X-TUNE

X-Stack PI meeting

Mary Hall (Utah), ***Samuel Williams (LBNL****), Paul Hovland (ANL)*

Background

Programming Challenges

- Exascale machines (CPU or GPU) will be capable of running MPI+OpenMP.
  - Unfortunately, CPU and GPU OpenMP4 implementations are different (presence of target clauses).
  - (single source) **portability is not possible** without #ifdef's guarding CPU and GPU code (**not scalable**)
- Performance portability
  - single source delivering close to theoretical performance will likely be impossible tomorrow
  - worse, OMP4 may never deliver close to peak performance on GPU architectures
  - **Thus architecture-specific solutions (CUDA, OpenACC) may be required (not scalable)**
- As architecture-specific implementations will be required…
  - Rare to find programmers equally capable of programming CPUs and GPUs.
  - **Keeping CPU and GPU implementations in sync will be a challenge**

    = impediment to integrating novel physics

    = impediment to integrating novel algorithms

    = impediment to integrating novel optimizations

Will Vendors Solve This For Us?

- Vendors have developed compilers and programming models to maximize performance/productivity/generality on their respective platforms.

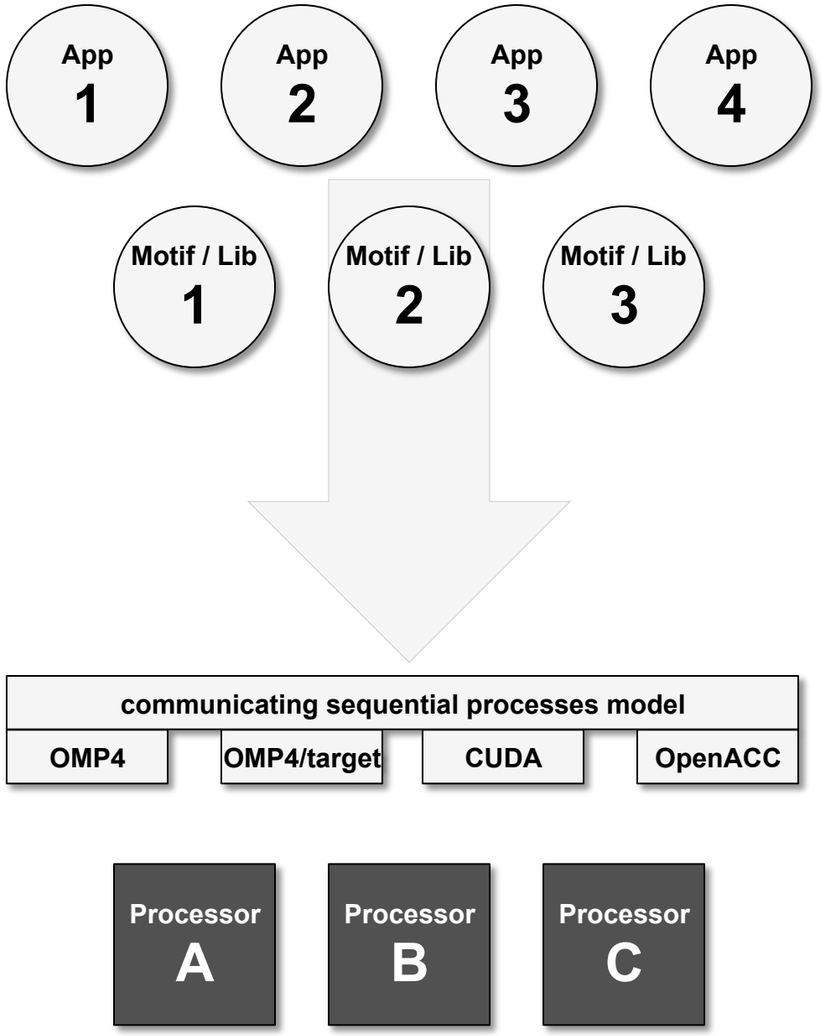
- Mandating vendors provide performance portability requires mandating a common programming model (i.e. portability)
  - requires one class of architecture giving up generality/productivity
  - requires the other class giving up specialization/performance/efficiency

- **Providing performance portability (to their competitors) is not in the financial interest of any vendor**
- DOE must solve this ourselves

ECP Vision

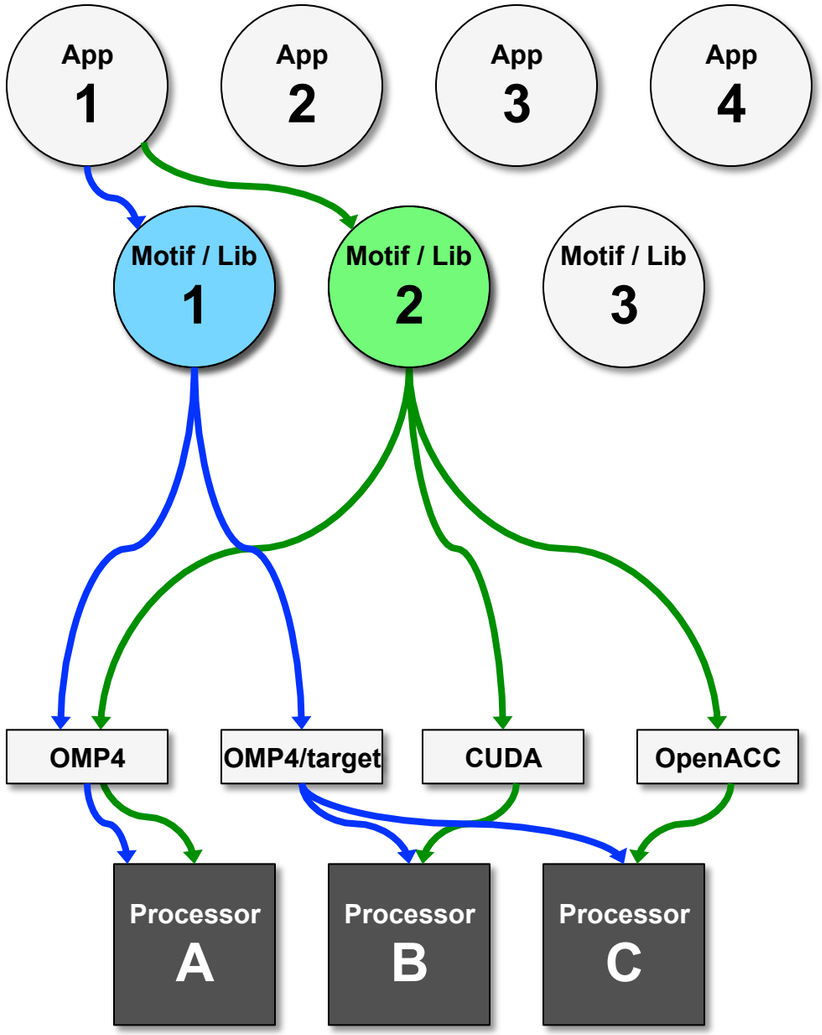
Tools Must Hide Complexity from Developers

- Ultimately, we want to map functionality expressed by applications to a range of target platforms without sacrificing…
  - portability
  - performance
  - generality/extensibility/maintainability
- In reality, we often think of apps in terms of "**motifs**" and libraries
  - often based on CSP model
  - increasingly require hybrid programming models to exploit architectures/mitigate scaling limitations



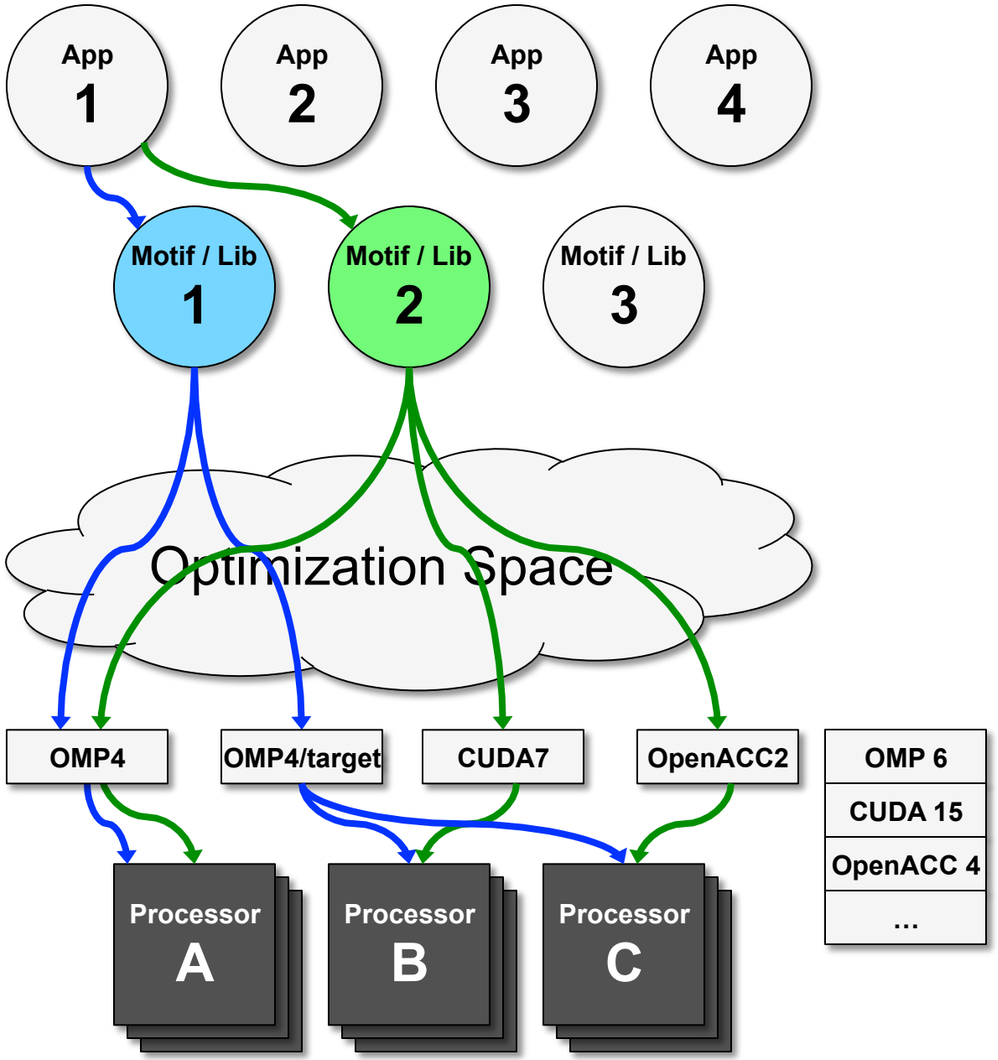
Tools Must Hide Complexity from Developers

- One could hope writing OpenMP4 would provide performance portability
- In reality, to use GPUs, one needs a different dialect of OpenMP4
  - ‘target’ clauses
  - **complexity has been exposed**
  - **(performance) portability has been impaired**
  - **maintainability/extensibility has been hurt**
- performance-sensitive motifs may maintain multiple implementations
  - required for portability
  - required for performance
  - possibly vendor-specific like CUDA
  - possibly architecture-specific (intrinsics)



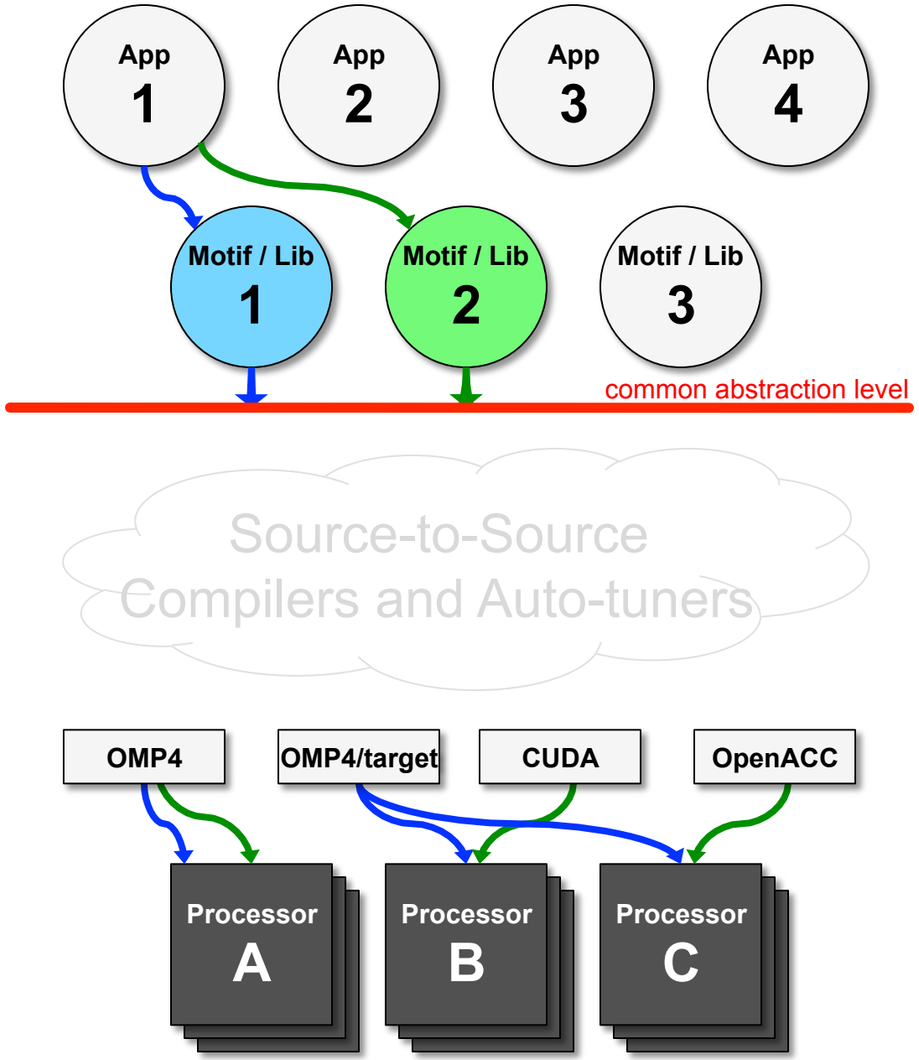
Tools Must Hide Complexity from Developers

- For each path there is a potentially huge optimization space between functional description and execution
  - architectures may require different optimizations
  - motifs/inputs may require different optimizations
- Worse, architectures continue to evolve (**optimizations useful today may not be optimal tomorrow**)
- Moreover, programming models continue to evolve (**code bloat required today may not be required tomorrow**)



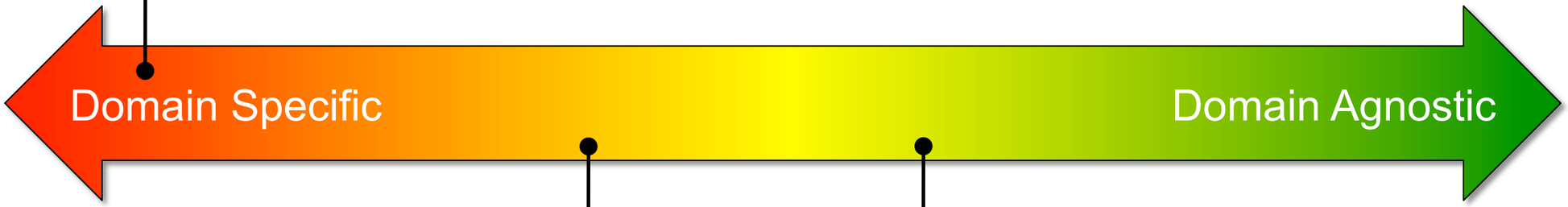
Tools Must Hide Complexity from Developers

- 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



Several Viable Approaches…

Unique DSL and DS-compiler for each motif
- **multiple targets** (OMP, CUDA, etc…)
- ability to leverage vendor compilers (source-to-source)
- **different compiler for each motif**
- **motif x target explosion (no backend reuse)**
- **difficult to compose transformations**



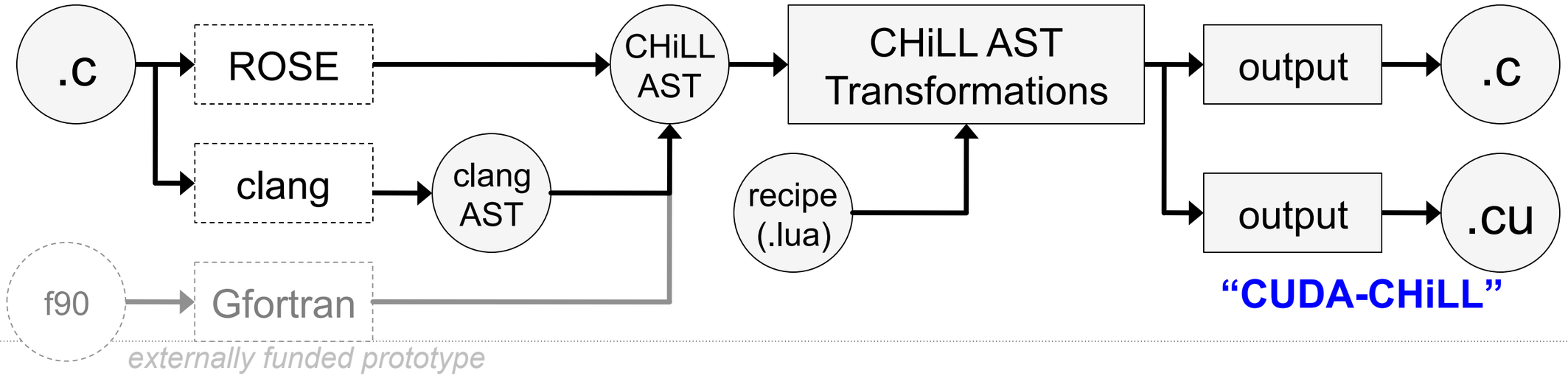
- Embedded DSL for each motif
- General purpose compiler augmented to:
  - parse each eDSL
  - perform **domain-specific transformations**
  - **auto-parallelization** (OMP, CUDA, etc…)
  - **AMM heuristics** rather than auto-tuning
  - leverage vendor compilers (source-to-source)

- Compiler Transformation Framework (**ability to compose complex transformations**)
- Augmented with:
  - **domain-inspired transformations/optimizations (requires user guidance/knowledge)**
  - **auto-parallelization** (OMP, CUDA, etc…)
  - **auto-tuning** rather than heuristics
  - ability to use vendor compilers (source-to-source)

X-TUNE

Background: CHiLL

- Source-to-Source Compiler/Transformation Framework developed at Utah
- Open Source (**GPL**)
- Maintained by **staff programmers** at Utah (contingent on funding)
- run on NERSC and Utah clusters (some attempts to create NERSC module)



externally funded prototype

‘Old School’ Auto-tuning

- Fixed functionality routines
- Code generator scripts
- Brute-force tuning search

Compiler-Driven Auto-tuning

- Arbitrary routines written in c
- Compiler parses c, generates AST, performs transformations on AST, outputs c
- User can specify additional transformations on AST
- Compiler produces multiple code variants (e.g. loop blockings)
- Intelligent search algorithms tune over code variants

X-TUNE Project

X-TUNE focused on…

- ✓ Optimizing Compilers (CHiLL)
- ✓ Vanilla C language
- ✓ Domain-in… (Stencils, GMG, …
- ✓ Hiding com… CPUs and G…
- ✓ Hiding on-nod… choices fr… the user
- ✓ Exposing knobs for compiler-ba… d, user-driven auto-tuning

X… UNE is NOT about…

- ✗ ~~lo… …uages~~
- ✗ ~~…ing models~~
- ✗ ~~…models~~
- ✗ ~~…d locality controls~~
- ✗ ~~…ilience~~
- ✗ ~~E… …rgy~~

Don't expose complexity to the programmer. Hide it!

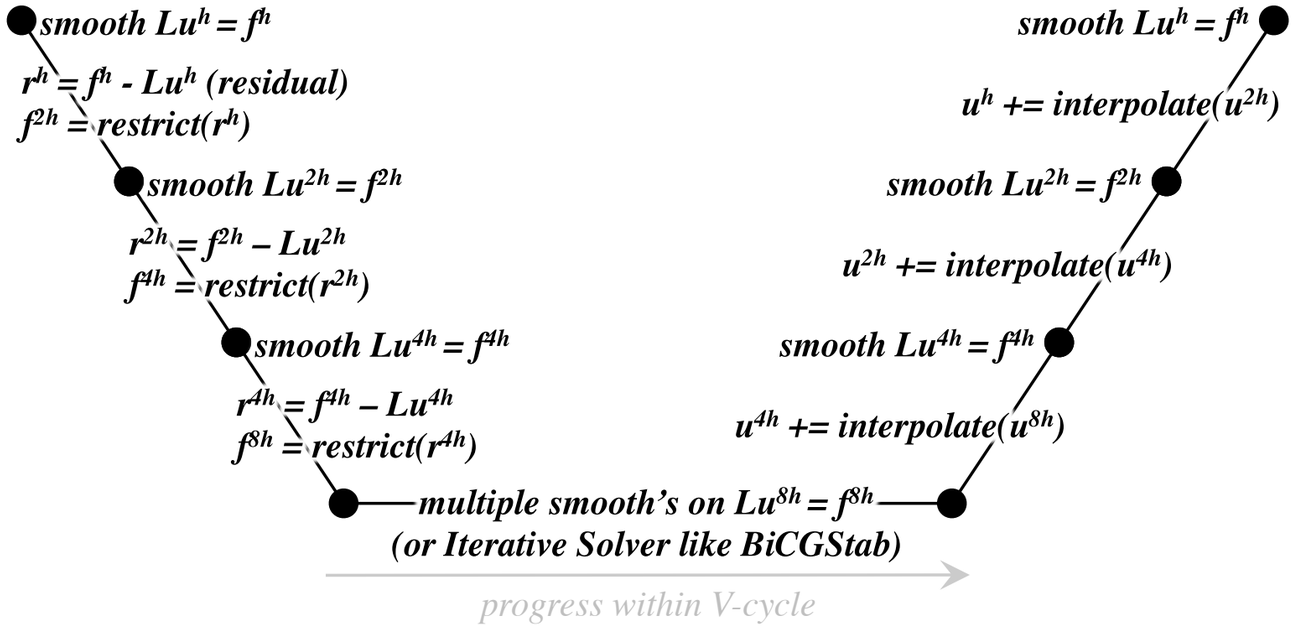
X-TUNE Hides Complexity from the Programmer

Use Case #1: Geometric Multigrid

- PDEs discretized onto a structured grid is a common theme in scientific computing

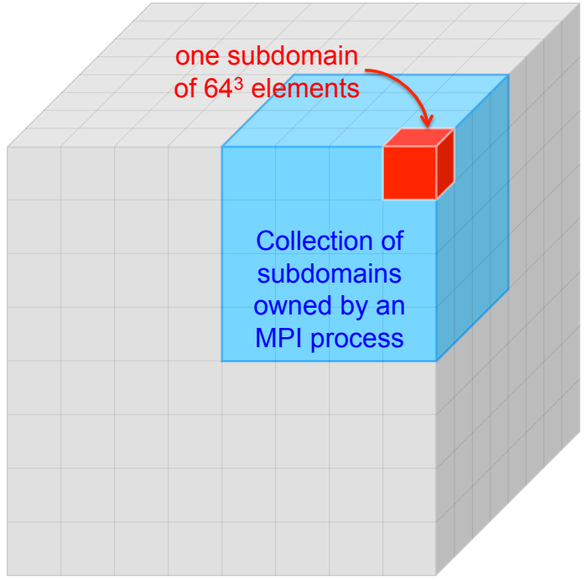
- Multigrid is a recursive technique for solving systems of linear equations
  - GMG is a specialization for structured grids (stencils)
  - **Stresses performance at different degrees of parallelism, locality, working set sizes, etc…**

    == hard/impossible to optimize every case by hand



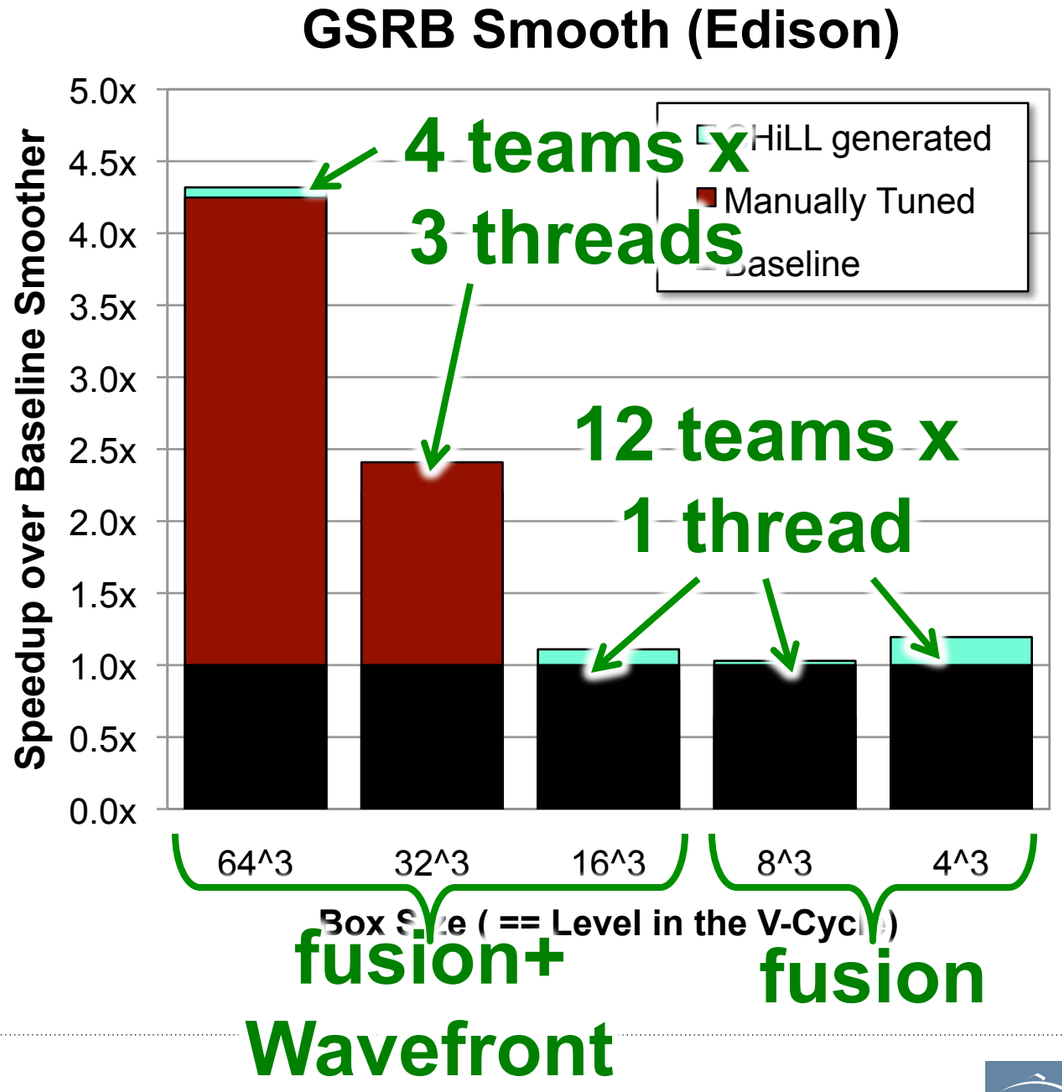
X-TUNE used the miniGMG Benchmark

- Proxies the GMG solves found in **CHOMBO** and **BoxLib** codes
- Distributed memory (MPI) implementation, **focused on on-node challenges…**
  - operator fusion
  - cache blocking
  - communication-avoiding smoothers and Krylov methods
  - high-order operators
- Previous work developed hand-optimized implementations
  - MPI+OpenMP including communication-avoiding implementations
  - MPI+CUDA (NVIDIA GPUs)
  - X-Tune Research: tools that productively deliver comparable performance
- n.b., miniGMG is the predecessor of the ExaCT CoDesign's HPGMG-FV Proxy App



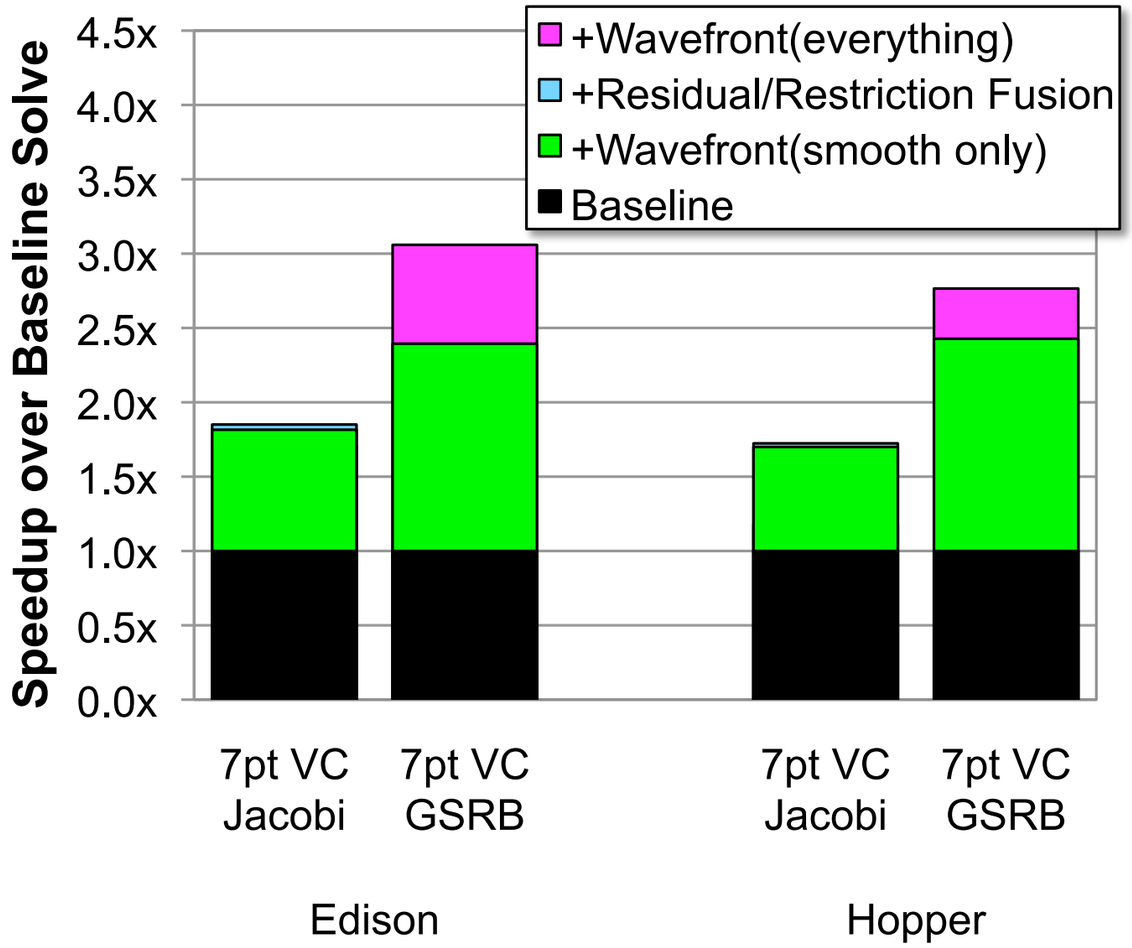
1. CHiLL Productively Delivers Performance

- miniGMG w/CHiLL…
  - fused operations
  - created a communication-avoiding wavefront
  - **auto-parallelized (OpenMP)**
- **auto-tuned** to find the best implementation for each box size…
  - wavefront depth (degree of comm. avoiding)
  - nested OpenMP configuration
  - inter-thread synchronization (barrier vs. P2P)
- For fine grids (large arrays) CHiLL attains nearly a **4.5x speedup** over the baseline and was **faster than hand-optimized.**



2. CHiLL went one step further

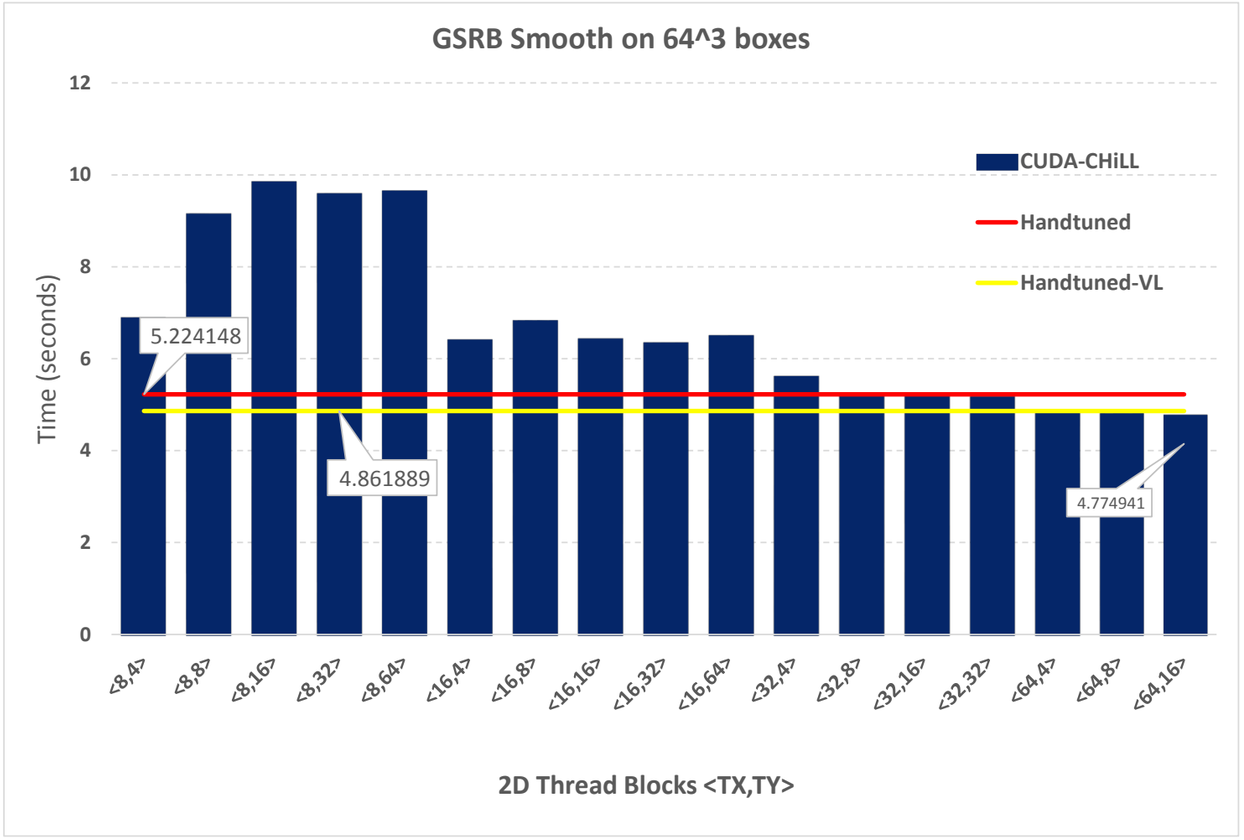
- CHiLL fused the residual and restriction operations into the wavefront as well.
  - read u^h, R^h, and coefficients once
  - perform 4 smooths (**no additional DRAM data movement**)
  - write smoothed u^h and new R^{2h}
  - **CHiLL exploits excess compute capacity**
- CHiLL can just as quickly apply these transformations to a different smoother and auto-tune it…
  - up to a **3x improvement in MGSolve** time
  - Jacobi suffered from more frequent inter-thread synchronization within the wavefront



3. CHiLL Hides Architectural Complexity

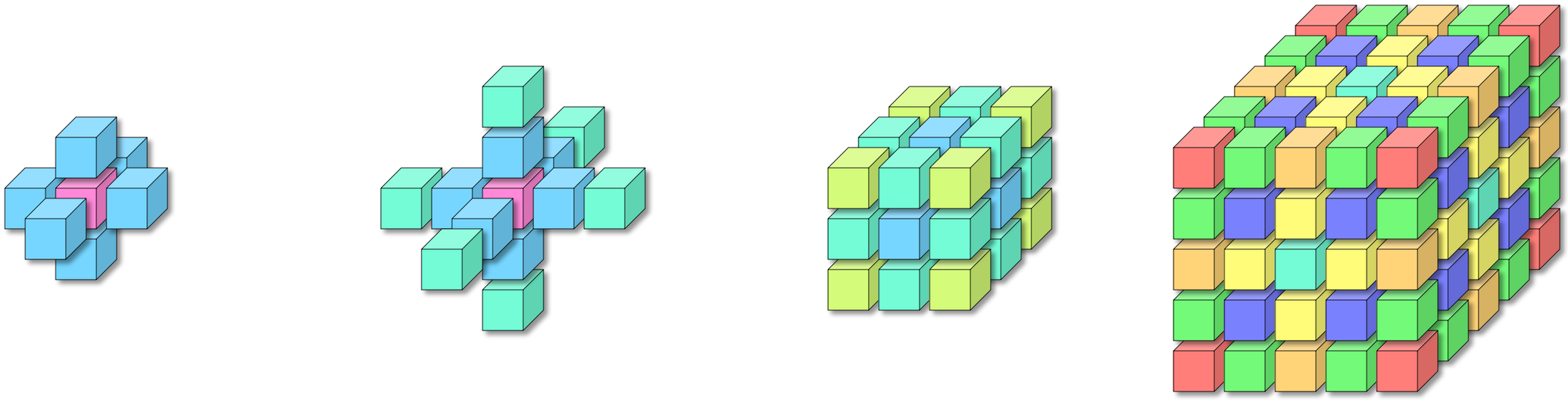
- CHiLL can obviate the need for architecture-specific programming models like CUDA
  - CUDA-CHiLL took the sequential GSRB implementation (.c) and **generated CUDA** that runs on NVIDIA GPUs
  - CUDA-CHiLL tunes for the current target machine whereas static implementations hand-optimize for yesterday's GPUs

    == avoids code rot
  - CUDA-CHiLL auto-tuned over the thread block sizes and is ultimately **2% faster** than the hand-optimized minigmg-cuda



4. CHiLL Enables Extensibility

- Applied mathematicians have a penchant for changing the math
- 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

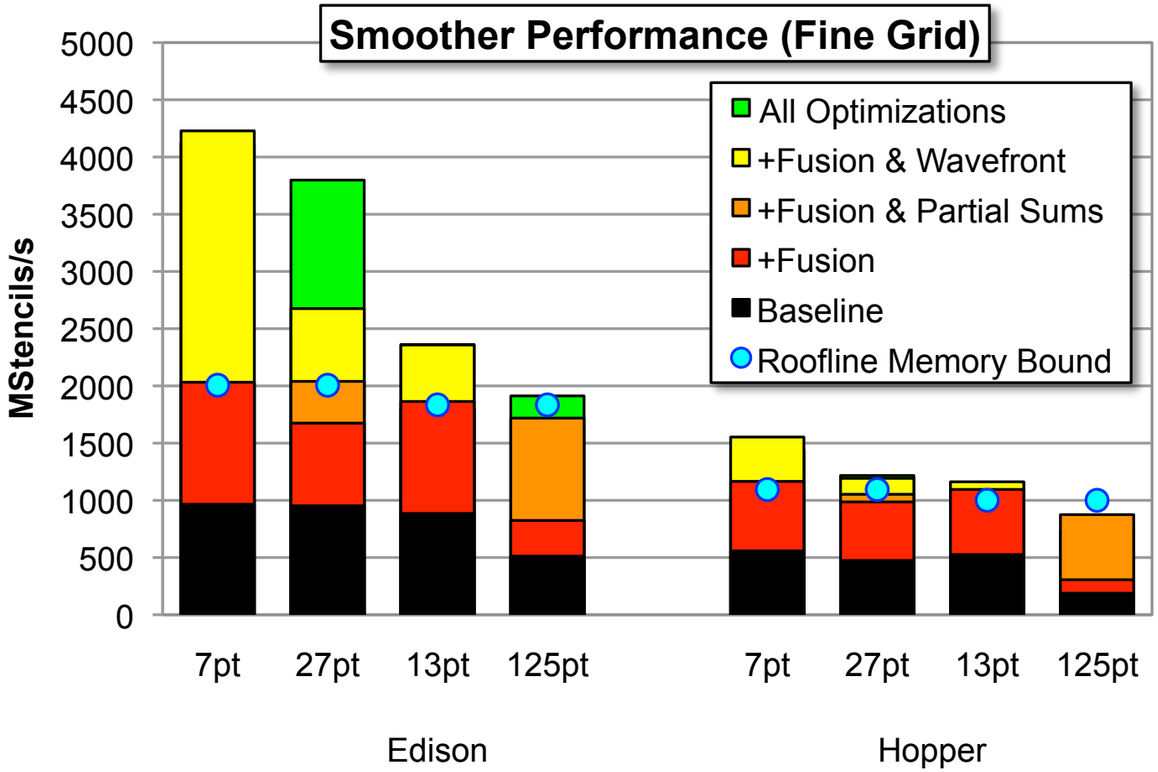


4. CHiLL Enables Extensibility

- CHiLL optimized/tuned each of these 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**.

    == productive & performance portability
  - Using a wavefront, CHiLL can nearly **double** the nominal Roofline performance for the 7- and 27-point operators.



X-TUNE Developed Efficient Search Algorithms

Use Case #2: Tensor Contractions

- Found in Chemistry, Spectral Element (and Finite Element) codes
- Consider example from 3D spectral element codes…

$$V_{ijk} = A^{l}_{k} B^{m}_{j} C^{n}_{i} U_{lmn} = \sum_{l=0}^{p} \sum_{m=0}^{p} \sum_{n=0}^{p} A^{l}_{k} B^{m}_{j} C^{n}_{i} U_{lmn}$$

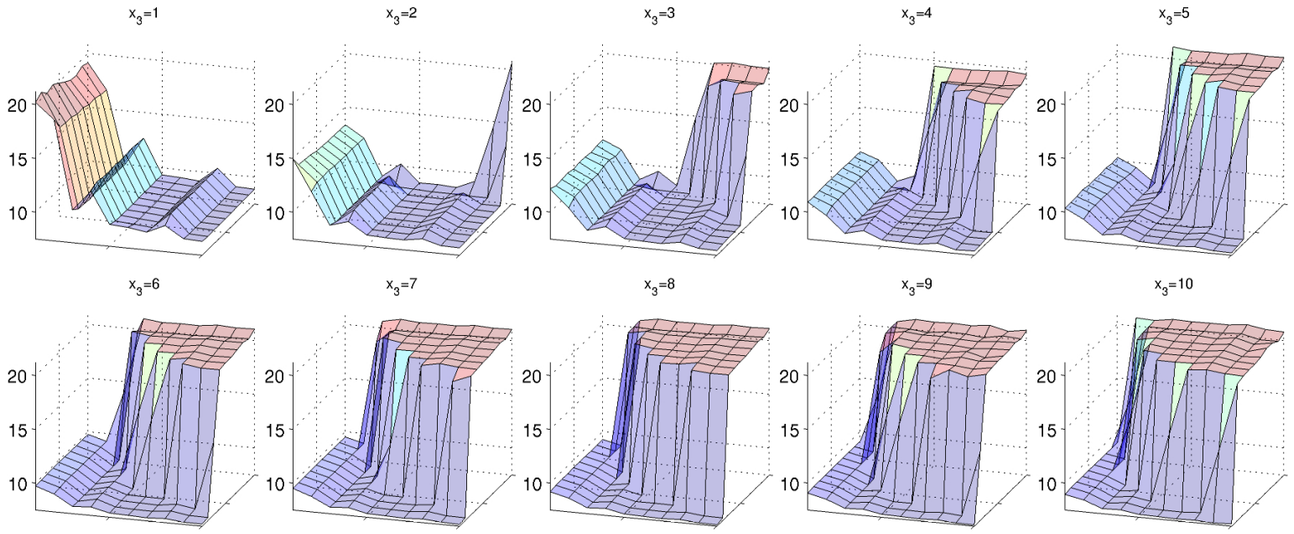
  - Naïvely, this is a 6-deep loop nest (each ijk has a summation over lmn)
  - Optimizations exploit symmetry/common subexpressions
- Challenges…
  - What is the optimal contraction order? … $O(p^6)$ -> $O(p^4)$
  - What is the optimal loop order? (N! different implementations)
  - Search space is discontinuous, noisy, and expensive to evaluate

    == intractable brute force search space

    == need intelligent search algorithms

SURF: Model-Based Search

- Search Using **Random Forests**
  - state-of-the-art statistical ML 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…
- Prototyped as module in ORIO



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

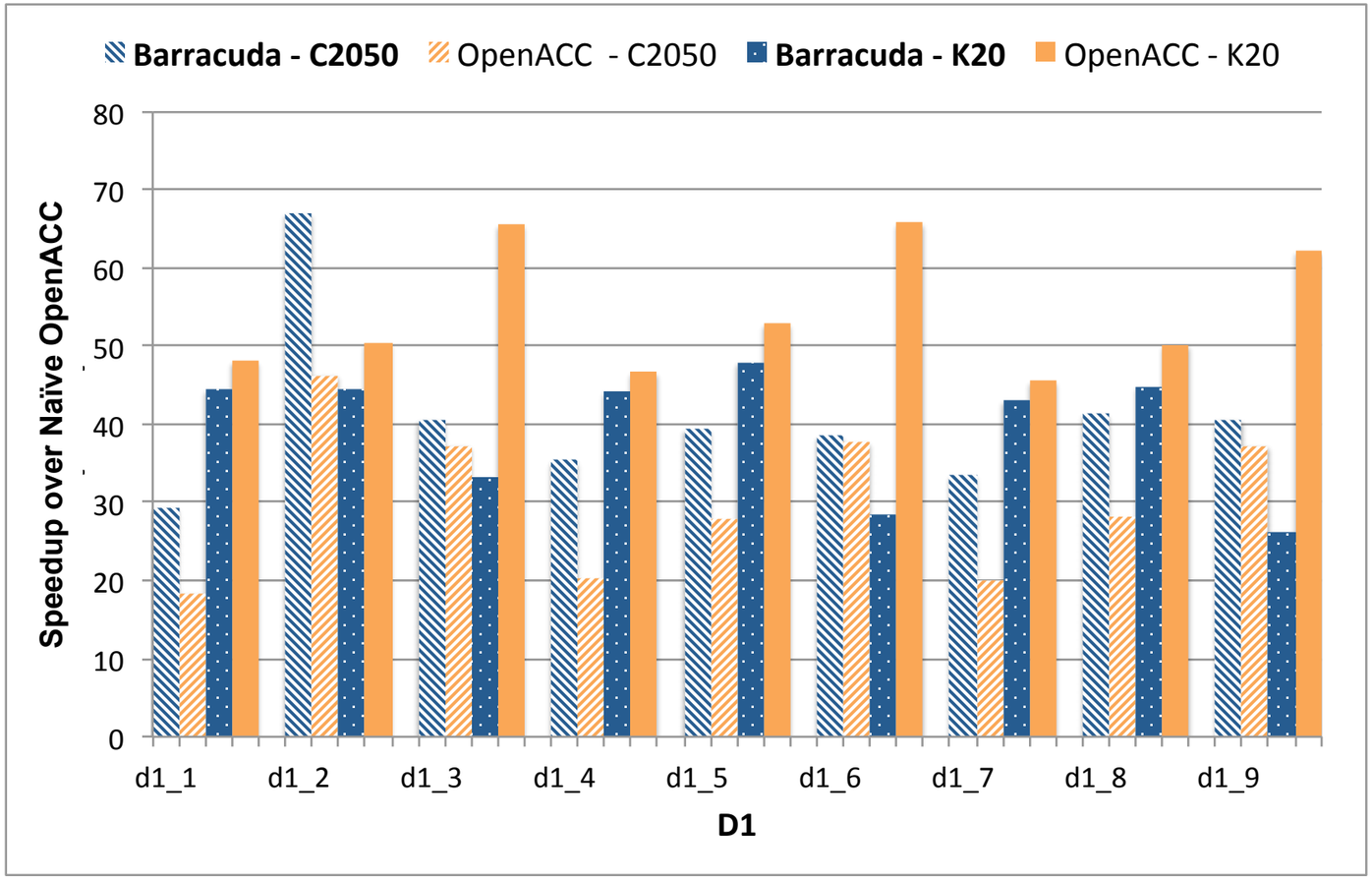
Baracuda: X-TUNE Demonstration Prototype

Baracuda Framework

- Framework for compiler- and model- driven auto-tuning of Tensor Contractions
  - Octopi: High-level domain-specific frontend; interfaces with TCR
  - TCR: Low-level mathematical interface to generated C, CUDA-CHiLL, SURF search tool
  - CUDA-CHiLL: Parses .c, optimizes, generates high-performance CUDA.
  - SURF: Search Using Random Forests module added to ORIO
- Baracuda conducts a model-driven search of the transformations possible for tensor contractions generating and evaluating each CUDA implementation.
- For research purposes….
  - Automatically generated CUDA was manually modified into OpenACC
  - ‘Naïve OpenACC’ simply expressed parallelism and gauges the vendor compiler’s ability to productively generate high-performance code
  - ‘OpenACC’ includes expert user micromanagement of parallelism

Optimizing NWChem with Baracuda

- 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
  - OpenACC = naïve + manual explicit control over hierarchical parallelism



Optimizing NEKBone with Barracuda

- Extracted representative tenor contractions from NekBone (CESAR CoDesign Center Proxy App)
  - 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 HSW core.

	Naïve OpenACC	Optimized OpenACC	Baracuda (CUDA)
K20	2.86	12.39	36.47
C2050	1.18	19.21	34.65

Potential ECP R&D

Observations

- X-TUNE prototyped compiler transformation technologies that…
  - **hide the complexity** of targeting emerging architectures without exposing complexity to the programmers
  - are **extensible** as novel optimizations or programming models emerge
  - are **complimentary to eDSL** solutions

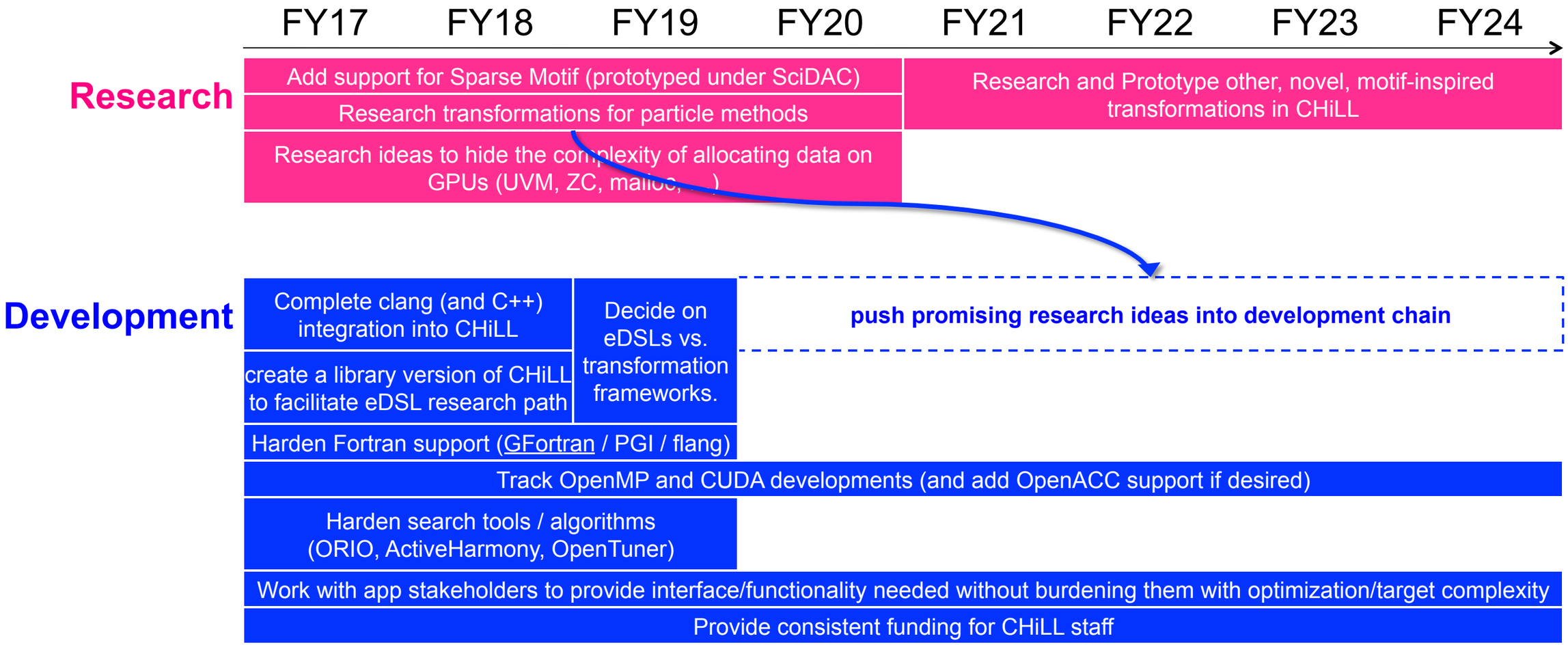
- X-TUNE developed new ML based search techniques to efficiently autotune in the presence of huge optimization spaces

- Funding for application integration/evaluation (CHOMBO) was cut from the original proposal. As such, **consensus on app-facing interface was not reached**.

Issues / Decision Points

- **NDA's with universities?**
  - many SW projects require university collaborations
  - FF, DF (, CORAL?, APEX?) NDA's bar universities
- Consensus on **vendor compilers vs. S2S w/auto-parallelization**
  - eDSLs vs. compiler transformation frameworks with domain-inspired optimizations?
  - What functional interface should we export to motif/framework/lib developers?
  - What tuning interface should we export?
- What is the target PM for accelerated systems?
  - OpenMP 4.1+, OpenACC 3.0+, or CUDA 7+?
  - **How do we easily switch between host parallelism and device parallelism?**
  - Do we have to compose multiple models (or dialects)?
  - (if CUDA) how do we **hide the complexity of data allocation** (UVM, ZC, malloc, …) from users?

ECP R&D Timeline



Other Issues

- Ensure hybrid MPI+x codes attain MPI bandwidth comparable to flat MPI codes?
- Ensure no MPI routine has a **worse case** computational complexity no worse than O(PlogP)
- Latency/Overhead of vendor threading runtimes
  - KNC OMP overheads can be horrendous (10s of us) compared to IVB
  - CUDA launch times are also O(10us)
  - High overheads demand **exponentially increasing work** per parallel region in the future
- Exposing performance counters (e.g. data movement) to users (not just root)
  - **Without trusted, user-accessible counter data, most performance analysis won't work**
- How do we avoid the network wall tomorrow?
  - Today, for some codes, poor code optimization can hide behind the memory wall.
  - Tomorrow, will on-node optimization matter if we are bottlenecked by the network?

Acknowledgements

- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

Publications / Software

- CHiLL v0.2: http://ctop.cs.utah.edu/ctop/?page_id=21
- Protonu Basu, “Compiler Optimizations and Autotuning for Stencils and Geometric Multigrid”, PhD Thesis, University of Utah, December 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.
- 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.
- Protonu Basu, Samuel Williams, Brian Van Straalen, Leonid Oliker, Mary Hall, "Converting Stencils to Accumulations for Communication-Avoiding Optimization in Geometric Multigrid", Workshop on Stencil Computations (WOSC), October 2014.
- Protonu Basu, Anand Venkat, Mary Hall, Samuel Williams, Brian Van Straalen, Leonid Oliker, "Compiler generation and autotuning of communication-avoiding operators for geometric multigrid", 20th International Conference on High Performance Computing (HiPC), December 2013, 452—461.

Questions and Discussion

Questions

- The goals of your project and its current status
  - Do you release your software as open source? **OSS/GPL**
  - Do you have DOE/NNSA users of your software? **no users depend on CHiLL. We used proxies and apps and demonstrated value of CHiLL**
  - Have facilities, vendors, or ISVs picked up your software? **Users have installed it on NERSC. Attempts to create NERSC module stalled**
  - What is the support model for your software? **Utah staff programmers funded by a variety of sources. Research contributions funded thru ASCR.**
  - Are there any applications in particular that the outcomes of your project are targeting? **CHOMBO/BoxLib BSAMR codes, Nek5K, NWChem.**

- Will the developed software technologies be mature enough to be part of the software stack on exascale systems expected to be selected in 2019 and installed in 2023?

  Transformation technologies are mature; frontend choices are maturing (F/C++); integration model (including interface) requires consensus with application stakeholders

- How would the proposed activities build on the research you have been carrying out with ASCR Research funding?

  Provide broader language support (C,C++,Fortran); integration model; target execution support; domain-inspired transformations

Questions

- What do you feel are the key challenges posed and opportunities offered by exascale systems for your specific area?

  See slide 33
- What is the R&D that you would like to carry out within the ECP?

  See slide 33
- What research remains for your project's outcomes to benefit key DOE applications?

  See slide 33
- What are the proposed activities that you believe would contribute to the ECP?

  See slide 33
- Your roadmap/timeline for maturing the software technologies and deploying them on exascale platforms, with a few intermediate milestones or decision points (forks in the roadmap). The timeline is of particular importance in selecting what the ECP will include in the development plans.

  See slide 33

Backup Slides

Adaptive Mesh Refinement (AMR)

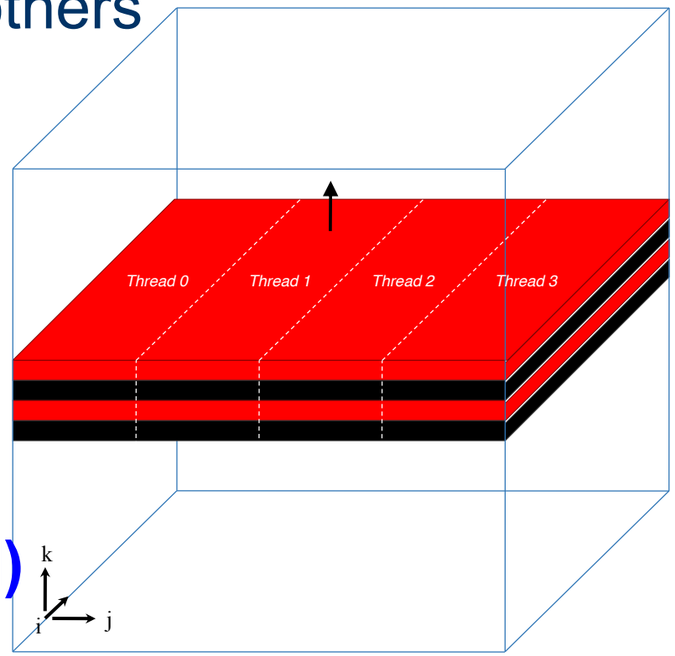
- Acceleration and memory savings technique that creates hierarchy of grids of different grid spacings (plus subcycling)
- **CHOMBO** and **BoxLib** are frameworks for block structured AMR
- Both use GMG solvers applied to AMR levels

GSRB Smoothers

- Gauss-Seidel Red Black is a go to smoother for geometric multigrid
- Complex update pattern is an impediment for most compilers…
  - iteration space is the union of four stride-2 rectangular domains
  - many research projects concentrate on Jacobi and Chebyshev-like smoothers
- miniGMG uses the 2nd order variable-coefficient Helmholtz ($\alpha u - \nabla \bullet \beta \nabla u$)
  - 7-point stencil with 6 weights (3 unique) plus an extra diagonal term
  - variable RHS, periodic BC's
- With sufficient thread parallelism, smoother is heavily memory-bandwidth bound.
  - Previous work prototyped communication-avoiding smoothers
  - Exchange deeper ghost zones, wavefront GSRB on each box
  - **Can we modify CHiLL to automatically generate a threaded wavefront ?**

GSRB Wavefronts

- With sufficient thread parallelism, smoother is heavily memory-bandwidth bound
  - operator is created piecemeal (laplacian, diagonal term, smoother)
    == multiple passes thru memory for each smooth.
- Previous work prototyped communication-avoiding smoothers
  - fuse operator and smoother loops
  - Exchange deeper ghost zones
  - Create a 2D wavefront that sweeps thru each box
    updating a red or black points on a plane
    == perform 4 updates but only read/write the data once
- **Can we modify CHiLL to automatically fuse loops and generate a OpenMP-threaded (and synchronized) wavefront from sequential code?**

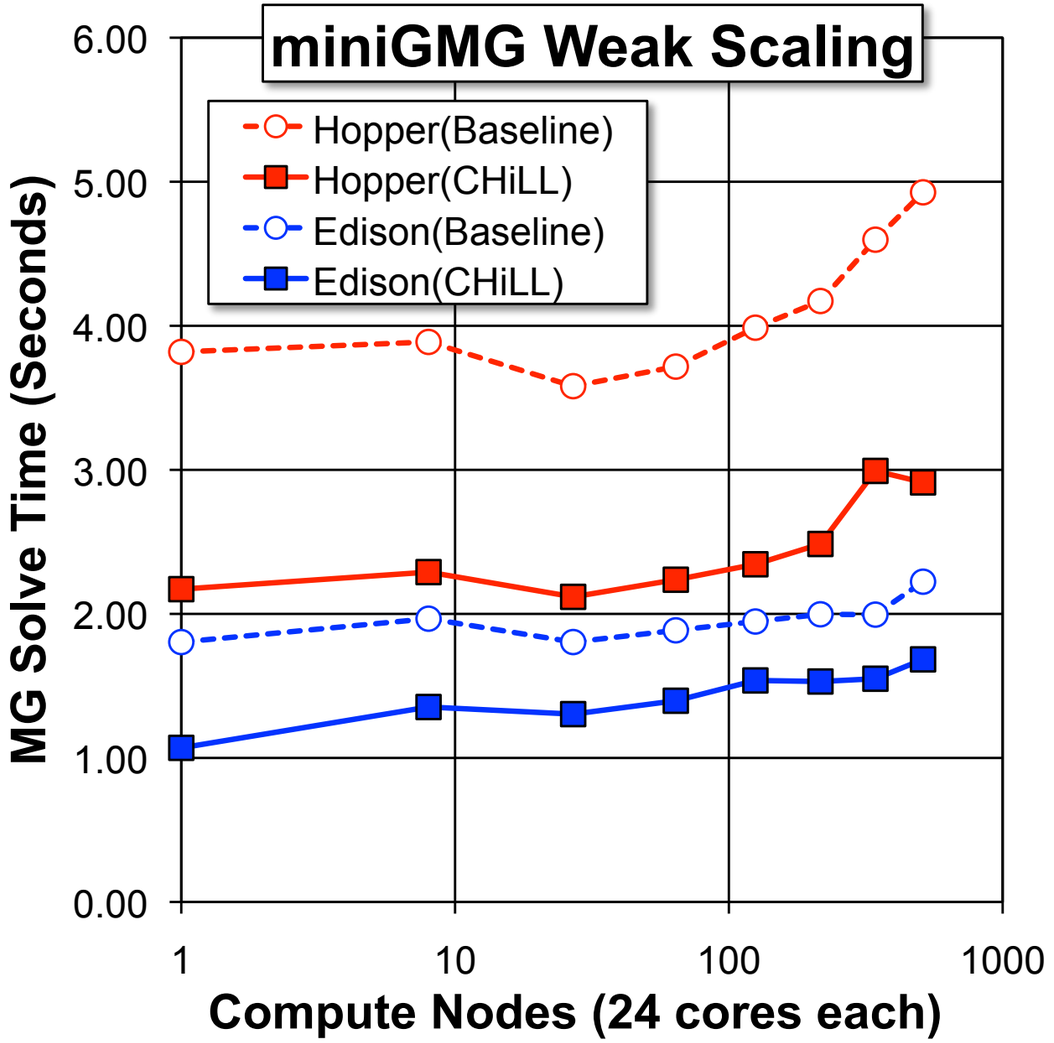


Performance with Scalability

GSRB smoother

Hopper and Edison

Baseline vs. CHiLL

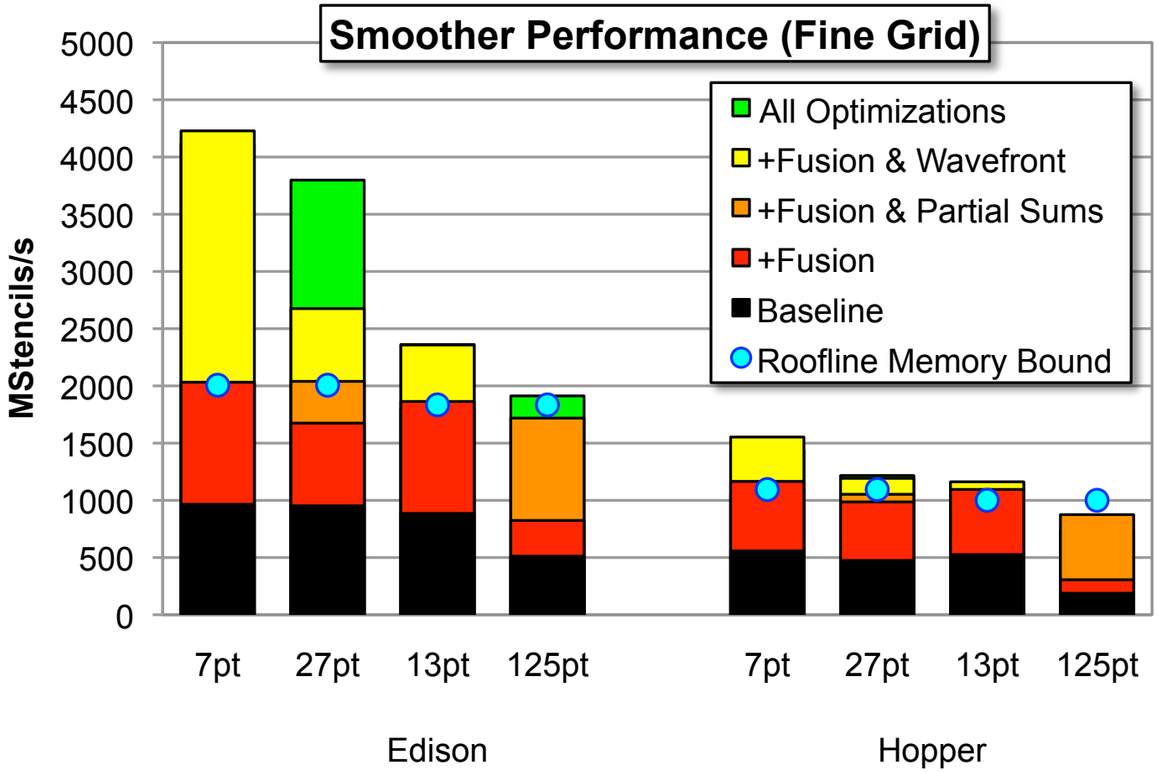


Smoother Performance (finest MG level)

- Without a communication-avoiding wavefront, CHiLL was able to deliver performance **near the Roofline limit** across both platforms and across all stencils

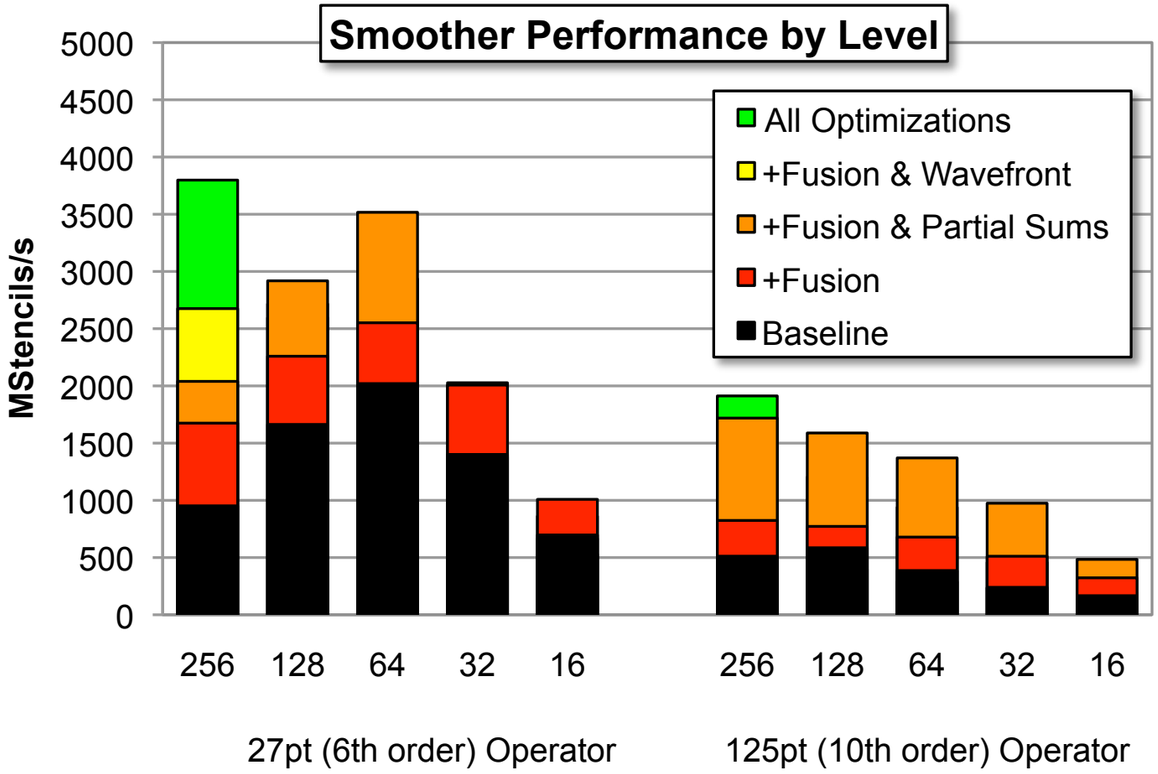
  == productive performance portability

- Using a wavefront, CHiLL can nearly double the nominal Roofline performance for the 7- and 27-point operators.



Smoother Auto-tuning

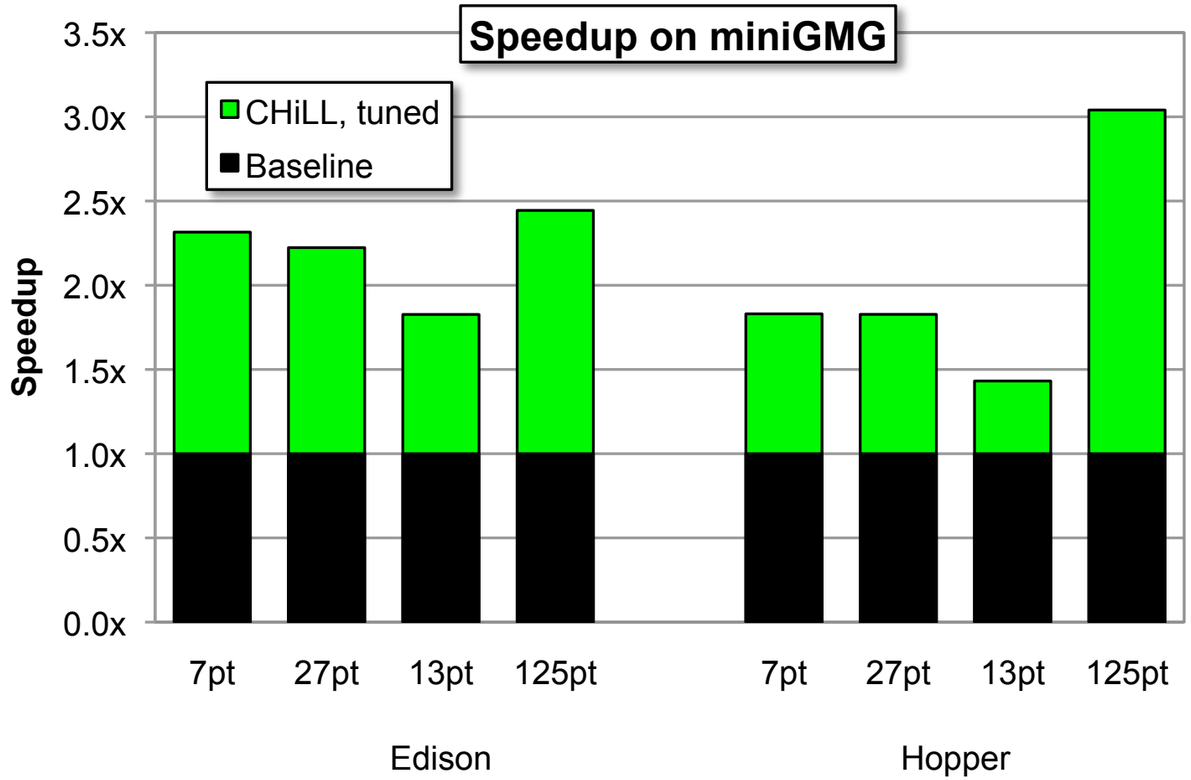
- Unlike a human, **CHiLL will tune its implementation for each level in the MG V-Cycle**
- As shown, the optimal configuration varies with MG level (256^3… 16^3)
- CHiLL delivers good performance on the first few levels (those that dominate performance) but struggles on the latency-limited levels



Overall MG Solver Speedup

- CHiLL attained around a **2x speedup** over the baseline implementation.
- Speedup is greater on platforms with excess compute capacity that can be exploited by a compiler via communication-avoiding optimizations

  == motivation for integrating compiler efforts with HW/SW co-design efforts



SURF: Model-based search