SAND2013-4937C

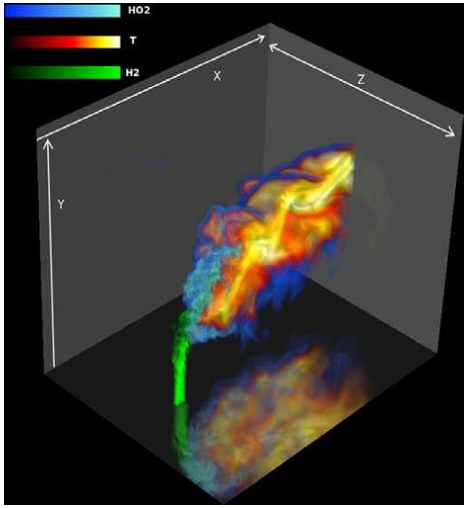
ExaCT Co-Design Center

Jacqueline Chen
Director
Sandia National Laboratories
Livermore, CA

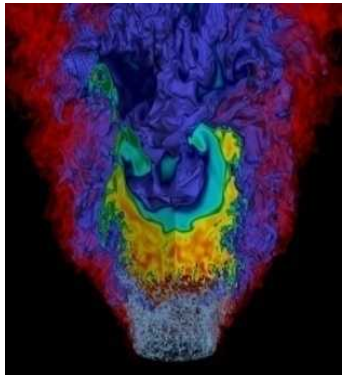
ASCAC Meeting
American Geophysical Union, Washington D.C.
March 5-6, 2013

Why Combustion

- **83% of U.S. energy comes from combustion of fossil fuels**
- **National goals**
  - Reduce greenhouse gas emissions by 80% by 2050
  - Reduce petroleum usage by 25% by 2020
- **New generation of high-efficiency, low emission combustion systems**
  - New designs for IC engines such as HCCI
  - Fuel flexible turbines for power generation
- **Rapidly evolving fuel streams**
  - Biodiesel for transportation
  - Syngas from gasification processes
  - Alcohols
- **These factors significantly increase the design space for new combustion technologies**



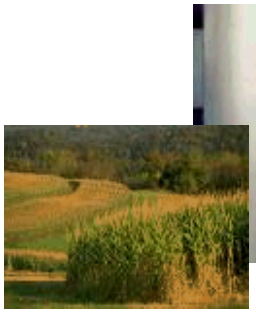
Nitrogen-diluted hydrogen jet in crossflow

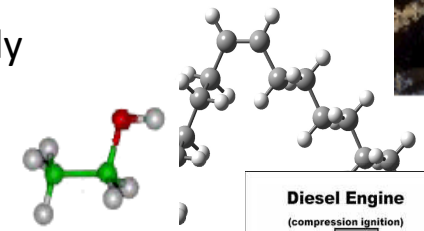


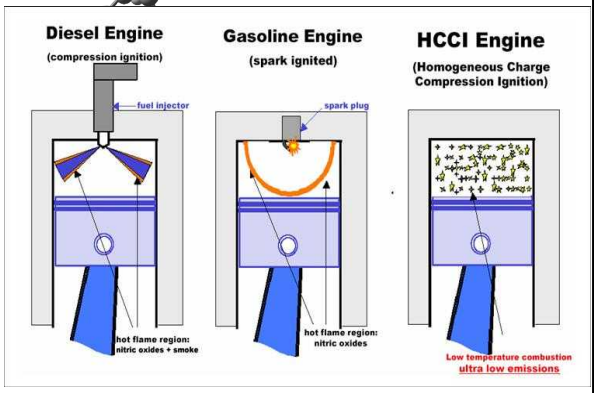
NOx emissions from a low swirl injector fuels by H2

Why Exascale

- **Current design methodologies are largely phenomenological**
  - Combustion models based on simplified theory
  - Global-step simple chemical kinetics
- **Significant increase in computational capability will dramatically reduce design cycle for new combustion technologies and new fuels**
  - Simulations with higher physical fidelity, particularly chemistry at high pressure
  - More predictive science-based turbulence/ chemistry interaction models
    - Address new mixed-mode regimes driven by efficiency and emissions
    - Differentiate effects of alternative fuels
- **Co-design center is focusing on direct numerical simulation methodologies**
  - Science base for novel fuels at realistic pressures
  - Not addressing complexity of geometry in engineering design codes







ExaCT Goal

- Goal of combustion exascale co-design is to consider all aspects of the combustion simulation process from formulation and basic algorithms to programming environments to hardware characteristics needed to enable combustion simulations on exascale architectures
  - Interact with vendors to help define hardware requirements for combustion simulation at the exascale
  - Interact with vendors and DOE CS community (X-stack) on requirements for programming environment and software stack needed for combustion simulation
  - Interact with applied mathematics community on mathematical issues related to exascale combustion simulation

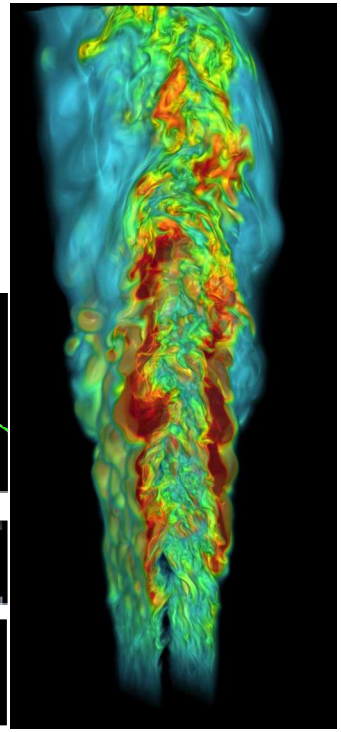
- Combustion is a surrogate for a much broader range of multiphysics computational science areas

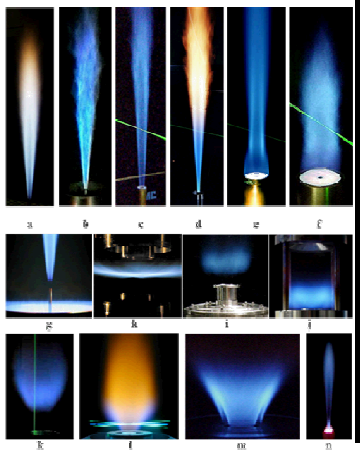
Petascale codes provide starting point for co-design process

- S3D
  - Compressible formulation
  - Eighth-order finite difference discretization
  - Fourth-order Runge-Kutta temporal integrator
  - Detailed kinetics and transport
  - Hybrid parallel model with MPI + OpenMP
  - OpenACC pragmas for GPU's (Titan CAAR)
- LMC
  - Low Mach Number model that exploits separation of scales between acoustic wave speed and fluid motion
  - Second-order projection formulation
  - Detailed kinetics and transport
  - Block-structure adaptive mesh refinement
  - Hybrid parallel model with MPI + OpenMP

Expectation is that exascale will require new code base

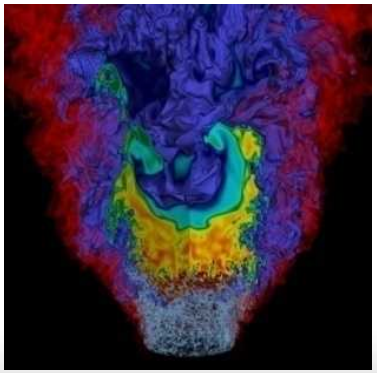
S3D simulation of HO_2 ignition marker in lifted flame





Laboratory scale flames

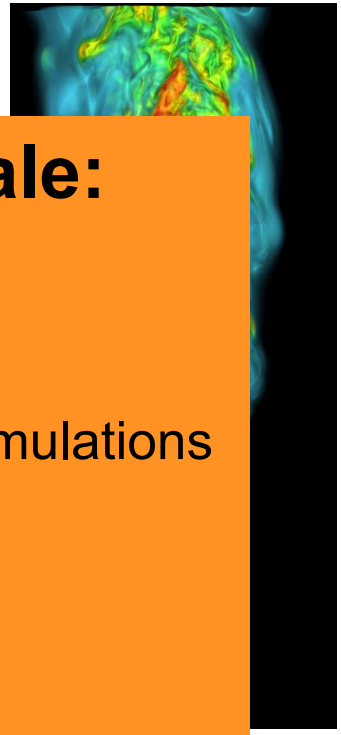
LMC simulation of NOx emissions from a low swirl injector



Petascale codes provide starting point for co-design process

- S3D
  - Compressible formulation

S3D simulation of HO_2 ignition

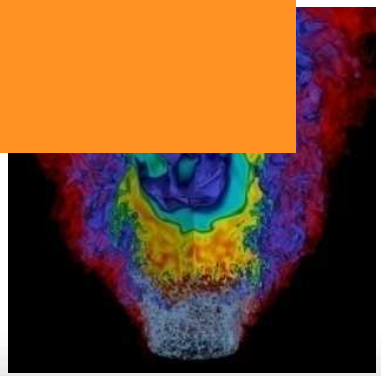


Target computational capability at exascale:

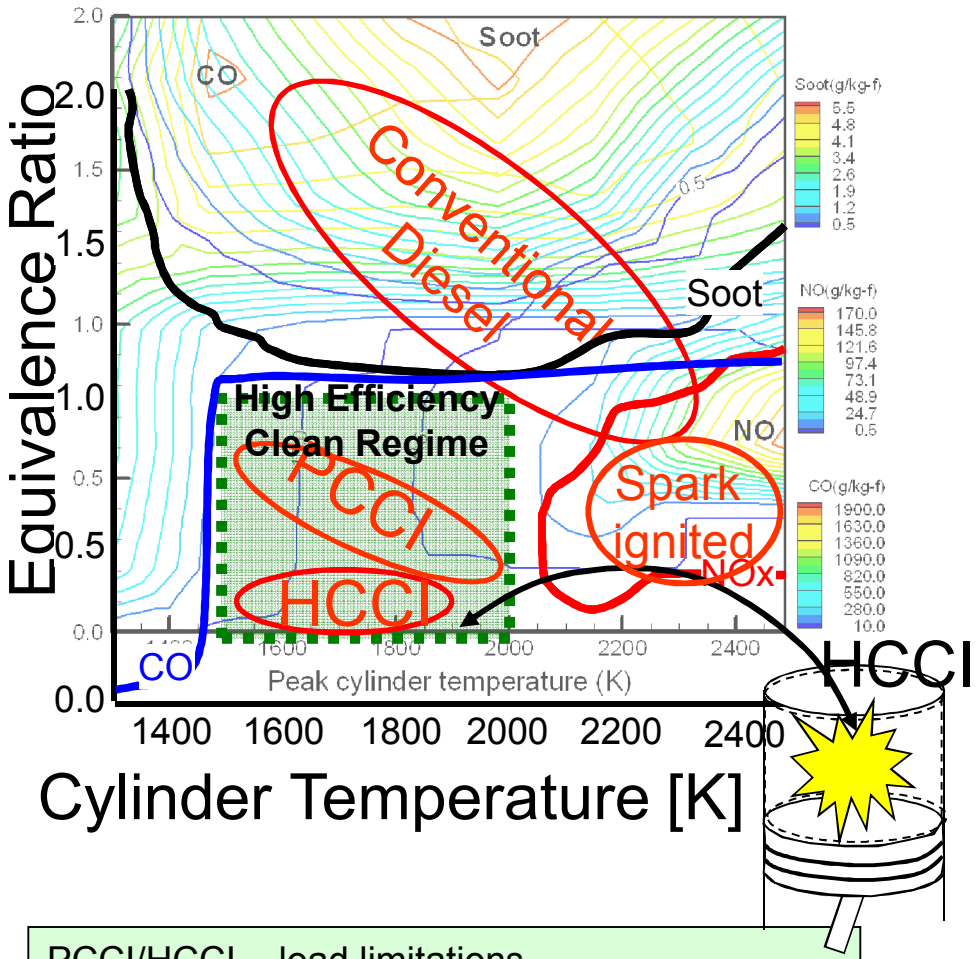
- High-fidelity physics
  - Detailed chemical kinetics
  - Multicomponent species transport
- Supports both compressible and low Mach number formulations
- Block-structured adaptive mesh refinement
- Higher-order spatial discretizations
- Higher-order temporal integration
- **Support for embedded UQ**
- ***In situ* analytics**

Expectation is that exascale will require new code base

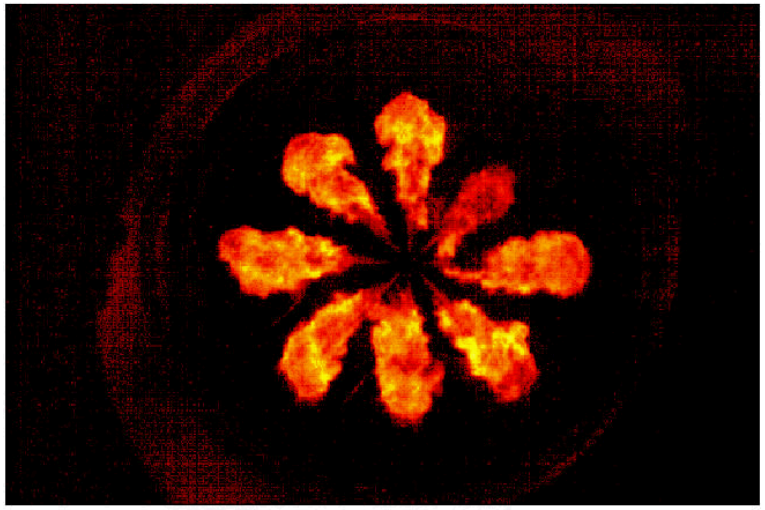
LMC simulation of NOx emissions from a low swirl injector



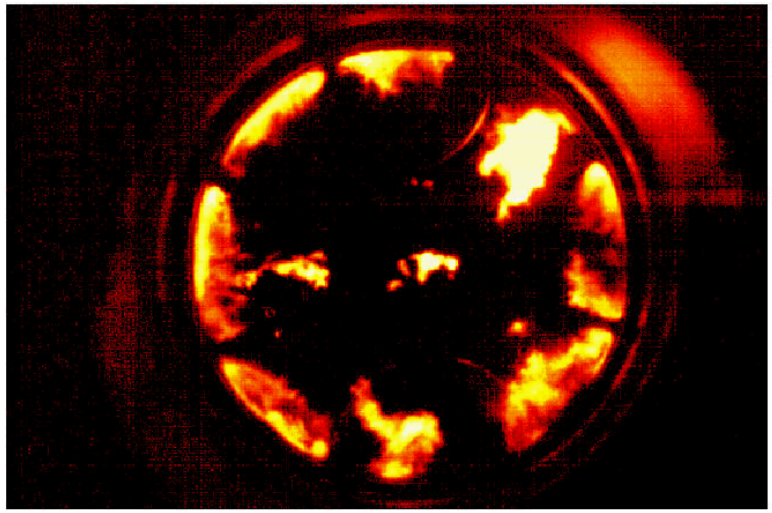
Exascale Target: Advanced Engine Combustion Regimes



Conventional diesel *

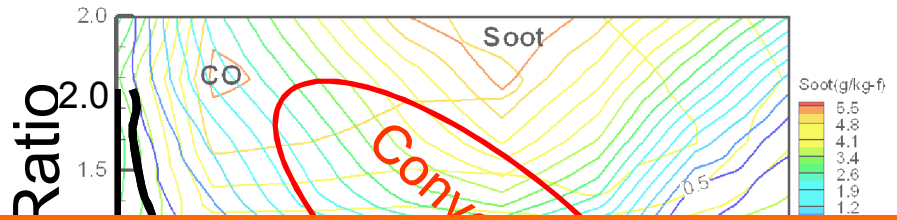


Early injection PCCI

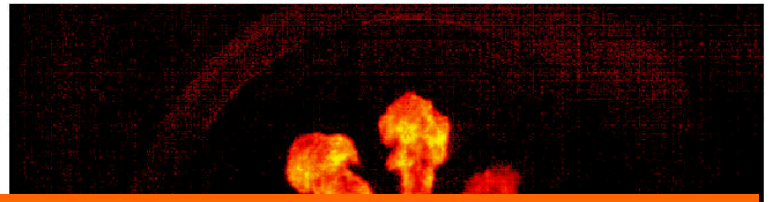


PCCI/HCCI – load limitations
Requires precise charge preparation and combustion control mechanisms
(for auto-ignition and combustion timing)

Exascale Target: Advanced Engine Combustion Regimes

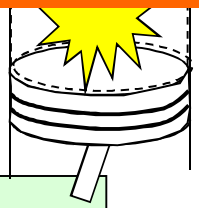


Conventional diesel *



- **Relevant turbulence, pressure, temperature (Re_t = 4300, 30-60 atm, 750-2000K)**
- **Domain and grid size**: 5cm^3, 5 micron grid,
- **Size of state**: 10^{12} grids x 100 dof x 8 bytes/dof ~ **1 PB**
- **High water memory use**: ~ 3 x (size of state) ~ **3 PB**
- **Number of time steps**: 6 ms/5 ns timesteps = **1.2e6 steps**
- **Total run time**: 1380e-06 s/time step/grids x 1.2e6 time steps x 10^{12} grids / 3600s/hr = 0.46e^{12} cpu-hrs (**20 days at billion way concurrency**)
- **Total amount of data for analysis**: **1.0 exabyte**

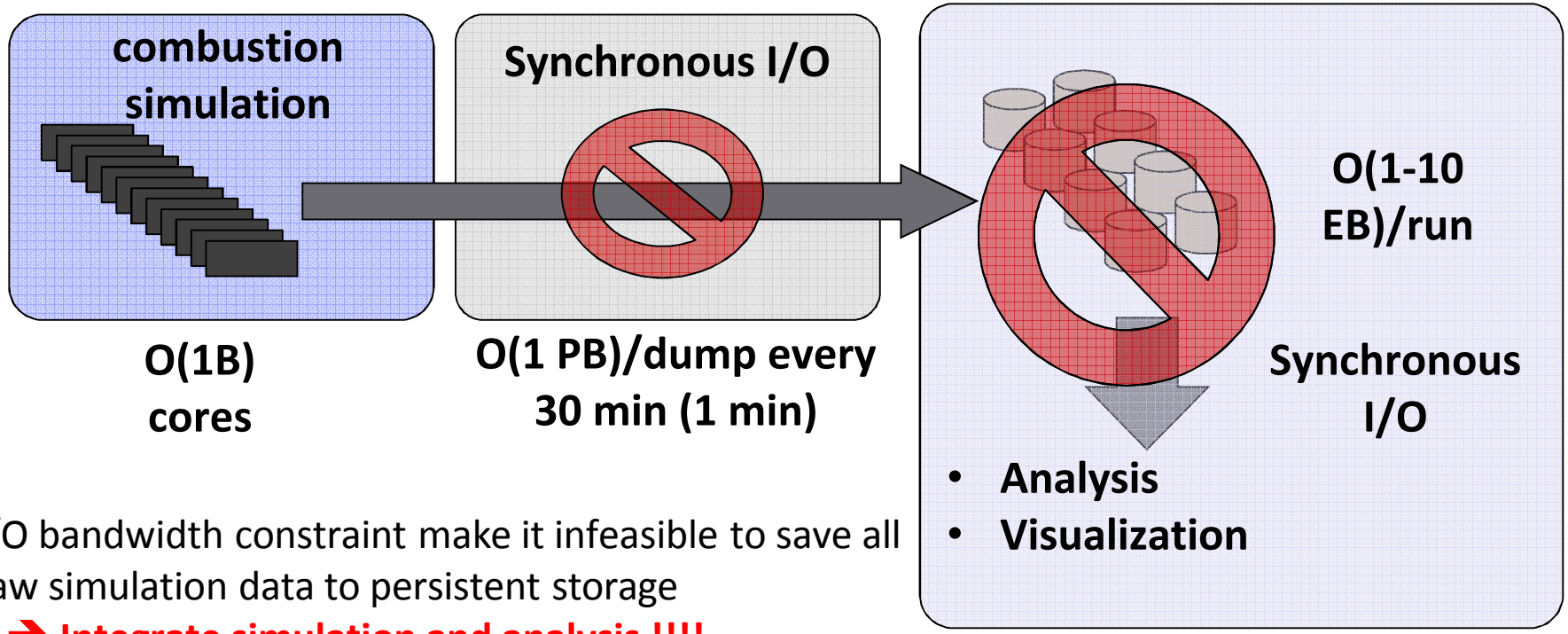
Cylinder Temperature [K]



PCCI/HCCI – load limitations
Requires precise charge preparation and combustion control mechanisms
(for auto-ignition and combustion timing)

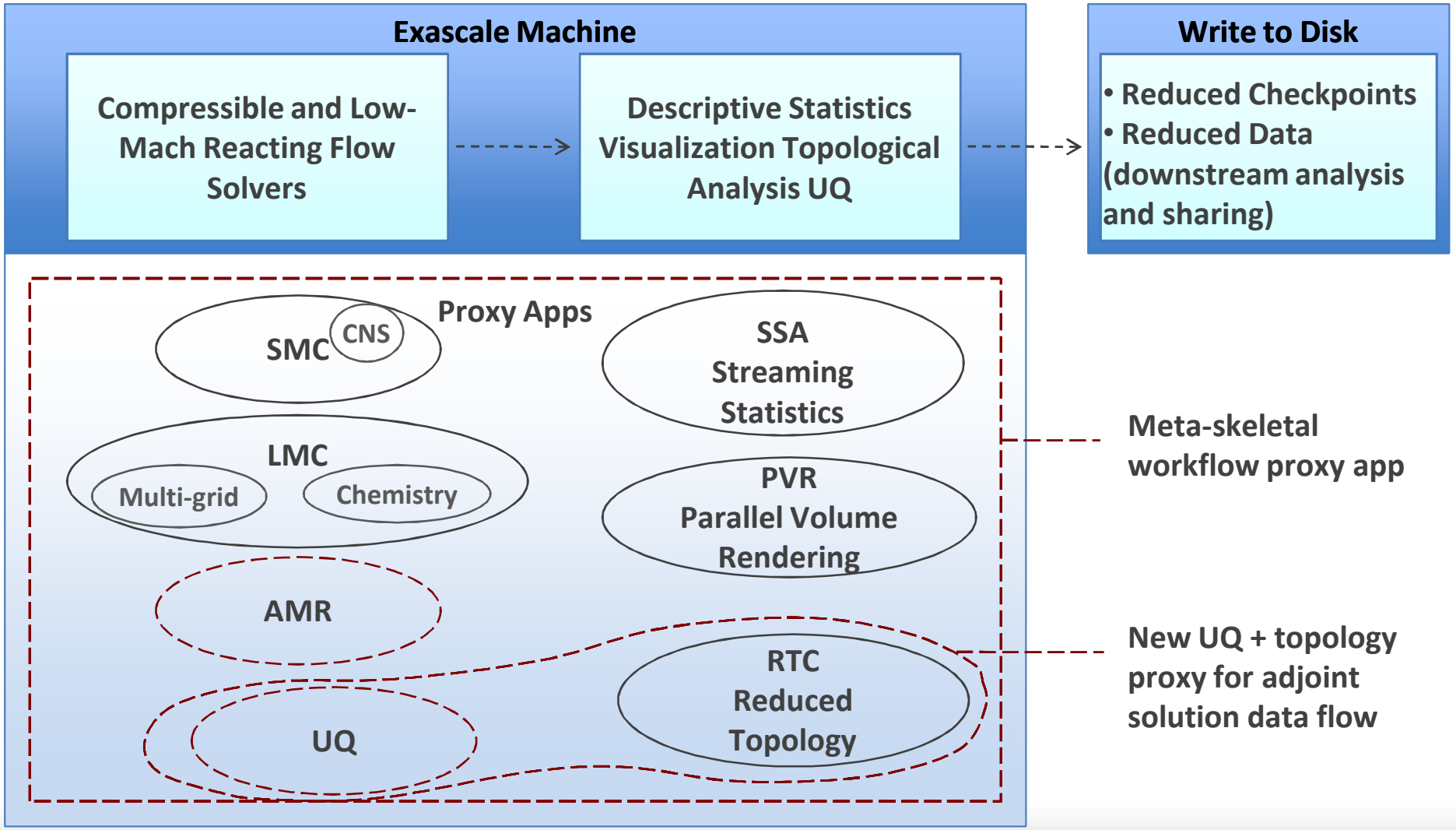
Petascale Workflow Model Won't Scale

Performing the simulation is not enough – need to analyze results



- I/O bandwidth constraint make it infeasible to save all raw simulation data to persistent storage
  - ➔ **Integrate simulation and analysis !!!!**
- Challenge: co-design a workflow that supports smart placement of analyses, visualization and UQ, tracking large graphs, reducing checkpointing size with *in situ* analytics
- Potential to reduce amount of data by 1000-fold

Overall Workflow Captured Through Proxy Apps



Solver Algorithmic Elements

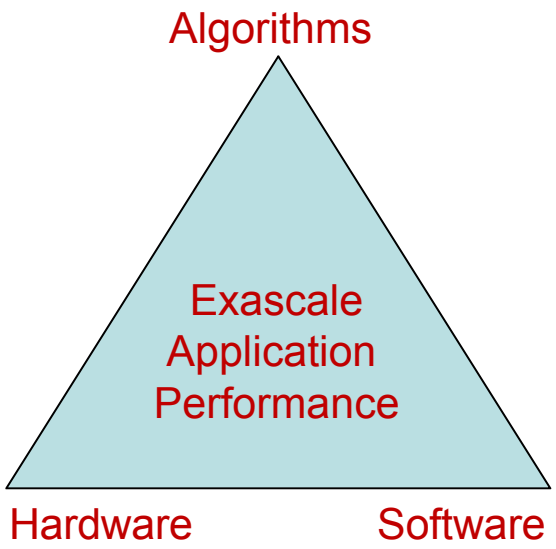
- Stencil operations on blocks of data
  - Stencils of different orders of accuracy 2^{nd} – 8^{th}
    - Advection
      - Advection stencils can include conditionals (merge)
    - Diffusion
    - Projection
  - Special cases – not pure
    - Boundary conditions
    - Coarse-fine interfaces
- Single point computationally intense physics
  - Thermodynamics
  - **Transport coefficients**
  - **Chemical rate evaluation**
- Multigrid linear solvers
  - Stencil operation but on increasingly smaller grids as one moves deeper into the V-cycle
    - Diffusion is relatively easy
    - Projection systems are more difficult
- Coarse-grained irregular work associated with AMR

Hardware/Software Co-Design Questions

- **Computational Throughput**
  - What is the instruction mix, and should we utilize chip area for vector units for special functions?
- **Registers**
  - How many registers are needed to capture scalar variables to avoid cache spills?
- **Memory Bandwidth**
  - How sensitive is the application to the memory bandwidth?
- **Cache**
  - To what extent can on-chip memory (cache) filter memory bandwidth?
  - Can software optimizations change the memory bandwidth requirements?
- **Memory Capacity**
  - What is the memory footprint of the application, and can NVRAM be used to increase memory capacity?
- **Network Interface**
  - How sensitive is the application to network interface (NIC) characteristics in terms of latency and injection bandwidth?
- **Interconnect Topology**
  - Which interconnect topologies offer the highest performance for the lowest cost (fat-tree, dragonfly, tori)
  - How does job placement affect performance?

Co-Design Methodology

- Measurement alone is not sufficient
  - Measurement can be quirky
  - Gives performance of today's algorithms on today's hardware using today's software stack
  - Difficult to make precise extrapolations
  - Difficult to assess algorithm variations
- Analytic performance model
  - Algorithm variations
  - Architectural features
  - Identify critical parameters
  - Predict trends
- Validate performance with hardware simulators / measurements
  - Confirm key predictions
  - Model what can't be predicted analytically

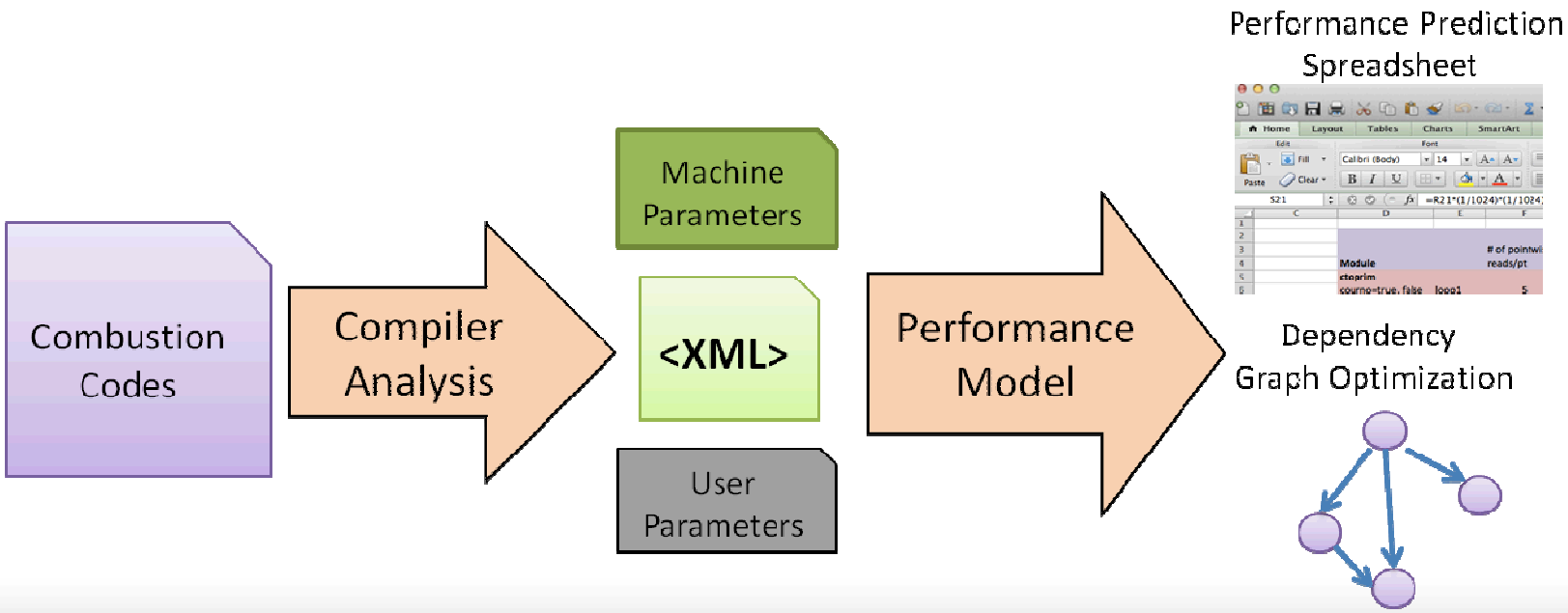


Details of the Methodology

- Developed notation for concise representation of algorithm characteristics
  - Captures floating point operations, loads and stores
  - Captures data dependencies across a range of granularities
  - Captures communication patterns
- Provides a mechanism for a formal definition of a computational skeleton
  - Computable from static code analysis (using ROSE)
  - Computable from mathematical description of algorithm
- What can one do with this description
  - Provides input for analytic performance model
  - Can be compared to measured performance on existing systems
  - Provides guidelines for identifying hardware simulation cases and interpreting results for potential new architectures (SST micro / vendor tools)
  - Provides baseline for achievable performance (Roofline model)
  - Defines “language” for driving network simulator (SST macro)
- For dynamic problems (AMR) use execution trace to define control flow

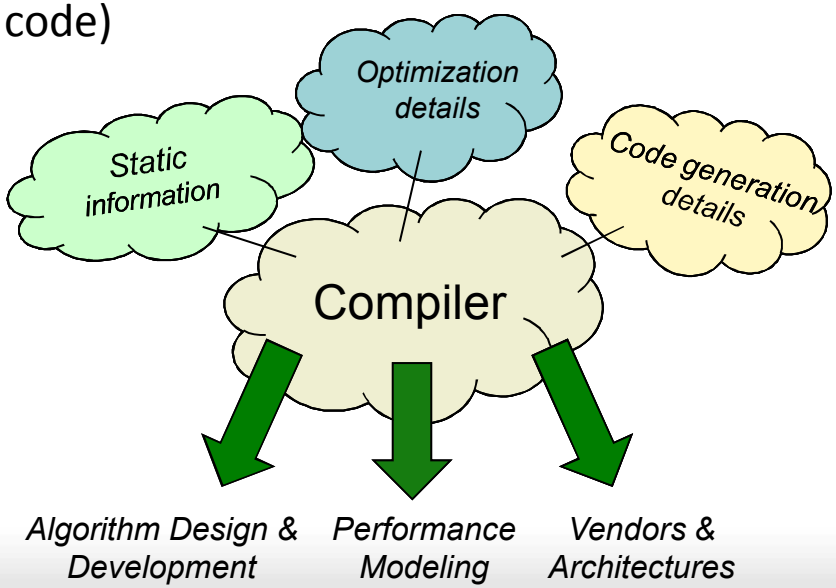
Performance Modeling Tool Chain

- Automatically predict performance for many input codes and software optimizations
- Predict performance under different architectural scenarios
- Much faster than hardware simulation and manual modeling
- Includes cache model to capture the working set reuse

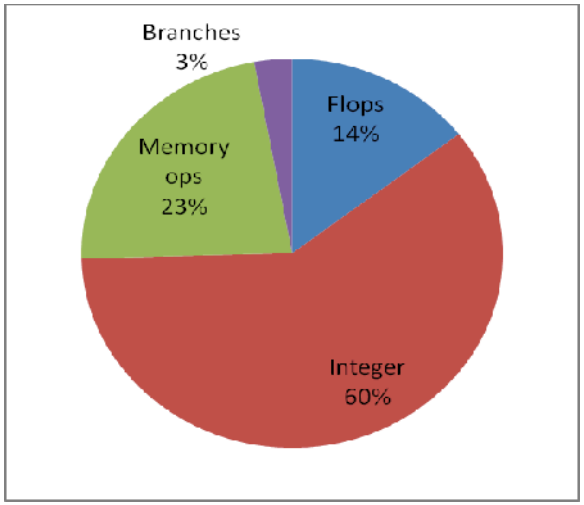


Leveraging the Compiler as a Co-Design Tool

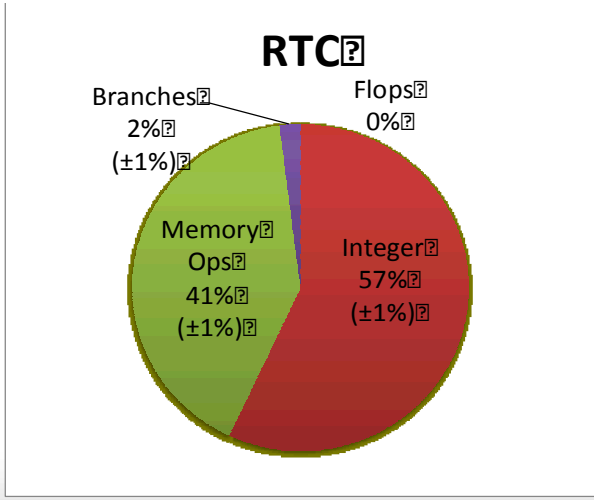
- Leveraging compiler infrastructure allows us to:
  - Characterize aspects of codes in a *hardware independent* and *customized* fashion tailored to co-design questions/goals
  - Drive modeling and architecture analysis and more effective communication with the broad co-design team (including vendors)
  - Explore *characteristics* in significantly less time than relying on architecture simulation (especially for large/complex code)



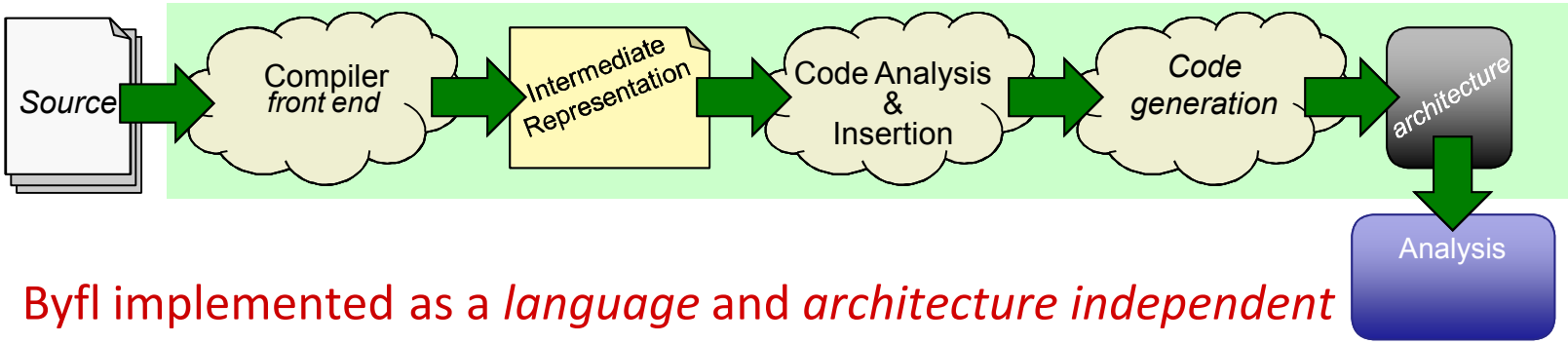
SMC Characterization



Topology Proxy Characterization



Example: Byfl – Compiler-Driven Dynamic Software Performance Counters



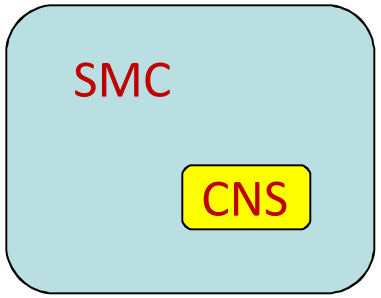
- Byfl implemented as a *language* and *architecture independent* middle-stage compiler pass
  - Analyzes code and inserts *targeted* software-counters into code
  - Supports Fortran, C, C++ (via any GNU toolchain frontend)
  - Builds upon LLVM -- a common intermediate representation (IR)
- Providing answers (estimates) to some initial questions from the vendors:
  - *What is the memory bandwidth/flop?*
  - *What is the instruction mix?*
  - *What is the read/load to write/store ratio of data?*

LLVM web site: http://llvm.org

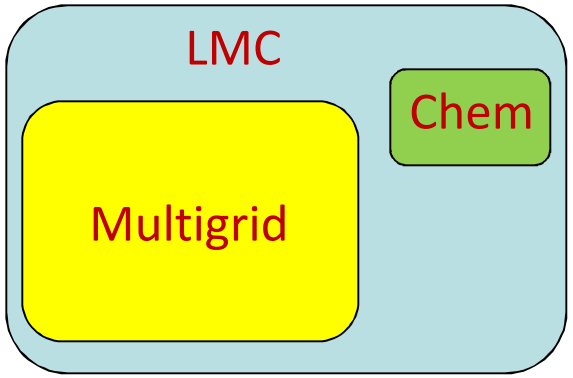
Open source: http://github.com/losalamos/Byfl

Proxy Applications - Solver

- Uniform grid compressible flow proxies
  - SMC – Multicomponent, reacting, compressible Navier-Stokes
    - Detailed chemistry and transport
    - Eighth-order in space, low storage third-order Runge Kutta in time
    - Fully explicit -- Time step limited by acoustics / chemical time scales
    - Hybrid implementation with MPI + OpenMP
  - CNS – Same stencil operations as SMC
    - SMC without species
    - Acoustic time step limitation



- Low Mach number + AMR
  - LMC – Exploit separation of scales between different processes when appropriate
    - Projection-based discretization strategy
    - Second-order in space and time
    - Semi-implicit treatment of advection and diffusion
    - Time step based on advection velocity
    - Stiff ODE integration methodology for chemical kinetics
    - Hybrid implementation with MPI + OpenMP
  - Multigrid solver -- > 50% of time in LMC
  - Chemical kinetics integration –14% of time in LMC



Proxy Machines

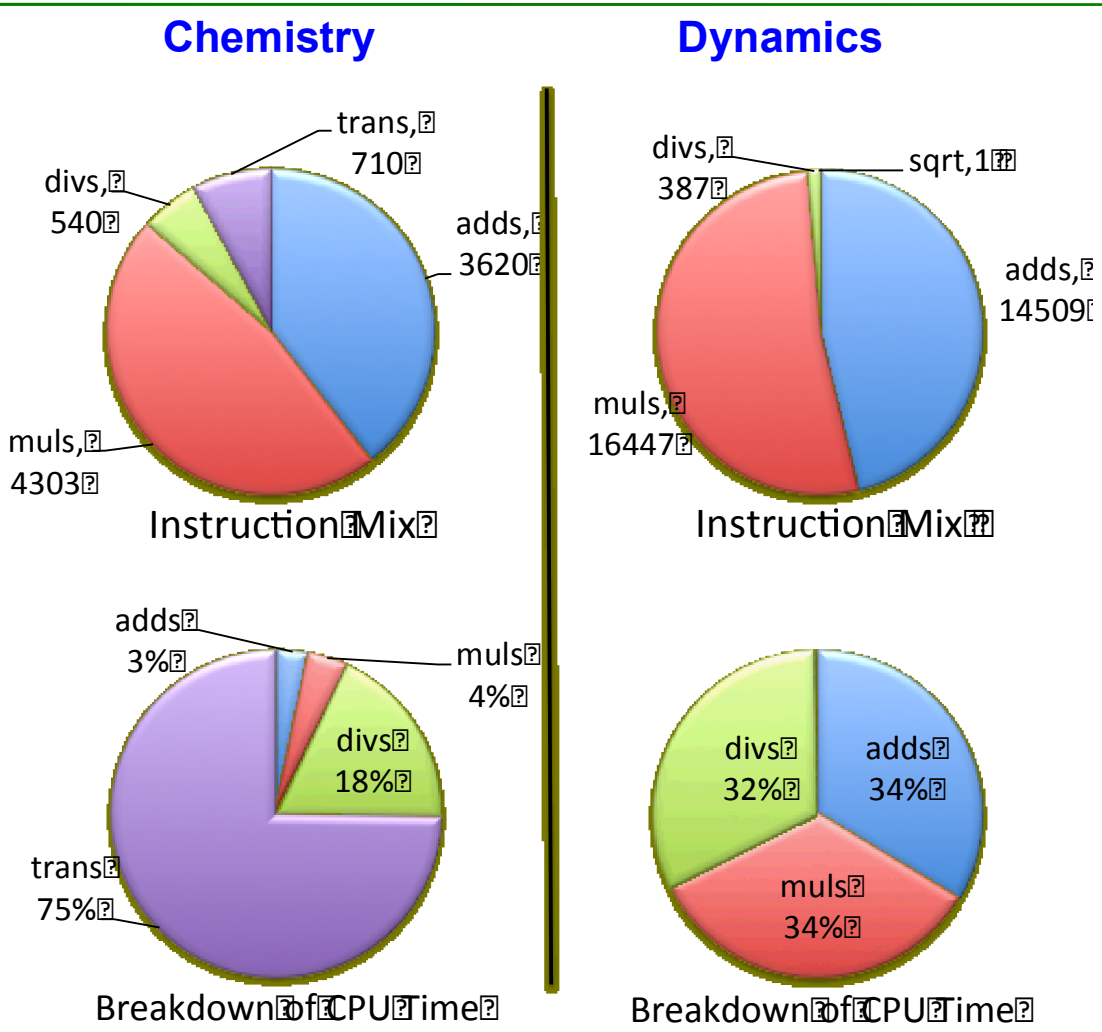
		Hopper	exaNode1	exaNode2	exaPIM
Memory BW	TB/s/node (chip)	0.05	1	4	TBD (higher)
Memory Size	GB/node (chip)	32	256	32	TBD (lower)
Flops	TF/node (chip)	0.03	10	10	10
# of Cores	Cores/chip	6	~1k	~1k	TBD
# of Chips	Chips/node	4	1-2	1-2	TBD
Cache (last L)	$/core (MB)	1	32	32	NA
Cache L1	$/core (KB)	64	16-64	16-64	NA
NIC BW	GB/s	1	100	400	TBD
NIC Latency	microseconds	1	0.4	~0	TBD
Registers	KB/chip				

- exaNode1 and 2 are many core architectures
- exaNode1 uses commodity NIC and memory technology
- exaNode2 uses custom on-board NIC and faster memory technology
- exaPIM: Processing Near Memory or Processor In Memory

Hardware/Software Co-Design Questions

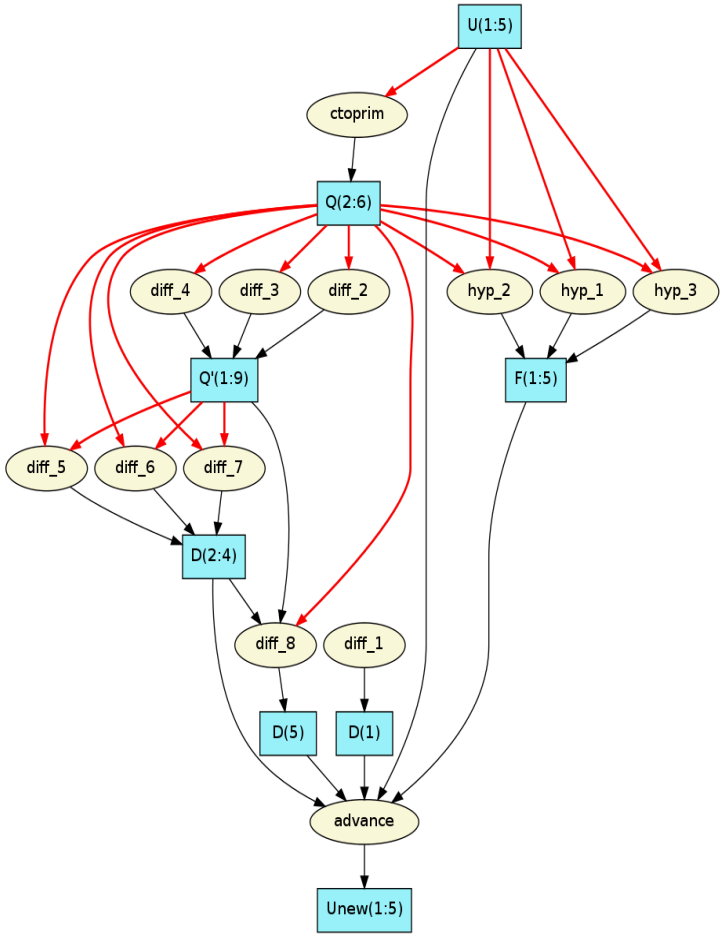
- ✓ **Computational Throughput**
  - What is the instruction mix, and should we utilize chip area for vector units for special functions?
- **Registers**
  - How many registers are needed to capture scalar variables to avoid cache spills?
- ✓ **Memory Bandwidth**
  - How sensitive is the application to the memory bandwidth?
- **Cache**
  - To what extent can on-chip memory (cache) filter memory bandwidth?
  - Can software optimizations change the memory bandwidth requirements?
- **Memory Capacity**
  - What is the memory footprint of the application, and can NVRAM be used to increase memory capacity?
- **Network Interface**
  - How sensitive is the application to network interface (NIC) characteristics in terms of latency and injection bandwidth?
- ✓ **Interconnect Topology**
  - Which interconnect topologies offer the highest performance for the lowest cost (fat-tree, dragonfly, tori)
  - How does job placement affect performance?

Basic Characterization of SMC Proxy



Even though transcendentals and division ops might be low in count, they can dominate the CPU time

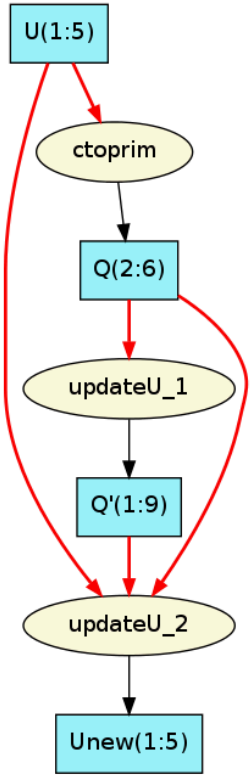
CNS Dependency Graph



Baseline

2.8 GB/sweep

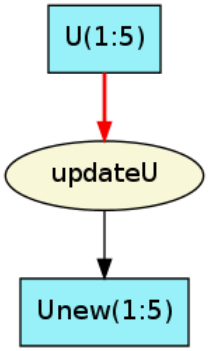
1.72 Bytes/Flop



Simple Fusion

1.5 GB/sweep (–45%)

0.93 Bytes/Flop

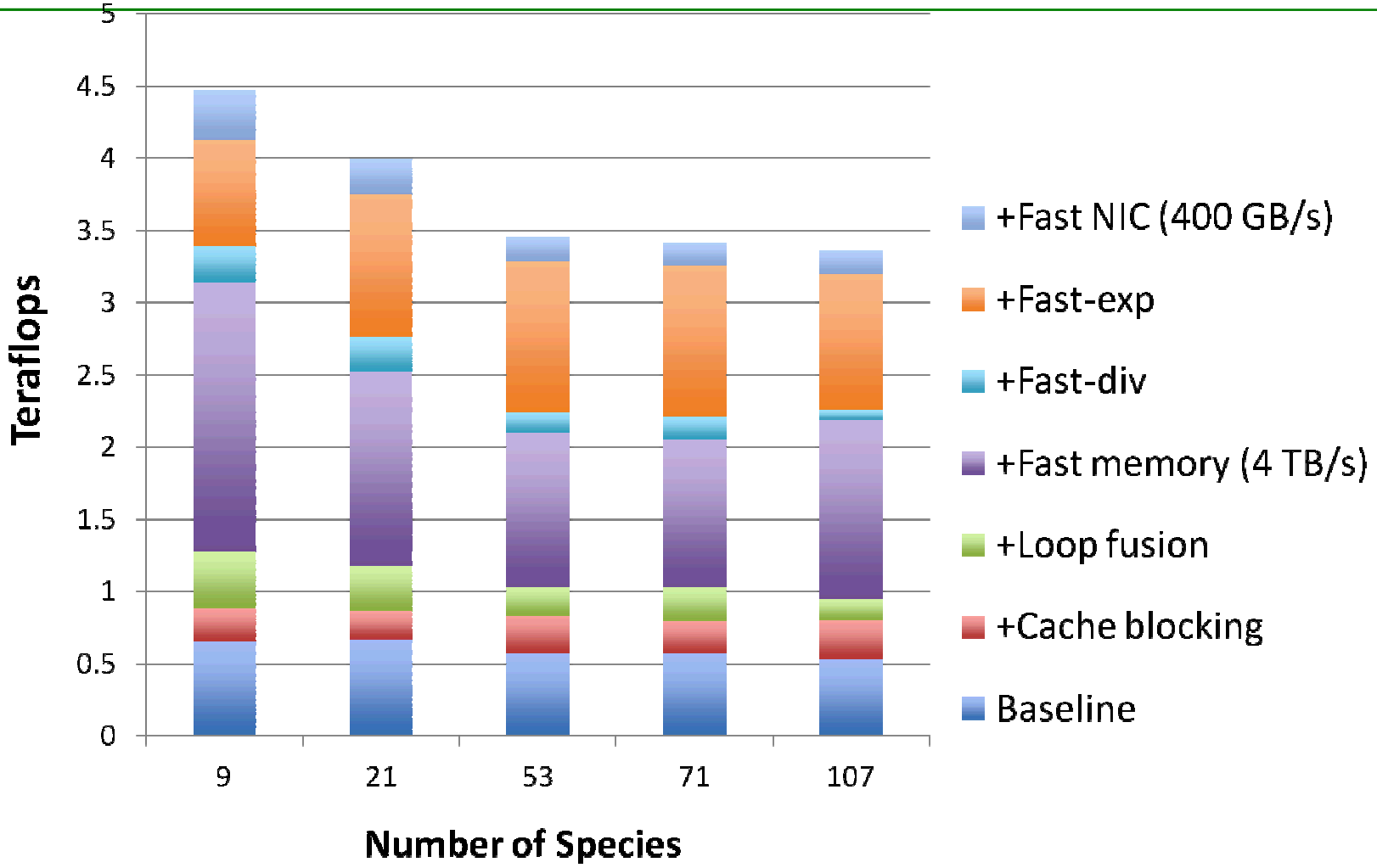


Aggressive Fusion

0.5 GB/sweep (–84%)

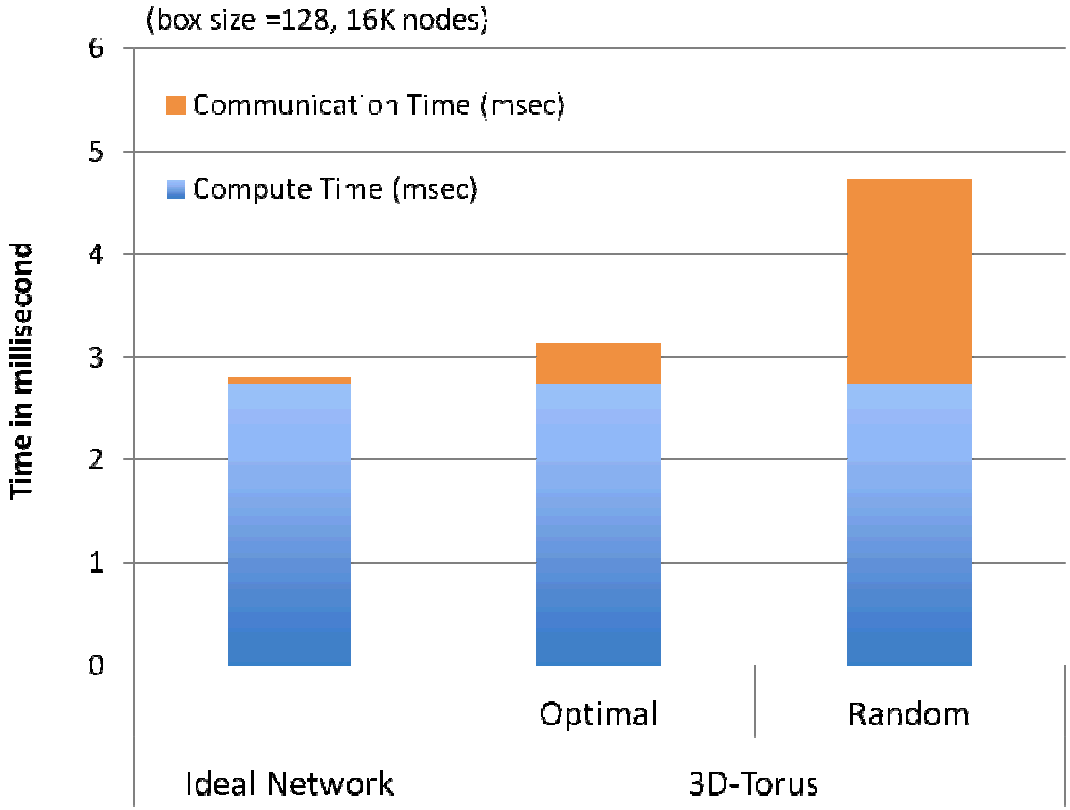
0.28 Bytes/Flop

Estimated performance improvements



Neither software optimizations alone nor hardware optimizations alone will not get us to the exascale, we have to apply both.

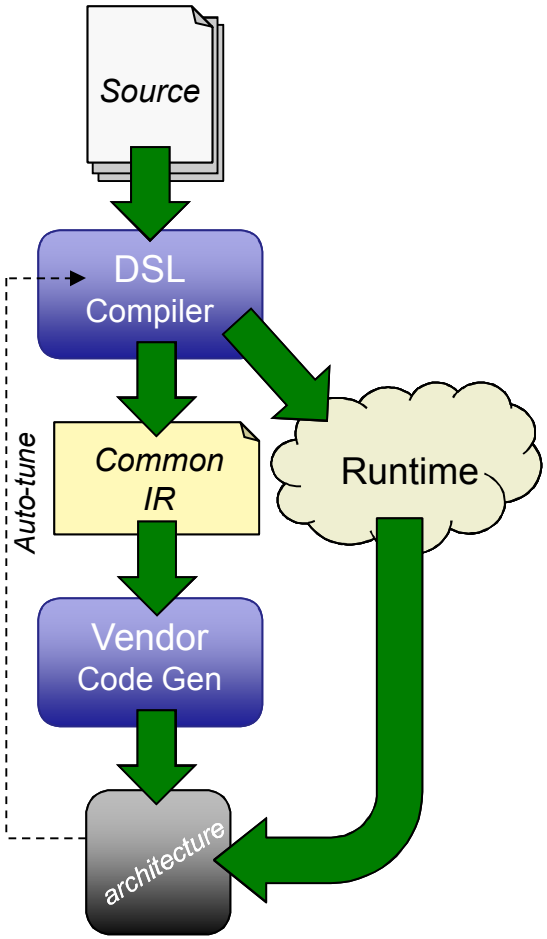
Impact of Data Layout on Performance



- Previous analysis assumes ideal network behavior
- Use SST macro to model contention on realistic network
- Results illustrate importance of locality on performance
  - Internode data placement needs to be topology aware

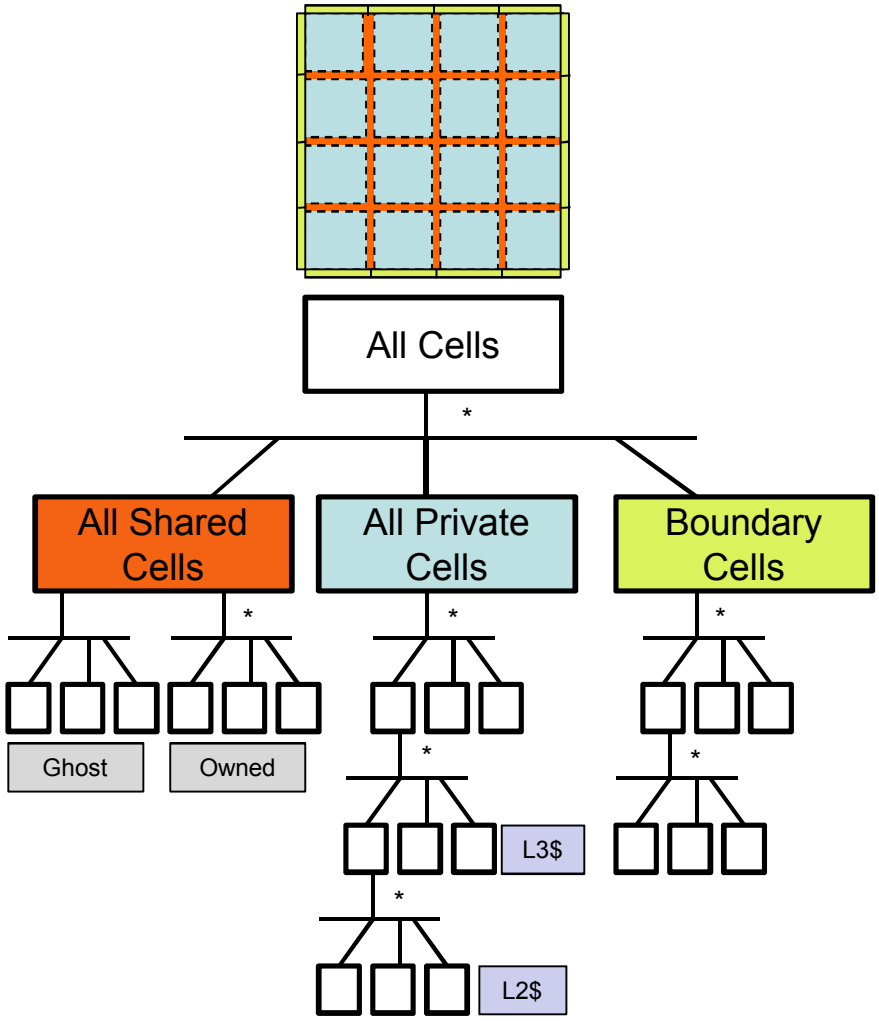
Domain-Specific Languages for Co-Design

- A domain-specific language (DSL) is a language of reduced expressiveness targeted at developers in a specific, focused problem domain
- Perfect fit for co-design.
  - Influenced and driven by both the domain and architecture
  - Contributes directly as an enabling technology that insulates applications from the complexity of the architecture. *Specialized software that works well with the co-designed hardware.*
- Starting points:
  - Team has previous experience in DSLs via ASC PSAP (Stanford) – unstructured CFD, DSL with LLVM (LANL)
  - *LLVM Common IR* for vendor toolchain interactions and supporting infrastructure (not building compiler from scratch). Using as FastForward interface for code generation (with AMD and NVIDIA). Intel? IBM?
  - Orion Stencil DSL
  - Memory-hierarchy-aware programming model and runtime (*Legion – leveraging initial DARPA funding*)
  - Terra Language (DSL building blocks)



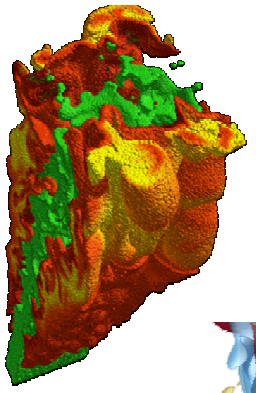
Legion: Programming Locality and Independence

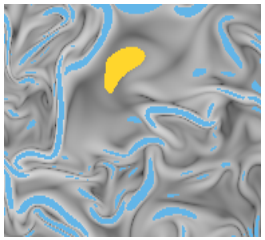
- Programming based on logical ***regions*** to describe organization of data and to explicit (first class values)
  - Tasks are defined on regions, access to regions with privileges (*read-only, read-write, reduce*) and coherence (*exclusive, atomic, etc.*)
  - **Mapping** of tasks and regions onto processors and memories.
    *Programmable from both **domain** and **architecture** points of view*
- Progress:
  - Initial, basic AMR implementation
  - CPU, GPU and cluster support
  - SC12 paper, code released to center participants
  - S3D implementation coming soon!

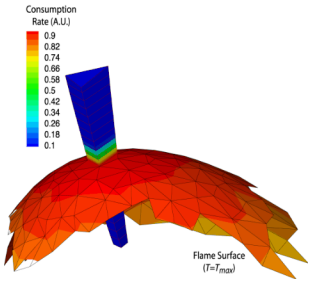


Extracting Knowledge from Simulation Requires Range of Analysis With Different Instruction Mixes

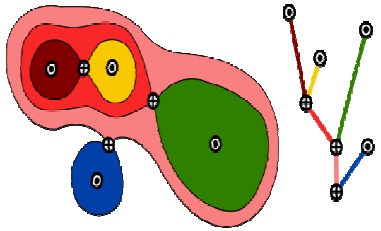
- Visualization
  - In situ multi-variate volume and particle rendering
  - Lagrangian particle querying and analysis
- Topological Segmentation and Flame/Flow Geometry
  - Topological segmentation:
    - Contour trees
    - Morse-Smale complex
    - Time tracking
    - Scalar field comparison
  - Distance field (level set)
  - Shape analysis
- Statistics
  - Filtering and averaging (spatial and temporal)
  - Statistical moments (conditional)
  - Statistical dimensionality reduction (joint PDFS)
  - Spectra (scalar, velocity, coherency)

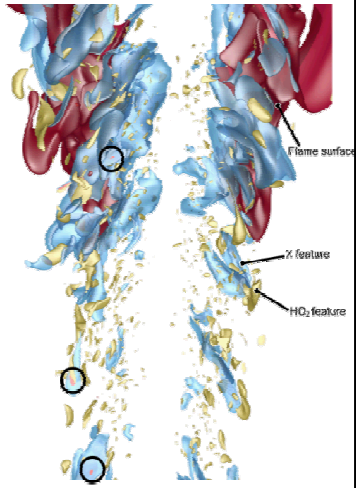


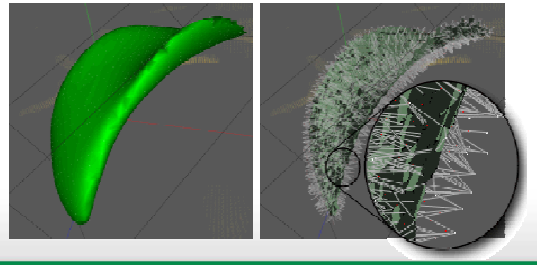




Flame-centric control volume analysis

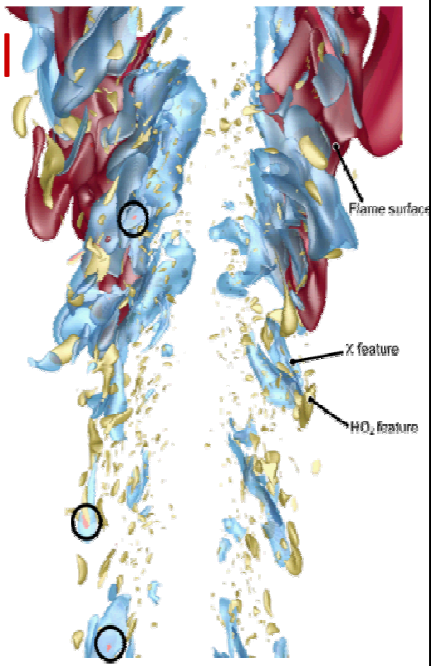






Intrusive Uncertainty Quantification - Chemistry Models are Source of Error in DNS:

- ***Motivation***: Inform chemical kineticists and chemical engineers in the fuels industry (BP, Exxon) on what chemical properties need to be pinned down more accurately for optimal utilization of a given fuel
- Embedded chemistry model source of uncertainty
  - Reaction rates
  - Missing reactions
  - Transport coefficients
- Combustion intermittency characterized by space-time localized phenomena of interest, tractable for UQ
- Solve adjoint equations backward in time: need the primal state at all times (data management opportunity for co-design)



At exascale, the model of compute first, analyze later will be infeasible: SDMA challenges at exascale

- The widening gap between compute power and available I/O rates will make it infeasible to save all necessary data for post processing
  - Data analysis and/or data triage must be performed on the fly
  - "On-system" workflow architectures must be explored
- Analysis codes have markedly different characteristics compared to simulation codes challenging current hardware and software stacks
  - Can require data-dependent computations and global communications, which are difficult/expensive to scale
- Understanding and modeling interaction / coordination behaviors of end-to-end workflows will be critical to co-design
  - Empirical evaluations must be used, along with analytic models to explore the complete co-design space

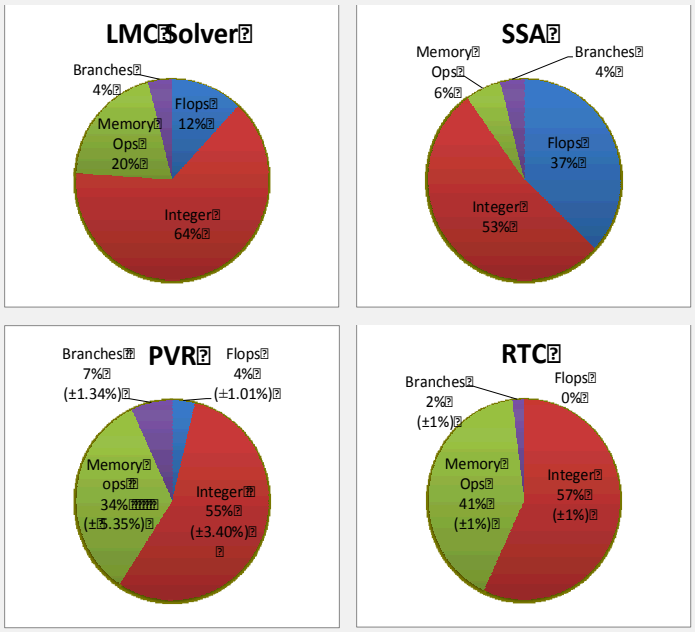
Approach to Characterize SDMA Requirements

- Develop proxy / skeletal applications for a representative set of data analyses methods and characterize machine independent characteristics, e.g. memory access patterns, communication patterns, etc.

- Define and characterize relevant workflow architectures and map (formally and empirically) the machine independent information to specific architectural choices

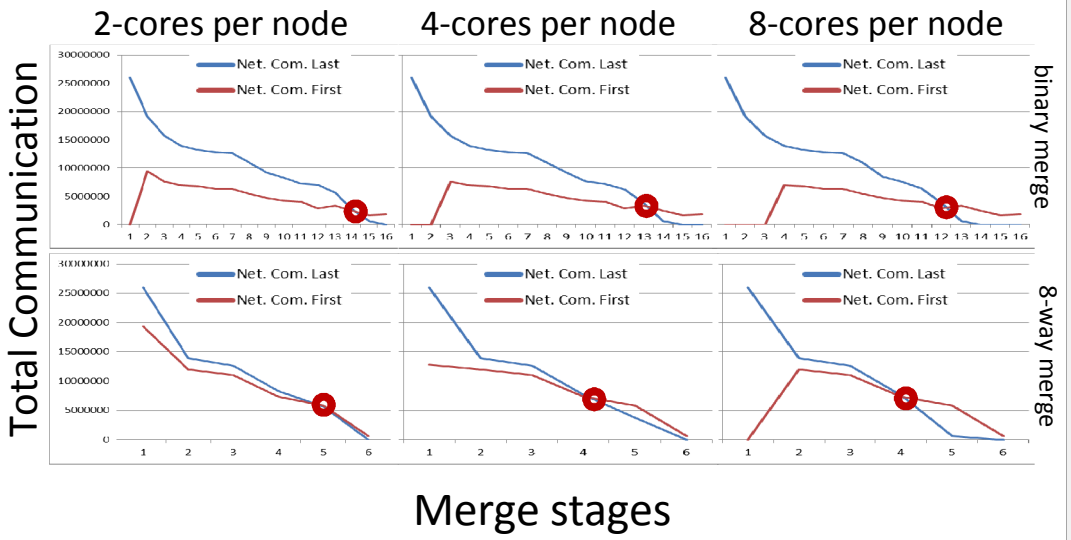
- Integrate analytics and simulation proxies to understand end-to-end performance characteristics of the combustion workflow & express this using the ***meta-skeleton*** abstraction

Initial lessons show highly heterogeneous behaviors and requirements



Analytics instruction mixes cover a wide range of behaviors

Algorithms range from Flop-free to having more Flops than solver

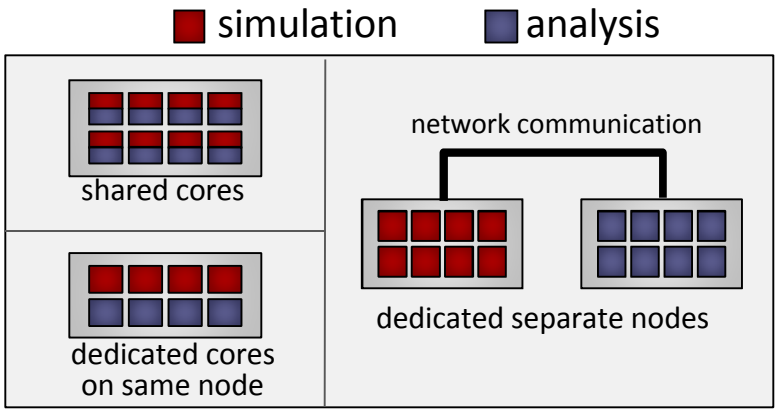


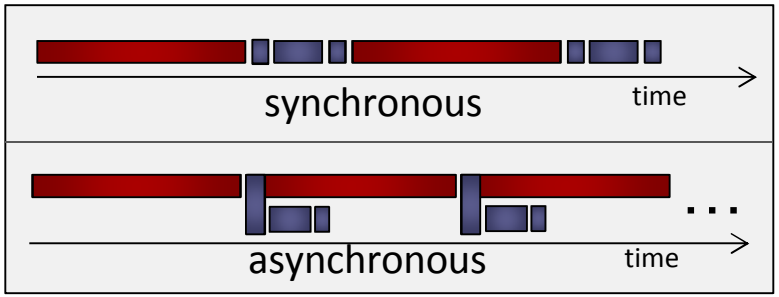
Mapping machine independent characterizations onto future workflow architectures is important

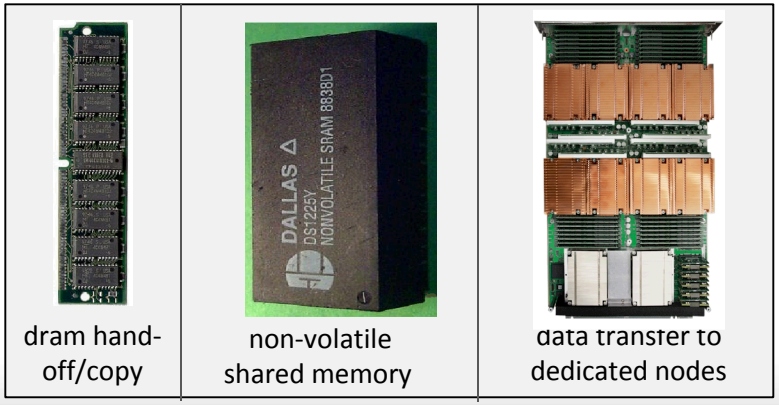
Algorithms can have a cross-over in communication costs for different workflow architectures

Rich Design Space of Workflows at Exascale

- **Location of analysis compute resources**
  - Same cores as the simulation (in-situ)
  - Dedicated cores on the same node (in-situ)
  - Dedicated nodes on the same machine (in-transit)
  - Dedicated nodes on external resource (in-transit)
- **Synchronization and scheduling**
  - Execute synchronously with simulation every n^{th} simulation time step
  - Execute asynchronously
- **Data access, placement, and persistence**
  - Shared memory access via hand-off / copy
  - Shared memory access via non-volatile near node storage (NVRAM)
  - Data transfer to dedicated nodes or external resources

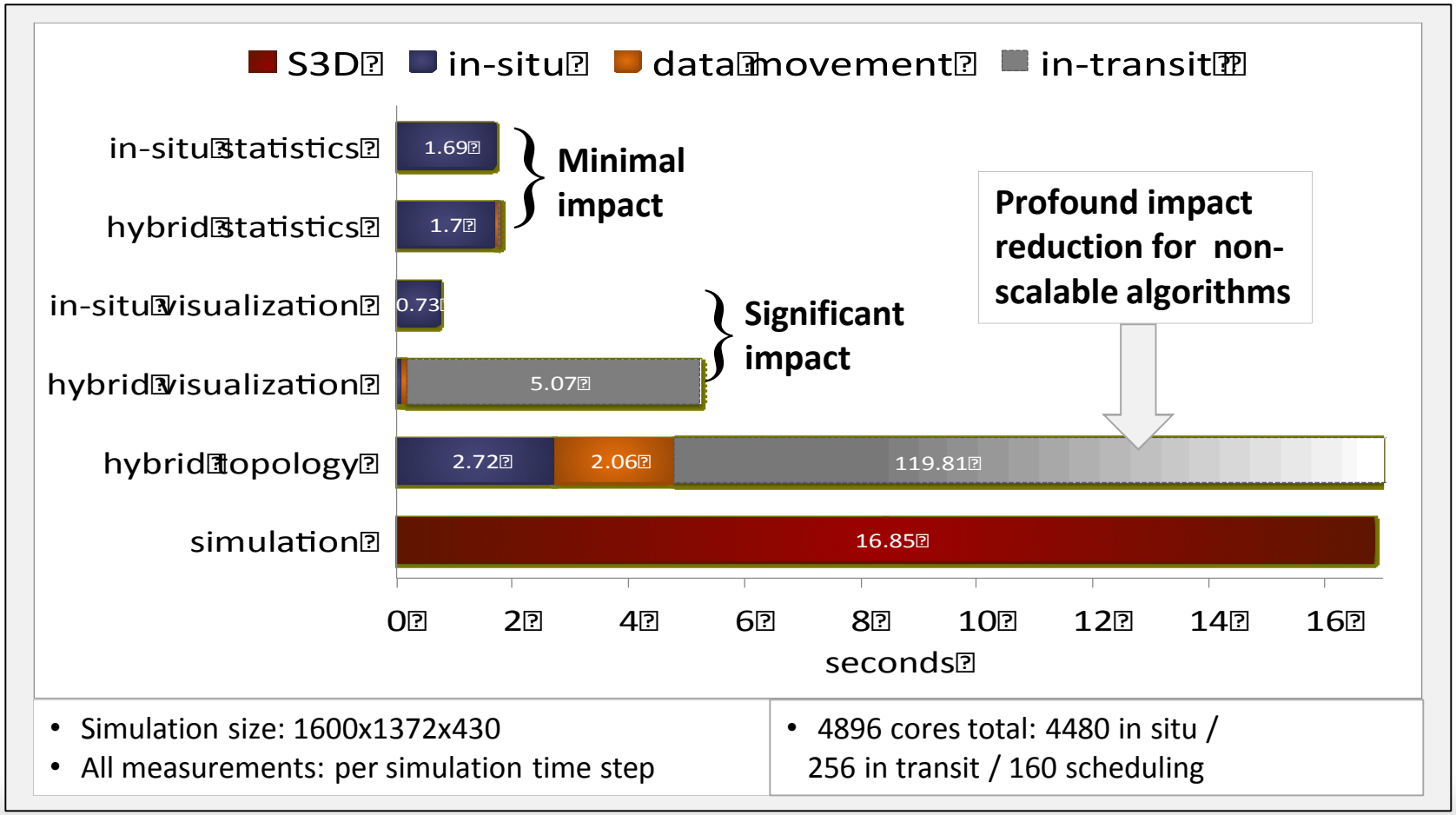






An empirical study demonstrates hybrid workflow architectures show promise for minimizing impact to the simulation

- Primary resources: execute main simulation & in situ computations
- Secondary resources: staging area for in transit computations

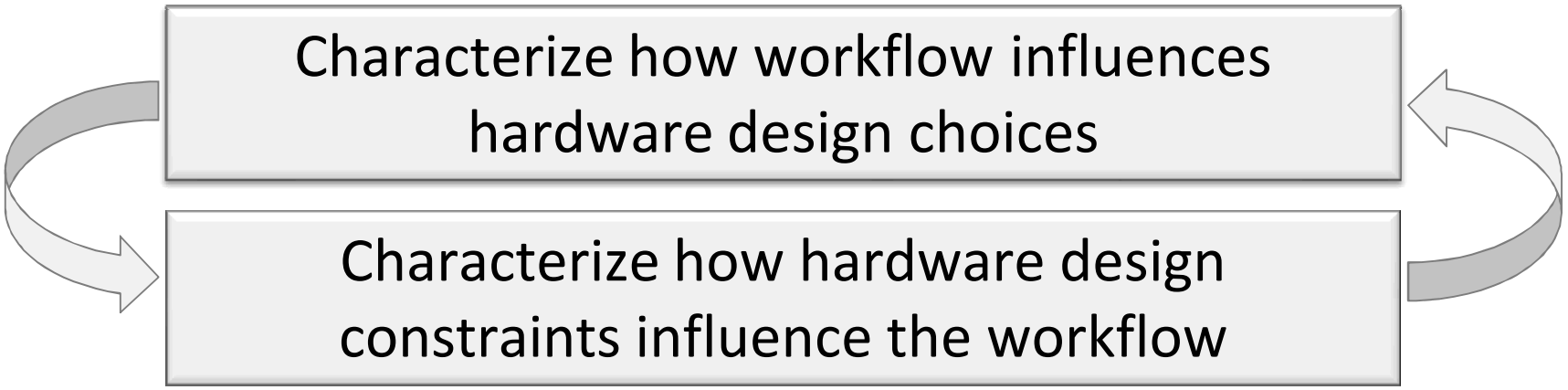


- Simulation size: 1600x1372x430
- All measurements: per simulation time step

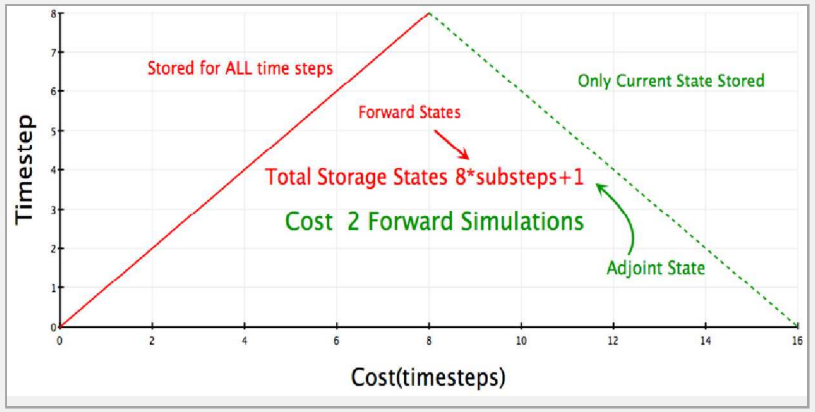
- 4896 cores total: 4480 in situ / 256 in transit / 160 scheduling

We are formulating a Meta-Skeleton abstraction to explore co-design of the end-to-end workflow

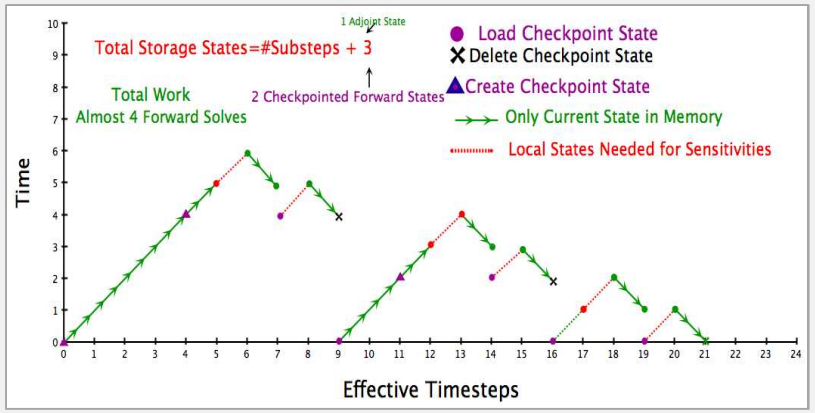
- End-to-end workflow comprises heterogeneous components
  - Individual proxy & skeletal apps are *per component* characterizations
  - The Meta-Skeleton models the dynamic interaction *between components*
- Meta-skeleton abstraction supports cross-layer co-design
  - Proxy apps, skeletal apps and kernels characterize components
  - Programming models provide abstractions to couple the components
  - Simulators map workflow onto future architectures



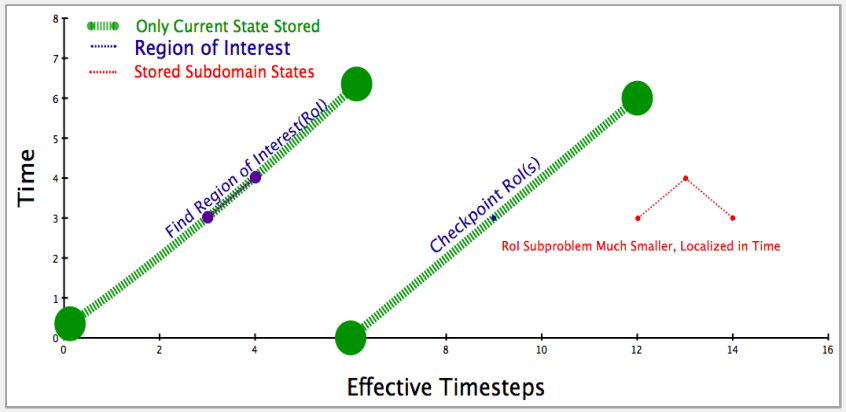
Intrusive UQ poses a co-design challenge: How to effectively manage the work and data flow

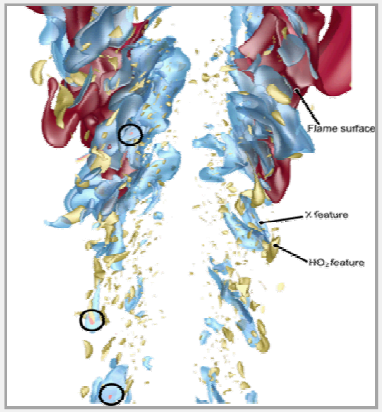


Storing entire primal state is infeasible: 5ZB



Recomputation reduces storage costs: 4 EB





Focusing computation to regions of interest (ROI) : 500PB/ROI

Adjoint UQ Proxy App available to explore trade-offs in the adjoint work & data flows

UQ Impact: Turbulent Combustion

- Effective, robust UQ methods for long-time, chaotic systems open research question.
- Space/Time localized phenomena of interest more tractable for current UQ methods(Extinction, Ignition)
- Embedded chemistry models inherent source of uncertainty
  - ❑ Reaction rates
  - ❑ Missing/under-modeled processes/reactions
  - ❑ Species concentrations, initial profiles
- Expression/reduction of basic science/prediction goals to mathematical UQ framework.
  - ❑ Leads to tractable and relevant set of UQ problems
  - ❑ Helps catalog varying sources of uncertainty

UQ Challenges: Nodal Proxy Apps

Two Classes of Intrusive UQ Proxy Apps

1. Adjoint-based Sensitivity Proxy App
   - Allows efficient sensitivity analysis for large number of parameters
   - Requires solution of adjoint problem, backwards in time.
   - Implementation impacts available fast-memory(need state(s), adjoint state).
   - Increases need for low-storage, high-order time integrator
   - CNS Adjoint Sensitivity UQ Proxy App available now.
2. Polynomial Chaos(PC) Proxy App
   - Even “S3D-like” implementation requires access to distinct realizations of solution state.
   - Increased memory access influences stencil, time-integrator, and parallel decomposition choices.
   - Proxy app allows exploration of different multirate time integration/operator split choices at the “realization”/PC coefficient level.
   - PC Proxy App available 5/12.

UQ Challenges: Data Management

Research of UQ impact on SDMA parallels proxy app thrust

1. Adjoint-based Sensitivity Analysis
   - Computation of adjoint requires solution access on all of a space-time domain "near" the phenomena of interest.
   - Leads to optimization over:
     - Available fast/slow memory
     - Costs of using slow(er) storage
     - Forward state recomputation(checkpointing)
     - Adjoint state(and resulting sensitivities) computation accuracy
2. Intrusive, Polynomial Chaos enabled UQ.
   - "S3D-like" implementation can result in significant data movement
   - AMR, multi-level methods, implicit time integration lead to even greater data challenges

Future Directions

- Develop more complete models for exascale combustion simulation
  - Architecture aware AMR algorithms
    - Incorporate interconnect topology into AMR data layout
    - Dynamic load-balancing that incorporates communication costs, data movement and dynamic machine behavior
  - Capture model for complete work flow
    - Encapsulate all aspects of analysis into performance model
- Explore hardware tradeoffs with vendors and CS collaborators
  - Refined analysis of node architectures
  - Analysis of network behavior for AMR at scale
  - Explore tradeoffs for machine balance
- Focused interaction with programming environment community to ensure that future programming models will support effective expression of methodology needed for combustion simulation

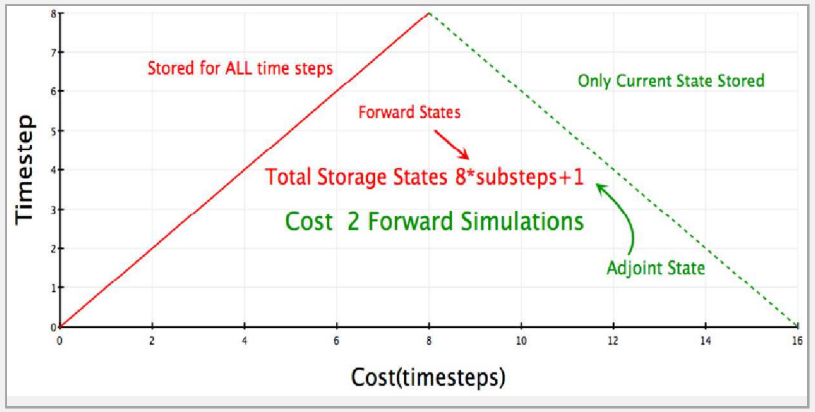
Contact

- www.exactcodesign.org
- jhchen@sandia.gov

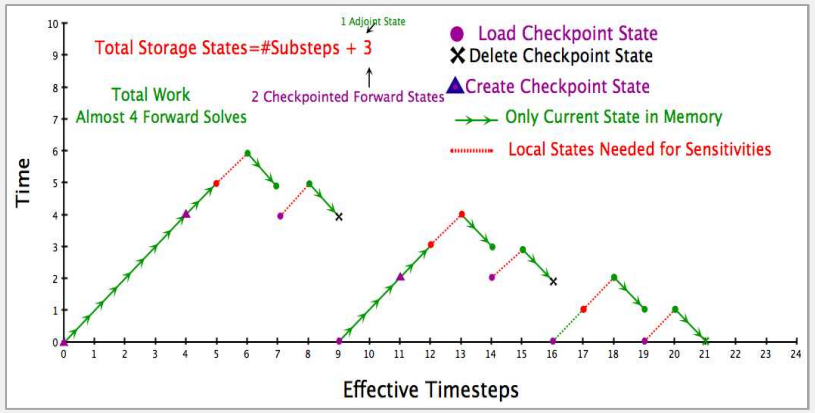
Integrating UQ into the combustion workflow introduces unprecedented challenges

- Goal: Evaluate sensitivities of quantities of interest with respect to
  - chemistry model parameters
  - modeled fields (e.g. reaction rate fields)
- The classical approach to solving this problem requires solving P+1 forward simulations, where P is the number of sensitivity evaluations
  - P can be very large (>> 1000) making this classical approach infeasible
- Adjoint approach: solve one auxiliary problem, the adjoint problem
  - Linearized about the primal solution
- Need the primal (forward) solution to solve the adjoint problem
  - Catch: adjoint problem has reverse causality

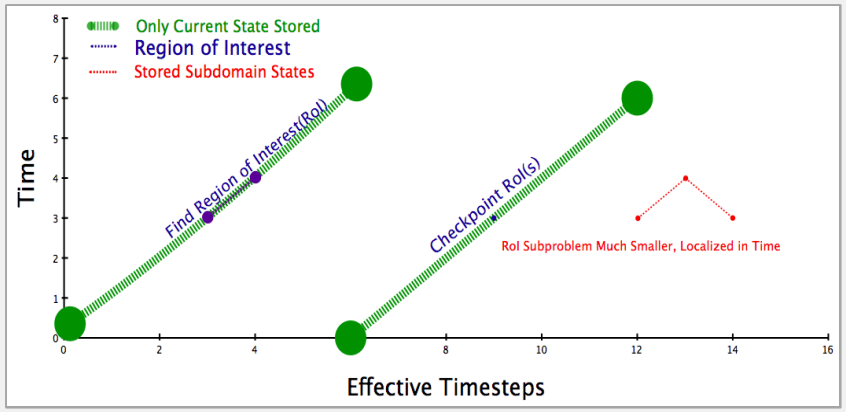
Adjoint solutions pose a co-design challenge: How to effectively manage the work and data flow

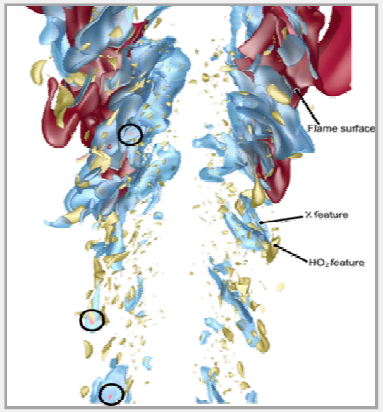


Storing entire primal state is infeasible: 5ZB



Recomputation reduces storage costs: 4 EB





Focusing computation to regions of interest (ROI) : 500PB/ROI

Proxy apps are being developed to explore trade-offs in the adjoint work & data flows

Current big compute and high-throughput experimental workflows are similar

Big Compute

Simulate

Store

Analyze

Experimental

Experiment

Store

Analyze

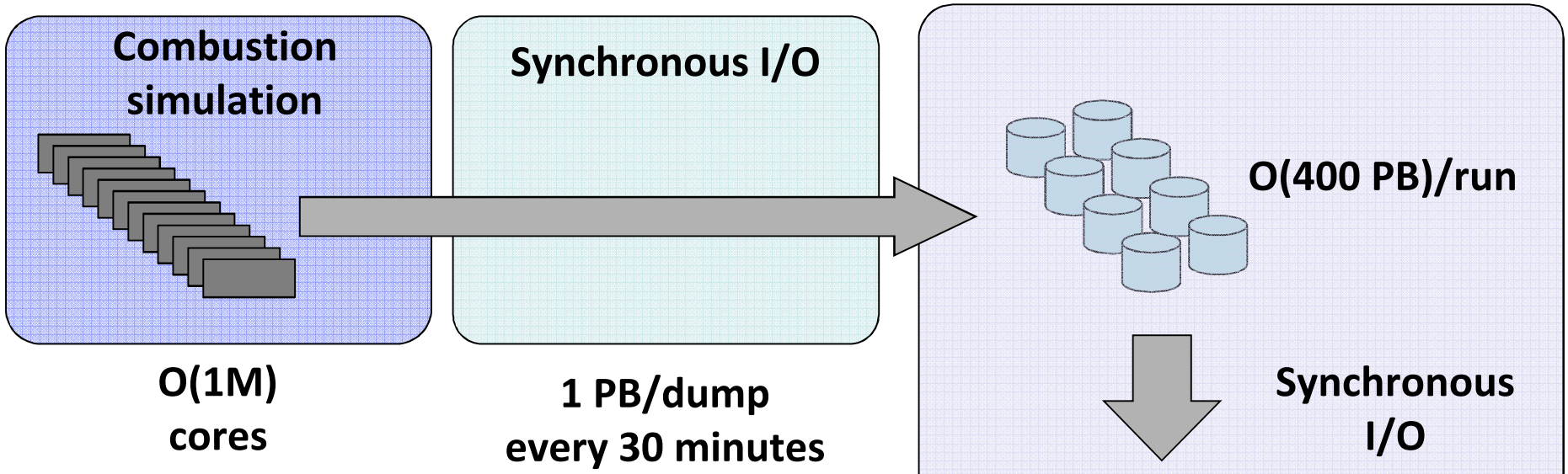
Need to verify experimental workflow

Both big compute and high-throughput experimental workflows face big data challenges

- Both big compute and high-throughput workflows will generate too much data to be stored directly
- Future workflows will move analysis closer to both simulations and experiments
  - Either full analysis or data reduction performed in situ
  - Secondary computational resources finish time-sensitive computations for computational steering
  - Reduced data stored for down-stream analysis
- **Co-design is needed to identify the architectural designs that result in global optimized performance**

BACKUP SLIDES

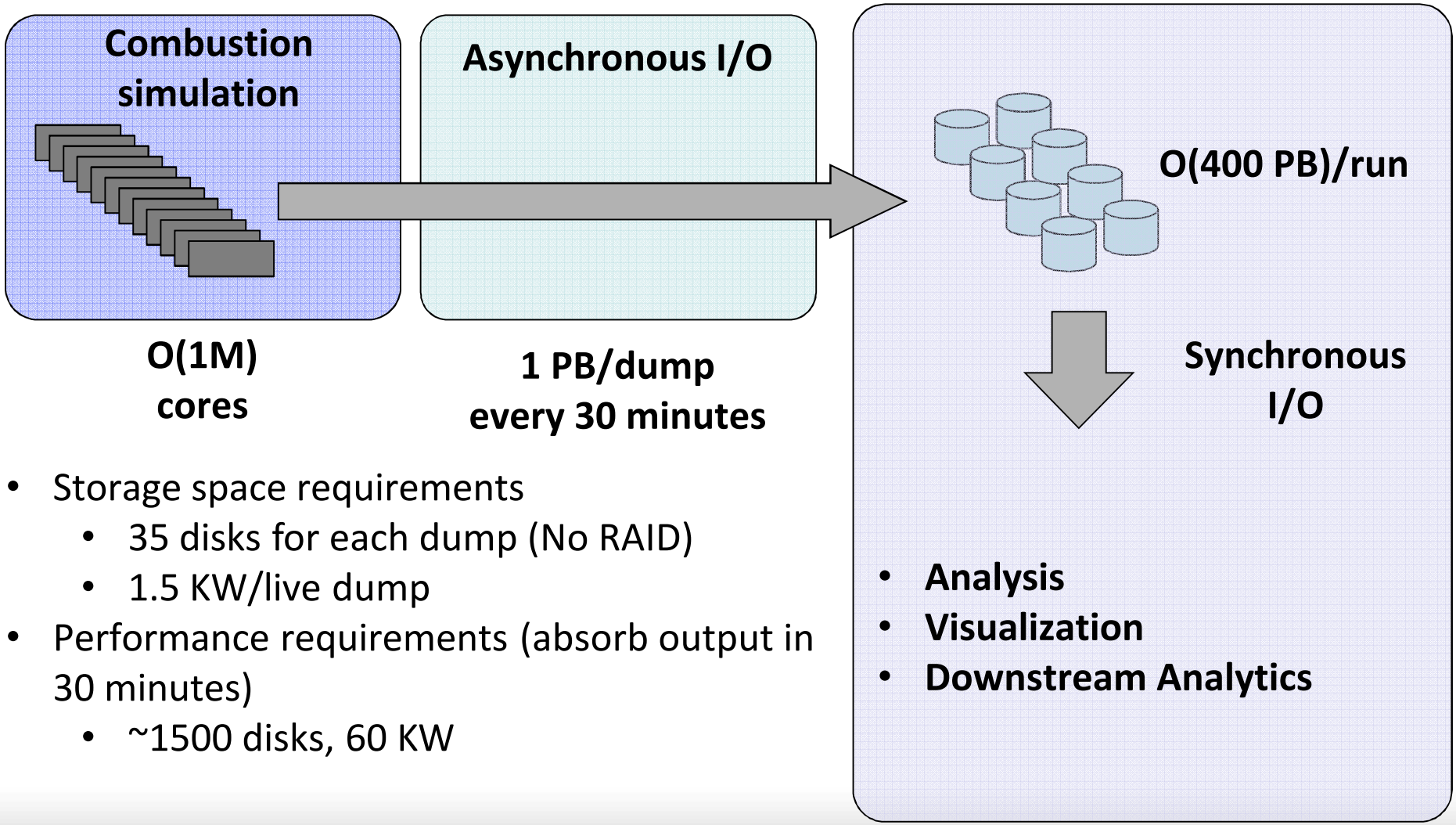
Synchronous I/O at exascale is expensive



- Storage space requirements
  - 35 disks for each dump (No RAID)
  - 1.5 KW/live dump
- Performance requirements
  - 5% overhead, ~1M disks, >40 MW
  - 10% overhead, ~500K disks, >20 MW
  - 50% overhead, ~100K disks, >4 MW

- **Analysis**
- **Visualization**
- **Downstream Analytics**

Asynchronous I/O is also expensive



- Storage space requirements
  - 35 disks for each dump (No RAID)
  - 1.5 KW/live dump
- Performance requirements (absorb output in 30 minutes)
  - ~1500 disks, 60 KW

UQ Impact: Turbulent Combustion

- Effective, robust UQ methods for long-time, chaotic systems open research question.
- Space/Time localized phenomena of interest more tractable for current UQ methods(Extinction, Ignition)
- Embedded chemistry models inherent source of uncertainty
  - ❑ Reaction rates
  - ❑ Missing/under-modeled processes/reactions
  - ❑ Species concentrations, initial profiles
- Expression/reduction of basic science/prediction goals to mathematical UQ framework.
  - ❑ Leads to tractable and relevant set of UQ problems
  - ❑ Helps catalog varying sources of uncertainty

UQ Challenges: Nodal Proxy Apps

Two Classes of Intrusive UQ Proxy Apps

1. Adjoint-based Sensitivity Proxy App
   - Allows efficient sensitivity analysis for large number of parameters
   - Requires solution of adjoint problem, backwards in time.
   - Implementation impacts available fast-memory(need state(s), adjoint state).
   - Increases need for low-storage, high-order time integrator
   - CNS Sensitivity Proxy App available now.
2. Polynomial Chaos(PC) Proxy App
   - Even “S3D-like” implementation requires access to distinct realizations of solution state.
   - Increased memory access influences stencil, time-integrator, and parallel decomposition choices.
   - Proxy app allows exploration of different multirate time integration/operator split choices at the “realization”/PC coefficient level.
   - PC Proxy App available 5/12.

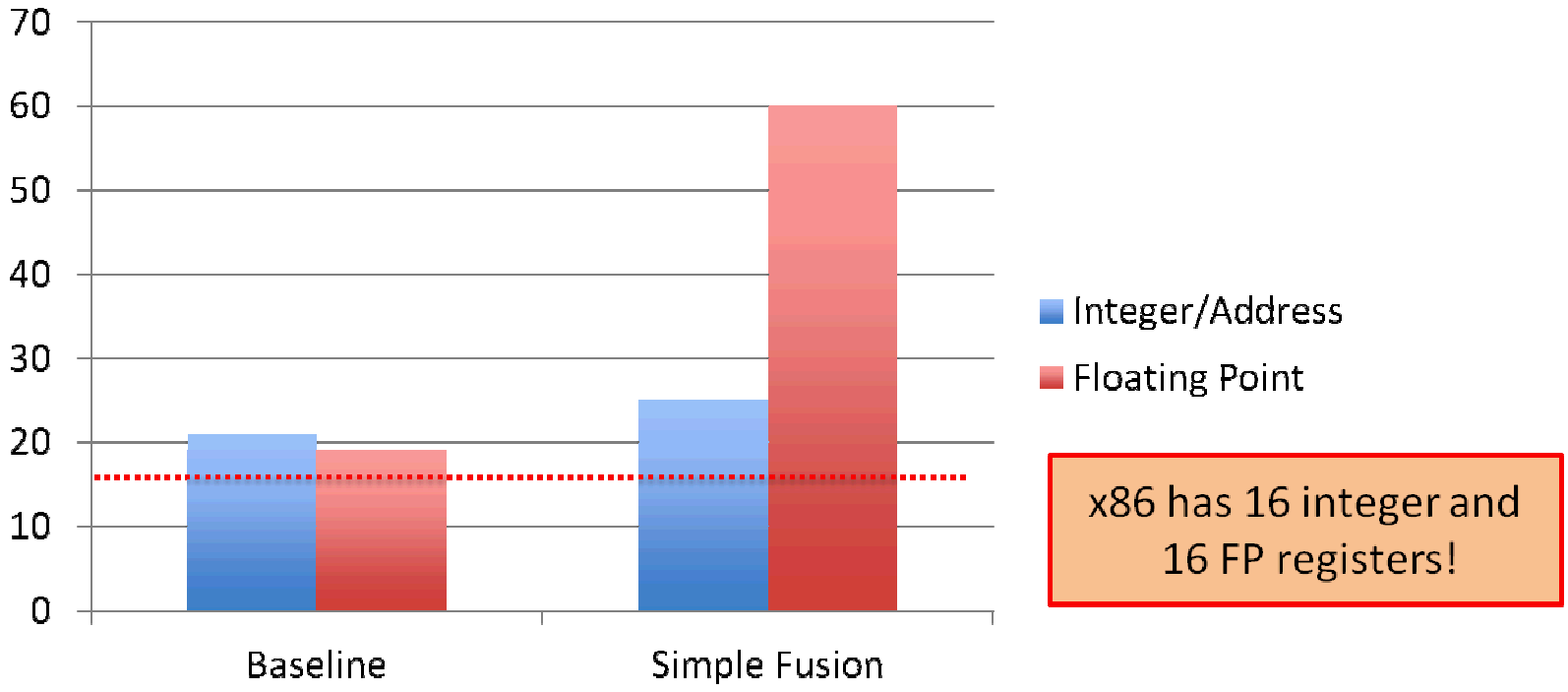
UQ Challenges: Data Management

Research of UQ impact on SDMA parallels proxy app thrust

1. Adjoint-based Sensitivity Analysis
   - Computation of adjoint requires solution access on all of a space-time domain "near" the phenomena of interest.
   - Leads to optimization over:
     - Available fast/slow memory
     - Costs of using slow(er) storage
     - Forward state recomputation(checkpointing)
     - Adjoint state(and resulting sensitivities) computation accuracy
2. Intrusive, Polynomial Chaos enabled UQ.
   - "S3D-like" implementation can result in significant data movement
   - AMR, multi-level methods, implicit time integration lead to even greater data challenges

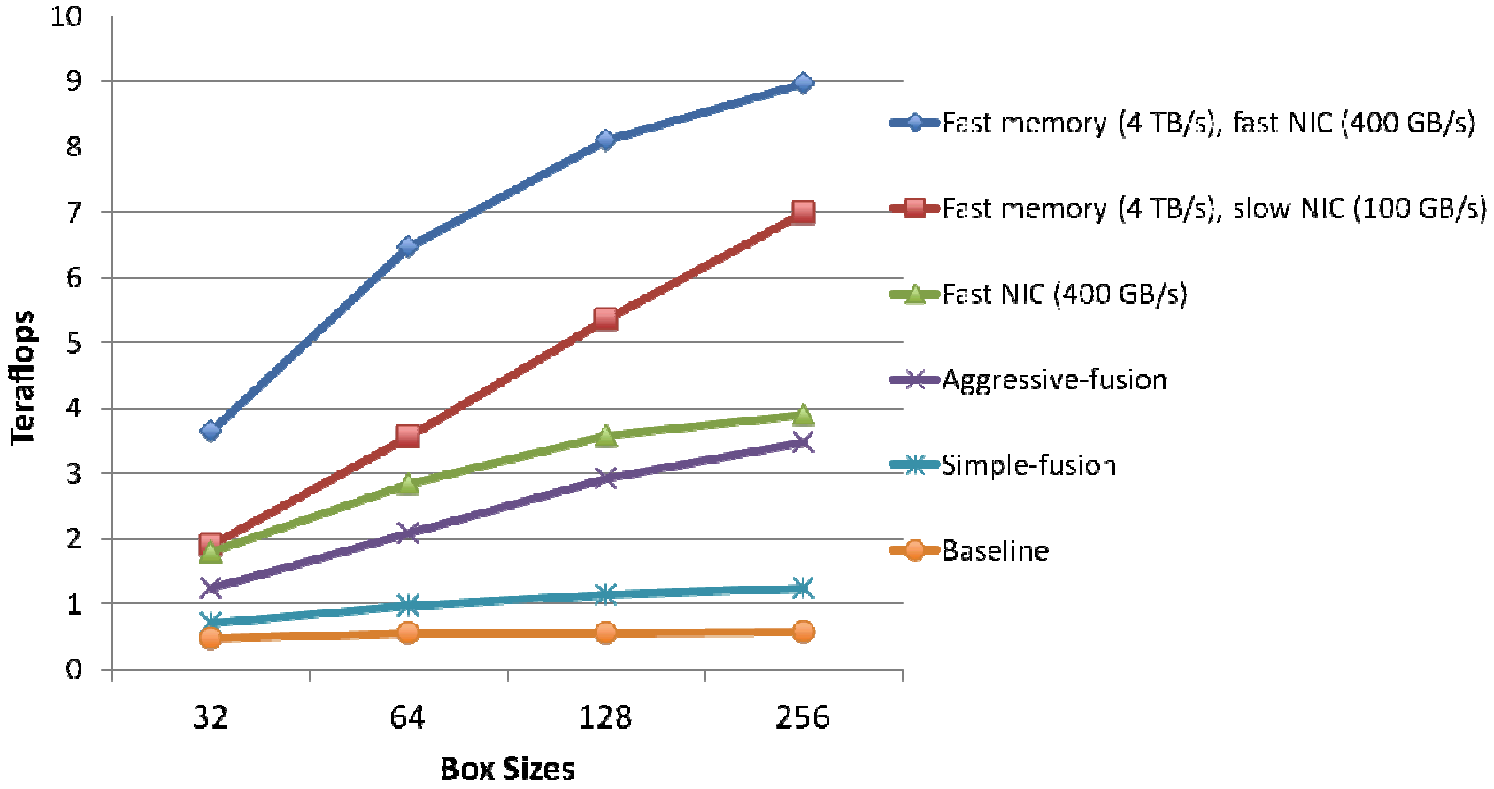
extras

Register State Estimates



- These estimates are for state variables only and do not include registers needed for arithmetic
- If there are not enough registers available, state variables spill into the L1 cache, increasing cache traffic and possibly affecting performance

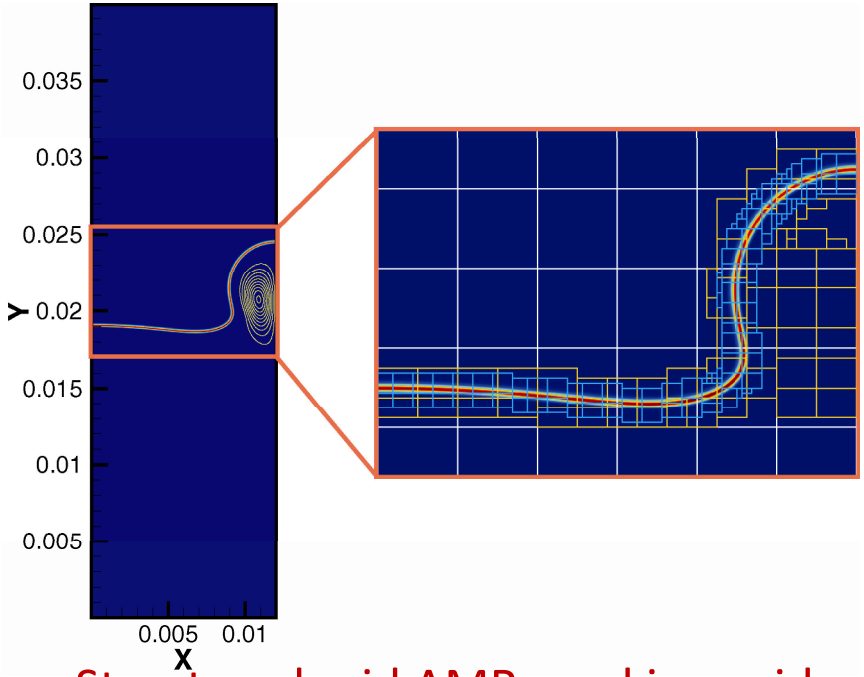
Impact of Exascale Node Characteristics on CNS Performance



- Aggressive loop fusion improves performance
- Fast memory doubles performance – but limits memory per node
- Network
  - High bandwidth network not significant for slow memory
  - High bandwidth network is significant with faster memory bandwidth
  - Latency not that important except for very small box sizes

Adaptive Mesh Refinement

- Need for AMR
  - Reduce memory
  - Reduce computation
- Block-structured AMR
  - Data organized into logically-rectangular structured grids
  - Amortize irregular work
  - Good match for multicore architectures
- AMR introduces extra algorithm issues not found in static codes
  - Metadata manipulation
  - Regridding operations
  - Dynamic communication patterns



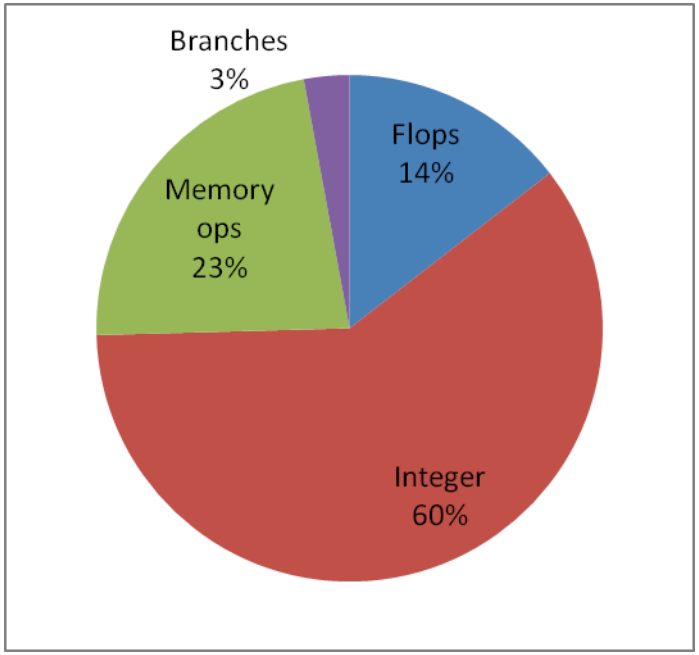
Structured grid AMR used in a wide range of DOE applications

- Astrophysics
- Climate
- Cosmology
- Defense science
- Materials science
- Subsurface flow

SMC Characterization using BYFL

- SMC – 8th order finite difference
  - Compact 2nd derivatives
  - Iterate 8th order first derivative – S3D-like
  - Trades Flops for communication
- Bytes to flops – 11.4 w/o cache

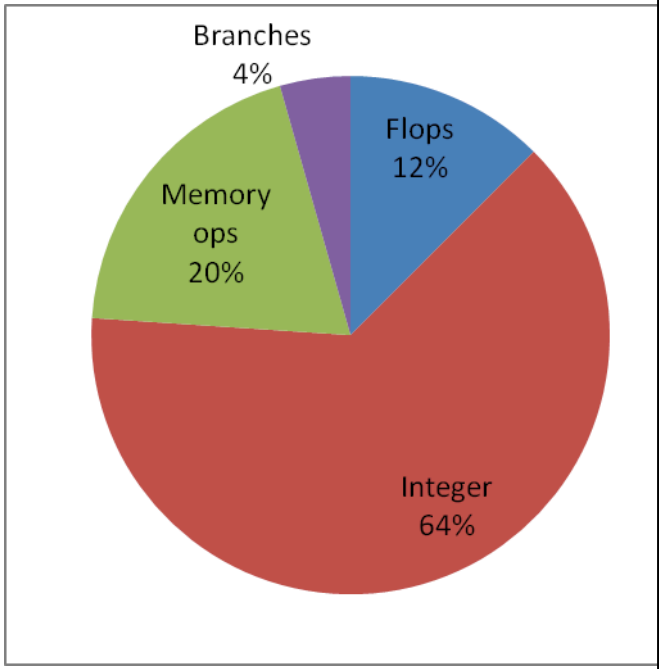
- Operation breakdown
  - 14% Flops
  - 23% Memory ops (loads and stores)
  - 60% Integer
  - 3% Branch



- Bulk of loads and stores are double precision floating point
- Significant data reuse
- Exp() plays a significant role in the floating point work
- Communication characteristics
  - 9.9% Iterated
  - 2.8% Compact

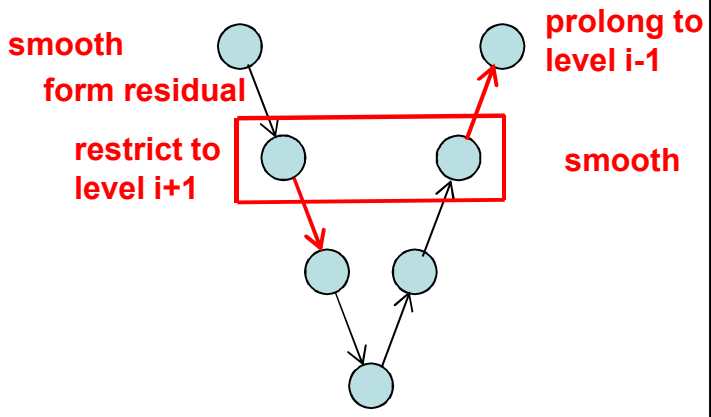
LMC characterization using BYFL

- Bytes to flops – 11.4 w/o cache
- Operation breakdown
  - 12% Flops
  - 20% Memory ops (loads and stores)
  - 64% Integer
  - 4% Branches
- Loads and stores
  - Double precision floating point still largest
  - Integer and pointer/address loads are significant (AMR feature)
- Data reuse much less than explicit codes
- Exp() plays a significant role in the floating point work
  - No other single routine carries a significant fraction of the work
- Communication characteristics – 20-25%



Multigrid Communication Characterization

- Look at how multigrid communication patterns are affected by network characteristics



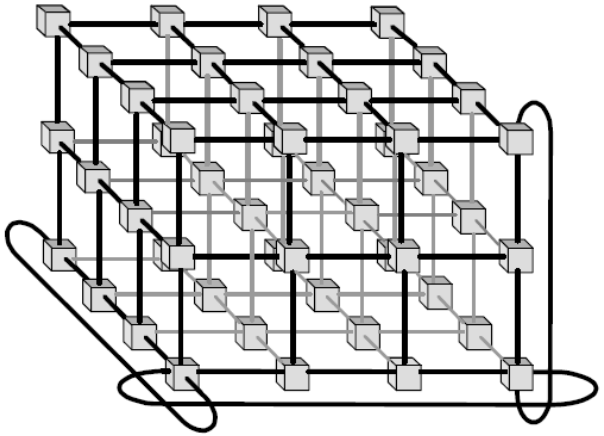
- PFMG
  - Structured algebraic multigrid method
  - Preserve stencil size
  - One of the options used in Boxlib
- Model communication in a PFMG cycle on 2 different networks (torus and fat tree) for a 7-point and a 27-point stencil
  - Includes latency, bandwidth and cost of additional hops
  - Results here for 27-point
  - Surrogate for higher-order stencils

Methodology

1. Create proxy machine models to represent exascale node architectures
   - Configurable machine parameters (cache size, memory bw)
2. Develop proxy applications to represent exascale combustion codes
   - CNS and SMC codes by J. Bell's group
3. Design a performance modeling framework to estimate performance of proxy apps on proxy machines
   - ExaSAT framework:
     - static compiler analysis + performance model

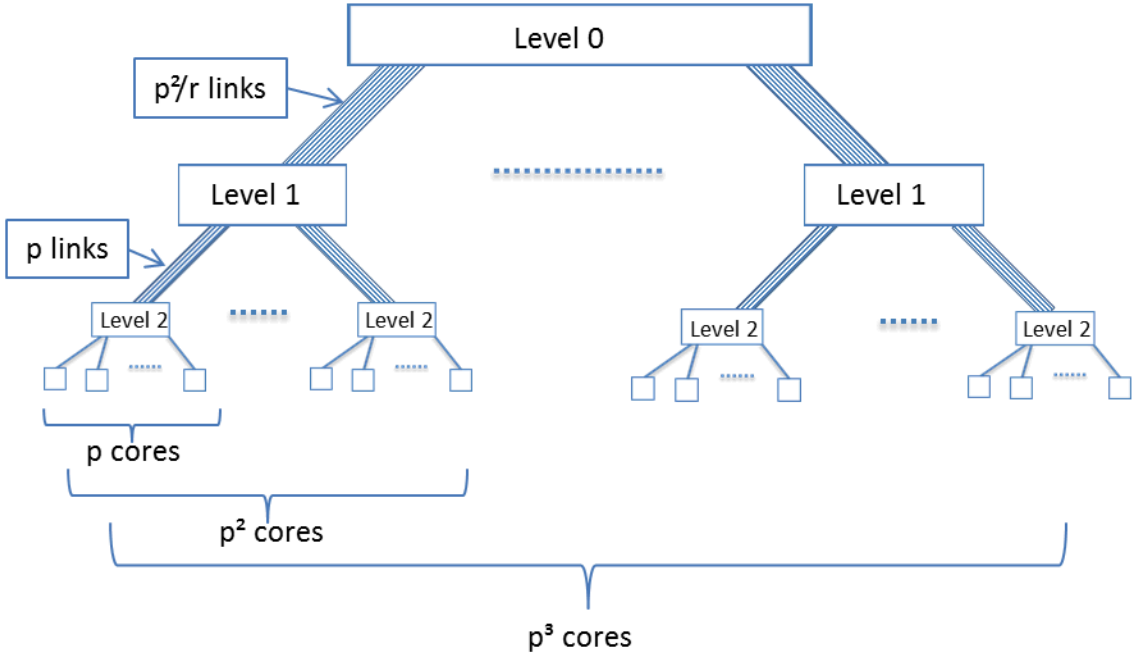
Networks considered

3D-Torus



p^3 cores (or end stations)

3-Level Fat tree



Both networks have same number of links, i.e. $3p^3$, but fat tree needs longer links

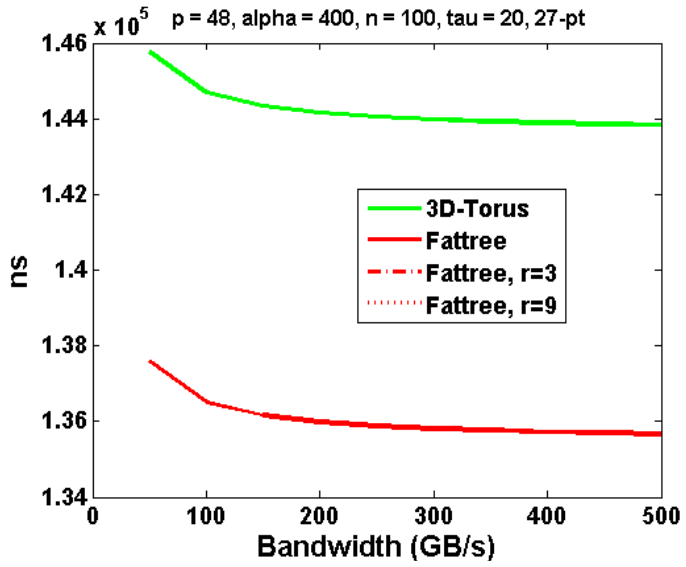
Tapering will reduce number of links for fat tree, and with it cost!

Parameters used in multigrid study

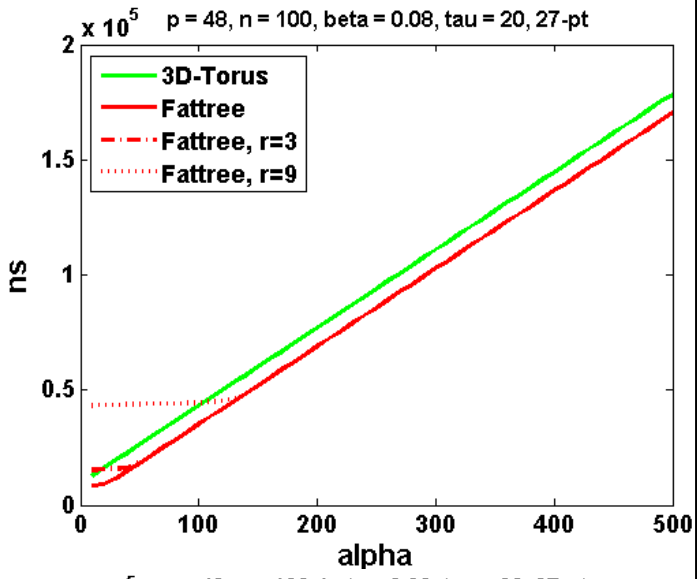
- $p = 48^3$ corresponding to 110,952 nodes
- Effect of bandwidth given latency
  - 400 ns latency
  - 20 ns latency
- Effect of latency given bandwidth
  - 100 GB / s
  - 400 GB / s
- Level of tapering at top level of the fat-tree
  - r = 1 (no tapering) -> 2308 links from level 0 to level 1
  - r = 3 -> 768 links
  - r = 9 -> 256 links
- Simple model for contention

Multigrid communication costs – 27 point stencil

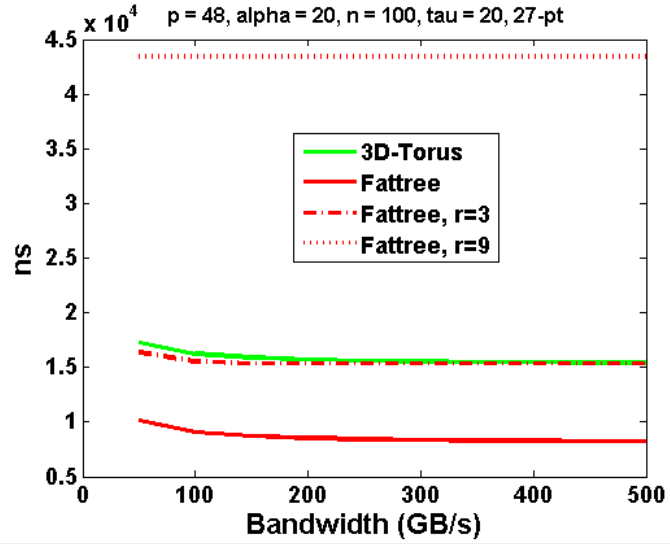
400 ns latency



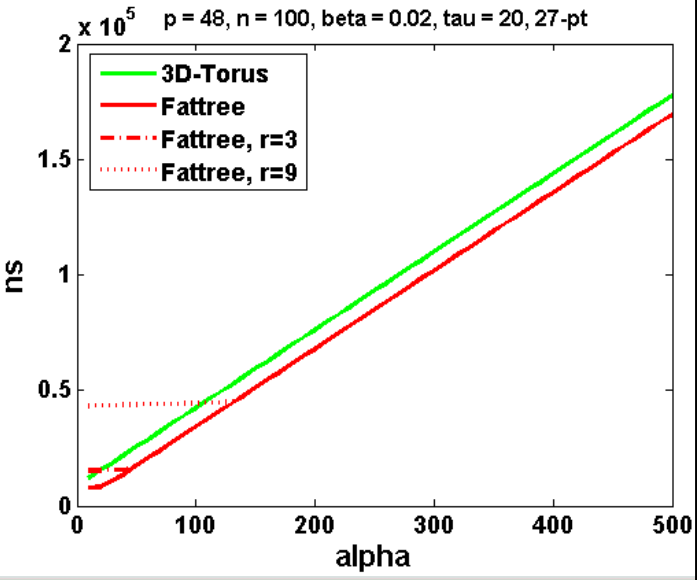
100 GB / s bandwidth



20 ns latency



400 GB / s bandwidth

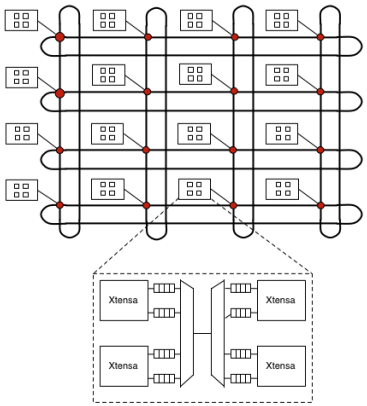


Summary – Results in Solver Area

- Characterization of baseline methodology
  - Instruction mix
  - Floating point intensity
  - Communication
  - Breakdown of execution time
- Developed initial suite of proxy apps
  - Compressible Navier Stokes without species
  - Generalization to multispecies with reactions
  - LMC
  - Multigrid algorithm – 7 and 27 point stencils
  - Chemical integration
  - Coming soon
    - Embedded UQ kernels
    - Additional AMR proxy apps
- Methodology for co-design
  - Algorithm description – code analysis or algorithm specification
  - Performance measurement
  - Analytic performance modeling
  - Judicious application of simulator
- Initial application to co-design equations with CNS and multigrid proxies

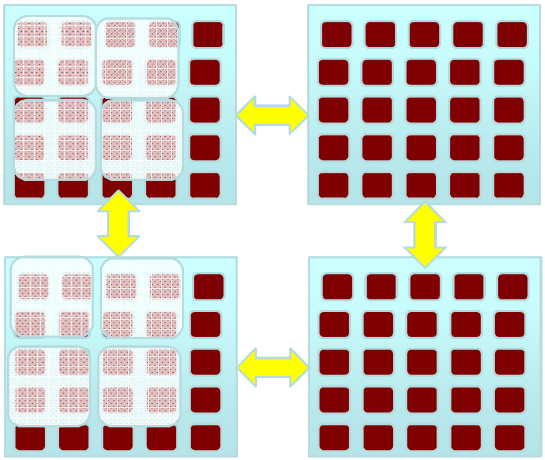
A Toy (parameterized) abstract machine model
(the full range of design choices for a machine)

Chip Scale



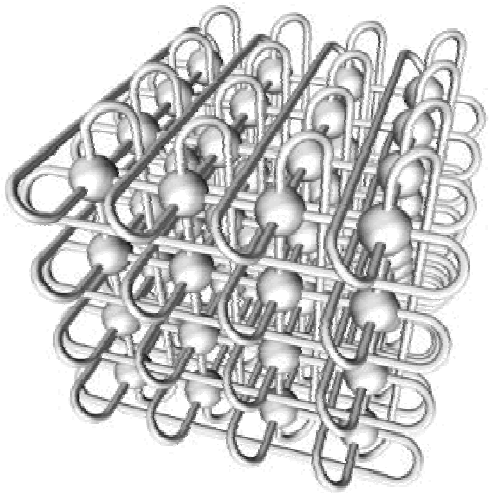
- Cores (many simple cores)
  - Flat clock rate
  - Multithreaded (n-threads)
  - SIMD (n-slots)
  - Fat+Thin cores (ratio)
- NoC
  - Constrained Topology (2D) *torus, mesh, ring*
- Cache Hierarchy (size, type, assoc)
  - Automatic caches
  - Scratchpad/software managed
  - NVRAM
  - Alternative coherency methods

Node Scale



- Non-uniform memory access (NUMA) between cores and memory channels
  - Topology may be important
  - Or perhaps just distance
- Memory
  - Increased NUMA domains
  - Intelligence in memory (or not)
- Fault Model for node
  - FIT rates, kinds of faults, granularity of faults/recovery

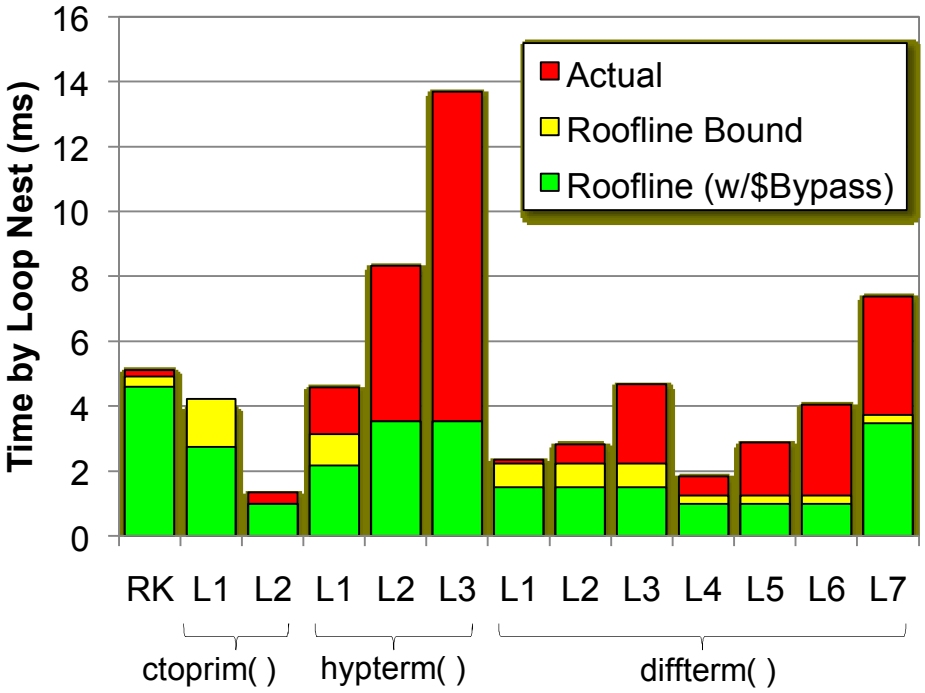
System Scale



- Interconnect
  - Constrained Topology (Torus, Tapered Dragonfly)
  - Bandwidth/latency/overhead for communication
- Primitives for data movement/sync
  - Global Address Space or messages only
  - Memory fences
  - Transactions / remote atomics

Measured Performance and Lower Bounds

- Measure performance of simplest proxy app – CNS
- Manually produced a Roofline model for each loop nest in CNS code
  - Red is CNS running on Hopper.
  - Green is lower bound for moving data
- Loops with large cache working set dramatically underperforms the roofline



Roofline model defines intersection of limits to performance from hardware elements

Optimization of Hypterm

- Consider one particularly underperforming function: hypterm()
- Roofline bounds performance to an 8x speedup over the original version
- What optimizations can be applied?
  - Vector-like semantics to improve prefetcher behavior
  - Loop fusion
  - Cache blocking
  - Buffering (for $bypass & conflict misses)
  - Task-like parallelization
  - Blocked data layout (instead of flat arrays)
  - SIMDization
- Overall, attained a 6.5x speedup
- Analysis needs to reflect realizable performance