X-TUNE
Autotuning for Exascale: Self-Tuning Software to Manage Heterogeneity

Mary Hall, PI, University of Utah

Paul Hovland, Stefan Wild, Krishna Narayanan, Jeff Hammond (ANL)

Lenny Oliker, Sam Williams, Brian van Straalen (LBNL)

Jacqueline Chame (USC/ISI)

Website:
http://ctop.cs.utah.edu/x-tune/

Exascale Software: A View in 2012

Thanks to exascale reports and workshops

- Multiresolution programming systems for different users
  - Joe/Stephanie/Doug **[Pingali, UT]**
  - Elvis/Mort/Einstein **[Intel]**
- Specialization simplifies and improves efficiency
  - Target specific user needs with domain-specific languages/libraries
  - Customize libraries for application needs and execution context
- Interface to programmers and runtime/hardware
  - Seamless integration of compiler with programmer guidance and dynamic feedback from runtime
- Toolkits rather than monolithic systems
  - Layers support different user capability, collaborative ecosystem
- Virtualization (over-decomposition)
  - Hierarchical, or flat but construct hierarchy when applicable?

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 (for exascale, tradeoff between performance/energy/reliability)
- Key Issues:
  - Identifying the search space
  - Pruning the search space to manage costs
  - Off-line vs. on-line search

Three Types of Autotuning Systems

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

X-TUNE

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

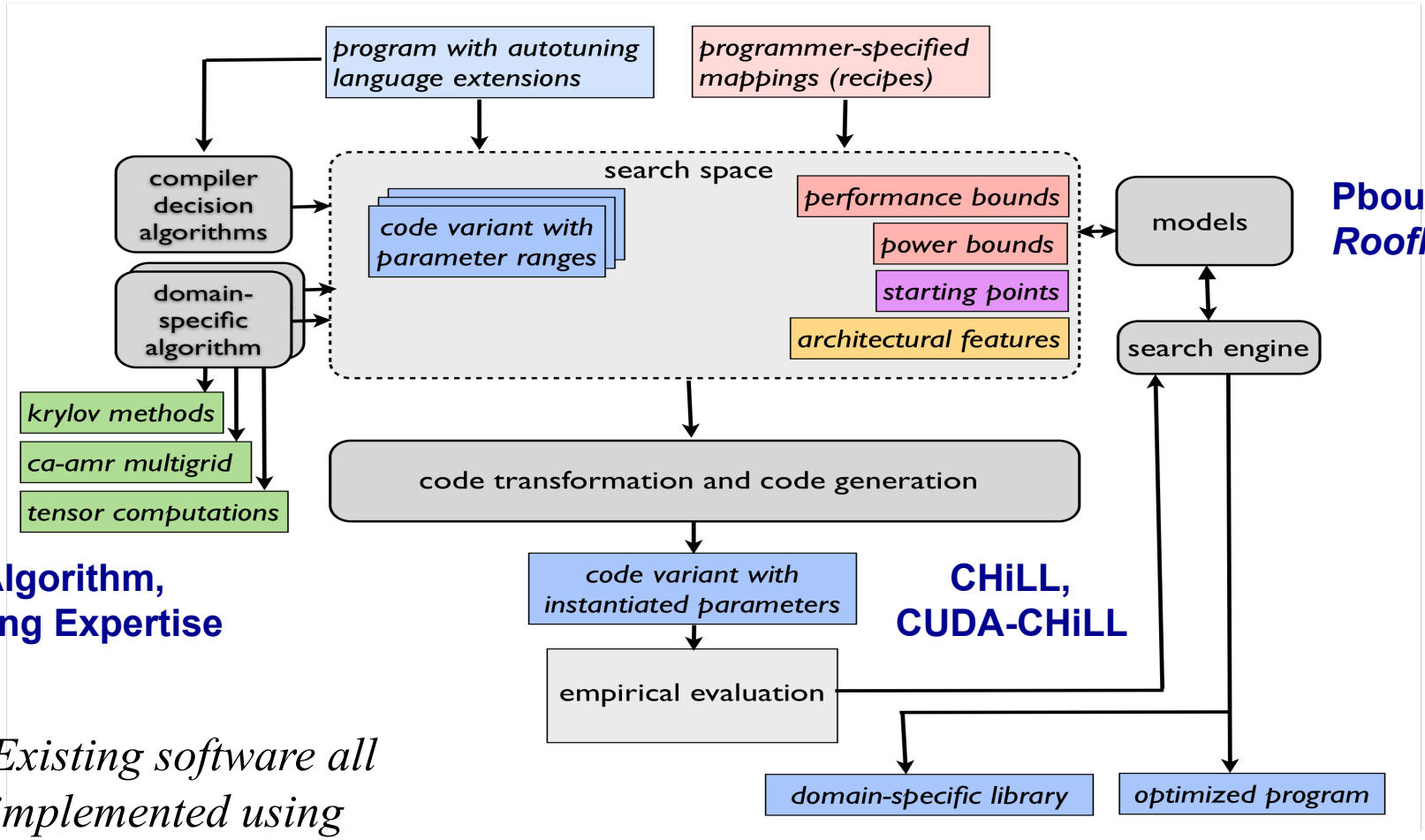
c. Compiler-based autotuning
   - Other examples: Saday et al., Swany et al., Eignenmann et al.
   - Related concepts: iterative compilation, learning-based compilation

X-TUNE Goals

A unified autotuning framework that seamlessly integrates programmer-directed and compiler-directed autotuning,

- Expert programmer and compiler work collaboratively to tune a code.
  - Unlike previous systems that place the burden on either programmer or compiler.
  - Provides access to compiler optimizations, offering expert programmers the control over optimization they so often desire.
- Design autotuning to be encapsulated in domain-specific tools
  - Enables less-sophisticated users of the software to reap the benefit of the expert programmers' efforts.
- Focus on Adaptive Mesh Refinement Multigrid (Combustion Co-Design Center,BoxLib,Chombo) and tensor contractions (TCE)

X-TUNE Structure



Existing software all implemented using ROSE AST.

Autotuning Language Extensions

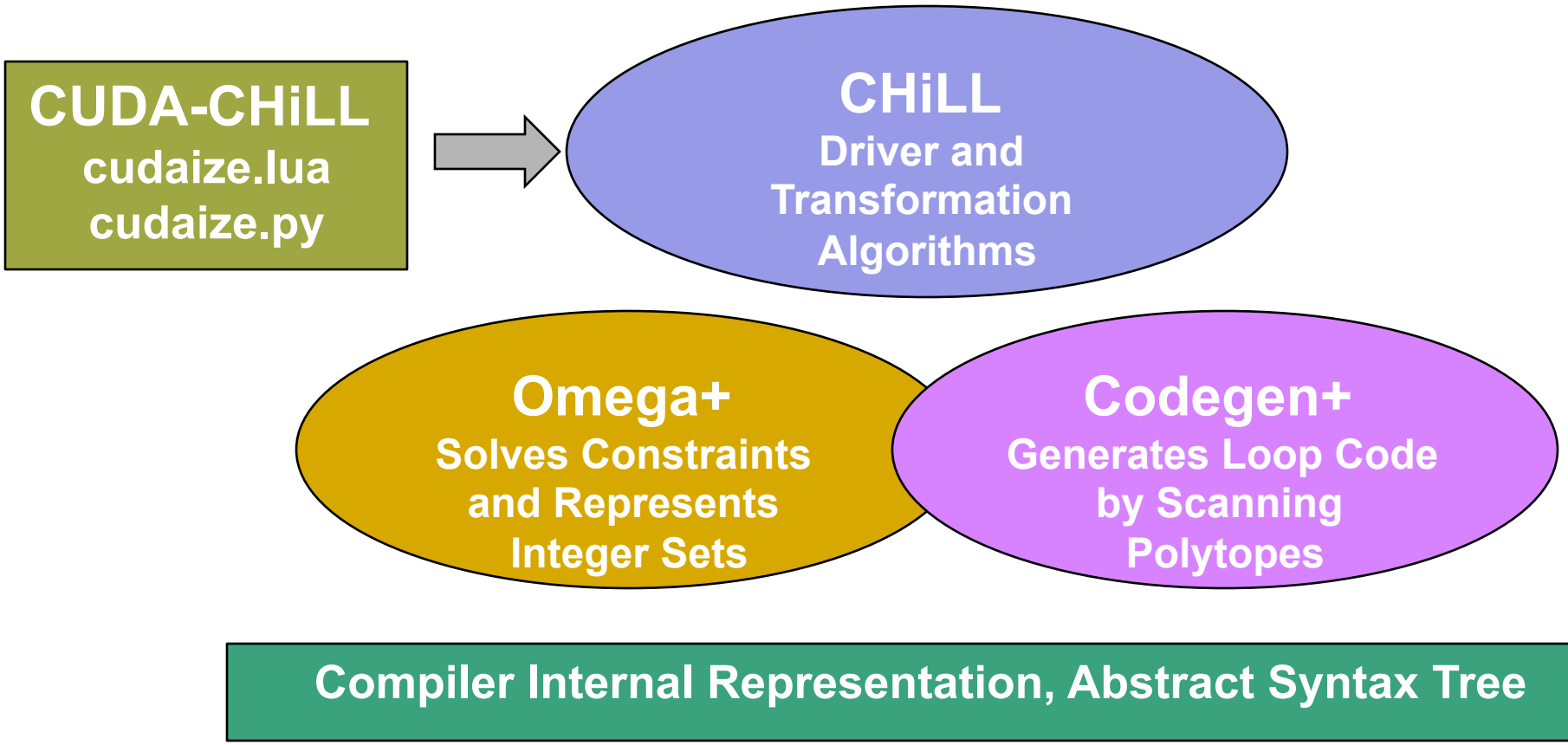
- **Tunable Variables**
  - An annotation on the type of a variable (as in Sequoia)
  - Additionally, specify range, constraints and a default value
- **Computation Variants**
  - An annotation on the type of a function (as in PetaBricks)
  - Additionally, specify (partial) selection criteria
  - Multiple variants may be composed in the same execution

Separate mapping description captures architecture-specific aspects of autotuning.

Compiler-Based Autotuning

- ***Foundational Concepts***
  - 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 with **transformation recipes**, or recipes generated automatically by compiler decision algorithm
  - Uses source-to-source transformation for portability, and to leverage vendor code generation
  - Requires ***restructuring of the compiler***

CHiLL Implementation



Transformation Recipes for Autotuning: Incorporate the Best Ideas from Manual Tuning

Nvidia GTX-280 implementation
Mostly corresponds to CUBLAS 2.x and Volkov's SC08 paper

```
1 tile_by_index({"i","j"},{TI,TJ},{l1_control="ii",l2_control="jj"},
                {"ii","jj","i","j"})
2 tile_by_index({"k"},{TK},{l1_control="kk"},
                {"ii","jj","kk","i","j","k"},strided)
3 tile_by_index({"i"},{TJ},{l1_control="tt",l1_tile="t"},
                {"ii","jj","kk","t","tt","j","k"})
4 cudaize("mm_GPU",{a=N*N,b=N*N,c=N*N},
          {block={"ii","jj"},thread={"t","tt"}})
5 copy_to_shared("tx","b",-16)
6 copy_to_registers("kk","c")
7 copy_to_texture("b")
8 unroll_to_depth(2)
```

```
1  tile_by_index({"i","j"},{TI,TJ},{l1_control="ii",l2_control="jj"},
                 {"ii","jj","i","j"})
2  tile_by_index({"k"},{TK},{l1_control="kk"},
                 {"ii","jj","kk","i","j","k"},strided)
3  tile_by_index({"i"},{TK},{l1_control="t",l1_tile="tt"},
                 {"ii","jj","kk","tt","t","j","k"})
4  tile_by_index({"j"},{TK},{l1_control="s",l1_tile="ss"},
                 {"ii","jj","kk","tt","t","ss","s","k"})
5  cudaize("mm_GPU",{a=N*N,b=N*N,c=N*N},
           {block={"ii","jj"},thread={"tt","ss"}})
6  copy_to_shared("tx","b",-16)
7  copy_to_texture("b")
8  copy_to_shared("tx","a",-16)
9  copy_to_texture("a")
10 copy_to_registers("kk","c")
11 unroll_to_depth(2)
```

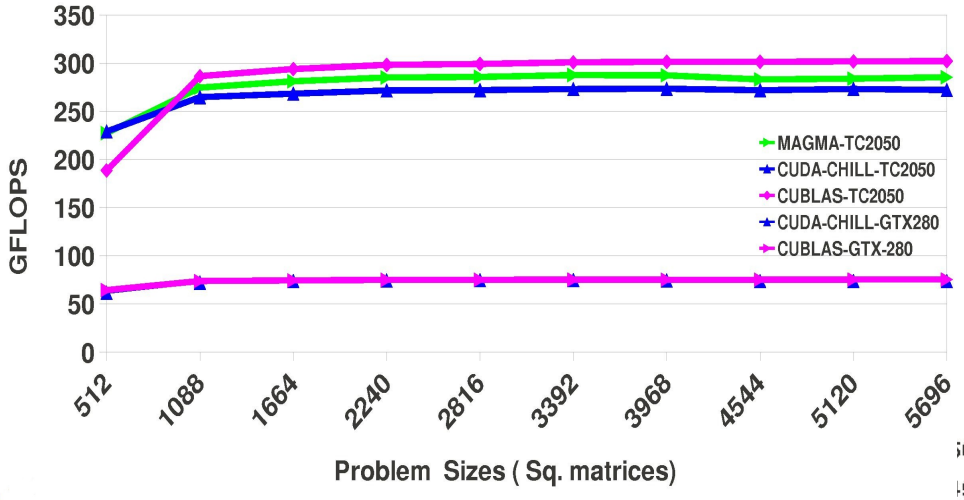
Nvidia TC2050 Fermi implementation
Mostly corresponds to CUBLAS 3.2 and MAGMA

Different computation decomposition leads to additional tile command

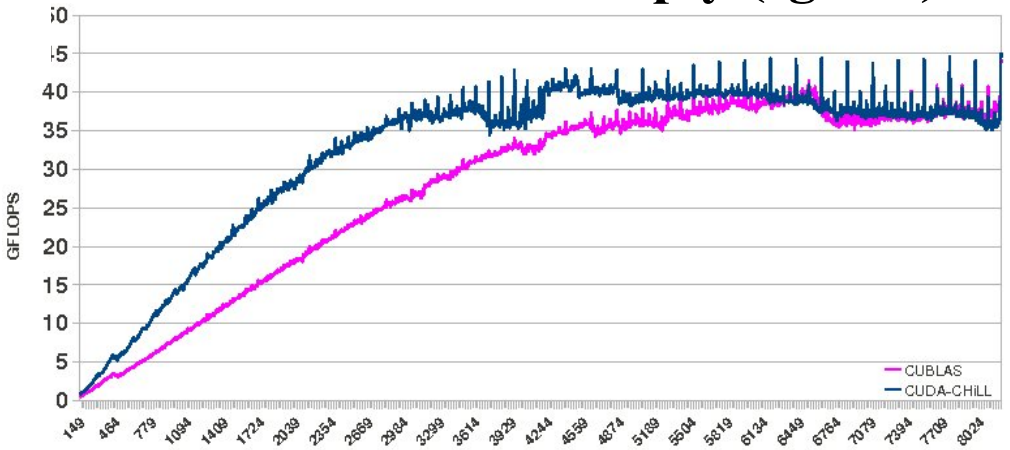
a in shared memory, both a and b are read through texture memory

Compiler + Autotuning can yield comparable and even better performance than manually-tuned libraries

Matrix-Matrix Multiply (dgemm)



Matrix-Vector Multiply (sgemv)



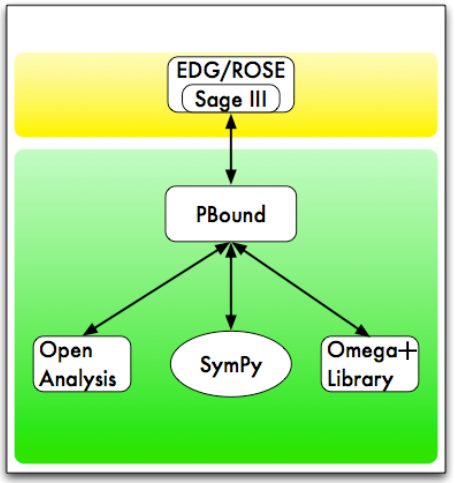
- Performance comparison with CUBLAS 3.2

"Autotuning, Code Generation and Optimizing Compiler Technology For GPUs," M. Khan, PhD Dissertation, University of Southern California, May 2012.

Argonne NATIONAL LABORATORY | BERKELEY LAB | THE UNIVERSITY OF UTAH | USC Viterbi School of Engineering Information Sciences Institute

Pbound: Performance Modeling for Autotuning

- Performance modeling increases the automation in autotuning
  - Manual transformation recipe generation is tedious and error-prone
  - Implicit models are not portable across platforms



- Models can unify programmer guidance and compiler analysis
  - Programmer can invoke integrated models to guide autotuning from application code
  - Compiler can invoke models during decision algorithms
- Models optimize autotuning search
  - Identify starting points
  - Prune search space to focus on most promising solutions
  - Provide feedback from updates in response to code modifications

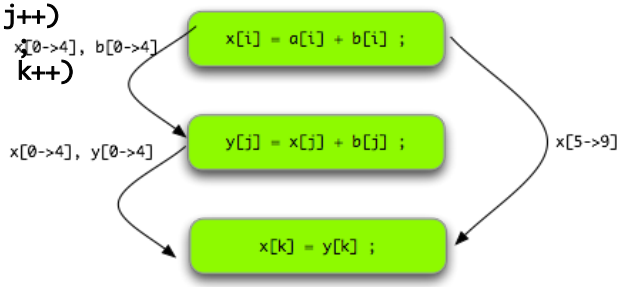
Argonne NATIONAL LABORATORY | BERKELEY LAB | THE UNIVERSITY OF UTAH | USC Viterbi School of Engineering Information Sciences Institute

Pbound: Reuse Distance and Cache Miss Prediction

Reuse distance

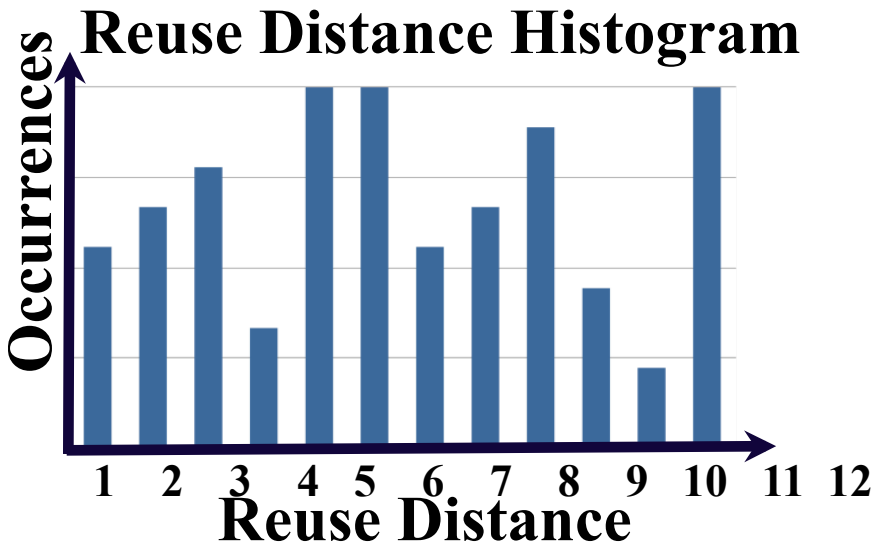
- For regular (affine) array references
  - Compute reuse distance, to predict data footprints in memory hierarchy.
  - Guides transformation and data placement decisions.

```
void foo() {
  int i, j, k, n;
  double y[10], x[10];
  double z[10], a[10], b[10];
  for (i = 0; i < 10; i++)
    x[i] = a[i] + b[i] ;
  for (j = 0; j < 5; j++)
    y[j] = x[j]+ b[j] ;
  for (k = 0; k < 10; k++)
    x[k] = y[k] ;
  return;
}
```



Cache miss prediction

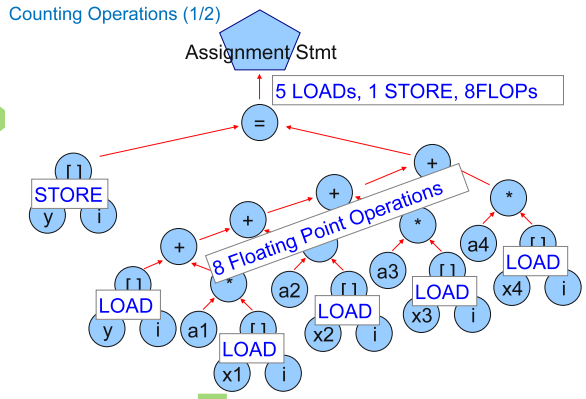
- Use to predict misses
- Assuming fully associative cache with n lines (optimistic case), a reference will hit if the reuse distance d<n.



Argonne NATIONAL LABORATORY · BERKELEY LAB · THE UNIVERSITY OF UTAH · USC Viterbi School of Engineering Information Sciences Institute

Pbound: Application signatures+architecture

```
void axpy4(int n, double *y, double a1, double *x1, double a2,
 double *x2, double a3, double *x3, double a4, double *x4) {
   register int i;
   for (i=0; i<=n-1; i++)
      y[i]=y[i]+a1*x1[i]+a2*x2[i]+a3*x3[i]+a4*x4[i];
}
```




```
#include "pbound_list.h"
void axpy4(int n, double *y, double a1, double *x1, double a2,
double *x2, double a3, double *x3, double a4, double *x4){
#ifdef pbound_log
pboundLogInsert("axpy.c@6@5",8,0,40 * ((n - 1) + 1) + 32,
          8 * ((n - 1) + 1),3 * ((n - 1) + 1) + 1,4 * ((n - 1) + 1));
 #endif
}
```


Argonne NATIONAL LABORATORY · BERKELEY LAB · THE UNIVERSITY OF UTAH · USC Viterbi School of Engineering Information Sciences Institute

How will modeling be used?

- Single-core and multicore models for application performance will combine architectural information, user-guidance, and application analysis
- Models will be coupled with decision algorithms to automatically generate CHiLL transformation recipes
  - Input: Reuse Information, Loop Information etc.
  - Output: Set of transformation scripts to be used by empirical search
- Feedback to be used to refine model parameters and behavior
- Small and large application execution times will be considered

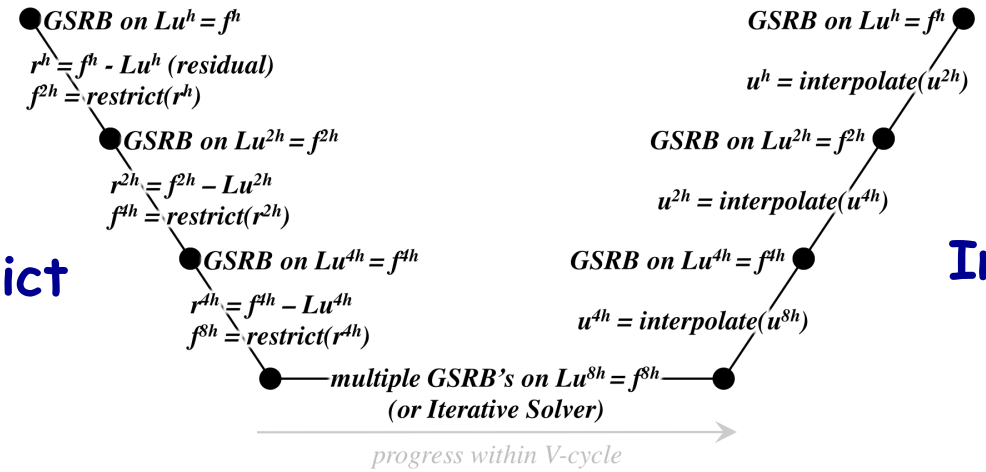
Example: Stencils and Multigrid

- Stencil performance bound, when bandwidth limited:

  Performance (gflops) <=

  stencil flops * STREAM bandwidth / grid size

- Multigrid solves Au=f by calculating a number of corrections to an initial solution at varying grid coarsenings ("V-cycle")
  - Each level in the v-cycle: perform 1-4 relaxes (~stencil sweeps).
  - Repeat multiple v-cycles reducing the norm of the residual by an order of magnitude each cycle.



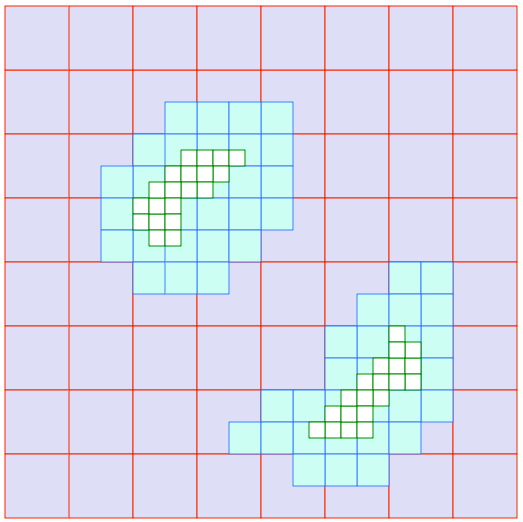
Restrict

Interpolate

Solve

Argonne NATIONAL LABORATORY · BERKELEY LAB · THE UNIVERSITY OF UTAH · USC Viterbi School of Engineering Information Sciences Institute

Multigrid and Adaptive Mesh Refinement



- Some regions of the domain may require finer fidelity than others.
- In Adaptive Mesh Refinement, we refine those regions to a higher resolution in time and space.
- Typically, one performs a multigrid "level solve" for one level (green, blue, red) at a time.
- Coarse-fine boundaries (neighboring points can be at different resolutions) complicate the calculation of the RHS and ghost zones for the level.
- **Each level is a collection of small (32^3 or 64^3) boxes to minimize unnecessary work.**
- These boxes will be distributed across the machine for load balancing (neighbors are not obvious/implicit)

Autotuning for AMR Multigrid

- Focus is addressing data movement, multifaceted:
  - Automate fusion of stencils within an operator. Doing so may entail aggregation of communication (deeper ghost zones)
  - Extend and automate the communication-avoiding techniques developed in CACHE.
  - Automate application of data movement-friendly coarse-fine boundary conditions.
  - Automate hierarchical parallelism within a node to AMR MG codes.
  - Explore alternate data structures
  - Explore alternate stencil algorithms (higher order, …)
- Proxy architectures: MIC, BG/Q, GPUs
- Encapsulate into an embedded DSL approach

Summary and Leverage

- Build integrated end-to-end autotuning, focused on AMR multigrid and tensor contractions
  - Language and compiler guidance of autotuning
  - Programmer and compiler collaborate to tune a code
  - Modeling assists programmer, compiler writer, and search space pruning.
- Leverage and integrate with other X-Stack teams
  - Our compiler technology all based on ROSE so can leverage from and provide capability to ROSE.
  - Domain-specific technology to facilitate encapsulating our autotuning strategies.
  - Collaborate with MIT on autotuning interface
  - Common run-time for a variety of platforms (e.g., GPUs and MIC), and supports a large number of potentially hierarchical threads.