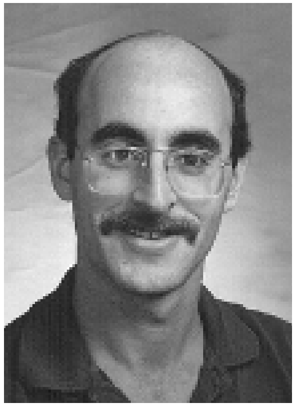
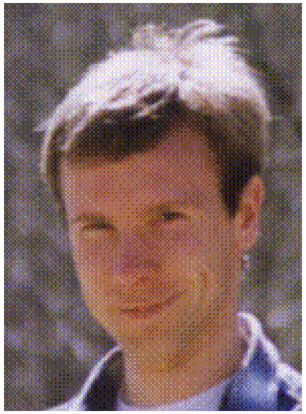
Paradyn Parallel Performance Tools



Barton P. Miller & Brian J. N. Wylie
(bart@cs.wisc.edu , wylie@cs.wisc.edu)

`paradyn@cs.wisc.edu`
`http://www.cs.wisc.edu/~paradyn`

Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706-1685, USA



Students:

Trey Cain, Chris Chambreau,
Karen Karavanic, Dan Nash,
Tia Newhall, Phil Roth,
Brandon Schendel, Chris Serra,
Ariel Tamches, Zhichen Xu,
Vic Zandy; Bryan Buck (UMD)

Associated Researchers:

Jeff Brown (LANL)
Karsten Decker (CSCS/SCSC)
Carlos Figueira (USB-Venez.)
Ian Foster (ANL)
Jeff Hollingsworth (UMD)
Douglas Pase (IBM)

Project Alumni:

Mark Callaghan, Jon Cargille
Marcelo Gonçalves, Bruce Irvin
Oscar Naìm, Sunlung Suen
Ling Zheng

Paradyn technology: Dynamic Instrumentation

A machine-independent interface to machine-level instrumentation and control!

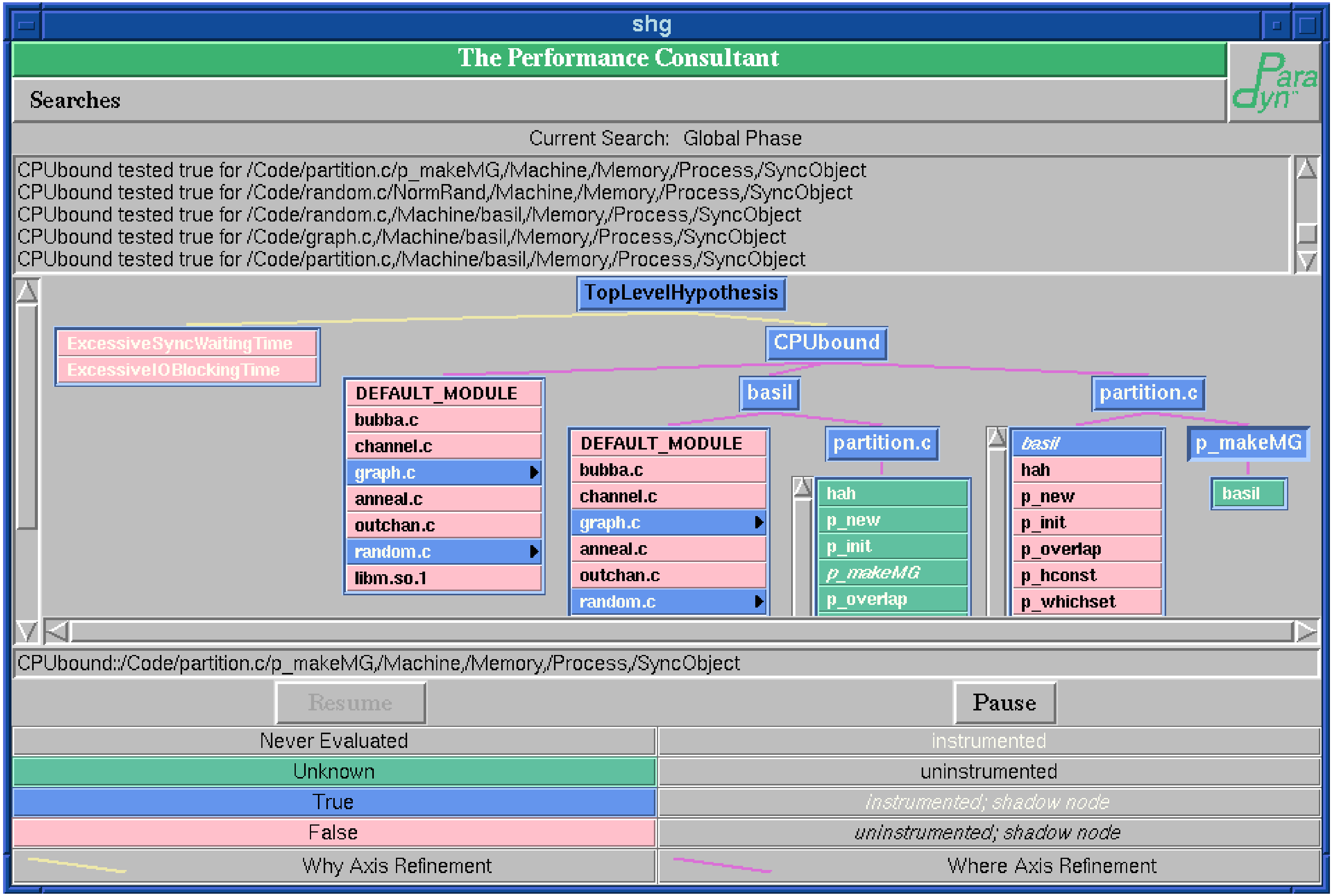
- ❑ On-the-fly insertion, removal and modification of instrumentation in the application program, during its execution.
  - No need for expensive (often impossible) recompilation nor relinking
  - Instrumentation only inserted when and where currently needed (and removed afterwards)
- ❑ Selected instrumentation points (function entry, exits and callsites) re-written and/or patched to jump to an instrumentation framework (known as a “base trampoline”) which now contains the relocated instructions overwritten in the original function.
- ❑ Instrumentation snippets synthesized from an abstract specification based on primitives and predicates, inserted into their own mini-trampolines daisy-chained from the base trampoline.
- ❑ Expressive metric definitions through the Metric Description Language (MDL)
- ❑ Dynamic monitoring and control of instrumentation overhead/intrusiveness

Paradyn technology: Performance Consultant

Automated, portable, scalable decision support for execution bottlenecks!

- Answers three key questions about a program's execution:
  - **Why** is it slow or inefficient? (synchronization, I/O, CPU utilization, memory, ...)
  - **Where** is this occurring? (machine, process, thread, module, function, tag, ...)
  - **When** does it occur? (initialization, computational kernel, checkpointing, ...)
- Regular structure created specifying the causes of possible bottlenecks makes automated searches possible
  - Hypotheses based on user-specified thresholds:
    e.g., synchronization blocking time < 25% of execution time
  - Evaluating bottleneck hypotheses triggers dynamic instrumentation requests (activating and deactivating instrumentation)
- Instrumentation costs relate the number of actively considered hypotheses to the instrumentation overhead and execution perturbation
- Identifies a focus or foci for more in-depth execution analysis and visualizations

Performance Consultant search in progress ...



Current Research: Improved Performance Consultant

Problem:

- Code/module/function hierarchy too wide for efficient searches: (system) libraries have 1000s of (unexecuted/uninteresting) functions...
- Module instrumentation not cheaper than function instrumentation: all functions must be instrumented in each module of interest
- Exclusive metrics more expensive than inclusive metrics: entry + exit(s) ***plus*** before and after every call site
- Search unrelated to actual program execution

New approach:

- Search based on dynamic call graph, using inclusive metrics

References:

- "A new scheme for Performance Consultant searches in the code hierarchy," Matthew Cheyney, http://www.cs.wisc.edu/~paradyn/PW98_notes/mcheyney_grays.ps.gz
- "Dynamic control of performance monitoring on large-scale parallel systems," Jeffrey K. Hollingsworth and Barton P. Miller, *Int'l Conf. on Supercomputing (ICS'93, Tokyo, Japan)*, July 1993

Current Research: Experiment Management

Problem:

- Performance data available from multiple runs (huge multi-dimensional space): simulations, benchmarking, tuning, regression testing, etc.

Approach:

- Provide infrastructure for manipulation and management of performance data
- Automatically compare execution data from multiple runs
- Faster bottleneck location initiated from historical execution analyses
- Useful for typical software development. Crucial in meta-computer environment: a "laboratory notebook" for performance studies.

References:

- "Experiment management support for performance tuning," Karen L. Karavanic and Barton P. Miller, *Proceedings of SC'97 (San Jose, CA, USA)*, Nov. 1997
- New report/paper in preparation
- **`karavan@cs.wisc.edu`**

Current Research: Fine-grained, adaptive instrumentation

Problem:

- Instrumentation is currently only medium-grained (function+callsite level)
- Instrumentation trampolines are multiple instruction jump sequences
- Inapplicable for instrumenting OS kernels

Approach:

- Fine-grained instrumentation (block level, atomic jump patching)
- Dynamic generation of customized/optimized code

References:

- “Fine-grained dynamic instrumentation of commodity operating system kernels,” Ariel Tamches and Barton P. Miller, *3rd Symp. on Operating Systems Design and Implementation (OSDI’99, New Orleans, LA, USA)*, Feb. 1999
- “Using dynamic kernel instrumentation for kernel and application tuning,” Ariel Tamches and Barton P. Miller, currently under review
- “Dynamic instrumentation of threaded applications,” Zhichen Xu, Barton P. Miller and Oscar Naìm, accepted for *PPoPP’99 (Atlanta, GA, USA)*, May 1999
- **`tamches@cs.wisc.edu`**, **`zhichen@cs.wisc.edu`**

Current Research: Dynamic Instrumentation API

Approach:

- Provide basic substrate for building new tools: ***DynInstAPI***
- Library of C++ classes for machine-independent mutatee code analysis, execution control, and run-time code generation and insertion into mutatee
- Collaboration with University of Maryland, IBM (DPCL), etc.
- Basis for CSCS/U.Basel's FIRST and TUM's OCM tools activities
- Form the basis for an emerging open standardization effort: next meeting during Paradyn Week (26 March 1999, Madison, WI, USA)

References:

- "An Application Program Interface (API) for runtime code generation," Jeffrey K. Hollingsworth, **http://www.cs.umd.edu/projects/dyninstAPI/**
- "MDL: A language and compiler for dynamic program instrumentation," Jeffrey K. Hollingsworth, Barton P. Miller, Marcelo J. R. Gonçalves, Oscar Naìm, Zhichen Xu and Ling Zheng, *5th Int'l Conf. on Parallel Architectures and Compilation Techniques (San Francisco, CA, USA)*, Nov. 1997
- **`hollings@cs.umd.edu`**

Paradyn™

UNIVERSITY OF WISCONSIN MADISON

Paradyn status

Paradyn is a research prototype for analyzing complex, long-running, large-scale, multi-language, multiple process/processor, heterogeneous, distributed applications!

- ❏ Latest released versions: *Paradyn* v2.1 (May 1998); *DynInstAPI* v1.2 (Sep. 1998)
  - Supported platforms: Solaris (SPARC & x86), WindowsNT (x86), AIX (RS6000)
  - Ports in progress: Linux (x86), Irix (MIPS), Digital Unix (Alpha)
  - Programs: C, Fortran, PVM, MPI, ...
- ❏ Distribution of sources, binaries and manuals free of charge for research use

References:

- "The Paradyn Parallel Performance Measurement Tools," Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam and Tia Newhall, *IEEE Computer* ***28***, 11, pp.37-46, (Nov. 1995)

- **http://www.cs.wisc.edu/~paradyn/**
- **`paradyn@cs.wisc.edu`**

Possible APART goals/achievements

- ❑ Set a research agenda bigger than (traditional) scientific/numerical computing!
  - Exponential growth in distributed database systems and information servers, which have complex and poorly-understood performance characteristics

- ❑ Demonstrate the applicability of automated & automatic performance analysis in the wider context of industrially-relevant parallel and distributed applications