Profiling and Optimization of Climate, Ocean and Weather Codes on Large Clusters.

Hans Joraandstad

Sun Solution Center for HPC

Sun Microsystems

13th Workshop on the Use of High Performance Computing in Meteorology

Purpose

- Discuss requirements and issues for profiling
- Describe a tool and method used
- Describe profiling information obtained
- Show examples of optimizations based on profiles
- Summary and recommendations

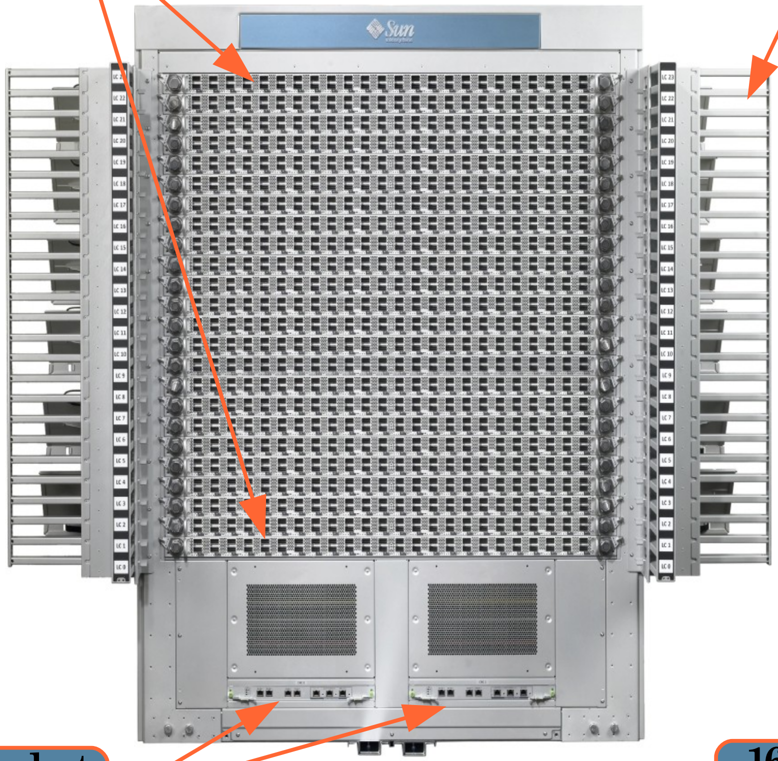
Agenda

- Purpose
- Sun Products, quick look
- Sun Solution Center for HPC, where I work
- Requirements and issues for profiling
- The tools and methods used
- What you get
- Examples
- Summary/Recommendations

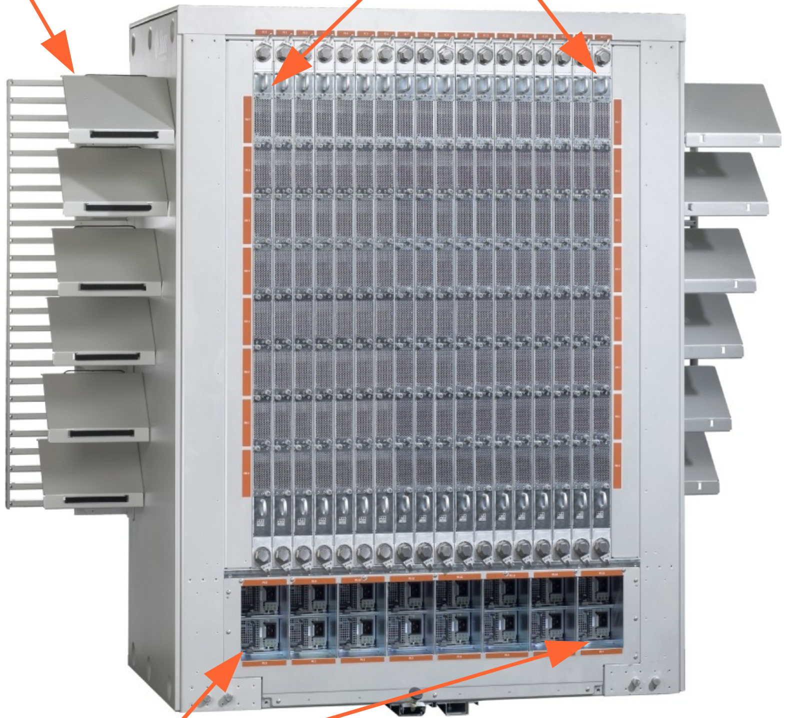
Sun Products, quick look

- AMD
- Intel
- Sparc
- Switches
- Racks
- Software

Sun Datacenter Switch 3456

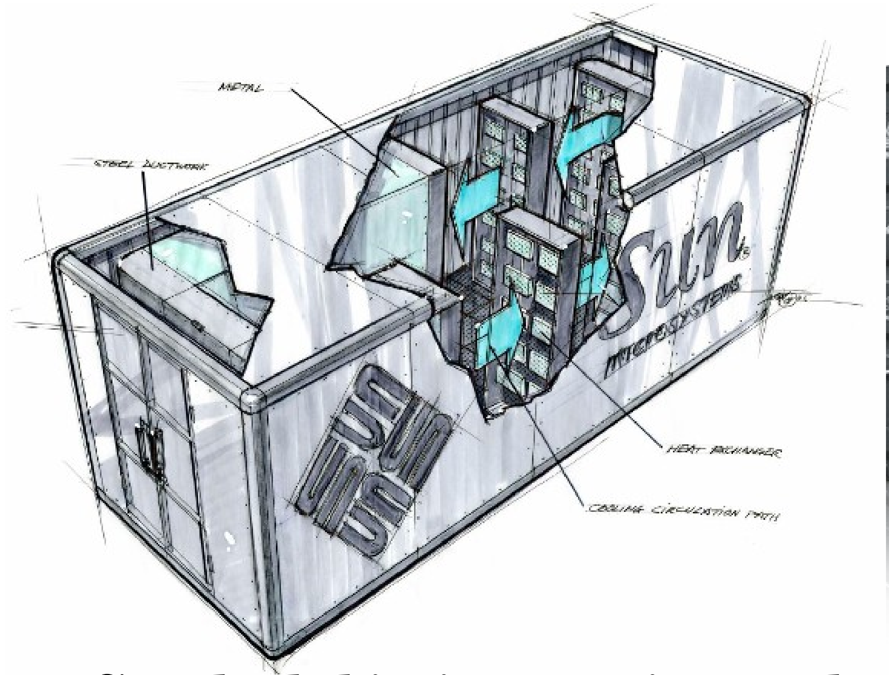


Sun DS 3456 Front View



Sun DS 3456 Rear View

Project Blackbox:
The Virtualized Datacenter



- Standard shipping container packaged with eight standard racks
- Integrated, high-efficiency power and cooling
- Supports a wide range of compute, storage and network infrastructure – build once, deploy anywhere when fully configured

Sun Solution Center for HPC

Located in Hillsboro, Oregon, USA

- Over 10 Teraflops deployed
  - > More than 600 x64 and UltraSPARC nodes
  - > Continually refreshed (located next to the factory)
  - > Built with Sun Grid Rack Systems
- Available for:
  - > Proofs of Concept
  - > Benchmarks
  - > Scalability testing
  - > Risk mitigation
- Leverages years of HPC expertise from specialists all over Sun

Why Profiling

- Where is the time spent
- To adapt or optimize code for a given architecture
  - > Different cpus, memory, MPI stack, OS, FS
- Find bottlenecks
  - > In computation, MPI communication and/or IO
- Study, predict and/or improve scalability
  - > What scales and if not then why not

Requirements and Issues in Profiling

- Choice of compiler and/or MPI stack (and OS)
  - > Built-in profiling tools vary (cost and learn to use)
  - > One tool for all?
- Use production version binary
  - > No recompilation should be necessary
  - > -g means provide debug info but still optimize
- Use in potentially complex run-environments
  - > Job scripts
  - > mpirun wrapper scripts

Requirements and Issues in Profiling

- Use with varying (and large) number of processes
  - Different behavior as #processes vary
  - To study scaling issues
- Must be non-intrusive
  - Little or no addition to elapsed time
  - Serious issue when used with large process counts
- Provide as much details as possible in one run

Sun Studio Performance Tools

- Part of Studio 12 Compiler Suite
- Can be downloaded for free
- Has collector that understands many MPI stacks
- Has a simple collect command (use like time)
- Has CLI and GUI analyzers
- In one run you can get
  - > Function level profile
  - > Gprof type caller-callee profile
  - > Annotated source with timers per line
  - > GUI can give you timeline view of call-stack

How to use/Method

- Non-intrusive requirement means selective profiling
- Modify or add wrapper script that
  - Finds the rank (from env. Variables)
  - Potentially decides how to bind (rank to core)
  - Start the local MPI process via
    - EXE=<executable or $* if passed as argument>
    - exec $EXE
    - exec ${BIND_ME} $EXE
    - exec [${BIND_ME}] $COLLECT $EXE for 1 or 2 ranks
- mpirun <arguments> $wrapper $executable .....

How to use/Method

- The collect command in its simplest form
  - > collect <executable> ......
  - > Collect without any options gives help
  - > Name experiment if profiling more than 1 rank
- To analyze
  - > er_print test.1.er
  - > analyzer test.1.er test.2.er .....

How to use/Method

- Used successfully with
- Sun ClusterTools, ScaliMPI, IntelMPI, OpenMPI, MPICH. Should work ok for other MPI stacks.
- Sun Studio, Intel, PGI, PathScale, GNU compilers
- Solaris, Solaris x86, SuSE, RedHat
  - There are some version limitations for Linux
- Sparc, Intel and AMD

What you get

- Function level profile
  - Functions in load objects can be 'grouped'
- Caller-callee (gprof) profile
- You can set time window to restrict to 'steady state' or to one 'time step'
- Annotated source with timers per line (-g)
- GUI (analyzer) gives timeline view
- Elapsed time increase very little if at all

What you get: Function level profile

WRF CHEM with 384 processes, IntelMPI, Harpertown, process 0 (+8s)

Excl. User CPU	Incl. User CPU	Function name
370.73	370.73	<Total>
128.6	130.1	MPIDI_CH3I_read_progress_expected
34.6	47.74	module_mp_thompson_mp_thompson_init_
12.09	12.09	sched_yield
11.69	11.69	patch_2_outbuf_r_
11.38	11.38	pow.L
9.91	9.91	module_mosaic_therm_mp_aerosol_water_
7.84	7.84	__powr8i4
7.44	21.18	module_mosaic_therm_mp_compute_activities_
7.01	35.59	module_mosaic_therm_mp_mesa_flux_salt_
6.9	6.94	<static>@0xb0db
6.75	6.75	ext_ncd_support_routines_mp_transpose_
6.54	11.46	module_mp_thompson_mp_qr_acr_qg_
6.51	6.51	module_advect_em_mp_advect_scalar_
6.25	6.25	f_unpack_int_
6.05	6.06	<static>@0xb16b
5.98	5.98	f_pack_int_
5.5	5.5	__I_MPI___intel_new_memcpy

What you get: Caller-callee

WRF CHEM with 384 processes, IntelMPI, Harpertown, process 0 (+8s)

Attr. User CPU sec.	Excl. User CPU sec.	Incl. User CP sec	Name
20.38	7.01	35.59	module_mosaic_therm_mp_mesa_flux_salt_
0.8	0.03	1.17	module_mosaic_therm_mp_astem_flux_wet_
7.44	7.44	21.18	*module_mosaic_therm_mp_compute_activities_
9.91	9.91	9.91	module_mosaic_therm_mp_aerosol_water_
2.32	7.84	7.84	__powr8i4
1.52	11.38	11.38	pow.L

What you get: Annotated source

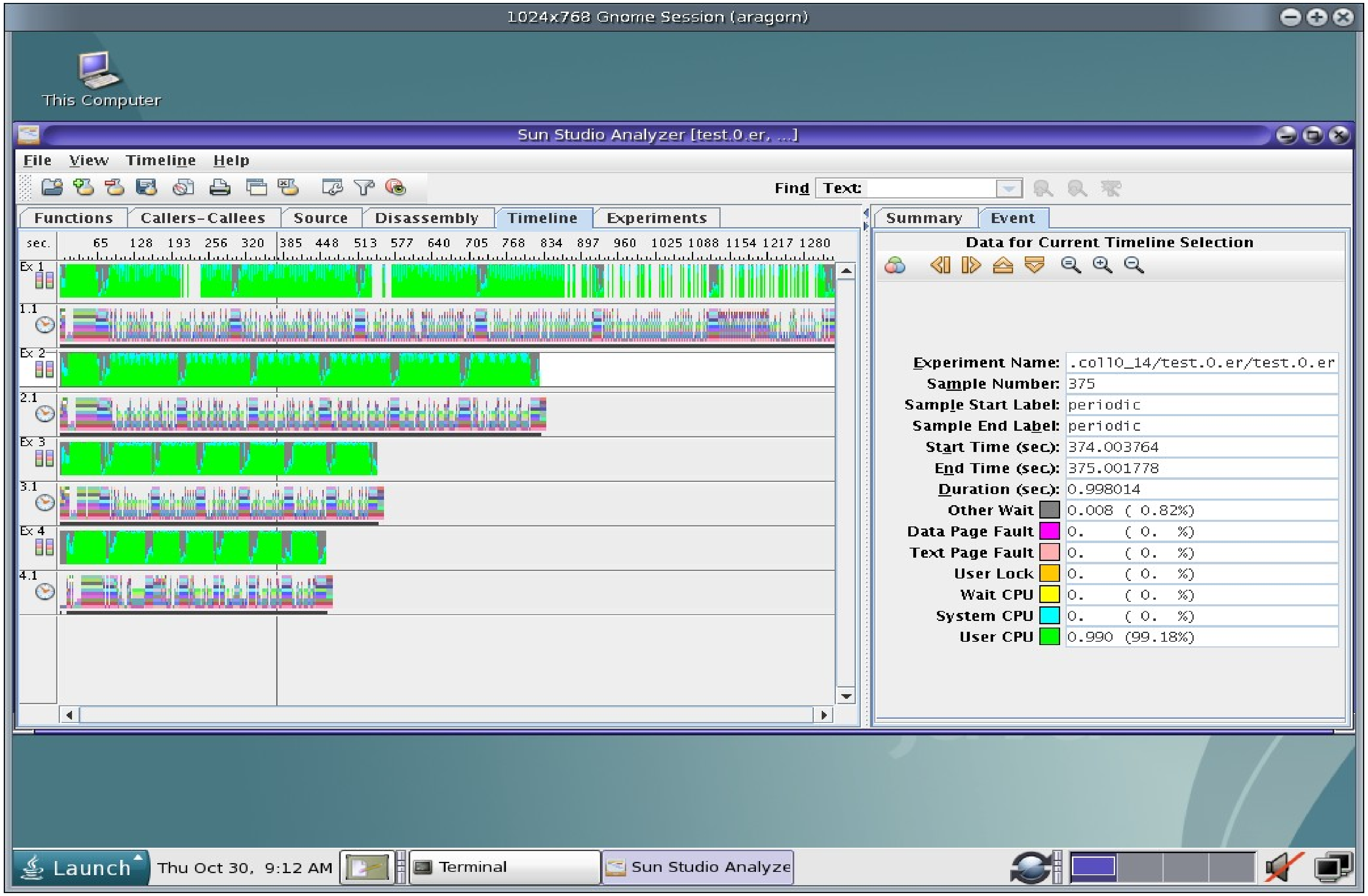
WRF CHEM with 384 processes, IntelMPI, Harpertown, process 0 (+8s)

```
      Excl      Incl      Line     Source
         0.03      0.03    11445 real(kind=8) function aerosol_water(jp,ibin) ! kg (water)/m^3 (air)
                                 <Function: module_mosaic_therm_mp_aerosol_water_>
                           11446 ! implicit none
                           11447 ! include mosaic.h
                           11448 ! subr. arguments
                           11449      integer jp, ibin
                           11450 ! local variables
                           11451      integer je
                           11452      real(kind=8) dum
                           11453 ! function
                           11454 ! real(kind=8) bin_molality
                           11455
                           11456
                           11457
         0.01      0.01    11458     dum = 0.0
                           11459     do je = 1, (nsalt+4) ! include hno3 and hcl in water calculation
##       8.88      8.88    11460       dum = dum + 1.e-9*electrolyte(je,jp,ibin)/bin_molality(je,ibin)
         0.87      0.87    11461     enddo
                           11462
                           11463     aerosol_water = dum
```

What you get: Timeline with GUI

- No still pictures can show the true value of
  - The GUI's possibilities and ease of use
  - The GUI timeline displays
- If you go to SC2008, visit Sun's booth

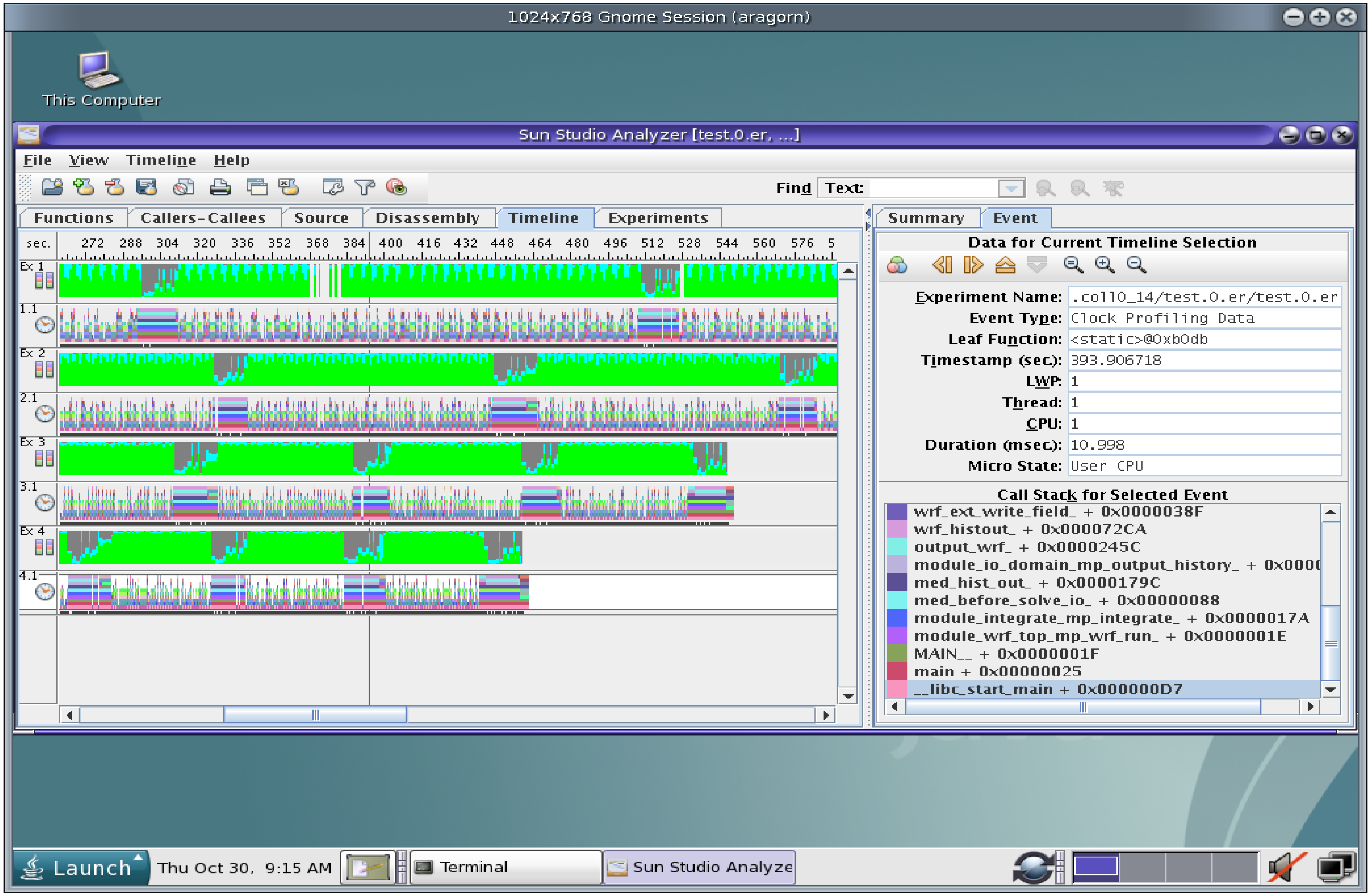
WRF CHEM 64-384, IntelMPI & HPTN


1024x768 Gnome Session (aragorn)
This Computer
Sun Studio Analyzer [test.0.er, ...]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Timeline Experiments
Summary Event
sec. 65 128 193 256 320 385 448 513 577 640 705 768 834 897 960 1025 1088 1154 1217 1280
Ex 1
1.1
Ex 2
2.1
Ex 3
3.1
Ex 4
4.1
Launch
Thu Oct 30, 9:12 AM
Terminal
Sun Studio Analyze

Data for Current Timeline Selection

Experiment Name:	.coll0_14/test.0.er/test.0.er
Sample Number:	375
Sample Start Label:	periodic
Sample End Label:	periodic
Start Time (sec.):	374.003764
End Time (sec.):	375.001778
Duration (sec.):	0.998014
Other Wait ⬛	0.008 (0.82%)
Data Page Fault 🟪	0. (0. %)
Text Page Fault 🟥	0. (0. %)
User Lock 🟧	0. (0. %)
Wait CPU 🟨	0. (0. %)
System CPU 🟦	0. (0. %)
User CPU 🟩	0.990 (99.18%)

WRF CHEM 64-384 Timeline Zoomed


1024x768 Gnome Session (aragorn)
This Computer
Sun Studio Analyzer [test.0.er, ...]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Timeline Experiments
Summary Event
sec. 272 288 304 320 336 352 368 384 400 416 432 448 464 480 496 512 528 544 560 576
Ex 1
1.1
Ex 2
2.1
Ex 3
3.1
Ex 4
4.1
Launch
Thu Oct 30, 9:15 AM
Terminal
Sun Studio Analyze

Data for Current Timeline Selection

Experiment Name:	.coll0_14/test.0.er/test.0.er
Event Type:	Clock Profiling Data
Leaf Function:	<static>@0xb0db
Timestamp (sec):	393.906718
LWP:	1
Thread:	1
CPU:	1
Duration (msec):	10.998
Micro State:	User CPU

Call Stack for Selected Event

```
wrf_ext_write_field_ + 0x0000038F
wrf_histout_ + 0x000072CA
output_wrf_ + 0x0000245C
module_io_domain_mp_output_history_ + 0x0000
med_hist_out_ + 0x0000179C
med_before_solve_io_ + 0x00000088
module_integrate_mp_integrate_ + 0x0000017A
module_wrf_top_mp_wrf_run_ + 0x0000001E
MAIN__ + 0x0000001F
main + 0x00000025
__libc_start_main + 0x000000D7
```

Examples

- NEMO
  - Small case
  - Profiler identifies computational improvements
  - Optimization easy with big win
- ROMS
  - Profiler identifies source of excessive MPI overhead
  - Optimization not easy but improved scaling
- WRF CHEM
  - Profiler used in scaling study

NEMO example (AMD 8356 4S4C)

Intel compiler, Scali MPI, 16 process run

Without profiling	8m44
With profiling	8m43

Excl. User CPU	Incl. User CPU	Name
501.24	501.24	<Total>
207.37	207.97	prtctl_mp_prt_ctl_
36.68	57.31	traadv_muscl_mp_tra_adv_muscl_
29.27	44.59	ldfslp_mp_ldf_slp_
25.68	25.68	traldf_iso_mp_tra_ldf_iso_
22.42	22.42	dynzdf_imp_mp_dyn_zdf_imp_
19.26	19.26	trazdf_imp_mp_tra_zdf_imp_
15.72	17.58	dynldf_bilap_mp_dyn_ldf_bilap_
9.23	18.1	dynspg_flt_mp_dyn_spg_flt_
8.76	8.77	diawri_mp_dia_wri_
7.09	15.18	lib_mpp_mp_mpp_lnk_3d_
6.8	15.16	dynzad_mp_dyn_zad_

NEMO example

- From source file of prtctl we see
  - 6 local arrays (tables and masks) set to 0 or 1
  - Same arrays set to values from arguments if present
  - A do loop with complex index computation followed by SUMs of products of tables and masks + printout
  - The initialization seems a bit heavy, like
    - zmask1(:,:,:) = 1.e0
    - IF(PRESENT(mask1)   ) zmask1 (:,:,:)= mask1 (:,:,:)
- but better to get annotated source

NEMO Example

Snippets from annotated source

	Excl	Incl	Line	Code
	15.13	15.13	110	ztab3d_1(:,:,:) = 0.e0
##	39.61	39.61	112	zmask1 (:,:,:) = 1.e0
##	31.09	31.09	121	IF(PRESENT(tab3d_1)) ztab3d_1(:,:,:)= tab3d_1(:,:,:)
	24.76	24.76	122	IF(PRESENT(tab3d_2)) ztab3d_2(:,:,:)= tab3d_2(:,:,:)
##	34.09	34.09	123	IF(PRESENT(mask1)) zmask1 (:,:,:)= mask1 (:,:,:)
	19.73	19.73	124	IF(PRESENT(mask2)) zmask2 (:,:,:)= mask2 (:,:,:)

Most of the time spent initializing and copying data

So remove unnecessary use of local arrays

(Essentially removing memory intensive code)

NEMO Example

Compare profiles from original and optimized runs

Original	Optimized	Function_name
501.2	322.3	<Total>
207.4	32.0	prtctl_mp_prt_ctl_
36.7	37.3	traadv_muscl_mp_tra_adv_muscl_
29.3	29.4	ldfslp_mp_ldf_slp_
25.7	26.2	traldf_iso_mp_tra_ldf_iso_
22.4	22.4	dynzdf_imp_mp_dyn_zdf_imp_
19.3	18.1	trazdf_imp_mp_tra_zdf_imp_
15.7	16.2	dynldf_bilap_mp_dyn_ldf_bilap_
9.2	9.6	dynspg_flt_mp_dyn_spg_flt_
8.8	8.8	diawri_mp_dia_wri_
7.1	7.1	lib_mpp_mp_mpp_lnk_3d_
6.8	6.8	dynzad_mp_dyn_zad_

Original run 8m43

Optimized run 5m43

2 day simulation, full run was 31 days, saves 45m
Side effect: Better scaling to more processes

ROMS 3.0 Example

- Very heavy MPI overhead -> negative scaling
  - 128 processes 577s
  - 192 processes 513s
  - 256 processes 508s
  - 384 processes 527s
- Timings from reduced simulation runs
- AMD Opteron 2S2C, PGI, ScaliMPI

ROMS 3.0 Example

- ScaliMPI showed which MPI calls took time
- Identified via gprof caller-callee profiles
  - > (Using trace back from MPI library usage)
  - > The code doing send/receive/wait
  - > The code doing gather operations
- The gather was related to 'station data'
  - > Many attributes collected and written for all stations
  - > Each attribute involved a large global gather
- Rewrite complicated due to lots of #ifdefs
- Initial profiles gone (so no pictures!)

ROMS 3.0 Example

Example from gprof AFTER the optimization (128 procs)

Attr	Excl	Incl	Name
7.91	0.61	8.56	mp_boundary__
2.24	0.07	2.33	mp_collect__
1.32	0	1.32	mp_reduce__
0.2	0	0.2	extract_sta_mod_extract_stacoll__
0	0	11.68	*mpi_allgather
11.68	0	11.68	MPI_Allgather

Before optimization extract_sta_mod was > 40s

ROMS 3.0 Example

- 2 source files were modified
- A sequence (for set of attributes) of collect (gather) and write (from process 0) changed to
  - First time thru sequence
    - Each process saves local station data contribution in buffer
  - At end of sequence, gather all buffers to master
  - Repeat sequence, only process 0 doing something
    - Get data out of buffer and write it
  - The process is complicated due to some attributes can get contributions from several processes (if station is close to process border)

ROMS 3.0 Example

- Timings with optimized station collect/write

> 128	processes	530	from	577
> 192	processes	403	from	513
> 256	processes	330	from	508
> 384	processes	309	from	527

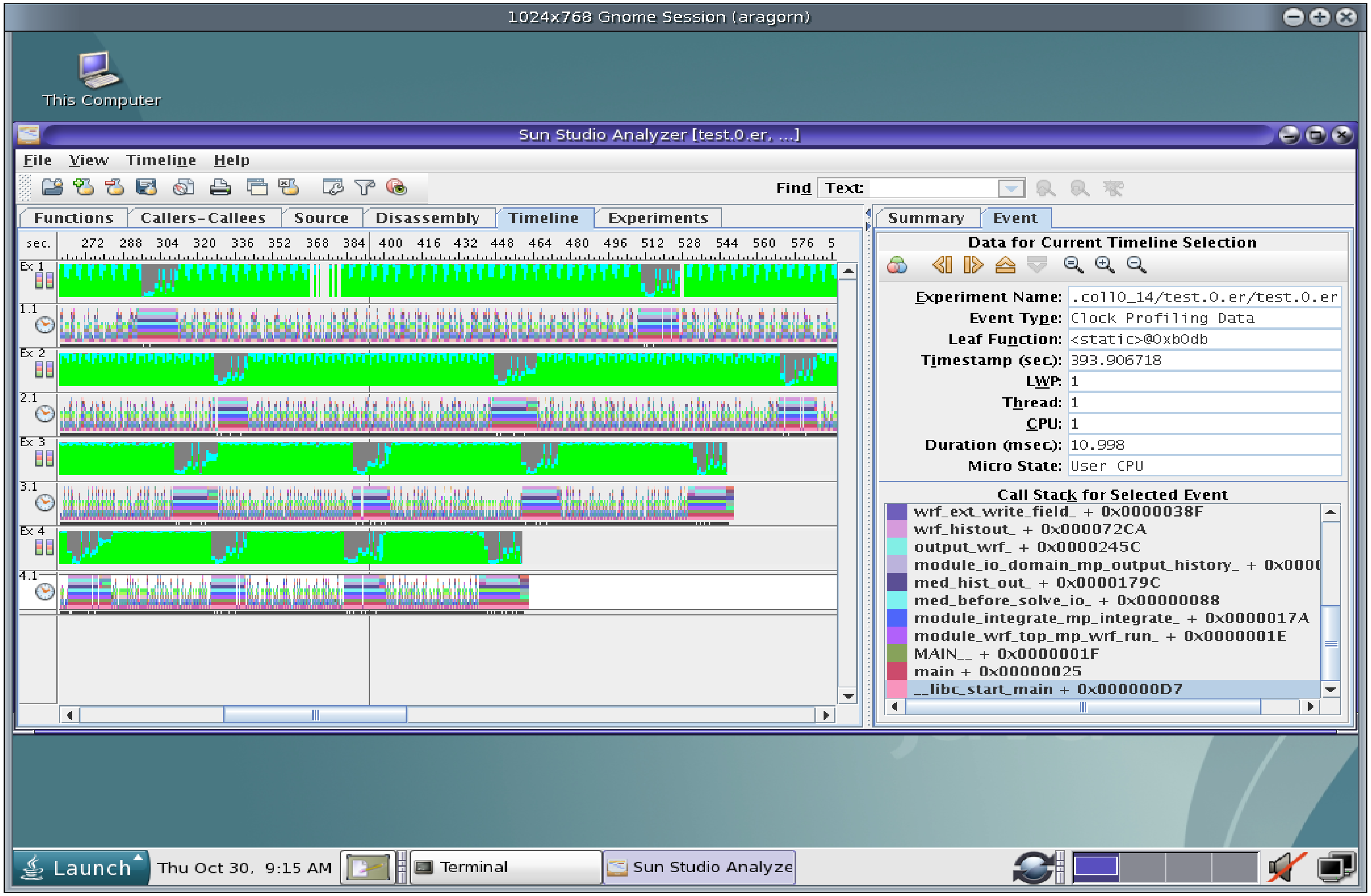
- Other MPI ovehead improved in ROMS3.x
- Code modifications to be sent to ROMS

Example: WRF CHEM Scaling Study

64	128	256	384	Function or library
1238.5	740.7	463.8	370.7	<Total>
254.6	130.2	38.5	15.6	<libc-2.3.4.so>
166.8	173.1	158.1	150	<libmpi.so.3.2>
74.1	34.2	18.4	11.4	pow.L
71.8	32.2	15.5	9.9	module_mosaic_therm_mp_aerosol_water_
49.1	22.8	11.3	7.4	module_mosaic_therm_mp_compute_activities_
48.2	22	11.4	7	module_mosaic_therm_mp_mesa_flux_salt_
40.6	18.9	8.7	5.5	module_mosaic_therm_mp_mesa_ptc_
34.8	35.1	34.9	34.6	module_mp_thompson_mp_thompson_init_
32.2	14.8	6.6	3.8	module_mosaic_therm_mp_calc_dry_n_wet_aerosol_props_
31.5	15.2	7.5	4.7	module_mosaic_therm_mp_ions_to_electrolytes_
29.3	14.5	8.9	6.5	module_advect_em_mp_advect_scalar_
26.7	13.4	5.6	3.8	module_mosaic_therm_mp_mesa_estimate_eleliquid_
25.9	12.5	5.9	3.7	module_cbmz_rodas_prep_mp_cbmz_v02r02_decomp_
22.4	9.6	4.9	3.1	module_mosaic_coag_mp_coagsolv_
21.6	9.9	4.8	2.7	module_mosaic_therm_mp_aerosol_phase_state_
21.5	12.4	9.7	7.8	__powr8i4
20.3	13	7.3	6.2	f_unpack_int_

Intel Harpertown, IntelMPI, Intel compiler, 6H simulation

WRF CHEM 64-384 Timeline Zoomed


1024x768 Gnome Session (aragorn)
This Computer
Sun Studio Analyzer [test.0.er, ...]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Timeline Experiments
sec. 272 288 304 320 336 352 368 384 400 416 432 448 464 480 496 512 528 544 560 576 5
Ex 1
1.1
Ex 2
2.1
Ex 3
3.1
Ex 4
4.1
Summary Event
Launch
Thu Oct 30, 9:15 AM
Terminal
Sun Studio Analyze

Data for Current Timeline Selection

Experiment Name:	.coll0_14/test.0.er/test.0.er
Event Type:	Clock Profiling Data
Leaf Function:	<static>@0xb0db
Timestamp (sec):	393.906718
LWP:	1
Thread:	1
CPU:	1
Duration (msec):	10.998
Micro State:	User CPU

Call Stack for Selected Event

```
wrf_ext_write_field_ + 0x0000038F
wrf_histout_ + 0x000072CA
output_wrf_ + 0x0000245C
module_io_domain_mp_output_history_ + 0x0000
med_hist_out_ + 0x0000179C
med_before_solve_io_ + 0x00000088
module_integrate_mp_integrate_ + 0x0000017A
module_wrf_top_mp_wrf_run_ + 0x0000001E
MAIN__ + 0x0000001F
main + 0x00000025
__libc_start_main + 0x000000D7
```

Compare WRF 2S2C vs 2S4C

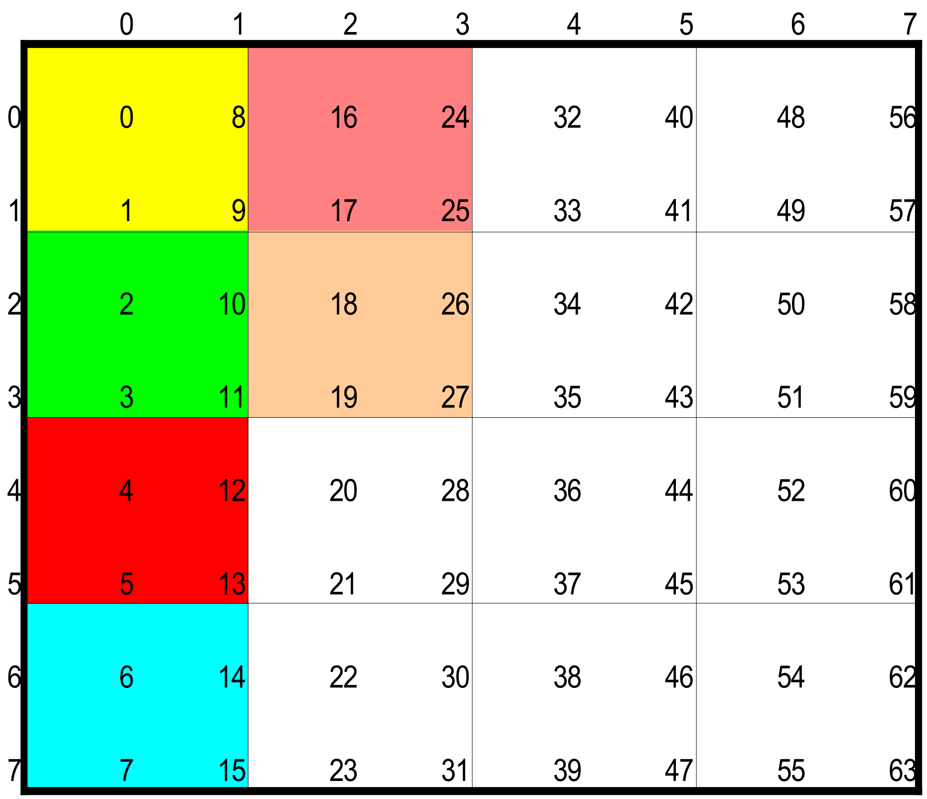
2S2C	2S4C	
1585.3	3154.9	<Total>
315.4	334.1	<libmpi.so>
83	211	module_small_step_em_mp_advance_w_
82.1	183.6	module_small_step_em_mp_advance_uv_
71.8	187.7	module_small_step_em_mp_advance_mu_t_
67.8	215.7	rsl_lite_pack_
63.5	141.4	module_advect_em_mp_advect_scalar_
61.5	136.2	module_small_step_em_mp_calc_p_rho_
56	145.2	module_small_step_em_mp_small_step_prep_
51.4	92.9	module_mp_etanew_mp_egcp01drv_
44.1	135.6	module_small_step_em_mp_sumflux_
39.6	38.2	__svml_powf4.A
39.1	48.6	module_bl_ysu_mp_ysu2d_
34.6	100	module_big_step_utilities_em_mp_curvature_
34.1	96.3	module_big_step_utilities_em_mp_zero_tend_
32.1	84.6	module_big_step_utilities_em_mp_horizontal_pressure_gra
28.5	75.5	module_big_step_utilities_em_mp_rhs_ph_
28.4	71	module_em_mp_rk_update_scalar_
27.8	81.2	module_em_mp_rk_addtend_dry_
25.1	53	module_big_step_utilities_em_mp_phy_prep_
24.9	17.5	module_ra_rrtm_mp_rtrn_
24.5	28.5	<libc-2.4.so>

Default rank assignment (4C)

	0	1	2	3	4	5	6	7
0	0	8	16	24	32	40	48	56
1	1	9	17	25	33	41	49	57
2	2	10	18	26	34	42	50	58
3	3	11	19	27	35	43	51	59
4	4	12	20	28	36	44	52	60
5	5	13	21	29	37	45	53	61
6	6	14	22	30	38	46	54	62
7	7	15	23	31	39	47	55	63

Rank 11 in node 3 talks to ranks 3 10 12 and 19, these are in nodes 1 3 4 5, thus communicating with 3 nodes

Reordered rank assignment



	0	1	2	3	4	5	6	7
0	0	8	16	24	32	40	48	56
1	1	9	17	25	33	41	49	57
2	2	10	18	26	34	42	50	58
3	3	11	19	27	35	43	51	59
4	4	12	20	28	36	44	52	60
5	5	13	21	29	37	45	53	61
6	6	14	22	30	38	46	54	62
7	7	15	23	31	39	47	55	63

Rank 11 in node 2 talks to ranks 3 10 12 and 19, these are in nodes 2 2 3 5, thus communicating with 2 nodes.

All ranks have 2 neighbors in same node!

Improves elapsed time with 2-3% (64-384 procs)

Summary

- Described profiler tool that is
  - Easy to integrate/use even if complex runscripts
  - Is non-intrusive (use selectively!)
  - Provides a lot of details for most compilers & MPIs
  - Is free, runs on Solaris and Linux
- To see more, visit SC2008 Sun Booth
- Visit www.sun.com

Recommendations

- Use selective MPI profiling
- Use specific binding, dont leave it to the tools
  - > (or be vary of the tools!)
- If possible, reorder machinefile if neighbor comm.
  - > This complicates the binding scripts!
- OpenMP and MPI hybrid models
  - > Profiling is a must

Profiling and Optimization of Climate, Ocean and Weather Codes on Large Clusters.

Hans.Joraandstad@Sun.COM