Performance Analysis at Scale: The Score-P Tools Infrastructure

cube

23 May 2016

Frank Winkler

On-site contractor for Vampir, CSMD





Disclaimer

It is extremely easy to waste performance!

- Bad MPI (50-90%)
- No node-level parallelism (94%)
- No vectorization (75%)
- Bad memory access pattern (99%)
- In sum: 0.008% of the peak performance (about 2 teraflops of Titan)







Disclaimer (2)

Performance tools will not automatically make your code run faster. They help you understand, what your code does and where to put in work.





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

Performance engineering workflow





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

Agenda

Performance Analysis Approaches

- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Motivation
- Functionality
- Architecture
- Workflow
- Advanced Features

Performance Analysis Tools

- Cube
- Vampir

Demo

• Performance Analysis of Jacobi Solver on Titan

Conclusions





Sampling



- Running program is periodically interrupted to take measurement
- Statistical inference of program behavior
 - Not very detailed information on highly volatile metrics
 - Requires long-running applications
- Works with unmodified executables





Instrumentation



- Measurement code is inserted such that every event of interest is captured directly
 - Can be done in various ways
- Advantage:
 - Much more detailed information
- Disadvantage:
 - Processing of source-code / executable necessary
 - Large relative overheads for small functions



Profiling vs. Tracing

Statistics





• Timelines







Terms Used and How They Connect







So what is the right choice?

SO, YOU HAVE DECIDED TO UNDERSTAND WHAT A PROGRAM EXACTLY DOES?





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

Agenda



- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Motivation
- Functionality
- Architecture
- Workflow
- Advanced Features

Performance Analysis Tools

Cube

• Vampir

Demo

Performance Analysis of Jacobi Solver on Titan

onclusions







Score-P: Motivation

- Several performance tools co-exist
- Separate measurement systems and output formats
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
- Limited or expensive interoperability
- Complications for user experience, support, training

Vampir	Scalasca	TAU	Periscope
VampirTrace	EPILOG /	TAU native	Online
OTF	CUBE	formats	measurement





Score-P: Functionality

- Typical functionality for HPC performance tools
 - Instrumentation (various methods)
 - Sampling (experimental)
- Flexible measurement without re-compilation
 - Basic and advanced profile generation
 - Event trace recording
- Programming paradigms:
 - Multi-process
 - MPI, SHMEM
 - Thread-parallel
 - OpenMP, Pthreads
 - Accelerator-based
 - CUDA, OpenCL, OpenACC (Prototype)

- Hybrid parallelism





Score-P: Architecture





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



- 0. Perform a reference run and note the run time to be able to refer to it later
- 1. Instrument your application with Score-P



• To see all available options for instrumentation:







• For CMake and autotools based build systems it is recommended to use the scorep-wrapper script instances



• Pass instrumentation and compiler flags at make:









2. Perform a measurement run with profiling enabled

• Example for generating a profile:

```
$ export SCOREP_ENABLE_PROFILING=true #default
$ export SCOREP_ENABLE_TRACING=false #default
$ export SCOREP_EXPERIMENT_DIRECTORY=profile
$ aprun <instrumented binary>
```

• To see all environment variables for the measurement:

```
$ scorep-info config-vars --full
SCOREP_ENABLE_PROFILING
[...]
SCOREP_ENABLE_TRACING
[...]
SCOREP_TOTAL_MEMORY
Description: Total memory in bytes for the measurement system
[...]
SCOREP_EXPERIMENT_DIRECTORY
Description: Name of the experiment directory
[...]
```





3. Compare profile runtime with reference runtime

- If overhead is too high:
 - Exclude short frequently called functions from measurement using hints from scorep-score







- 4. Create an optimized profile with filter applied if measurement overhead of full instrumented profile is too high
 - Create a filter file and list functions to be excluded

```
$ vim scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
matmul_sub
matvec_sub
binvcrhs
SCOREP_REGION_NAMES_END
```

• Example for generating a profile with filter applied:

```
$ export SCOREP_ENABLE_PROFILING=true
$ export SCOREP_ENABLE_TRACING=false
$ export SCOREP_FILTERING_FILE=scorep.filt
$ export SCOREP_EXPERIMENT_DIRECTORY=profile_with_filter
$ aprun <instrumented binary>
```





5. Perform analysis on (optimized) profile data

• Flat profile analysis with cube_stat:

<pre>\$ cube_stat -t 3 -p profile_with_filter/profile.cubex</pre>										
cube::Region	NumberOfCalls	ExclusiveTime	InclusiveTime							
!\$omp do @z_solve.f:52	51456.000000	131.579771	131.579771							
!\$omp do @y_solve.f:52	51456.000000	122.818761	122.818761							
!\$omp do @x_solve.f:54	51456.000000	117.027571	117.027571							

• Call-path profile analysis with Cube:

\$ cube profile_with_filter/profile.cubex





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



6. Define an appropriate filter for a tracing run

- Exclude functions from measurement which require a large trace buffer to reduce total trace size
- Use scorep-score with full instrumented profile



• Test the effect of your filter on the trace file

\$ scorep-score -f scorep.filt profile.cubex







5. Perform a measurement run with tracing enabled and the filter applied

- \$ export SCOREP_ENABLE_PROFILING=false
- \$ export SCOREP_ENABLE_TRACING=true
- \$ export SCOREP_EXPERIMENT_DIRECTORY=trace
- \$ export SCOREP_FILTERING_FILE=scorep.filt
- \$ aprun <instrumented binary>

6. Perform analysis on the trace data with Vampir



\$ vampir trace/traces.otf2

Score-P: Workflow Summary







Score-P Advanced Features: Sampling

- Alternative to compiler instrumentation to generate profiles or traces
- Regulate the trade-off between overhead and correctness
- Libunwind/1.1 to capture current stack
- Sampling interrupt sources:
 - Interval timer, PAPI, Perf
- Example for enabling sampling for measurement run:

\$ export SCOREP_ENABLE_UNWINDING=true
 \$ export SCOREP_SAMPLING_EVENTS=PAPI_TOT_CYC@1000000

- Combination of instrumented and sampled events (not for compiler instrumented events)
- Calling context information for every event





Score-P Advanced Features: Memory Rec.

- Memory (de)allocations are recorded via the libc/C++ API
- Recording of memory location's call-site in sampling mode
 - Debugging symbols required (-g)
- Interplay of memory usage and application's execution
 - CUBE: (De)allocation size, maximum heap memory, leaked bytes
 - Vampir: Memory usage in "Counter Timelines"
- Enabling memory recording for measurement run:

\$ export SCOREP_MEMORY_RECORDING=true





Agenda



- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Motivation
- Functionality
- Architecture
- Workflow
- Advanced Features

Performance Analysis Tools

Cube

• Vampir

Demo

Performance Analysis of Jacobi Solver on Titan

onclusions





Cube

- Profile analysis tool for displaying performance data of parallel programs
- Originally developed as part of Scalasca toolset
- Available as a separate component of Score-P
- Representation of values (severity matrix) on three hierarchical axes
 - Performance property (metric)
 - Call-tree path (program location)
 - System location (process/thread)
- Three coupled tree browsers









Cube: Analysis Presentation



Vampir



- Event trace analysis tool for displaying performance data of complex parallel programs
- Show dynamic run-time behavior graphically at a fine level of detail
- Provide summaries (profiles) on performance metrics

Timeline charts

• Show application activities and communication along a time axis



Summary charts

• Provide quantitative results for the currently selected time interval







Vampir: Performance Charts Overview

Timeline Charts



Master Timeline



- Summary Timeline
- Performance Radar



- Process Timeline
- Counter Data Timeline

- all threads' activities over time per thread
- all threads activities over time per activity
- all threads' perf-metric over time
- single thread's activities over time
 - single threads perf-metric over time

Summary Charts



Function Summary

Message Summary



Process Summary



Communication Matrix View





• Trace visualization of FDS (Fire Dynamics Simulator)











Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler





Summary Timeline





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



Process Timeline





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler





Counter Timeline















Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler





Vampir: Where Do the Metrics Come From?

• Custom Metrics Built-In Editor

Active Image: Custom Metrics Image: Custom Metrics <
Active Image: Constraint of the state o
Image: Time Spent in MPI_ MPI_Wait Inclusive Image: Time Spent in MPI_





Function Summary









🚦 Process Summary









Communication Matrix View





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

Vampir at Scale: FDS with 8192 cores

• Fit to chart height feature in Master Timeline

000	♥ Trace View - Chester:/lustre/atlas/proj-shared/stf010/w	inklerf/MPI_SCALE/reference_8	3192_8_procs_trace_filter/trace/trace	s.otf2 – Vampir 👷
i 🗮 📷 🔛 🐻	🌔 😒 😹 👪 差 🖏 🕼 🖍 😜)s	964.104 s	964 s
00	Timeline	0 c 700 c 800 c		Function Summary
U.S.				
Master thread:1			2,000,000 3	MPL Barrier
Master thread:364			2 300 760 65	4 s MAIN
Master thread:480 - Master thread:664 -			1.997.224	.331 s divg.divergence part 1
Master thread:864			464,25	8.627 s pres.pressure solver
Master thread:1004 - Master thread:1306 -			258,	161.183 s MPI_Allreduce
Master thread:1506			98	8,383.802 s MPI_Waitall
Master thread:1769 - Master thread:2000 -			7	7,675.037 s MPI_File_write_at
Master thread:2138			5	1,696.982 s MPI_Gatherv
Master thread:2320 - Master thread:2500 -			4	8,432.479 s MPI_Allgatherv
Master thread:2643			4	1,030.476 s MPI_Init_thread
Master thread:2909				2,106,409 s divg.divergence_part_2_
Master thread:3522				1,076,264 c MPL Startall
Master thread:3967				1,970.204 3 MIT_Startan
Master thread:4210				Context View
Master thread:4601				× Trace Info 🗾 🔺 🕂
Master thread:4746			Property	Value
Master thread:5005			File	Chester:/lustre/atlas/
Master thread:5410			Creator	Score-P 2.0-trunk
Master thread:5661 Master thread:5799	Overview of the		Version	2.0
Master thread:5975	entire application run		Number of Proces	sses 8,192
Master thread:6400	entile application run		Timer Resolution	454.540618 ps
Master thread:6608 - Master thread:6798 -	across all processes			
Master thread:7003				Function Legend
Master thread:/153 - Master thread:7332 -	based on available		MPI	
Master thread:7484	nixels on screen		Pressure	
Master thread:7727 - Master thread:7897 -				
Master thread:8132			Monitor	





• 5831 processes: 343xMPI with 8xOpenMP and 8xCUDA



Connected: Chester



Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



Group threads and CUDA streams





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

31



Collapse all MPI processes





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



• Fit to chart height for all collapsed MPI processes

\odot \bigcirc \bigcirc \bigcirc							V Tr	race View – C	hester:/lus	tre/atlas/pro	oj-shared/s	tf010/winkl	erf/LSMS/T	race_343xr	mpi_8xomp	_1xgpu/tra	ces.otf2 – \	ampir			r i i i i i i i i i i i i i i i i i i i
i 🗮 🗟 📓 👪	6	6 🔄	88 👪	差 🔄	1	1	Os			Liberal	dan kanalan dan ka Kabupatén kanalan kan	an a darlad a practice An la distance a darlad	eenen aaroon ma Vederdadaalaalaa		منار ار او او او او	977.7	67 s	يىرىيى بىرىيى 1. 1. يا يەلىرا يا	al de la	ninian managementa a santa minana da	n mar ann de an ann an Araban a An Araban an Araban a
	0s	50 s	100 s	150 s	20 <u>0</u> s	25 <u>0</u> s	30 <u>0</u> s	350 s	40 <u>0</u> s	450 s	50 <u>0</u> s	55 <u>0</u> s	600 s	65 <u>0</u> s	70 <u>0</u> s	750 s	80 <u>0</u> s	85 <u>0</u> s	90 <u>0</u> s	950 s	All Processes, Accumulated Exclusive Time per F
Master thread:0 Master thread:0 Master thread:0 Master thread:15 Master thread:15 Master thread:25 Master thread:30 Master thread:30 Master thread:30 Master thread:40 Master thread:50 Master thread:50 Master thread:60 Master thread:60 Master thread:60 Master thread:60 Master thread:90 Master thread:100 Master thread:100 Master thread:100 Master thread:100 Master thread:100 Master thread:115 Master thread:100 Master thread:115 Master thread:120 Master thread:120 Master thread:135 Master thread:120 Master thread:135 Master thread:145 Master thread:145 Master thread:145 Master thread:150 Master thread:160 Master thread:160 Master thread:160 Master thread:160 Master thread:160 Master thread:160 Master thread:160 Master thread:160 Master thread:250 Master thread:200 Master thread:	05	50 S	100 s	∠ 50 s	200 s	20 s	0s	350 s	400 s	Implies and the second se	500 s	550 s	600 s	650 s	700 s	977.17 750 s	800 s	850 s	900 s	950 s	Function Summary Processes, Accumulated Exclusive Time per F Os All Processes, Accumulated Exclusive Time per F Os S 1.567,480.515 s cuEventSynchronize 244,366.217 s void zblock_lut*, int*, int*) 222,039.009 s zgemm_sm35x8x64x8x16 141,139.875 s Isomp impliciation.cpp:469 122,371.407 s cnewint 106,307.5 s Tomp impliciation.cpp:251 12,792.68 s Isomp impliciation.cpp:251 12,792.68 s int buildLIZ(IZInfoTypes.8) 10,987.346 s .228femiPlu0_54_S0_S0_i 9,081.354 s MPL_Allreduce 8,702.104 s cuEventRecord 7,270.66 s MPL_Recv 5,309.138 s cuMemcpyHtoDAsync_v2 4,514.468 s newint 3,662.155 s madsum 3,437.672 s int zz.zi 2,510.595 s ISomp implicitrices.cpp:37 2,415.126 s .217cudaMeme2EvPT_51_i 1,138.65 s semrel 1,138.5 s semrel 2,720.665 s .217cudaMeme2EvPT_51_i 1,138.6 s
Master thread:280 Master thread:285 Master thread:290 Master thread:295 Master thread:300 Master thread:305 Master thread:310																					Application CUDA_API MPI CUDA_FLUSH CUDA_SYNC OMP_LOOP
Master thread:315 Master thread:320 Master thread:325 Master thread:330 Master thread:335 Master thread:340																		n sisi Vali Philip			OMP_SYNC OMP_PARALLEL OMP_API CUDA_KERNEL Monitor





Agenda



- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Motivation
- Functionality
- Architecture
- Workflow
- Advanced Features

Performance Analysis Tools

- Cube
- Vampir

Demo

• Performance Analysis of Jacobi Solver on Titan

onclusions





Demo: Jacobi Solver

- Jacobi Example
 - Iterative solver for system of equations

 $U_{old} = U$

 $u_{i,j} = bu_{old,i,j} + a_x(u_{old,i-1,j} + u_{old,i+1,j}) + a_y(u_{old,i,j-1} + u_{old,i,j+1}) - rHs / b$

 Code uses OpenMP, CUDA and MPI for parallelization





- Domain decomposition
 - Halo exchange at boundaries:
 - Via MPI between processes
 - Via CUDA between hosts and accelerators







Demo: Jacobi Solver / Setup

• Connect to Titan via X forwarding and copy sources

```
$ cd $MEMBERWORK/[projid]
```

```
$ cp /sw/sources/vampir/tutorial/jacobi.tar.gz .
```

```
$ tar xzvf jacobi.tar.gz
```

```
$ cd jacobi
```

Change programming environment and load modules

```
$ module swap PrgEnv-{pgi,gnu}
```

```
$ module load cudatoolkit
```

```
$ module load scorep
```

Compile benchmark and submit job

```
$ make
$ qsub -A [projid] run.pbs
$ less jacobi.o[JOB_ID]
Jacobi relaxation Calculation: 8192 x 8192 mesh with
2 processes and 16 threads + one Tesla K20X for each process.
614 of 2049 local rows are calculated on the CPU to balance the load
between the CPU and the GPU.
0, 0.489197
100, 0.002397
[...]
total: 8.425432 s
```





Demo: Jacobi Solver / Profiling

Build instrumented executable

```
$ make clean
$ make scorep
scorep --cuda cc ... -o bin/jacobi_mpi+openmp+cuda
```

Submit job for profiling run







Demo: Jacobi Solver / Profile Analysis

• Perform flat profile analysis with cube_stat

<pre>\$ cd bin.scorep</pre>			
<pre>\$ cube_stat -t 10 -p jacobi_mpi+</pre>	openmp+cuda_pro	ofile/profile.c	cubex
cube::Region	NumberOfCalls	ExclusiveTime	InclusiveTime
!\$omp for @jacobi_cuda.c:188	32000.000000	131.797289	131.797289
!\$omp implicit barrier	32000.000000	104.298683	104.298683
!\$omp for @jacobi_cuda.c:258	32000.000000	42.999056	50.568642
[]			

• Perform call-path profile analysis with Cube

\$ cube jacobi_mpi+openmp+cuda_profile/profile.cubex



Demo: Jacobi Solver / Scoring

• Do we need a filter? (Overhead and memory footprint)

<pre>\$ scorep-score jacobi_mpi+openmp+cuda_profile/profile.cubex Estimated aggregate size of event trace: 10MB</pre>	No filtering required.									
Estimated requirements for largest trace buffer (max buf): 5MB										
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 41MB										
(hint: When tracing set SCOREP_TOTAL_MEMORY=41MB to avoid inter	mediate									
flushes or reduce requirements using USR regions filters.)										
<pre>flt type max_buf[B] visits time[s] time[%] time/visit[us]</pre>	region									
ALL 4,924,060 310,504 308.53 100.0 993.63	ALL									
OMP 4,135,850 256,417 287.31 93.1 1120.46	OMP									
CUDA 494,338 38,025 10.40 3.4 273.53	CUDA									
COM 156,260 12,020 10.46 3.4 870.58	COM									
MPI 137,222 4,012 0.30 0.1 73.96	MPI									
MEMORY 260 20 0.06 0.0 2972.15	MEMORY									
USR 130 10 0.00 0.0 10.26	USR									





Demo: Jacobi Solver / Tracing

• Submit job for tracing run

```
$ cd ...
$ less run trace.pbs
[...]
export SCOREP ENABLE PROFILING=false
export SCOREP ENABLE TRACING=true
export SCOREP EXPERIMENT DIRECTORY=jacobi mpi+openmp+cuda trace
export SCOREP CUDA ENABLE=yes
export SCOREP TIMER=clock gettime
export SCOREP MEMORY RECORDING=yes
export SCOREP TOTAL MEMORY=50MB
[...]
aprun -n 2 -d 16 -N 1 ./jacobi mpi+openmp+cuda 8192 8192 0.15
$ qsub -A [projid] run trace.pbs
$ less jacobi.o[JOB ID]
Jacobi relaxation Calculation: 8192 x 8192 mesh with
2 processes and 16 threads + one Tesla K20X for each process.
614 of 2049 local rows are calculated on the CPU to balance the load
between the CPU and the GPU.
   0, 0.489197
 100, 0.002397
 [...]
 900, 0.000269
total: 9.895828 s
```





Demo: Jacobi Solver / Trace Analysis

• Perform analysis on the trace data with Vampir





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



Agenda

Performance Analysis Approaches

- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Motivation
- Functionality
- Architecture
- Workflow
- Advanced Features

Performance Analysis Tools

- Cube
- Vampir

Demo

Performance Analysis of Jacobi Solver on Titan

Conclusions



Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



Conclusions

Score-P

- Common instrumentation and measurement infrastructure for various analysis tools
- Hides away complicated details
- Provides many options and switches for experts

General Workflow

- Instrument your application with Score-P
- Perform a measurement run with profiling enabled
- Perform profile analysis with Cube
- Use scorep-score to define an appropriate filter
- Perform a measurement run with **tracing enabled** and the filter applied
- Perform in-depth analysis on the trace data with Vampir







If you have any questions or need help, please don't hesitate to contact me under winklerf@ornl.gov.

Detailed information under:

http://www.vi-hps.org/projects/score-p or

https://www.olcf.ornl.gov/support/software/



Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



Score-P Advanced Features: Metrics

• Available PAPI metrics

 Preset events: common set of events deemed relevant and useful for application performance tuning

\$ papi_avail

Native events: set of all events that are available on the CPU (platform dependent)

\$ papi_native_avail

• Available resource usage metrics

```
$ man getrusage
[... Output ...]
struct rusage {
   struct timeval ru_utime; /* user CPU time used */
   struct timeval ru_stime; /* system CPU time used */
   [... More output ...]
```





Score-P Advanced Features: Metrics (2)

• Recording hardware counters via PAPI

\$ export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_FP_INS

• Recording operating system resource usage

\$ export SCOREP_METRIC_RUSAGE=ru_maxrss,ru_stime





Vampir: Visualization Modes (1)

• Directly on front end or local machine

\$ vampir





Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler



Vampir: Visualization Modes (2)

• On local machine with remote VampirServer



Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

• Indentification of program phases

000	V Trace Vie	w – Cheste	:/lustre/	atlas/proj-sh	ared/stf010	/winklerf/	MPI_SCALE	/reference	_8192_8_procs	_trace_filter/trace/traces.otf2 - Vampir	E A
🚟 🗟 🍇 👪	i 🗧 🔁		ا 🟄	ā 🐌	1	Os				964.104 s	964 s
05	100 c	200 c	300 c	Timeline	500 c	600 s	700 c	800 c	000 c	Function Sum	mary
0,3	100 5	2003	30,0 3	400 5	50,0 5	000 5	700 3	00,0 5	500 5	2 000 000 c	sive time per runction
Master thread:1										2,000,000 s 0s	
Master thread:207										2,537,706.44 s	API_Barrier
Master thread:480										2,300,760.654 s	MAIN
Master thread:664										1,997,224.331 s	livg.divergence_part_1_
Master thread:864										464.258.627 s	pres.pressure solver
Master thread:1004										258,161,183 5	
Master thread:1506										98 383 802 c	MPL Waitall
Master thread:1769										77 675 027 -	ADI File unite et
Master thread:2000										77,075.037 S	API_FILe_write_at
Master thread:2138										51,696.982 s	MPI_Gatherv
Master thread:2500										48,432.479 s	4PI_Allgatherv
Master thread:2643										41,030.476 s	MPI_Init_thread
Master thread:2909										14,804.909 s	livg.divergence_part_2_
Master thread:3222										2,196.408 s	MPI_File_open
Master thread:3678										1.976.264 s	MPI Startall
Master thread:3967											_
Master thread:4210										Context Vi	0.14
Master thread:4414											
Master thread:4746											
Master thread:4910										Property	Value
Master thread:5005										File	Chester:/lustre/atlas/
Master thread:5410										Creator	Score-P 2.0-trunk
Master thread:5661										Version	2.0
Master thread:5975										Number of Processes	8,192
Master thread:6400										Timer Resolution	454.540618 ps
Master thread:66798											
Master thread:7003										Function Leg	Jend
Master thread:7153										MPI	
Master thread:7332										Pressure	
Master thread:7484										Divergence	
Master thread:7										Application	
Master thread:8	tializati	on Ph	lase		Con	nputa	ation	Phas	e 📕	Monitor	
											Connected: Chester





Load imbalance in initialization phase







• Load imbalance in initialization phase (2)









Computation phase







Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

• Unnecessary synchronization in computation phase

\odot \bigcirc \bigcirc	V Trace View - Cł	hester:/lustre/atlas/proj-s	shared/stf010/winklerf/MPI_SCA	LE/reference_8192_8_proc	s_trace_filter/trace/traces.c	otf2 – Vampir ⊮™
i 🚍 📷 🔤 🚻	. 🖸 🍮 📑	🔢 🛎 🔄 🔯			592.50 s	592.51 s
		Timeline				Function Summary
	592.496 s	592.498 s 592	2.500 s 592.502 s	592.504 s	All Processes, Accur	nulated Exclusive Time per Function
Master thread:0	pres.pressure solver		MPI Barrier	MAIN	40 s 2	0 s Os
Master thread:1	pres.pressure		MPI Barrier	MAIN	50.985 S	MPI_Barrier
Master thread:2	pres.pressure	DI Dorrior	MPI Barrier	MAIN	10	20.507 s pres.pressure_solver_
Master thread:3	pres.pressure	PI_bamer	MPI_Barrier	MAIN_	10	0 287 s MPL Startall
Master thread:4	pres.pressure	nneeded	MPI_Barrier	MAIN_		0.258 s MPI Waitall
Master thread:5	pres.pressure	iniceaca.	MPI_Barrier	MAIN_		
Master thread:6	pres.pressure_solver_	\neg	MPI_Barrier	MAIN_		
Master thread:7	pres.pressure_solver_		MPI_Barrier	MAIN_		
Master thread:8	pres.pressure_solver_	-	MPI_Barrier	MAIN_		
Master thread:9	pres.pressure_solver_	-	MPI_Barrier	MAIN_		
Master thread:10	pres.pressure_solver_	-	MPI_Barrier	MAIN_		
Master thread:11	pres.pressure_solver_	-	MPI_Barrier	MAIN_		
Master thread:12	pres.pressure_solver_	-	MPI_Barrier	MAIN_		Context View
Master thread:13	pres.pressure_solver_	-	MPI_Barrier	MAIN_	^ <u></u> ×	😹 Master Timeline 📃 🗾 🔸
Master thread:14	pres.pressure_solver_	-	MPI_Barrier	MAIN_	Property	Value
Master thread:15	pres.pressure_solver_	-	MPI_Barrier	MAIN_	Display	Master Timeline
Master thread:16	pres.pressure_solver_	-	MPI_Barrier	MAIN_	Туре	Collective
Master thread:17	pres.pressure_solver_	MPI_Barr	ier	MAIN	Commission	
Master thread:18	pres.pressure_solver_	MPI_Bar	rier	MAIN_	Communicator	MPI COMM WORLD
Master thread:19	pres.pressure_solver_	<mark>MPI_B</mark>	arrier	MAIN_	Collective Operatio	n BARRIER
Master thread:20	pres.pressure_solver_	MPI_Bar	rier	MAIN_		
Master thread:21	pres.pressure_solver_	MPI_Ba	rrier	MAIN_		
Master thread:22	pres.pressure_solver_	MPI_Ba	rrier	MAIN_		Even the second
Master thread:23	pres.pressure_solver_	- MPI_Bai	rrier	MAIN_	MDI	Function Legend
Master thread:24	pres.pressure_solver_	MPI_Ba	rrier	MAIN_	Pressure	
Master thread:25	pres.pressure_solver_	MPI_Barr	ier	MAIN_	Divergence	
Master thread:26	pres.pressure_solver_	- MPI_Bar	rier	MAIN_	Application	
Master thread:27	pres.pressure_solver_	- MPI_Bar	rier	MAIN_	Monitor	
		:	-			
						Connected: Chester





• Inefficient cache usage in computation phase

\odot \bigcirc \bigcirc	✓ Trace View – Chester:/lustre/atlas	/proj-shared/stf010/winklerf/M	PI_SCALE/reference_8192_8_procs_	_trace_filter/trace/traces.otf2 - Vampir	R ₂
: 🚟 🛣 🌉 📕	o 🌕 📑 🔚 👪 🔌 🐴	👪 🖈 💡 i 🚺		719.763	s 719.780 <i>s</i> 68 ms
	Tim 719.7675 s 719.7700 s	eline 719.7725 s 719.7750 s	719 7775 s	Function Su	mmary usive Time per Function
		15		50 s 25 s	Os
Master thread:0	MAINMPI_Allgat	nerv MPL Gatherv MA	IN	64.878 s	MAIN_
Master thread:1	MAINMPI_Allg	atherv MAIN		18.711 s	pres.pressure_solver_
Master thread:2	MAINMPI_Allg	atherv 🛉 🛉 MAIN_		15.492 s	MPI_Gatherv
Master thread:3	MAINMPI_Allg	atherv 🛉 🛉 🛉 MAIN_		14.749 s	MPI_Allgatherv
Master thread:4	MAINMPI_Allg	atherv 🛉 🛉 MAIN_		6.425 s	MPI_Barrier
Master thread:5	MAIN	atherv	-		
Master thread:6	MAINMPI_Allg	atherv MAIN_			
Master thread:7	MAINMPI_Allg	atherv 🛉 🛉 MAIN_			
Master thread:8	MAINMPI_Allg	atherv MAIN_	-		
Master thread:9	MAINMPI AU	MAIN_			
Master thread:10	MAIN	ow Flops/s 🛯 🏧	-		
Master thread:11	MAIN				
Master thread:12	MAIN				
		higher 12	1		
Master thread:0, Values	of Metric "PAPI_FP_OPS" over				
125 M ·····		cache			
100 M ·····					
S 75 M ·····		miss rate.			
[™] 50 M·····	<u> </u>				
25 M					
0-					
Master thread:0, Values	of Metric "PAPI_L2_DCM" over Time				
30.0 M·····				Function L	egend
22.5 M					
₩ 15.0 M····				Divergence	
7.5 M·····				Application	
_				Monitor	
	: :				
					Connected: Chester



Performance Analysis at Scale: The Score-P Tools Infrastructure – Frank Winkler

