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

Mary Hall

April 6, 2017

Participants

	Current	Previous
University of Utah	Mary Hall, Tharindu Rusira, Bob Wheeler, Derick Huth	Protonu Basu, Axel Rivera, Manu Shantharam
Lawrence Berkeley National Laboratory	Sam Williams, Protonu Basu	Lenny Oliker, Brian van Straalen, Phil Colella
Argonne National Laboratory	Prasanna Balaprakash, Paul Hovland	Thomas Nelson (Colorado), Jeff Hammond, Sri Krishna Narayanan, Stefan Wild
USC/ISI		Jacqueline Chame

THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Which version would you prefer to write?

```
/* Laplacian 7-point Variable-Coefficient Stencil */
for (k=0; k<N; k++)
   for (j=0; j<N; j++)
      for (i=0; i<N; i++
         temp[k][j][i] = b * h2inv * (
           beta_i[k][j][i+1] * ( phi[k][j][i+1] – phi[k][j][i] )
                 -beta_i[k][j][i] * ( phi[k][j][i] – phi[k][j][i-1] )
         +beta_j[k][j+1][i] * ( phi[k][j+1][i] – phi[k][j][i] )
         -beta_j[k][j][i] * ( phi[k][j][i] – phi[k][j-1][i] )
         +beta_k[k+1][j][i] * ( phi[k+1][j][i] – phi[k][j][i] )
         -beta_k[k][j][i] * ( phi[k][j][i] – phi[k-1][j][i] ) );
```

```
/* Helmholz */
for (k=0; k<N; k++)
   for (j=0; j<N; j++)
      for (i=0; i<N; i++)
         temp[k][j][i] = a * alpha[k][j][i] * phi[k][j][i] –
                               temp[k][j][i];
```

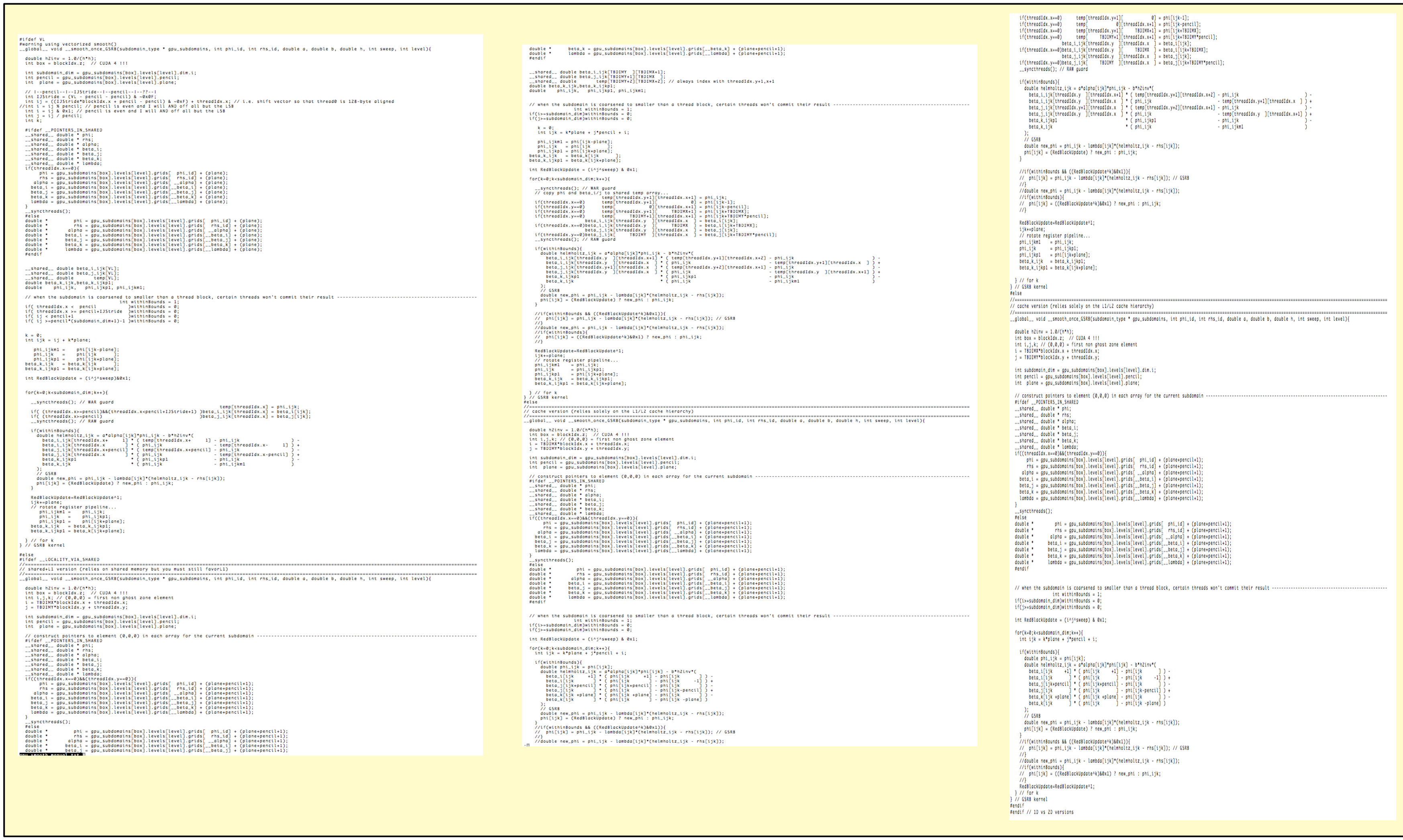
```
/* Gauss-Seidel Red Black Update */
for (k=0; k<N; k++)
   for (j=0; j<N; j++)
      for (i=0; i<N; i++){
         if ((i+j+k+color)%2 == 0 )
         phi[k][j][i] = phi[k][j][i] – lambda[k][j][i] *
            (temp[k][j][i] – rhs[k][j][i]);}
```

Code A: miniGMG baseline smooth operator approximately 13 lines of code

Code B: miniGMG optimized smooth operator approximately 170 lines of code

THE UNIVERSITY OF UTAH · Argonne NATIONAL LABORATORY · BERKELEY LAB · X-TUNE

And now GPU code?



Code C: miniGMG optimized smooth operator for GPU, 308 lines of code for just kernel

Which version would you prefer to write?

/* local_grad_3 computation from nek5000 */

w[nelt i j k] += Dt[l k] U[nelt n m l] D[j m] D[i n]

Code A:

1 line mathematical representation

Input to OCTOPI

/* local_grad3 from nek5000, generated CUDA code */

```
void local_grad_3(double *U,double *ur,double *us,double *Dt,double *ut,double *D){

 double *devI1Prt;
 double *devO3Prt;
 double *devO2Prt;
 double *devO1Prt;
 double *devI2Prt;
 double *devI3Prt;

 struct timeval time1, time2;
 double time;
 std::ofstream timefile;
std::ofstream MlsearchFile;

   cudaMalloc(((void **)(&devI1Prt)),100 * sizeof(double ));
 cudaMalloc(((void **)(&devI2Prt)),1000000 * sizeof(double ));
 cudaMalloc(((void **)(&devI3Prt)),100 * sizeof(double ));
 cudaMalloc(((void **)(&devO1Prt)),1000000 * sizeof(double ));
 cudaMalloc(((void **)(&devO2Prt)),1000000 * sizeof(double ));
 cudaMalloc(((void **)(&devO3Prt)),1000000 * sizeof(double ));


 cudaMemcpy(devI1Prt,D,100 * sizeof(double ),cudaMemcpyHostToDevice);
 cudaMemcpy(devI2Prt,U,1000000 * sizeof(double ),cudaMemcpyHostToDevice);
 cudaMemcpy(devI3Prt,Dt,100 * sizeof(double ),cudaMemcpyHostToDevice);

 dim3 dimGrid0 = dim3(1000,10);
 dim3 dimBlock0 = dim3(10,10);
  gettimeofday(&time1, 0);
  local_grad_3_GPU_0<<<dimGrid0,dimBlock0>>>(devO1Prt,devO2Prt,devO3Prt,devI1Prt,devI2Prt,devI3Prt);
  cudaThreadSynchronize();
  gettimeofday(&time2, 0);

  time = (1000000.0*(time2.tv_sec-time1.tv_sec) + time2.tv_usec-time1.tv_usec)/1000000.0;
  timefile.open("./times/time__10_m_k_nelt_n.txt", std::ofstream::out | std::ofstream::app );
  timefile<<"Time spent in rose__orio_chill_ _10_m_k_nelt_n.cu: "<<time<<std::endl;
  timefile.close();
  MlsearchFile.open("./times/MlsearchFile__10_m_k_nelt_n.txt", std::ofstream::out | std::ofstream::app );
  MlsearchFile<<"local_grad_3 "<<time<<" [UF_0] 10 [PERMUTE_0_TX0] m [PERMUTE_0_TY0] k [PERMUTE_0_BX0] nelt
se__orio_chill__10_m_k_nelt_n.cu"<<std::endl;
  MlsearchFile.close();

  cudaMemcpy(ur,devO1Prt,1000000 * sizeof(double ),cudaMemcpyDeviceToHost);
  cudaMemcpy(us,devO2Prt,1000000 * sizeof(double ),cudaMemcpyDeviceToHost);
  cudaMemcpy(ut,devO3Prt,1000000 * sizeof(double ),cudaMemcpyDeviceToHost);

  cudaFree(devI1Prt);
  cudaFree(devO3Prt);
  cudaFree(devO2Prt);
  cudaFree(devO1Prt);
  cudaFree(devI2Prt);
  cudaFree(devI3Prt);

}
```

```
__global__ void local_grad_3_GPU_0(double *ur,double *us,double *ut,double *D,double *U,double *Dt)
{
  int bx;
  double newVariable0;
  double newVariable1;
  double newVariable2;
  bx = blockIdx.x;
  int by;
  by = blockIdx.y;
  int tx;
  tx = threadIdx.x;
  int ty;
  ty = threadIdx.y;
  int t4;
  int t6;
  int t8;
  int t10;
  newVariable0 = ur[bx * 10 * 10 * 10 + by * 10 * 10 + tx * 10 + ty];
  newVariable1 = us[bx * 10 * 10 * 10 + by * 10 * 10 + tx * 10 + ty];
  newVariable2 = ut[bx * 10 * 10 * 10 + by * 10 * 10 + tx * 10 + ty];
    newVariable0 = newVariable0 + D[0 * 10 + ty] * U[bx * 10 * 10 * 10 + by * 10 * 10 + tx * 10 + 0];
  newVariable1 = newVariable1 + U[bx * 10 * 10 * 10 + by * 10 * 10 + 0 * 10 + ty] * Dt[tx * 10 + 0];
  newVariable2 = newVariable2 + U[bx * 10 * 10 * 10 + 0 * 10 * 10 + tx * 10 + ty] * Dt[by * 10 + 0];
  newVariable0 = newVariable0 + D[(0 + 1) * 10 + ty] * U[bx * 10 * 10 * 10 + by * 10 * 10 + tx * 10 + (0 + 1)];
  newVariable1 = newVariable1 + U[bx * 10 * 10 * 10 + by * 10 * 10 + (0 + 1) * 10 + ty] * Dt[tx * 10 + (0 + 1)];
  newVariable2 = newVariable2 + U[bx * 10 * 10 * 10 + (0 + 1) * 10 * 10 + tx * 10 + ty] * Dt[by * 10 + (0 + 1)];
  newVariable0 = newVariable0 + D[(0 + 2) * 10 + ty] * U[bx * 10 * 10 * 10 + by * 10 * 10 + tx * 10 + (0 + 2)];
  newVariable1 = newVariable1 + U[bx * 10 * 10 * 10 + by * 10 * 10 + (0 + 2) * 10 + ty] * Dt[tx * 10 + (0 + 2)];
  newVariable2 = newVariable2 + U[bx * 10 * 10 * 10 + (0 + 2) * 10 * 10 + tx * 10 + ty] * Dt[by * 10 + (0 + 2)];
 newVariable0 = newVariable0 + D[(0 + 2) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 2)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 2) * 10 + ty] * Dt[tx * 10 + (0 + 2)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 2) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 2)];
 newVariable0 = newVariable0 + D[(0 + 3) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 3)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 3) * 10 + ty] * Dt[tx * 10 + (0 + 3)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 3) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 3)];
 newVariable0 = newVariable0 + D[(0 + 4) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 4)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 4) * 10 + ty] * Dt[tx * 10 + (0 + 4)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 4) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 4)];
 newVariable0 = newVariable0 + D[(0 + 5) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 5)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 5) * 10 + ty] * Dt[tx * 10 + (0 + 5)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 5) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 5)];
 newVariable0 = newVariable0 + D[(0 + 6) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 6)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 6) * 10 + ty] * Dt[tx * 10 + (0 + 6)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 6) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 6)];
 newVariable0 = newVariable0 + D[(0 + 7) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 7)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 7) * 10 + ty] * Dt[tx * 10 + (0 + 7)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 7) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 7)];
 newVariable0 = newVariable0 + D[(0 + 8) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 8)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 8) * 10 + ty] * Dt[tx * 10 + (0 + 8)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 8) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 8)];
 newVariable0 = newVariable0 + D[(0 + 9) * 10 + ty] * U[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + (0 + 9)];
 newVariable1 = newVariable1 + U[by * 10 * 10 * 10 + bx * 10 * 10 + (0 + 9) * 10 + ty] * Dt[tx * 10 + (0 + 9)];
 newVariable2 = newVariable2 + U[by * 10 * 10 * 10 + (0 + 9) * 10 * 10 + tx * 10 + ty] * Dt[bx * 10 + (0 + 9)];
 ur[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + ty]  = newVariable0;
us[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + ty]  = newVariable1;
ut[by * 10 * 10 * 10 + bx * 10 * 10 + tx * 10 + ty]  = newVariable2;

}
```

Code B:

Generated CUDA+harness, 122 lines of code

Exascale Challenges: Code B is not Unusual

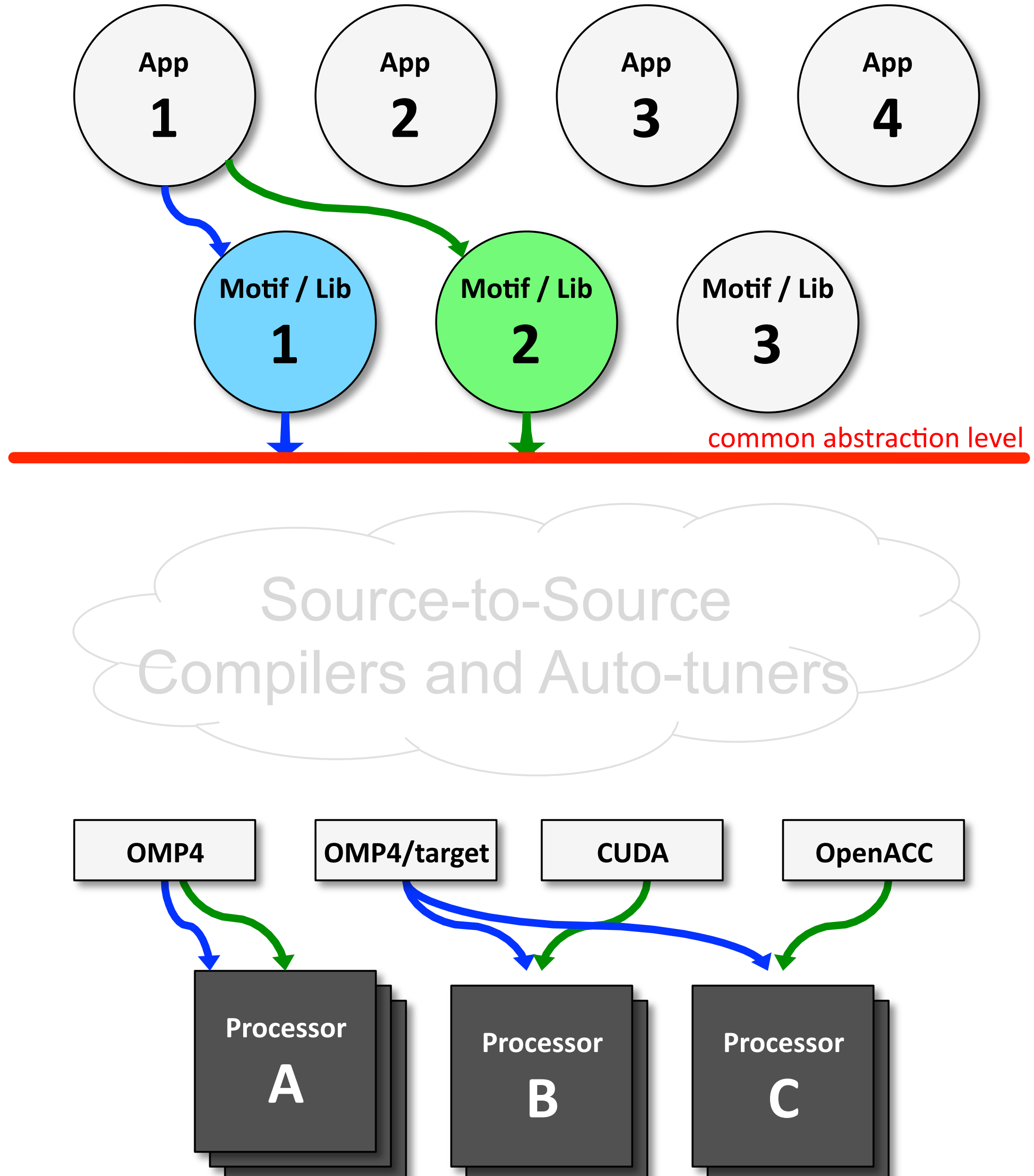
- Performance portability?
  - Particularly across fundamentally different CPU and GPU architectures
- Programmer productivity?
  - High performance implementations will require low-level specification that exposes architecture
- Software maintainability and portability?
  - May require multiple implementations of application

Current solutions

- Follow MPI and OpenMP standards
  - Same code unlikely to perform well across CPU and GPU
  - Vendor C and Fortran compilers not optimized for HPC workloads
- Some domain-specific framework strategies
  - Libraries, C++ template expansion, standalone DSL
  - Not ***composable*** with other optimizations

Programming Systems Must Hide Complexity!

- Define a common abstraction(s) that programmers can target
- Use compilation tools to map to optimal implementation
  - hide **programming model choices** from users
  - hide **architectural complexity** from users
  - hide **tuning** from users



THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

X-TUNE Goals

- Application programmer expresses key computation at a high level ***(Code A)***
  - Sequential C or domain-specific specification
- Code transformations are applied
  - Existing and domain-specific transformations
  - Generates a collection of optimized implementations
  - Includes thread-level code generation (OpenMP and CUDA)
- Autotuning
  - Searches the space of implementations to find the best match to execution context
  - Selects optimized implementation ***(Code B)***
- Automation mitigates correctness, productivity, portability, maintainability concerns

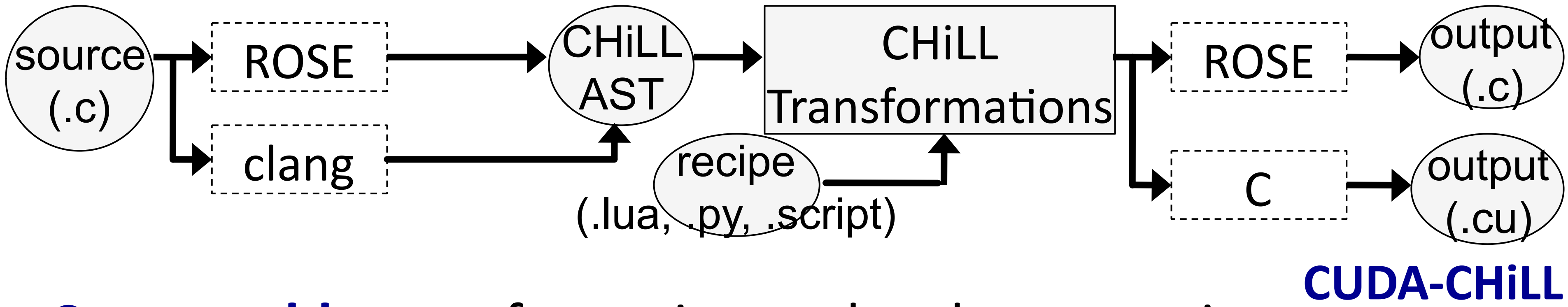
X-TUNE automates the process of converting Code A to Code B.

See today's demonstrations!

X-TUNE Approach

- For each ***motif***, start with manually-tuned code or work with developer of new code
  - What transformations are needed to target specific architectures?
  - What performance questions can be addressed by autotuning?
- Attempt to automate
  - Exploit existing compiler transformations
  - Develop new domain-specific transformations and required analysis and code generation support
  - Develop decision algorithms
- Collect application code from collaborators, Co-Design Centers and other DOE application teams
  - Generalize from experiments with manually-tuned code

X-TUNE Key Ideas



- **Composable** transformation and code generation
  - Leverage rich set of existing transformations and code generation capability in **polyhedral framework**
- Extensible to domain-specific transformations and decision algorithms
  - **Compose** with existing transformations
- Optimization strategies and parameters exposed to autotuning via transformation recipes
- Search space navigation
  - Search framework can be standalone tool (e.g., Orio, OpenTuner, Active Harmony, Nitro)

THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Example Motifs Supported by X-TUNE

Motif	Input	Existing Transformations	Domain-specific transformations	Autotuning	Search
Geometric Multigrid	Sequential C computation (w/ MPI and OpenMP harness)	Communication-avoiding: fusion, tile, wavefront (skew&permute), OpenMP, CUDA	Ghost zones, Partial sums	Ghost zone depth, threading, strategy at each level of V-cycle	Simple, full space
Tensor Contraction	Mathematical Formula	Tile, permute, scalar replacement, unroll, CUDA	Rewriting, Decision algorithm	Loop order, CUDA threading	SURF
Sparse Matrix Computation	Sequential C with CSR matrix	Tile, permute, skew, unroll, reduction, scalar expansion, OpenMP, CUDA	Generate inspectors, coalesce, make-dense, compact, split, level sets	Threading, matrix repr.	Simple, full space

X-TUNE DEMO

SUPER/NSF

THE UNIVERSITY OF UTAH

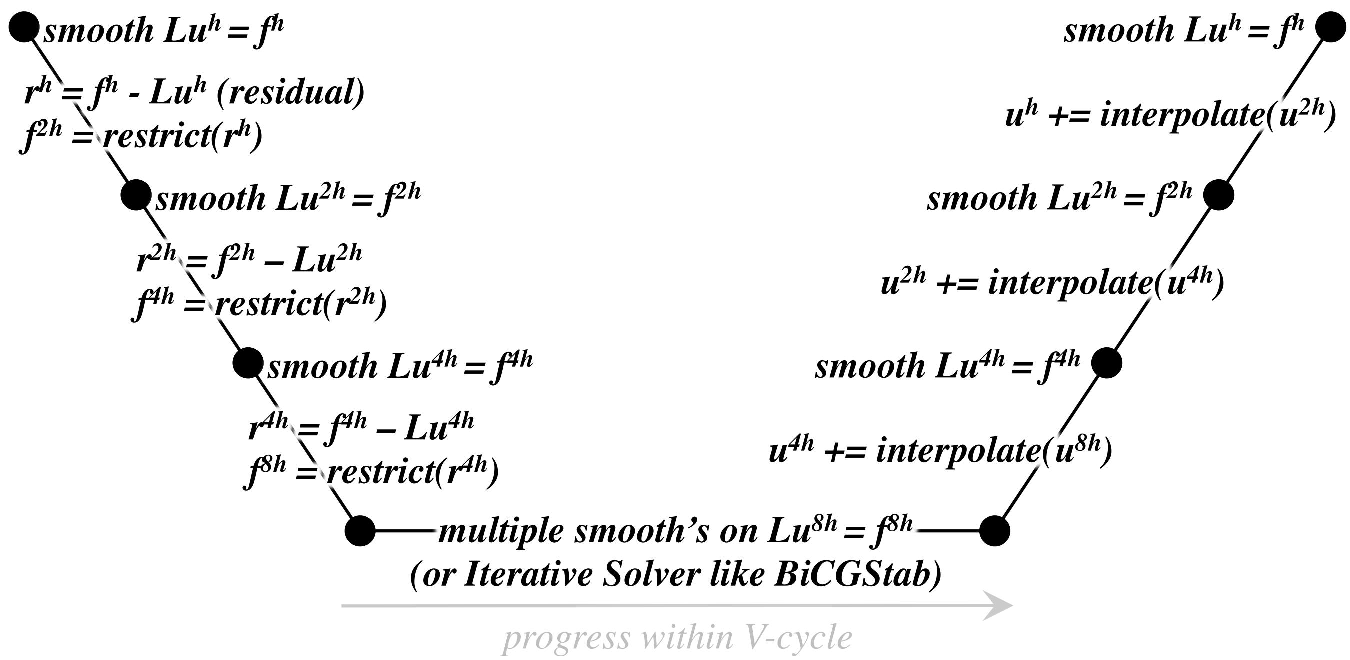
Argonne NATIONAL LABORATORY

BERKELEY LAB

X-TUNE

Geometric Multigrid

- Multigrid solves elliptic PDEs in O(N) computational complexity by using a hierarchical approach.

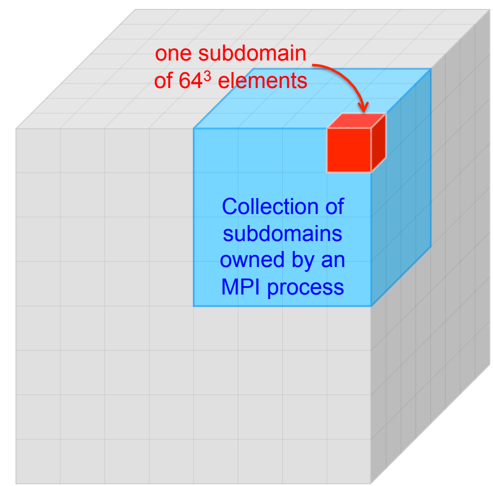


- ❖ Geometric Multigrid (**GMG**) is specialization in which the operator (A) is simply a stencil on a structured grid (i.e. *matrix-free*)
- ❖ **Stresses performance at different degrees of parallelism, locality, working set sizes, etc…**
- ❖ **Optimization strategy varies across different levels of V-cycle, even on one architecture!**

THE UNIVERSITY OF UTAH   Argonne NATIONAL LABORATORY   BERKELEY LAB   X-TUNE

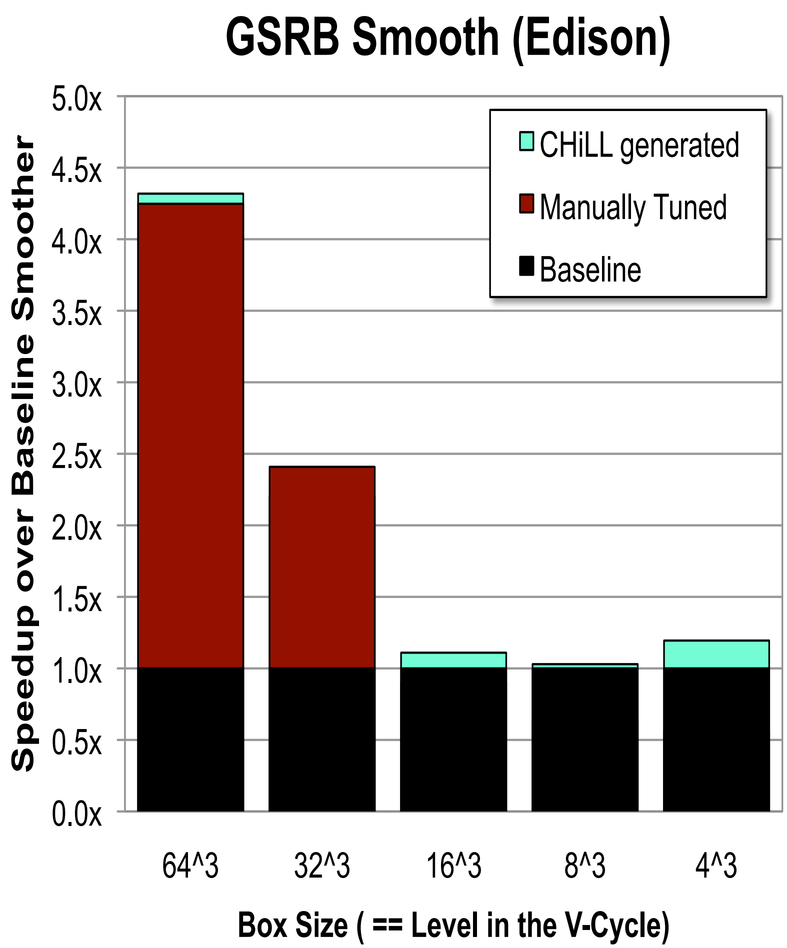
miniGMG Benchmark

- **miniGMG proxies the MG solves in BoxLib/Chombo codes**
  - **Predecessor to ExACT Co-Design Center's HPGMG**
- Distributed memory (MPI) implementation
- **operator** is configurable
  - 7pt variable coefficient **proxies LMC**
  - 7pt constant coefficient is simpler
  - 125pt/27pt/13pt high-order stencils
- **smoother** in the v-cycle is configurable
  - Gauss Seidel, Red-Black (GSRB) = **proxies**
  - Jacobi (mathematically weaker)
- **bottom solver** is configurable
  - multiple GSRB's
  - Krylov solver like **BiCGStab**, CG, CA-BiCGStab, CA-CG, etc…



Optimized Code A can beat Code B!

- miniGMG optimized w/CHiLL
  - fused operations
  - created a communication-avoiding wavefront
  - **auto-parallelized (OpenMP)**
- **autotuning** finds the best implementation for each box size
  - wavefront depth (degree of comm. avoiding)
  - Turn on/off optimizations (fusion, wavefront)
  - nested OpenMP configuration
  - inter-thread synchronization (barrier vs. P2P)
- For fine grids (large arrays) CHiLL attains a **4.3x speedup** over baseline

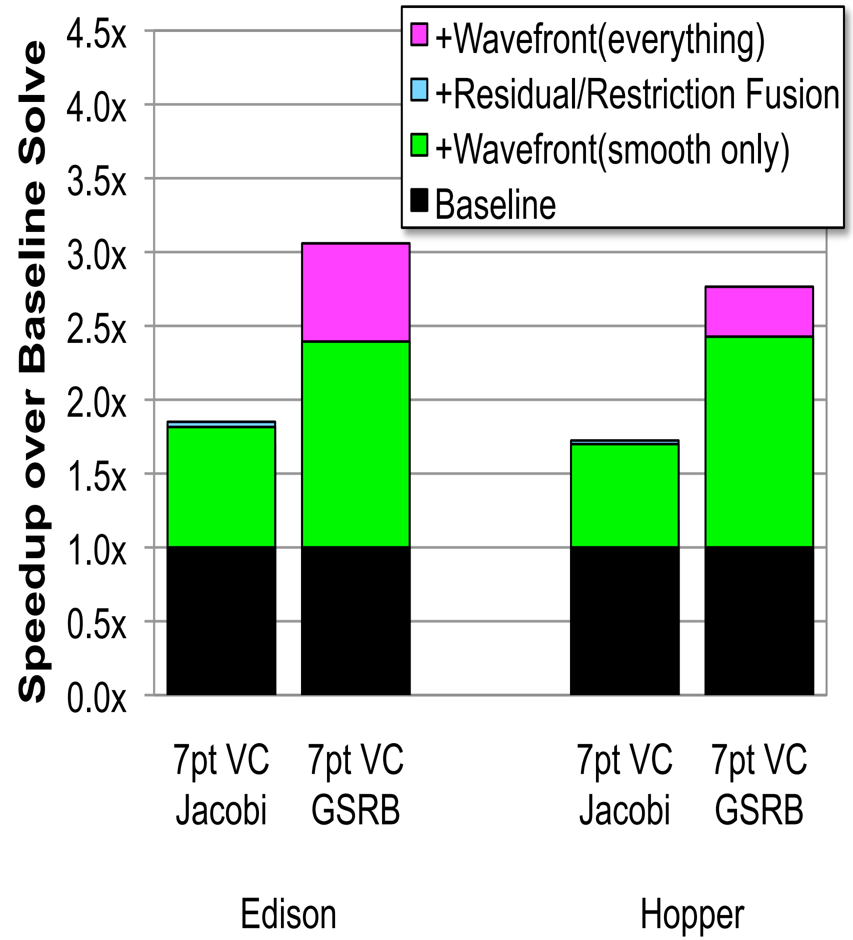


THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Flexibility

- Fuse the residual and restriction operations into the wavefront as well
  - read u^h, R^h, and coefficients once
  - perform 4 smooths (**no additional data movement**)
  - write smoothed u^h and new R^{2h}

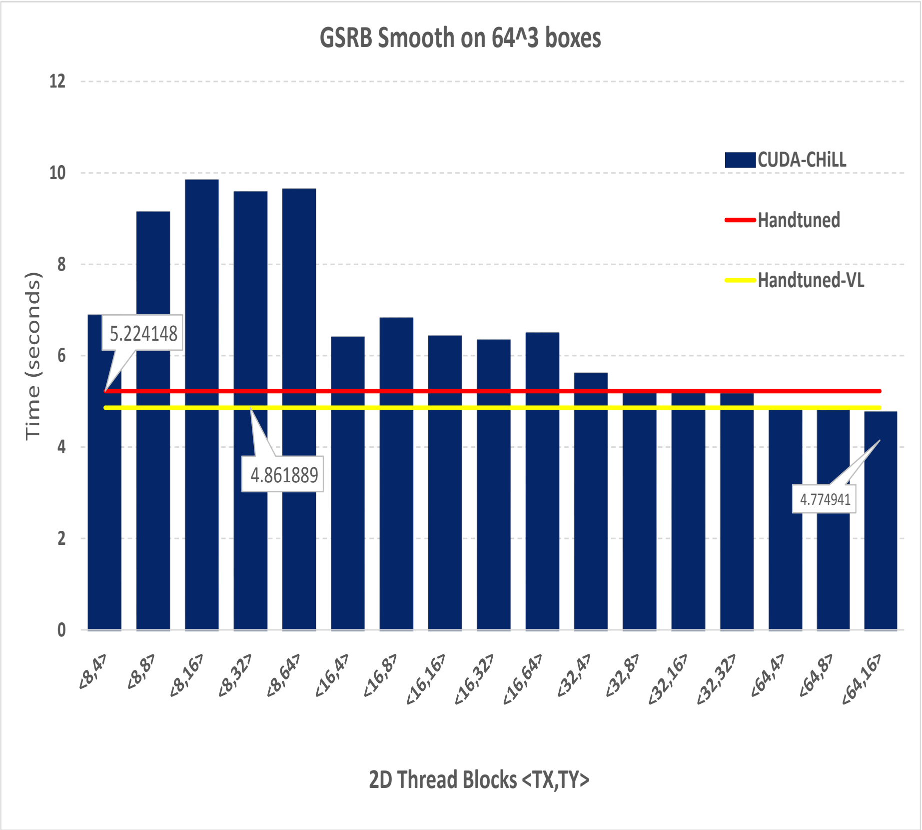
- Apply these transformations to a different smoother and autotune it
  - up to **3x improvement in MGSolve**



THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

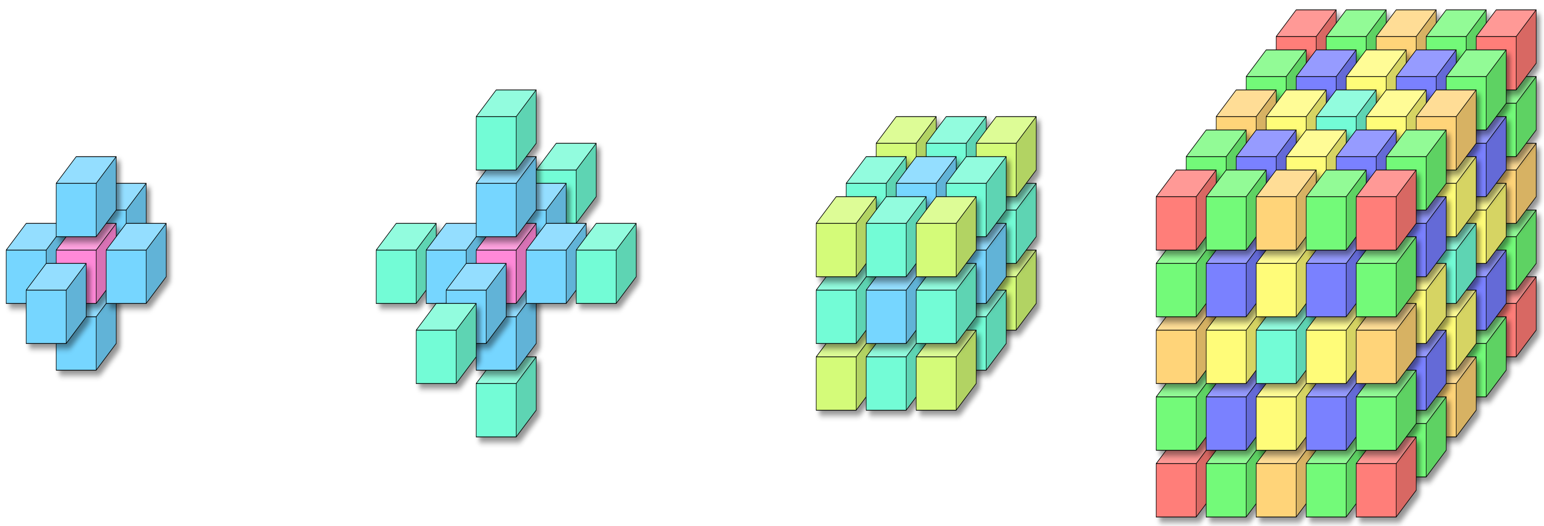
Retargetable and Performance Portable: Optimized Code A can beat Code C!

- CHiLL can obviate the need for architecture-specific programming models like CUDA
  - CUDA-CHiLL took the sequential GSRB implementation (.c) and **generated CUDA** that runs on NVIDIA GPUs
  - CUDA-CHiLL tunes for the current target machine whereas static implementations hand-optimize for yesterday's GPUs
  - CUDA-CHiLL autotuned over the thread block sizes and is ultimately **2% faster** than the hand-optimized minigmg-cuda (**Code C**)



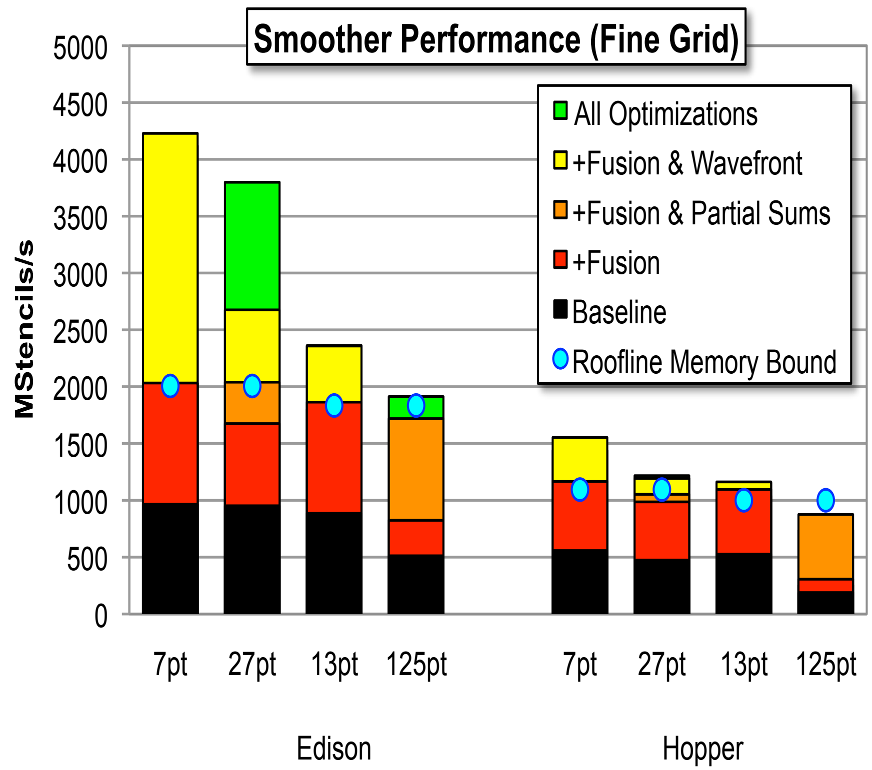
Extensible to Domain-Specific Optimizations

- Applied mathematicians are exploring how changing the stencils may be better suited for future architecture trends
- Consider the following variations (stencils) on the discretization of the Laplacian
  - low-level implementations (optimized OMP4) may provide high performance
  - but are one-off solutions as requisite optimizations/tuning change from one stencil to the next



Extensible to Domain-Specific Optimizations

- CHiLL optimized/tuned each of these stencils
  - Introduced **partial sums** optimization to avoid redundant computation for compute-bound high-order stencils
  - selected unique optimizations for each stencil and at each level of the MG V-Cycle
  - Without a communication-avoiding wavefront, CHiLL delivered performance **near the Roofline bound**.
  - Using a wavefront, CHiLL can nearly **double** the nominal Roofline performance for the 7- and 27-point operators.



THE UNIVERSITY OF UTAH   Argonne NATIONAL LABORATORY   BERKELEY LAB   X-TUNE

Example Transformation Recipes

- These can be manually-written (miniGMG) or automatically generated (tensor contraction)

/* jacobi_box_4_64.py, 27-pt stencil, 64^3 box size */

```
from chill import *

#select which computation to optimize
source('jacobi_box_4_64.c')
procedure('smooth_box_4_64')
loop(0)
original() # fuse wherever possible

#create a parallel wavefront
skew([0,1,2,3,4,5],2,[2,1])
permute([2,1,3,4])

#partial sum for high order stencils and fuse result
distribute([0,1,2,3,4,5],2)
stencil_temp(0)
stencil_temp(5)
fuse([2,3,4,5,6,7,8,9],1)
fuse([2,3,4,5,6,7,8,9],2)
fuse([2,3,4,5,6,7,8,9],3)
fuse([2,3,4,5,6,7,8,9],4)
```

/* gsrb.lua, variable coefficient GSRB, 64^3 box size */

```
init("gsrb_mod.cu", "gsrb",0,0)
dofile("cudaize.lua") # custom commands in lua

# set up parallel decomposition, adjust via autotuning
TI=32
TJ=4
TK=64
TZ=64

tile_by_index(0, {"box","k","j", "i"},{TZ,TK, TJ, TI},
{l1_control="bb", l2_control="kk", l3_control="jj",
l4_control="ii"},{"bb","box","kk","k","jj","j","ii","i"})

cudaize(0, "kernel_GPU",
{_temp=N*N*N*N,_beta_i=N*N*N*N,
_phi=N*N*N*N},{block={"ii","jj","box"},
thread={"i","j"}},{})
```

THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Tensor Contraction: Spectral Element Method from nek5000/nekbone (CESAR)

$$C = A \otimes B \underline{u}$$

- A and B are square matrices
- $\underline{u}$ is a component vector
- In 2-d, C can be computed:

$$c_{i,j} = \sum_{l} \sum_{k} a_{j,l} b_{i,k} u_{k,l}$$

Order O(n^4)

Optimize by rewriting to the following:

$$C = (A \otimes I)(I \otimes B)\underline{u}$$

Partial Results: $\underline{w} = (I \otimes B)\underline{u} \longrightarrow w_{i,j} = \sum_{l} u_{i,l}\, b^{T}_{l,j}$

Order O(n^3)

Final Results: $C = (A \otimes I)\underline{w} \longrightarrow c_{i,j} = \sum_{k} a_{i,k} w_{k,j}$

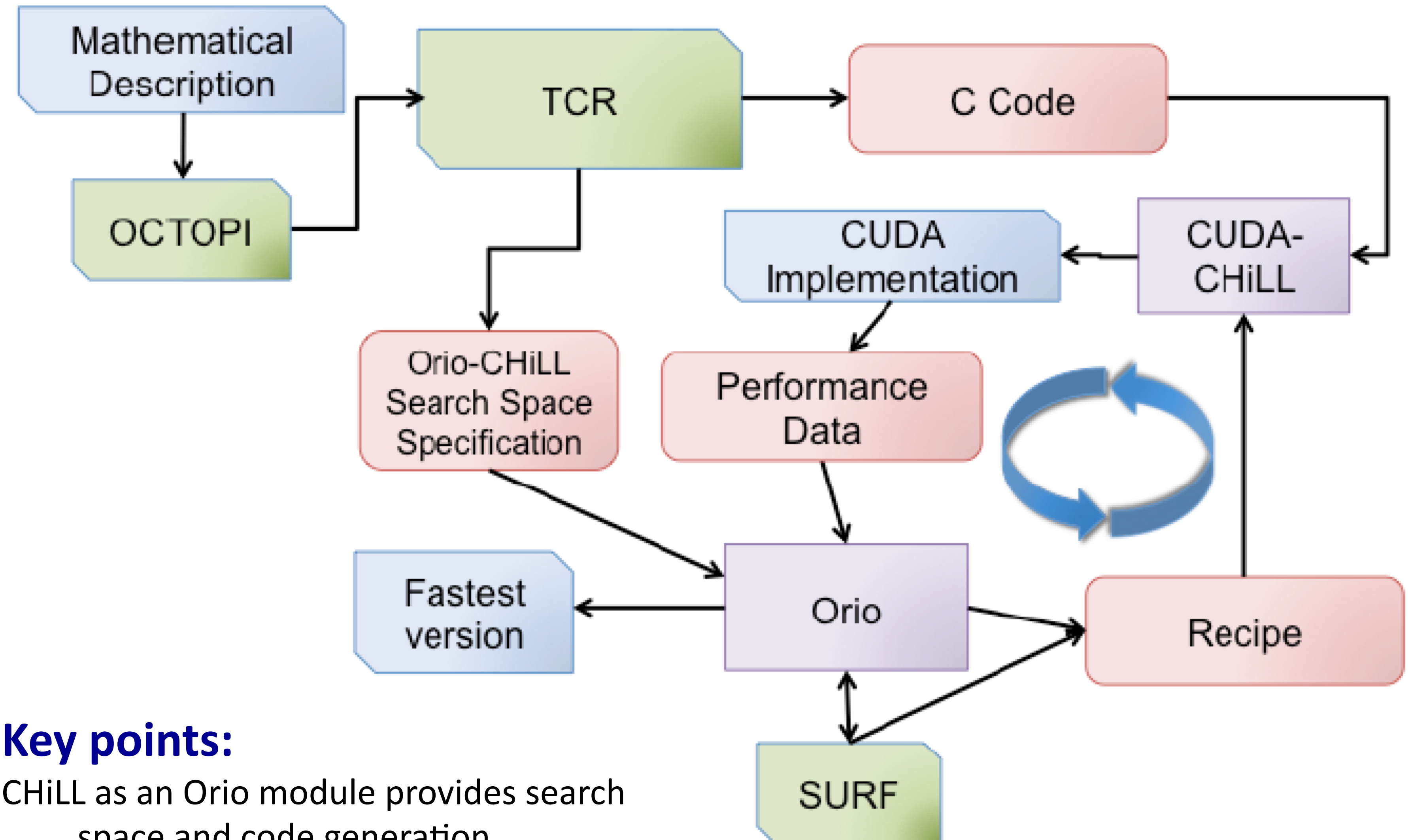
THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Tensor Contraction: Challenging Mapping, Particularly for GPU

- What is the optimal contraction order?
- What is the optimal loop order? (N! different implementations)
- GPU challenges: small dimensions, memory hierarchy effects
- Search space is discontinuous, noisy, and expensive to evaluate
- X-TUNE Approach:
  - Fully automate from mathematical description to GPU code generation (**Code A** to **Code B**)
  - Automate and reduce search time across intractable brute force search space

THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Baracuda Framework



Key points:

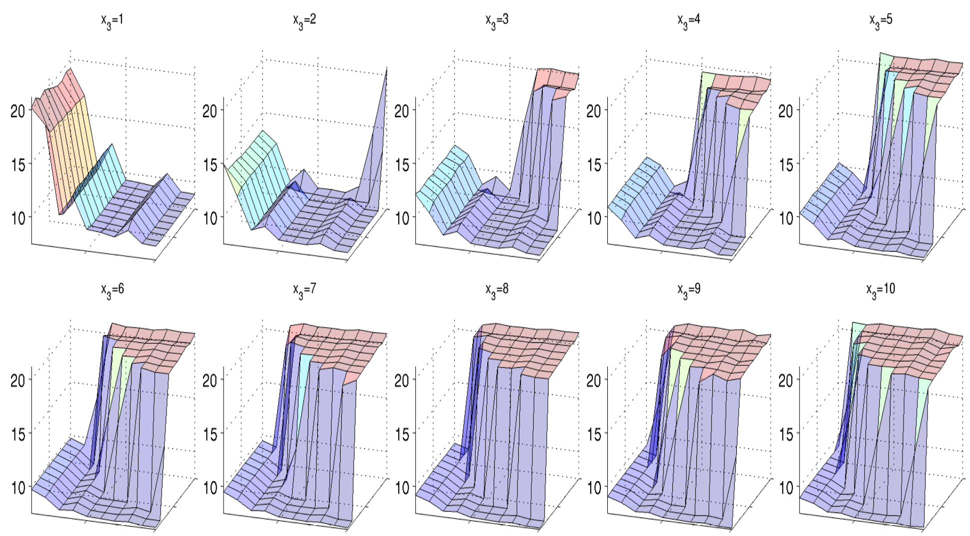
CHiLL as an Orio module provides search space and code generation

SURF manages exploration of search space

X-TUNE

SURF: Model-Based Search

- Search Using **Random Forests**
  - state-of-the-art statistical machine learning algorithm
  - handles binary permutation parameters
  - handles nonlinear parameter interactions
- Approach
  - start with promising small set of parameter configurations
  - evaluate performance
  - fit surrogate model (ML)
  - predict new set of high-performing configurations
  - iterate

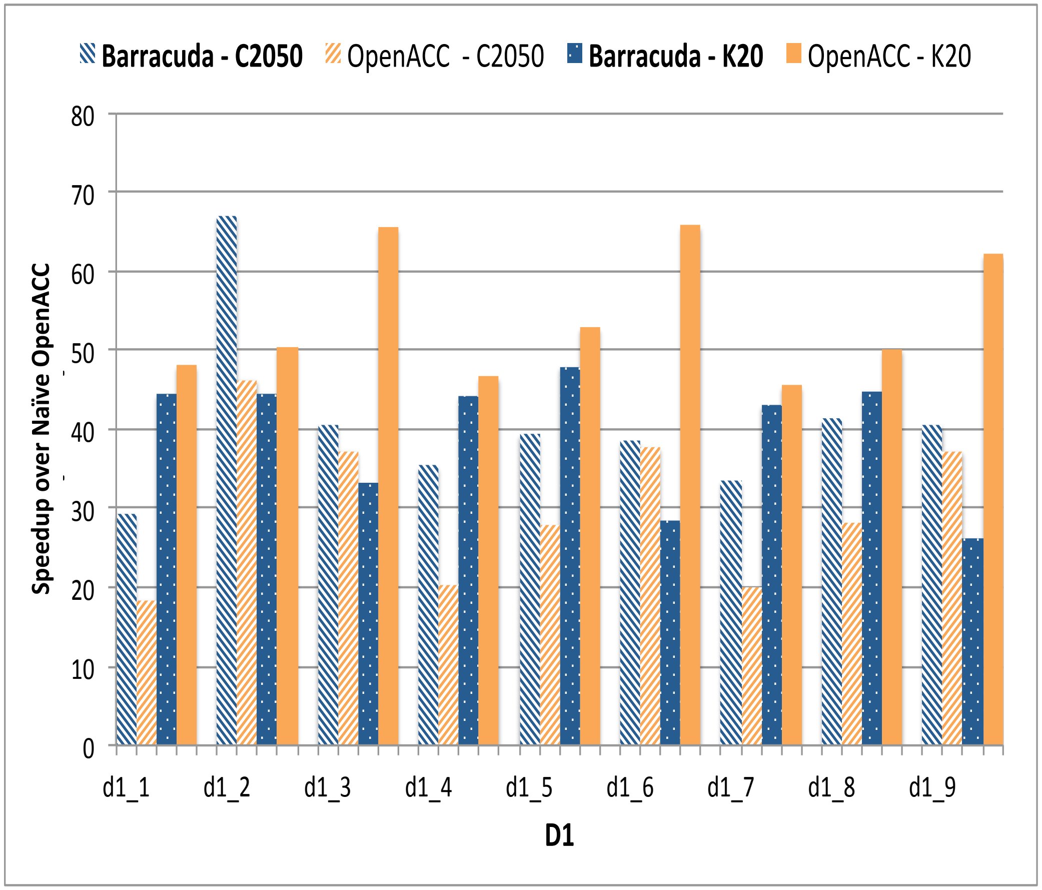


Example Surrogate Model Fitted to Sampled Performance (iterative refinement improves the statistical model)

THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Optimizing NWChem

- Extracted representative on-node tensor contractions from NWChem/TCE
  - many small contractions
  - atypical of OpenACC use model
- Baracuda generates optimized CUDA for NVIDIA's Fermi or Kepler GPUs
- Manually modified CUDA to OpenACC
  - Naïve replaces CUDA with OpenACC, but uses same loop order and parallelization
  - OpenACC = naïve + manual explicit control over hierarchical parallelism



THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Optimizing Nekbone

- Nekbone (CESAR CoDesign Center Proxy App) with optimized local_grad_3 and local_grad_3t
  - Many, small (e.g. 12x12x12) contractions
  - Nominally implemented as many BLAS3 calls
- Baracuda generates optimized CUDA for NVIDIA's Fermi or Kepler GPUs
- Compare to single Haswell core.

Speedup over 1-core Haswell	Naïve OpenACC	Optimized OpenACC	Baracuda (CUDA)
K20	2.86	12.39	36.47
C2050	1.18	19.21	34.65

Summary of X-TUNE Accomplishments

- Demonstrated for two motifs, Geometric Multigrid and Tensor Contraction
  - Automated architecture-specific optimization from high-level specification
  - Performance rivaling manually-tuned code and sometimes better
  - Approach can achieve performance portability, productivity and maintainability
- Implementation status
  - GMG optimizations integrated into CHiLL, tensor contraction uses existing CHiLL with additional frontend and Orio
  - CHiLL publicly available on github
  - Installed on Edison (user space)
  - Demonstrations this evening

THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE

Publications

- Papers
  - P. Basu, M. Hall, M. Khan, S.Maindola, S.Muralidharan, S.Ramalingam, A.Rivera, M.Shantharam, A.Venkat. Towards Making Autotuning Mainstream. International Journal of High Performance Computing Applications, 27(4), November 2013.
  - P. Basu, S. Williams, A. Venkat, B. Van Straalen, M. Hall, and L. Oliker. Compiler generation and autotuning of communication-avoiding operators for geometric multigrid. In High Performance Computing Conference (HIPC), 2013.
  - P. Basu, S. Williams, A. Venkat, B. Van Straalen, M. Hall, and L. Oliker. Compiler generation and autotuning of communication-avoiding operators for geometric multigrid. In Workshop on Optimizing Stencil Computations (WOSC), 2013.
  - Protonu Basu, Samuel Williams, Brian Van Straalen, Mary Hall, Leonid Oliker, Phillip Colella, "Compiler-Directed Transformation for Higher-Order Stencils", International Parallel and Distributed Processing Symposium (IPDPS), May 2015.
  - Thomas Nelson, Axel Rivera, Prasanna Balaprakash, Mary Hall, Paul D. Hovland, Elizabeth Jessup, Boyana Norris, "Generating Efficient Tensor Contractions for GPUs", International Conference on Parallel Processing (ICPP), September 2015.

- Thesis and Dissertations
  - Axel Rivera. Using Autotuning for Accelerating Tensor-Contraction on GPUs, Masters thesis, University of Utah, December 2014.
  - Protonu Basu, "Compiler Optimizations and Autotuning for Stencils and Geometric Multigrid", PhD Dissertation, University of Utah, December 2015.
  - Thomas Nelson, "DSLs and Search for Linear Algebra Performance Optimization," PhD Dissertation, University of Colorado, December 2015.

THE UNIVERSITY OF UTAH Argonne NATIONAL LABORATORY BERKELEY LAB X-TUNE