Compiler-Based Autotuning Technology

Lecture 1: Autotuning and Its Origins

Mary Hall

July, 2011

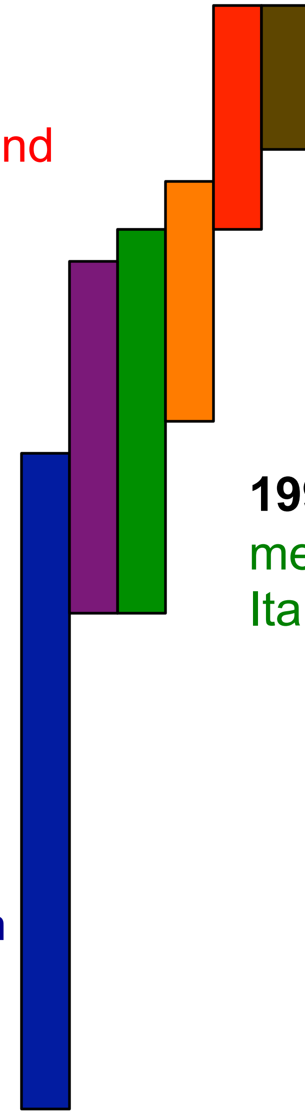
* This work has been partially sponsored by DOE SciDAC as part of the Performance Engineering Research Institute (PERI), DOE Office of Science, the National Science Foundation, DARPA and Intel Corporation.

THE UNIVERSITY OF UTAH

Instructor: My Research Timeline

2005-present: Auto-tuning compiler technology (memory hierarchy, multimedia extensions, multi-cores and GPUs)

2007-present: Reports on compiler, exascale software and archiving research directions



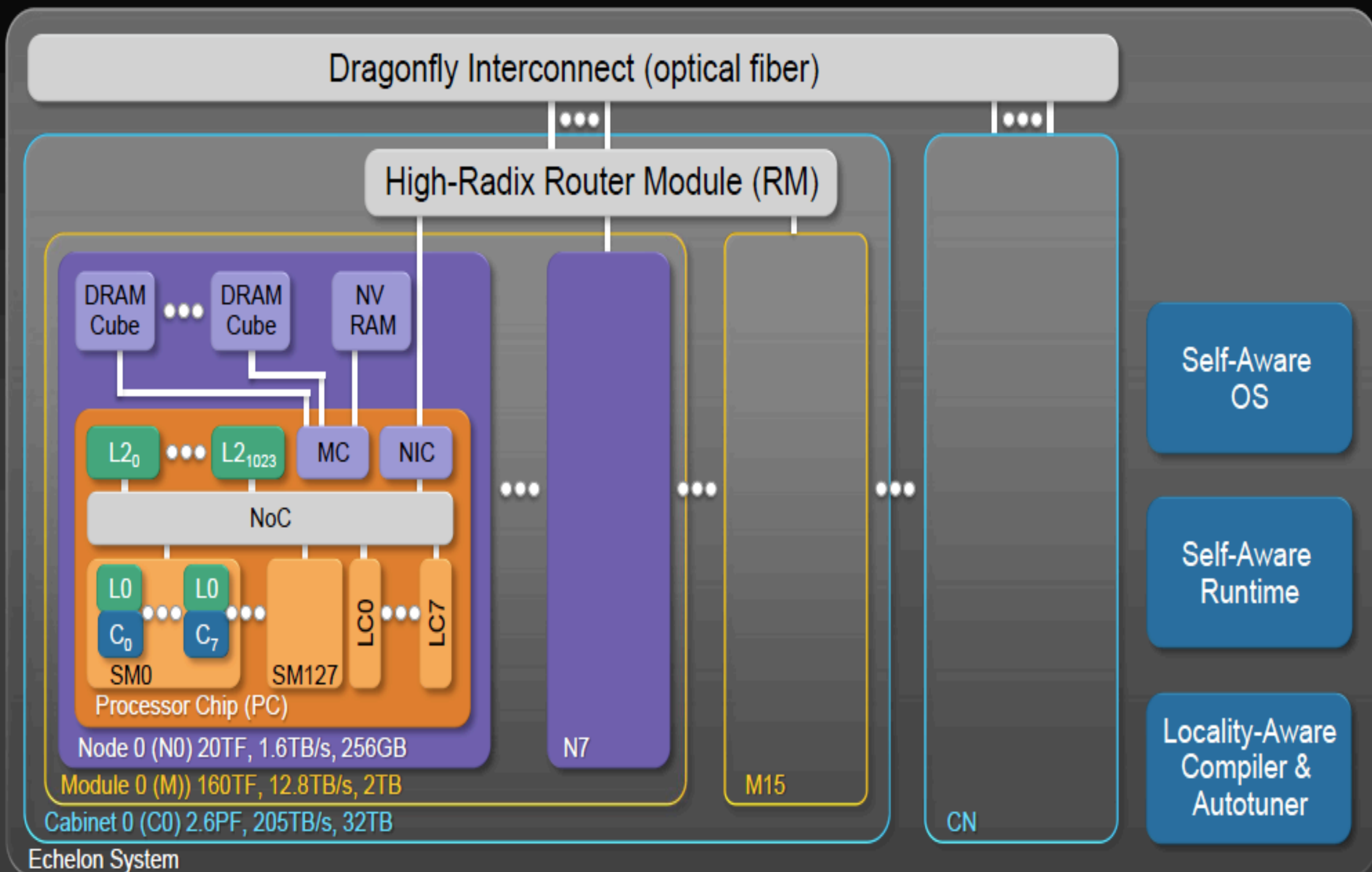
2001-2006: Compilation for multimedia extensions (DIVA, AltiVec and SSE)

1998-2004: DEFACTO design environment for FPGAs (C to VHDL)

1998-2005: DIVA Processing-in-memory system architecture (HP Itanium-2 architecture)

1986-2000: Interprocedural Optimization and Automatic Parallelization, Rice D System and Stanford SUIF Compiler

Echelon System Sketch
from "GPU Computing To Exascale and Beyond", Bill Dally, SC10



(DOE SciDAC) PERI Autotuning Tools

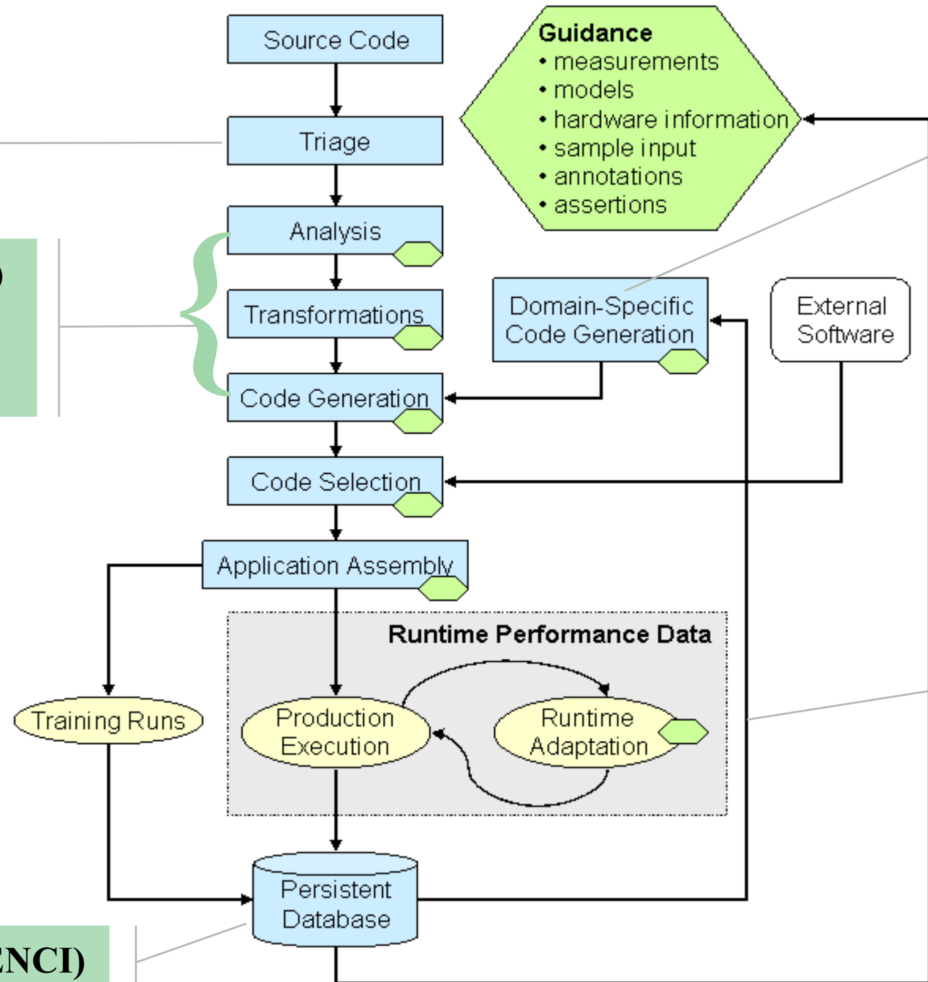
HPC Toolkit (Rice)
ROSE (LLNL)

CHiLL (USC/ISI and Utah)
ROSE (LLNL)
Orio (Argonne)

OSKI (LBNL)

Active Harmony (UMD)
GCO (UTK)

PerfTrack (LBNL, SDSC, RENCI)



Motivation: A Looming Software Crisis

- Architectures are getting increasingly complex
  - Multiple cores, deep memory hierarchies, software-controlled storage, shared resources, SIMD compute engines, heterogeneity, ...
- Performance optimization is getting more important
  - Today's sequential and parallel applications ***may not*** be faster on tomorrow's architectures.
  - Especially if you want to add new capability!
  - Managing ***data locality*** even more important than parallelism.
  - Managing ***power*** of growing importance, too.

Complexity!

Motivation: What is Autotuning?

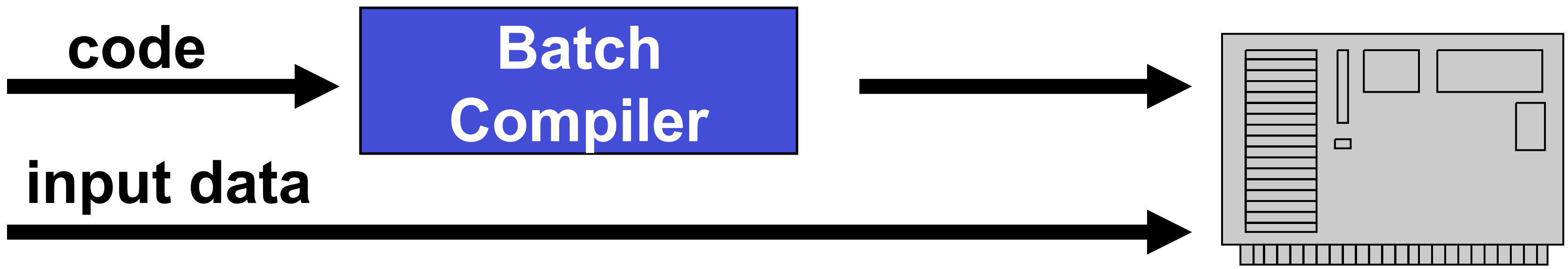
- Definition:
  - Automatically generate a "search space" of possible implementations of a computation
    - A ***code variant*** represents a unique implementation of a computation, among many
    - A ***parameter*** represents a discrete set of values that govern code generation or execution of a variant
  - Measure execution time and compare
  - Select the best-performing implementation
- Key Issues:
  - Identifying the search space
  - Pruning the search space to manage costs
  - Off-line vs. on-line search

Motivation: My Philosophy

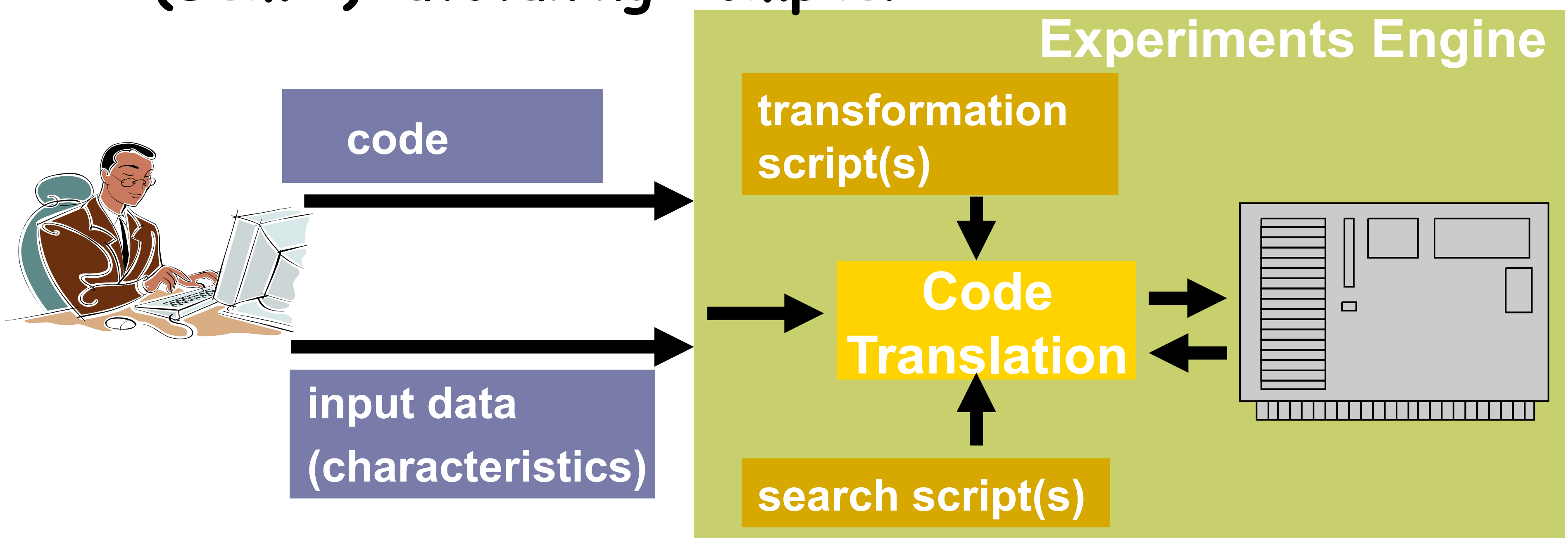
- Identify search space through a high-level description that captures a large space of possible implementations
- Prune space through compiler domain knowledge and architecture features
- Provide access to programmers! (controversial)
- Uses source-to-source transformation for portability, and to leverage vendor code generation
- Requires ***restructuring of the compiler***

Motivation: Collaborative Autotuning "Compiler"

Traditional view:



(Semi-)Autotuning Compiler:



THE UNIVERSITY OF UTAH

Outline of Course

L1: **Autotuning and its Origins (today!)**

L2: **Tuning code with CHiLL**

L3: **A Closer Look at Polyhedral Compiler Frameworks**

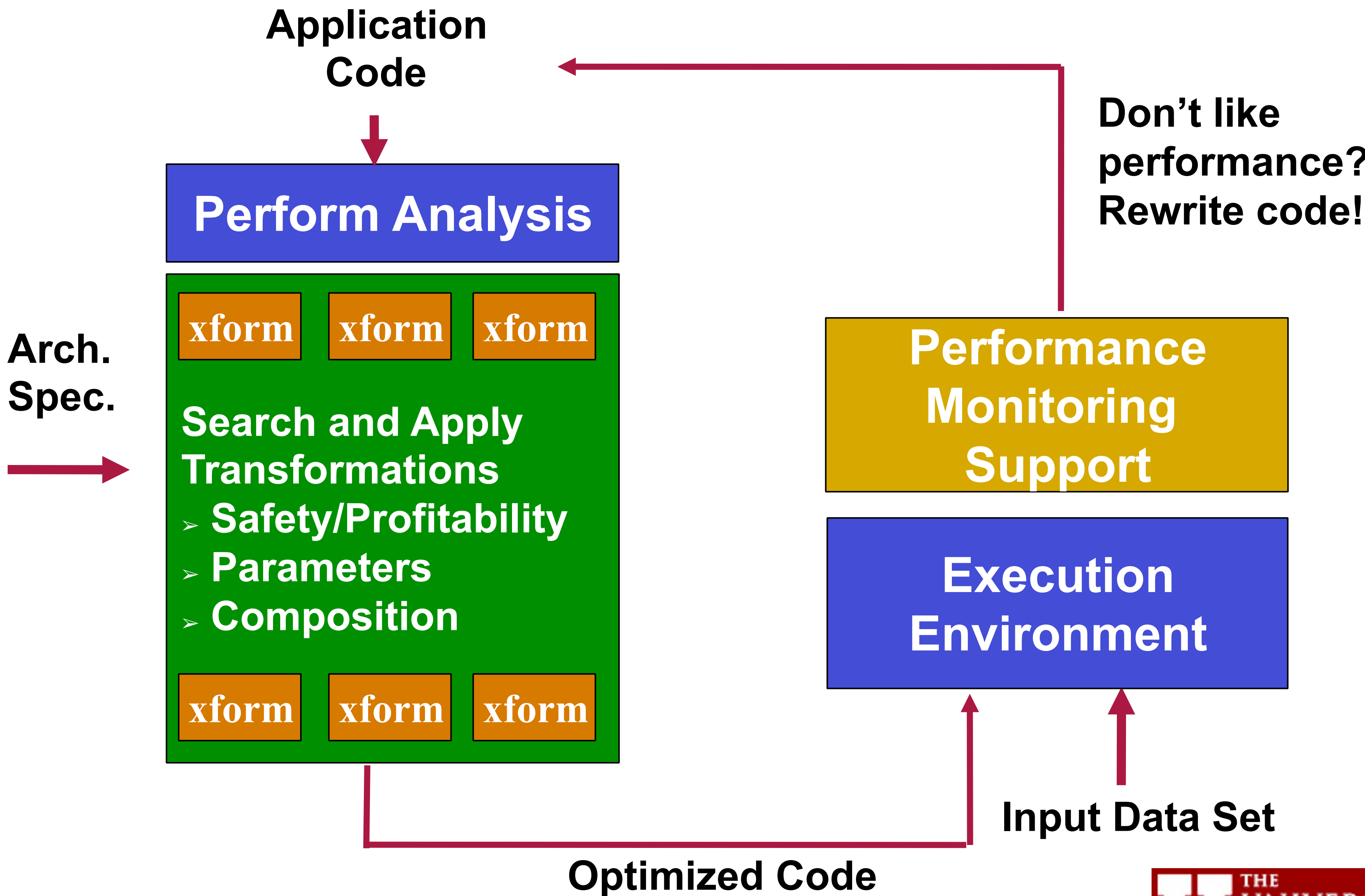
L4: **Autotuning for GPU Code Generation**

L5: **Autotuning High-End Applications**

Today's Lecture: Autotuning and its Origins

1. Traditional Compiler Organization
2. Origins in hardware optimization
3. Related Compiler Organization
   - Use of learning algorithms in compiler
4. Autotuning systems
   - Library-specific autotuning
   - Application-specific autotuning
   - Compiler-based autotuning
5. Detailed look at ATLAS, OSKI, SPIRAL, Active Harmony, PetaBricks and Sequoia

1. Historical Organization of Compilers



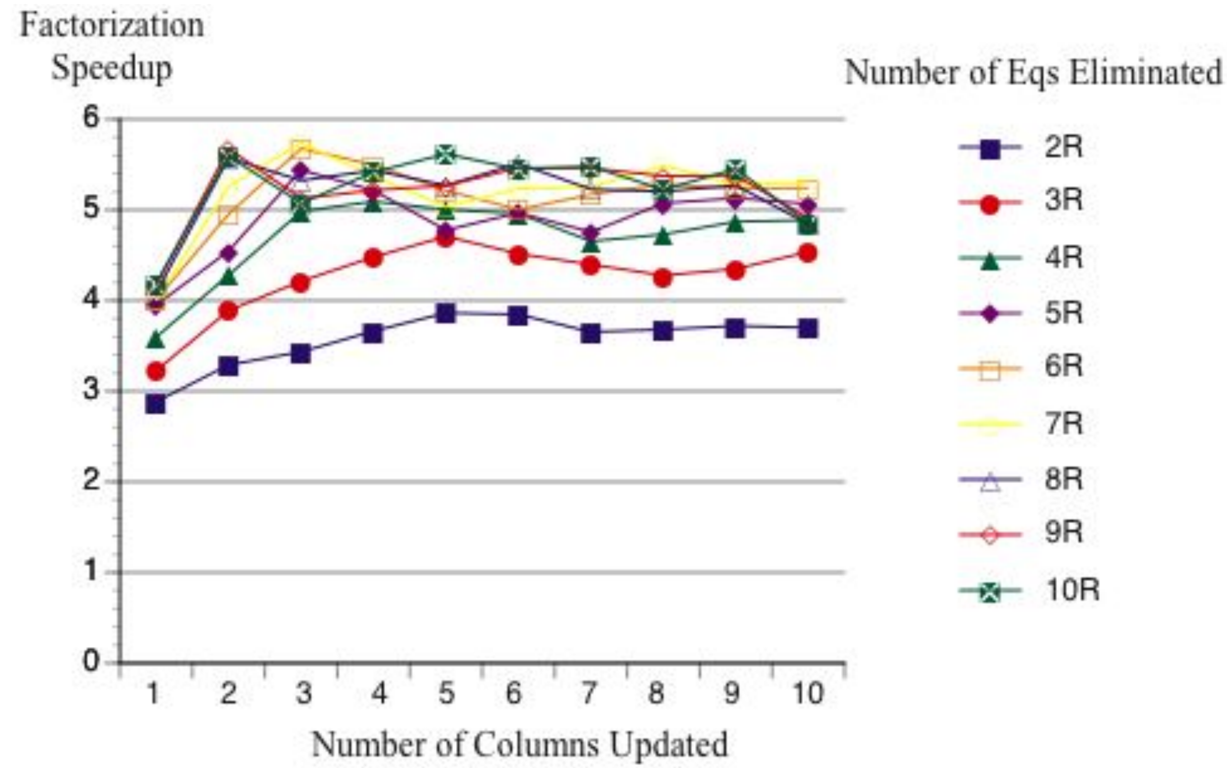
1. Historical Organization of Compilers

- What's not working
  - Transformations and optimizations often applied in isolation, but significant interactions
  - Static compilers must anticipate all possible execution environments
  - Potential to slow code down; many users say "never use O3"
  - Users write low-level code to get around compiler which makes things even worse

1. Example of Programmer-Guided Transformations

LS-DYNA Solver Performance Results

- Application programmer has written code variants for every possible unroll factor of two innermost loops
- Straightforward for compiler to generate this code and test for best version



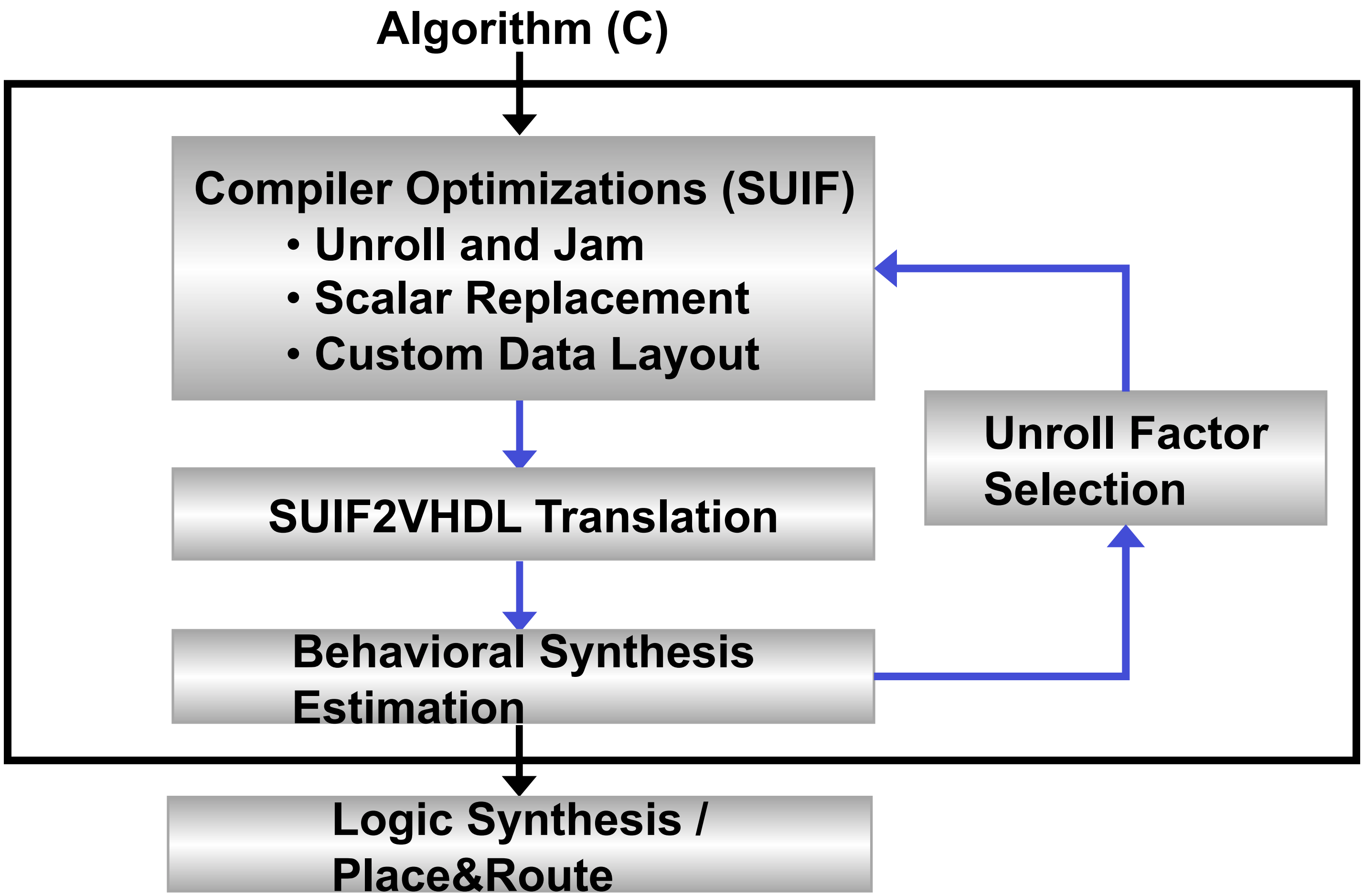
THE UNIVERSITY OF UTAH

2. Related Approach in Hardware Design

- Autotuning is related to hardware (and hardware-software) design space exploration
  - The process of analyzing various functionally equivalent implementations to identify the one that best meets objectives.
- Early example:
  - Vinoo Srinivasan et al., "Hardware Software Partitioning with Integrated Hardware Design Space Exploration," Design, Automation and Test in Europe Conference and Exhibition, p. 28, Design Automation and Test in Europe (DATE '98), 1998

THE UNIVERSITY OF UTAH

2. Automatic Design Space Exploration in DEFACTO

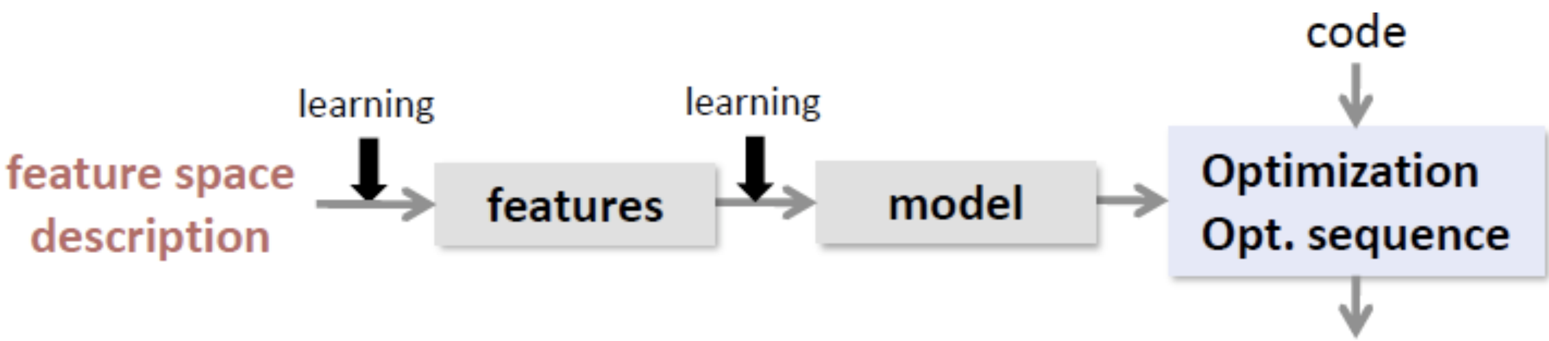


- Overall, less than 2 hours
- 5 minutes for optimized design selection

THE UNIVERSITY OF UTAH

3. Related Compiler Organization: Iterative Compilation with Learning

- A preceding body of work on using learning techniques (and sometimes profiling) to make optimization decisions
  - Cooper et al., Eigenmann et al., Stephenson et al, Cavazos et al., …
- Examples from
  - Instruction scheduling, optimization flag selection, optimization sequence, unroll factor selection, …



4. Three Types of Autotuning Systems

a. Autotuning libraries
   - Library that encapsulates knowledge of library's performance under different execution environments
   - Dense linear algebra: **ATLAS, PhiPAC**
   - Sparse linear algebra: **OSKI**
   - Signal processing: **SPIRAL, FFTW**

b. Application-specific autotuning
   - **Active Harmony** provides parallel rank order search for tunable parameters and variants
   - **Sequoia** and **PetaBricks** provide language mechanism for expressing tunable parameters and variants

c. Compiler-based autotuning
   - Focus of this course

4a. Motivation for Autotuning Libraries

- Many codes spend the bulk of their computation time performing very common operations
  - Particularly linear algebra and signal processing
- Enhance performance without requiring low-level programming of the application
- Much research has been devoted to achieving high performance
  - Search space reasonably well understood
  - Performance can still be improved using autotuning

3a. ATLAS (BLAS)

- Self-tuning linear algebra library
- Early description in SIAM 2000
- ATLAS first popularized notion of self-tuning libraries
- Clint Whaley quote: “No such thing as enough compute speed for many scientific codes”
- Precursor: PhiPAC, 1997

3a. ATLAS (BLAS)

ATLAS Method of Software Adaptation

① Parameterization:

- Parameters provide different implementations (e.g., tile size)
- Easy to implement but limited

② Multiple Implementations:

- Linear search of routine list (variants)
- Simple to implement, simple for external contribution
- Low adaptability, ISA independent, kernel dependent

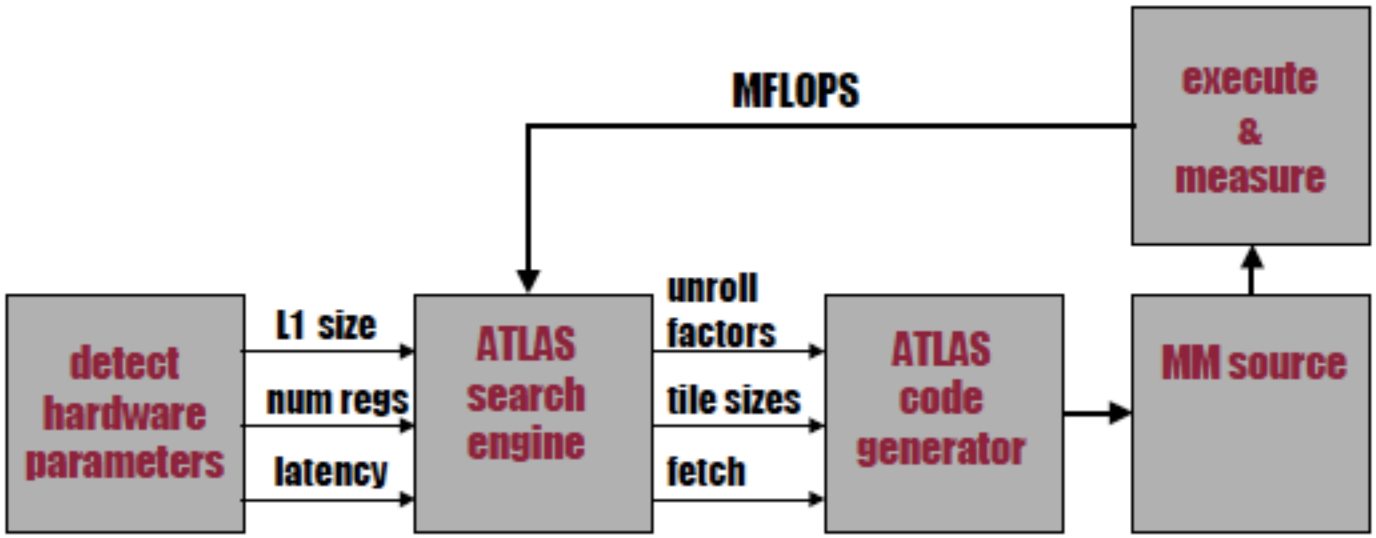
③ Source Generator:

- Heavily parameterized program generates varying implementations
- Very complicated to program, search and contribute
- High adaptability, ISA independent, kernel dependent

Slide source: Clint Whaley

3a. Structure of ATLAS Source Generator

GEMM as building block for other Level 3 BLAS functions



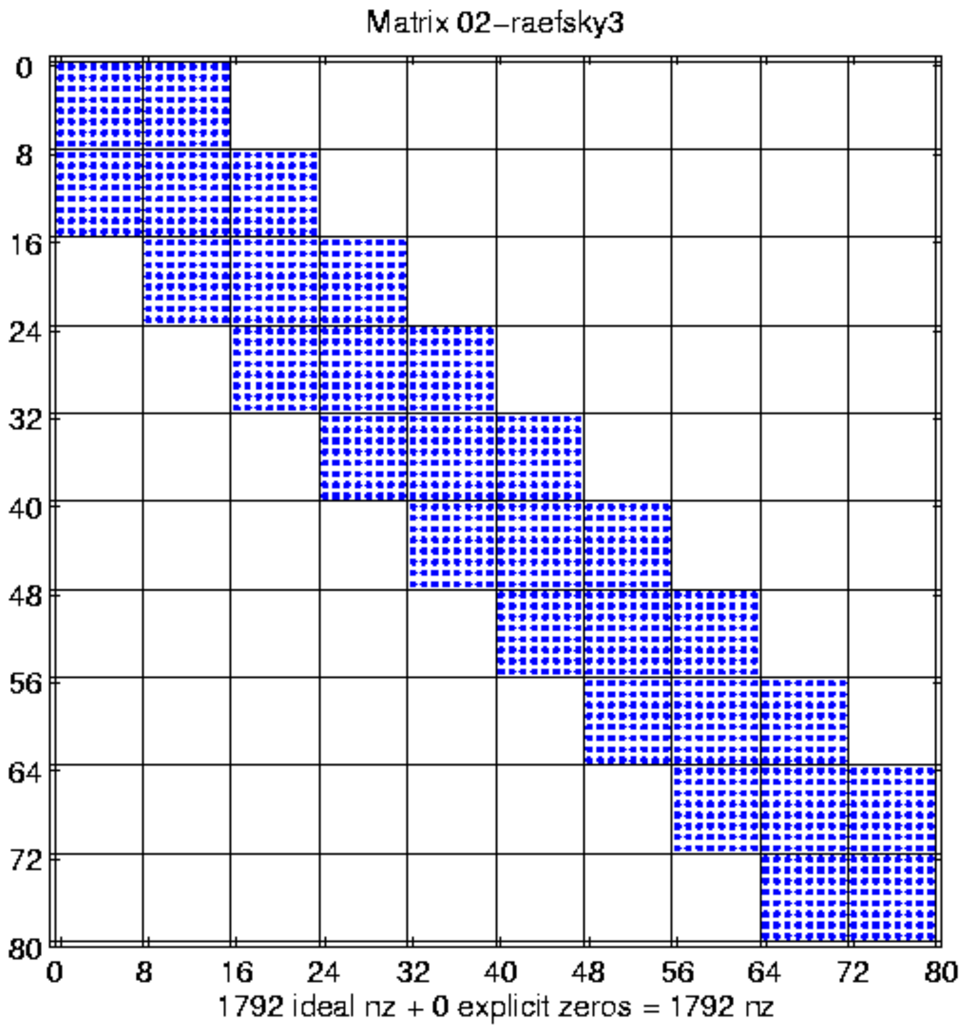
Slide source: Jacqueline Chame

3a. OSKI (Sparse BLAS)

- Sparse matrix-vector multiply < **10% peak, decreasing**
  - Indirect, irregular memory access
  - Low computational intensity *vs.* dense linear algebra
  - Depends on matrix (**run-time**) and machine
- Tuning is becoming more important
  - **2× speedup from tuning, will increase**
- Unique challenge of sparse linear algebra
  - Matrix structure dramatically affects performance
  - To the extent possible, exploiting structure leads to better performance

Slide source: Rich Vuduc

3a. Example of Matrix Structure in OSKI

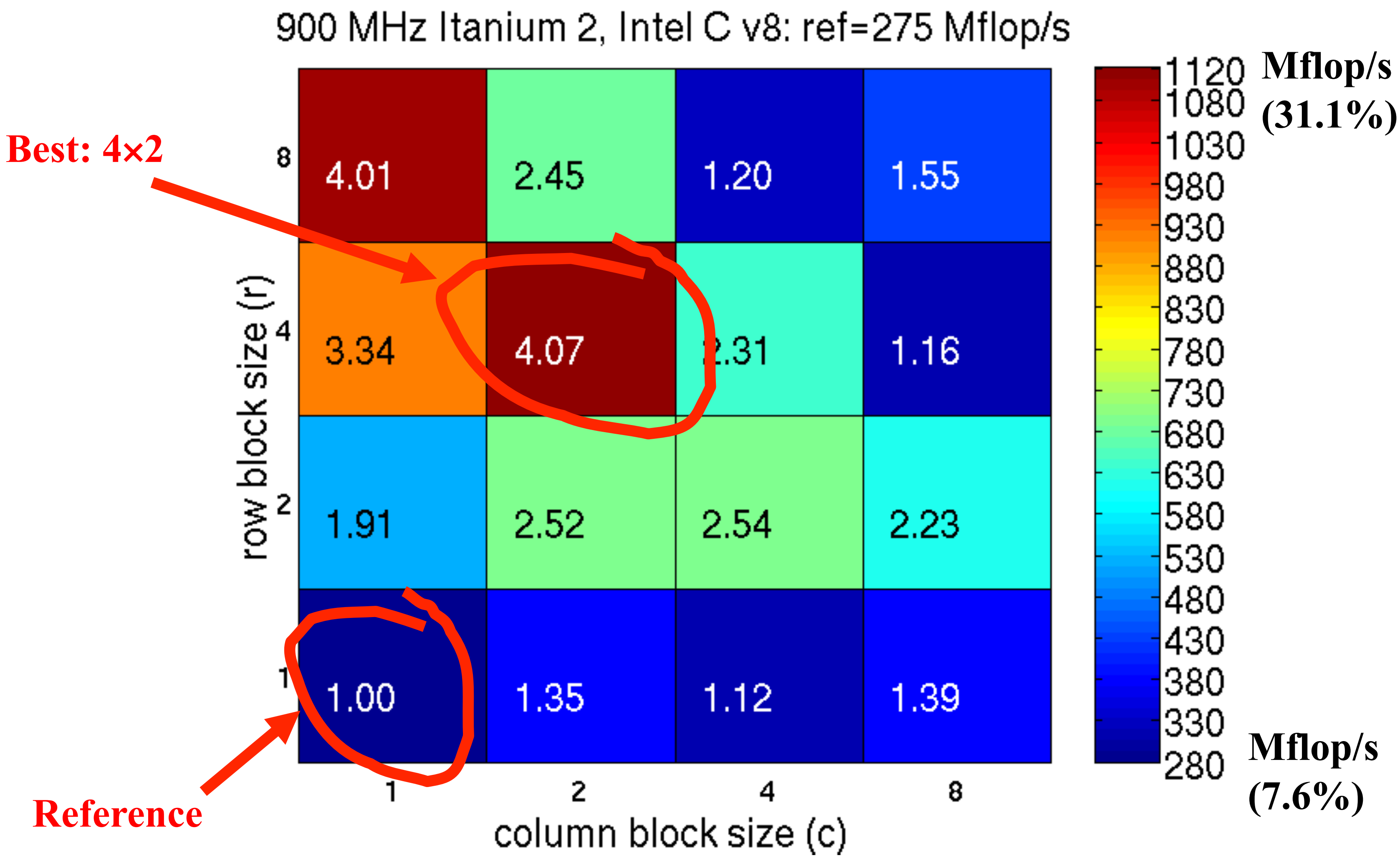


- **Exploit 8×8 blocks**
  - **Store blocks & unroll**
  - **Compresses data**
  - **Regularizes accesses**
- **As r×c ↑, speed ↑**

Slide source: Rich Vuduc

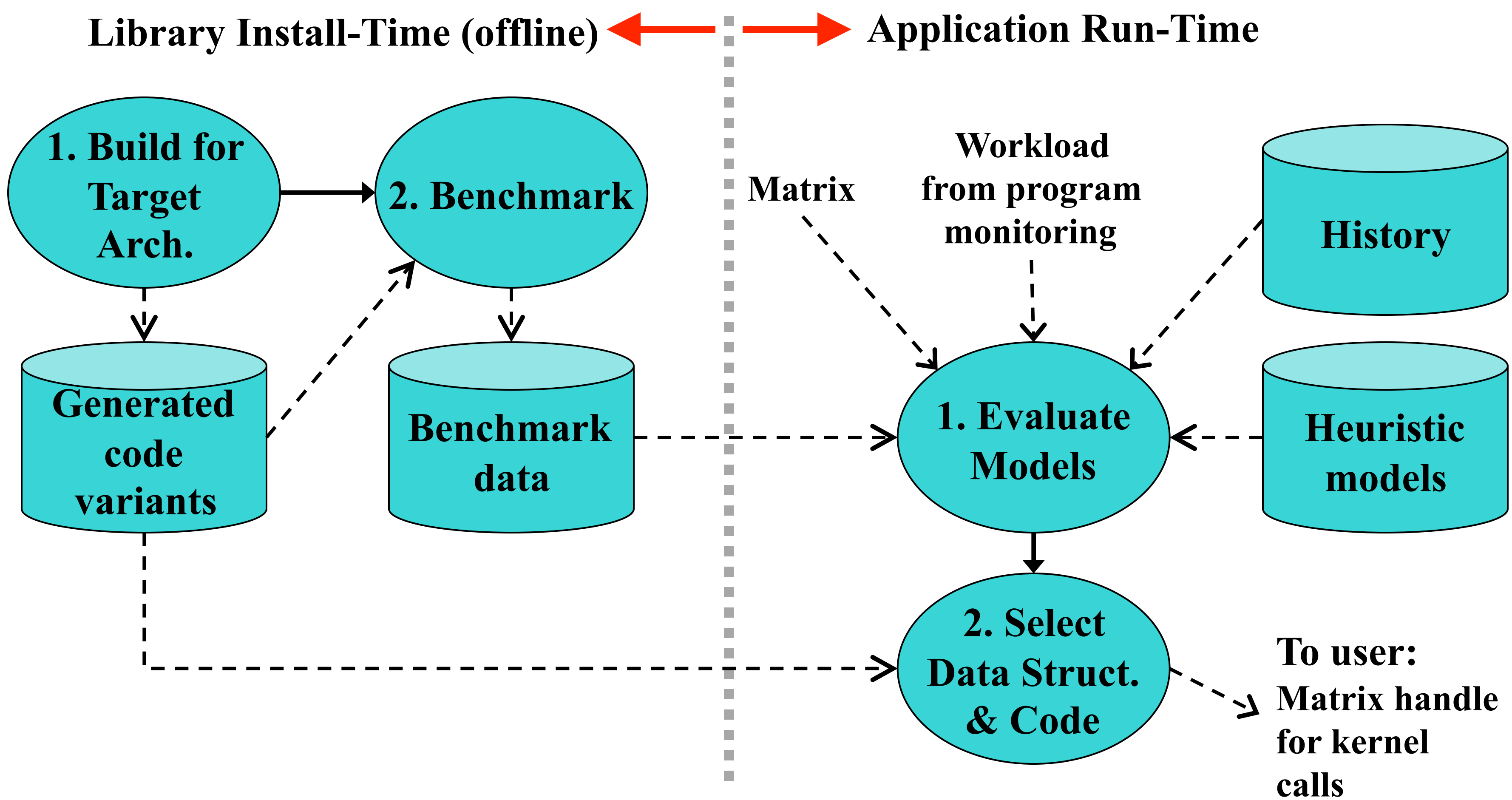
THE UNIVERSITY OF UTAH

3a. Example of Matrix Structure in OSKI: Speedups on Itanium 2 for different block sizes



Slide source: Rich Vuduc

3a. Structure of OSKI



Slide source: Rich Vuduc

3a. SPIRAL (Signal Processing)

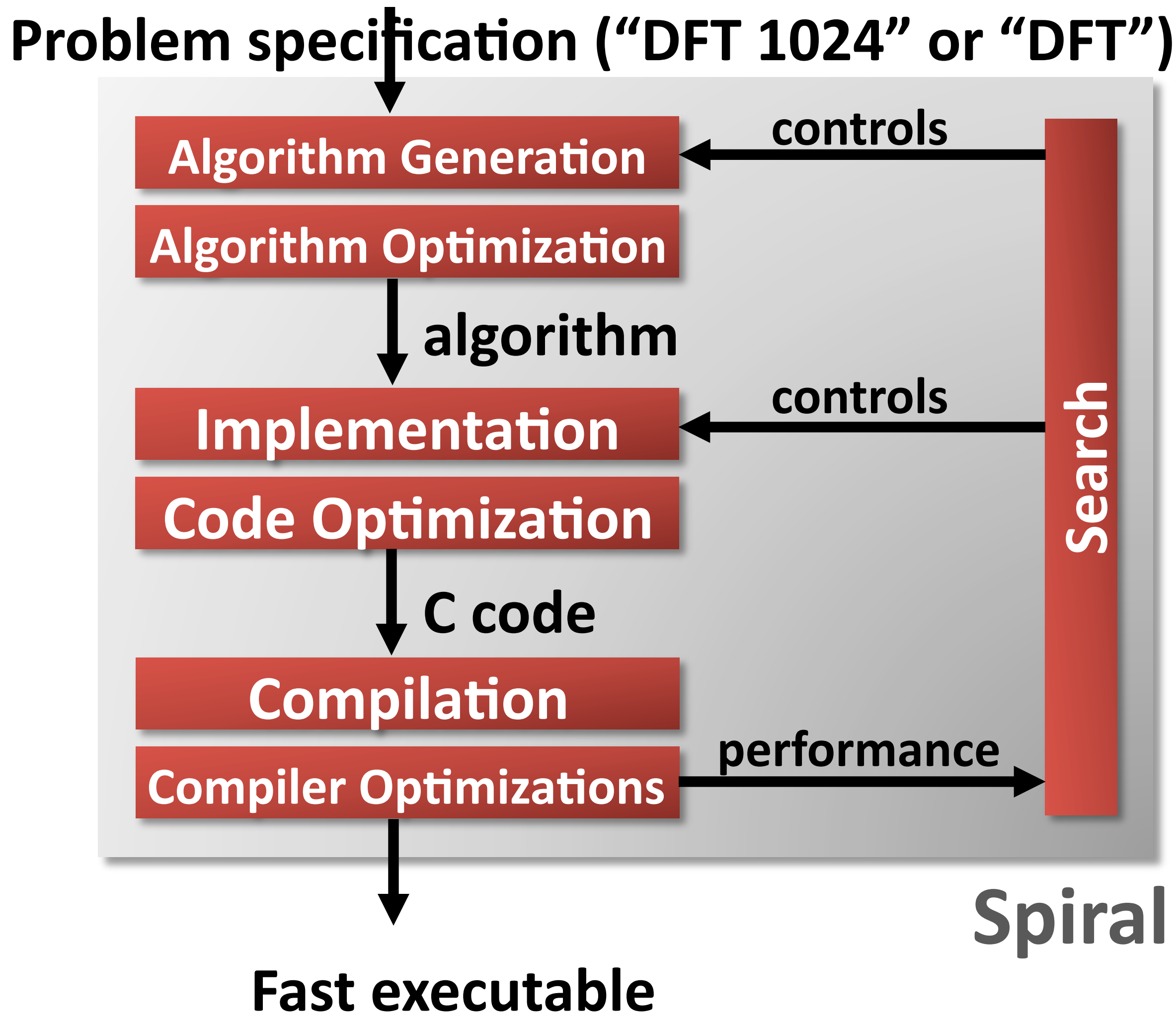
Complete automation of the implementation and optimization task

Basic ideas:

•***Declarative representation*** of algorithms

•***Rewriting systems*** to generate and optimize algorithms at a high level of abstraction

• *Similar concepts in FFTW*



Slide source: Franz Franchetti

4a. SPIRAL: Rules in Domain-Specific Language

Linear Transforms

$$\mathbf{DFT}_n \rightarrow (\mathbf{DFT}_k \otimes \mathrm{I}_m)\,\mathrm{T}^n_m(\mathrm{I}_k \otimes \mathbf{DFT}_m)\,\mathrm{L}^n_k, \quad n = km$$

$$\mathbf{DFT}_n \rightarrow P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \quad n = km,\ \gcd(k,m) = 1$$

$$\mathbf{DFT}_p \rightarrow R_p^T(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \quad p \text{ prime}$$

$$\mathbf{DCT\text{-}3}_n \rightarrow (\mathrm{I}_m \oplus \mathrm{J}_m)\,\mathrm{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \cdot(\mathrm{F}_2 \otimes \mathrm{I}_m)\begin{bmatrix}\mathrm{I}_m & 0 \oplus -\mathrm{J}_{m-1} \\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m)\end{bmatrix}, \quad n = 2m$$

$$\mathbf{DCT\text{-}4}_n \rightarrow S_n \mathbf{DCT\text{-}2}_n\, \mathrm{diag}_{0\le k<n}(1/(2\cos((2k+1)\pi/4n)))$$

$$\mathbf{IMDCT}_{2m} \rightarrow (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m)\left(\left(\begin{bmatrix}1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix}-1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right)\right)\mathrm{J}_{2m}\,\mathbf{DCT\text{-}4}_{2m}$$

$$\mathbf{WHT}_{2^k} \rightarrow \prod_{i=1}^{t}(\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t$$

$$\mathbf{DFT}_2 \rightarrow \mathrm{F}_2$$

$$\mathbf{DCT\text{-}2}_2 \rightarrow \mathrm{diag}(1, 1/\sqrt{2})\,\mathrm{F}_2$$

$$\mathbf{DCT\text{-}4}_2 \rightarrow \mathrm{J}_2\,\mathrm{R}_{13\pi/8}$$

Viterbi Decoding

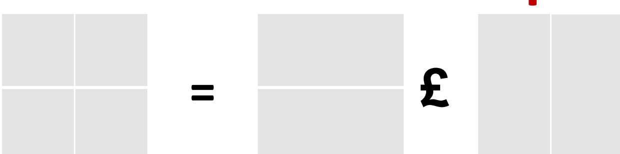
$$\underset{\mathrm{vec}(v)}{\mathrm{Vit}} \rightarrow \underbrace{\left(\prod(L \times I) \circ (I \otimes C)\right)}_{\mathrm{vec}(v)} \circ Id$$

$$\rightarrow \left(\prod \underbrace{(L \times I) \circ (I \otimes C)}_{\mathrm{vec}(v)}\right) \circ Id$$

$$\rightarrow \left(\prod(L \otimes I_v \times I) \circ (I \otimes C \otimes I_v) \circ (\vec{L} \times I)\right) \circ Id$$

$$\rightarrow \prod(L \otimes I_v \times I) \circ (I \otimes (B \otimes I_v)) \circ (\vec{L} \times I)$$

Matrix-Matrix Multiplication



$$\mathrm{MMM}_{1,1,1} \rightarrow (\cdot)_1$$

$$\mathrm{MMM}_{m,n,k} \rightarrow (\otimes)_{m/m_b \times 1} \otimes \mathrm{MMM}_{m_b,n,k}$$

$$\mathrm{MMM}_{m,n,k} \rightarrow \mathrm{MMM}_{m,nb,k} \otimes (\otimes)_{1\times n/nb}$$

$$\mathrm{MMM}_{m,n,k} \rightarrow ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathrm{MMM}_{m,n,k_b}) \circ ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn})$$

$$\mathrm{MMM}_{m,n,k} \rightarrow (L^{mn/n_b}_m \otimes I_{n_b}) \circ ((\otimes)_{1\times n/n_b} \otimes \mathrm{MMM}_{m,n_b,k}) \circ (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))$$

Synthetic Aperture Radar (SAR)

$$\mathrm{SAR}_{k\times m \rightarrow n\times n} \rightarrow \mathrm{DFT}_{n\times n} \circ \mathrm{Interp}_{k\times m\rightarrow n\times n}$$

$$\mathrm{DFT}_{n\times n} \rightarrow (\mathrm{DFT}_n \otimes \mathrm{I}_n) \circ (\mathrm{I}_n \otimes \mathrm{DFT}_n)$$

$$\mathrm{Interp}_{k\times m\rightarrow n\times n} \rightarrow (\mathrm{Interp}_{k\rightarrow n} \otimes_i \mathrm{I}_n) \circ (\mathrm{I}_k \otimes_i \mathrm{Interp}_{m\rightarrow n})$$

$$\mathrm{Interp}_{r\rightarrow s} \rightarrow \left(\bigoplus_{i=0}^{n-2} \mathrm{InterpSeg}_k\right) \oplus \mathrm{InterpSegPruned}_{k,\ell}$$

$$\mathrm{InterpSeg}_k \rightarrow \mathrm{G}_f^{u\cdot n \rightarrow k} \circ \mathrm{iPrunedDFT}_{n\rightarrow u\cdot n} \circ \left(\frac{1}{n}\right) \circ \mathrm{DFT}_n$$

Slide source: Franz Franchetti

3b. Motivation for Application-level tuning

- Parameters and variants arise naturally in portable application code
- Programmer expresses tunable parameters, input data set properties and algorithm variants
- Tools automatically generate code and evaluate tradeoff space of application-level parameters

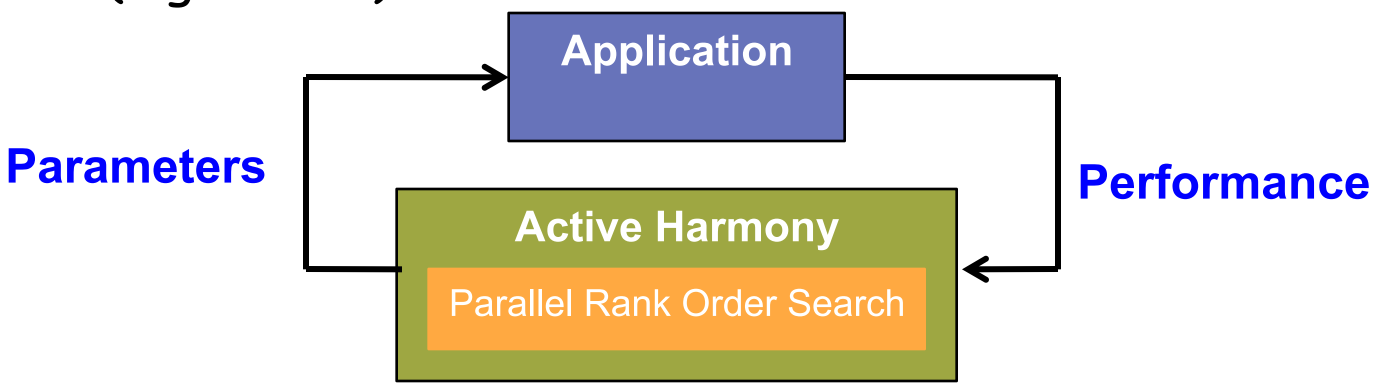
Example: Molecular Dynamics Visualization

```
Parameter cellSize, range = 48:144, step 16

ncell = boxLength/cellSize

for i = 1, ncell
   /* perform computation */
```

```
Const cellSize = 48

ncell = boxLength/48

for i = 1, 48
   /* perform computation */
```

3b. Application-level tuning using Active Harmony

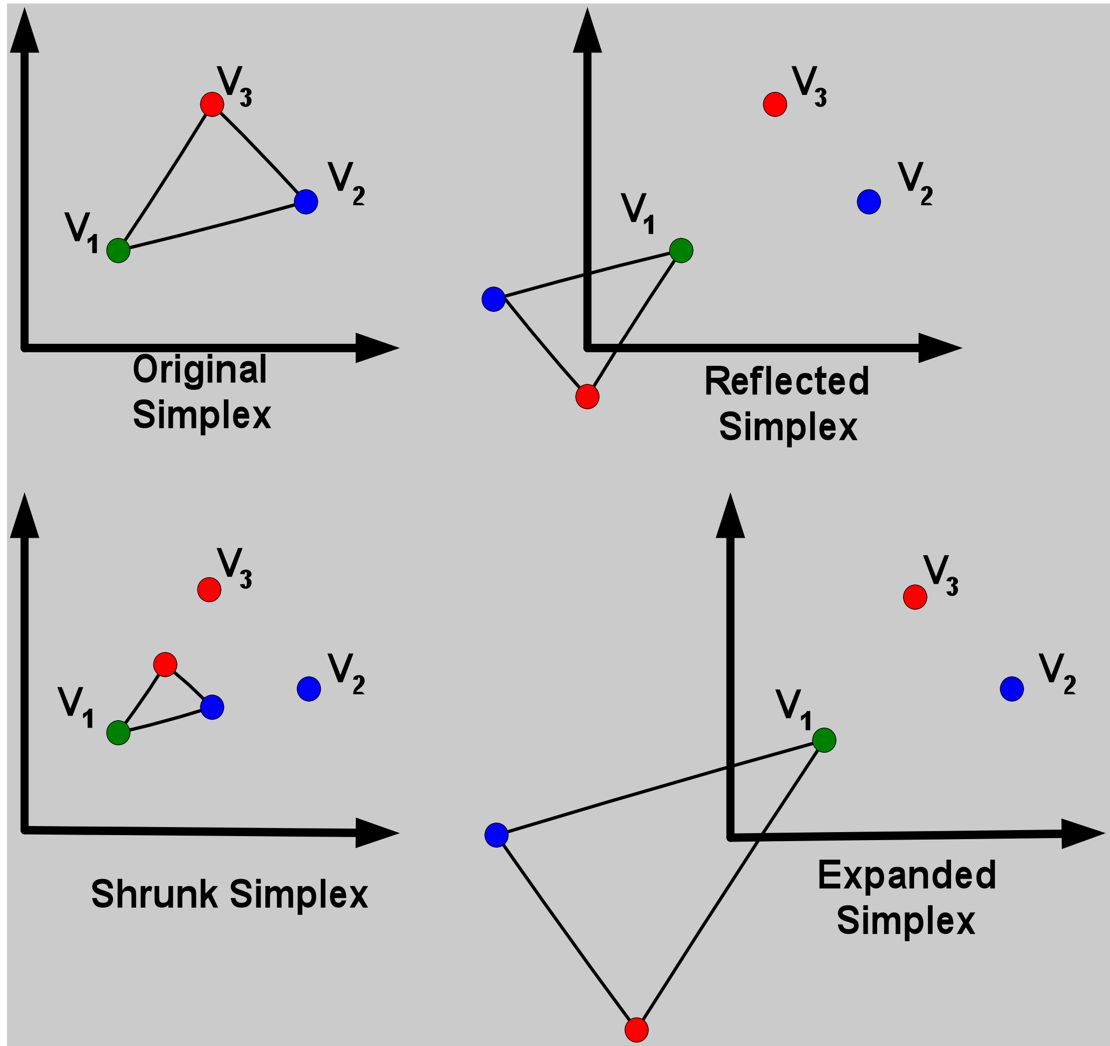
- Search-based collaborative approach
  - Simultaneously explore different tunable parameters to search a large space defined by the user
    - e.g., Loop blocking and unrolling factors, number of OpenMP threads, data distribution algorithms, granularity controls, …
  - Supports both online and offline tuning
  - Central controller monitors performance, adjusts parameters using search algorithms, repeats until converges
  - Can also generate code on-demand for tunable parameters that need new code (e.g. unroll factors) using code transformation frameworks (e.g. CHiLL)



Slide source: Ananta Tiwari

3b. Active Harmony Parallel Rank Order Algorithm

- All, but the best point of simplex moves
- Computations can be done in parallel
- N parallel evaluations for N +1 point simplex



Slide source: Ananta Tiwari

4b. Language support for application-level tuning using PetaBricks

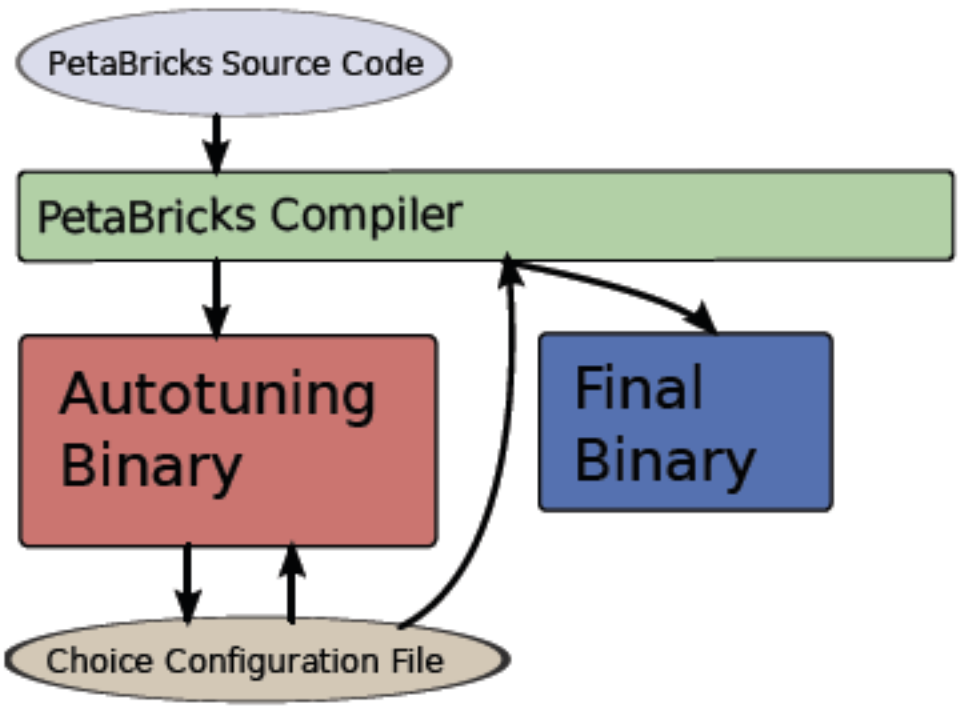
- Algorithmic choice in the language is the key aspect of PetaBricks
- Programmer can define multiple rules to compute the same data
- Compiler re-uses rules to create hybrid algorithms
- Can express choices at many different granularities

Example: Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```


Slide source: Saman Amarasinghe

4b. Language support for application-level tuning using PetaBricks



① PetaBricks source code is compiled

② An autotuning binary is created

③ Autotuning occurs creating a choice configuration file

④ Choices are fed back into the compiler to create a final binary

Slide source: Saman Amarasinghe

4b. Application-level tuning is similar using Sequoia

- Example shows variants representing hierarchical implementation of matrix multiply
- These two tasks represent different variants for different levels of the memory system
- Tunable parameters P, Q and R adjust data decomposition

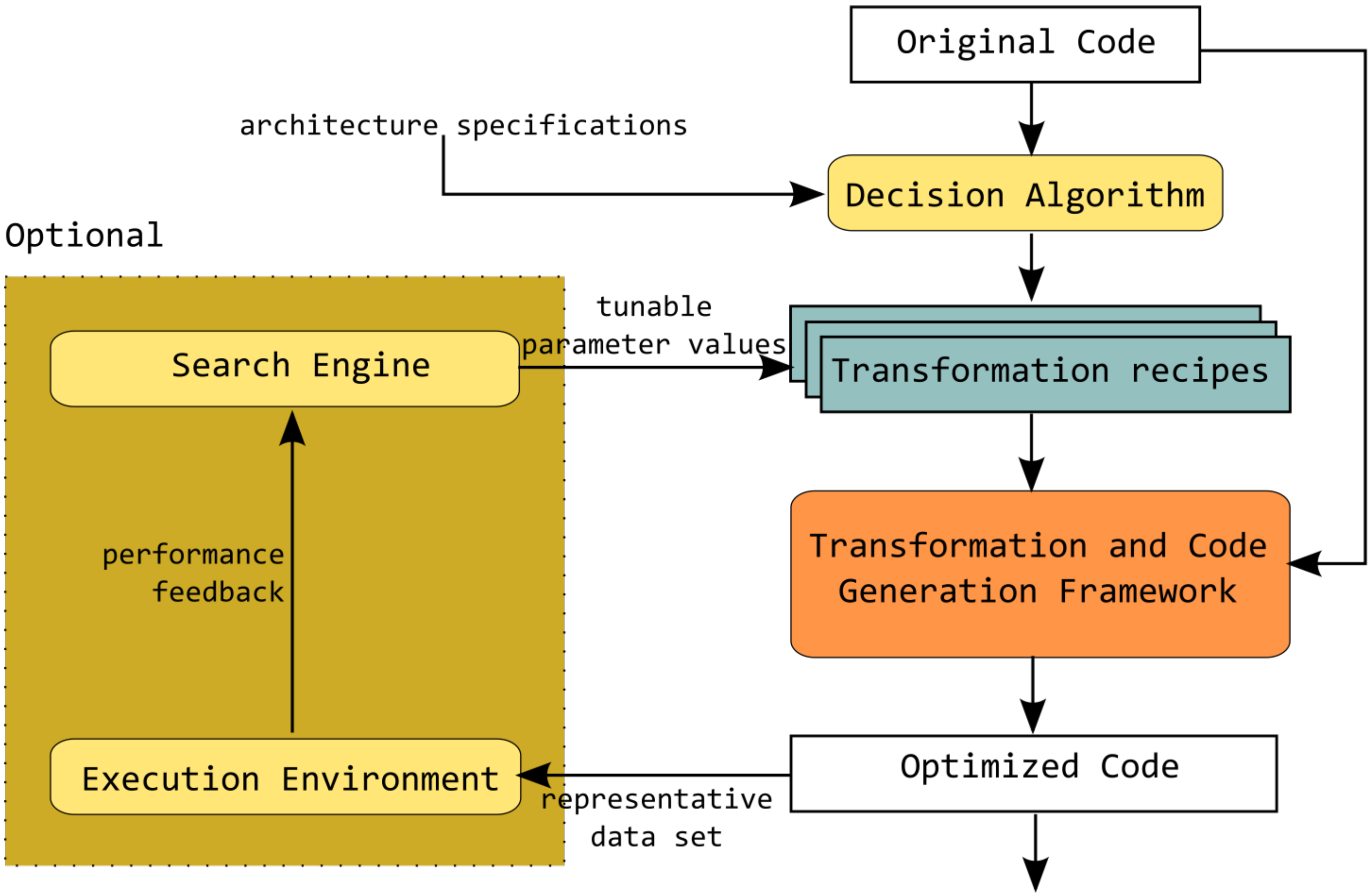
```
task matmul::inner(in    float A[M][T],
                   in    float B[T][N],
                   inout float C[M][N])
{
  tunable int P, Q, R;

  mappar( int i=0 to M/P,
          int j=0 to N/R) {
     mapseq( int k=0 to T/Q ) {

        matmul(A[P*i:P*(i+1);P][Q*k:Q*(k+1);Q],
               B[Q*k:Q*(k+1);Q][R*j:R*(j+1);R],
               C[P*i:P*(i+1);P][R*j:R*(j+1);R]);

     }
  }
}

task matmul::leaf(in    float A[M][T],
                  in    float B[T][N],
                  inout float C[M][N])
{
   for (int i=0; i<M; i++)
     for (int j=0; j<N; j++)
       for (int k=0;k<T; k++)
          C[i][j] += A[i][k] * B[k][j];
}
```

Example from Mike Houston, CScaDS 2007

4c. Motivation for Compiler-Based Autotuning Framework

- Parameters and variants arise from compiler optimizations
  - Parameters such as tile size, unroll factor, prefetch distance
  - Variants such as different data organization or data placement, different loop order or other representation of computation
- Beyond libraries
  - Can specialize to application context (libraries used in unusual ways)
  - Can apply to more general code
- Complementary and easily composed with application-level support

4c. CHiLL Compiler-Based Autotuning Framework



4c. Combining Models, Heuristics and Empirical Search

Compiler Models (static)

- How much data reuse?
- Data footprint in memory hierarchy levels
- Profitability estimates of optimizations

Heuristics

- "Place" data in specific memory hierarchy level based on reuse
- Copy data tiles mapped to caches or buffers

Empirical Search

- Generate parameterized code variants
- Measure performance to evaluate and choose next point to search
- Heuristics limit variants
- Constraints from models limit parameter values

Summary of Lecture

- Sampling of autotuning systems
  - Autotuning libraries
  - Application-level autotuning
  - Compiler-based autotuning
- "Search space" of implementations arises from
  - Parameters
  - Variants
- Lecture mostly focused on structure of systems and expressing/generating search space

THE UNIVERSITY OF UTAH

References

ATLAS: J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc and R. C. Whaley, "Self Adapting Linear Algebra Algorithms and Software", Proceedings of the IEEE, Volume 93, Number 2, pp. 293-312, February, 2005.

OSKI: R. Vuduc, J. Demmel, and K. Yelick. "OSKI: A library of automatically tuned sparse matrix kernels". Proceedings of SciDAC 2005, Journal of Physics: Conference Series, June 2005.

SPIRAL: M. Püschel, J. M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, R. W. Johnson and N. Rizzolo. "SPIRAL: Code Generation for DSP Transforms". Proceedings of the IEEE, 93(2):232-275, 2005.

FFTW: M. Frigo. 1999. A fast Fourier transform compiler. In Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation (PLDI '99).

Active Harmony: A. Tiwari, J. K. Hollingsworth, "End-to-end Auto-tuning with Active Harmony". In Performance Tuning of Scientific Applications, D. Bailey, R.F. Lucas and S. Williams, ed., Chapman & Hall/CRC Computational Science Series, 2010.

Sequoia: K. Fatahalian, T. Knight, M. Houston, M. Erez, D. Horn, L. Leem, H. Park, M. Ren, A. Aiken, W. Dally and P. Hanrahan, "Sequoia: Programming the Memory Hierarchy". In Proceedings of Supercomputing 2006, Nov. 2006.

PetaBricks: J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. 2009. "PetaBricks: a language and compiler for algorithmic choice". In Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation (PLDI '09).

THE UNIVERSITY OF UTAH