

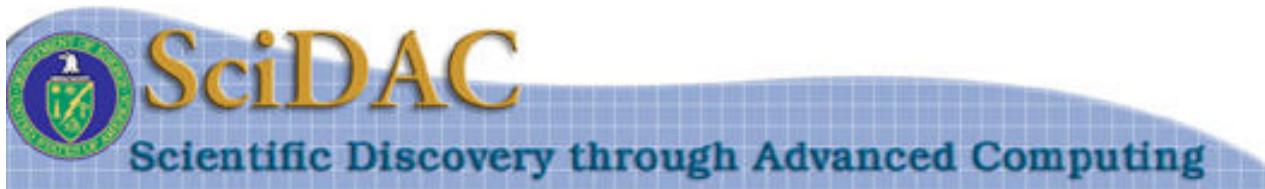
PERC Tools

Bronis R. de Supinski, Allan Snavely and Ying Zhang
bronis@llnl.gov, allans@sdsc.edu, zhang8@cs.uiuc.edu,

Performance Evaluation Research Center
Scientific Discovery through Advanced Computing Program
Office of Science, Department of Energy

August 5, 2003

<http://perc.nersc.gov/tutorials/scicomp8>



Session I: Overview and Level I Tools 8:30 – 10:00 AM

Bronis R. de Supinski
bronis@llnl.gov

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

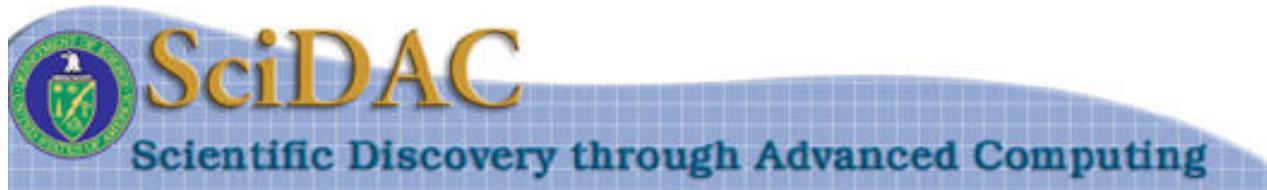


Table of Contents

Session I (8:30 – 10:00AM): Overview and Level I Tools

✍ Overview and Introduction

- ✍ PERC
- ✍ Types of Performance Tools
- ✍ POP

✍ Tutorial Basics

✍ Level 1 Performance Data Gathering and Analysis Tools

- ✍ Statistical sampling tool: gprof
- ✍ Performance monitor tool: hpmcount
- ✍ MPI profiling tool: mpiP

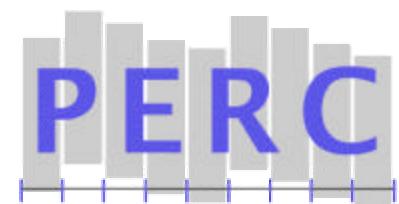


Table of Contents (page 2)

Session II (10:30AM – 12:30PM): Level II Tools: SvPablo

SvPablo Overview

-  Goal and architecture

SvPablo tutorial

-  SvPablo installation
-  Source code instrumentation
-  Running instrumented code
-  Performance data capture, browsing and analysis
-  Examples

On-going research and future directions



Table of Contents (page 3)

Session III (2:00 – 3:30PM): Performance Modeling

- ✍ **Introduction to Convolution Methods**
- ✍ **Machine Profiles**
- ✍ **Application Signatures: Level 3 Performance Data**
 - ✍ Paraver
- ✍ **Dimemas, a Tool for Convoluting**

Table of Contents (page 4)

Session IV (4:00 – 5:30PM): MetaSim and Dimemas

- ✍ **MetaSim Tracer**
- ✍ **MetaSim Convolver**
- ✍ **Putting All the Pieces Together**
- ✍ **Tutorial Conclusion**
 - ✍ Taking it home: web site and more
 - ✍ Questions

PERC Overview

Mission:

-  Develop a science of performance
-  Engineer tools for performance analysis and optimization

Focus on large, grand-challenge calculations, especially large-scale scientific codes used in SciDAC projects

Participating Institutions:

Argonne Natl. Lab.	Univ. of California, San Diego
Lawrence Berkeley Natl. Lab.	Univ. of Illinois
Lawrence Livermore Natl. Lab.	Univ. of Maryland
Oak Ridge Natl. Lab.	Univ. of Tennessee, Knoxville

Website: <http://perc.nersc.gov>



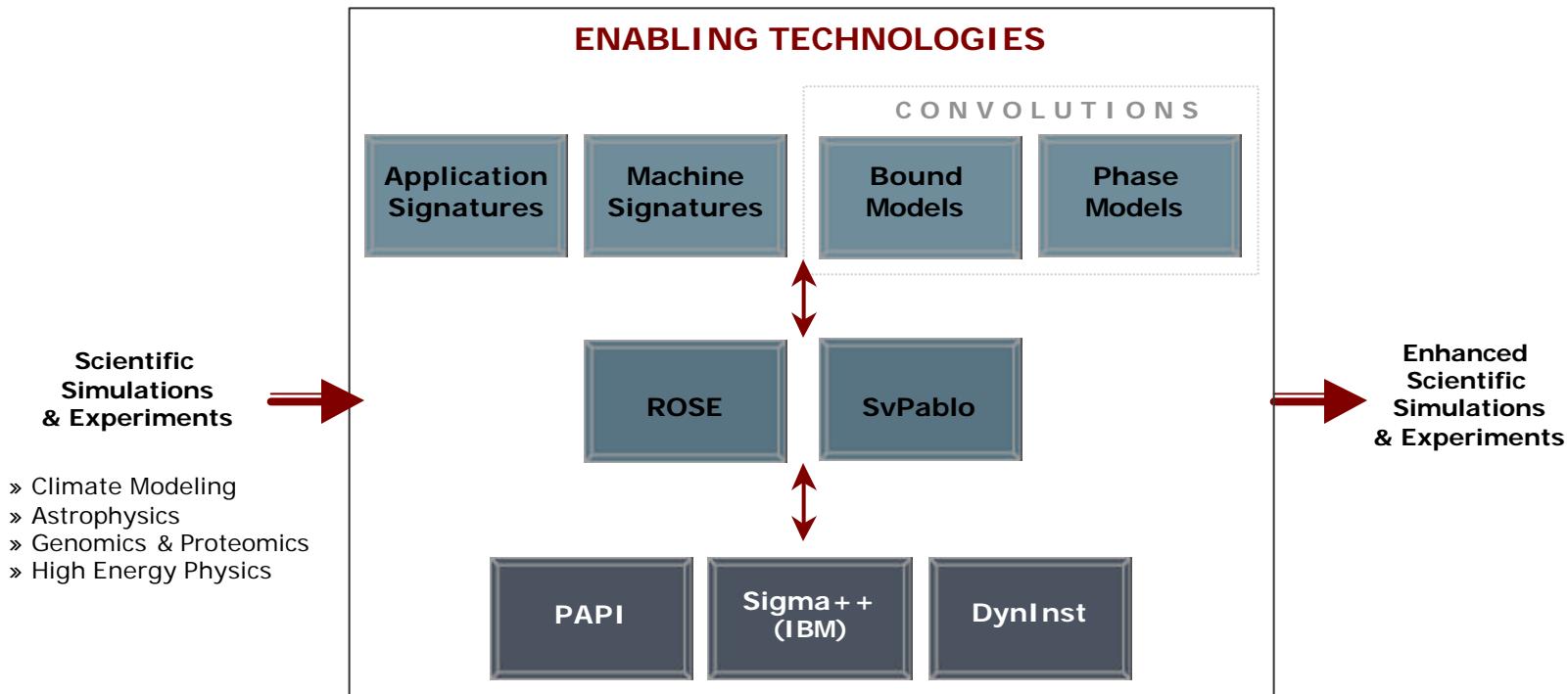
PERC Objectives

- ✍ Understand key application factors that affect performance
- ✍ Understand key system factors that affect performance
- ✍ Develop models that accurately predict performance
- ✍ Develop an enabling infrastructure of tools for performance monitoring, modeling and optimization
- ✍ Validate these ideas and infrastructure via close collaboration with DOE Office of Science and others
- ✍ Transfer the technology to end users
 - ✍ Working directly with application teams
 - ✍ Tutorials



PERC Organization

- Developing a *science* for understanding performance of scientific applications on high-end computer systems and *engineering* strategies for improving performance on these systems



Types of Performance Tools

✍ Data Gathering and Analysis Tools

✍ Level 1 Tools

- ✍ Provide coarse data – typically over a whole program run
- ✍ Easy to use - almost no changes to executable (relinking, maybe)

✍ Level 2 Tools

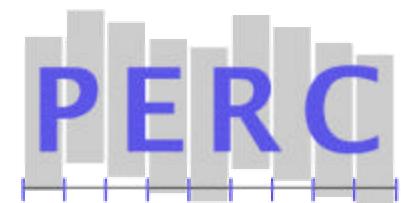
- ✍ Data broken down by module, routine and their descendants
- ✍ More time consuming to gather, often require recompiling

✍ Level 3 Tools

- ✍ “Application Signatures” are primarily level 3 data
- ✍ Detailed data including memory access and message passing traces
- ✍ Very time consuming to gather, may require source level modifications

✍ Modeling Tools

- ✍ Machine signatures (aka microbenchmarks)
- ✍ Convolution and prediction tools
 - ✍ MetaSim
 - ✍ Dimemas
 - ✍ Performance Bounding Tool



Parallel Ocean Program (POP)

✍ Important climate modeling application

- ✍ Ocean general circulation model
- ✍ Developed at LANL under sponsorship of DOE CHAMP Program
- ✍ Ocean component of the Community Climate System Model (CCSM) (NCAR)
- ✍ Also used for high resolution global ocean and regional studies

✍ Scalable, compute intensive HPC application

- ✍ Finite difference formulation of 3D flow equations on a shifted polar grid
- ✍ Extensively examined application that runs relatively efficiently on several platforms including the Earth Simulator
- ✍ We'll use the one degree horizontal grid ("x1") model in our exercises, the same resolution as used in CCSM simulations



Tutorial Basics

✍ **Web site:** <http://perc.nersc.gov/tutorials/scicomp8>

- ✍ Open browser
- ✍ Choose bookmark “PERC ScicomP8 Tutorial”

✍ **Exercises will be run on NERSC machine “Hockney”**

- ✍ Distribute accounts during this and next few slides

✍ **Tutorial exercise format**

- ✍ Load exercise module
- ✍ Change to exercise subdirectory
- ✍ Review simple example code
- ✍ Build and run simple example, review results
- ✍ Review already gathered results for more complex examples

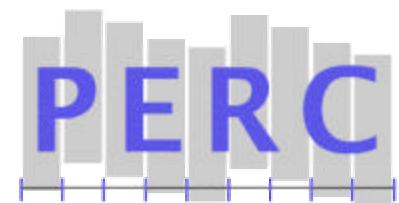
Caveats

☞ This tutorial is not about:

- ☞ Climate modeling
- ☞ Building and running POP
- ☞ Optimizing POP
- ☞ Installing ssh and X server packages
- ☞ Installing PERC software and other performance tools

☞ This tutorial is about:

- ☞ Understanding types of performance tools
- ☞ Running performance tools on simple examples
- ☞ Understanding performance tool output of real codes
- ☞ Providing sufficient details to recreate all exercises and to do all of the above (besides climate modeling)



Logging into Hockney

Use ssh

- ☞ X server software must be running
- ☞ On windows systems, enable X tunneling in ssh application
- ☞ Host is hockney.nersc.gov
- ☞ Unix: type “ssh -P -X -l <user_name> hockney.nersc.gov<cr>”

Establish basic environment

- ☞ Type “module load use.perc<cr>
 - ☞ Establishes various paths
 - ☞ Sets appropriate environment variables
 - ☞ Could be done in .login or .profile
- ☞ Test X connection: type “xclock &<cr>”
- ☞ Additional module commands will establish exercise directories



Building and Running POP

- ☞ Type “module load scicomp8/pop.src<cr>”
- ☞ Included for exercise recreation
 - ☞ Tutorial participants don’t do the following
 - ☞ Note: makefile omitted to avoid long build times...
 - ☞ Note: these exercises use POP v1.4.3
- ☞ Building POP on hockney
 - ☞ Type “setenv ARCH ibm_mpi<cr>”
 - ☞ Type “./set_run_dir <dir_name> x1<cr>”
 - ☞ Edit ibm_mpi.gnu to specify virtual layout:
“DNPROCS = -DNPROC_X=001 –DNPROC_Y=002”
 - ☞ Edit pop_in:
“stop_option = ‘nstep’”
“stop_count = 2”
 - ☞ Type “make<cr>”
- ☞ Running POP on hockney
 - ☞ Use load leveler script (e.g., pop.ll) (uses “poe ./pop –procs 2”)
 - ☞ Type “llsubmit pop.ll<cr>”



Execution Profiling: gprof

✍ Classic and basic performance tool

- ✍ Function call counts
- ✍ Total function execution times from statistical sampling
- ✍ Call graph-based: information for descendants

✍ Gathering execution profile

- ✍ Compile and link with “-pg”
- ✍ Run application – creates gmon.out file (gmon.out.N w/MPI on IBM)
- ✍ Generate profile report – e.g., “gprof <a.out> gmon.out* > gprof.out”

✍ Exercise: Review profile data for POP

- ✍ Type “module load scicomp8/pop.gprof<cr>”
- ✍ Type “cd ~/pop.gprof<cr>”
 - ✍ Includes modified makefile (ibm_mpi.gnu), executables
- ✍ Type “more gprof.out.0<cr>”
- ✍ Type “tail -n 1030 gprof.out.0<cr> | more”



Snippet of Call Graph Profile for POP

ngularity: Each sample hit covers 4 bytes. Time: 153.90 seconds

index	%time	self	descendents	called/total		parents	
				called+self		name	index
				called	total		
<hr/>							
[4]	73.9		1.90	111.78	2/2	.__step_mod_MOD_step [3]	
Focus efforts on these three routines & their descendants		1.90	111.78	2		.__baroclinic_MOD_baroclinic_driver [4]	
		2.10	51.49	80/80		.__baroclinic_MOD_tracer_update [5]	
		1.87	26.47	80/80		.__baroclinic_MOD_clinic [9]	
		0.00	23.68	80/80		.__vertical_mix_MOD_vmix [10]	
		1.83	0.00	40/752		.__state_mod_MOD_state [8]	
		1.78	0.00	2/2		.__vertical_mix_MOD_impmixu [35]	
		0.93	0.77	80/80		.__advection_MOD_adv_flux [36]	
		0.86	0.00	1/4		.__vertical_mix_MOD_impmixt [29]	
		0.00	0.00	80/247		.__boundary_MOD_boundary_2d_real	
	[481]						



Review of Flat Profile Report for POP

ngranularity: Each sample hit covers 4 bytes. Time: 153.90 seconds

%	cumulative	self		self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
22.4	34.44	34.44	752	45.80	45.80	.__state_mod_MOD_state [8]
18.9	63.54	29.10	80	363.75	454.25	.__hmix_gm_MOD_hdifft_gm [6]
8.4	76.46	12.92	80	161.50	261.02	.__hmix_aniso_MOD_hdiffu_aniso [12]
4.2	83.00	6.54				.__mcount [20]
3.8	88.88	5.88	80	73.50	73.50	.__stencils_MOD_hupw3 [22]
3.7	94.62	5.74	2	2870.00	4547.11	.__solvers_MOD_pcg [18]
2.6	98.68	4.06	20180272	0.00	0.00	.__sin [25]
2.6	102.67	3.99	20243776	0.00	0.00	.__cos [26]
2.2	106.11	3.44	4	860.00	860.00	.__vertical_mix_MOD_impvmixt [29]
2.1	109.36	3.25	1	3250.00	3250.00	.__prognostic_MOD_init_prognostic [30]
1.8	112.07	2.71	2	1355.00	1917.59	.__vmix_kpp_MOD_blmix [27]
1.7	114.68	2.61	80	32.62	106.12	.__advection_MOD_advt_upwind3 [19]
1.6	117.17	2.49	2	1245.00	1446.06	.__vmix_kpp_MOD_ri_iwmix [32]
1.5	119.44	2.27	80	28.38	39.81	.__advection_MOD_advu [31]
1.5	121.68	2.24	563	3.98	3.98	.__stencils_MOD_ninept_4 [33]
1.4	123.91	2.23	2	1115.00	1884.23	.__vmix_kpp_MOD_blddepth [28]



Level 1 Hardware Performance Monitor Data: hpmcount

HPM Toolkit

- ☞ Developed by Luiz DeRose of IBM's Advanced Computing Technology Center (ACTC) (A Supplementary PERC participant)
- ☞ Provides convenient access to hardware performance monitors
 - ☞ Requires PMAPI – now part of AIX; previously a kernel extension
 - ☞ Requires compiling with “–qarch=pwr3” or “-O3” on NERSC systems
- ☞ Includes library-based mechanism for level 2 information
 - ☞ Requires “module load hpmt toolkit” on NERSC systems
 - ☞ Not covered here – we use SvPablo for similar purposes

hpmcount

- ☞ Simple interface: type “hpmcount <executable><cr>”
- ☞ Works with MPI programs – use “-o prefix” for separate output files
- ☞ NERSC support includes hpmavg for combining hpmcount output files



hpmcount Event Sets

☞ hpmcount provides access to fixed events

- ☞ Uses “groups” on Power4 systems
- ☞ Uses event sets on Power3 systems, including hockney

☞ Four event sets:

Set 1	Set 2	Set 3	Set 4
Cycles	Cycles	Cycles	Cycles
Instr. completed	Instr. completed	Loads dispatched	Instr. dispatched
TLB misses	TLB misses	L1 load misses	Instr. completed
Stores completed	Stores dispatched	L2 misses	Cycles w/o instr. compl.
Loads completed	L1 store misses	Stores dispatched	I cache misses
FPU0 ops	Loads dispatched	L2 store misses	FXU0 ops
FPU1 ops	L1 load misses	Write back count	FXU1 ops
FMAss executed	LSU idle	LSU idle	FXU2 ops

☞ Choose with “-s” or HPM_EVENT_SET env var



hpmcount Exercises

- ✍ Type “`module load scicomp8/simple.hpm<cr>`”
- ✍ Type “`cd ~/simple.hpm<cr>`”
- ✍ Type “`module load gcc<cr>`”
- ✍ Type “`make<cr>`” (makes all 3 simple examples)
- ✍ **Simple Exercise I: Floating point operations**
 - ✍ Type “`more flops.c<cr>`”
 - ✍ Type “`poe hpmcount -s 1 ./flops –procs 1<cr>`”
 - ✍ Review output for floating point operation count: correct?



More hpmcount Exercises

✍ Simple Exercise II: L1 cache resident problem

- ✍ Type “more small_footprint.c<cr>”
- ✍ Type “poe hpmcount –s 2 ./small_footprint –procs 1<cr>”
- ✍ Review output for L1 cache hit rate: believe it?

✍ Simple Exercise III: L2 cache resident problem

- ✍ Type “more large_footprint.c<cr>”
- ✍ Type “poe hpmcount –s 2 ./large_footprint –procs 1<cr>”
- ✍ Review output for L1 cache hit rate: Ouch!

✍ Exercise: POP cache performance

- ✍ Type “module load scicomp8/pop.hpm<cr>”
- ✍ Type “cd ~/pop.hpm<cr>”
- ✍ Includes LoadLeveler script “pop.ll”
- ✍ Type “hpmavg es1*<cr>” to combine event set 1 results



hpmavg Output for hpmcount POP Exercise

```
#####
hpmcount (v 2.4.2) summary (aggregate of 2 POE tasks)
```

Section-independent statistics

```
Maximum execution time (wall clock time)      : 152.18 seconds
Average execution time (wall clock time)       : 152.18 seconds
Average amount of time in user mode            : 139.88 seconds
Average amount of time in system mode          : 5.83 seconds
Total maximum resident set size                : 1429 Mbytes
Total shared memory use in text segment        : 37312.27 Mbytes*sec
Total unshared memory use in data segment       : 4194304.00 Mbytes*sec
```

Section-specific statistics

```
Section: __DEFAULT__
```



More hpmavg Output for POP

PM_CYC (Cycles)	: 55120 M
PM_INST_CMPL (Instructions completed)	: 47896 M
PM_TLB_MISS (TLB misses)	: 136 M
PM_ST_CMPL (Stores completed)	: 6039 M
PM_LD_CMPL (Loads completed)	: 17551 M
PM_FPU0_CMPL (FPU 0 instructions)	: 11316 M
PM_FPU1_CMPL (FPU 1 instructions)	: 5829 M
PM_EXEC_FMA (FMAs executed)	: 8028 M
Average utilization rate	: 90.000 %
Avg number of loads per TLB miss	: 128.649

Remaining hpmavg Output for POP

Load and store operations	: 23590 M
Avg load/store intensity	: 0.493
Total Instruction rate (MIPS)	: 314.74 MIPS
Average Instruction rate (MIPS)	: 157.37 MIPS
Instructions per cycle	: 0.869
HW Float points instructions per Cycle	: 0.311
Total Floating point instructions + FMAs	: 25174 M
Total Float point instructions + FMA rate	: 165.43 Mflip/s
Average Float point instructions + FMA rate:	82.71 Mflip/s
Average FMA percentage	: 63.783 %
Average computation intensity	: 1.0675

#####

Level 1 MPI Performance Data: mpiP

☞ mpiP Basics

- ☞ Developed by Jeff Vetter (and others) at LLNL
 - ☞ <http://www.llnl.gov/casc/mpip>
- ☞ Easy to use tool
 - ☞ Statistical-based MPI profiling library
 - ☞ Requires relinking but no source level changes
 - ☞ Compiling with “-g” is recommended
- ☞ Provides average times for each MPI call site
- ☞ Has been shown to be very useful for scaling analysis

☞ mpiP on NERSC machines

- ☞ Must use thread-safe libraries, etc. (i.e., use mp*_r compiler)
- ☞ Use mpiP module - defines mpiP environment variable
 - ☞ Provides required library path and libraries
 - ☞ Add “\$(mpiP)” to link line in makefile
 - ☞ Same as adding “-L\${mpiP_root}/lib -lmpiP -lbfd -liberty –lintl”



mpiP Exercises

- ✍ Type “module load mpiP<cr>”
- ✍ Type “module load scicomp8/simple.mpiP<cr>”
- ✍ Type “cd ~/simple.mpiP<cr>”
- ✍ Type “make<cr>” – makes both examples
- ✍ **Simple Exercise I: Student body right**
 - ✍ Type “more hot-potato.c<cr>”
 - ✍ Type “llsubmit hot-potato.ll<cr>”
 - ✍ Type “more hot-potato.*.mpiP<cr>”
 - ✍ Make clean, add “-g” and try again
- ✍ **Simple Exercise II: Mix of MPI call sites**
 - ✍ Type “more medley.c<cr>”
 - ✍ Type “llsubmit medley.ll<cr>”
 - ✍ Type “more medley.*.mpiP<cr>”



More mpiP Exercises

Exercise: POP MPI performance

-  Type “module load scicomp8/pop.mpiP<cr>”
-  Type “cd ~/pop.mpiP<cr>”
-  Includes LoadLeveler script: “pop.mpiP.ll”
-  Type “more mpiP.out<cr>” to view output:

```
@ mpiP
@ Command : ./pop
@ Version          : 2.4
@ MPIP Build date : Jul 18 2003, 11:41:57
@ Start time       : 2003 07 18 15:01:16
@ Stop time        : 2003 07 18 15:03:53
@ MPIP env var    : [null]
@ Collector Rank   : 0
@ Collector PID    : 25656
@ Final Output Dir: .
@ MPI Task Assignment: 0 h0107.nersc.gov
@ MPI Task Assignment: 1 h0107.nersc.gov
```



More mpiP Output for POP

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

Task	AppTime	MPITime	MPI%
0	157	1.89	1.21
1	157	6.01	3.84
*	313	7.91	2.52

@--- Callsites: 6 -----

ID	Lev	File	Line	Parent_Funct	MPI_Call
1	0	global_reductions.f	0	??	Wait
2	0	stencils.f	0	??	Waitall
3	0	communicate.f	3122	.MPI_Send	Cart_shift
4	0	boundary.f	3122	.MPI_Send	Isend
5	0	communicate.f	0	.MPI_Send	Type_commit
6	0	boundary.f	0	.MPI_Send	Isend



Still More mpiP Output for POP

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

Call	Site	Time	App%	MPI%
Waitall	4	2.22e+03	0.71	28.08
Waitall	6	1.82e+03	0.58	23.04
Wait	1	1.46e+03	0.46	18.41
Waitall	2	831	0.27	10.51
Allreduce	1	499	0.16	6.31
Bcast	1	275	0.09	3.47
Isend	2	256	0.08	3.24
Isend	4	173	0.06	2.18
Barrier	1	113	0.04	1.43
Irecv	2	80.3	0.03	1.01
Irecv	4	40.6	0.01	0.51
Cart_create	3	28	0.01	0.35
Cart_coords	3	17.4	0.01	0.22
Type_commit	5	12.7	0.00	0.16



Remaining mpiP Output for POP

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

Isend	1	12.7	0.00	0.16
Bcast	3	12.4	0.00	0.16
Barrier	5	12.2	0.00	0.15
Cart_shift	5	12	0.00	0.15
Irecv	1	10.7	0.00	0.13
Isend	6	9.28	0.00	0.12

@--- Callsite statistics (all, milliseconds): 53 -----

Name	Site	Rank	Count	Max	Mean	Min	App%	MPI%
Allreduce	1	0	1121	2.35	0.182	0.079	0.13	10.79
Allreduce	1	1	1121	11.1	0.263	0.129	0.19	4.90
Allreduce	1	*	2242	11.1	0.222	0.079	0.16	6.31

