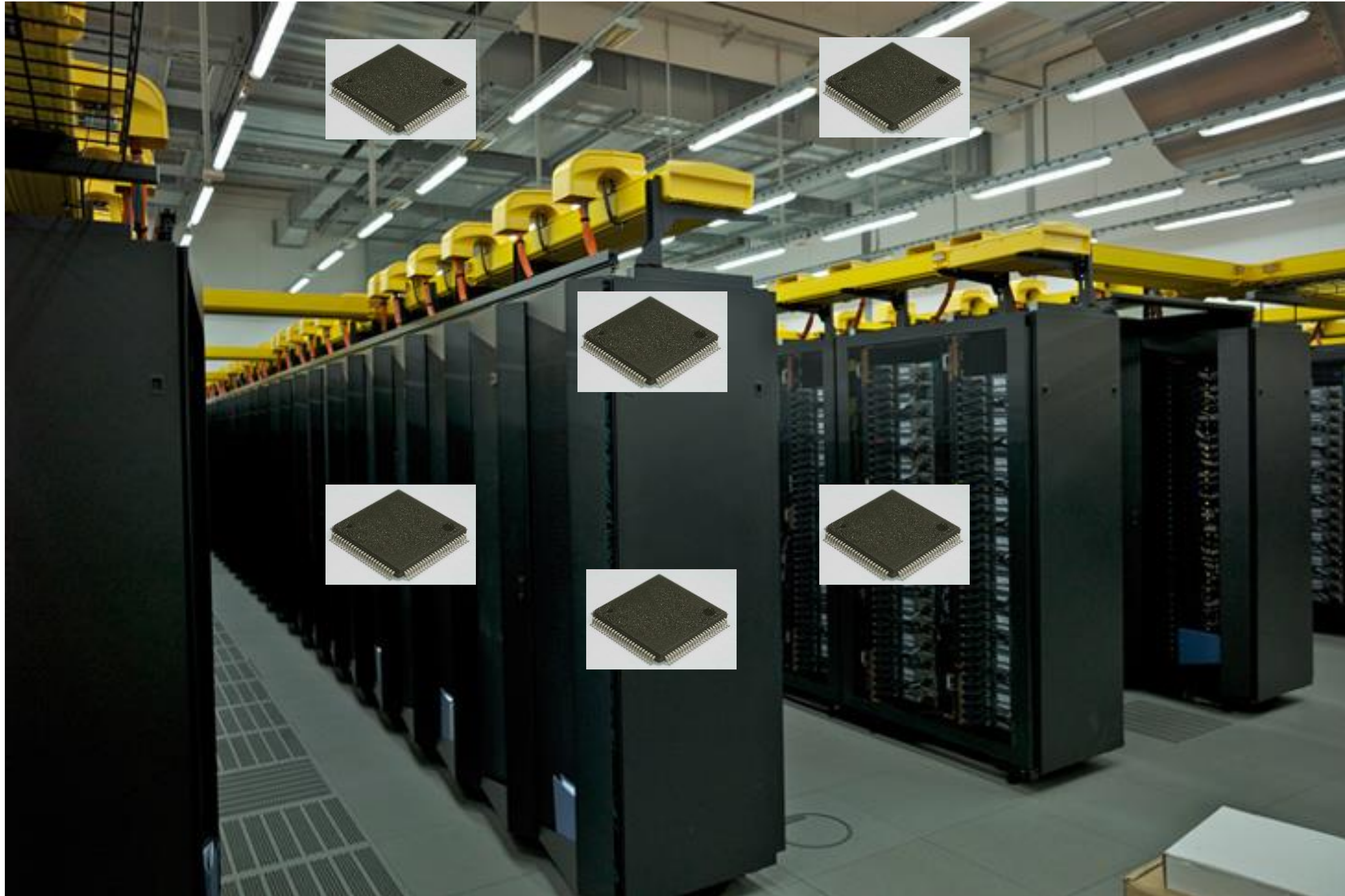
On the Shoulders of Giants

Can we Learn Diagnosis from SoC's Larger Siblings?

Philipp Wagner
Institute for Integrated Systems
Technische Universität München

Multicore Application Debugging Workshop
MAD 2014
Oct. 8, 2014 @ Athens, Greece

From HPC to SoC



Learn Diagnosis from HPC

crash, data loss, wrong computation, …

too much communication, inefficient threading, …

hard fail — inefficiency

debugging

profiling, performance engineering

Why learn from HPC?

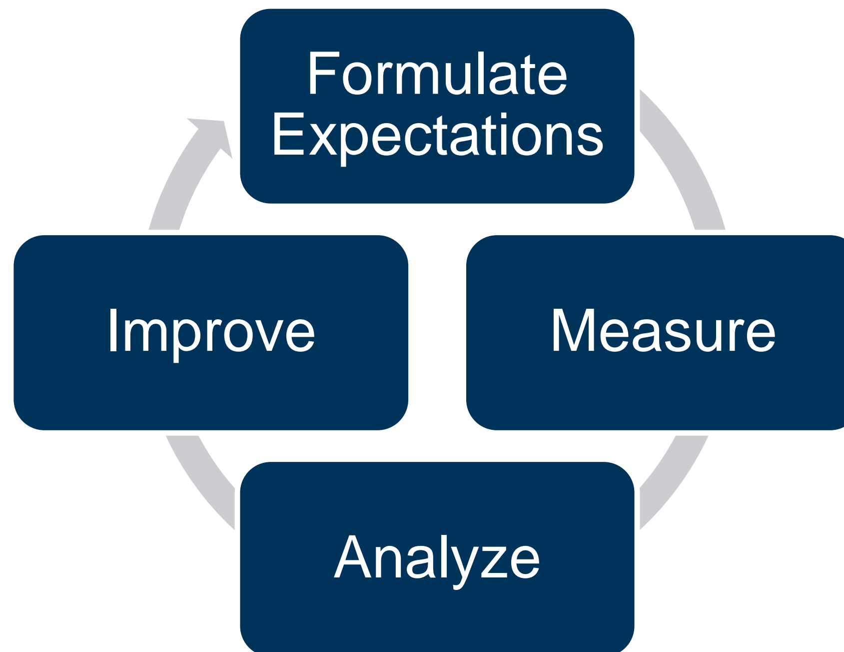
SuperMUC today: ~155,000 cores
SoC today: ~16 cores

SoC in 202… 000 cores
SoC in 204…: 155,000 cores

can we handle diagnosis for 16 cores?

ITRS Roadmap 2011, 1.4x growth of core count in networking per year

The Circle of Performance Diagnosis



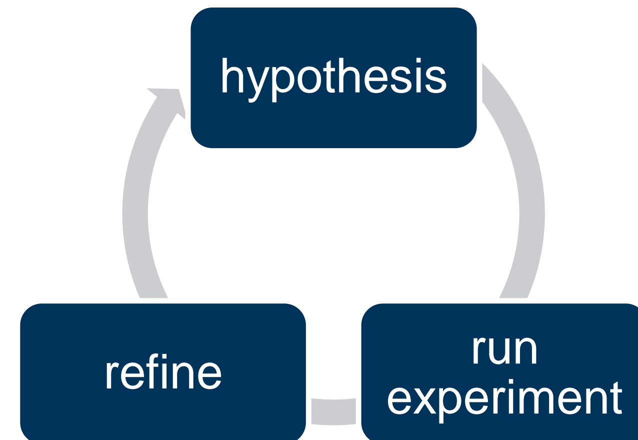
Collecting Data – Approaches

- "Big Data"
  - Collect everything and analyze later
  - in SoC: full system trace; not sufficient off-chip bandwidth

- "Smart Data"
  - determine a minimal set of data to collect
  - performance experiments: search in the time domain

Collecting Data – Performance Experiments

- working **hypothesis** why the system could be performing poorly
  `L2 cache misses are exceeding a threshold`
- test hypothesis: **run experiment**
  - determine focus: core, process, thread, function, …
  - configure data collection
  - run application
- evaluate experiment
  - hypothesis holds? **refine** experiment (adjust focus)
- result: search tree



SoC Reuse Check
applicable to iterative software on SoCs (control loops, stream processing,…)

Describing Performance Problems – Why?

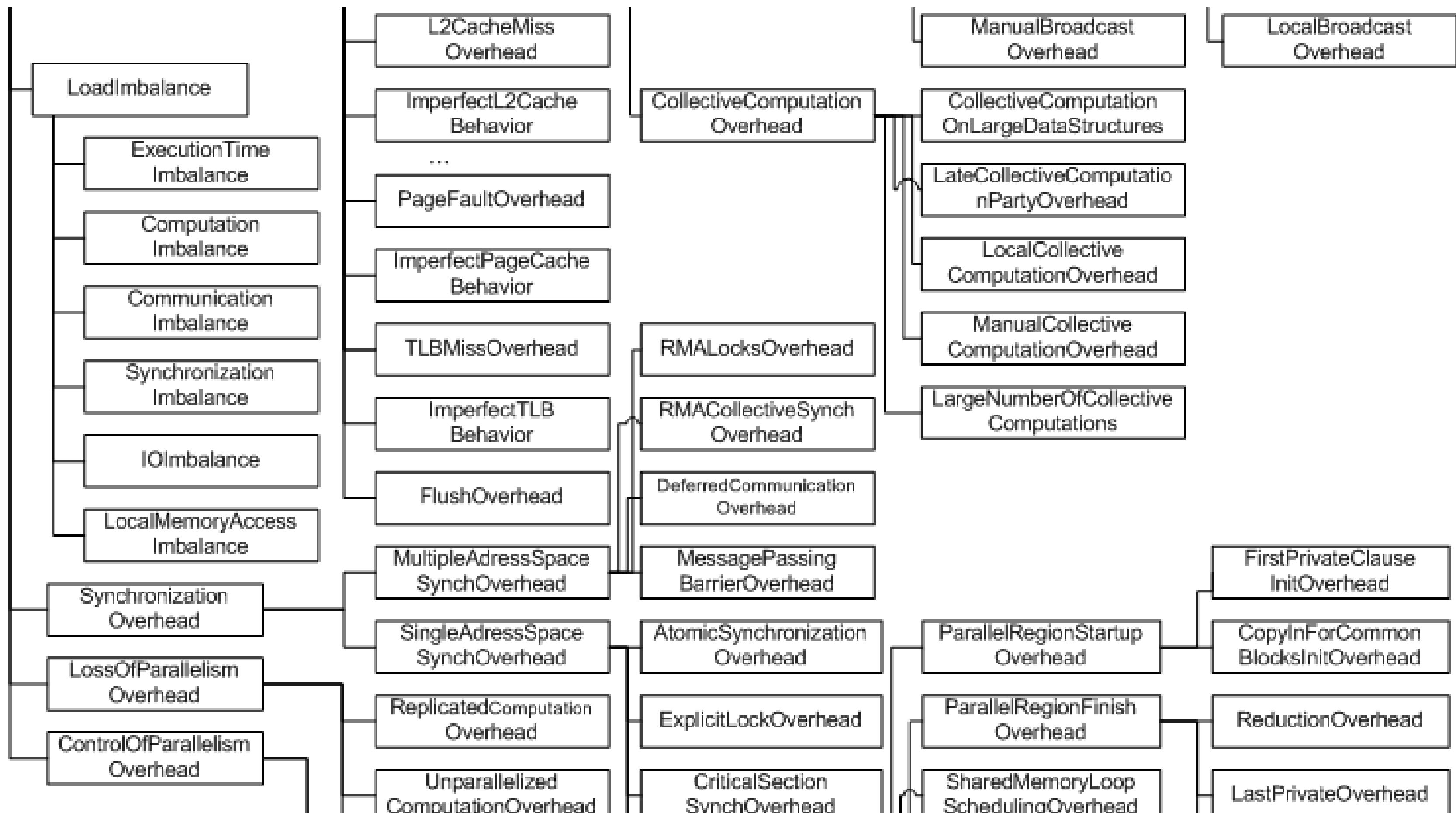
- make performance expectations explicit
  - others - “expert knowledge”
  - your own
- create detailed metrics describing system performance
- prerequisite to automation, i.e. to increased productivity

Describing Performance Problems – How?

- **pattern matching** in performance traces
  - EDL (regular expression matching, 1983), EARL (“analysis scripts”, 1999)
- **rule-based specification** of performance bottlenecks
  - set of hypothesis, proof and refinement rules
- **formal language** for performance properties: APART Specification Language (ASL)
  - Properties consist of a description of the required input data, a boolean condition (property holds?), a confidence metric, and a severity score to allow for filtering
  - no known implementation, Periscope only contains hand-coded C++ describing performance properties
  - Java version: JavaPSL (no code available either)

SoC Reuse Check
be inspired by the ideas

Ideas for Performance Metrics



Pert of a graphic showing JavaPSL planned performance metrics (AKSUM project)
http://www.dps.uibk.ac.at/projects/aksum/Overview.php

Collecting Data – Instrumentation

- Source to Source
  - “automated `printf()` insertation”
  - Program Database Toolkit (PDT), Univ. of Oregon
- Binary/dynamic instrumentation
  - Dyninst toolkit, Cobi (from Scalasca)
- Compiler instrumentation
  - using compiler plugins (supported by LLVM and GCC)
- Interposition/Library Wrappers
  - use GCC’s/clang’s `-finstrument-functions` and a preloaded library
- Interpreter/VM Instrumentation
  - JVM Tool Interface (JVMTI)
- Hardware Performance Counters
- Configurable debugging hardware
  - ARM CoreSight, Infineon MCDS, …

SoC Reuse Check
already good HW debug support in production, powerful SW instrumentation ready for use

Visualizing Data

- Oldie but goodie: **call graph**
  - can be extended to system level (e.g. node, core, process, thread, function, basic block)
  - enhancement: prune/collapse subgraphs with no relevant information
- **Timeline view**
  - see barriers and patterns
  - Vampir NG, Paraver

Visualizing Data – Timeline View

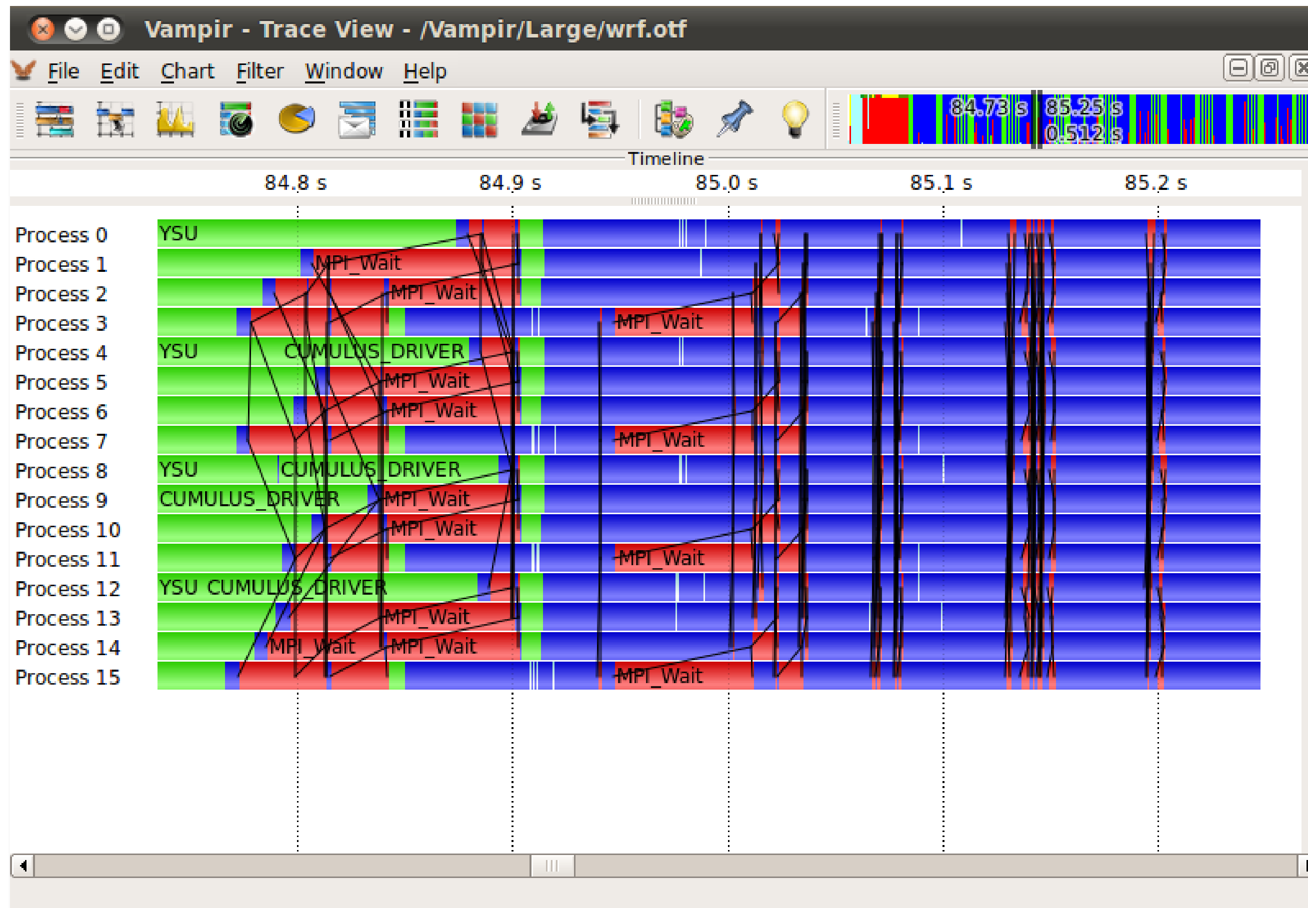


Image from the Vampir Manual

Visualizing Data

- Oldie but goodie: **call graph**
  - can be extended to system level (e.g. node, core, process, thread, function, basic block)
  - enhancement: prune/collapse subgraphs with no relevant information
- **Timeline view**
  - see barriers and patterns
  - Vampir NG, Paraver
- **Multi-dimensional views**
  - Cube (Scalasca): three dimensions: performance metric, call path, system resource

Visualizing Data – Multi-Dimensional View (Cube)

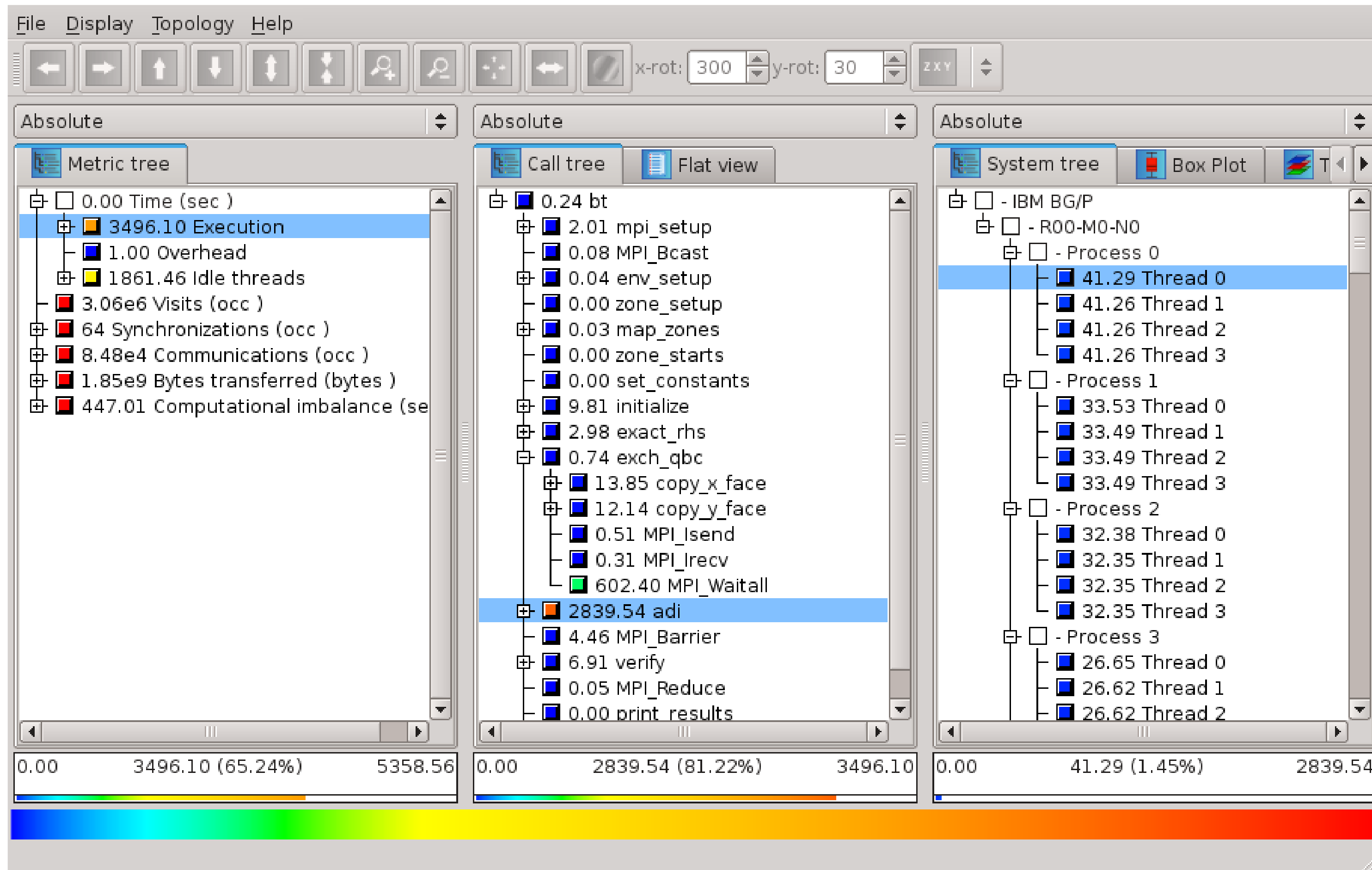
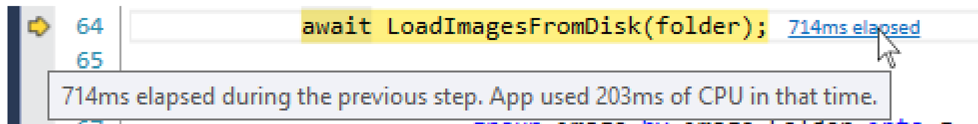


Image from the Cube 4.1 User Guide

Visualizing Data

- Oldie but goodie: **call graph**
  - can be extended to system level (e.g. node, core, process, thread, function, basic block)
  - enhancement: prune/collapse subgraphs with no relevant information
- **Timeline view**
  - see barriers and patterns
  - Vampir NG, Paraver
- **Multi-dimensional views**
  - Cube (Scalasca): three dimensions: performance metric, call path, system resource
- **Annotate source code**
  - VisualStudio PerfTips
  - Linux `perf annotate`, Valgrind/KCacheGrind



SoC Reuse Check
take as many ideas as we can!

Solving Problems

- IBM High Productivity Computing Systems Toolkit (HPCS)
  - database containing patterns of known performance problems
  - database also contains possible solutions, which can be applied directly
- AutoTune
  - extend Periscope with suggestions how to improve performance and energy efficiency
  - modify parameters, e.g. MPI buffer sizes
  - almost finished EU project

SoC Reuse Check
still a long way to go, let's see – but why not?

What can we Take from HPC to the SoC World?

- Formulate Expectations
  - Be inspired by formal languages like ASL
- Measure
  - Great tools in addition to HW tracing are available!
- Analyze
  - Some automated options for problem classification
  - Visualization remains a challenge
- Improve
  - Still mostly your job