

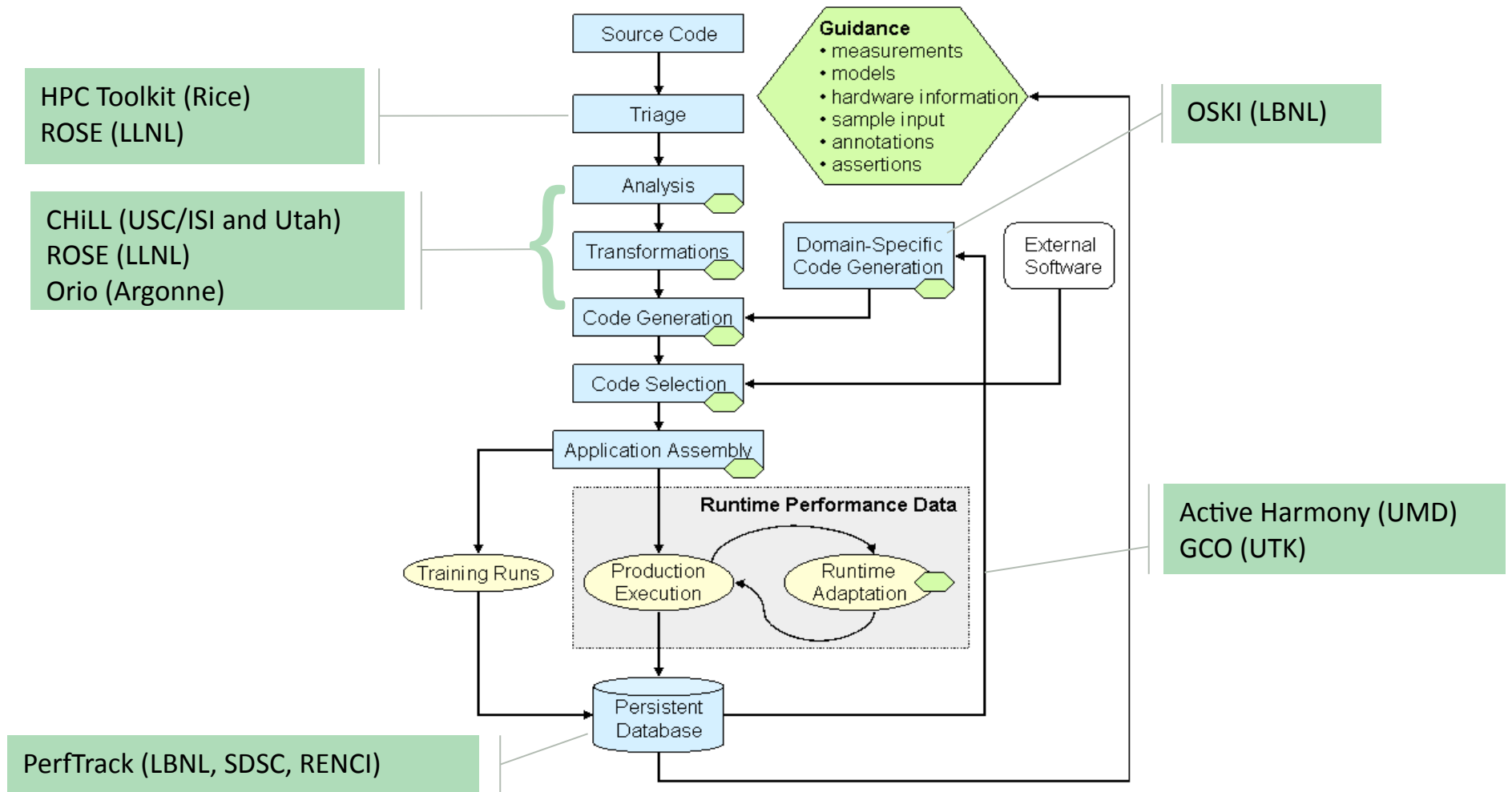
Big Questions in Autotuning

Mary Hall

August 10, 2009

Sponsored in part by DOE SciDAC Performance Engineering Research Institute (PERI), DOE Office of Science award ER25834, and NSF awards CSR-0615412 and OCI-0749360 and by Intel Corporation.

Autotuning in PERI



Autotuning: What Works?

- Important, well-understood libraries, especially for individual sockets
 - FFT and signal processing, dense linear algebra, sparse linear algebra, stencils, sorting, searching
- Parameter sweeps and variant selection
 - What parameter value(s) lead to the best performance?
 - What algorithm or implementation variant leads to the best performance?
 - Parameters and variants expressed in application or derived by compiler
- Integrated into a compiler and/or programming model
 - Most effective on dense array-based computations
 - Benefits from programmer interaction

Autotuning: Expanding its Applicability

- Broadening applicability
 - Dynamic behavior such as graph algorithms
 - When global changes to data structures, layouts, code structure are needed
- Expressing autotuning
 - Language, compiler and run-time strategies and interaction
- On-line autotuning
 - Limit decisions so practical during application execution
 - Validation
- Scaled-up autotuning
 - Evaluate alternative communication strategies
- Other concerns beyond performance
 - Energy
 - Resilience
 - (see *Software Challenges in Extreme Scale Systems*, Sarkar, Harrod and Snively, SciDAC 2009.)

Software for Exascale Computing

- Lower memory-computation ratio due to system cost
 - Petascale ~ 1 byte/FLOP
 - Exascale ~ 0.01 bytes/FLOP (projected)
- Therefore, **weak scaling** won't deliver 1000x increase in concurrency
- 1000x must come from **strong scaling** and software improvements
 - Reduce task granularity by 1000x
 - Reduce synchronization granularity by 1000x
 - Reduce communication overhead by 100x
 - Reduce sequential bottlenecks by 1000x

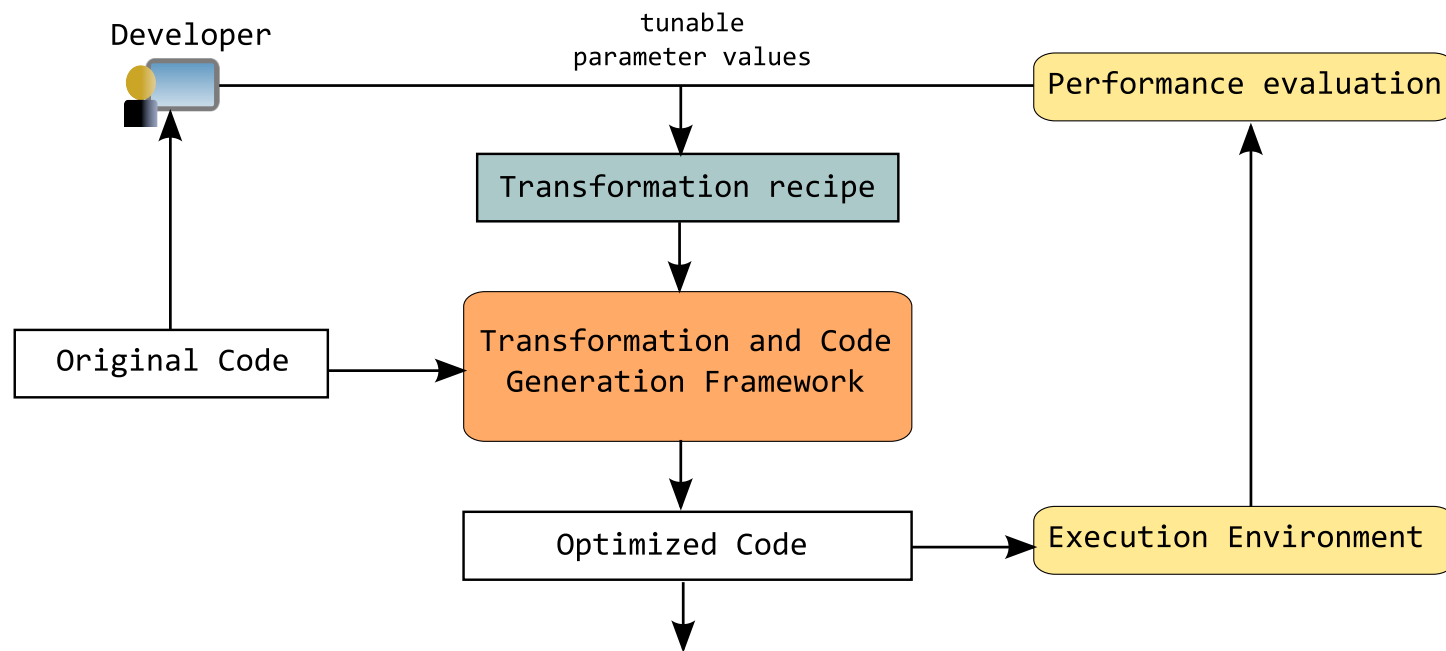
From "Towards a New Software Stack for Extreme Scale Computing," Vivek Sarkar, Exascale Energy Challenges Workshop, held at SC08, November 2008.



Rest of this Talk

1. Working with the developers:
collaborative autotuning
2. Enabling technology: common
interfaces
getting the research community
working together
3. Integrating autotuning into the
application build

1. Collaborative Autotuning



Concepts:

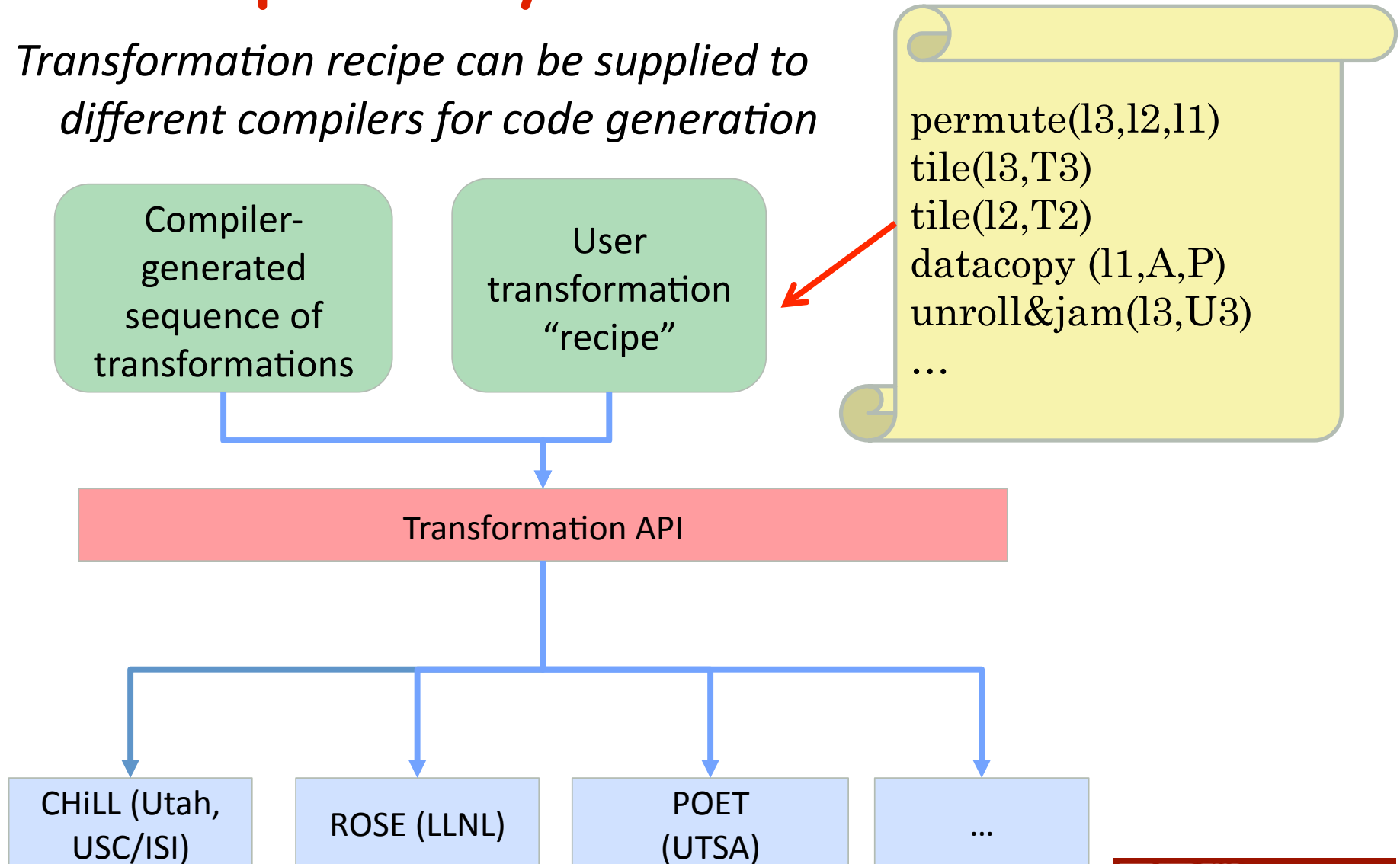
- Automate or accelerate application/library developer's autotuning process
- Developer describes parameters and variants at high level, compiler automatically generates code, search engine pinpoints best solution
- Same interface for compiler-based autotuning provides path to increased automation

2. Common Interfaces for Tool Interoperability

- Concept:
 - Leverage prior investment in tool development
 - Provide path forward for composing the best tools together for specific requirements
 - Improve research process, comparisons, experimental repeatability, ...

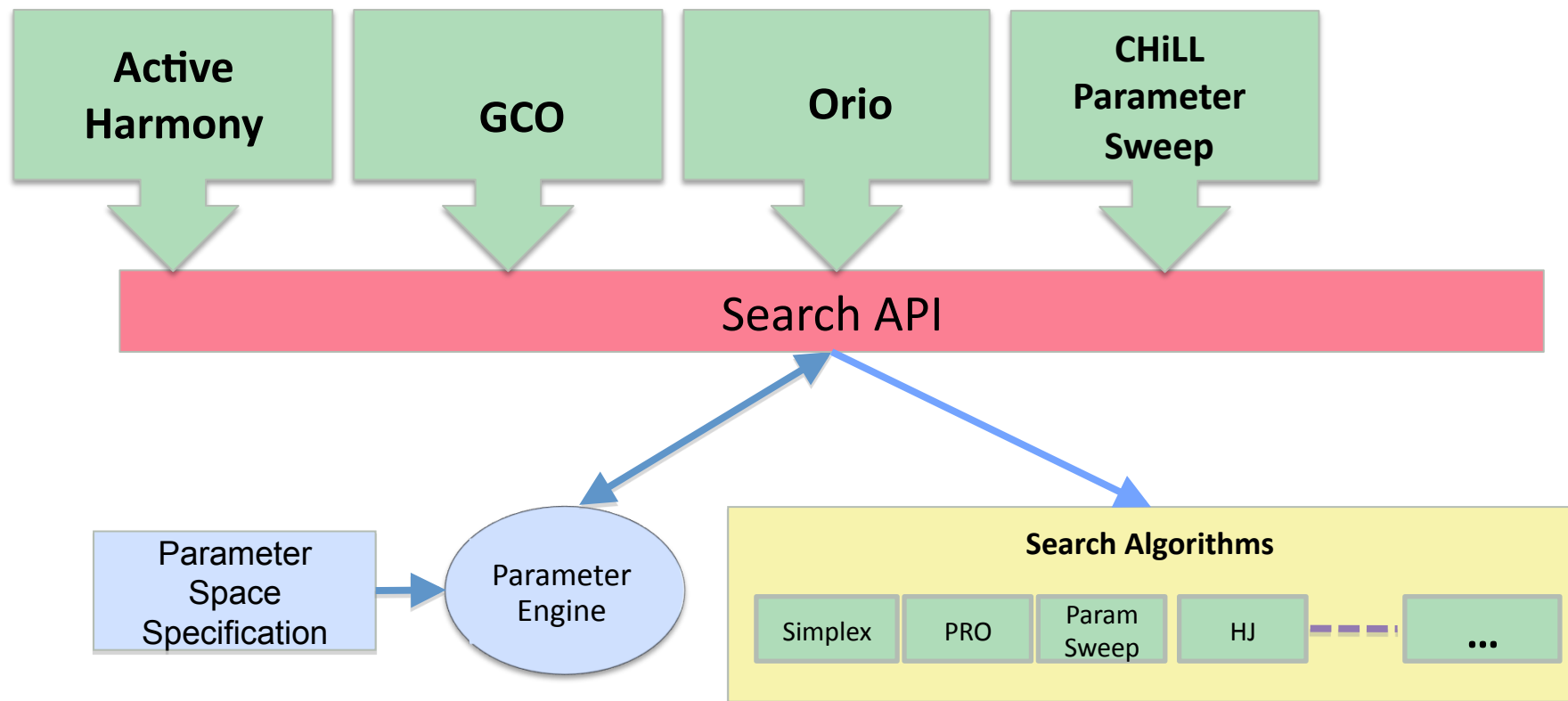
2. Common Interfaces for Tool Interoperability: Code Transformation

Transformation recipe can be supplied to different compilers for code generation



2. Common Interfaces for Tool Interoperability: Search

Search algorithms can be plugged into generalized search framework



3. Integrating Autotuning into Application Build

- Code will need to be autotuned in context of new architectures and input data sets
- Attempt multiple tools to find which one yields the best result, or compose best results for different kernels
- Solution portable across architectures, or architecture-specific portion encapsulated
- Online

Summary of 3 Core Ideas

1. Working with the developers:
collaborative autotuning
2. Enabling technology: common
interfaces
getting the research community
working together
3. Integrating autotuning into the
application build