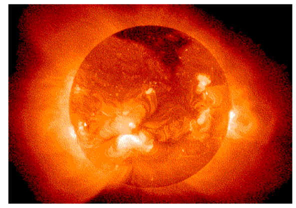
Experiment Management Support for Performance Tuning

Karen L. Karavanic
karavan@cs.wisc.edu

Computer Sciences Department
University of Wisconsin
1210 W. Dayton Street
Madison, WI 53706-1685

Current performance tools can shine a lot of light on execution characteristics:

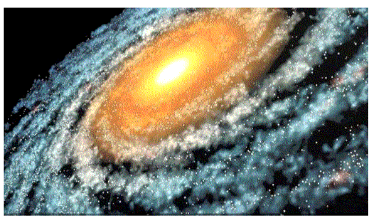


There are good research and commercial tools that can

profile, graph, animate, visualize, auralize, use VR

to tell you a lot about what your program is doing.

But we're missing the big picture



Even if you measure performance in great detail, that's only one small point in a multi-dimensional space.

Motivation

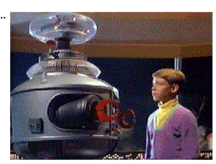
Current Parallel Performance Tools: performance of a single program run

The actual evaluation tasks: inherently multi-execution:

- Performance Tuning
- Software Evolution and Porting to New Platforms
- Different Data Sets
- Benchmarking
- Regression Testing
- Model and Simulation Validation
- Dynamic Run-time Environments

Where Does this Leave Programmers?

Lost in a multi-dimensional experiment space...



The axes include:

- repeated executions
- input data sets
- program revisions
- program versions
- number of nodes
- underlying hardware (processor, network, memory)
- underlying software (communication libraries, math libraries, compiler)
- programming languages
- optimization levels

Our Solution: A Two-Part Approach

1. The Program Space
   - store data from multiple program runs
   - manipulate, query, and visualize data

2. Points in the Space
   - automated analysis of points, vectors, planes
   - each program execution is a point

The Program Space

- Enclosing Space
  - metadata: attributes of each execution
- Each Point
  - Single Program Execution
  - Data: Structural, Performance
- Operations Over the Space
  - queries:

    "which executions ran on the SP/2?

    "show me average CPU time for function foo, only for versions compiled with -O3"

    "what is maximum I/O wait time for runs on more than 16 nodes?"

    "show me CPU time for whole program versus number of CPU's, for all executions"

The Program Space (cont'd)

- Operations Over the Space (cont'd)
  - schema evolution:
    "add a new dimension to describe architecture."
- Comparing Points in the Space
  - "what is different about the performance of these two program runs?"
  - "what code differs between these two versions?"

Talk Overview

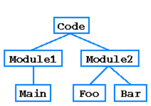
1. Points in the Program Space ("Program Events")
   - Representing and Comparing Structural Data
   - Representing and Comparing Performance Data
2. Performance Diagnosis using multiple Program Events
   [work in progress]
3. Research Directions

Representing an Execution: The Program Event

- A *resource* is a representation of a logical or physical component of a program execution.
  Module1, Main, Process3, node24

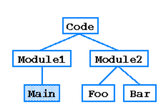
Representing an Execution: The Program Event

- 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) where r is a resource and Tis the set of all children of r



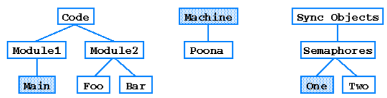
Representing an Execution: The Program Event

- 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) where r is a resource and Tis the set of all children of r
- 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.
  Example: /Code/Module1/Main



Representing an Execution: The Program Event (cont)

- A *Program Event* is a representation of a program run or execution. It is a forest composed of zero or more unique resource hierarchies:
  E = {R0, R1, ... Rn}, n >= 0
- A *focus* F is formed by selecting one resource node from each of the resource hierarchies R:
  </Machine, /Code/Module1/Main, /Machine, /SyncObjects/Semaphores/One>



UNIVERSITY OF WISCONSIN MADISON

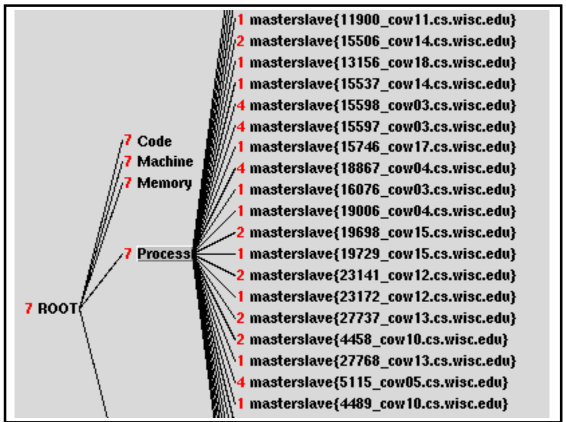
Experiment Management...... Page 13

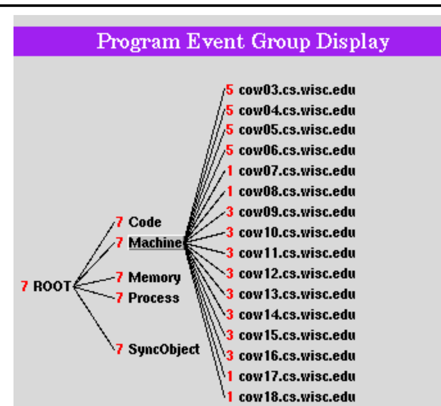
Comparing Executions

- Goals:
  - What has changed in the code? In the run-time environment?
  - By how much?
- The Structural Difference Operator
  - A top-down level-wise comparison of resource trees
  - More than two Executions?
- The Performance Difference Operator
  - Which performance measures have changed? By how much?

UNIVERSITY OF WISCONSIN MADISON

Experiment Management...... Page 14



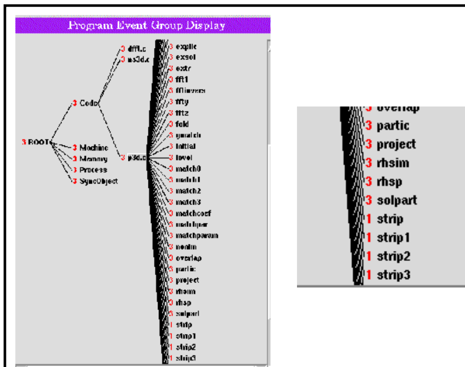


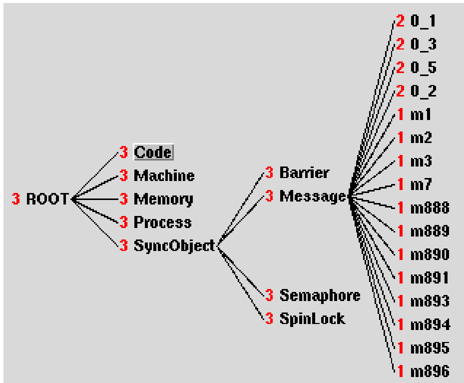
Case Studies 1: Changing Communication Libraries

- Parallel message-passing FFT code written using PVM, ported to MPI
- Goal: provide feedback on the structural changes in the application code to the scientist
- Method: gather data using Paradyn and apply our Structural Difference Operator
- Result: *we can focus attention directly to the structural changes which resulted from porting to MPI*

UNIVERSITY OF WISCONSIN MADISON

Experiment Management...... Page 18





Representing Performance Data

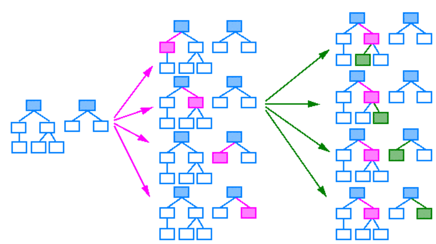
We represent each performance result r = P(e, m, f, t) where :

- e: a program execution
- m: the metric, a measurable execution characteristic: CPU time, SyncWaitTime
- f: the focus for this performance measure, eg
  CPU time for focus **</Code/Module1/Main, /Machine, /Process >**
- t: the time interval, specifies when during an execution the data was collected.

A performance result (r) may be a simple scalar or more complex object
tables of values, time histograms, traces

Representing Performance Data

Focus provides a partial ordering of the data:



Comparison Operators: The Performance Difference

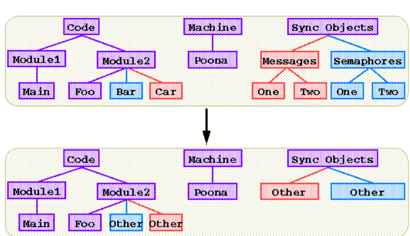
```
</Code, /Machine, /SyncObjects>

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


Comparison Operators: The Performance Difference

- Starting point is Program Event Group:

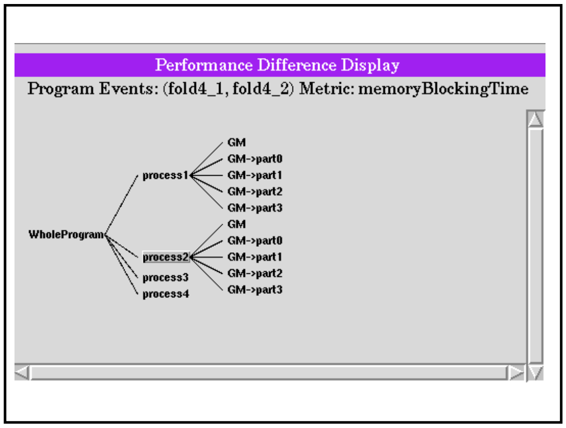


Comparison Operators: The Performance Difference

- Traverse performance results in a top-down manner
- Apply Distance Metric d to each pair of performance results:

  d = r1 - r2
  d = P(e1, m1, f1, t1) - P(e2, m2, f2, t2)

- If d significantly large, mark different and continue comparison
- Stop when d is insignificant or zero
- Display changes in Performance Difference Display



Performance Diagnosis Using Multiple Program Events

- Automated Performance Diagnosis
  - ATExpert, MPP Apprentice (CRAY)
  - Poirot (Helm, Malony - U of Oregon)
  - Projections:Expert (Charm - Sinha, Kale)
- Run-time Performance Diagnosis
  - Paradyn's Performance Consultant (Hollingsworth, Miller)
  - Problems:
    - toss out useful data and start all over each time
    - may miss behaviors of interest
    - feedback based on single execution

Performance Diagnosis Using Multiple Program Events

- New Approach:
  - Use historical knowledge
  - Faster Diagnosis
  - Less chance of missing useful data
- Goals:
  - How does behavior of this piece differ between Program Events?
  - Are there new (different) pieces of interest?
  - What is the behavior of this unexamined piece?

Performance Diagnosis Using Multiple Program Events

- Add Prior Knowledge to the Performance Consultant
  - Use historical data to generate priorities for hypotheses
  - Performance Consultant uses these priorities to order search
  - Decrease probability of missing important data
- Diagnosis as Comparison
  - Use historical data to generate hypotheses
  - Sample performance during execution
  - Runtime Performance Difference

Ongoing Research in Experiment Management Support for Performance Tuning

- Underlying storage mechanism
- Expand the set of difference operators:
  - different performance results: AIMS trace data
  - different types of analysis: benchmarking studies
  - better structural comparisons
- Historic performance data in runtime performance diagnosis

Ongoing Research in Experiment Management

- Applications to scientific experimentation in general
- Collaborative Research with M. Livny and Y. Ioannidis
- Visualization Techniques
- DOE2000 Laboratory Notebook

