Supercomputing for Everyone Series: Performance Tuning Summer School

L3: Use simple tools for simple questions

Dr. Holger Brunst
Technische Universität Dresden, Germany

August 17–21, 2015
Organization of this class

- course runs August 17-21, from 11 a.m. until 4.30 p.m. (EDT)
- 20 sites, each site should have an instructor/TA/helper
- please put your microphones on mute, if you are not asking questions
- set up in lectures (60 min) and exercises (75 min)
- Q&A time at end of each lecture
- chat for discussions
- class material and links: [http://go.iu.edu/CtB](http://go.iu.edu/CtB)
Lunch breaks are important

<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
<th>Session Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>11.00-12.30</td>
<td>L1_Performance is ambiguous</td>
</tr>
<tr>
<td></td>
<td>12.30-13.15</td>
<td>X1_First steps on Blue Waters</td>
</tr>
<tr>
<td></td>
<td>13.15-14.15</td>
<td><strong>Lunch break</strong></td>
</tr>
<tr>
<td></td>
<td>14.15-15.15</td>
<td>L2_Evaluation first!</td>
</tr>
<tr>
<td></td>
<td>15.15-16.30</td>
<td>X2_Pen, paper, and performance</td>
</tr>
<tr>
<td>Tuesday</td>
<td>11.00-12.00</td>
<td>L3_Use simple tools for simple questions</td>
</tr>
<tr>
<td></td>
<td>12.00-13.15</td>
<td>X3_The command line is your friend</td>
</tr>
<tr>
<td></td>
<td>13.15-14.15</td>
<td><strong>Lunch break</strong></td>
</tr>
<tr>
<td></td>
<td>14.15-15.15</td>
<td>L4_Tuning needs persistence</td>
</tr>
<tr>
<td></td>
<td>15.15-16.30</td>
<td>X4_Benchmarks provide the baseline</td>
</tr>
<tr>
<td>Wednesday</td>
<td>11.00-12.00</td>
<td>L5_Core tuning pays off n-times</td>
</tr>
<tr>
<td></td>
<td>12.00-13.15</td>
<td>X5_How to access data efficiently</td>
</tr>
<tr>
<td></td>
<td>13.15-14.15</td>
<td><strong>Lunch break</strong></td>
</tr>
<tr>
<td></td>
<td>14.15-15.15</td>
<td>L6_Visual tools can be fun, sometimes</td>
</tr>
<tr>
<td></td>
<td>15.15-16.30</td>
<td>X6_Zoom and scroll</td>
</tr>
<tr>
<td>Thursday</td>
<td>11.00-12.00</td>
<td>L7_Sharing may double the sorrow (OpenMP)</td>
</tr>
<tr>
<td></td>
<td>12.00-13.15</td>
<td>X7_OpenMP enables quick 'n easy gains</td>
</tr>
<tr>
<td></td>
<td>13.15-14.15</td>
<td><strong>Lunch break</strong></td>
</tr>
<tr>
<td></td>
<td>14.15-15.15</td>
<td>L8_Hand made parallelization hurts (MPI)</td>
</tr>
<tr>
<td></td>
<td>15.15-16.30</td>
<td>X8_From bad MPI to good MPI</td>
</tr>
<tr>
<td>Friday</td>
<td>11.00-12.00</td>
<td>L9_Climb the mount Olympus with GPUs</td>
</tr>
<tr>
<td></td>
<td>12.00-13.15</td>
<td>X9_Hybrid tuning in practice</td>
</tr>
<tr>
<td></td>
<td>13.15-14.15</td>
<td><strong>Lunch break</strong></td>
</tr>
<tr>
<td></td>
<td>14.15-15.15</td>
<td>L10_The grand final</td>
</tr>
<tr>
<td></td>
<td>15.15-16.30</td>
<td>X10_Useful bits and pieces</td>
</tr>
</tbody>
</table>

* All times are EDT
Collecting data is a barrier
Performance recording needs time and memory
Measuring short intervals with sampling
Performance tools have many things in common

OUTLINE
First, we need a monitor

- observes and tracks activities

- system programmer:
  - find frequently used segments

- system manager:
  - measure resource utilization

- system analyst:
  - characterize workload for capacity planning
Measurements are triggered by events

• discrete process

• event triggers performance reading

• events
  – time
  – memory reference
  – processor counter
  – application phase
  – basic block entered
  – disk access
  – network message
  – event counter (overflow)
Events can be internal or external

- internal, e.g.
  - application phase started
  - a certain basic block was entered

- external, e.g.
  - timer/processor interrupt
  - network message arrived
Direct and indirect measurement

- **direct trigger**
  - a.k.a. *instrumentation*
  - event is what we want to measure
  - e.g. arrival of a message

- **indirect trigger**
  - a.k.a. *sampling*
  - record a non-related system state
  - e.g. utilization when timer or hardware counter interrupt occurs
Measuring instrumented events is accurate

Exactly 7 events are observed
Measuring instrumented events is accurate

- measure performance when event occurs
- modify system to observe event
- overhead
  - infrequent events → small overhead
  - frequent events → large overhead
- can significantly alter behavior
- overhead assessment not easy
- good for experiments with low-frequency events
Sampling events statistically is handy

Observes 3 events out of 5 samples
Sampling events statistically is handy

- measures performance at fixed time intervals
- overhead
  - is independent of the number of times a specific event occurs
  - is a function of the sampling frequency
- not every event will be recognized
- events that occur infrequently may be completely missed
- produces statistical view on the overall behavior
- each experiment run is likely to produce a different result
And it is hard to say what’s better

**instrumentation**
- exact timing
- needs modification
- (too) many details
- limited overhead control

**sampling**
- statistical approach
- works with binary
- can miss details
- good overhead control
Heisenberg also applies to software observations

- measurement influences experiment
  - takes time
  - alters instruction sequence
  - OS calls

```c
  t_start = read_timer()
  // task being timed
  {
    ...
  }
  t_end = read_timer()
  elapsed_time =
    (t_end-t_start) * t_cycle
```
Efficient software monitors suffer from Heisenberg

• not a problem if
  – measured interval is much larger than overhead
  – measurements are rare
• overhead can be subtracted
• altered instruction sequence cannot be fixed
• solution:
  – pure hardware monitoring (too expensive)
  – simulation (too slow and complex)
Collecting data is a barrier
Performance recording needs time and memory
Measuring short intervals with sampling
Performance tools have many things in common
Profiling creates summary information

- profiling is typically based on sampling
- general statistical measurement technique
- data gathered from subset of total population

- assumption: since the samples were chosen completely at random, the characteristics of the overall population will approximately follow the same proportion as do the characteristics of the subset actually measured

- software profile: samples taken at fixed times
- interrupt service routine records information
- Post-process to obtain overall profile
Profiling interrupts programs at fixed intervals

- at each interrupt
  - examine PC on return address stack
  - use address map to translate this PC to subroutine I
  - increment array element H[i]
Profiling tells you what happened on average

1. interrupt program
2. get program counter and caller/callpath from return address stack
3. find subroutine $i$ in address map
4. update number of occurrences in histogram

Stack

PC: 742

Address Map

0-711: Sub. 1
712-1024: Sub. 2
1025-2099: Sub. 3

Histogram

Tracing also tells you *when* something happened

- tracing is often based on instrumentation
- records individual event + time
- optionally stores system state
- requires significantly more storage
- needs time to save the states
- can alter program behavior significantly
Tracing creates a timeline of program actions
Collecting data is a barrier
Performance recording needs time and memory
Measuring short intervals with sampling
Performance tools have many things in common

OUTLINE
Measure very short intervals with profiling

- question: how to measure events shorter than clock resolution?
  - i.e. events with $T_e < T_c$

- obviously, we cannot measure them directly!
- even $T_e > nT_c$ for small $n$ is hard to measure.
  - too much overhead!

- however, we can make many measurements of short event
  - obtain a statistical estimate of the duration
Not all intervals will be detected

\[ T_e \quad T_e \]

\[ T_c \quad T_c \quad T_c \quad T_c \]

\[ +1 \quad +1 \quad +1 \]

Time
Short intervals and the Bernoulli experiment

- Bernoulli experiment
  - Result = +1 with probability $p$
  - Result = +0 with probability $(1-p)$

- repeat $n$ times
  - approximates a binomial distribution
  - measurement cannot be guaranteed to be independent
    - usually sufficient in practice
Average duration of an event

- \( m = \) number of times event was observed
- \( n = \) total number of measurements
- Average duration of event is:
  - \( T_e = \frac{m}{n} \times T_c \)
Problems when measuring short intervals statistically

- Problem 1: events need to take place within the timers resolution from time to time
- Problem 2: only average duration values
- Problem 3: bursts
Collecting data is a barrier
Performance recording needs time and memory
Measuring short intervals with sampling
Performance tools have many things in common

OUTLINE
Apps need to be prepared for data recording

- probes are inserted by
  - a preprocessor / a source-to-source translation tool
  - a compiler
  - linking against a pre-instrumented library
  - a binary-rewriter
  - the run-time system
  - even you sometimes!

- prior to execution: static!
- at runtime: dynamic!
Results can be either inclusive or exclusive

- **inclusive result**
  - aggregation across all sublevel results into a single value

- **exclusive result**
  - results only reflect a given level
  - sublevels are subtracted

- **example**: function `foo` that calls a function `bar`
Measurement techniques for dummies

**how is the data measured?**

- **sampling**
  - you get probability results
- **instrumentation**
  - you get exact* results

**how is the data recorded?**

- **profiling**
  - you get a table
- **tracing**
  - you get a sorted event log
Tracing is insightful but a bit tricky

shows individual
– function & loop invocations
– messages

annotated with
– timestamp, location, event type
– event-specific information (e.g., communicator)

event trace = chronologically ordered sequence of events
Profiling is efficient but simple

shows accumulated
- total
- average/maximum/minimum

of metric
- duration
- invocation count
- hardware counters

for
- functions, call sites, basic blocks, loops

profile = summarization of events over execution interval
Types of profiles

• flat profile
  – distribution of metrics per routine

• call-path profile
  – takes into account calling context
  – distribution of metrics per executed call path
  – call-graph profile: two levels only

• special-purpose profiles
  – address specific aspects like MPI or OpenMP metrics
When to use profiling and tracing

Profiling

• provides summary information
• enables line based statistics
• includes counter statistics
• works best for long runs

• has a fixed overhead

Tracing

• provides detailed execution log
• reveals synchronization issues
• reveals cause of imbalances
• is also good for debugging

• works best with small runs
• needs tweaking
Tradeoff between accuracy and expressiveness

- **accuracy**
  - intrusion overhead
    - measurement needs time and reduces performance
  - perturbation
    - measurement alters program behavior
    - e.g., memory access or communication patterns
- **granularity**
  - how many measurements?
  - how much processing during measurement?
Address the tradeoff with a hierarchical workflow

- do I have a performance problem?
  - time / speedup / scalability measurements
- what is the key bottleneck (computation / communication)?
  - MPI / OpenMP / flat profiling
- Where is the key bottleneck?
  - Call-path profiling, detailed basic block profiling
- Why is it there?
  - Hardware counter analysis, trace selected parts to keep trace size manageable
- Does the code have scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function-by-function
A combo of methods and tools is usually needed

- instrumentation
- measurement
  - sampling vs. instrumentation
  - profiling vs. tracing
- analysis
  - performance statistics
  - event visualization
  - automated analysis
  - data mining

Questions!? [https://connect.iu.edu/ptune15](https://connect.iu.edu/ptune15)