It's About Time
Multi-Resolution Timers for Scalable Performance Debugging

James B. White III (Trey)
trey@ornl.gov

NATIONAL CENTER
FOR COMPUTATIONAL SCIENCES

OAK RIDGE NATIONAL LABORATORY

Acknowledgement

Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

Motivation

- A goal of high-performance computing is to accelerate computation
  - Developers must optimize software performance
  - Optimization effort guided by timing
- Main strategies for instrumenting software
  - Automated tools
  - Explicit calls to timing libraries
- Each strategy has (dis)advantages

Automated tools

+ No changes to source code
+ Wide range of analysis capabilities
– Hardware-vendor dependent
– Or unavailable on newest supercomputers
– Timing data for highly parallel runs can overwhelm file systems and post-processing tools

Example: CrayPAT

- Automated tools on Cray systems
  - Automatically instrument executables
  - Run application to generate performance-data file
  - Generate reports or analyze with interactive GUI
- Performance-data file grows linearly with task count
  - Serial post-processing tools run out of memory for runs of a few thousand tasks (4500-core POP run)
  - Unusable for large runs (up to 23,000 cores today)

Timing library

+ More portable
  + Library distributed with source
+ Lower overhead
  + Target specific regions of software
- Simple text output
  + Easy to understand
  – **Inadequate to diagnose performance problems**
– Must change source code

Example: CCSM

- Community Climate System Model
  - Separate components for atmosphere, ocean, land, and sea ice
  - Each component uses separate groups of processors
- CCSM timers for load balancing of components
  - Single text file
  - Maximum time per task for each timer, with additional analysis for load balancing
  - Not adequate for optimizing each component

Example: CCSM

- Community Atmosphere Model (CAM)
  - Separate timer file for each task
  - Huge volume, and still inadequate to characterize performance variability
- Parallel Ocean Program (POP)
  - Short report included in standard output
  - Only min, max, and avg for each timer
  - Obscures load imbalance and performance variability

Reduced multi-resolution timers

- Designed to find load imbalance and performance variability in POP
- Variety of data reductions
- Multi-resolution histograms
- Reduced
  - "Similar" histograms are combined

Timer interface?

- Always refer to each timer by a string
  - `call start_timer("physics")`
  - `call stop_timer("physics")`
- Easy to use
  - No declarations
  - No argument passing
- Overhead
  - String matching and search (or hash) for every call
  - Startling overhead for vector systems

Timer interface?

- Use a type or integer handle
  - `type(timer) :: t`
  - `t = new_timer("physics")`
  - `call start_timer(t)`
  - `call stop_timer(t)`

Atop call stack or in separate module

In many different procedures

- Efficient
  - Minimal overhead on start and stop
- Inconvenient
  - Must deal with timer initialization
  - Proliferation of timer arguments or "`use`" statements

Timer interface, a compromise

- Use local integer IDs with "`save`" attribute
  - `integer, save :: t = 0`
  - `call get_timer(t, "physics")`
  - `call start_timer(t)`
  - `call stop_timer(t)`

All within the procedure of interest

- First call to "`get_timer`" does search or creates a new timer, assigns it to "`t`"
- Subsequent calls to "`get_timer`" just check that "`t`" corresponds to name "`"physics"`"
  - One string compare
- All local, no arguments or separate initialization

Timer overhead

- `get_timer(i,name)`
  - Initialize timer variables on very first call
  - Linear search through timer names if "`i`" isn't set
  - If name isn't found, create new timer
    - Lightweight if space exists
    - Otherwise reallocate and copy all timers
  - If "`i`" is set, confirm name is right
- `start_timer`
  - Check argument and timer state, two "`if`" statements
  - `system_clock`
- `stop_timer`
  - `system_clock`
  - Check argument and timer state
  - Update ten state variables
- No measurement of overhead, but easy to add

Ten state variables?

- Turn off timer (1)
- Increment timer event count (2)
- Increase timer total ticks (3)
- Update max ticks (4) and max event (5)
- Update min ticks (6) and min event (7)
- Update 2nd max (8) and 2nd-max event (9)
- Find histogram bin and increment (10)

2nd max?

- Second-highest tick count for a single event
- Suggested by Pat Worley
- Useful when one event includes application initialization work
  - Can still measure variability of non-init events
- Might be useful for other types of singleton outliers

Event histograms

- Bins count events in particular tick ranges
- Bins for tick counts/ranges:
  - 0,1,2,3,4,5,6,7,8,9
  - 10-19,20-29,…,90-99
  - 100-199,200-299,…,900-999
  - 1000-1999,…,9000-9999
  - …, "`count_max`"

  } *Multi-resolution*

- Finding a bin
  - Linear search, starting from smallest tick range
    - One "`if`" test for 0-9 ticks, two for 10-99, *etc.*
  - One integer division

Reporting

- `call report_timers(unit,comm)`
- Timers are local to each MPI task
- No parallel communication needed until report
- A given timer may be unique to one task or subsets of tasks within communicator
- Timers on separate tasks with same string name considered the “same” timer
- First step of report: global name list

Global name list

- Each task sorts local timer names (alphabetical order)
- Binary tree of MPI communication
  - Receive list
  - Merge with own, removing repeats
  - Send resulting list to parent
- Root broadcasts final sorted list
- Each task creates map from global-name indices to local-timer indices
- Some global names may not exist locally

Report

- Summary
  - Basic profile (table of contents)
- Details for each timer
  - Statistics
  - Reduced histograms

Summary report

- Max, min, avg of each timer *per task*
  - Max = largest total time for a single MPI task
  - Max ≠ greatest aggregate time
- Output sorted by this max
- Each task performs sort using data from "`mpi_allreduce`"
- Results gathered at root using "`mpi_reduce`"

Summary report from POP

```
*** profile for 360 tasks ***
 max total | min total | avg total | timer
-----------|-----------|-----------|--------------------
    841.11 |    841.08 |    841.09 | step
    760.65 |     70.28 |    181.93 | barotropic_driver
    760.49 |     68.89 |    179.52 | pcg_chrongear iteration
    697.03 |      6.82 |    570.62 | baroclinic_driver
      4.69 |      0.00 |      3.56 | pcg_chrongear preconditioning
```

Summary report from POP

```
*** profile for 360 tasks ***
 max total | min total | avg total | timer
-----------|-----------|-----------|--------------------
    841.11 |    841.08 |    841.09 | step
    760.65 |     70.28 |    181.93 | barotropic_driver
    760.49 |     68.89 |    179.52 | pcg_chrongear iteration
    697.03 |      6.82 |    570.62 | baroclinic_driver
      4.69 |      0.00 |      3.56 | pcg_chrongear preconditioning
```

Load imbalance

Detailed reports

- One for each timer
- In max-sort order
- First list tasks calling that timer
  - Could use a tree that reduces task ranges
  - But just one datum per task, so gather all to root
  - Root generates list, reducing ranges to "<*first*>–<*last*>"
    - Example: 1,2,3,4,5,6,7,8,9,10,11 → 1-11
- Then list all simple global stats I can think of

Detailed report from POP

```
*** barotropic_driver
tasks: 0-359
events per task: 128 max, 128 min, 128 avg (360 tasks)
     longest event:     6.262 sec (6262 ticks), task 359, event 1
   max 2nd longest:     6.210 sec (6210 ticks), task 359, event 2
    shortest event:     0.465 sec (465 ticks), task 286, event 103
 longest avg event:     5.942 sec (5942 ticks), task 359, 128 event(s)
shortest avg event:     0.549 sec (549 ticks), task 286, 128 event(s)
         avg event:     1.421 sec (1421 ticks), 46080 event(s)
         max total:   760.647 sec (760647 ticks), task 270, 128 event(s)
         min total:    70.280 sec (70280 ticks), task 285, 128 event(s)
         avg total:   181.927 sec (181927 ticks), 360 tasks
   aggregate total: 65493.801 sec (65493795 ticks)
```

Detailed report from POP

```
*** barotropic_driver
tasks: 0-359
events per task: 128 max, 128 min, 128 avg (360 tasks)
     longest event:     6.262 sec (6262 ticks), task 359, event 1
   max 2nd longest:     6.210 sec (6210 ticks), task 359, event 2
    shortest event:     0.465 sec (465 ticks), task 286, event 103
 longest avg event:     5.942 sec (5942 ticks), task 359, 128 event(s)
shortest avg event:     0.549 sec (549 ticks), task 286, 128 event(s)
         avg event:     1.421 sec (1421 ticks), 46080 event(s)
         max total:   760.647 sec (760647 ticks), task 270, 128 event(s)
         min total:    70.280 sec (70280 ticks), task 285, 128 event(s)
         avg total:   181.927 sec (181927 ticks), 360 tasks
   aggregate total: 65493.801 sec (65493795 ticks)
```

Load imbalance

Histograms

- Must reduce from one histogram per task (per timer)
- First attempt: combine histograms with same “shape”
  - Each task sorts histogram bins in order of event counts
  - Histograms with same **bin order** have same “shape”
  - Combine histograms of same shape using min, max, avg
- Number of possible histograms
  - = Number of different orders of bins
  - = (Number of bins)! [factorial]
  - > Number of MPI tasks
- Worst case, no reduction!

Reduced histograms

- Must reduce from one histogram per task (per timer)
- But don't want to lose signal of load imbalance and performance variability
- For each histogram, find the bin with the max number of events
- Combine histograms with same max bin
- Number of possible histograms
  - = Number of bins
  - < MPI tasks

Histogram report

Many histograms combined into four

```
*** baroclinic_driver
…
event histogram for tasks: 0,2-4,13-22,25,27,31-44,47-53,55-62,65-71,74-80,84-89,91-
96,102-107,109-114,121-124,127-132,138-141,145-151,155-159,163-169,172-178,180,182-
184,193-202,205,207,211-224,227-233,235-242,245-251,254-260,264-269,271-276,282-
287,289-294,301-304,307-312,318-321,325-331,335-339,343-349,352-358
6000-6999 ticks: 001 max (task 000), 001 min (task 000), 001 avg
5000-5999 ticks: 127 max (task 000), 127 min (task 000), 127 avg

event histogram for tasks: 23-24,45-46,54,63-64,72-73,81-83,90,97-101,108,115-
120,125-126,133-137,142-144,152-154,160-162,170-171,179,181,185-192,206,208-210
6000-6999 ticks: 001 max (task 023), 001 min (task 023), 001 avg
5000-5999 ticks: 001 max (task 045), 000 min (task 023), 000 avg
4000-4999 ticks: 127 max (task 023), 126 min (task 045), 126 avg

event histogram for tasks: 1,6,11
6000-6999 ticks: 01 max (task 001), 01 min (task 001), 01 avg
7 tick(s): 75 max (task 001), 69 min (task 006), 73 avg
6 tick(s): 58 max (task 006), 52 min (task 001), 54 avg

event histogram for tasks: 5,7-10,12,26,28-30,203-204,225-226,234,243-244,252-253,
261-263,270,277-281,288,295-300,305-306,313-317,322-324,332-334,340-342,350-351,359
6000-6999 ticks: 001 max (task 005), 001 min (task 005), 001 avg
7 tick(s): 057 max (task 005), 000 min (task 204), 006 avg
6 tick(s): 127 max (task 262), 070 min (task 005), 120 avg
5 tick(s): 005 max (task 204), 000 min (task 005), 000 avg
```

Histogram report

Load imbalance

```
*** baroclinic_driver
…
event histogram for tasks: 0,2-4,13-22,25,27,31-44,47-53,55-62,65-71,74-80,84-89,91-
96,102-107,109-114,121-124,127-132,138-141,145-151,155-159,163-169,172-178,180,182-
184,193-202,205,207,211-224,227-233,235-242,245-251,254-260,264-269,271-276,282-
287,289-294,301-304,307-312,318-321,325-331,335-339,343-349,352-358
6000-6999 ticks: 001 max (task 000), 001 min (task 000), 001 avg
5000-5999 ticks: 127 max (task 000), 127 min (task 000), 127 avg

event histogram for tasks: 23-24,45-46,54,63-64,72-73,81-83,90,97-101,108,115-
120,125-126,133-137,142-144,152-154,160-162,170-171,179,181,185-192,206,208-210
6000-6999 ticks: 001 max (task 023), 001 min (task 023), 001 avg
5000-5999 ticks: 001 max (task 045), 000 min (task 023), 000 avg
4000-4999 ticks: 127 max (task 023), 126 min (task 045), 126 avg

event histogram for tasks: 1,6,11
6000-6999 ticks: 01 max (task 001), 01 min (task 001), 01 avg
7 tick(s): 75 max (task 001), 69 min (task 006), 73 avg
6 tick(s): 58 max (task 006), 52 min (task 001), 54 avg

event histogram for tasks: 5,7-10,12,26,28-30,203-204,225-226,234,243-244,252-253,
261-263,270,277-281,288,295-300,305-306,313-317,322-324,332-334,340-342,350-351,359
6000-6999 ticks: 001 max (task 005), 001 min (task 005), 001 avg
7 tick(s): 057 max (task 005), 000 min (task 204), 006 avg
6 tick(s): 127 max (task 262), 070 min (task 005), 120 avg
5 tick(s): 005 max (task 204), 000 min (task 005), 000 avg
```

Load imbalance

- Cartesian distribution of blocks of grid cells
- Some blocks are all land, no ocean
- Tasks with land blocks have no work
- Try packing ocean blocks to low-number tasks
  - See slides from 2007 Cray Technical Workshop
- Use just enough tasks to handle ocean blocks

Packed

```
*** baroclinic_driver
…
event histogram for tasks: 0-122,180-302
6000-6999 ticks: 001 max (task 000), 001 min (task 000), 001 avg
5000-5999 ticks: 127 max (task 000), 127 min (task 000), 127 avg

event histogram for tasks: 123-179
6000-6999 ticks: 001 max (task 123), 001 min (task 123), 001 avg
4000-4999 ticks: 127 max (task 123), 127 min (task 123), 127 avg

event histogram for tasks: 303-359
6000-6999 ticks: 001 max (task 303), 001 min (task 303), 001 avg
7 tick(s): 009 max (task 340), 000 min (task 303), 003 avg
6 tick(s): 127 max (task 309), 118 min (task 340), 123 avg
5 tick(s): 001 max (task 303), 000 min (task 304), 000 avg
```

Extra tasks

Packed, using 304 tasks

```
*** baroclinic_driver
…
event histogram for tasks: 0-150,152-302
6000-6999 ticks: 001 max (task 000), 001 min (task 000), 001 avg
5000-5999 ticks: 127 max (task 000), 127 min (task 000), 127 avg

event histogram for task: 151
6000-6999 ticks: 001 max (task 151), 001 min (task 151), 001 avg
4000-4999 ticks: 127 max (task 151), 127 min (task 151), 127 avg

event histogram for task: 303
6000-6999 ticks: 001 max (task 303), 001 min (task 303), 001 avg
7 tick(s): 006 max (task 303), 006 min (task 303), 006 avg
6 tick(s): 121 max (task 303), 121 min (task 303), 121 avg
```

What about scalability? 4500 tasks?

```
*** baroclinic_driver
…
event histogram for tasks: 12-16,56-66,85-92,131-143,150-151,160-168,206-232,235
-245,283-307,310-323,327-328,356-383,385-406,412-414,429-458,460-490,495,497-499
,502-1282,1284-1357,1359-1432,1435-1508,1510-1551,1554-1583,1586-1601,1603-1622,
1624-1625,1629-1658,1662-1676,1679-1696,1704-1733,1737-1751,1754-1771,1779-1808,
1814-1825,1830-1848,1853-1883,1889-1900,1905-1924,1926-1957,1964-1975,1981-2031,
2040-2050,2055-2106,2113-2125,2130-2181,2187-2199,2206-2256,2261-2275,2282-2332,
2335-2345,2348,2358-2420,2431-2494,2505-2518,2520-2569,2580-2581,2584-2588,2590-
2592,2594-2644,2655-2656,2658-2663,2665-2667,2670-2700,2702-2720,2729-2730,2732-
2736,2747-2774,2777-2796,2801-2805,2807-2808,2823-2849,2855-2872,2874-2880,2897-
2924,2931-2955,2958-2959,2972-2998,3007-3029,3032-3034,3047-3072,3082-3101,3103-
3109,3125-3147,3158-3172,3175,3178-3183,3201-3221,3233-3248,3257-3259,3277-3296,
3309,3311-3325,3332-3334,3351-3372,3381-3382,3386-3401,3426-3446,3456-3457,3460-
3476,3502-3520,3530-3548,3550-3552,3580-3594,3605-3613,3615-3623,3625-3627,3659-
3662,3664-3669,3679-3688,3691-3698,3700-3701,3732,3734-3737,3741-3742,3754-3764,
3767-3778,3805-3812,3829-3839,3843-3849,3851-3853,3877-3886,3905-3914,3918-3928,
3951-3963,3983-3989,3993-4004,4025-4040,4058-4064,4068-4078,4099-4115,4132-4139,
4142-4154,4176-4191,4207-4229,4249-4266,4282-4304,4319-4320,4322-4342,4357-4379,
4393-4418,4431-4432,4434-4455,4468-4493
500-599 ticks: 001 max (task 0012), 001 min (task 0012), 001 avg
300-399 ticks: 127 max (task 0012), 127 min (task 0012), 127 avg

event histogram for tasks: 2182,2184-2186,2200-2205
500-599 ticks: 001 max (task 2182), 001 min (task 2182), 001 avg
10-19 ticks: 126 max (task 2182), 126 min (task 2182), 126 avg
9 tick(s): 001 max (task 2182), 001 min (task 2182), 001 avg

event histogram for tasks: 2,4
500-599 ticks: 001 max (task 0002), 001 min (task 0002), 001 avg
8 tick(s): 110 max (task 0002), 076 min (task 0004), 093 avg
7 tick(s): 051 max (task 0004), 017 min (task 0002), 034 avg
```

What about scalability? 4500 tasks?

```
event histogram for tasks: 0-1,3,5-6,8-11,17-20,23,33-55,67-81,93-94,108-130,144
-149,152-157,169,182-193,195-205,233-234,256-263,265-267,271-282,308-309,330-331
,334-337,341,345-355,384,409-411,421-428,459,500-501
500-599 ticks: 001 max (task 0000), 001 min (task 0000), 001 avg
8 tick(s): 061 max (task 0003), 000 min (task 0008), 001 avg
7 tick(s): 106 max (task 0033), 065 min (task 0009), 081 avg
6 tick(s): 062 max (task 0009), 000 min (task 0000), 044 avg

event histogram for tasks: 7,21-22,24-32,82-84,95-107,158-159,170-181,194,246-25
5,264,268-270,324-326,329,332-333,338-340,342-344,407-408,415-420,491-494,496,12
83,1358,1433-1434,1509,1552-1553,1584-1585,1602,1623,1626-1628,1659-1661,1677-16
78,1697-1703,1734-1736,1752-1753,1772-1778,1809-1813,1826-1829,1849-1852,1884-18
88,1901-1904,1925,1958-1963,1976-1980,2032-2039,2051-2054,2107-2112,2126-2129,21
83,2257-2260,2276-2281,2333-2334,2346-2347,2349-2357,2421-2430,2495-2504,2519,25
70-2579,2582-2583,2589,2593,2645-2654,2657,2664,2668-2669,2701,2721-2728,2731,27
37-2746,2775-2776,2797-2800,2806,2809-2822,2850-2854,2873,2881-2896,2925-2930,29
56-2957,2960-2971,2999-3006,3030-3031,3035-3046,3073-3081,3102,3110-3124,3148-31
57,3173-3174,3176-3177,3184-3200,3222-3232,3249-3256,3260-3276,3297-3308,3310,33
26-3331,3335-3350,3373-3380,3383-3385,3402-3425,3447-3455,3458-3459,3477-3501,35
21-3529,3549,3553-3579,3595-3604,3614,3624,3628-3658,3663,3670-3678,3689-3690,36
99,3702-3731,3733,3738-3740,3743-3753,3765-3766,3779-3804,3813-3828,3840-3842,38
50,3854-3876,3887-3904,3915-3917,3929-3950,3964-3982,3990-3992,4005-4024,4041-40
57,4065-4067,4079-4098,4116-4131,4140-4141,4155-4175,4192-4206,4230-4248,4267-42
81,4305-4318,4321,4343-4356,4380-4392,4419-4430,4433,4456-4467,4494-4499
500-599 ticks: 001 max (task 0007), 001 min (task 0007), 001 avg
7 tick(s): 063 max (task 0338), 000 min (task 0026), 003 avg
6 tick(s): 126 max (task 0251), 064 min (task 0338), 117 avg
5 tick(s): 014 max (task 3173), 000 min (task 0007), 006 avg
```

360→4500 tasks

4→5 histograms

Long task lists

4500 tasks, packed

```
*** baroclinic_driver
…
event histogram for tasks: 0-3267
500-599 ticks: 001 max (task 0000), 001 min (task 0000), 001 avg
300-399 ticks: 127 max (task 0001), 126 min (task 0000), 126 avg
200-299 ticks: 001 max (task 0000), 000 min (task 0001), 000 avg

event histogram for tasks: 3268-4499
500-599 ticks: 001 max (task 3268), 001 min (task 3268), 001 avg
6 tick(s): 126 max (task 3503), 110 min (task 3436), 118 avg
5 tick(s): 017 max (task 3436), 001 min (task 3503), 008 avg
```

3268 tasks, packed

```
*** baroclinic_driver
…
event histogram for tasks: 122-128,154,169-197,210,238-254
200-299 ticks: 104 max (task 0127), 065 min (task 0197), 087 avg
100-199 ticks: 063 max (task 0197), 024 min (task 0127), 040 avg

event histogram for tasks: 0-121,129-153,155-168,198-209,211-
237,255-3267
200-299 ticks: 064 max (task 0166), 007 min (task 3054), 020 avg
100-199 ticks: 121 max (task 3054), 064 min (task 0166), 107 avg
```

Conclusions

- Multi-resolution timers
  - Local to each task
  - Identified by a string
  - Multi-resolution histograms of events
- Reduction in reporting
  - Timers with same string name are combined across tasks
  - Max, 2nd max, min, avg, with extreme tasks and events
  - Histograms with same max bin are combined
- Scalability
  - Max-bin reduction keeps histogram count low
  - Signals of load imbalance and performance variability remain
  - Task list can get long
- Some potential improvements
  - Measure and report overhead
  - OpenMP support
  - Nicer reporting

It's About Time
Multi-Resolution Timers for Scalable Performance Debugging

James B. White III (Trey)
trey@ornl.gov

NATIONAL CENTER
FOR COMPUTATIONAL SCIENCES

OAK RIDGE NATIONAL LABORATORY