Experiment Management Support for Performance Tuning

Karen L. Karavanic
karavan @cs.wisc.edu
http://www.cs.wisc.edu/~karavan
Computer Sciences Department
University of Wisconsin -- Madison
Publications

What is a Parallel Performance Tool?

- **Goal**
  - Provide Meaningful Feedback about Application Behavior

- **Insert Instrumentation**
  - Hardware vs. Software
  - Profiling vs. Tracing

- **Analyze Application Performance**
  - Descriptive Feedback (visualization)
  - Prescriptive Feedback
What is a Parallel Performance Tool?
What is a Parallel Performance Tool?
What is a Parallel Performance Tool?

- “What is the scaling behavior of my code?”
- “How do these results compare to other platforms?”
- “Did performance of foo improve when we compiled with -O3? By how much?”
- “Are the performance requirements being met?”
- “What’s the max I/O wait time we’ve seen for runs on more than 16 nodes?”
- “How accurate was my predictive model?”
What is a Parallel Performance Tool?

- Current Performance Tools: Performance of a single program run
- The Actual Evaluation Tasks: Inherently multi-execution
  - Performance Tuning
  - Software Evolution
  - Porting to new platforms
  - Different Data Sets
  - Benchmarking
  - Regression Testing
  - Model and Simulation Validation
  - Dynamic runtime environments
Our Solution: Experiment Management

- **Performance Tuning as Scientific Experimentation:**
  - An experiment includes one or more instrumented program runs
  - Data: Structural, Performance, Index
  - Automation
- **Name, store, analyze, visualize data from many runs**
- **Performance Data is heterogeneous**
The Multi-Dimensional Experiment Space

- Input data set
- Code version
- Underlying hardware (Processors, Interconnect, Memory)
- Underlying software (libraries)
- Language
- Compiler
- Optimization level
Our Solution: Experiment Management

- **The Program Space**
  - Each program execution is a point
  - Dimensions: platform, version, input data, language, etc.
  - Data: structural, performance, metadata

- **Operations**
  - Automated analysis of points, vectors, planes
    - Simple Queries
    - Comparison Operations
  - Schema Evolution
Talk Outline

Motivation

Representing and Comparing Structural Data

Representing and Comparing Performance Data

Performance Diagnosis Using Multi-Execution Data

Future Work and Conclusions
Representing an Execution

- A resource is a representation of a logical or physical component of a program execution -- Module1, Main, Process3, node24
Representing an Execution

- A resource is a representation of a logical or physical component of a program execution -- Module1, Main, Process3, node24
- A resource hierarchy $R$ is a tree of the form $R = (r, T)$
Representing an Execution

- A resource is a representation of a logical or physical component of a program execution -- Module1, Main, Process3, node24
- A resource hierarchy $R$ is a tree: $R = (r, T)$
- A resource name is formed by concatenating the labels along the unique path within the resource hierarchy from the root to the node representing the resource: /Code/Module1/Main
Representing an Execution (cont’d)

- A Program Event is a representation of a program run or execution. It is a forest: $E = \{R_0, R_1, \ldots R_n\}, n \geq 0$
Representing an Execution (cont’d)

- A Program Event is a representation of a program run or execution. It is a forest: \( E = \{R_0, R_1, \ldots, R_n\}, n \geq 0 \)
- A focus \( F \) is formed by selecting one resource node from each of the resource hierarchies \( R \):
  \[
  \langle/\text{Code/Module1/}\text{Main,/,Machine,/,SyncObjects/Semaphores/One}\rangle
  \]
Comparing Program Events

• **Goals:**
  – What has changed in the code? In the run-time environment?
  – How different are the two executions?

• **The Structural Difference Operator**
  – A top-down comparison of two or more Program Events
Comparison Operators: the Structural Difference
Comparison Operators: The Structural Difference

\[ E_1 \]
-\[ E_2 \]
= \[ \{ E_1, E_2 \} \]
Case Study: Changing Communication Libraries

- Parallel message-passing FFT code versions
  - MPI
  - PVM
- **Goal:** provide feedback on the structural changes in the application code to the scientist
- **Method:** construct Program Events using Paradyn data and apply Structural Difference Operator
- **Result:** we focus attention directly to the structural changes which resulted from changing libraries
Talk Outline

Motivation

Representing and Comparing Structural Data

Representing and Comparing Performance Data

Performance Diagnosis Using Multi-Execution Data

Future Work and Conclusions
Representing Performance Data

- **We represent each performance result** \( r \) **as** \( P(e, m, f, t) \):
  - \( e \): a program execution
  - \( m \): the metric, a measurable execution characteristic
    - CPU time, Synchronization waiting time
  - \( f \): the focus for this performance measure
    - CPU time for focus \(<\text{Code/Module1/Main, /Machine, /Process}>\)
  - \( t \): the time interval

- **A performance result may be a simple scalar or more complex object**: list of values, Paradyn histograms, traces
Representing Performance Data

Focus provides a partial ordering of the data:
Comparison Operators: The Performance Difference

<Code, /Machine, /SyncObjects>

<Code/?, /Machine, /SyncObjects>
<Code, /Machine/?, /SyncObjects>
<Code, /Machine, /SyncObjects/>?

<Code/?, /Machine, /SyncObjects>
<Code/?, /Machine/?, /SyncObjects>
<Code/?, /Machine, /SyncObjects/>?
<Code, /Machine/?, /SyncObjects>
<Code/?, /Machine, /SyncObjects>
<Code, /Machine, /SyncObjects/>?
Comparison Operators: The Performance Difference

- Starting point is Program Event Group:
Comparison Operators: The Performance Difference

- Traverse performance results in a top-down manner
- Apply Distance Metric $d$ to each pair of performance results:
  
  $d = r_1 - r_2$
  
  $d = P(e_1, m_1, f_1, t_1) - P(e_2, m_2, f_2, t_2)$

- If $d$ significantly large, mark different and continue comparison
- Stop when $d$ is insignificant or zero
- Display changes in Performance Difference Display
Case Study: A Performance Tuning Study

• **Original Study**
  - Protein-folding simulation from UW-Madison Chemical Engineering ported to Wisconsin COW
  - Data: Paradyn histograms gathered using Paradyn/Blizzard
  - 13,000 resource nodes, 3-4 person-weeks

• **Our Goal**
  - focus attention to performance changes between versions in tuning study using Performance Difference Operator
Performance Difference Display

Program Events: (fold4_1, fold4_2) Metric: memoryBlockingTime

WholeProgram
  process1
    GM
    GM->part0
    GM->part1
    GM->part2
    GM->part3
  process2
    GM
    GM->part0
    GM->part1
    GM->part2
    GM->part3
  process3
  process4
Talk Outline

Motivation

Representing and Comparing Structural Data

Representing and Comparing Performance Data

Performance Diagnosis Using Multi-Execution Data

Future Work and Conclusions
Performance Diagnosis Using Multiple Program Events

• **Tools for Automated Performance Diagnosis**
  – ATExpert, MPP Apprentice (CRAY)
  – Poirot (Helm, Malony - U of Oregon)
  – Paradyn’s Performance Consultant (Hollingsworth, Miller)

• **Collaborative Effort**
  – APART (FZ Juelich, UW-Madison, U of Oregon, etc.)
Performance Diagnosis Using Multiple Program Events

• **Improve Existing Methods -- Performance Consultant**
  – Directing the Search Strategy
  – Data Collection across multiple program runs

• **New Methods**
  – Automated Performance Difference
Paradyn’s Performance Consultant (Some Background)

- **Experiment:**
  - Why + Where? Threshold
- **Search Expansion**
  - Refinement
  - Dynamic Instrumentation
- **Perturbation**
  - Information vs uncertainty
- **Application Familiarity**
Performance Diagnosis Using Multiple Program Events

- **Goals**
  - Shorten time to find important bottlenecks
  - Decrease unhelpful instrumentation
  - Determine precise location of all significant bottlenecks
Performance Diagnosis Using Multiple Program Events

- **Adding Prior Knowledge to the Performance Consultant**
  - Prunes
    - remove resource hierarchy subtree
    - what to prune?
    - robustness
  - Priorities
    - order the PC search
    - partition hypothesis-focus pairs
  - Thresholds
    - too low --> high cost
    - too high --> miss behaviors
Time to Find Bottlenecks

- Execution Time (in seconds)
- Percent of Bottlenecks Reported

Graph showing the time to find bottlenecks with different strategies:
- Base
- Prunes
- Priorities
- Prunes and Priorities
Talk Outline

Motivation

Representing and Comparing Structural Data

Representing and Comparing Performance Data

Performance Diagnosis Using Multi-Execution Data

Future Work and Conclusions
Thesis Contributions

• Representation for the set of executions collected over the life of an application

• Techniques for automatically describing the structural and performance differences between two runs of a program

• Automated runtime performance diagnosis using historical data
Acknowledgments

This research is supported by:

• Department of Energy
• NASA Graduate Student Researchers Program
• National Science Foundation