TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Center for Information Services and High Performance Computing (ZIH)

Performance engineering from the application point of view

NLPE@HLRS – ZIH Tools Day

21 June 2024

Center for Information Services & High Performance Computing

Performance factors of parallel applications

"Sequential" performance factors

— Computation
— Cache and memory
— Input / output

"Parallel" performance factors

— Partitioning / decomposition
— Communication (i.e., message passing)
— Multithreading
— Synchronization / locking

What to Measure

So you have some hypothesis about how your code will behave

This requires certain data

- Simple scaling models: execution time, possibly subdivided between serial and parallel parts
- Roofline model: operations/second and bytes/second corresponding to one or more rooflines
- Load balancing: distribution of time spent in computation and communication
- Critical path: detailed measurement of execution time across all nodes and threads

Allows you to ignore certain other data

- Example: load balancing
- Detection typically based on communication wait states
- Don’t need to analyze computation details for that

When possible, measure only what you need to test your hypothesis

- All-in-one-run only when it’s unavoidable

Measurement Practices

Measurements on HPC systems are noisy

- Shared resources: anything short of full-system DAT probably shares something (and maybe even then, if you use site-shared filesystems)
- Nondeterminism: cache effects, which nodes were allocated, small race conditions

Particularly relevant to wall time, but can affect other metrics

As with all scientific measurements, repeat the experiment

- Especially if the initial results look weird!

Measurement issues

Accuracy

— Intrusion overhead

Measurement itself needs time and thus lowers performance

— Perturbation

Measurement alters program behavior

E.g., memory access pattern

— Accuracy of timers & counters

Granularity

— How many measurements?

— How much information / processing during each measurement?

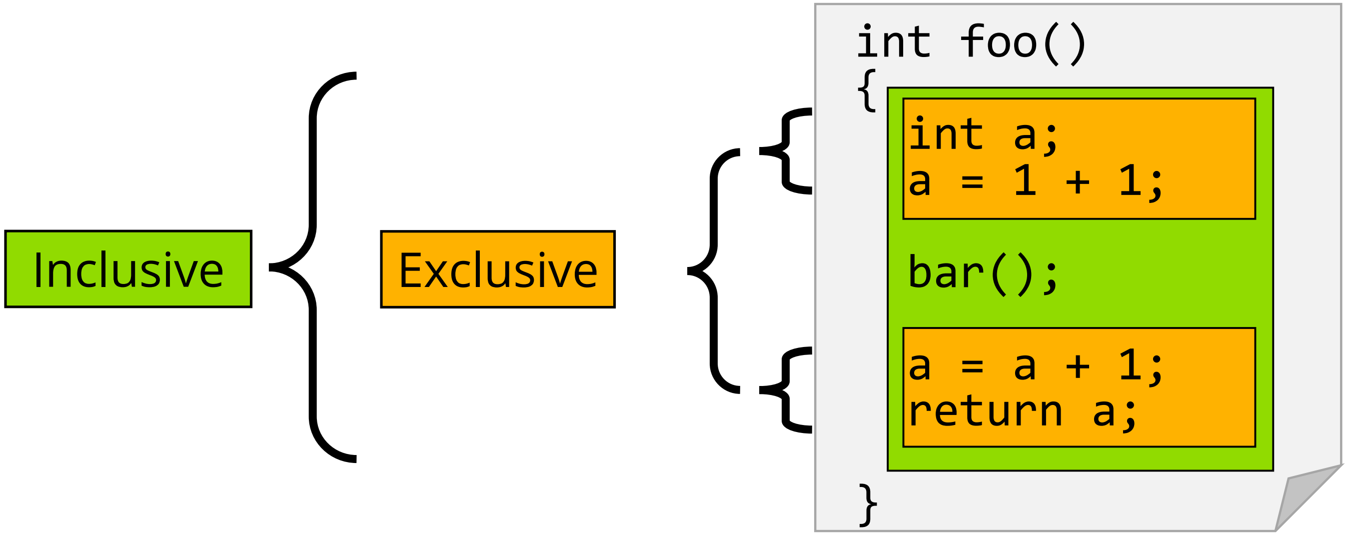
Tradeoff: Accuracy vs. Expressiveness of data

Execution time

- Wall-clock time
  - Includes waiting time: I/O, memory, other system activities
  - In time-sharing environments also the time consumed by other applications
- CPU time
  - Time spent by the CPU to execute the application
  - Does not include time the program was context-switched out
    - Problem: Does not include inherent waiting time (e.g., I/O)
    - Problem: Portability? What is user, what is system time?
- Problem: Execution time is non-deterministic
  - Use mean or minimum of several runs

Inclusive vs. Exclusive values

- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



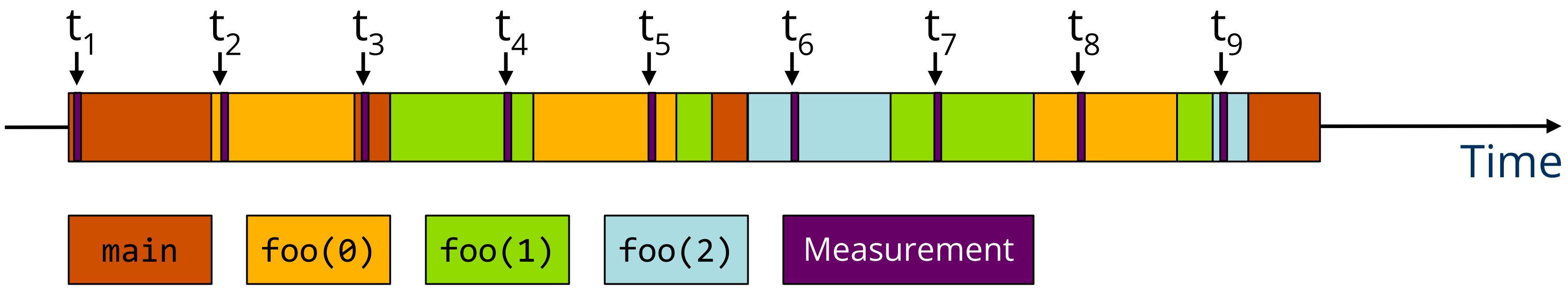
Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Instrumentation

- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing

- How is performance data analyzed?
  - Online
  - Post mortem

Sampling




```
int main() {
  int i;

  for (i=0; i < 3; i++)
    foo(i);

  return 0;
}

void foo(int i) {

  if (i > 0)
    foo(i - 1);

}
```

Running program is periodically interrupted to take measurement

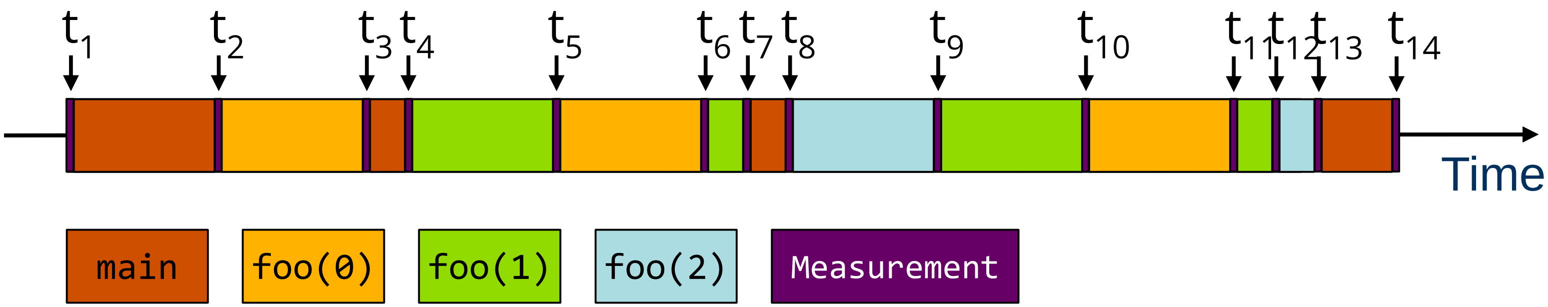
- Timer interrupt, OS signal, or HWC overflow
- Service routine examines return-address stack
- Addresses are mapped to routines using symbol table information

Statistical inference of program behavior

- Not very detailed information on highly volatile metrics
- Requires long-running applications

Works with unmodified executables

Instrumentation




```
int main() {
  int i;
  Enter("main");
  for (i=0; i < 3; i++)
    foo(i);
  Leave("main");
  return 0;
}

void foo(int i) {
  Enter("foo");
  if (i > 0)
    foo(i - 1);
  Leave("foo");
}
```

Measurement code is inserted such that every event of interest is captured directly

— Can be done in various ways

Advantage:

— Much more detailed information

Disadvantage:

— Processing of source-code / executable necessary

— Large relative overheads for small functions

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Instrumentation
- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing
- How is performance data analyzed?
  - Online
  - Post mortem

Profiling / Runtime summarization

Recording of aggregated information

- Total, maximum, minimum, ...

For measurements

- Time
- Counts
  - Function calls
  - Bytes transferred
  - Hardware counters

Over program and system entities

- Functions, call sites, basic blocks, loops, ...
- Processes, threads

Profile = summarization of events over execution interval

Types of profiles

- Flat profile
  - Shows distribution of metrics per routine / instrumented region
  - Calling context is not taken into account
- Call-path profile
  - Shows distribution of metrics per executed call path
  - Sometimes only distinguished by partial calling context (e.g., two levels)
- Special-purpose profiles
  - Focus on specific aspects, e.g., MPI calls or OpenMP constructs
  - Comparing processes/threads

Tracing

Recording detailed information about significant points (events) during execution of the program

- Enter / leave of a region (function, loop, …)
- Send / receive a message, …

Save information in event record

- Timestamp, location, event type
- Plus event-specific information (e.g., communicator, sender / receiver, …)

Abstract execution model on level of defined events

Event trace = Chronologically ordered sequence of event records

Tracing Pros & Cons

Tracing advantages

- Event traces preserve the temporal and spatial relationships among individual events
  (☞ context)
- Allows reconstruction of dynamic application behaviour on any required level of abstraction
- Most general measurement technique
  - Profile data can be reconstructed from event traces

Disadvantages

- Traces can very quickly become extremely large
- Writing events to file at runtime may causes perturbation

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Instrumentation
- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing
- How is performance data analyzed?
  - Online
  - Post mortem

Online analysis

- Performance data is processed during measurement run
  - Process-local profile aggregation
  - Requires formalized knowledge about performance bottlenecks
  - More sophisticated inter-process analysis using
    - “Piggyback” messages
    - Hierarchical network of analysis agents
- Online analysis often involves application steering to interrupt and re-configure the measurement

Post-mortem analysis

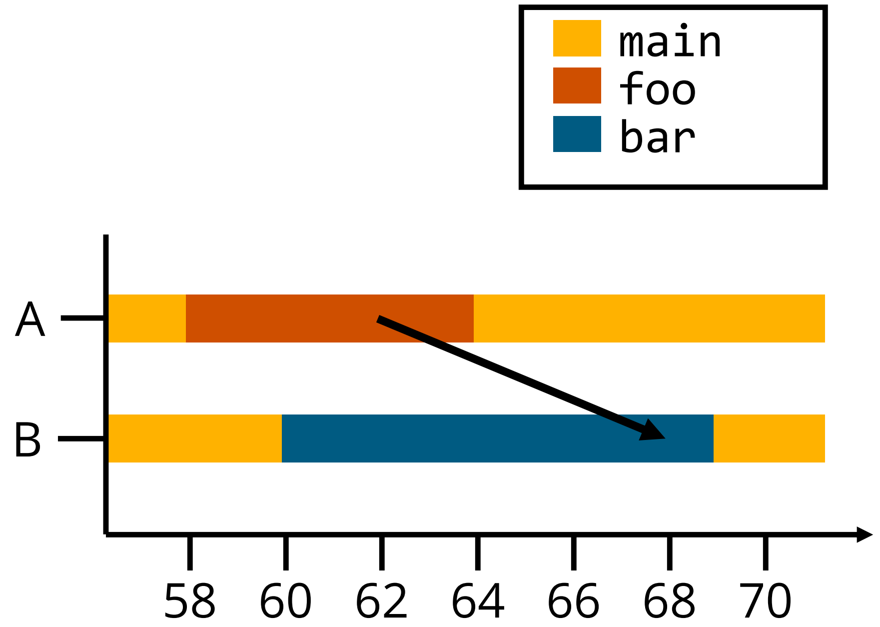
- Performance data is stored at end of measurement run
- Data analysis is performed afterwards
  - Automatic search for bottlenecks
  - Visual trace analysis
  - Calculation of statistics

Example: Time-line visualization

Global trace view

...		
58	A	ENTER foo
60	B	ENTER bar
62	A	SEND to B
64	A	EXIT foo
68	B	RECV from A
69	B	EXIT bar
...		

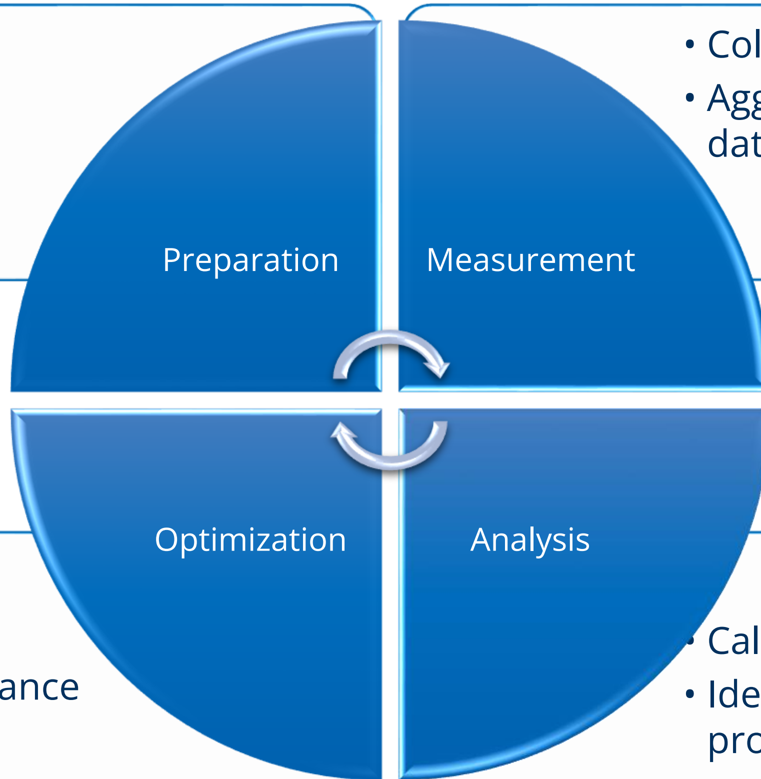
Post-Mortem Analysis



Performance engineering workflow

- Build model of predicted performance
- Select data to measure
- Prepare application with symbols
- Insert extra code (probes/hooks)

- Collection of performance data
- Aggregation of performance data



- Modifications intended to eliminate/reduce performance problem

- Calculation of metrics
- Identification of performance problems
- Presentation of results