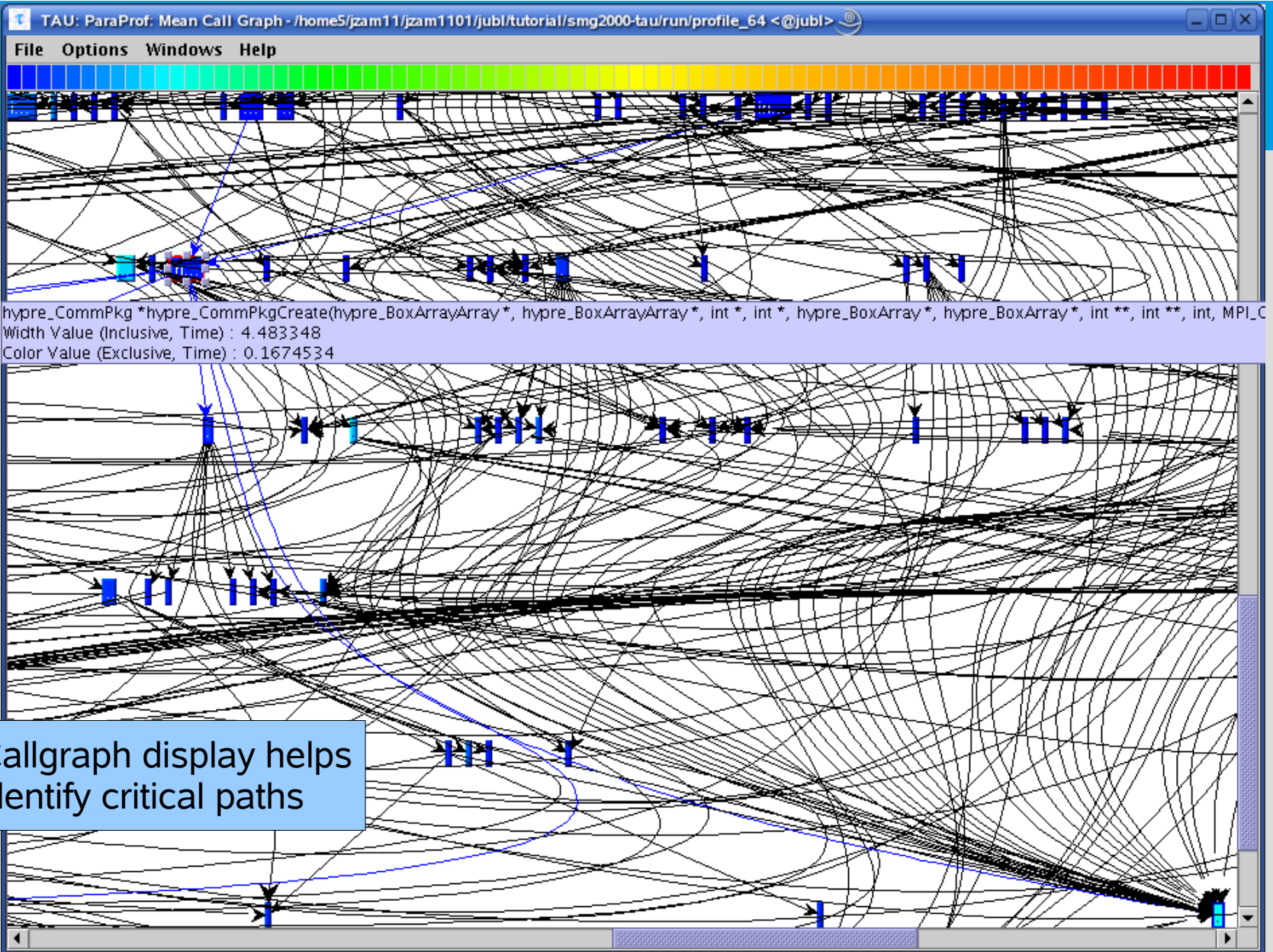


Linked graphical displays show breakdowns of time by function and process

Main display shows imbalance

Metric: Time
Value: Exclusive





Callgraph display helps identify critical paths

TAU profile notes

- One profile per MPI rank dumped in working directory
 - silently overwrites existing files!
- Measurement overhead/dilation is considerable
- TAU_THROTTLE environment variable automatically disables measurement of frequently called routines
 - makes measurement overhead/dilation more reasonable
 - but doesn't distinguish MPI routines!

JUBL module scalasca/0.9

- Automatic scalable performance analysis toolset
 - open source from FZ Jülich: [//scalasca.org/](http://scalasca.org/)
 - based on pioneering KOJAK project, focussed on MPI
 - tested on a range of HPC systems at large scales [20k+]
- Usage
 - prefix compile/link commands with **skin** instrumenter
 - prefix execution command with **scan** collector & analyzer
 - default settings are for runtime summarisation
 - extensible with hardware counter group
 - use **-t** flag to activate trace collection & analysis
 - provides additional wait & imbalance metrics
 - produces analysis reports in CUBE3 format
 - view results with **square** analysis report explorer

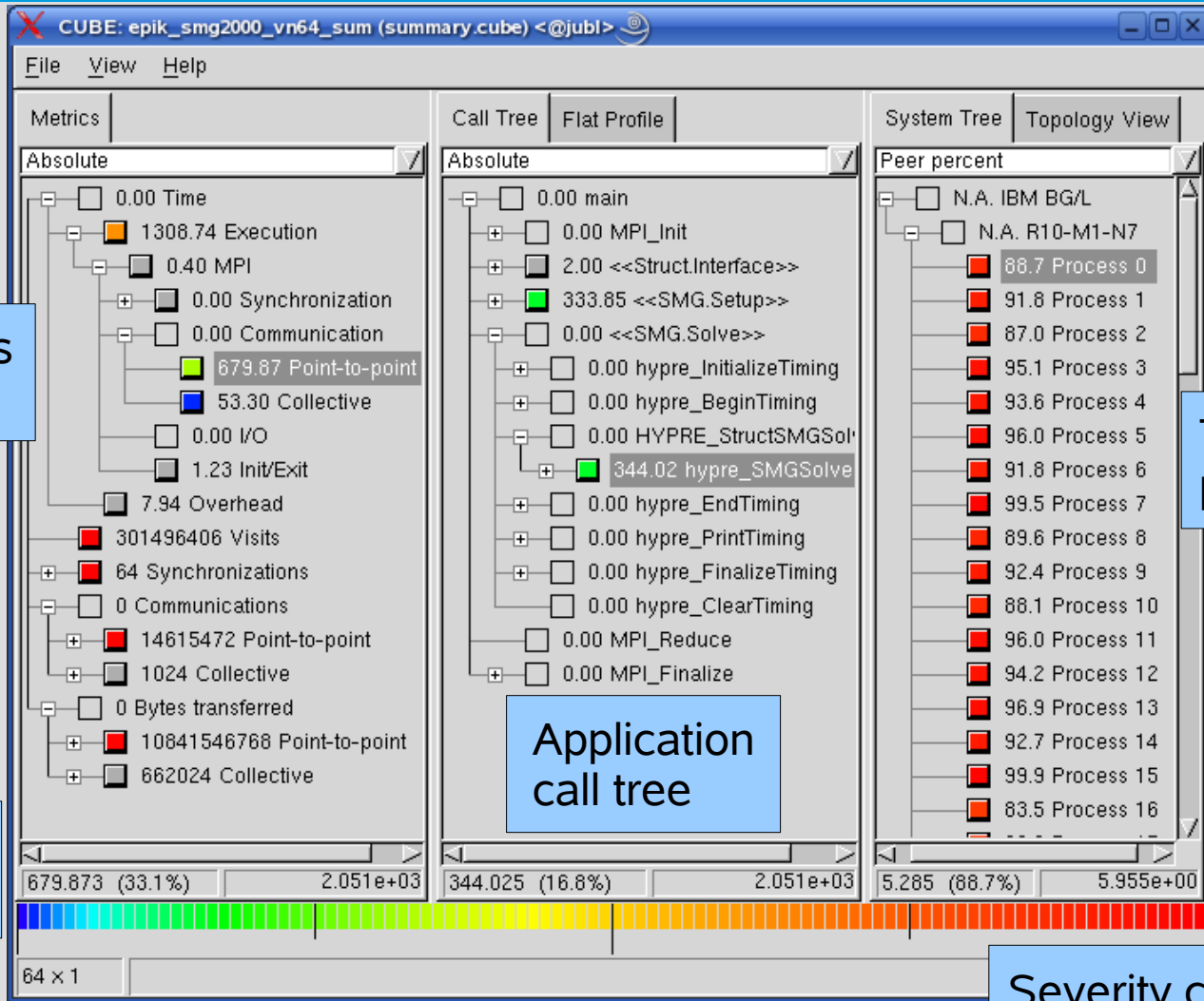
Scan summary output extract

```
% scan mpirun -np 64 -mode vn -exe smg2000 ...
S=C=A=N: ./epik_smg2000_vn64_sum collection & analysis
S=C=A=N: Collect start
mpirun -np 64 -mode vn -exe smg2000 ...
[00000]EPIK: Created archive ./epik_smg2000_vn64_sum
  Running with these driver parameters:
    (nx, ny, nz) = (64, 64, 32)
  ...
  Iterations = 5
  Final Relative Residual Norm = 3.089109e-05
[00000]EPIK: Closing experiment ./epik_smg2000_vn64_sum
[00000]EPIK: 3053 unique paths
...
[00000]EPIK: Closed experiment ./epik_smg2000_vn64_sum
S=C=A=N: Collect done
S=C=A=N: Skipping trace analysis
S=C=A=N: ./epik_smg2000_vn64_sum complete
```

Scan trace collection&analysis output extract

```
% scan -t mpirun -np 64 -mode vn -exe smg2000 ...
S=C=A=N: ./epik_smg2000_vn64_trace collection & analysis
S=C=A=N: Collect start
mpirun -np 64 -mode vn -exe smg2000 ...
[00000]EPIK: Created new archive ./epik_smg2000_vn64_trace
[00000]EPIK: Activated ./epik_smg2000_vn64_trace [75000000 B]
  Running with these driver parameters:
  ...
  Final Relative Residual Norm = 3.089109e-05
[00000]EPIK: Closing experiment ./epik_smg2000_vn64_trace
[00000]EPIK: Closed experiment ./epik_smg2000_vn64_trace
S=C=A=N: Collect done
S=C=A=N: Analyze start
mpirun -np 64 -mode vn scout ./epik_smg2000_vn64_trace
Analyzing experiment archive ./epik_smg2000_vn64_trace
S=C=A=N: Analyze done
S=C=A=N: ./epik_smg2000_vn64_trace complete
```

Scalasca summary



Hierarchies of metrics

Tree of MPI processes

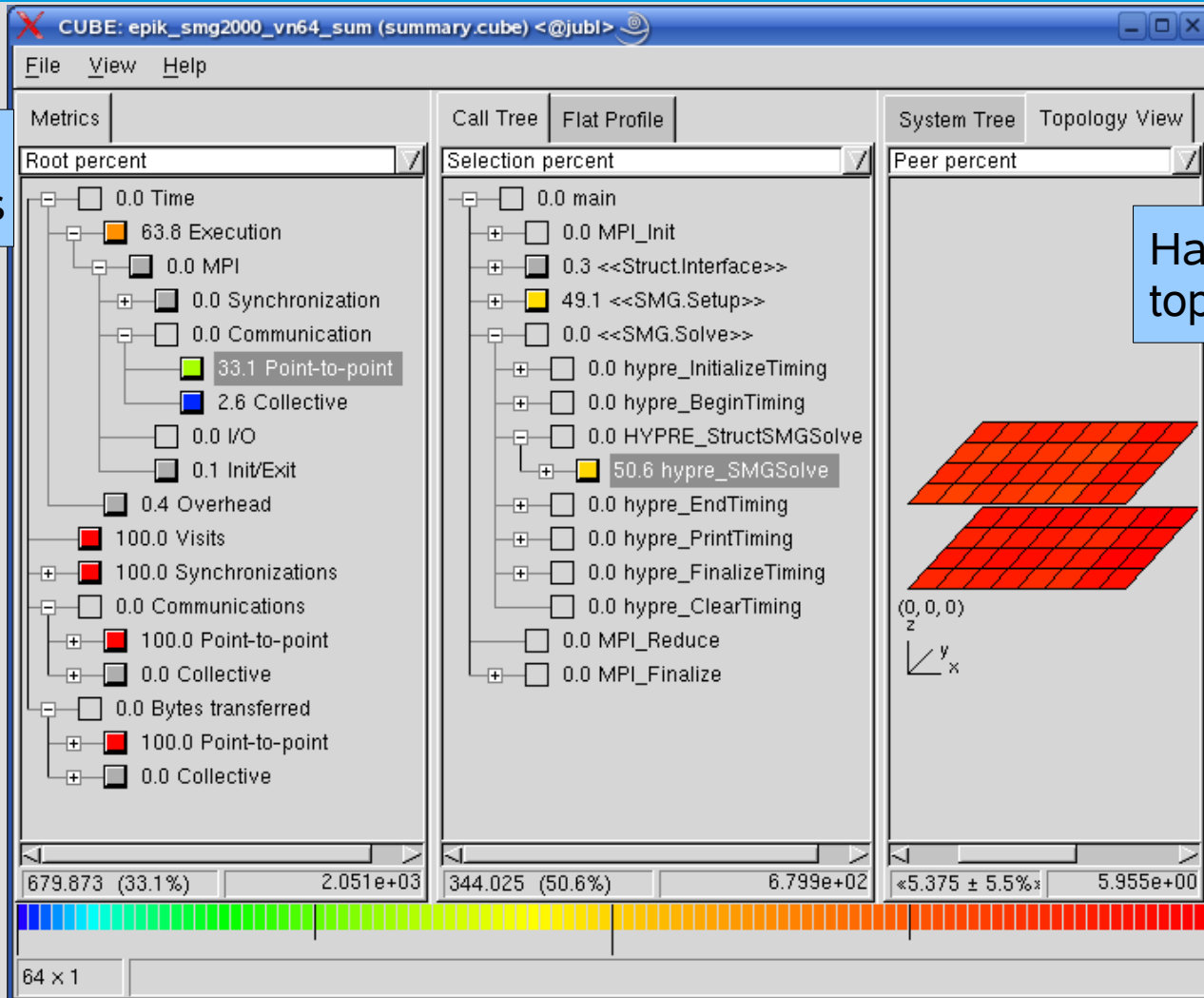
Application call tree

Selected values

Severity colour scale

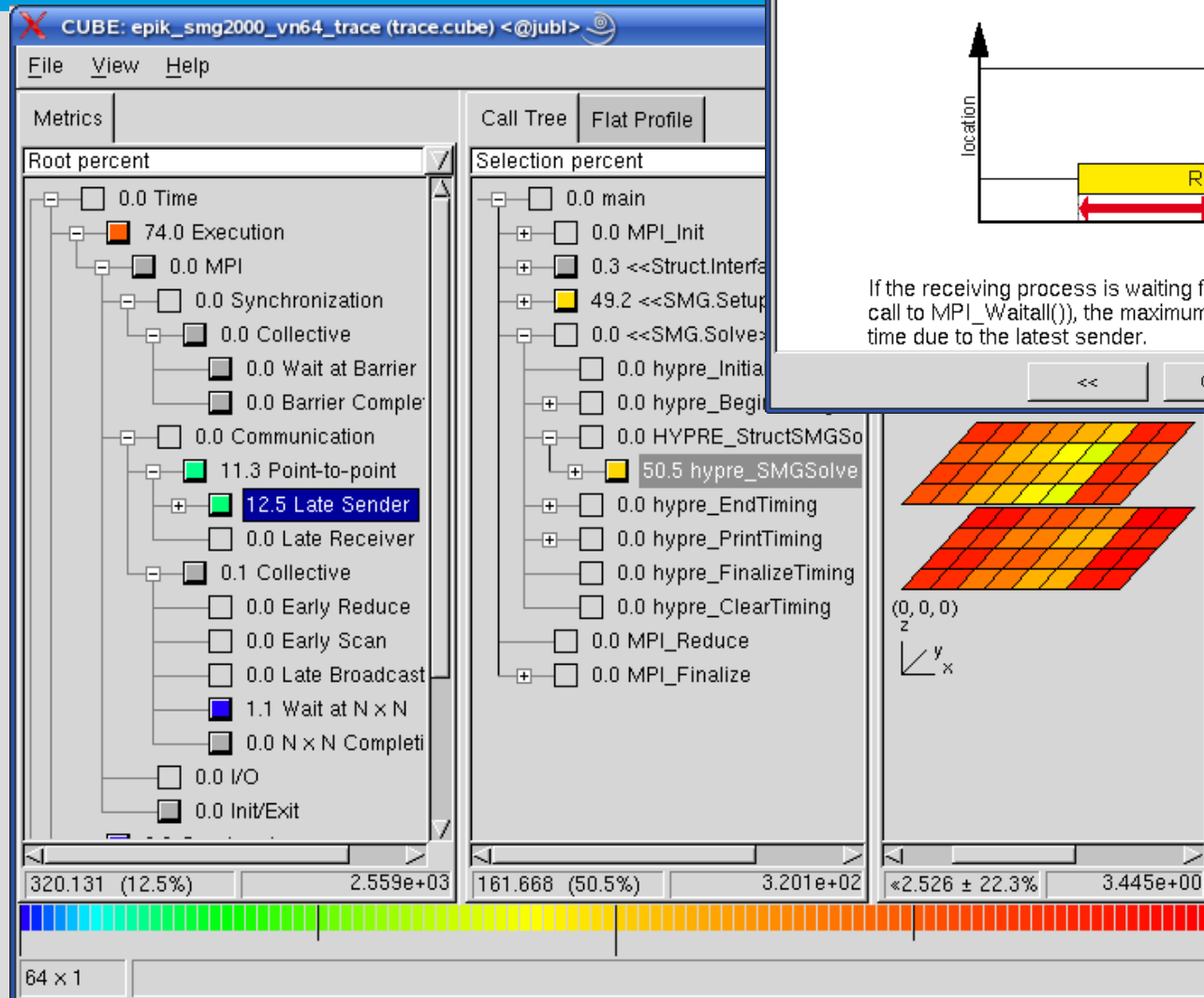
Scalasca summary

Alternative value modes



Hardware topology

Scalasca trace: Late Sender time



Late Sender Time

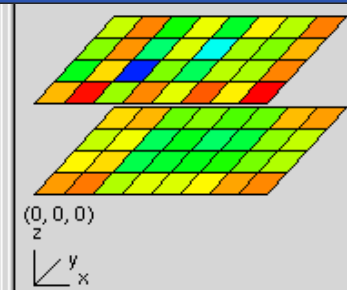
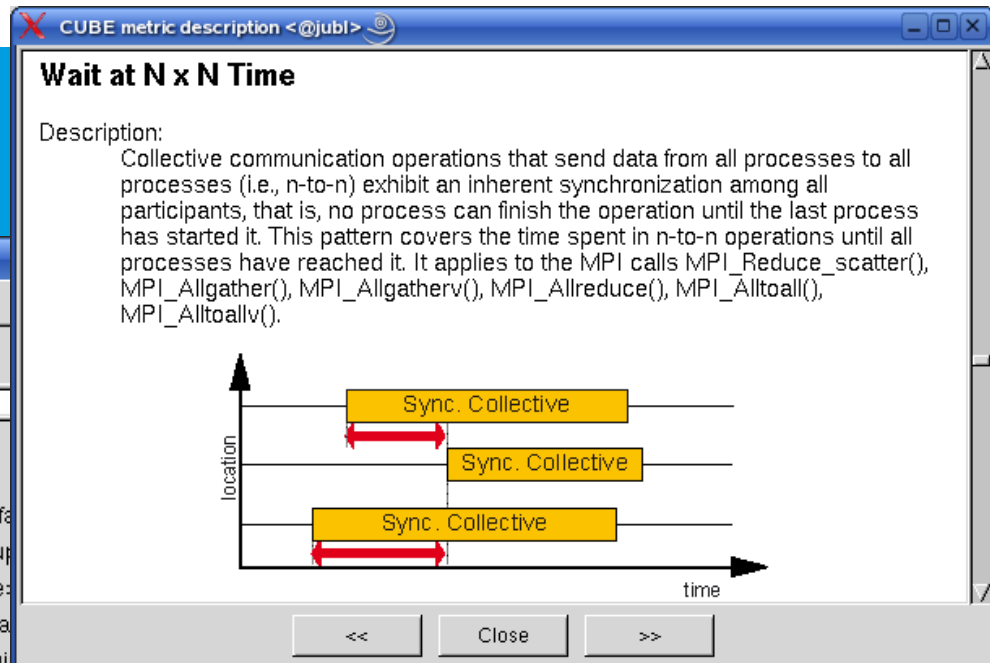
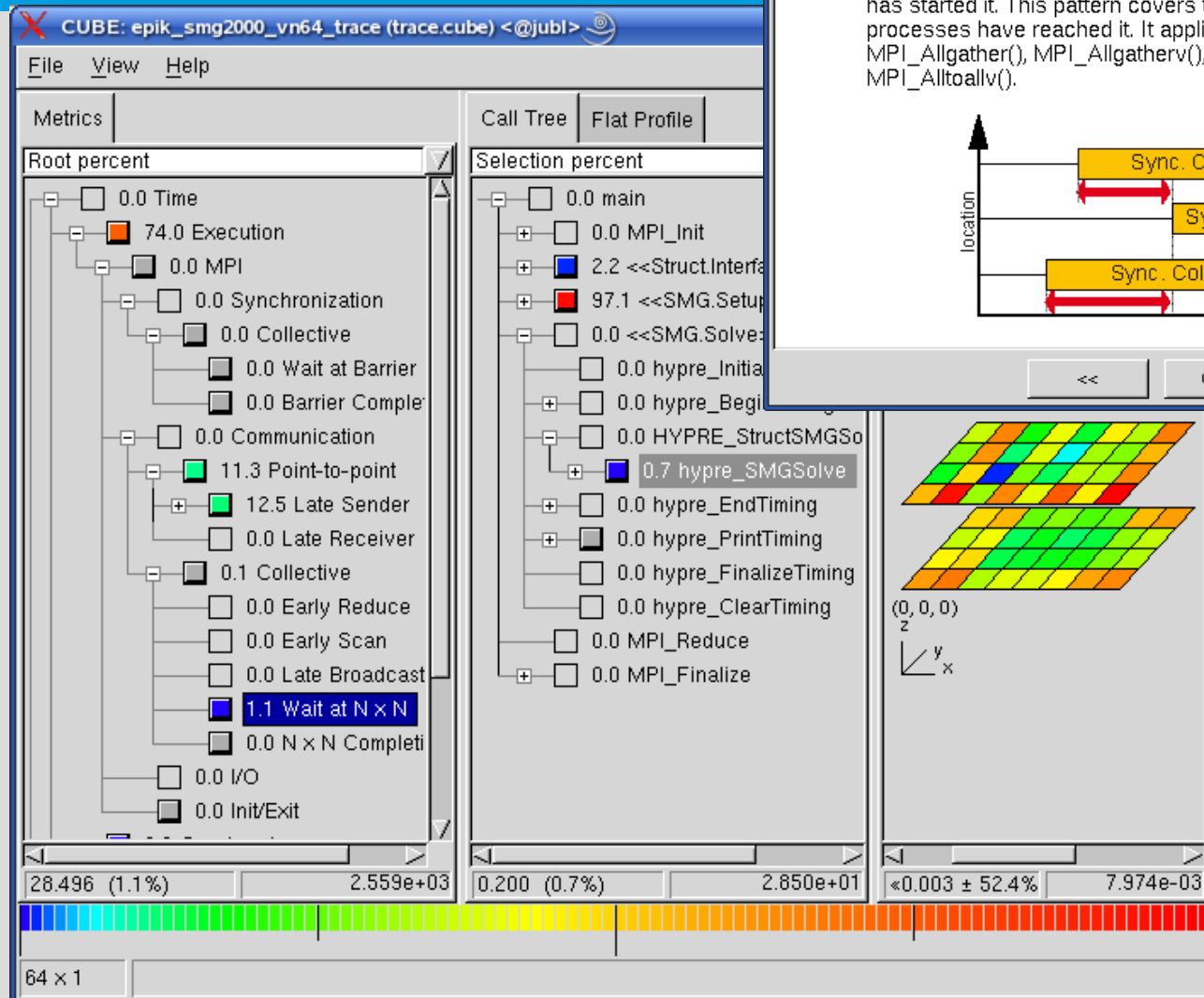
Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g., MPI_Recv() or MPI_Wait()) that is posted earlier than the corresponding send operation.

If the receiving process is waiting for multiple messages to arrive (e.g., in an call to MPI_Waitall()), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

<< Close >>

Additional metrics only available from trace analysis: right-click on metric for on-line help

Scalasca trace: Wait at N x N time



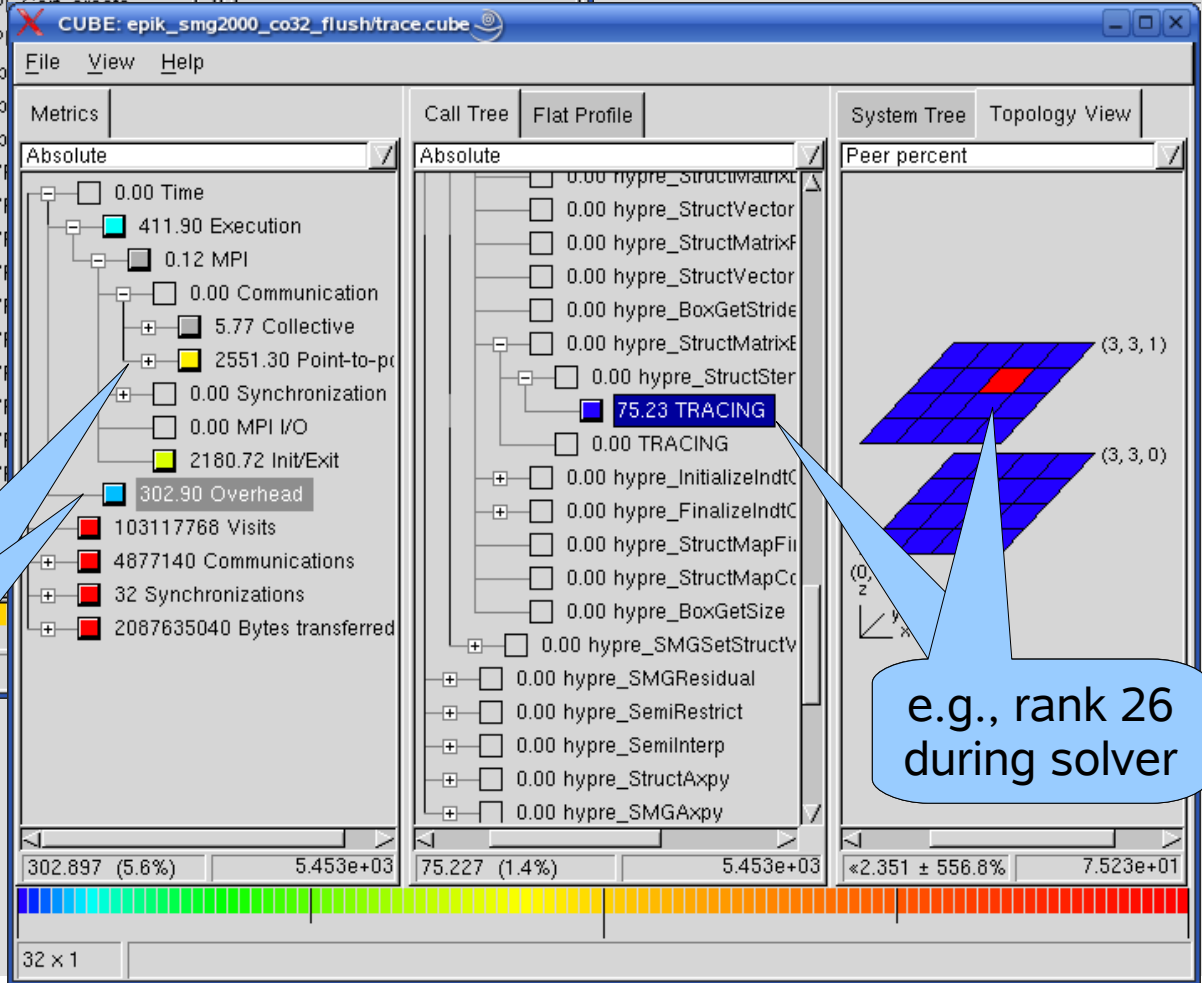
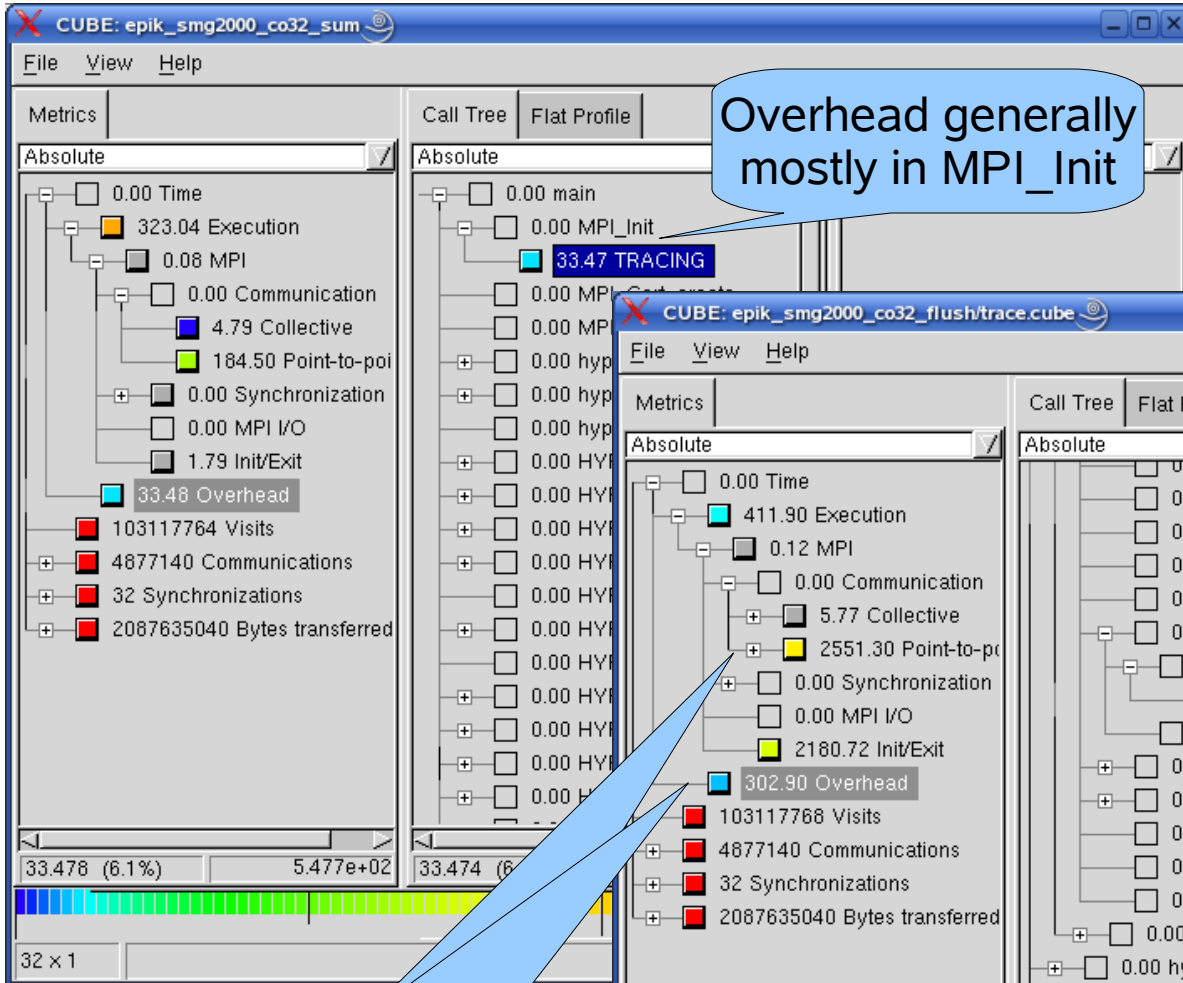
Additional metrics only available from trace analysis

Scan trace collection&analysis output extract

```
S=C=A=N: ./epik_smg2000_co32_trace collection & analysis
[00000]EPIK: Created new archive ./epik_smg2000_co32_trace
[00000]EPIK: Activated ./epik_smg2000_co32_trace [90000000 B]
  Running with these driver parameters:
    (nx, ny, nz) = (32, 32, 32)
  ...
[00026]EPIK: Flushed 89999777 bytes
  Iterations = 5
  Final Relative Residual Norm = 3.089109e-05
[00000]EPIK: Closing experiment ./epik_smg2000_co32_trace
[00000]EPIK: Flushed 77477352 bytes
[00000]EPIK: 3057 unique paths
[00026]EPIK: Flushed 5576022 bytes
  ...
[00000]EPIK: Closed experiment ./epik_smg2000_co32_trace
S=C=A=N: Collect done
  ...
```

Trace flush overhead

Overhead generally mostly in MPI_Init

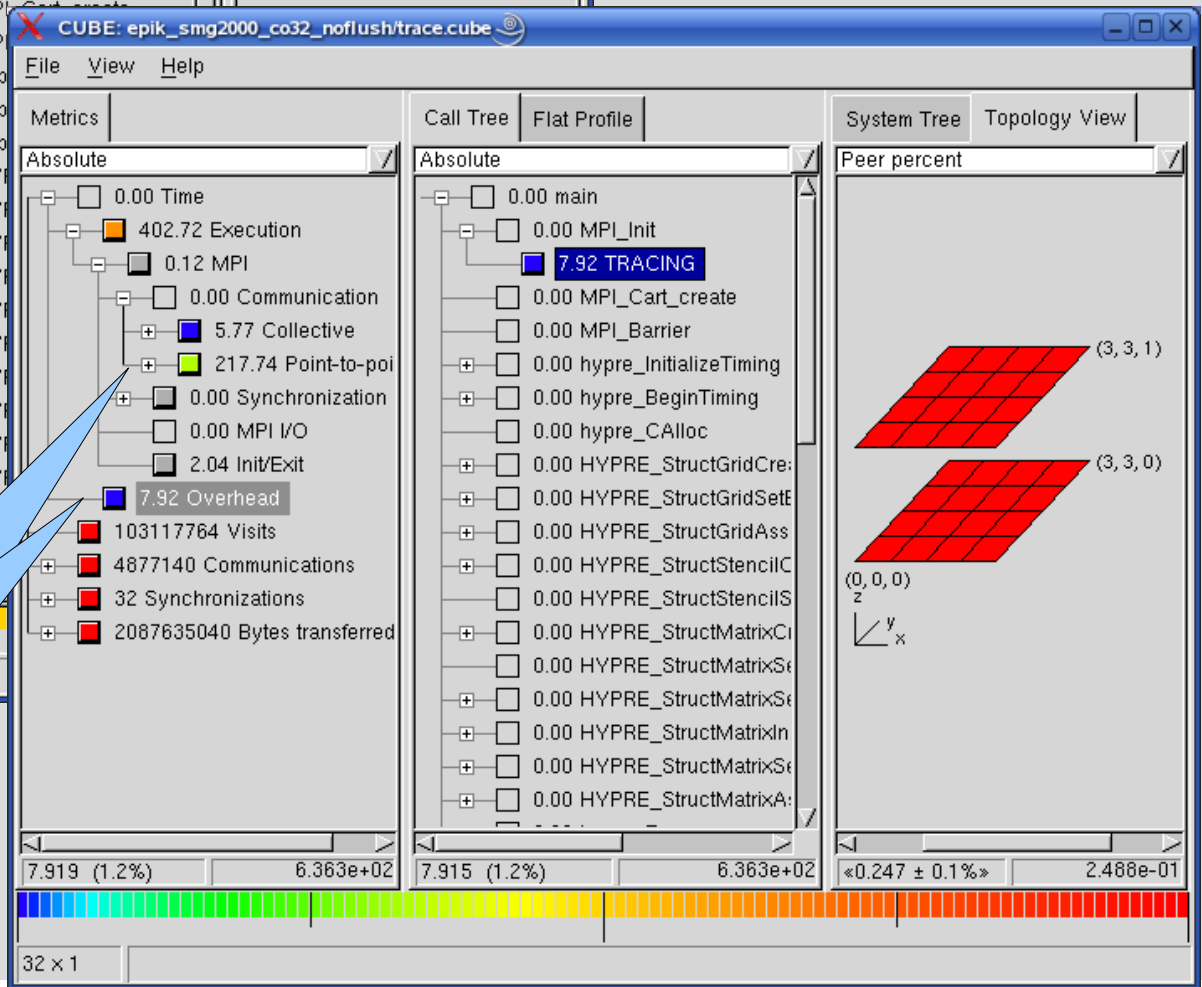
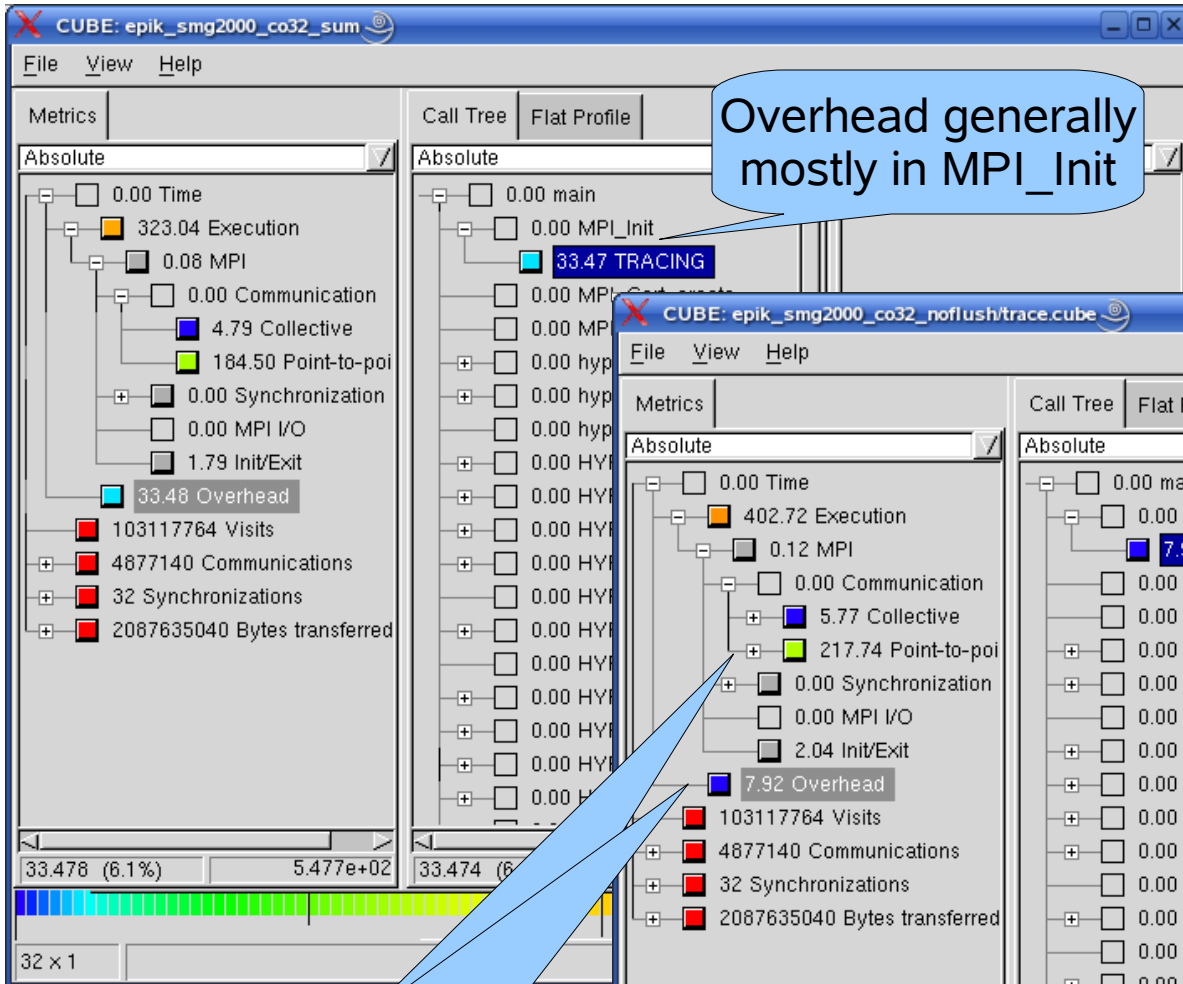


Much larger overhead of intermediate trace flushes, communication distortion

e.g., rank 26 during solver

Trace flush avoided

Overhead generally mostly in MPI_Init



With larger trace buffers to avoid intermediate flush get more accurate trace

Selective instrumentation/measurement

- Unimportant functions can be determined from summary analysis report
 - form leaves of callpath-tree (w/o MPI)
 - negligible proportion of (exclusive) execution time
 - high proportion of (exclusive) visit count
- Eliminating them reduces overheads
 - runtime processing, storage & analysis
- Makes them “invisible” in the analysis
 - logically become part of their calling functions (as if they were in-lined!)

cube3_score with (sorted) region breakdown

```
% cube3_score -r epik_smg2000_mano_64/summary.cube
```

bl	type	max_tbc	time	%	region
...					
	MPI	2061936	293.30	23.89	MPI_Waitall
	COM	2346840	14.79	1.20	hypr_FinalizeCommunication
	COM	2346840	23.49	1.91	hypr_InitializeCommunication
	MPI	7495240	11.38	0.93	MPI_Irecv
	MPI	8149850	41.43	3.37	MPI_Isend
	USR	9426048	10.47	0.85	hypr_StructStencilElementRank
	USR	9426048	21.69	1.77	hypr_StructMatrixExtractPointerByIdx
	USR	11063016	16.80	1.37	hypr_MAlloc\$AF10_5
	USR	11454432	25.82	2.10	hypr_MAlloc
	USR	11763336	26.90	2.19	hypr_CAlloc
	USR	23496576	38.16	3.11	hypr_Free
	ANY	<u>162589938</u>	1227.61	100.00	ALL (254 regions)
	MPI	17649090	456.64	37.20	+ MPI (13 regions) pure MPI
	COM	9905832	321.80	26.21	+ COM (32 regions) combined MPI&USR
	USR	135034968	311.13	25.34	+ USR (207 regions) pure User

max_tbc = estimated maximum trace buffer requirement (bytes per process)
to store all events that would be generated in an identical trace

cube3_score with trial region blacklist

```
% cube3_score -r -l smg2000.bl epik_smg2000_mano_64/summary.cube
bl type      max_tbc      time          % region
...
  MPI      2061936    293.30      23.89  MPI_Waitall
  COM      2346840     14.79       1.20  hypr_FinalizeCommunication
  COM      2346840     23.49       1.91  hypr_InitializeCommunication
  MPI      7495240     11.38       0.93  MPI_Irecv
  MPI      8149850     41.43       3.37  MPI_Isend
*  USR      9426048     10.47       0.85  hypr_StructStencilElementRank
*  USR      9426048     21.69       1.77  hypr_StructMatrixExtractPointerByIdx
*  USR     11063016     16.80       1.37  hypr_MAlloc$AF10_5
*  USR     11454432     25.82       2.10  hypr_MAlloc
*  USR     11763336     26.90       2.19  hypr_CAlloc
*  USR     23496576     38.16       3.11  hypr_Free

  ANY     162589938   1227.61    100.00  ALL      (253 regions)
  MPI      17649090     456.64     37.20  + MPI ( 13 regions) pure MPI
  COM       9905832     321.80     26.21  + COM ( 32 regions) combined MPI&USR
  USR     135034968     311.13     25.34  + USR (207 regions) pure User

  BL      103570824    182.11     14.83  BL ( 9 regions) blacklisted
  -BL     59019114    1045.50     85.17  ALL-BL (244 regions) remainder
```

Scalasca notes

- Scalasca trace analyzer (**scout**) is unable to load traces larger than around 75/140MB (VN/CO) for largest rank
- Identify trace buffer requirements from summary report
 - **cube3_score <epik>/summary.cube**
 - Use **-r** flag for breakdown of requirements per region and investigate benefits of blacklisting with **-l<file>**
- Use **ELG_BUFFER_SIZE** to specify rank trace buffer sizes
- Blacklisting routines skips handling measurements for them for faster and more accurate measurement & analysis
 - reduces size of traces (and trace buffer requirements)
 - reduces measurement overhead (summary & trace)
 - effectively “inlines” regions from measurement perspective
 - file of routines specified with **ELG_BLACKLIST**

Hardware performance counters (metrics)

- Provided by most modern microprocessors
 - low-overhead measurement of low-level execution detail
 - offer insight into efficiency of execution units, caches, etc.
 - often restrictions on groupings of counters
 - documentation varies by manufacturer
- PAPI provides portable access to available counters
- Available counters (and their interpretation) are platform/processor-specific
 - especially on BlueGene!
- Usefulness of BlueGene counters for application execution analysis & tuning still unclear

JUBL module papi/3.5.0

- Portable API to access hardware performance counters
 - open source from UTK: [//icl.cs.utk.edu/papi/](http://icl.cs.utk.edu/papi/)
 - supports a wide range of systems
 - defines presets for common events, as well as native events
 - used by various performance tools, e.g., TAU & Scalasca
- Usage
 - Manual instrumentation with PAPI library calls
 - Link with: `-L$(PAPI_LIBRTS) -lpapi -lbgl_perfctr.rts`
 - Determine available presets and native counters
 - `mpirun -np 1 $PAPI_BINRTS/papi_avail -a`
 - `mpirun -np 1 $PAPI_BINRTS/papi_native_avail`

BG/L PAPI preset events

Available preset events and hardware information.

```
-----  
Model string and code      : PVR=0x5202:0x1891   Serial=R00-M0-NA-C:J16-U01  
CPU Revision              : 20994.062500  
CPU Megahertz             : 700.000000  
Number Hardware Counters  : 52  
Max Multiplex Counters    : 32  
-----
```

```
-----  
Name                      Code          Deriv  Description  
PAPI_L3_TCM               0x80000008  No    Level 3 cache misses  
PAPI_L3_LDM               0x8000000e  Yes   Level 3 load misses  
PAPI_L3_STM               0x8000000f  No    Level 3 store misses  
PAPI_FMA_INS              0x80000030  No    FMA instructions completed  
PAPI_TOT_CYC              0x8000003b  No    Total cycles  
PAPI_L2_DCH               0x8000003f  Yes   Level 2 data cache hits  
PAPI_L2_DCA               0x80000041  Yes   Level 2 data cache accesses  
PAPI_L3_TCH               0x80000057  No    Level 3 total cache hits  
PAPI_FML_INS              0x80000061  No    Floating point multiply instructions  
PAPI_FAD_INS              0x80000062  No    Floating point add instructions  
PAPI_BGL_OED              0x80000067  No    BGL Oedipus operations  
PAPI_BGL_TS_32B           0x80000068  Yes   BGL Torus 32B chunks sent  
PAPI_BGL_TS_FULLL        0x80000069  Yes   BGL Torus no token UPC cycles  
PAPI_BGL_TR_DPKT          0x8000006a  Yes   BGL Tree256 byte packets  
PAPI_BGL_TR_FULLL        0x8000006b  Yes   BGL UPC cycles (CLOCKx2) tree rcv is full  
-----
```

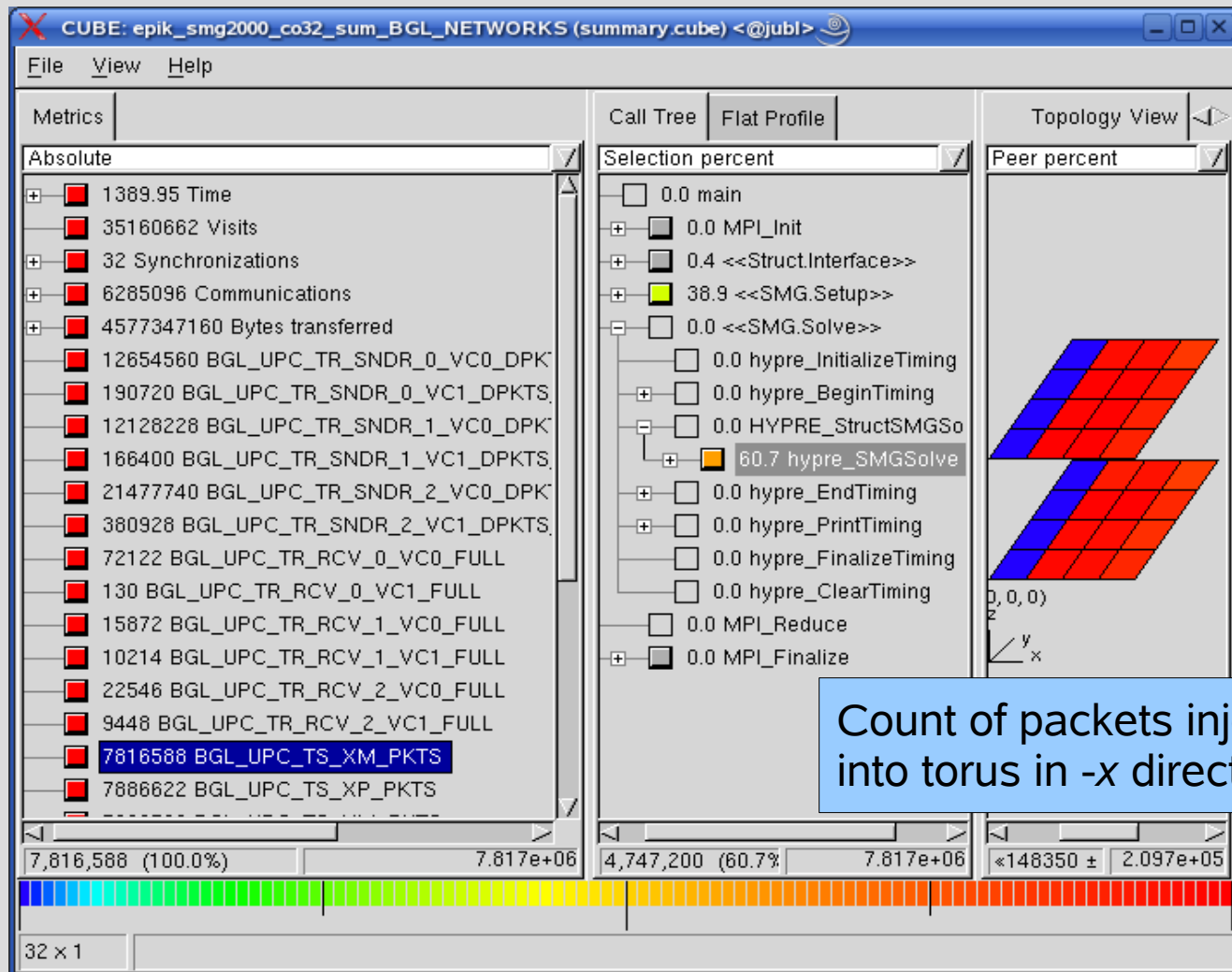
BG/L PAPI native events

- Output logged in `$PAPI_ROOT/bgl_papi_native_avail.out` lists over 328 (undocumented?) BGL-specific events
 - 4+4 FPU & 2NDFPU arithmetic unit events
 - 4+4 FPU & 2NDFPU load/store unit events
 - 29+29 UPC_PU0 & UPC_PU1 events
 - 34 UPC_L3 events for Level-3 cache
 - 74 UPC_TS events for torus network
 - 146 UPC_TR events for tree network
 - No events for PowerPC cores!
- Multiplexing restricts valid counter combinations
- Counters shared by both cores: use CO-mode
- Create your own definitions for “FLOPS” count!

Scalasca example/METRICS.SPEC_BGL

- Predefined groups of counters
 - BGL_NETWORKS
 - BGL_TORUS
 - BGL_TORUS_PKTS BGL_TORUS_FULL
 - BGL_TORUS_ALLX BGL_TORUS_ALLY BGL_TORUS_ALLZ
 - BGL_TREE
 - BGL_TREE_PKTS BGL_TREE_FULL
 - BGL_TREE_SR_0 BGL_TREE_SR_1 BGL_TREE_SR_2
 - BGL_CACHES
 - BGL_UPC_PU0 BGL_UPC_PU1
 - BGL_UPC_L3_SOME BGL_UPC_L3_CONT
 - (BGL_FPU ARITH+LDST)
 - BGL_FPU_A BGL_FPU_B BGL_FPU_C BGL_FPU_D

Scalasca summary expt with HWC metrics



JUBL tool review

- **HPCT -pg** good for callgraph profile, but no MPI information
- **HPCT -mpitrace** can provide low-overhead MPI profiling, but incomplete profiles/trace and no callgraph information
- **mpiP** is a complete MPI communication profiler
 - -k controls completeness of callgraph vs. overhead
- **TAU** provides a complete MPI+callgraph+HWC profiler, but overhead is substantial, even with default throttling
 - lower overhead configurations are probably possible
- **Scalasca** provides complete MPI+callpath+HWC runtime summarization and tracing with automatic trace analysis
 - trace analysis offers deeper insight than possible with profiling
 - blacklisting often necessary to keep overhead under control and avoid overlarge traces

General notes

- Measurement experiment typically limited to parallel region between MPI_Init and MPI_Finalize
 - experiment preparation before/during MPI_Init
 - experiment finalisation during/after MPI_Finalize
- Tools typically rely on normal application termination
 - if execution is killed or exits without successful MPI_Finalize measurement experiment report not produced or incomplete
 - additional finalisation time should be included in job runtime
- Measurement will necessarily dilate parallel execution
 - small amounts of dilation are unavoidable
 - dilation is uneven and most severe for small routines
 - often such routines are in-lined by compiler optimiser
 - might need to adjust instrumentation/measurement

I/O notes

- Application file I/O performance often highly variable
 - depends on load on shared filesystem/network resources
 - and application/system configuration at time of measurement
 - tuning requires very careful extensive benchmarking
 - worst case performance very different from typical case
 - none of the available tools deal well with this
- Optimal I/O is no I/O!
 - preferable to eliminate non-essential I/O during measurement
 - configure tools to avoid intermediate measurement I/O (e.g., trace buffer flushes) where appropriate
 - configure measurement or analysis to exclude I/O phases
 - typically part of one-off application initialisation/finalisation cost which would be amortised in long production execution

Scaling notes

- Typically want to analyse execution at production scale
 - performance may vary during course of execution
 - performance at small scales may differ considerably
 - best understood in context of performance scaling study
- Large-scale performance measurement & analysis often a “grand challenge” in its own right
 - quantity of measurements & analyses are proportional to number of processes (for constant application complexity)
 - event traces grow linearly with duration of measurement
- Advisable to start small and increase scale in stages
 - profiling/summarisation requirements invariant with time
 - trace targeted short sections (e.g., only a few iterations)

Communication & synchronisation

- Expect communication costs to grow with num. processes
 - Examine number of messages & bytes transferred
 - redundant data transfers become increasingly costly
 - however, read+broadcast often better than parallel read
 - Synchronising operations become more serious
 - blocking/waiting time is wasted time
 - collectives may synchronise all ranks unnecessarily
- Origins of imbalance need careful examination
 - often earlier than it ultimately manifests

Computation

- Not a strength of performance tools covered here
 - HPCT -pg/Xprofiler can examine node performance
 - Hardware counters offer some insight into performance of caches and FPU co-processors
- Optimised libraries are customised to BG/L platform
 - often provide a drop-in replacement for user functions
- Optimising compilers can often do a good job
 - at least when given a reasonable chance
- Tuning experts study assembly code produced by compiler
 - and work with the compiler to do better
- Most tuning can be done at very small scale
 - with only a single compute node (or less!)

What about performance tools on BG/P?

- MPI-based tools expected to port straightforwardly
 - Support for OpenMP and MT will require considerable work
- PAPI likely to require dedicated porting effort
 - Hardware counters and access/control will be different
- Larger scale likely to encounter new scalability bottlenecks
- MPI application bottlenecks expected to be similar
 - investigate/resolve bottlenecks now to be prepared
- Processor/co-processor tuning?

Conclusion

- Performance is productivity
 - get results faster, or more results in same amount of time
 - make previously intractable results possible at all
- If performance isn't measured regularly, it quickly goes bad!
 - the execution environment is constantly changing, even if your application isn't
- Variety of performance tools available on BlueGene/L for in-depth analysis of computation & communication
 - start with simple profiling/summarization for an overview
 - progress to tracing to investigate communication inefficiencies
- Know when performance is “good enough for now”
 - return to performance analysis later when it's a problem again