ES3

Exascale Software Stack: Present, Future

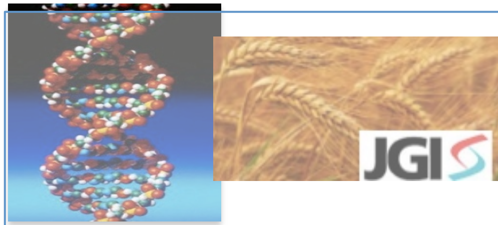
Sonia R. Sachs
PACT'2014
August 26, 2014

Reflecting upon hero programming

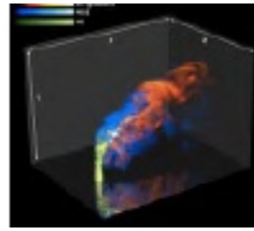
- **Most of us have been hero programmers**
  - Early systems, 8KB of memory, no debugging tools
  - Assembler programming for microprocessors and the many challenges of developing and debugging code
  - Parallel programming and the many additional challenges
  - A short list of current heroes of libraries, kernels, and applications programming

Hero programming culture needs to change in order for us to achieve our vision of the future for Exascale computing and beyond

DOE Extreme Scale Science

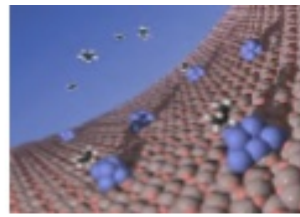


Genomics
(e.g., Plant Genome Assembly)

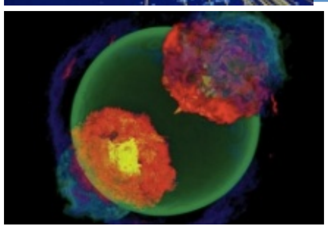


Combustion
(e.g., Turbulent, chemically reacting systems)

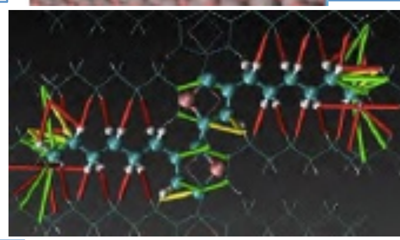
HEP Energy Frontier
(e.g., Higgs Boson discovery used billions of simulated proton-proton events)



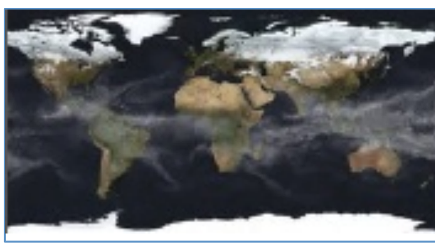
Quantum Models
(e.g. in computational biology, chemistry models)



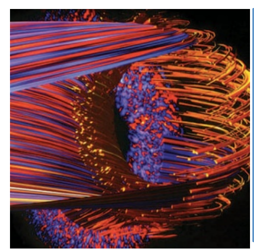
HEP: Cosmic Frontier
(e.g., Supernova Simulations)



Materials Design
(e.g., ab-initio electronic structure methods for excited states)



Climate Models
(e.g., Large-ensemble multi-decadal predictions)



Fusion Energy
(e.g., modeling of fusion plasmas)

- Energy Technologies: photovoltaics, internal combustion devices, batteries
- Novel materials: energy applications, electronics
- Manufacturing technologies

Office of Science, ASCR has significant role in Exascale computing

The mission of the Advanced Scientific Computing Research (ASCR) program is to *discover, develop, and deploy the computational and networking capabilities* that enable researchers to analyze, model, simulate, and predict complex phenomena important to the Department of Energy.

Exascale Computing
We Need to Reinvent Computing

Traditional path of 2x performance improvement every 18 months has ended

- For decades, Moore's Law plus Dennard scaling provided more, faster transistors in each new process technology
- This is no longer true – we have hit a power wall!
- The result is unacceptable power requirements for increased performance

We cannot procure an exascale system based on today's or projected future commodity technology

- Existing HPC solutions cannot be usefully scaled up to exascale
- Energy consumption would be prohibitive (~300MW)

Exascale will require partnering with U.S. computing industry to chart the future

- Industry at a crossroads and is open to new paths
- Time is right to push energy efficiency into the marketplace

Bill Harrod, Exascale Computing Initiative (ECI)

Exascale Computing
The Vision

- **Exascale computing**
  - Achieve order 10^{18} operations per second and order 10^{18} bytes of storage
    - 1,000X capabilities of today's platforms
    - Within 2X-3X of today's power envelope (~20MW)
    - 20 pJ per average operation (~40X improvement over today's systems)
  - Set the US on a new trajectory of progress – towards a broad spectrum of computing capabilities over the next decades
- **Productive, performance portable, and adaptive system**
  - Programming environments that are accessible, easier to use, and enable the development of platform-independent, high performance code.
  - Execution environments that enable the dynamic, adaptive management of system resources for efficiency & scalability
- **Highly Resilient system**
  - Application and runtime level resilience methods
  - self-diagnosis, self-healing
- **Based on marketable technology**
  - Not a one-shot system
  - Scalable, sustainable technology

Exascale: The New Computing Frontier

To be deployed in the early 2020's

Text adapted from Bill Harrod, Exascale Update, ASCAC meeting Nov., 2013

Exascale Programming Environments

Exascale Challenges

- Many reports on ASCR website
- Funding Opportunity Announcements (FOAs)
- Top 10 Challenges: ASCAC website under Charges/Reports

New Programming Models

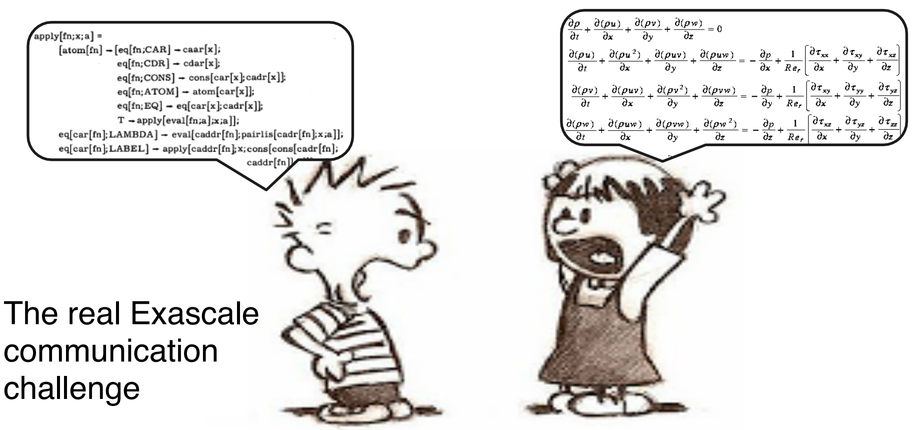
- Processing along the very heterogeneous and complex data path
- Data-centric constructs
- Declarative programming interface
- Tuning for locality and data movement
- Controlling parallel semantics and name space
- And much more…

New Programming Environments

- Automation in transformations, mappings, refinements, and optimizations
- Multiple categories of programmers in the loop

New Programming Environments

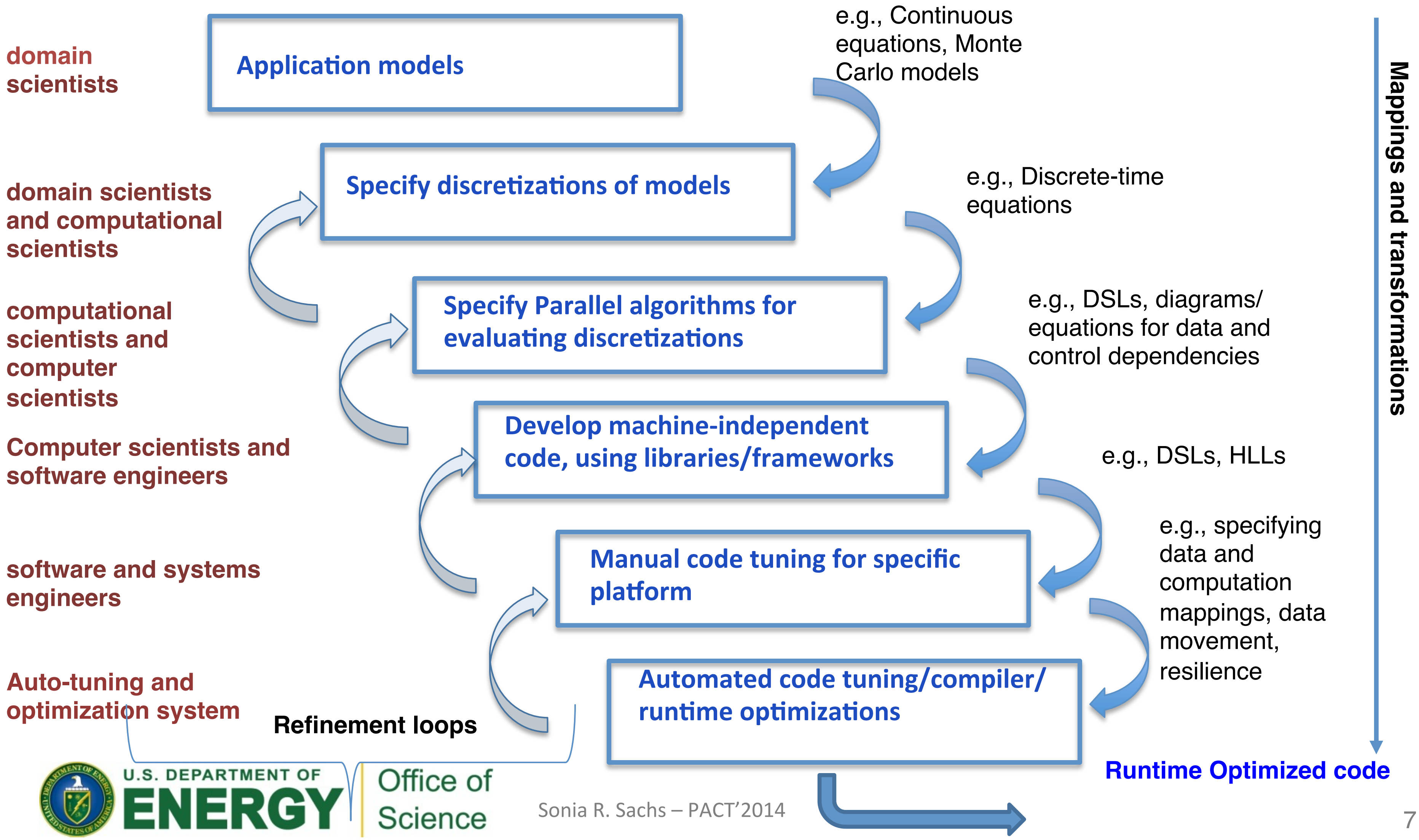
- Rethinking DSLs for addressing the “real” Exascale communication challenge



The real Exascale communication challenge

Exascale Programming Environment

The Vision



X- Stack: the present

www.stackwiki.modelado.org

DEGAS (Kathy Yelick)

Hierarchical and resilient PGAS programming models (within and across nodes), compilers and runtime support.

Traleika (Shekhar Borkar)

Exascale programming system, execution model and runtime, applications, and architecture explorations, with open and shared simulation infrastructure.

D-TEC (Dan Quinlan and Saman Amarasinghe)

Complete software stack solution, from DSLs to optimized runtime systems code.

XPRESS (Ron Brightwell)

Software architecture and interfaces that exploit the ParalleX execution model, prototyping several of its key components.

PIPER (Martin Shultz)

Tools for debugging and analysis of performance, power, and energy

X-Tune (Mary Hall)

Unified autotuning framework that integrates programmer-directed and compiler-directed autotuning.

GVR (Andrew Chien)

Global view data model for architecture support for resilience.

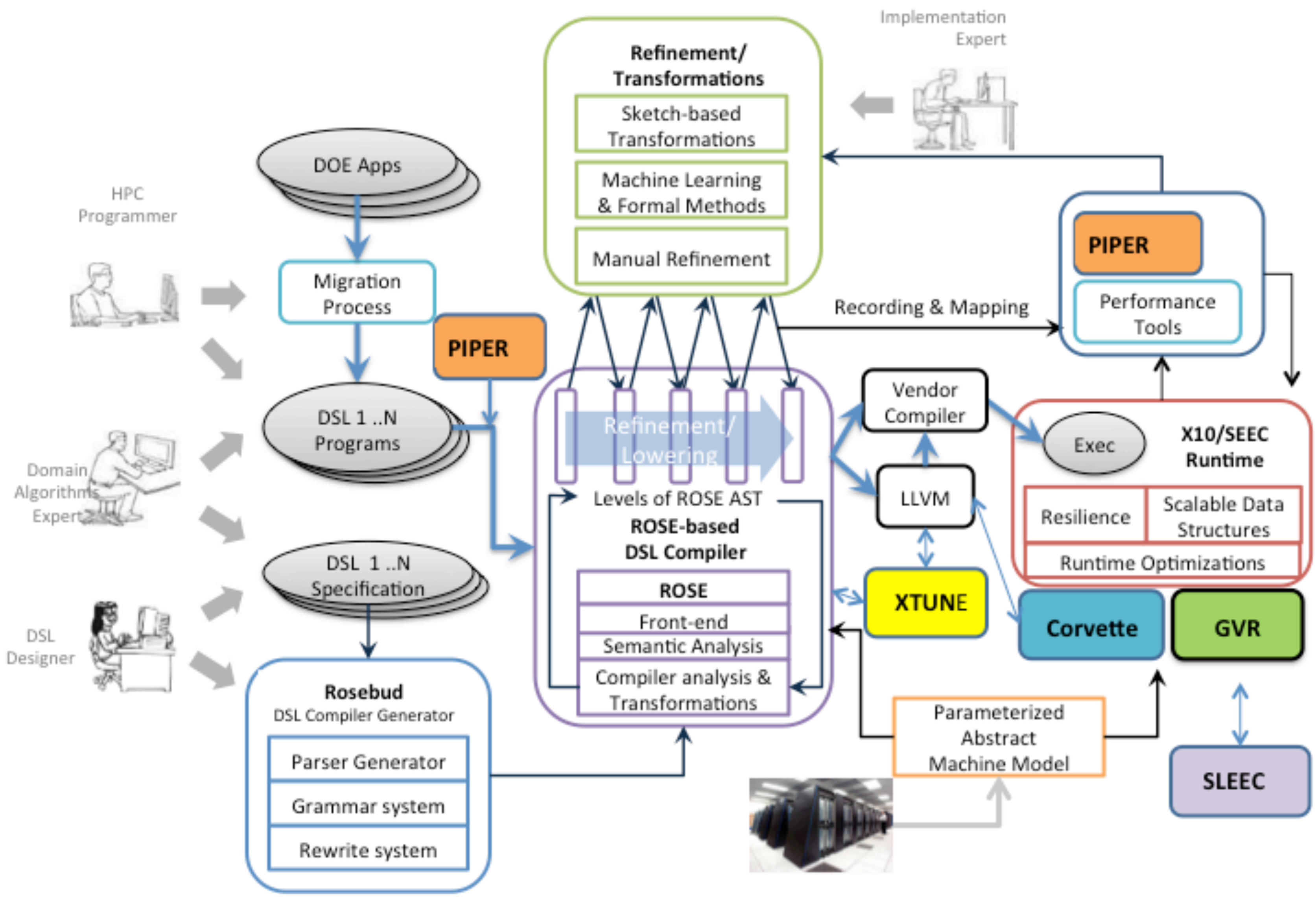
CORVETTE (Koushik Sen)

Automated bug finding methods to eliminate non- determinism in program execution and to make concurrency bugs and floating point behavior reproducible.

SLEEC (Milind Kulkarni)

Semantics-aware, extensible optimizing compiler that treats compilation as an optimization problem.

D-TEC Programming Environment integrated with other X-Stack technologies



D-TEC DSL (Halide) and Refinement/Transformations technologies applied to HPGMG miniapp

Original program (C + OpenMP)

```
#define OMP_THREAD_WITHIN_A_BOX(threads_per_team) \
  if(threads_per_team>1) num_threads(threads_per_team) collapse(2)
int i,j,k;
#pragma omp parallel for private(k,j,i) OMP_THREAD_WITHIN_A_BOX(level->threads_per_box)
for(k=0-ghostsToOperateOn;k<dim+ghostsToOperateOn;k++){
for(j=0-ghostsToOperateOn;j<dim+ghostsToOperateOn;j++){
for(i=0-ghostsToOperateOn;i<dim+ghostsToOperateOn;i++){
  int ijk = i + j*jStride + k*kStride;
  double Ax_n = a*alpha[ijk]*x_n[ijk] - b*h2inv*(
     beta_i[ijk        ]*(valid[ijk-1      ]*( x_n[ijk] + x_n[ijk-1      ]) - 2.0*x_n[ijk])
   + beta_j[ijk        ]*(valid[ijk-jStride]*( x_n[ijk] + x_n[ijk-jStride]) - 2.0*x_n[ijk])
   + beta_k[ijk        ]*(valid[ijk-kStride]*( x_n[ijk] + x_n[ijk-kStride]) - 2.0*x_n[ijk])
   + beta_i[ijk+1      ]*(valid[ijk+1      ]*( x_n[ijk] + x_n[ijk+1      ]) - 2.0*x_n[ijk])
   + beta_j[ijk+jStride]*(valid[ijk+jStride]*( x_n[ijk] + x_n[ijk+jStride]) - 2.0*x_n[ijk])
   + beta_k[ijk+kStride]*(valid[ijk+kStride]*( x_n[ijk] + x_n[ijk+kStride]) - 2.0*x_n[ijk]));
  double lambda = 1.0 / (a*alpha[ijk] - b*h2inv*(
     beta_i[ijk        ]*(valid[ijk-1      ] - 2.0)
   + beta_j[ijk        ]*(valid[ijk-jStride] - 2.0)
   + beta_k[ijk        ]*(valid[ijk-kStride] - 2.0)
   + beta_i[ijk+1      ]*(valid[ijk+1      ] - 2.0)
   + beta_j[ijk+jStride]*(valid[ijk+jStride] - 2.0)
   + beta_k[ijk+kStride]*(valid[ijk+kStride] - 2.0)
   )) ;
  x_np1[ijk] = x_n[ijk] + c1*(x_n[ijk]-x_nm1[ijk]) + c2*lambda*(rhs[ijk]-Ax_n);
}}}

//… scheduling constraints in a different file
level->concurrent_boxes = level->num_my_boxes;
if(level->concurrent_boxes > omp_threads)level->concurrent_boxes = omp_threads;
if(level->concurrent_boxes <           1)level->concurrent_boxes = 1;
level->threads_per_box = omp_threads / level->concurrent_boxes;
if(level->threads_per_box > level->box_dim*level->box_dim)
   level->threads_per_box = level->box_dim*level->box_dim; // JK collapse
if(level->threads_per_box > level->box_dim*level->box_dim*level->box_dim/64)
   level->threads_per_box = level->box_dim*level->box_dim*level->box_dim/64;
if(level->threads_per_box<1)level->threads_per_box = 1;
```

Separation of concerns

Algorithm: describes the computation

- write once by the domain expert
- Much smaller and simpler

```
Func Ax_n("Ax_n"), lambda("lambda"), chebyshev("chebyshev");
Var i("i"),j("j"),k("k");
Ax_n(i,j,k) =  a*alpha(i,j,k)*x_n(i,j,k) - b*h2inv*(
    beta_i(i,j,k)  *(valid(i-1,j,k)*(x_n(i,j,k) + x_n(i-1,j,k)) - 2.0f*x_n(i,j,k))
  + beta_j(i,j,k)  *(valid(i,j-1,k)*(x_n(i,j,k) + x_n(i,j-1,k)) - 2.0f*x_n(i,j,k))
  + beta_k(i,j,k)  *(valid(i,j,k-1)*(x_n(i,j,k) + x_n(i,j,k-1)) - 2.0f*x_n(i,j,k))
  + beta_i(i+1,j,k)*(valid(i+1,j,k)*(x_n(i,j,k) + x_n(i+1,j,k)) - 2.0f*x_n(i,j,k))
  + beta_j(i,j+1,k)*(valid(i,j+1,k)*(x_n(i,j,k) + x_n(i,j+1,k)) - 2.0f*x_n(i,j,k))
  + beta_k(i,j,k+1)*(valid(i,j,k+1)*(x_n(i,j,k) + x_n(i,j,k+1)) - 2.0f*x_n(i,j,k)));
lambda(i,j,k) = 1.0f / (a*alpha(i,j,k) - b*h2inv*(
    beta_i(i,j,k)  *(valid(i-1,j,k) - 2.0f)
  + beta_j(i,j,k)  *(valid(i,j-1,k) - 2.0f)
  + beta_k(i,j,k)  *(valid(i,j,k-1) - 2.0f)
  + beta_i(i+1,j,k)*(valid(i+1,j,k) - 2.0f)
  + beta_j(i,j+1,k)*(valid(i,j+1,k) - 2.0f)
  + beta_k(i,j,k+1)*(valid(i,j,k+1) - 2.0f)));
chebyshev(i,j,k) = x_n(i,j,k) + c1*(x_n(i,j,k)-x_nm1(i,j,k))+
                   c2*lambda(i,j,k)*(rhs(i,j,k)-Ax_n(i,j,k));
```

Schedule: describes execution recipe

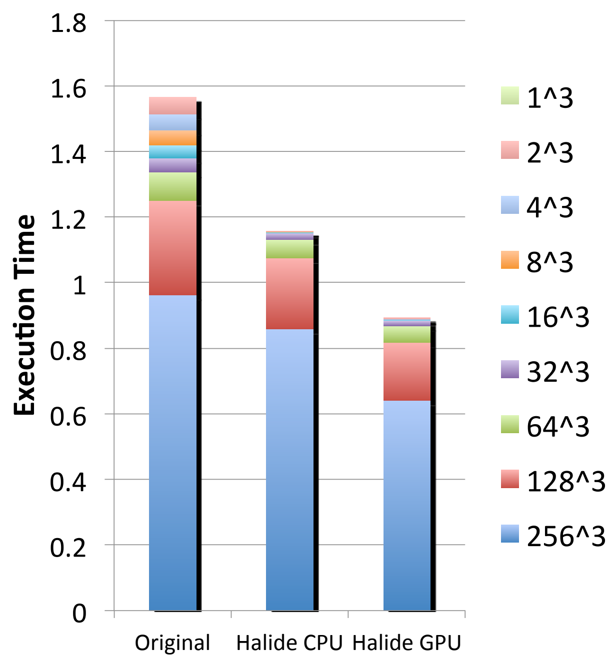
- machine dependent
- Written by performance engineer or auto-generated by autotuning

Optimized C code to Halide

Porting the algorithm was quick and straightforward

Halide performance

Autogenerated schedule for CPU
Hand created schedule for GPU
No change to the algorithm



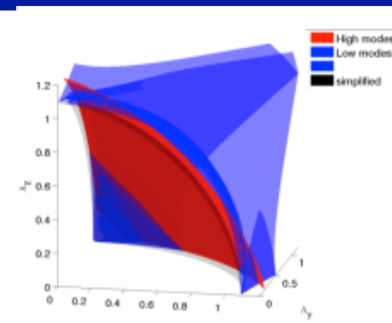
D-TEC DSL (Rosebud, Rose) technologies to Code Generate for Stencils

Scientific Achievement

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

Significance and Impact

- Stencil code can be generated directly from mathematical equations expressed in Maple language.
- 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.

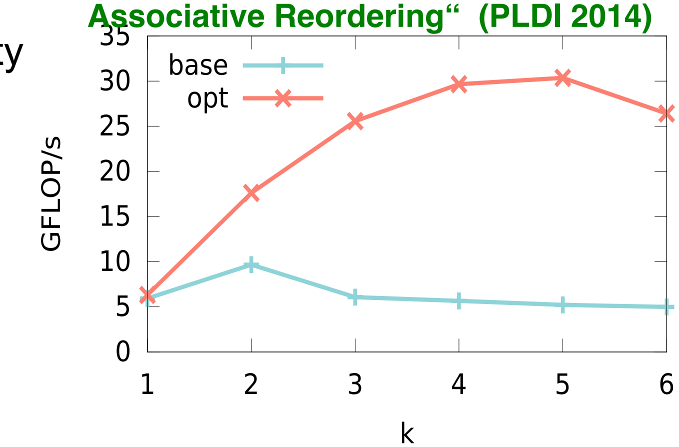


$$\left(\frac{c\Delta t}{\Delta x}\right)^{1.75} + \left(\frac{c\Delta t}{\Delta y}\right)^{1.75} + \left(\frac{c\Delta t}{\Delta z}\right)^{1.75} < 1.09$$

Scientific Achievement

- Mode analysis reveals essential details about temporal stability for 4th order 3D discretization of electromagnetics (shown above)
- Maple-generated code achieves ~94% of computation efficiency compared to a hand-tuned optimized solid mechanics code (2D 2nd order stencil).
- Novel use of associative reordering to significantly enhance performance of high-order stencil computations

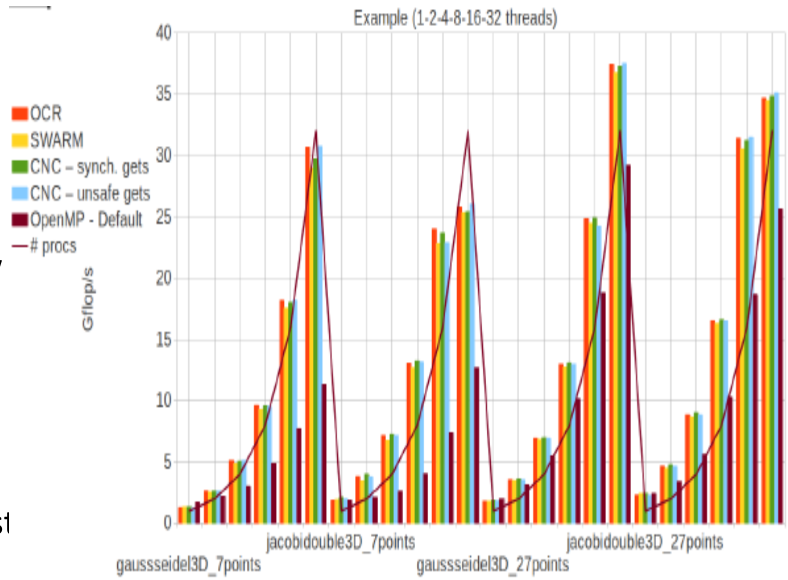
K. Stock, M. Kong, L.N. Pouchet, T. Grosser, F. Rastello, J. Ramanujam, and P. Sadayappan, **"A Framework for Enhancing Data Reuse via Associative Reordering" (PLDI 2014)**



U.S. DEPARTMENT OF ENERGY | Office of Science | Lawrence Livermore National Laboratory | ROSE | UCLA | THE OHIO STATE UNIVERSITY

Automatic optimization to exascale runtime and hardware

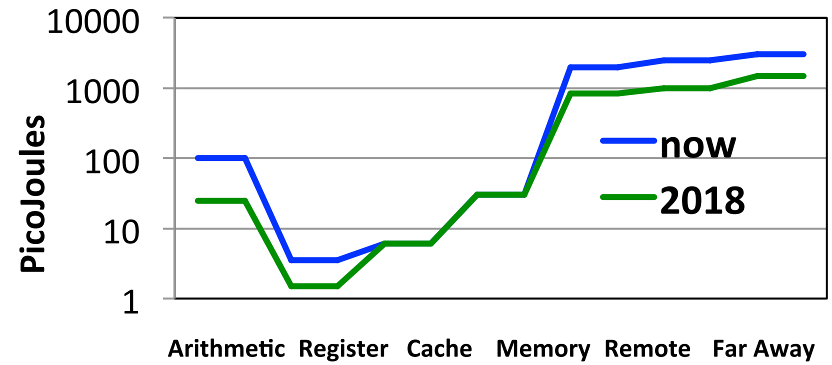
- Problem
  - Exascale hardware will be much more complex to program than just multicore or GPU, MPI or OpenMP – new controls reflecting power constraints
- Solution
  - Automatically parallelize and optimize code for exascale hardware
  - Automatic generation of DMA and scratchpad controls
  - Build on R-Stream parallelizing compiler
- Recent results
  - Auto generation to range of exascale runtimes
  - Trade locality and parallelism simultaneously
  - Virtual scratchpad to achieve results on conventional hw
  - Validated scaling properties of runtimes
  - Demonstrated runtime agnostic layer for deep hierarchy
- Impact
  - Automatic parallelism increases productivity, performance, portability and limits software life cycle cost



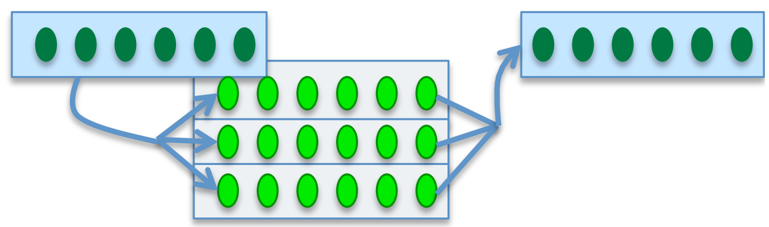
Communication Avoidance in DEGAS

- **Problem**
  - Communication dominates time and energy
  - This will be worse in the Exascale era
- **Solution: Dynamic Exascale Global Address Space (DEGAS)**
  - Optimize latency by overlapping with computation and other communication
  - Use faster one-sided communication
  - Use new Communication-Avoiding Algorithms (provably optimal communication)
  - Automatic compiler optimizations

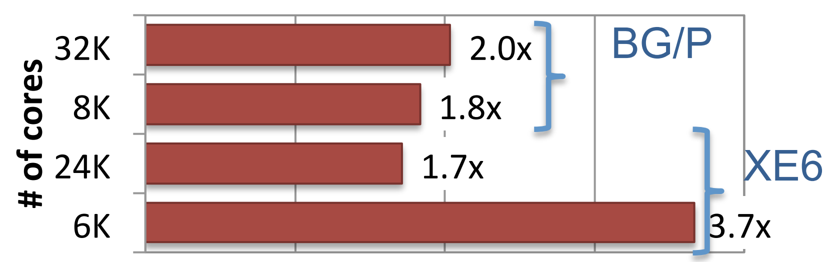
– **Impact**
  - Dense linear algebra study shows 2X speedups from *both overlap and avoidance*
  - New "HBL" theory generalizes optimality to arbitrary loops with array expressions
  - First step in automating communication-optimal compiler transformations



New Communication Optimal "1.5D" N-Body Algorithm: *Replicate and Reduce*



Speedup of New 1.5D Algorithm over Old



[GGSZTY] "Communication Avoiding and Overlapping for Numerical Linear Algebra," SC12.
[DGKSY] "A Communication-Optimal N-Body Algorithm for Direct Interactions," IPDPS 2013.
[CDKSY] "Communication Lower Bounds and Optimal Algorithms for Programs That Reference Arrays — Part 1", UCB TR 2013

DEGAS Leads to More Scalable Meraculous Application for Genomics Grand Challenge

Meraculous assembler is used in production at the Joint Genome Institute

- Wheat assembly is a "grand challenge"
- Hardest part is contig generation (large in-memory hash table)
- Involve irregular data-intensive computations

DEGAS X-Stack project

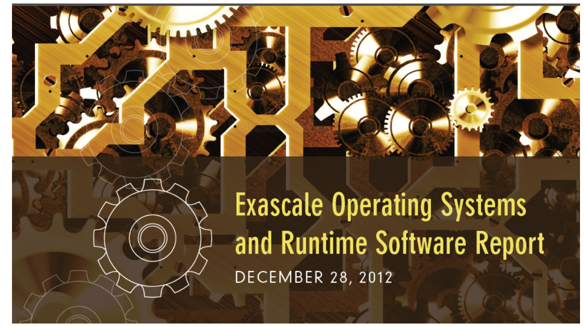
- Hardest part rewritten using PGAS language + asynchronous communication + adaptive scheduling
- Scaled the graph algorithm to 15K cores on NERSC's Edison

Reduced assembly time
Human: from 44 hours to 20 secs
Wheat: from "doesn't run" to 32 secs

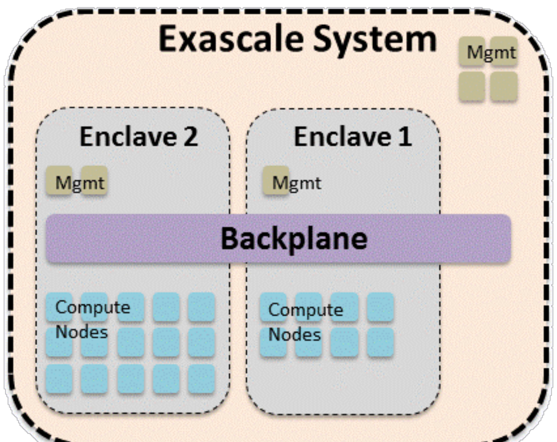
Paper has been accepted at SC'14

Exascale Execution Environments

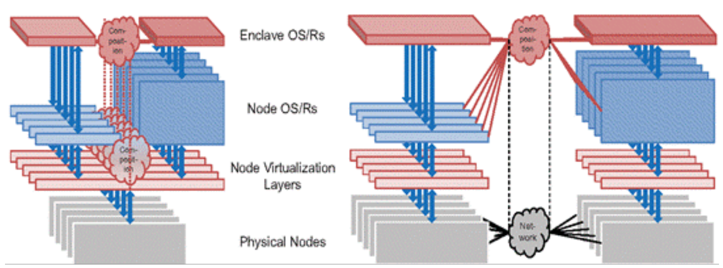
- **OS/R program started Aug 2013:**
  - Create alternative platform-neutral OS/R prototypes and high impact/high risk technologies that eventually converge to one, vendor sustained OS/R.



ARGOS (Pete Beckman, ANL)
- New Node OS/R
- New Lightweight Runtime (self-aware, goal-based, active)
- Backplane for management
- Global OS/R and Optimization



HOBBES (Ron Brightwell, Sandia)
- Lightweight Virtualization
- Application Composition
- Global Information Bus
- Energy and Power
- Resilience
- Programming Models support



Hobbes supports both intra-node and inter-node composition of applications via shared memory or the network respectively. Node virtualization layer is used to isolate the node OS/R.

X-ARC (Stephen Hofmeyr, LBNL, and John Kubiatowicz, UC Berkeley)
- Cross nodes Adaptive resource control
- Support for New Programming Models
- Advanced Memory Management
- Power Awareness
- System Services for Resilience

Future: Exascale Programming and Execution Environments

Near Future (2015-2016):

- One or two programming environments evolve from current programming environments and technologies

Future under ECI (2016-2023):

- R&D for programming and execution environments: much beyond ES^3. We are considering open issues in many research areas. A few examples are:
  - New programming models
  - Interoperability of DSLs with new and existing languages
  - Self-aware, introspective runtime systems
  - Ultra-lightweight task migration and execution.
  - Compilers and runtime systems: Automation in parallelization, optimizations, mappings, transformations, refinements
  - Dealing with hierarchical memory systems, processing in- and near memory, heterogeneous processors, accelerators, etc.
  - Dynamic power, correctness, and resilience optimization
  - Interfaces to the applications and to the hardware
  - Formal methods for verifying correctness
  - New debugging tools, new tools to manage power, resilience, performance

Exascale Computing Timeline