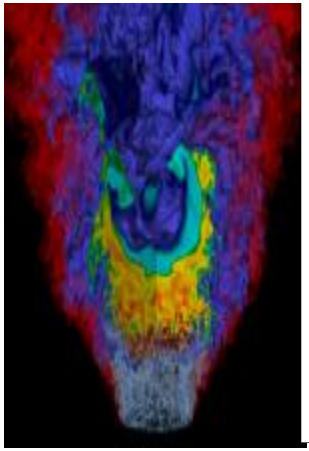
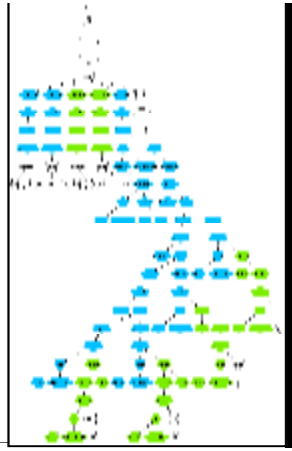
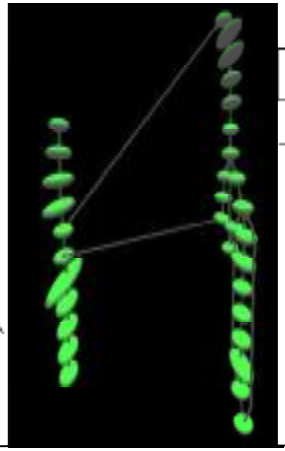
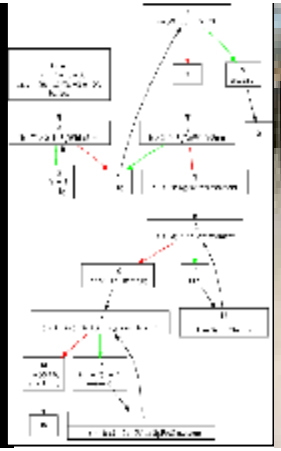


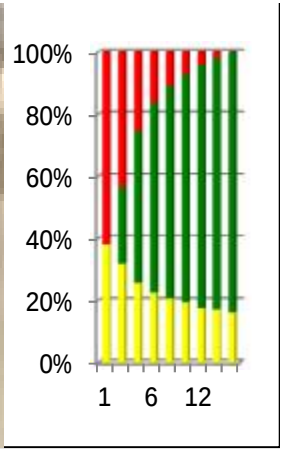
```
int aFunction(int a, int b)
{
        int c=b;
        return a;
}

main()
{
  int a,b,c,d,e;
  int i=4;
  for (i=0;i<10;i++)
  {
    int j=55;
    c=i+j;
    c=aFunction(i,c);
    a=aFunction(a+1,b);
  }
  #pragma SliceTarget
  a;
  return 0;
}
```









2012 X-Stack: Programming Challenges, Runtime Systems, and Tools - LAB 12-619

DSL Technology for Exascale Computing (D-TEC)

Lead PI and DOE lab:
Daniel J. Quinlan
Lawrence Livermore National Laboratory

Co-PIs and Institutions
Saman Amarasinghe, Armando Solar-Lezama, Adam Chlipala, Srinivas Devadas,
Una-May O'Reilly, Nir Shavit, Youssef Marzouk @ Massachusetts Institute of Technology
John Mellor-Crummey & Vivek Sarkar @ Rice University
Vijay Saraswat & David Grove @ IBM Watson
P. Sadayappan & Atanas Rountev @ Ohio State University
Ras Bodik @ University of California at Berkeley
Craig Rasmussen @ University of Oregon
Phil Colella @ Lawrence Berkeley National Laboratory
Rich Vuduc @ Georgia Tech
Scott Baden @ University of California at San Diego

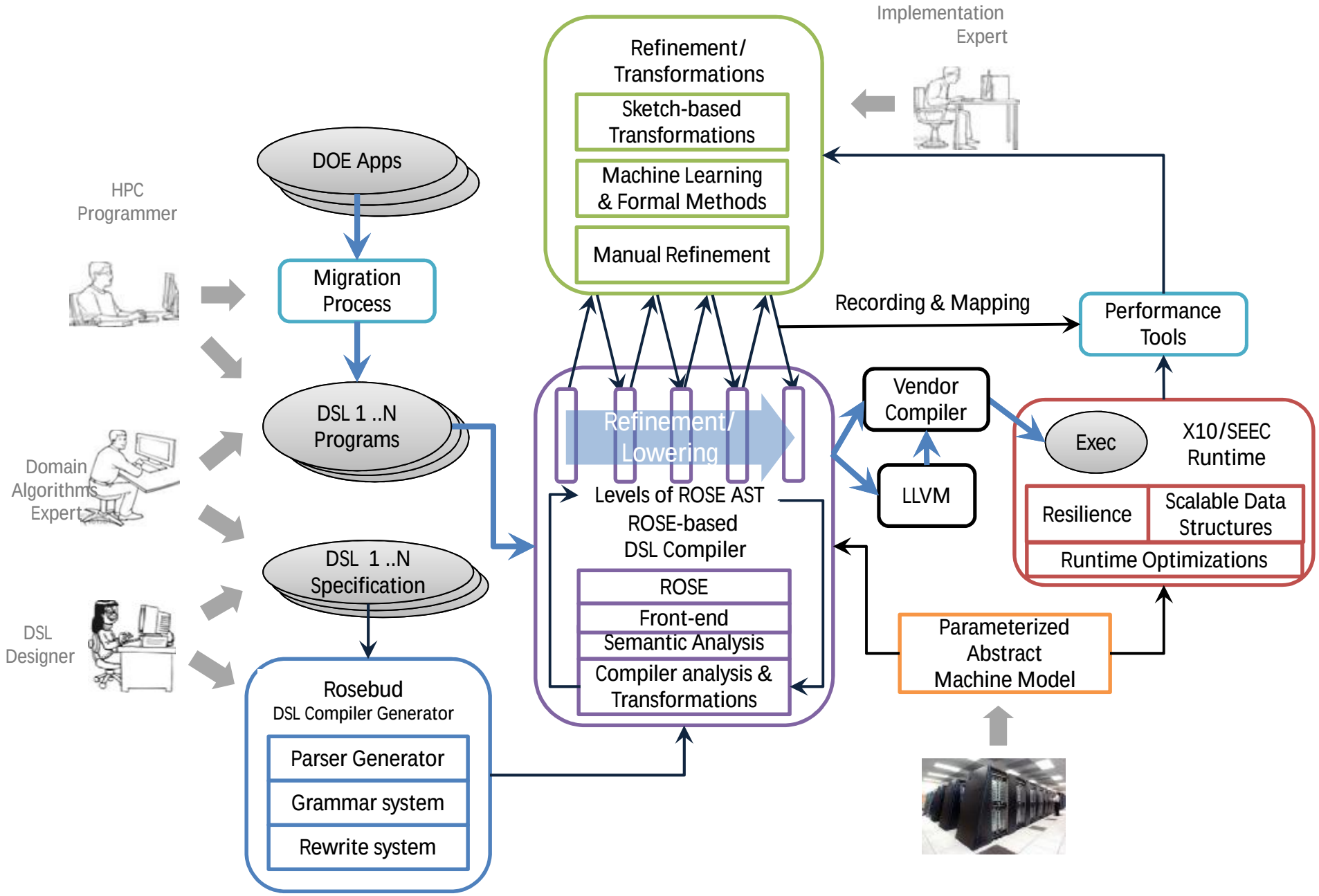
DSLs are a Transformational Technology

Domain Specific Languages capture expert knowledge about application domains. For the domain scientist, the DSL provides a view of the high-level programming model. The *DSL compiler* captures expert knowledge about how to map high-level abstractions to different architectures. The DSL compiler's analysis and transformations are complemented by the general compiler analysis and transformations shared by general purpose languages.

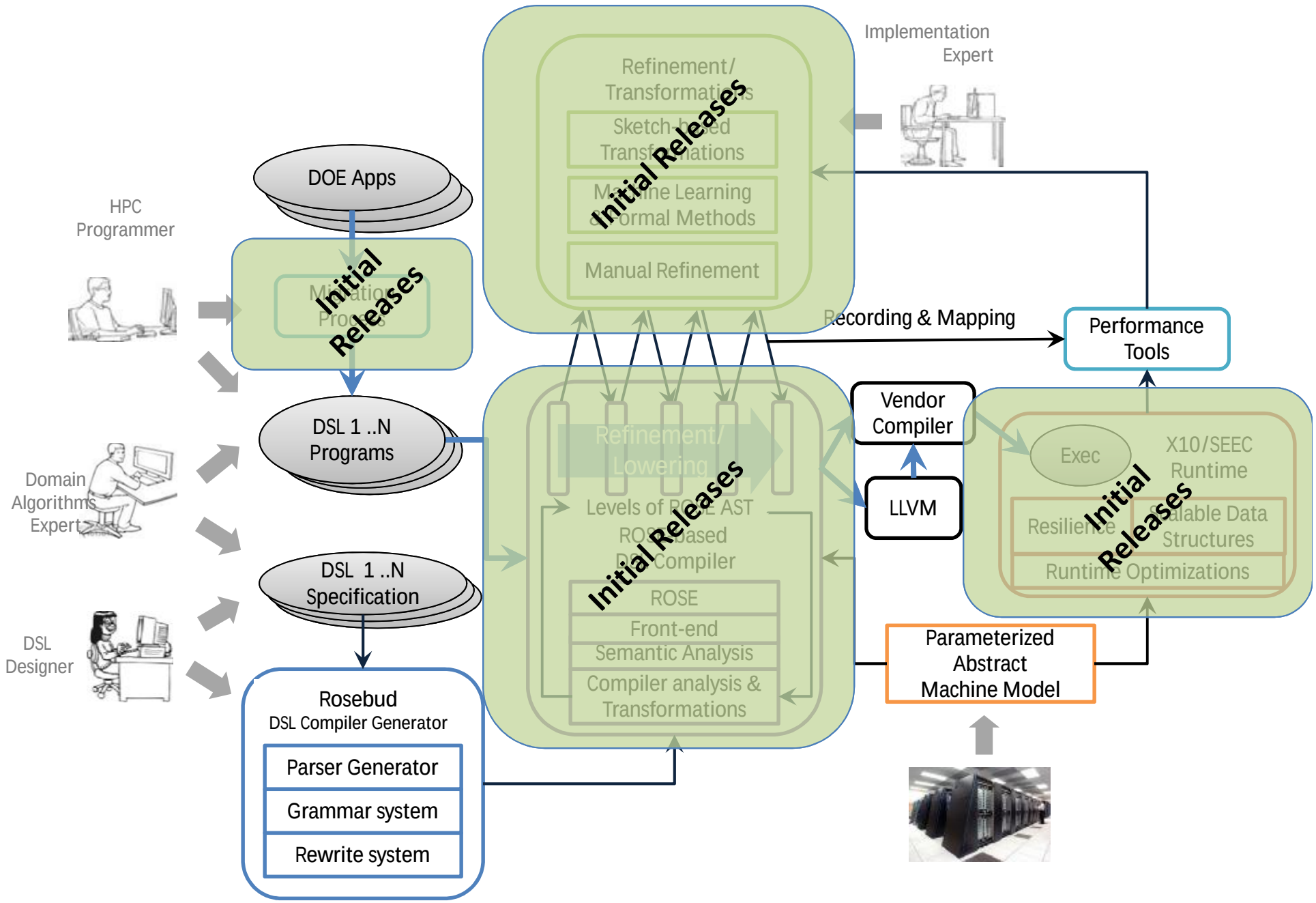
- There are different types of DSLs:
  - Embedded DSLs: Have custom compiler support for high level abstractions defined in a host language (abstractions defined via a library, for example)
  - General DSLs (syntax extended): Have their own syntax and grammar; can be full languages, but defined to address a narrowly defined domain
- DSL design is a responsibility shared between application domain and algorithm scientists
- Extraction of abstractions requires significant application and algorithm expertise
- We have an application team at 7.5% of the total funding
  - provide expertise that will ground our DSL research
  - ensure its relevance to DOE & enable impact by the end of three years

- *Saman and Dan have decided to merge proposals to provide the strongest possible proposal specific to DSLs; the merged effort will be led by Dan at LLNL*

The D-TEC approach addresses the full Exascale workflow



D-TEC Status


Implementation Expert
Refinement/ Transformations
Sketch-based Transformations
Machine Learning & Formal Methods
Manual Refinement
Initial Releases
DOE Apps
HPC Programmer
Initial Releases
DSL 1 ..N Programs
Recording & Mapping
Performance Tools
Vendor Compiler
LLVM
Refinement/ Lowering
Levels of ROSE AST
ROSE-based DSL Compiler
ROSE
Front-end
Semantic Analysis
Compiler analysis & Transformations
Initial Releases
Exec
X10/SEEC Runtime
Resilience
Scalable Data Structures
Runtime Optimizations
Initial Releases
Domain Algorithms Expert
DSL 1 ..N Specification
DSL Designer
Parameterized Abstract Machine Model
Rosebud
DSL Compiler Generator
Parser Generator
Grammar system
Rewrite system

CNS Miniapp

- An example app in BoxLib (block-structured AMR library) and a proxy app for ExaCT.
- Compressible Navier Stokes equations with constant viscosity and thermal conductivity.

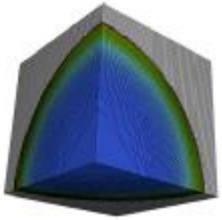
$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{1}$$

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) + \nabla p = \nabla \cdot \boldsymbol{\tau}, \tag{2}$$

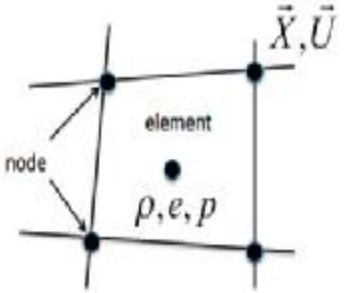
$$\frac{\partial \rho E}{\partial t} + \nabla \cdot [(\rho E + p)\boldsymbol{u}] = \nabla \cdot (\lambda \nabla T) + \nabla \cdot (\boldsymbol{\tau} \cdot \boldsymbol{u}), \tag{3}$$

- Three main computation-intensive functions:
  - ctoprim: Computing Q (vector with components: p, u, v, w, p, T) with given U (vector with components p, pu, pv, pw, pE).
  - hypterm: Updating U according to the left-hand side of equations 1~3.
  - diffterm: Computing the right-hand side of equations 1~3.
- 3rd order Runge-Kutta scheme is used for time advancing.

LULESH Miniapp

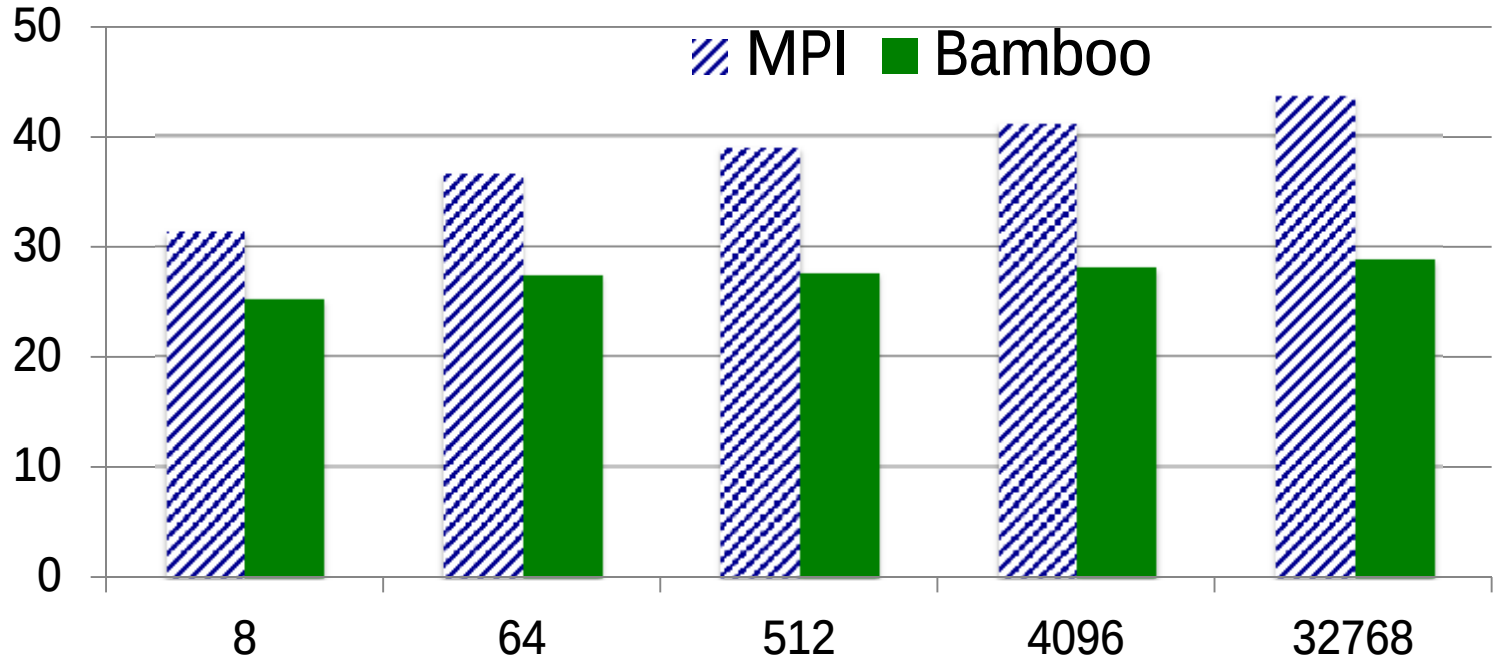


- Livermore Unstructured Lagrange Explicit Shock Hydrodynamics
  - Developed as a proxy application at LLNL under DARPA UHPC
  - Representing computations and algorithms in LLNL-based ALE3D code
- Solving one octant of the spherical Sedov problem (blast wave) using Lagrangian hydrodynamics for a single material.
- Equations are solved using a staggered mesh approximation.
  - Thermodynamic variables are approximated as piece-wise constant functions within each element.
  - Kinematic variables are defined at the element nodes.

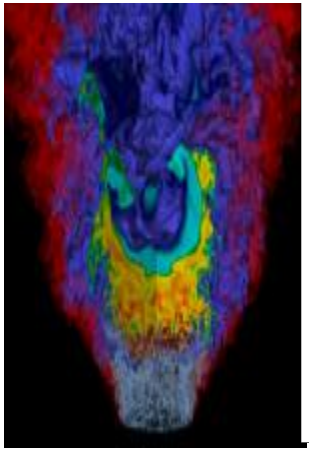


- Lagrange Leapfrog Algorithm advances solution involving 3 parts
  - Advance node quantities
  - Advance element quantities
  - Calculate time constraints

Lulesh: Compiler transformations can overlap communication and computation for improved performance

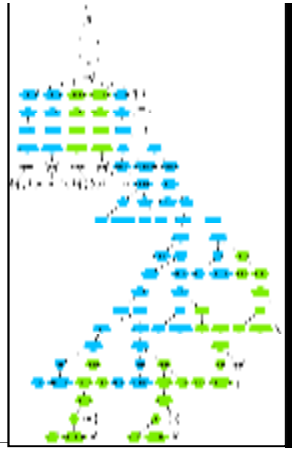


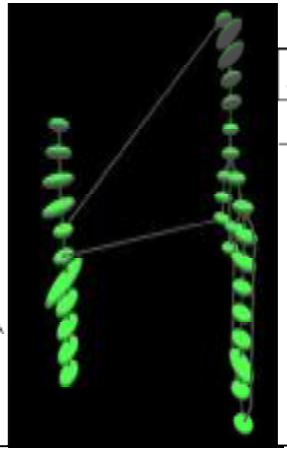
- Weak scaling study on Hopper
- 10 iterations
- Experimental configurations:
- MPI: 16 processes/node (4/socket of 6 cores), local domain = 92^3/process
- Bamboo: 16 worker threads/node (4/socket and 1/core), 8 tasks/worker
- Local domain = 46^3 per task

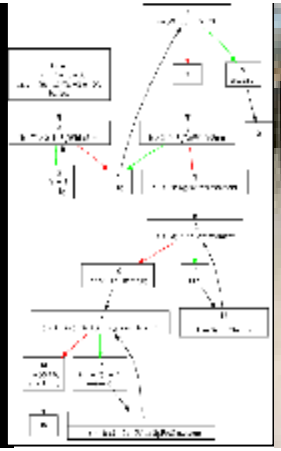


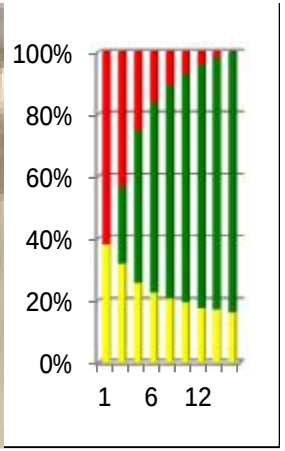
```
int aFunction(int a, int b)
{
        int c=b;
        return a;
}

main()
{
  int a,b,c,d,e;
  int i=4;
  for (i=0;i<10;i++)
  {
    int j=55;
    c=i+j;
    c=aFunction(i,c);
    a=aFunction(a+1,b);
  }
  #pragma SliceTarget
  a;
  return 0;
}
```









DSL examples

DTEC Project

Georgia Tech IBM MIT Lawrence Livermore National Laboratory RICE OHIO STATE UNIVERSITY BERKELEY LAB OREGON

Eight Example DSLs

1) D-TEC Halide http://halide-lang.org Image processing algorithms Cloverleaf, miniGMG, boxlib Uses C++ Custom IR Stencil optimizations (fusion, blocking, parallelization, vectorization) Schedules can produce all levels of locality, parallelism and redundant computation. OpenTuner for automatic schedule generation. LLVM X86 multicores, Arm and GPU Working system. Demonstrated for DOE Miniapps; also used commercially by Google and Adobe. Interfaces with the OpenTuner (http://opentuner.org) to automatically generate schedules. Working on visualizing/debugging tool.

2) *DTEC* Shared Memory DSL http://rosecompiler.org MPI HPC applications on many core nodes Internal LLNL App Uses C (maybe C++ and Fortran in future) ROSE IR Shared memory optimization for MPI processes on many core architectures permits sharing large data structures between processes to reduce memory requirements per core. ROSE + any vendor compiler Many core architectures with local shared memory Implementation released (4/28/2014) Being evaluated for production use at LLNL.

3) *D-TEC* X-GEN for heterogenous computing http://rosecompiler.org/ HPC applications running on NVIDIA GPUs boxlib, internal kernels Uses C and C++ ROSE IR (AST) loop collapse to expose more parallelism, Hardware-aware thread/block configuration, data reuse to reduce data transfer, round-robin loop scheduling to reduce memory footprint ROSE source-to-source + NVIDIA CUDA compiler NVIDIA GPUs Implementation released with ROSE (4/29/2014) Matches or outperforms caparable compilers targeting GPUs. Generate event traces for gpuplot to identify serial bottleneck

4) *D-TEC* NUMA DSL http://rosecompiler.org HPC applications on NUMA-support many core CPU internal LLNL App Uses C++ ROSE IR NUMA-aware data distribution to enhance data locality and avoid long memory latency. Multiple halo exchanging schemes for stencil codes using structured grid. ROSE + libnuma support Many core architecture with NUMA hierarchy implementation in progress. 1.7x performance improvement compared to OpenMP implementation for 2D 2nd order stencil computation. PAPI is used for for performance profiling. libnuma and internal debugging scheme are used to verify memory distribution among NUMA nodes.

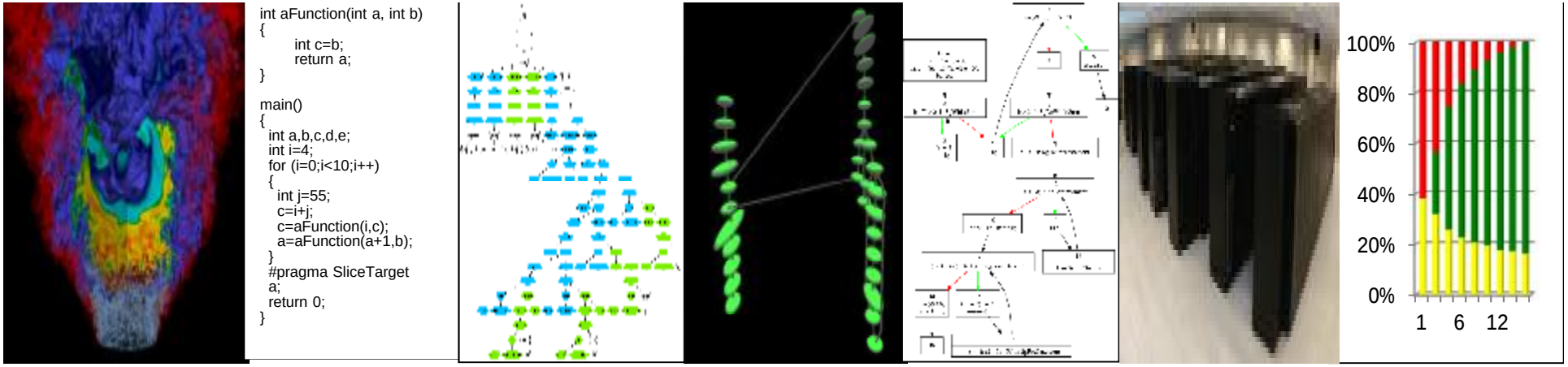
5) *D-TEC* OpenACC https://github.com/tristanvdb/OpenACC-to-OpenCL-Compiler Accelerated computing Not yet. C (possible C++ and Fortran). Pragma parser for ROSE. ROSE IR Uses on tiling to map parallel loops to OpenCL ROSE (with OpenCL kernel generation backend), OpenCL C Compiler (LLVM) Any accelerator with OpenCL support (CPUs, GPUs, XeonPhi, ...) - Basic kernel generation - Directives parsing - Runtime tested on Nividia GPUs, Intel CPUs, and Intel XeonPhi Reaches ~50 Gflops on Tesla M2070 on matrix multiply. (M2070: ~1Tflops peaks, ~200 to ~400 Gflops effective on linear algebra ; all floating point). A profiling interface collects OpenCL profiling information in a database.

6) *D-TEC* Rely http://groups.csail.mit.edu/pac/rely/ Reliability-aware computing and Approximate computing Internal kernels Subset of C with additional reliability annotations. Custom IR A language and a static analysis framework for verifying reliability of programs given function-level reliability specifications. Chisel, a code transformation tool built on top of Rely, automatically selects operations that can execute unreliably with minimum resource consumption, while satisfying the reliability specification. Generates C source code. Binary code generator Implementation in progress. Analysis of computational kernels from multimedia and scientific applications.

7) *D-TEC* Simit Computations on domains expressible as a graph Internal physics simulations, Lulesh, MiniFE, phdMesh, MiniGhost Uses C++ Custom IR Fusion, Blocking, Vectorization, Parallelization, Distribution, Graph Index Sets LLVM X86 multicores, GPU and later distributed systems Design and implementation in progress Has a visual backend.

8) Maple based DSL to represent mathematical operators and automate construction of discretizations for cartesion and curvalinear coordinate grids. Supports automated generation of higher order stencils for 2^{nd}, 4^{th}, 6^{th}, and 8^{th} order operators.

9) AMR Stencil DSL (specification and implementation design of DSL compiler in progress, but initial work on optimizations done by OSU)



RoseBud

Scott Warren (Rice)

Georgia Tech IBM MIT Lawrence Livermore National Laboratory RICE OHIO STATE UNIVERSITY BERKELEY LAB OREGON

Rosebud DSL Framework

- Goal: make DSLs easier to *build* and to *use*
  - reduce clerical effort, reinvention of wheels, manual coding
  - improve robustness, composability, debugging & profiling
- Approach: "one stop shopping" system for DSL construction
  - all aspects of DSL compilation
    - parsing, ASTs, semantics, rewriting, optimization, runtime, tool integration
  - all flavors of DSL, used with any ROSE host language
    - host-syntax ("embedded"), custom-syntax, stand-alone
  - rich compiler infrastructure via ROSE
    - C++ / Fortran analysis & synthesis, abstractions, optimizations, runtime
- Expected impact:
  - improve DSL availability, usability, and performance for HPC
  - foster DSL adoption in HPC via community-wide development & sharing

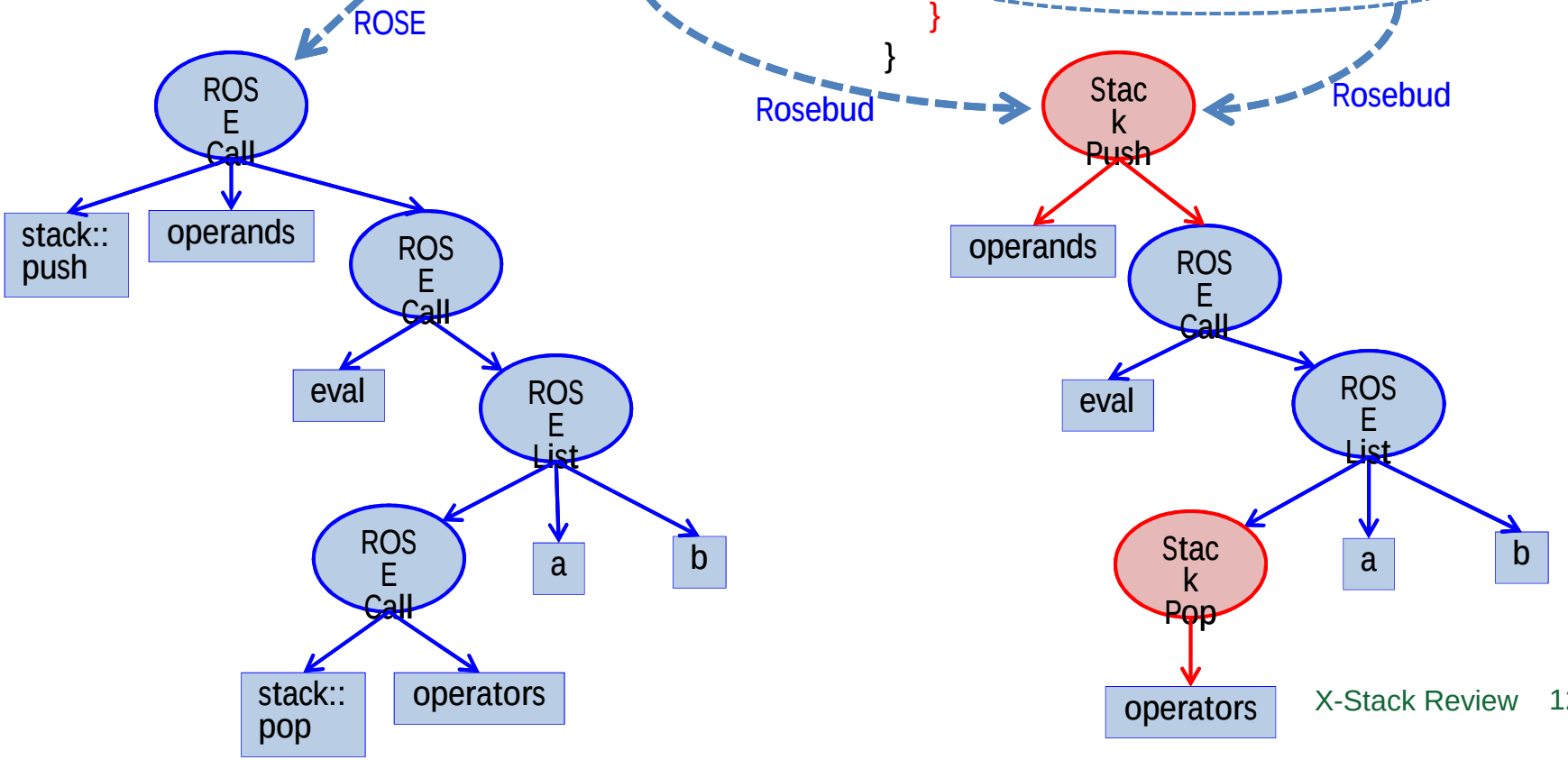
Rosebud Example: *Stacks DSL*

Host syntax ("embedded")

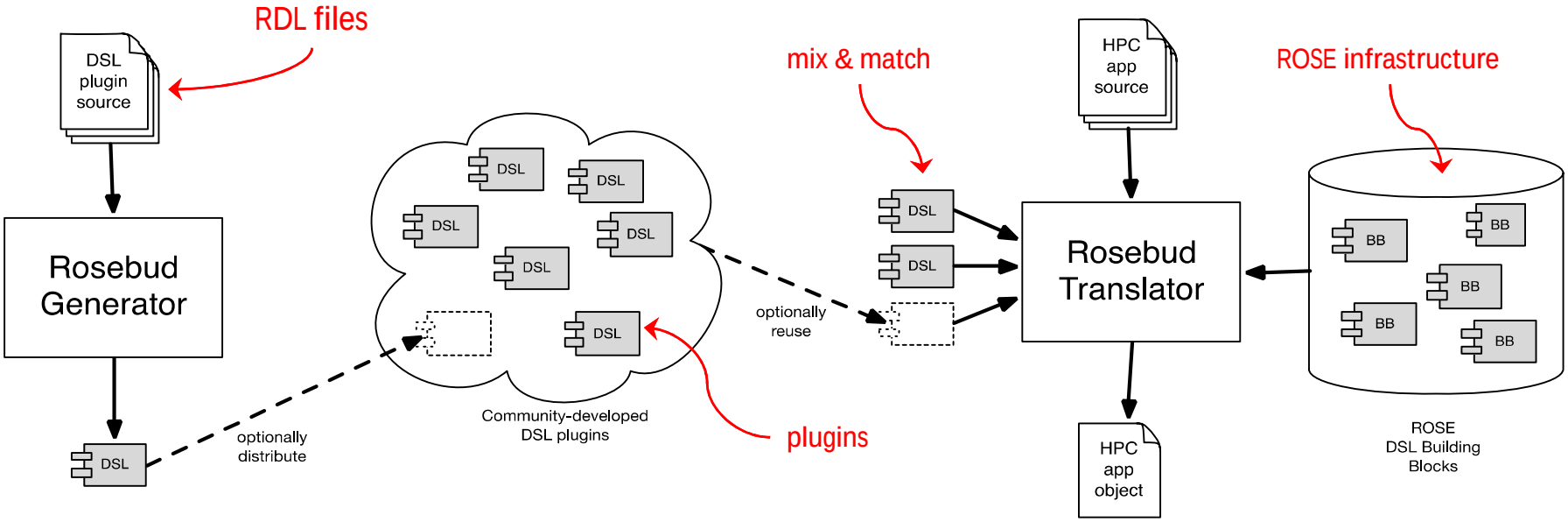
```
while ( priority ( token ) <= priority ( operators . top ()) )
{
    operands . pop (a);
    operands . pop (b)
    operands . push ( eval ( operators .pop (), a, b));
}
```

Custom syntax

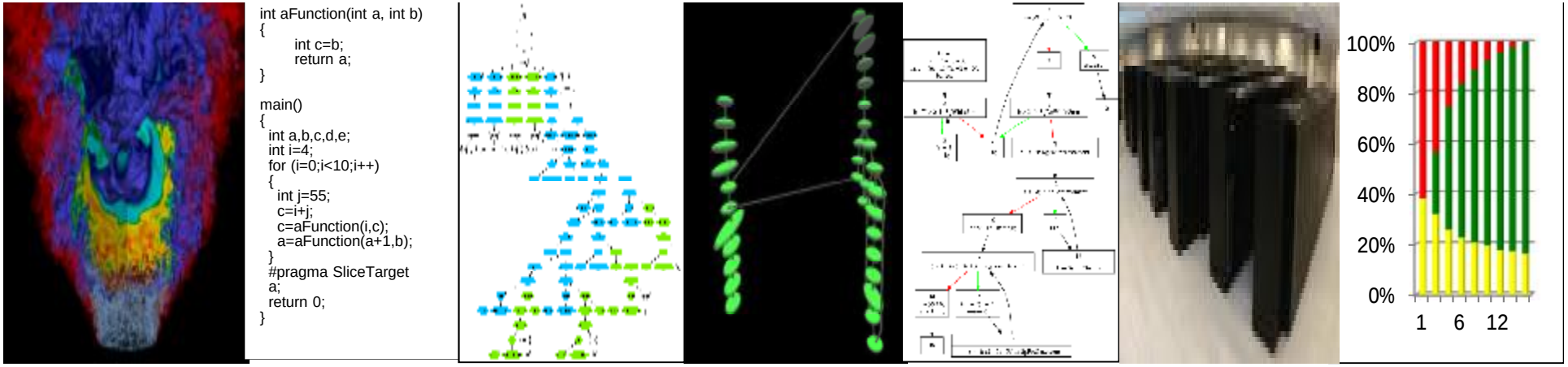
```
while( priority (token) <= priority (top of operators) )
{
    with stack operands
    {
        pop to a;  pop to b;
        push eval (pop off operators, a, b);
    }
}
```



Rosebud Status



- Detailed design is complete
  - Rosebud Definition Language, plugin architecture, parsing & type checking, O-O structure
- Infrastructure & high level code are complete
  - build system, common core, *Generator*, *Translator*, *PlugUtil*
  - plugin database, host plugins for C++ and Fortran (preliminary)
  - documentation, tests, example DSLs
- *Generator* can compile RDL files to plugins
  - includes & imports, RDL parsing, symbol tables, plugin writing
  - only handles **`concrete syntax`** sections so far
- *Translator* can load plugins and parse mixed-language source files
  - Stacks example plugin, Stacks in C++, Stacks in Fortran
- *PlugUtil* can install, uninstall, list, and dump plugins



Refinement

Armando Solar-Lezama (MIT)

Georgia Tech IBM MIT Lawrence Livermore National Laboratory RICE OHIO STATE UNIVERSITY BERKELEY LAB OREGON

Overview

- Synthesis in support of the DTEC vision of pervasive DSLs

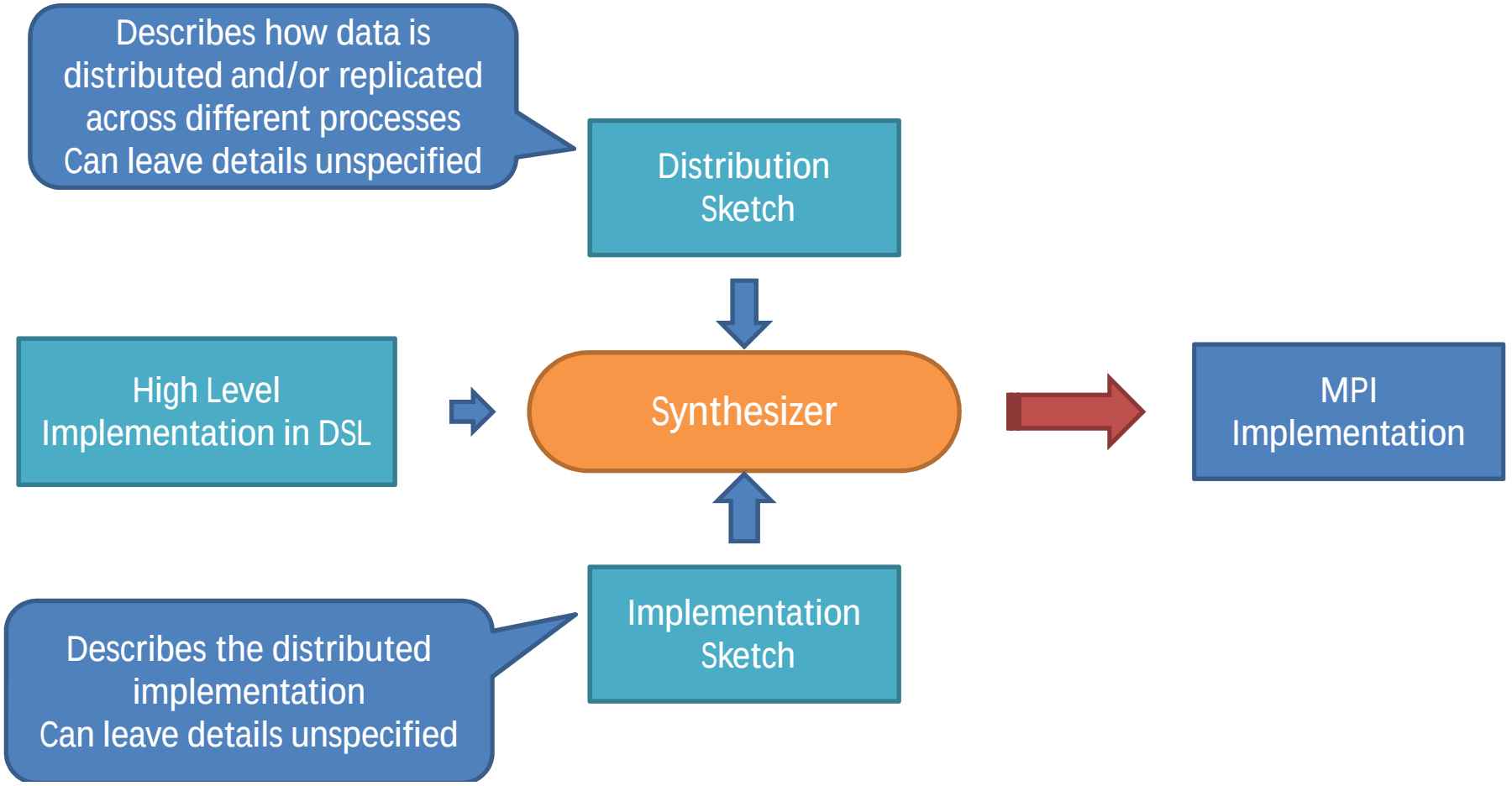
1) MSL Language for refinement based development of MPI code

2) Synthesis to aid migration from legacy code to DSL

3) Solver Aided DSLs with Rosette

MSL: Synthesis Based Refinement for MPI

- **Support experimentation with different approaches to**
  - Data-distribution
  - Communication/Computation tradeoffs



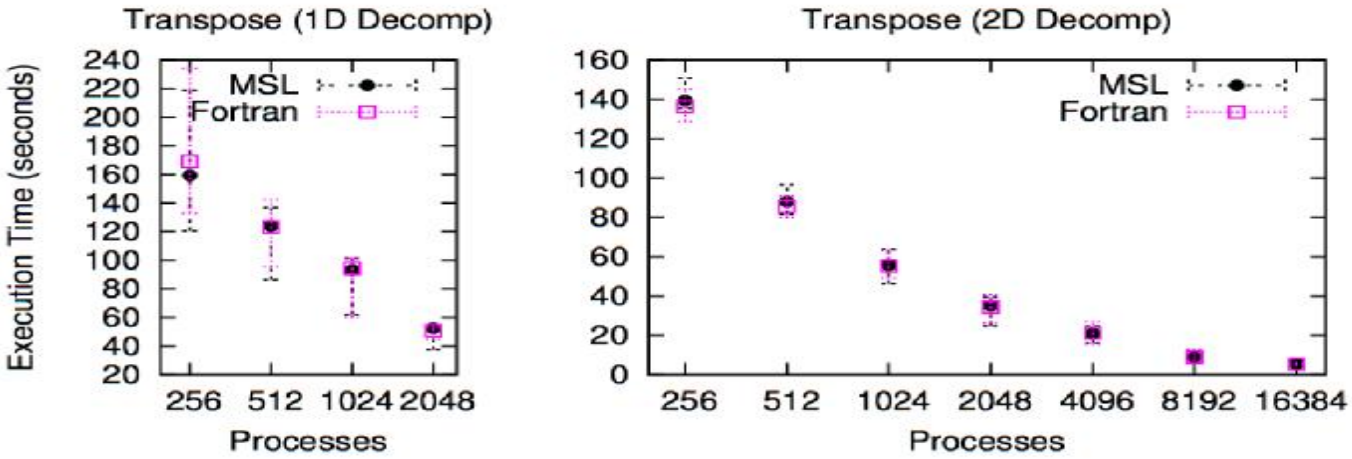
Key Results

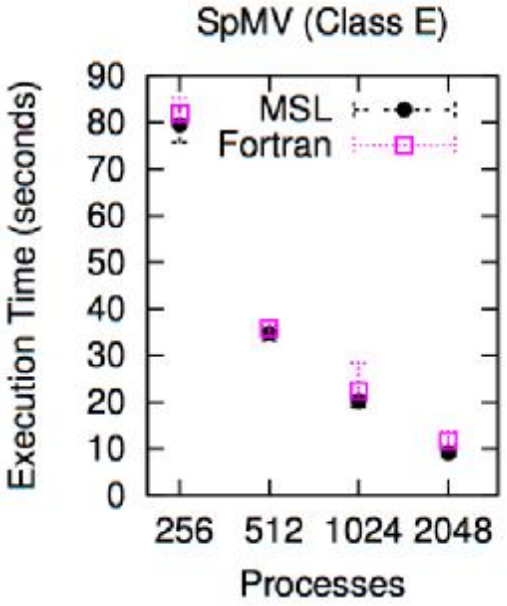
- **Implemented 3 Kernels with the help of synthesis**
  - SPMV
    - Important for many applications (Mantevo MiniFE miniapp)
    - Demonstrates ability to reason about irregular computation
  - Transpose
    - Important part of BigFFT miniapp
    - Requires support multiple communicators to
  - Multigrid
    - Non-trivial communication patterns

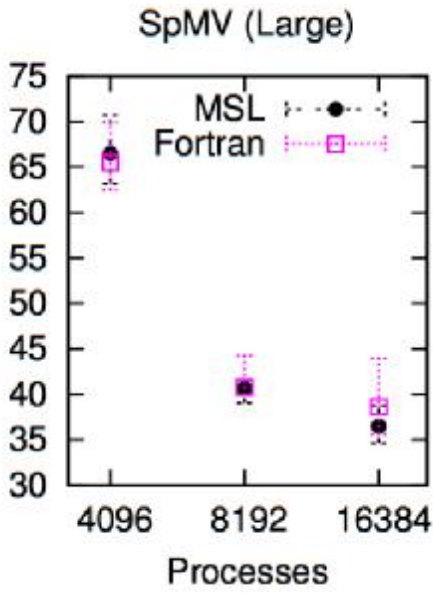
- **Synthesized the details of looping and communication code from a sequential reference implementation**

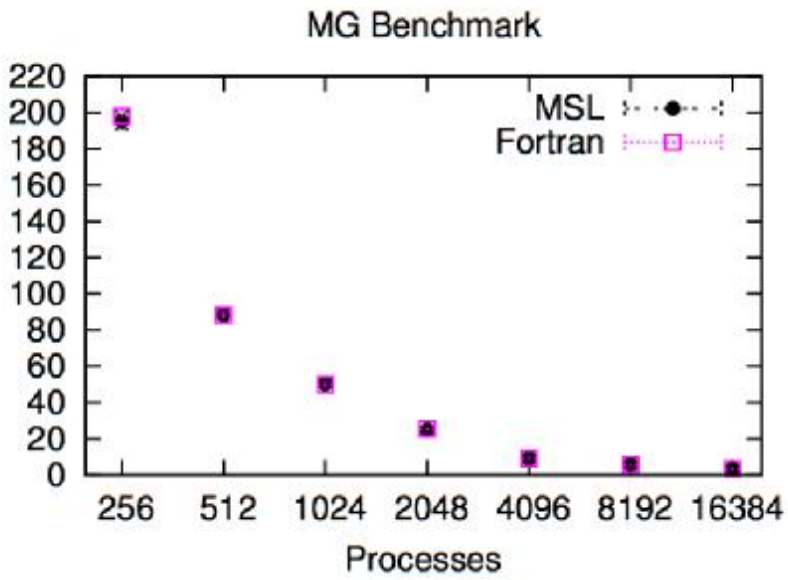
- **Scalability comparable to hand-written Fortran**
  - Scaled up to 16K processes.

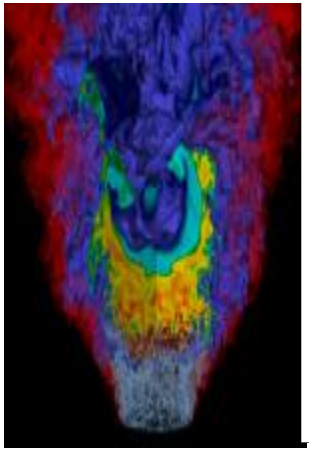
Experiments





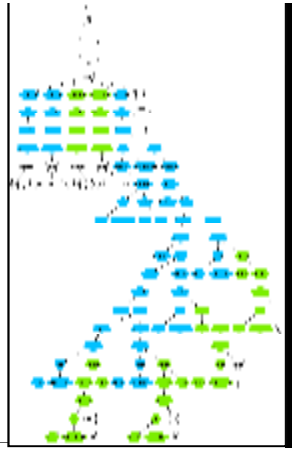


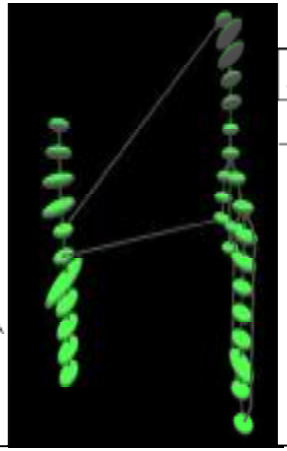


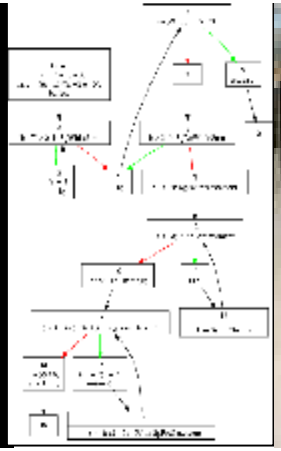


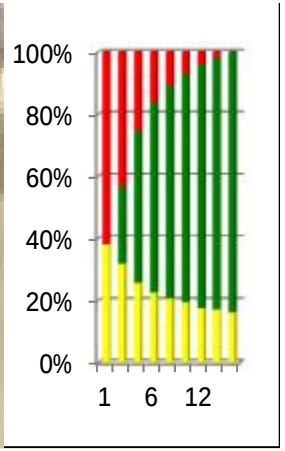
```
int aFunction(int a, int b)
{
        int c=b;
        return a;
}

main()
{
  int a,b,c,d,e;
  int i=4;
  for (i=0;i<10;i++)
  {
    int j=55;
    c=i+j;
    c=aFunction(i,c);
    a=aFunction(a+1,b);
  }
  #pragma SliceTarget
  a;
  return 0;
}
```









Compiler Research

Dan Quinlan (LLNL)
LLNL, Rice, OSU, UO, UCLA

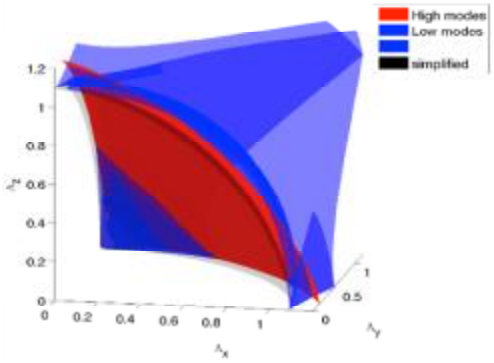
D-TEC

Techniques for Building Domain Specific Languages (DSLs)

Maple DSL Code Generation for Stencils

Scientific Achievement

- Exploiting Maple DSL to generate high order stencil codes using Cartesian and curvilinear coordinates.
- Automatic mode analysis for stencil computation.



$$\left(\frac{c\Delta t}{\Delta x}\right)^{1.75} + \left(\frac{c\Delta t}{\Delta y}\right)^{1.75} + \left(\frac{c\Delta t}{\Delta z}\right)^{1.75} < 1.09$$

Significance and Impact

- Stencil code can be generated directly from mathematical equations expressed in Maple language.
- Simplifying programming effort for stencil computation. ~ 10 lines of essential Maple code can generate a 4th order wave equation using Cartesian (1165 lines Fortran output) and curvilinear coordinates (3626 lines Fortran output).
- Mode analysis is automatically generated with stencil codes from Maple DSL.
- Providing complex stencil code variants (higher order or different coordinate) for researches in performance tuning and compiler optimization.

Scientific Achievement

- Mode analysis reveals essential details about temporal stability for high order discretization. (4th order 3D case shown in figure)
- Various stencil codes in different complexities generated for scientific computing and compiler research purposes.

ROSE

Lawrence Livermore National Laboratory

D-TEC

Techniques for Building Domain Specific Languages (DSLs)

ROSE X-Gen Accelerator Support

Scientific Achievement

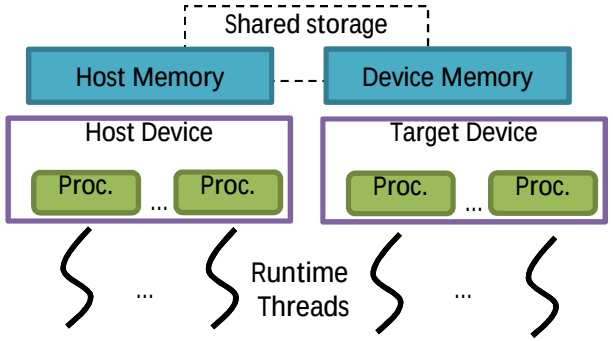
- Automated generation of accelerator support
- Competitive performance compared to other compilers
- Extensions to support multiple GPUs

Significance and Impact

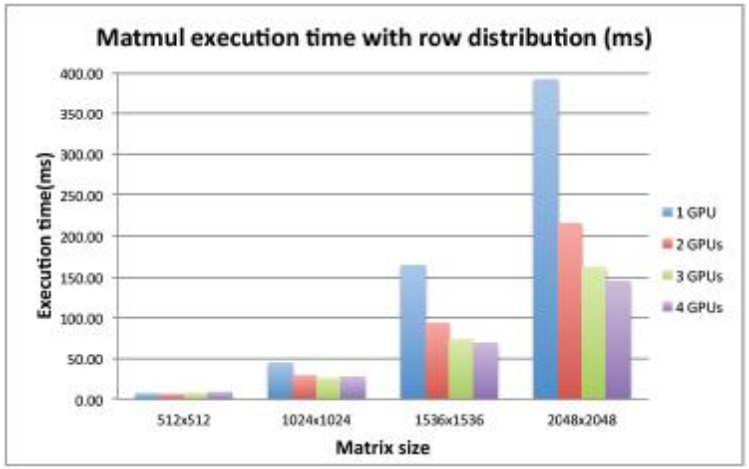
- Directive-based programming models are representative and popular DSLs
- Extending compiler support to effectively support accelerators such as GPUs will significantly improve the productivity of extreme-Scale computing

Research Details

- Language extensions to represent semantics of GPU computing: data /computing offloading, multiple GPUs, etc
- Compiler transformation to generate CUDA kernels from input code annotated with directives
- Runtime support for kernel launch, data distribution, memory management, scheduling, and reduction operations



Hardware Model : Host (CPU) and target (GPU) devices, with separated memory spaces (could be shared also)



Matrix multiplication with directives for multiple GPUs: computation/data offloading and data distribution

C. Liao, Y. Yan, B. R. de Supinski, D. J. Quinlan, and B. Chapman, "Early experiences with the accelerator model," in the era of low power devices and accelerators, Springer, 2013, pp. 84-98.

ROSE

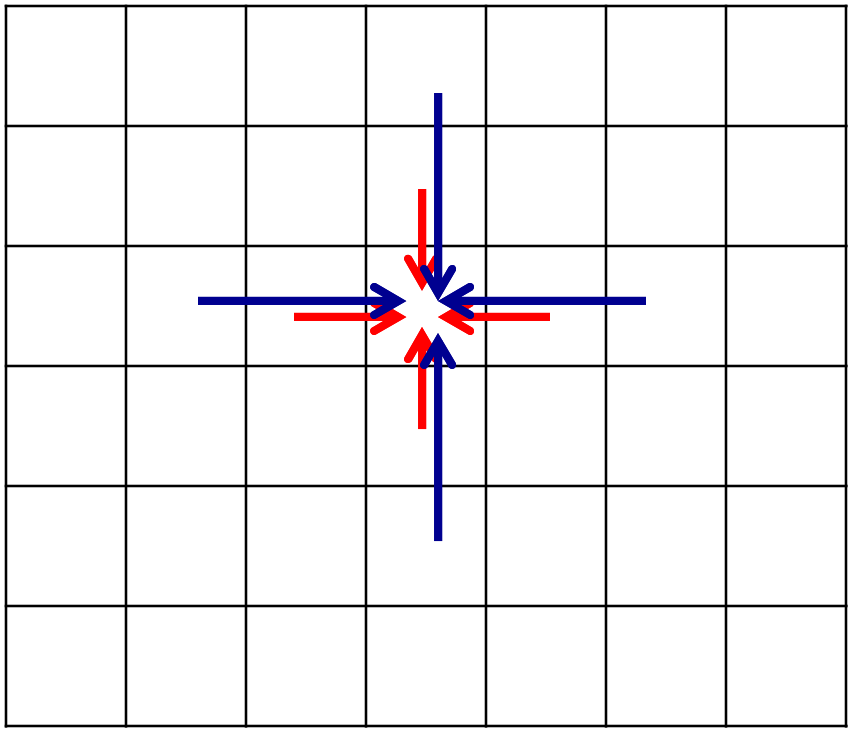
Lawrence Livermore National Laboratory

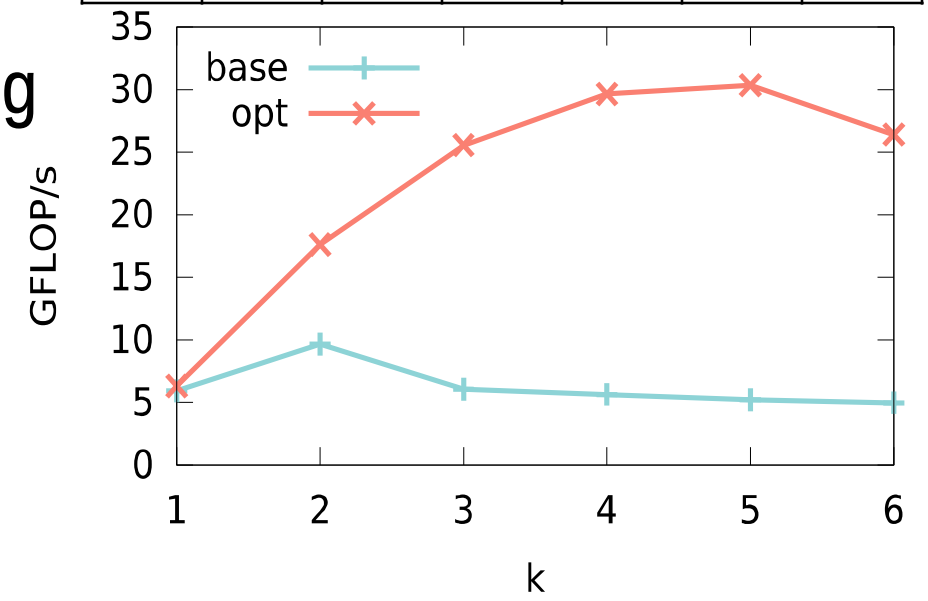
Domain-Specific Compiler Optimization (OSU)

- Code generation and optimization for stencil DSLs
  - Goal: Translate stencil specifications to efficient code
  - Multi-target code generation
    - Multicore CPU – split tiling with data layout transformations (ICS'13); GPU – overlapped tiling (ICS'12); FPGA (FPGA'13)
  - State-of-the-art optimizations
    - Enhanced vectorization via tiled codelets (PLDI'13)
    - Optimization of high-order stencils (PLDI'14)
- Enhancements of polyhedral optimizer PolyOpt within ROSE
- Current Focus: DSL for Phil Colella's AMR stencil calculus
  - Abstractions for restricted subset of calculus (with help from LBNL)
  - Develop domain-specific optimizations, plus multi-target code generation and lower-level optimizations
    - Extensions to handle multiple grids and interactions at their boundaries
    - Extensions to multi-level grids and their interactions

Optimizing High-Order Stencils

- Higher order stencils have higher arithmetic intensity , but performance is flat with increasing order: register saturation
- New approach: Use global information about stencil compute pattern over grid, and exploit assoc. reordering [Stock et al., PLDI ‘14]
- Demo at Technology Marketplace





Towards DSL for AMR Stencil Calculus

- AMR-based PDE/ODE solvers
  - High-level expressions for sweeps over disjoint set of rectangular sub-domains, at different levels of refinement
  - Operators are typically point-wise applications or stencil operations over a local neighborhood
- Library based framework (C++ classes) can implement semantics, but very slow
- Many opportunities for domain-specific compiler optimization
  - Fusion of stencils along different spatial dimensions
  - Multi-level tiling
  - Memory management for temporary intermediates
  - Preliminary results from manual domain-specific optimization

$$
\begin{aligned}
&\textbf{for all } d \in 0, \ldots, D-1 \textbf{ do} \\
&\quad F^d : H_d(\Omega^{c,k}) \bigcup H_d^{-1}(\Omega^{c,k}) \rightarrow \mathbb{R} \\
&\quad F^d := \partial_d^{CtoF,h_c,P}(\langle \tilde{\varphi} \rangle^{c,k}) \\
&\quad \textbf{for all } k' \in \{0, \ldots, n^f_{grids} - 1\} \textbf{ do} \\
&\qquad \delta F^{(d,k',\pm)} := \mp H_d^{\pm}(F^d) \textbf{ on } \zeta_{k'}^{\pm,d} \\
&\quad \textbf{end for} \\
&\quad L^{c,k} += \partial_d^{FtoC,h_c} F_d \\
&\textbf{end for}
\end{aligned}
$$

C++ Class Library	0.1
Stencil -> C: Direct	100
Stencil -> C + Transform	350

Performance on quad-core Intel Sandybridge i7-2600 (Mega-stencils per second)

Exp_CNS Proxy Application

- Compressible Navier Stokes Solver
  - Finite-difference scheme using 8th order stencil along spatial directions and 3rd order Runge Kutta scheme over time
  - 11 3D arrays for fundamental variables advanced through time, and many temporary intermediates
- Computation can be expressed using stencil calculus
- Currently explored optimizations improve performance, but further enhancements possible
  - Loop transformations enable significant reduction between data traffic between DRAM and L3 cache
  - Transformed code no longer limited by main-memory BW
  - New kernel optimization approaches needed: reduce core stall cycles

	Mem Reads	Mem Writes	Performance
Reference	46.3 GB	15.4 GB	6.9 GFLOPs
Optimized	14.1 GB	7.6 GB	10.8 GFLOPs

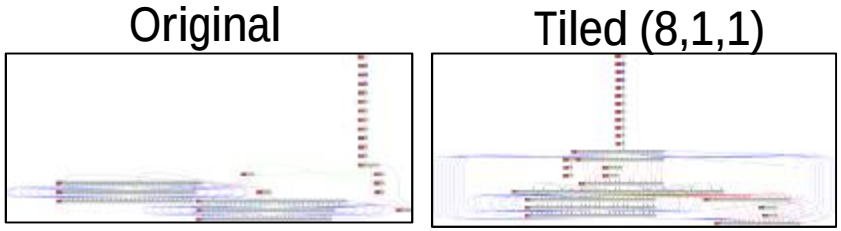
D-TEC

Techniques for Building Domain Specific Languages (DSLs)

Polyhedral Transformation Verification

Scientific Achievement

Verified semantic equivalence of variants of polyhedral transformed codes with constant loop bounds.



Two state transition graphs determined to represent equivalent programs.

Significance and Impact

We verified whether 1487 PolyOpt generated tiling and fusion variants for the Polybench 3.2 benchmarks are semantics preserving. This work allowed to detect a bug in the PolyOpt 0.2 code generator that is not determined by any of the existing tests. Future work will involve this new form of verification in the release process of PolyOpt.

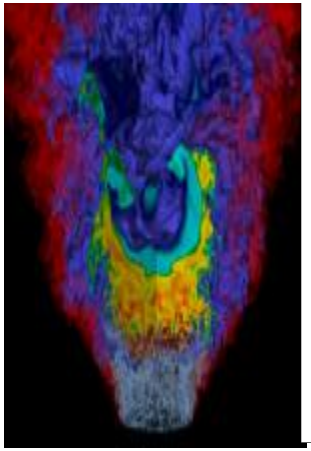
Research Involved

- Verification is independent of floating point precision.
- Static analysis determines a state transition graph.
- Rewrite system normalizes code associated with each state transition.
- Matching of each variant's code in SSA Form determines equivalence.

Authors: M. Schordan, P. Lin, D. Quinlan (LLNL), L. Puchet (UCLA)

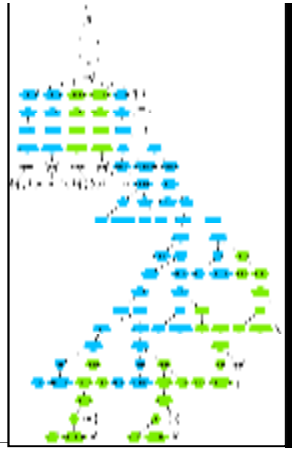
Lawrence Livermore National Laboratory

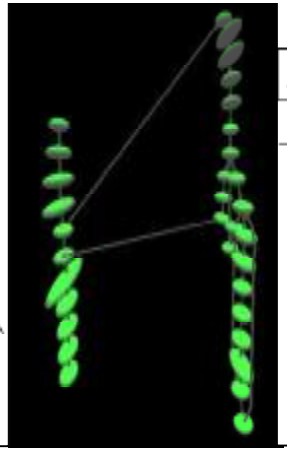
UCLA

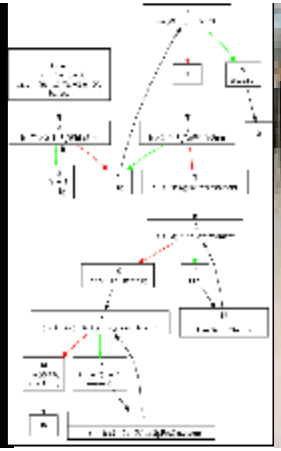


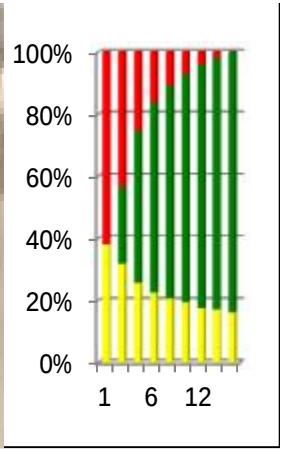
```
int aFunction(int a, int b)
{
        int c=b;
        return a;
}

main()
{
  int a,b,c,d,e;
  int i=4;
  for (i=0;i<10;i++)
  {
    int j=55;
    c=i+j;
    c=aFunction(i,c);
    a=aFunction(a+1,b);
  }
  #pragma SliceTarget
  a;
  return 0;
}
```









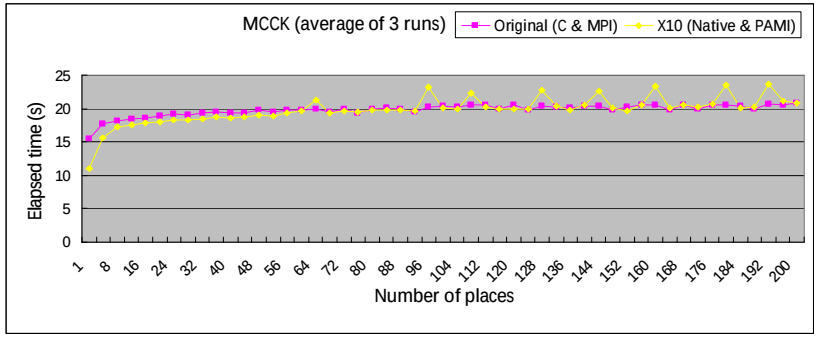
Runtime

David Grove (IBM)

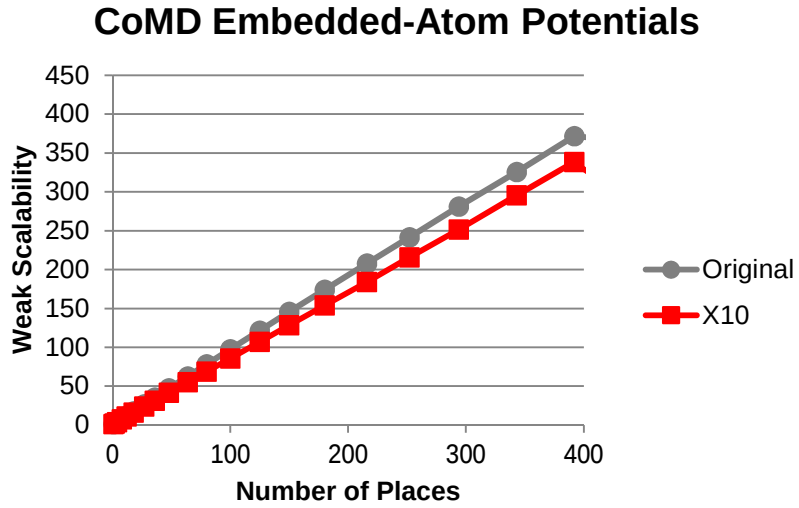
X10/APGAS Proxy Applications

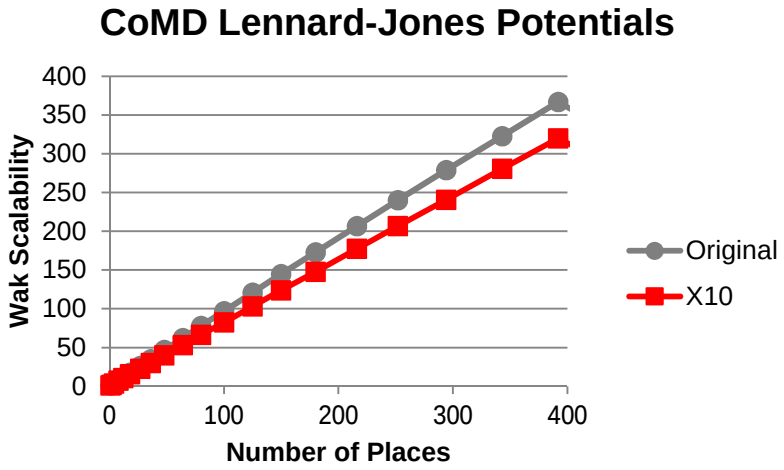
- **X10 implementations of three DoE proxy apps**
  - LULESH v2.0
  - MCCK
  - CoMD
- **Objectives**
  - Drive development of APGAS runtime using DoE relevant computation/communication workloads
  - Foundation for exploring APGAS/X10 based libraries and embedded DSLs that capture computational patterns
- **Status**
  - Proxy applications available open source (see x10-lang.org)
  - Comparable serial and small-scale performance
  - Future: further refinement of app code; at-scale evaluation

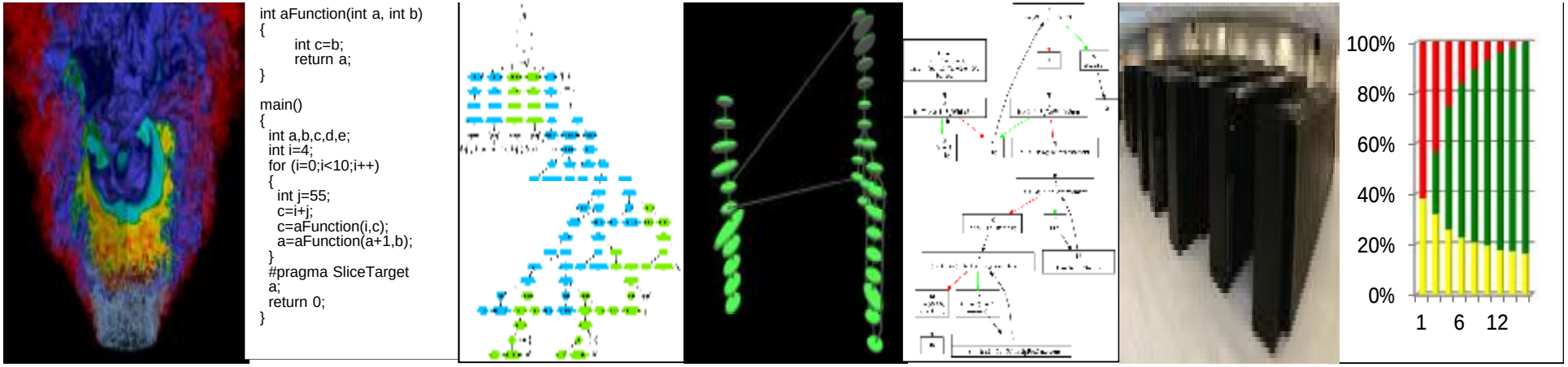
Initial Results (Power775)



Proxy App	X10 SLOC	Ref SLOC
Lulesh	2300	5000
MCCK	750	900
CoMD	3000	3000







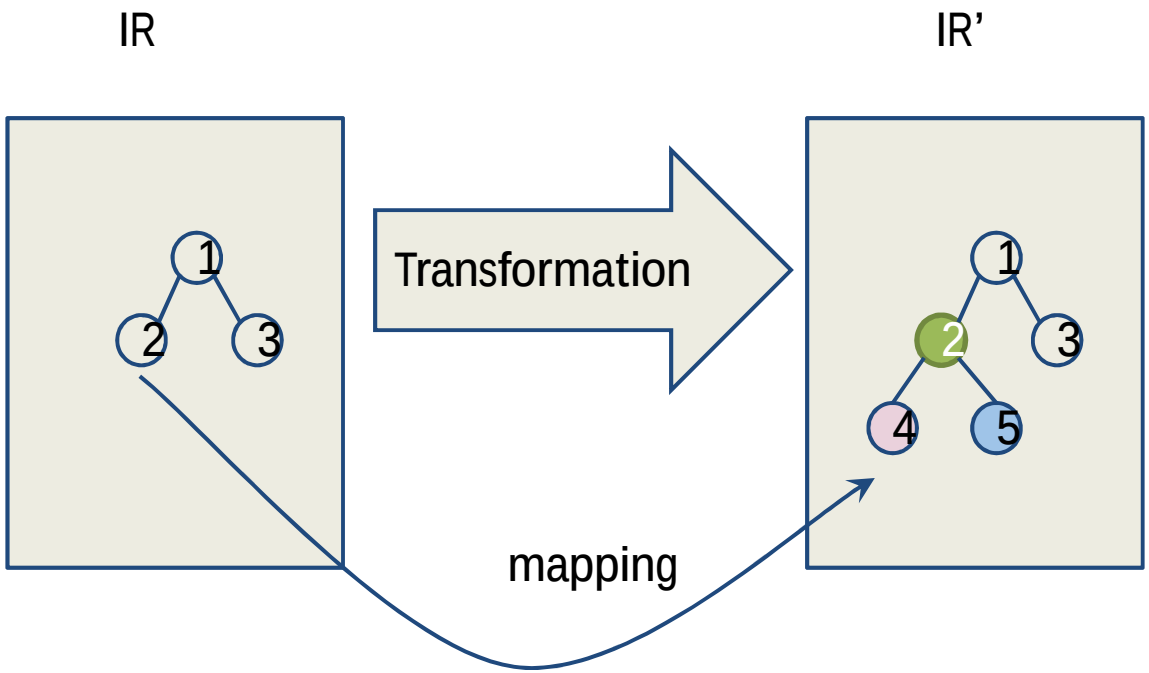
Tools: IR – Mapping for Tools Support

Chunhua Liao (LLNL)

IR-Mapping: motivation

- **Problem: gap between high-level DSL and low-level IRs (intermediate representation) after code transformation**
  - Traditional compilers only save single-level source location info.
- **Benefits of tracking transformations and maintaining multi-level IR mapping**
  - Program analysis:
    - Propagate high-level semantics to low level IRs:
      - e.g. readonly, ordered, continuous storage, etc.
    - Attribute low-level performance metrics to high-level IRs
  - Program transformation
    - Optimizations, debugging

Transformation and IR mapping



Trans_name (input_nodes, output_nodes)
e.g. add_children ((2), (2,4,5))

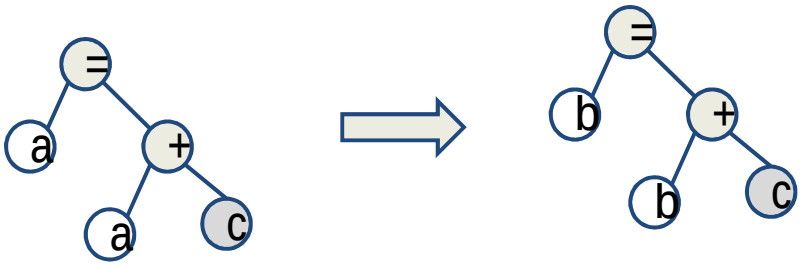
Model Transformations: Hierachical

- **Atomic transformations**
  a. Creation: from scratch vs. copy-construction
  b. Modification: self-contained changes
  c. Attachment: insert into AST
  d. Detachment: remove from AST
  e. Destroy: deallocation
- **Composite transformations**
  a. Replacement
  b. Dead code elimination
  c. Loop interchange, tiling, unrolling
  d. Inlining, outlining
  e. OpenMP Lowering

Example: variable renaming

a = a +c; --> b = b + c

renaming variables within a subtree

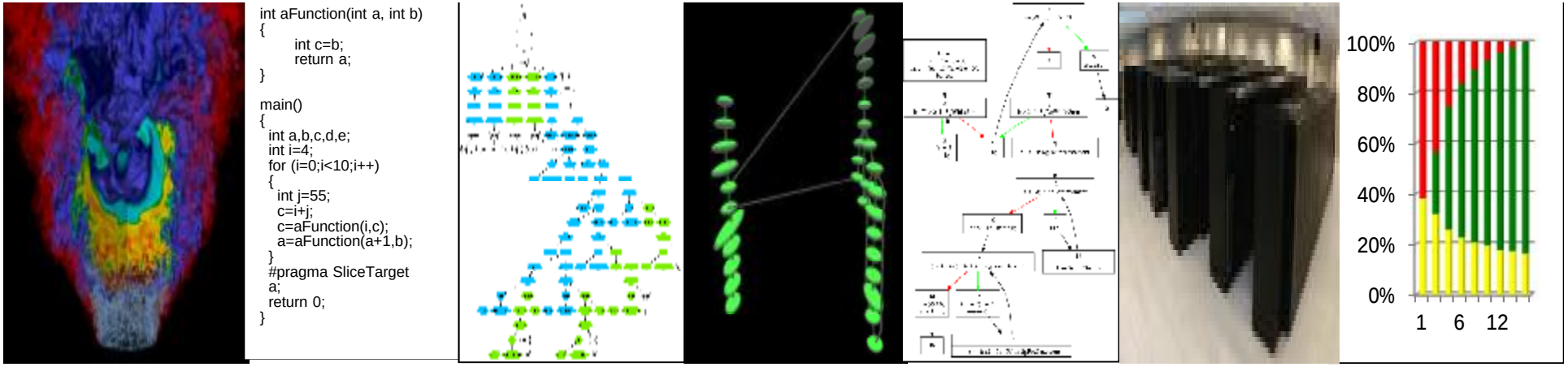


variable_renaming_root, {(a,a),(b,b)}

Composite transformation has mapping relations aggregated from atomic/leaf transformations

variable_renaming, {(a), (b)}

variable_renaming, {(a), (b)}



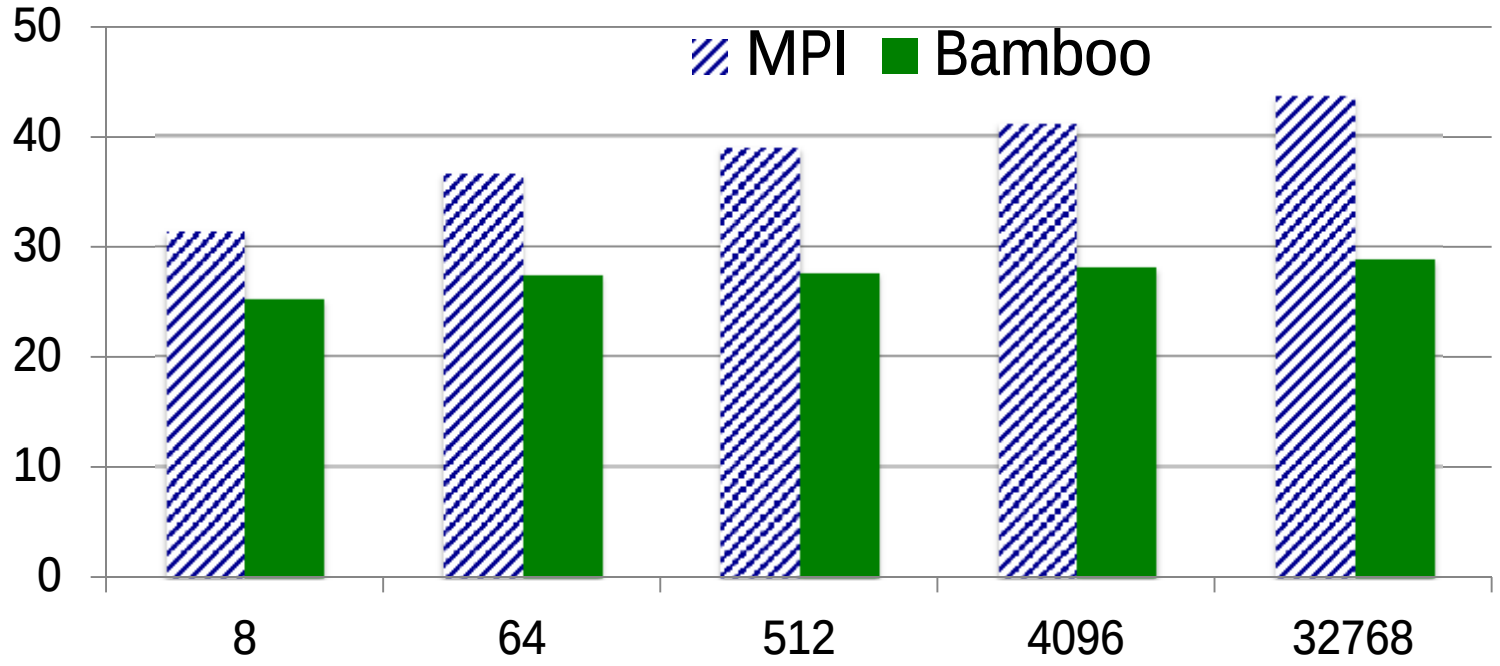
Tools: Source-to-source transformation of a legacy application to tolerate latency on exascale systems

Scott B. Baden
Dept. of Computer Science and Engineering
University of California, San Diego

Experience with Bamboo on LULESH (1.0) Proxy App

- Experiments on Hopper @ NERSC
  - Weak scaling study
  - Bamboo translator restructured the code to hide communication automatically (compiler available at bamboo.ucsd.edu)
- Reduced waits on communication
  - Significant performance improvement:
    20% on 8 cores → 35% on 32K cores
  - Communication cost of MPI variant is growing with the # cores
  - Virtual processing helped improve overlap: 8 tasks per core
  - Important on Exascale levels with many more cores
- Additional improvements
  - The provided MPI restricts the # cores: $P^{1/3}$ must be an integer
  - Bamboo's provision for virtual processing relaxes this restriction

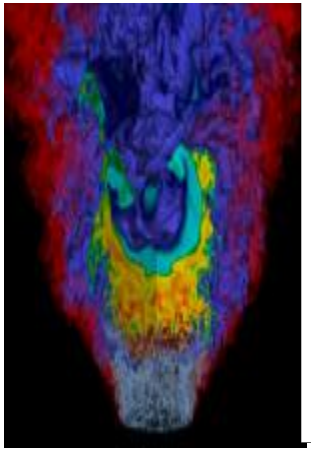
Compiler transformations can overlap communication and computation for improved performance



- Weak scaling study on Hopper
- 10 iterations
- Experimental configurations:
- MPI: 16 processes/node (4/socket of 6 cores), local domain = 92^3/process
- Bamboo: 16 worker threads/node (4/socket and 1/core), 8 tasks/worker
- Local domain = 46^3 per task

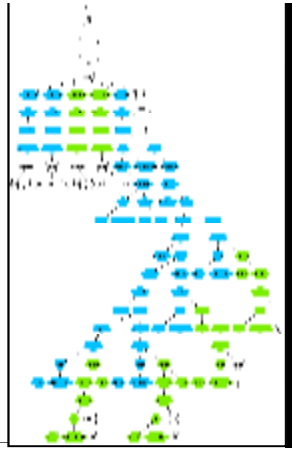
Hiding communication delays in legacy applications

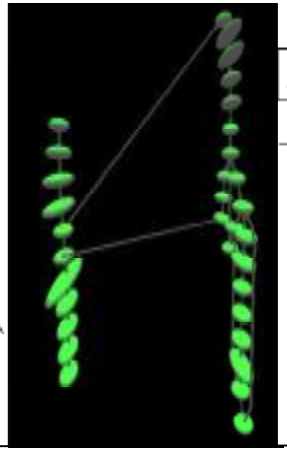
- Problem
  - Communication delays are continuing to grow, limiting scalability
  - Intrusive recoding required to tolerate communication delays in legacy MPI applications
  - Many programmers lack the background to carry out the required restructuring
- Solution
  - Use a domain specific translator (Bamboo) to restructure the code
  - Hide communication automatically
  - Available at http://bamboo.ucsd.edu
- Impact
  - Avoids the need to implement complicated split-phase code by hand
  - Legacy applications re-engineered (almost) automatically
  - Nearly all large-scale parallel scientific applications are written either in MPI directly or via a library implemented in MPI

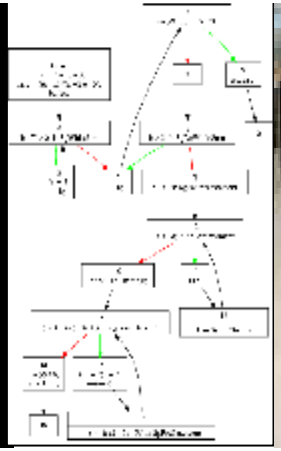


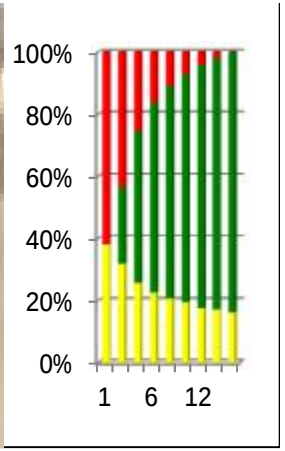
```
int aFunction(int a, int b)
{
        int c=b;
        return a;
}

main()
{
  int a,b,c,d,e;
  int i=4;
  for (i=0;i<10;i++)
  {
    int j=55;
    c=i+j;
    c=aFunction(i,c);
    a=aFunction(a+1,b);
  }
  #pragma SliceTarget
  a;
  return 0;
}
```









Applications

Phil Colella (LBL)

Stencil Calculus on Rectangular Grids

$$\boldsymbol{i}, \boldsymbol{p} \cdots \in \mathbb{Z}^D$$

$$\mathcal{S} = (S_0, \ldots, S_{D-1}) \ , \ S_d(\boldsymbol{i}) = \boldsymbol{i} + \boldsymbol{e}^d$$

$\boldsymbol{e}^d$ is the unit vector in the d direction

then $\sum a_{\boldsymbol{q}} \mathcal{S}^{\boldsymbol{q}}$ defines a stencil operator, $\mathcal{S}^{\boldsymbol{p}} = \prod_{d=0}^{D-1} S_d^{p_d}$

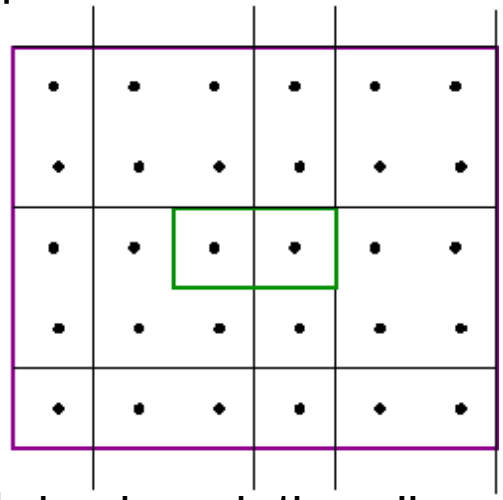
$$U : \Lambda \to \mathbb{T} \ , \ \Lambda \subset \mathbb{Z}^D \qquad \left(\left(\sum a_{\boldsymbol{q}} \mathcal{S}^{\boldsymbol{q}}\right)(U)\right)_{\boldsymbol{i}} = \sum a_{\boldsymbol{q}} U_{\boldsymbol{i}+\boldsymbol{q}}$$

Other operators that define set calculus, centering, interlevel operations, pointwise application of functions:

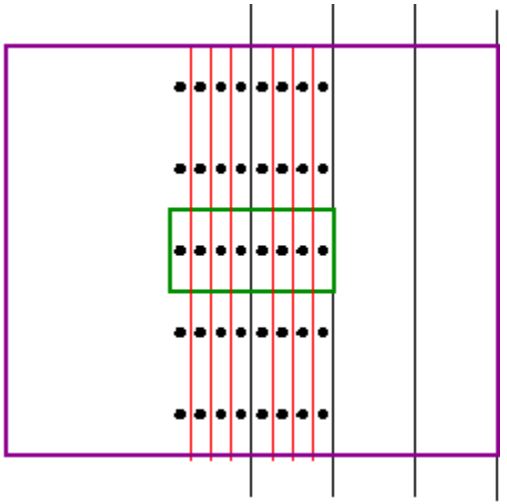
$$f : \mathbb{T} \to \mathbb{T}' \ , \ f@U : \Lambda \to \mathbb{T}' \ , \ (f@U)_{\boldsymbol{i}} = f(U_{\boldsymbol{i}})$$

Example: Coarse-fine interpolation

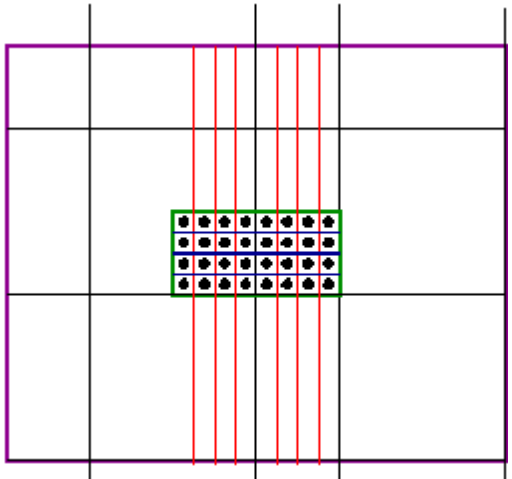
Want to interpolate from coarse grid centers to the region bounded by the green box using a factor of 4 refinement ratio, and a tensor product of 1D five-point interpolation operators.



Purple box bounds the cells needed to perform the interpolation.



Interpolation in the x-direction



Interpolation in the y-direction

$$\mathcal{I}_d = \sum_{s=0}^{n_{ref}-1} S_d^s (\mathcal{P}^{fc}_{\boldsymbol{u}+\boldsymbol{e}^d(n_{ref}-1)})^* \sum_{s'=-s_0}^{s_0} a^{1D}_{s,s'} S_d^{s'}$$

$$n_{ref} = 4 \ , \ s_0 = 2.$$

Example: Coarse-fine interpolation

```
// Implementation in Stencil DSL.

Stencil<double> interp[DIM];

// Interpolation stencils have already been defined, using the
// stencil expression on the previous slide.

MDArray<double> coarse; // Input coarse data.

Box bxDest; // input fine box.

// compute bxCoarse, coarse grid required for interpolation
Box bxCoarse = grow(coarsen(bxDest,N_REF),S_0);

MDArray<double> tmp = coarse on bxCoarse;

// “on” is a new keyword, restricting the RHS to a subdomain.

  for (int dir = 0; dir < DIM ; dir++)
        {
          tmp = interp[dir](tmp);
        }
MDArray<double> fine = tmp on bxDest;
```