Thread Scalability of Profiling Data Collection

Marty Itzkowitz **marty.itzkowitz@oracle.com**

Senior Principal Engineer, Oracle Solaris Studio Performance Tools

2015 Scalable Tools Workshop, Lake Tahoe, CA

Monday, August 3, 2015

Agenda

- The Problem
- Methodology
  - Overhead
  - Scalability Target Code
  - Testing Infrastructure and the Scalability Test Script
- Measurements
- The Measured Data
- Conclusions

The Problem

- Oracle is producing larger and larger servers
  - SPARC-T5 has 256 virtual CPUs
  - SPARC-M6 has 1536 virtual CPUs
  - SPARC-M7 has 4096 virtual CPUs
- Many machines are partitioned, but some are not
- Users want to exploit all the threads
- Performance Analyzer is needed to tune such code
  - Scalability must be addressed
- Project undertaken to examine data collection scalability
  - All data is <u>VERY PRELIMINARY</u> -- much more work is needed

Methodology

- Build a test program that uses N threads, and M iterations
  - Set M to give ~15 seconds of execution per thread.
- Measure the entire time it takes to run as a function of N
  - Baseline runs -- no data collection
  - Data collection runs -- with data collection
    - Clock-profiling at regular and low resolution, and “paused”, with recording turned off
    - Clock and HWC profiling on two counters, same conditions
  - Do repeated runs to measure variance
- Compute overhead: (data-collection run time) - (baseline run time)

Contributions to Overhead

- Process start
  - Open shared library for data collection, initialize data collection
  - Overhead in dealing with thread creation, termination
- Profiling events
  - Cost of processing each event
- Process termination
  - Data collection termination, archiving
- This study concentrates on profiling events

Profiling Event Overhead

- For each profiling event
  - OS receives interrupts from HW, re-enables events (if necessary)
  - OS delivers a signal to the data collection library
  - Data collection library gets signal
    - Unwind the stack, prepare event record
    - Write stack (if necessary); write event
    - Restart data collection (if necessary)
    - Returns from the signal
- Event overhead is directly connected to profiling rate
  - Normal rate: ~ 100 events/second/thread
  - Low-resolution: ~ 10 events/second/thread
  - Paused data collection: clock events are generated, but no unwind or I/O
    - HWC events are not generated

Scalability Target Code

- Simple C program, `thr_scale.c`
  - `-t N` -- gives thread count (to a maximum of 4096)
  - `-c M` -- gives iteration count
- Main program
  - Launches N threads, posts N semaphores, waits for N threads to finish
  - Prints per-thread real-time and CPU-time
- Thread code
  - Each starts with a unique function, calls a common `thread_work` function
    - Table of starting functions and their code generated by `genfunc` code
  - `thread_work` waits for a semaphore, loops for M iterations, and then exits

Testing Infrastructure

- Infrastructure for Performance Analyzer nightly test suite
  - More than a decade of development
  - Powerful web site for monitoring and navigating the runs
- A single test is a target code and a set of “tags” grouped in {}'s
  - Tags for compile options, FZ, FZ_O, FZ_DBG, …; B32/B64
  - Tags specifying compiler: CC_TRAIN, CC_GNU, CC_INTEL, CC_OLD, …
  - Tags for how to collect data: PM_NO_DATA, PM_COLLECT, PM_ER_KERNEL, …
  - Tags specifying data: DA_CLK, DA_HWC, DA_LCLK_HWD, DA_PROFILE, …
  - Tags for profiling resolution: DA_HIRES, DA_LORES
  - Tags for data collection control: DA_SIG, DA_SIGRES
  - Tags for Java, MPI, OpenMP, and many, many more

Scalability Test Script

- Compile the scalability target code, as per compile tags
- One tag specifies THREAD_LIST, a list of thread counts to use
  - TSC_THR_72 -- 72 CPU machine, 4 - 80 in multiples of 4
  - TSC_THR_72X -- 72 CPU machine, finer resolution around 72
  - TSC_THR_256 -- 256 CPU machine, 16-272 in multiples of 16
  - TSC_THR_256X -- 256 CPU machine, finer resolution around 256
- One tag gives CALC_COUNT, an iteration count (== ~15 seconds)
- Specify data collection method and type of data (or baseline == no data)
- Run the test code for each count in THREAD_LIST
  - Testing infrastructure reports run time
    - Supports data verification
  - Each test writes file with data collection method, thread counts, times for each

Measurements

- Test run is described in `com.list.master` file
  - `com` == "compact"; sets of tags in braces are expanded
  - Testing infrastructure made it easy to implement the scalability test suite
    - Just add sets of tags in braces, *e.g.*:
      - {,DUP,DUP2} ==> three repeats of the test
- Each machine ran 6 suites, each suite having 4 sequential iterations
  - Three iterations with the full thread list appropriate to the machine size
  - Three iterations with the fine-grained thread list around CPU count
- Each iteration ran 50 tests
  - Five baseline, three each of each of fifteen data-collection types
  - Overall iteration time was remarkably consistent: within 15 seconds over 18,600 seconds
- Each single test does ~18 runs of the test code with 18 different thread counts

Data Collection Environment

- Relatively isolated machines
  - No other users
  - No NFS server activity
- All files on local disks
  - Test-code sources, testing infrastructure, compilers, *etc.*
  - Experiments recorded to local disk
- Test machines:
  - Oracle SPARC T5, 256 virtual CPUs, 3.6 GHz, running Solaris 11.2
  - Oracle X5 (Haswell E/EP chip), 72 virtual CPUs, 3.6 GHz, running Solaris 11.3
  - Oracle X5 (Haswell E/EP chip), 72 virtual CPUs, 3.6 GHz, running Linux OEL_UEK_6.5
    - x86 machines had frequency scaling and turbo mode disabled

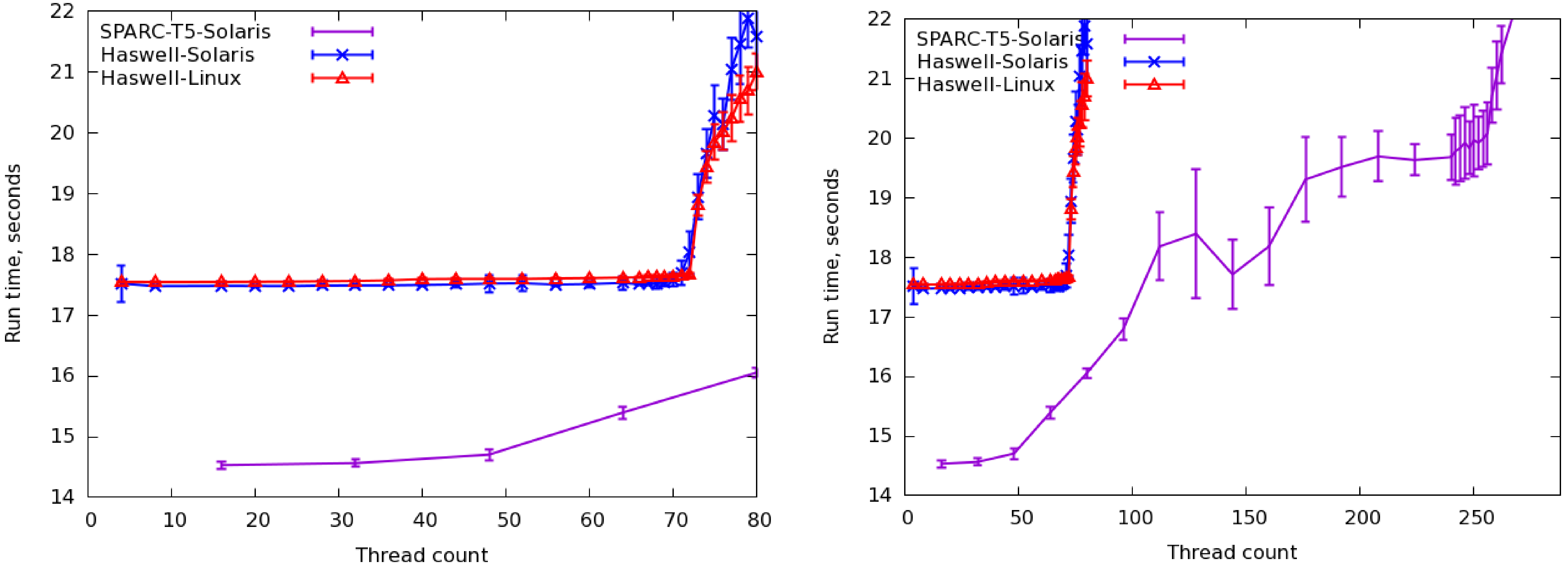
      (Thanks to Nik Molchanov for setting up the machines)

Data Reduction

- Process 1200 summary files for each architecture
  - 24,000 runs (SPARC-Solaris); 25,200 runs (x86-Solaris); or 25,200 runs (x86-Linux)
  - Computed mean and standard deviation for baseline, minimum and maximum
  - Computed mean and standard deviation for each data type, minimum and maximum
    - Computed ratio to baseline, difference of mean from baseline mean
- Write sixteen data files for each architecture
  - One for Baseline, and one for each data-collection type
  - Each line in file has thread count, mean and standard deviation for that data type
- Use gnuplot to plot the data

  (Thanks to Eugene Loh for help with gnuplot)
- Baseline points represent the mean and standard deviation over 60 runs
- Data-collection points represent the mean and standard deviation over 36 runs

Data: Baseline

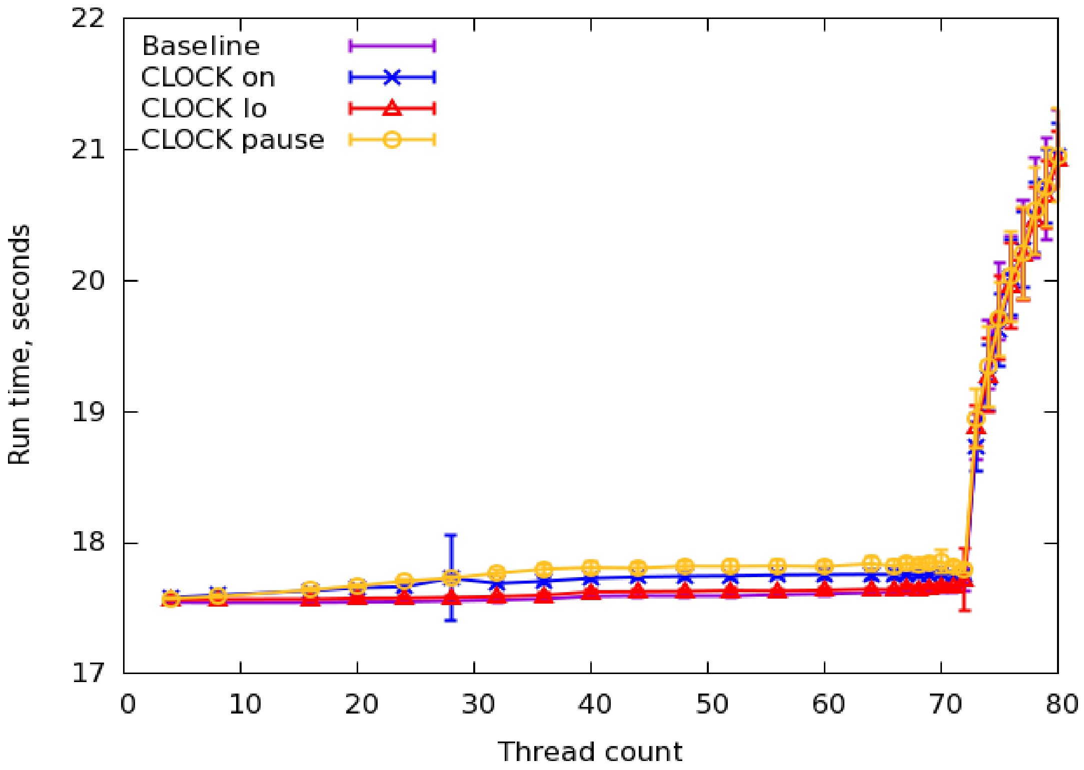


Note: Iteration count on SPARC-T5 was not the same as on Haswell

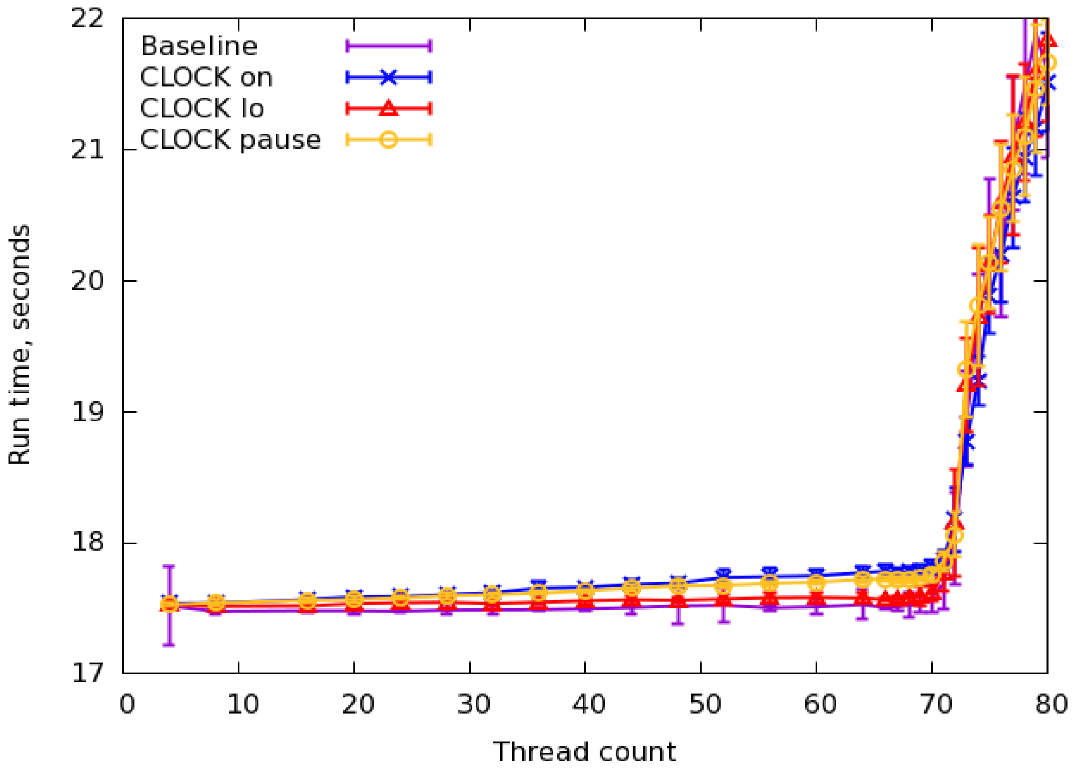
Why does SPARC-T5 start to slow down with relatively few threads?

Data: Clock-Profiling, X86

Haswell-Linux



Haswell-Solaris

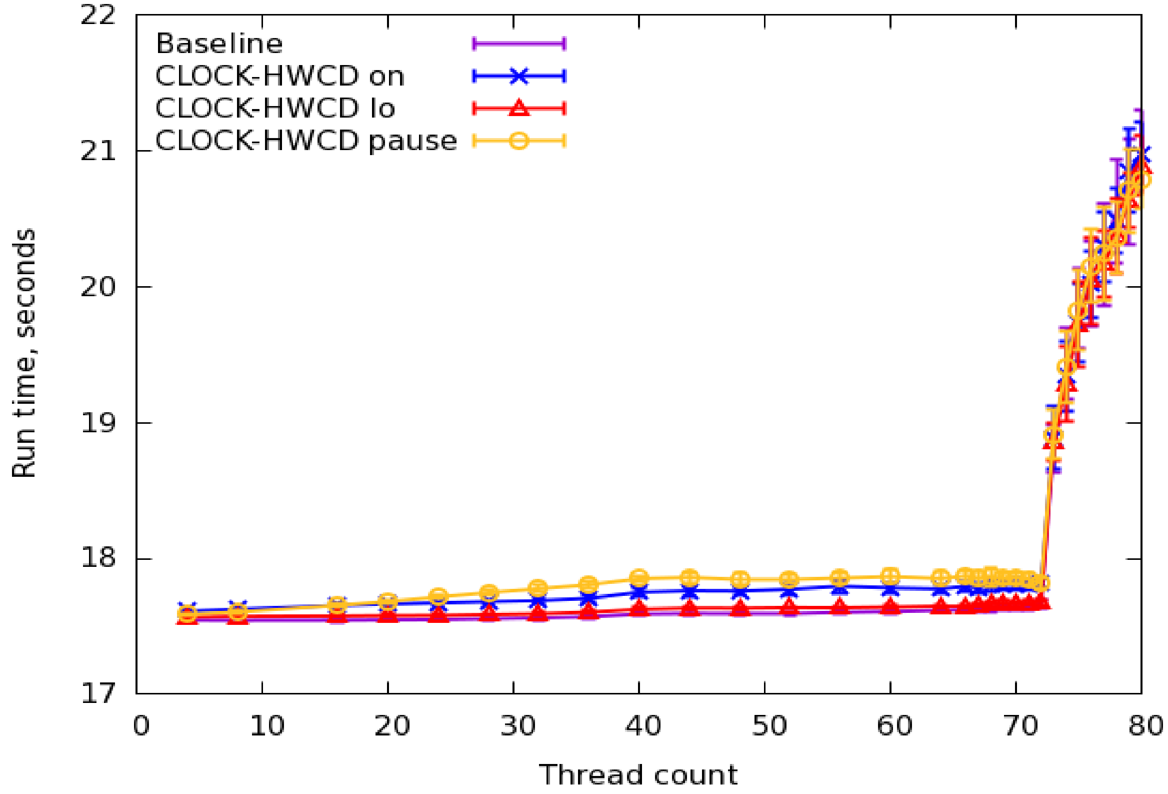


Why is there high variance on Haswell-Linux, CLOCK on, 28 threads?

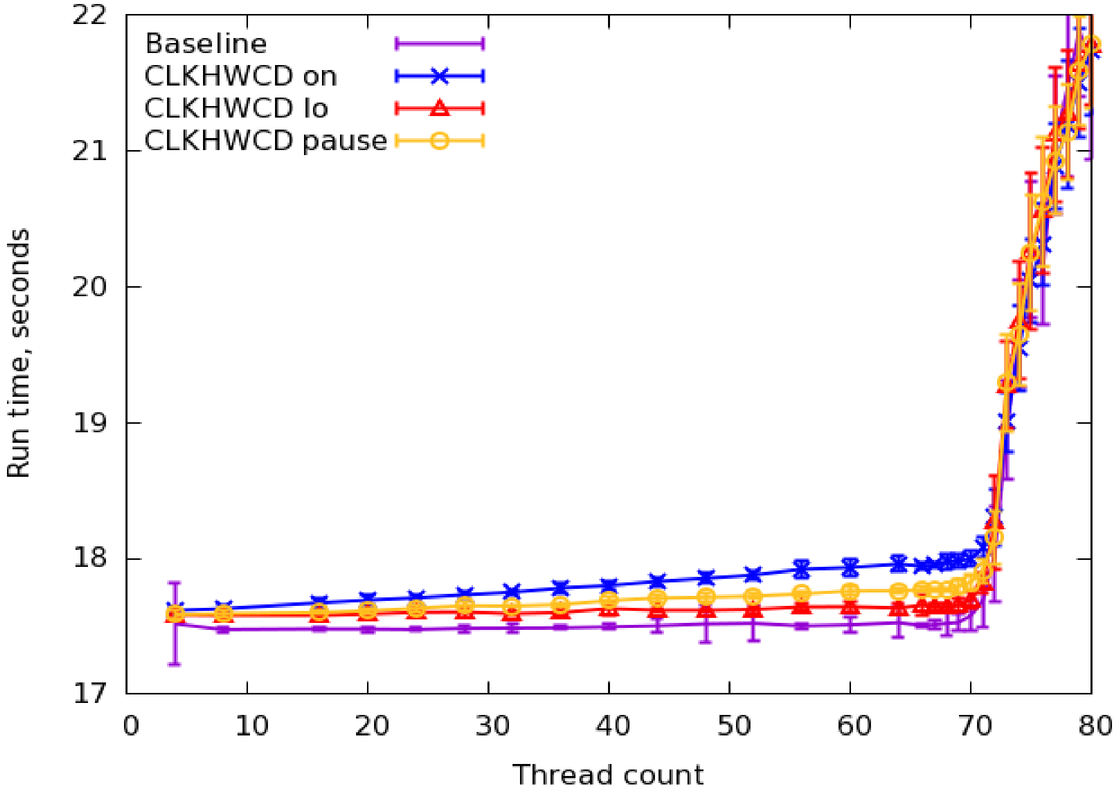
How can Haswell-Linux CLOCK pause take longer than CLOCK on?

Data: Clock and HWC Profiling with Two Counters, X86

Haswell-Linux



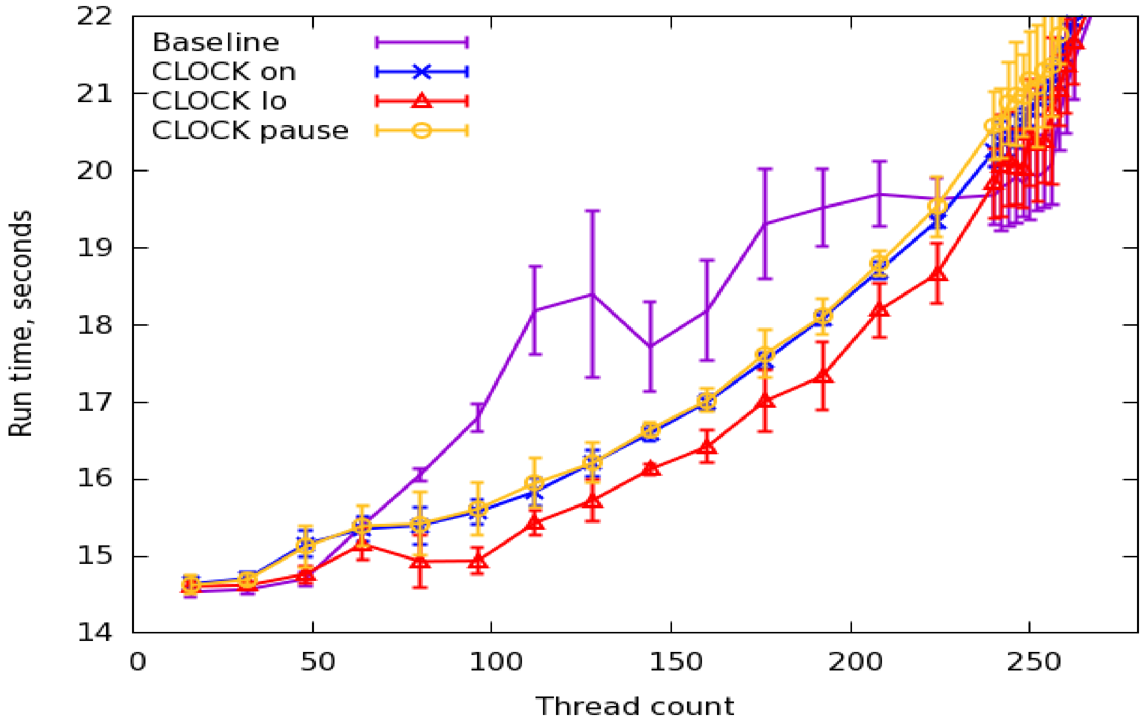
Haswell-Solaris

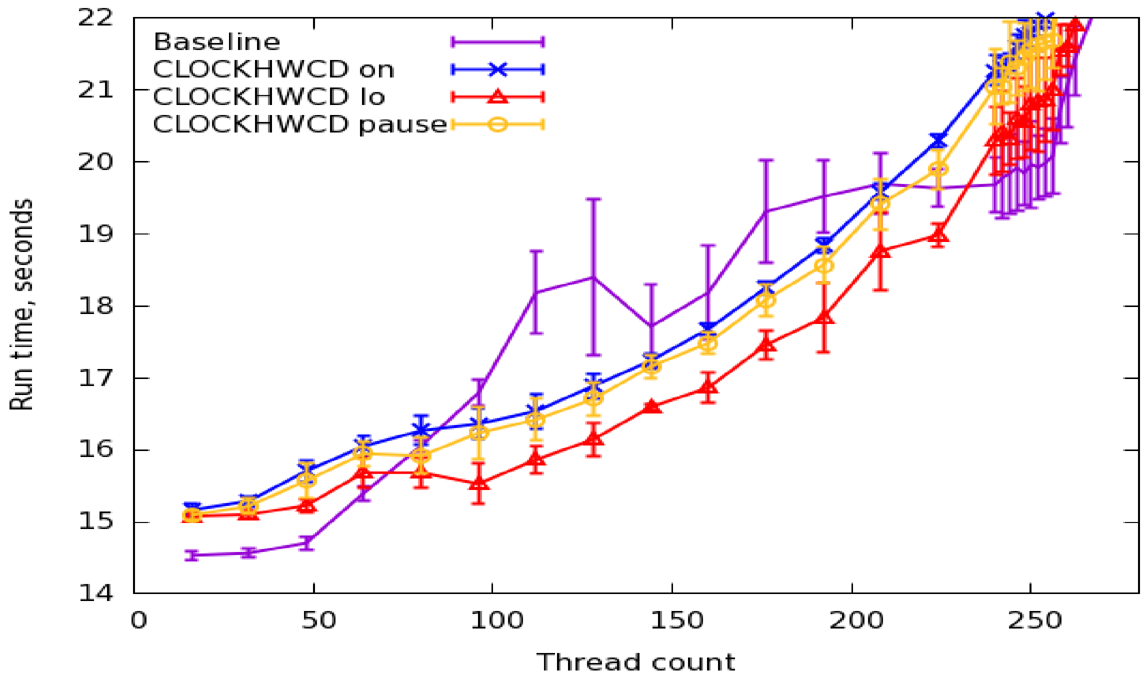


How can Haswell-Linux CLOCK-HWCD pause take longer than CLOCK-HWCD on?

Data: Clock Profiling and Clock and HWC Profiling with Two Counters, SPARC

SPARC-T5-Solaris





Why are all data collection times faster than the Baseline?
Why is the Baseline variance so high?

Discussion

- Good news: little or no data-collection overhead
- Picasso said “Computers are useless; they can only give you answers”
  - He was mistaken: this study gave us lots of questions!
- Future work
  - Answer questions:
    - Profile Solaris kernels while these tests are running
      - Understand kernel behavior supporting data collection
    - Understand variability over identical runs: Memory and thread placement? Something else?
  - Separate out stack unwind from other costs
    - Expand leaf routine to give more varied callstacks ==> more I/O, less caching of stacks
    - Note that X86 unwind is quite different from SPARC unwind
  - Extend measurements to largest machines

Oracle Solaris Studio 12.4

Download and use is free! (Support is not, however)

Access to Latest Release:

http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html