

Standardized Search Space Representation for Empirical Optimization

Ananta Tiwari

Jeffrey K. Hollingsworth

University of Maryland, College Park



PERI Overview

- Three main research areas
 - Performance modeling and Prediction
 - Performance Engineering of high profile applications
 - Automated Performance Tuning
- Tool Integration and Sharing
 - Feeding information from modeling into Auto Tuning Frameworks
 - Bringing together Tools with complementary strengths

PERI Search Working Group

- To develop a way for search based empirical auto tuning systems to:
 - Share search specifications
 - Exchange optimization engines
- Approach:
 - Standardized Parameter Space Representation
 - A search API

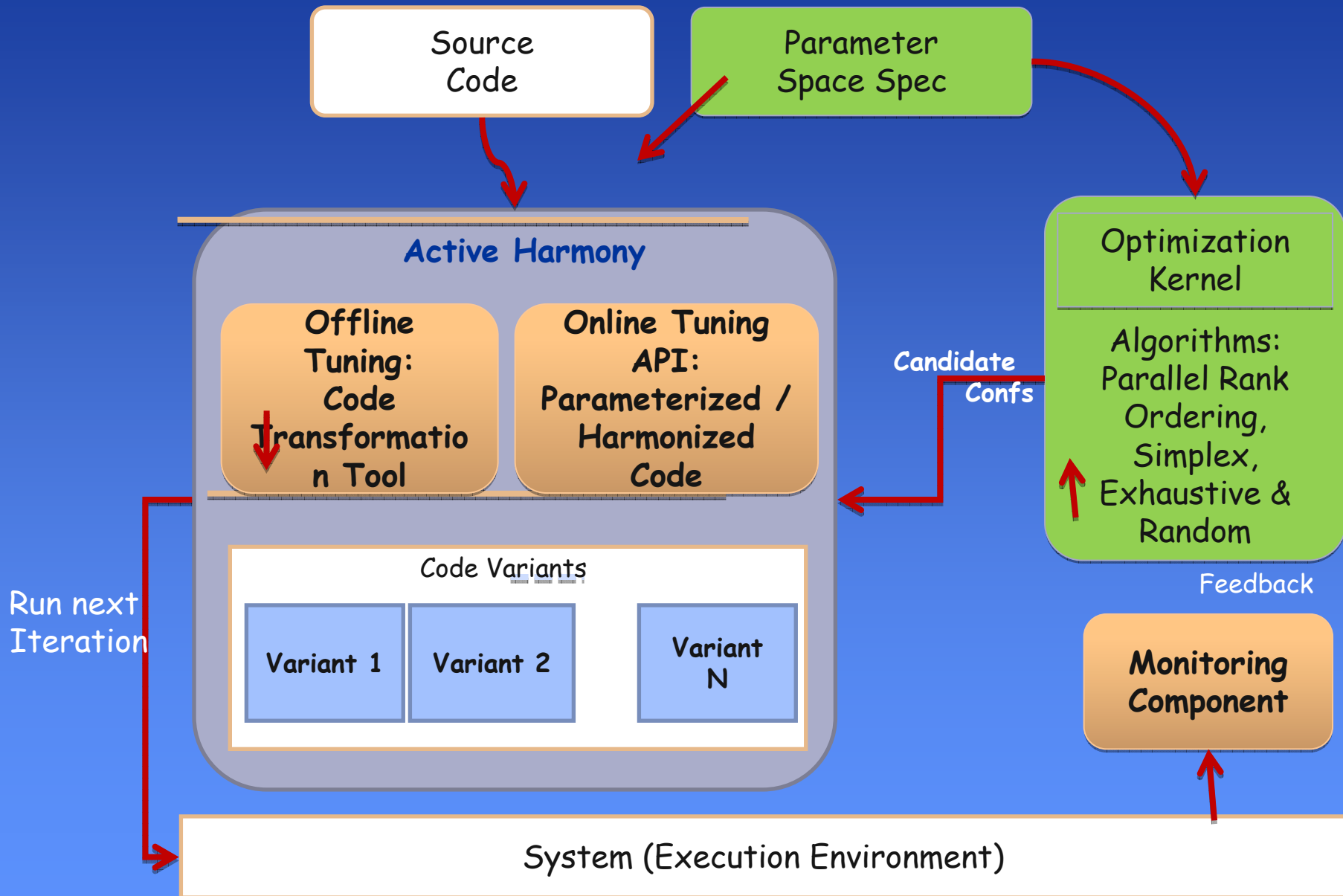
Background: Empirical Optimization

- Systematically search a collection of automatically generated alternative code variants and parameter values
- Empirically Tuned libraries and programs produce better code than native compilers

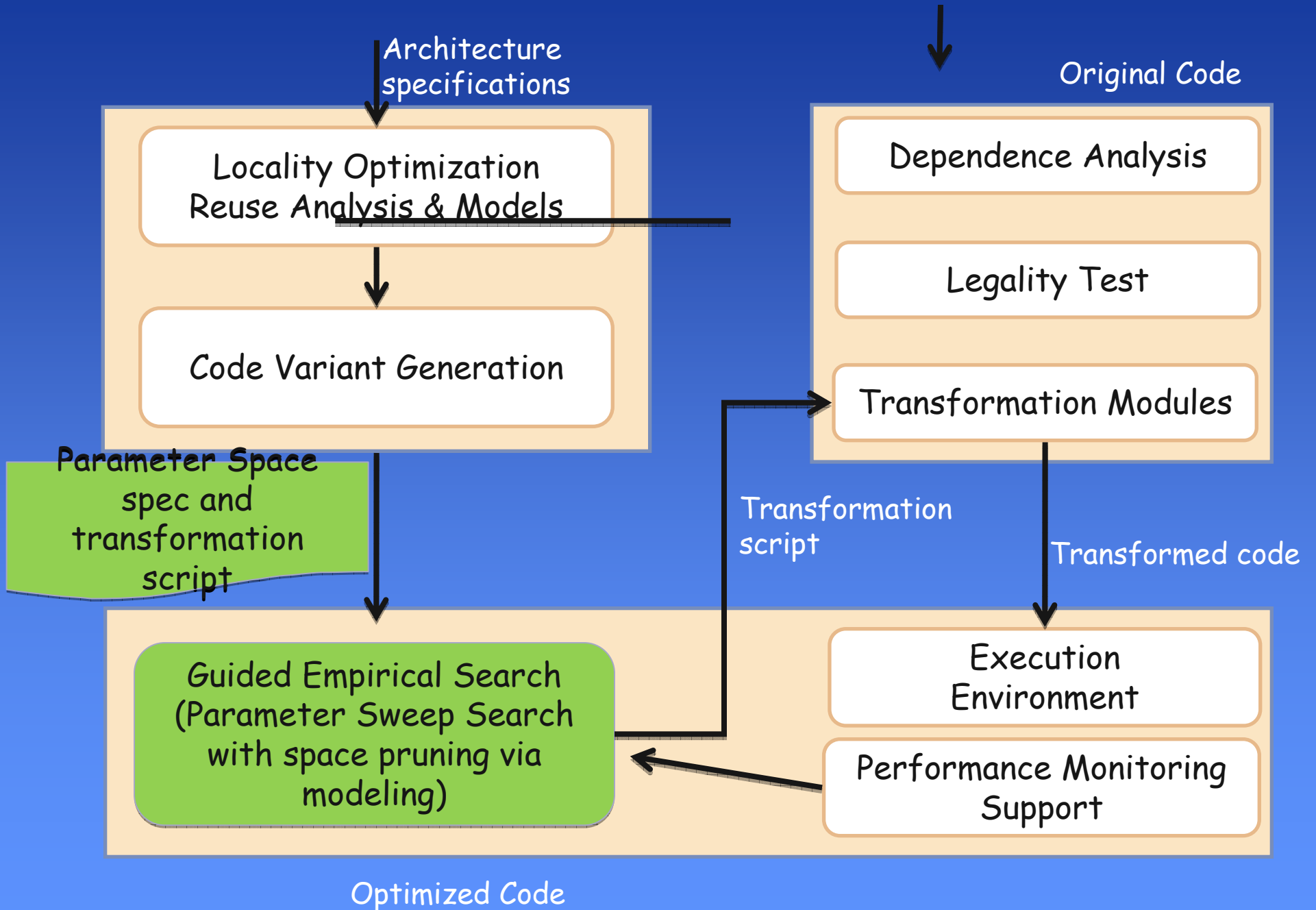
Constrained Optimization

- Empirical Optimization turns into a constrained optimization problem
 - Search space exponential and high dimensional
 - Bound Search Space with information from modeling
- Examples
 - Select tile sizes such that the tiles fit in the L1 cache of the target architecture
 - Select input parameters P and Q such that $P \geq Q$

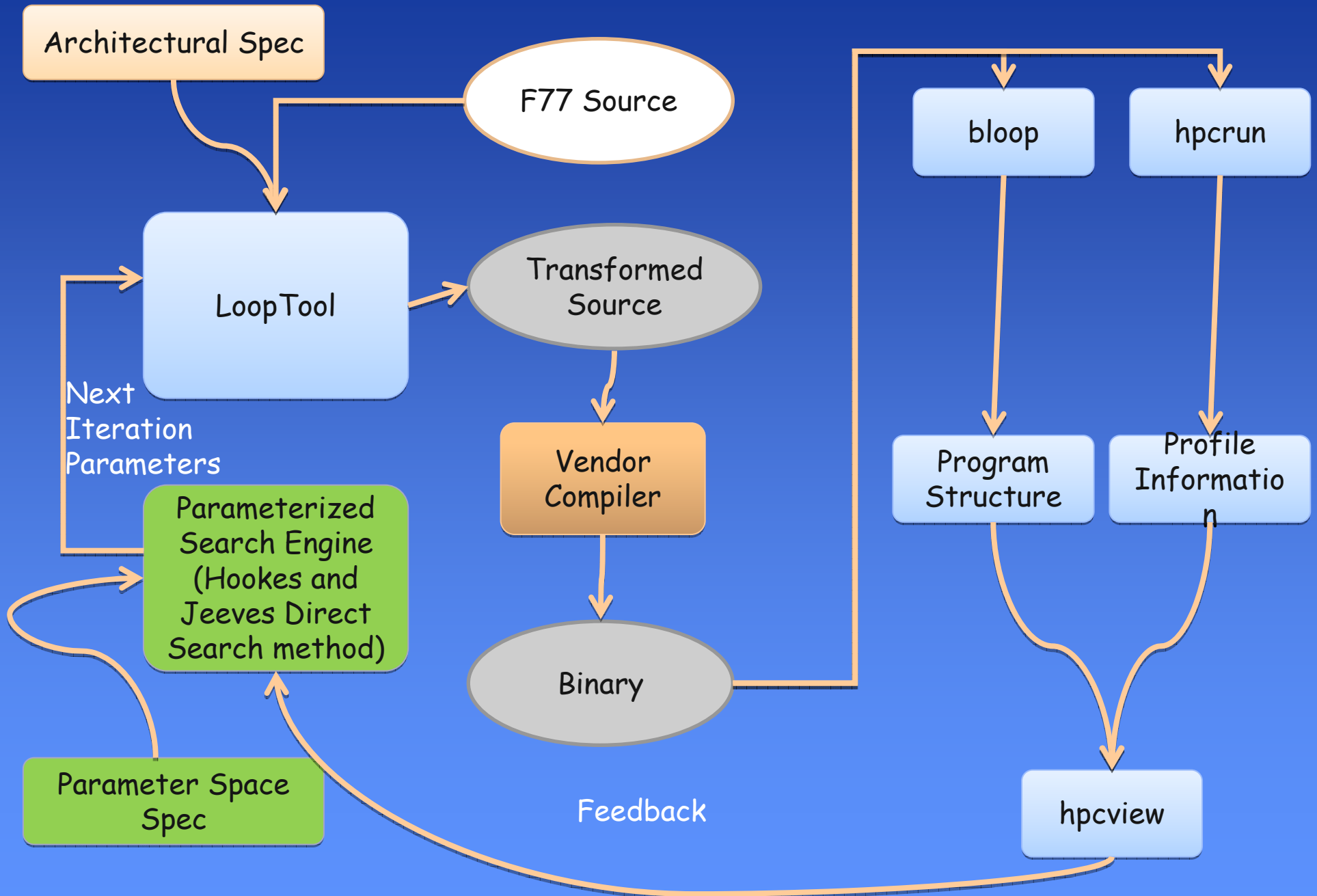
UMD Active Harmony Framework



USC Framework

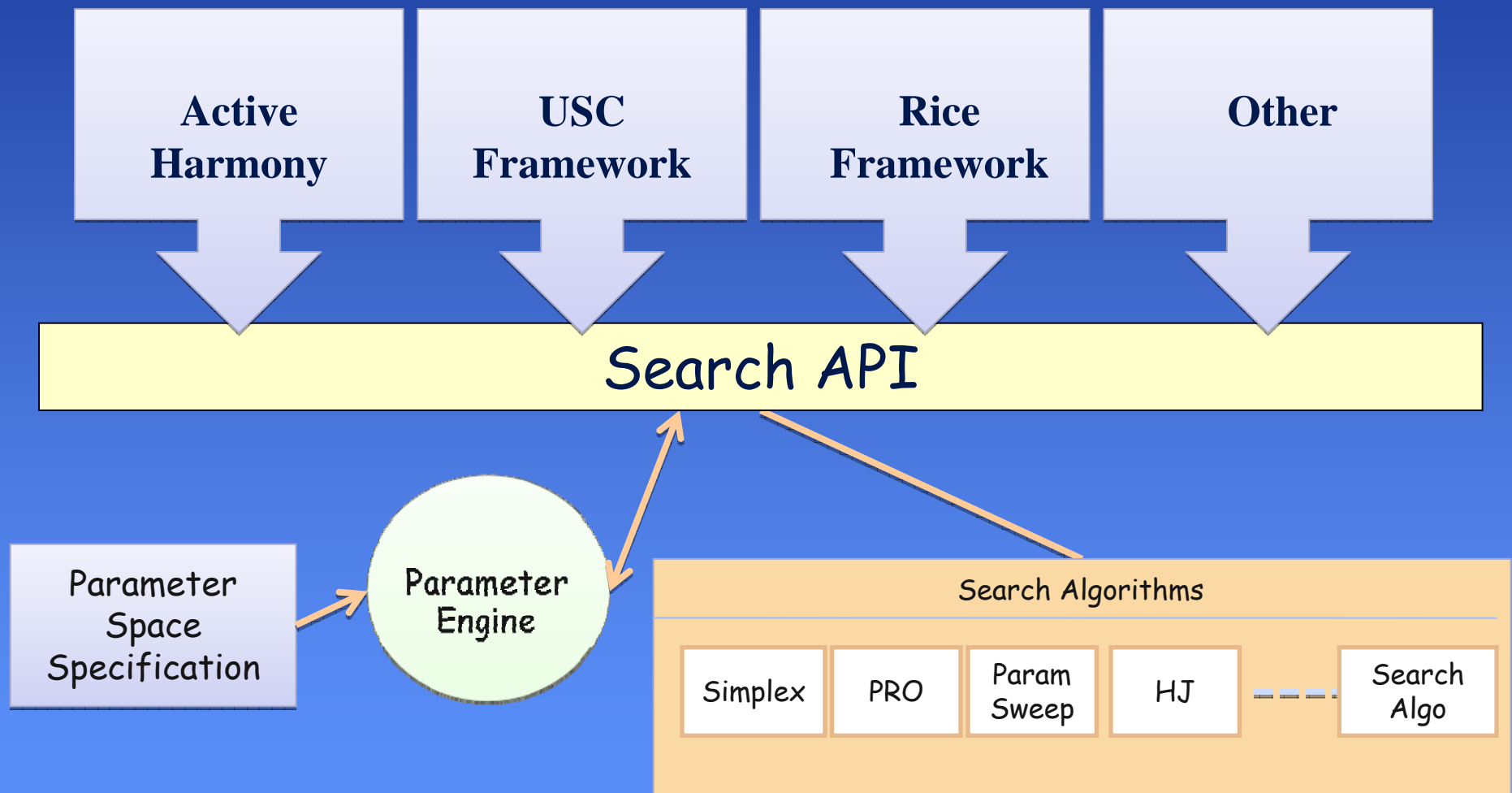


Rice Framework



Generalized

Frameworks



Parameter Specification Language - Requirements

- Define the search space:
 - Represent the search space symbolically
 - Specify parameter types (integer vs. float)
 - Represent parameter domain (range, step etc.)
- Represent constraints from:
 - tools
 - applications (via automated analysis)
 - programmers
- Provide support for arbitrary expression and function evaluation

Requirements ...

- Express search hints:
 - Ordering/ranking parameters (*unroll before tiling*)
 - Group parameters, code regions and/or constraints into sets
 - Represent data from static modeling, historical runs
- Support for mapping language constructs
 - Identify where in the source code (e.g. what loop) the optimization is taking place
- Specify when and how to gather objective function value (compile-time vs. application runtime)

Draft Specification Language

- Six main components:
 - Code Region Declaration
 - Region Set Declaration
 - Parameter Declaration
 - Constraint Declaration
 - Constraint Specification
 - Ordering Info
- Provides a rich expression syntax

What we might specify? Ex. #1

```
parameter space simple_example
```

```
{  
  parameter x int {  
    range [1:1:3];  
    default 3;  
  }  
  
  parameter y int {  
    range [1:1:3];  
    default 2;  
  }  
  
  parameter z int {  
    range [1:1:3];  
    default 1;  
  }  
}
```

```
# And then the constraints.
```

```
constraint c1 {  
  x ≥ z;  
}  
  
constraint c2 {  
  y > z;  
}  
  
# Constraint specification.  
specification {  
  c1 AND c2;  
}  
  
# Ordering information is  
optional.  
}
```

What we might specify? Ex. #2

```
parameter space tiling {
  code_region loopI;
  code_region loopJ;
  region_set loop [loopI, loopJ];
  # declare tile_size parameter
  parameter tile_size int {
    range [2:2:256]
    default 32;
    region loop;
  }
```

```
# Arbitrary constraint
constraint c1 {
  (loopI.tile_size *
   loopJ.tile_size * 3 * 4) ≤
  2048;
}
```

```
# rectangular tiles better.
constraint c2 {
  loopI.tile_size > loopJ.tile_size;
}
```

```
constraint c3 {
  loopJ.tile_size > loopI.tile_size;
}
```

```
specification {
  (c1 AND c2) OR (c1 AND c3);
}
```

What we might specify? Ex. #3

```
parameter space pstswm {  
  ...  
  # FTopt determines what FFT algorithm to use.  
  parameter FTopt enum {  
    enumeration [distributed, single_transpose, double_transpose];  
    default distributed;  
  }  
  # LTopt determines which LT algorithm to use.  
  parameter LTopt enum {  
    enumeration {distributed, transpose_based};  
    default distributed;  
  }  
  constraint pq {  
    (p*q) == 16;  
  }  
  # When FTopt is 'double_transpose', LTopt has to be 'transpose_based'  
  constraint ftLT {  
    (FTopt.value=double_transpose) IMPLIES (LTopt.value=distributed);  
  }  
  specification {  
    pq AND ftLT;  
  }  
}
```

Language Syntax and Implementation

- Looking into GNU-MathProg modeling language
 - Can this language address all the requirements that we have discussed?
 - May need to add syntactic sugar on top
- Looking into Python Constraint Module
 - No support for "on-demand" derivation of search points

Search API

- Needed functionality
 - Evaluate point
 - Run code at a point in search space
 - Likely to be a-sync to allow parallel search
 - Store/Read values for point in search space
 - Will include point in space, value, context (data set/machine info)
 - Query Spec
 - Learn about parameters, constraints
 - May use existing Math Prog API
 - Query Search Strategy Info

Search API

● Related Questions

- Migrate ordering and grouping info to search API?
- How can we use historical data?
 - Incorporating information from peri-db
- Representation of the states
 - Types of iterators
 - “On Demand” evaluation needed to prevent space representation explosion

Conclusions

- First step towards integrating search-based auto-tuning frameworks
- Once a decision on parameter space language is made, search API will be rolled out

- Thank you!

- PERI Search Wiki:

- http://peri-scidac.org/wiki/index.php/Search_Working_Group

- Refs:

[1]: Cristian Tapus, I-Hsin Chung, Jeffrey K. Hollingsworth, "Active Harmony: Towards Automated Performance Tuning", Proceedings of SuperComputing, November 2002
[Active Harmony Project site: www.dyninst.org/harmony]

[2]: Chun Chen, Phd Dissertation, "Model Guided Empirical Optimization for Memory Hierarchy", May 2007.

[3]: Apan Qasem, Phd Dissertation, "Automatic Tuning for Scientific Applications", July 2007