

“Algorithm research has
been driven by hard to
use machines.”
–Rob Schreiber (HP Labs)



“People who are serious about
software should make their
own hardware.”
–Alan Kay (Xerox PARC)

Application-Driven Co-Design

James Belak

Deputy Director, belak@llnl.gov

Exascale Co-Design Center for Materials in Extreme Environments

Modeling and Simulations of Exascale Systems and Applications

9-10 August 2012

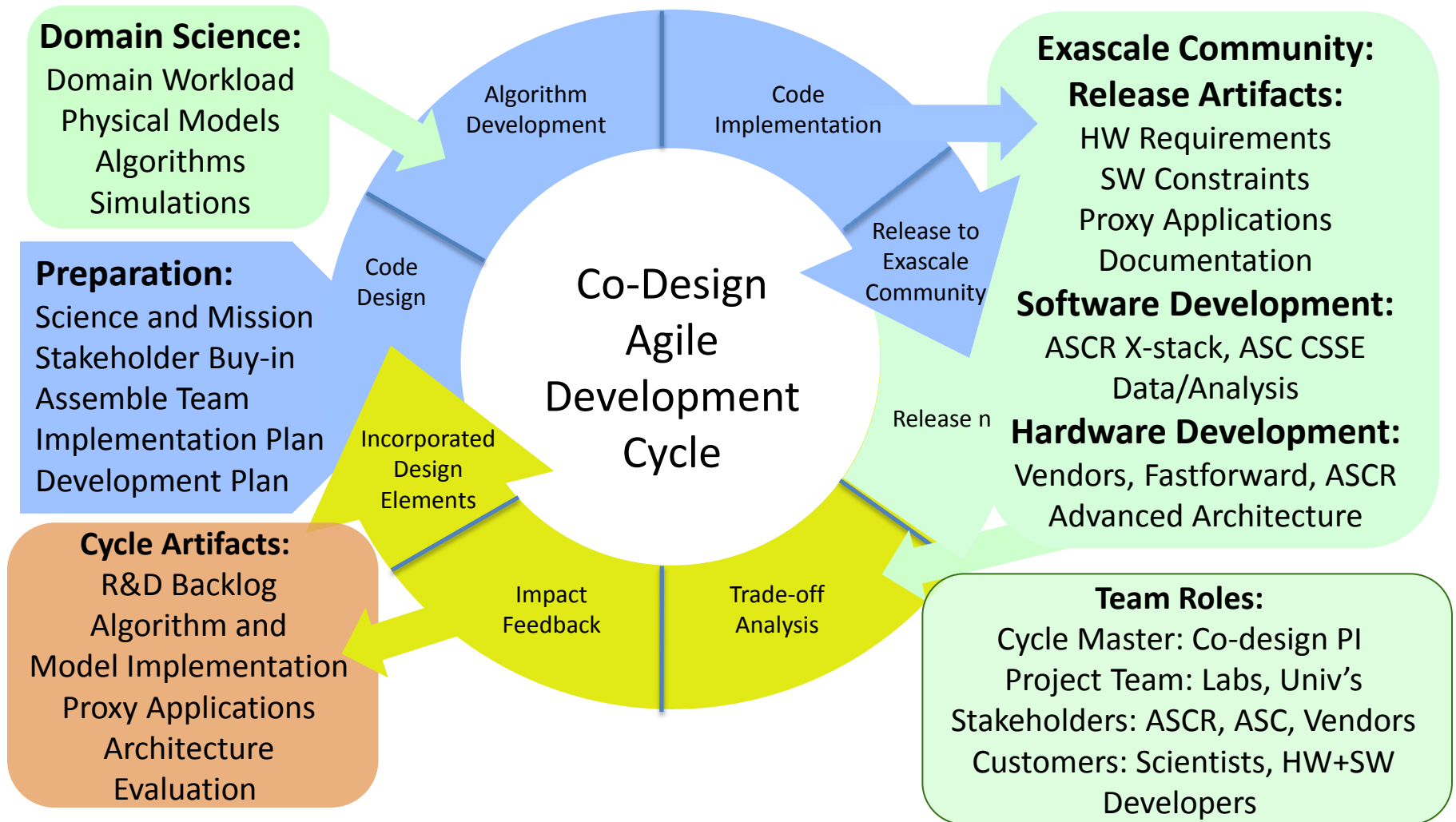
Seattle

“(Application driven) co-design is the process where scientific problem requirements influence computer architecture design, and technology constraints inform formulation and design of algorithms and software.”
–Bill Harrod (DOE)

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-PRES-XXXXXX

Creation of a functional exascale simulation environment requires our co-design process to be *adaptive, iterative, and lightweight* – i.e. agile





CESAR: Center for Exascale Simulation of Advanced Reactors

Director: Robert Rosner (ANL/UChicago) rrosner@ci.uchicago.edu

Deputy Director: Andrew Siegel (ANL) siegela@mcs.anl.gov

Enable a coupled, next-generation nuclear reactor core simulation tool capable of efficient execution on exascale computing platforms.

Combustion: Combustion Exascale Co-Design Center

Director: Jacqueline Chen (SNL) jhchen@sandia.gov

Deputy Director: John Bell (LBNL) jbell@lbl.gov

Enable combustion scientists to perform first principles direct numerical simulations with sufficient physics fidelity to answer fundamental questions to meet pollutant and greenhouse gas emissions targets, reduce dependence on petroleum and promote economic competitiveness.

ExMatEx: Exascale Co-Design Center for Materials in Extreme Environments

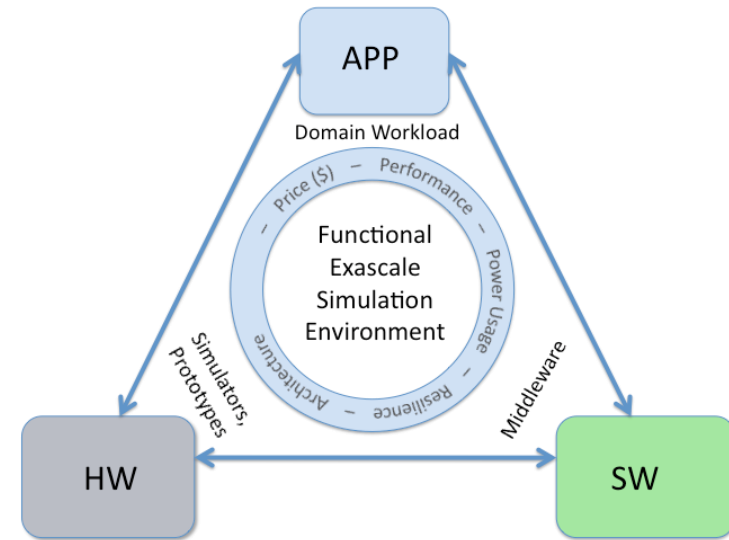
Director: Tim Germann (LANL) germann2@lanl.gov

Deputy Director: Jim Belak (LLNL) belak@llnl.gov

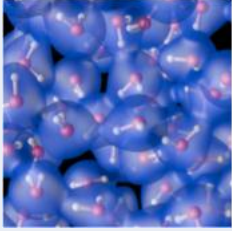
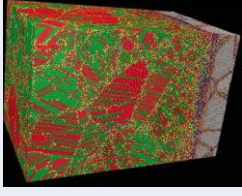
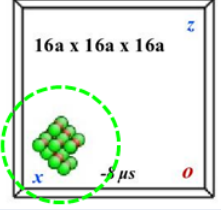
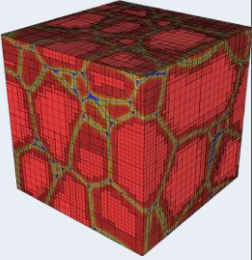
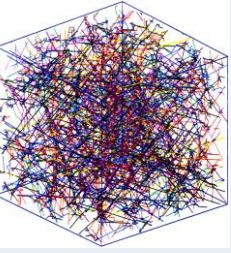
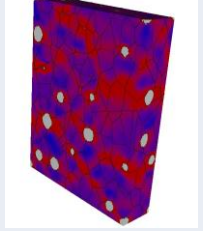
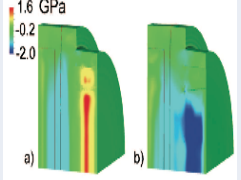
Establish the interrelationship between software and hardware required for materials simulation at the exascale while developing a multiphysics simulation framework for modeling materials subjected to extreme mechanical and radiation environments.

ExMatEx Co-Design Project Goals

- Our goal is to establish the interrelationship between hardware, middleware (software stack), programming models, and algorithms required to enable *a productive exascale environment* for multiphysics simulations of materials in extreme mechanical and radiation environments.
- We will exploit, rather than avoid, the greatly increased levels of concurrency, heterogeneity, and flop/byte ratios on the upcoming exascale platforms.
- This task-based approach leverages the extensive concurrency and heterogeneity expected at exascale while enabling fault tolerance within applications.
- The programming models and approaches developed to achieve this will be broadly applicable to a variety of multiscale, multiphysics applications, including astrophysics, climate and weather prediction, structural engineering, plasma physics, and radiation hydrodynamics.

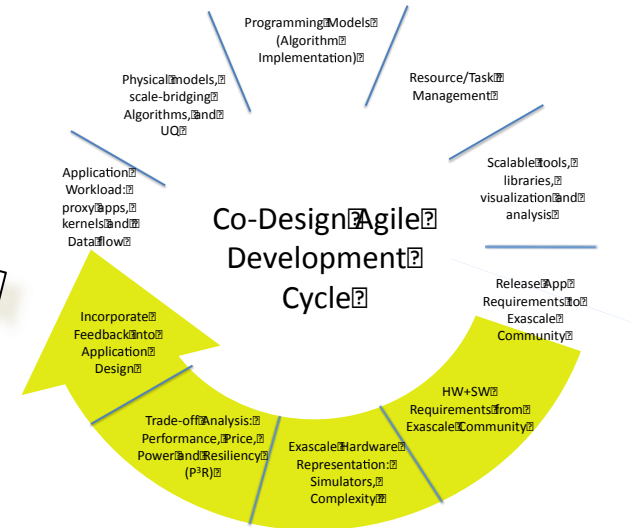
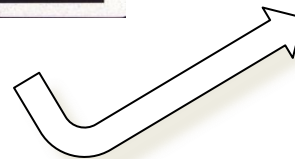


Exascale is about better Physics Fidelity: Engineering assessment of material behavior is limited by physics fidelity

Ab-initio	MD	Long-time	Phase Field	Dislocation	Crystal	Continuum
Inter-atomic forces, EOS, excited states	Defects and interfaces, nucleation	Defects and defect structures	Meso-scale multi-phase evolution	Meso-scale strength	Meso-scale material response	Macro-scale material response
						
Code: Qbox/LATTE Motif: Particles and wavefunctions, plane wave DFT, ScaLAPACK, BLACS, and custom parallel 3D FFTs Prog. Model: MPI + CUBLAS/CUDA	Code: SPaSM/ddcMD/COMD Motif: Particles, explicit time integration, neighbor and linked lists, dynamic load balancing, parity error recovery, and <i>in situ</i> visualization Prog. Model: MPI + Threads	Code: SEAKMC Motif: Particles and defects, explicit time integration, neighbor and linked lists, and <i>in situ</i> visualization Prog. Model: MPI + Threads	Code: AMPE/GL Motif: Regular and adaptive grids, implicit time integration, real-space and spectral methods, complex order parameter Prog. Model: MPI	Code: ParaDiS Motif: “segments” Regular mesh, implicit time integration, fast multipole method Prog. Model: MPI	Code: VP-FFT Motif: Regular grids, tensor arithmetic, meshless image processing, implicit time integration, 3D FFTs. Prog. Model: MPI + Threads	Code: ALE3D/LULESH Motif: Regular and irregular grids, explicit and implicit time integration. Prog. Model: MPI + Threads

Agile proxy application development

- Petascale single-scale SPMD and scale-bridging MPMD proxy apps will be used to explore algorithm and programming model design space with domain experts, hardware architects and system software developers.
- Proxy apps communicate the application workload to the hardware architects and system software developers, and are used in models/simulators/emulators to assess performance, power, and resiliency.
- These proxy applications will not be "toy models", but will realistically encapsulate the work of mathematical algorithms of the full applications.



Proxy applications communicate the computational workload of “real” applications into the co-design process

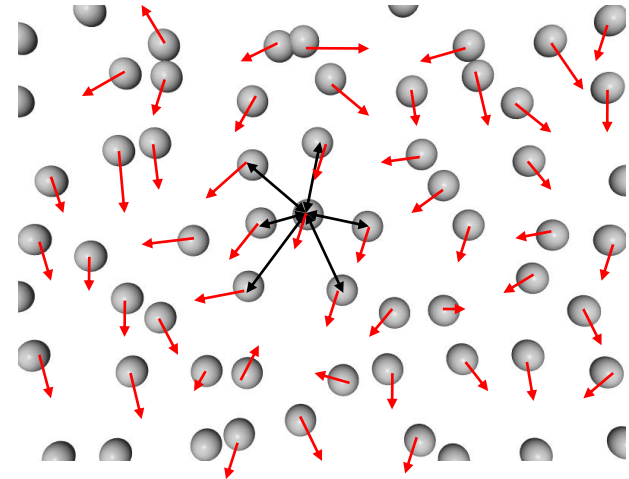
- Initial Proxy Apps represent the computational work as we expose it on petascale systems today
- Proxy apps will evolve through co-design to represent the computational work as it must be exposed to exascale systems
- Initial Materials Science Proxy Suite:
 - CoMD: Co-designed molecular dynamics, exploration of exascale data layout and programming models (what works at exascale?) (jamal@lanl.gov)
 - LULESH: Shock hydrodynamics portion of ALE3D, originally developed for DARPA UHPC, coarse scale app for scale bridging (keasler1@llnl.gov)
 - VPFFT: Microstructure sensitive crystal plasticity, initial fine scale app for scale bridging (li31@llnl.gov)
 - MTREE: Exploration of exascale data layout for scale bridging adaptive sampling (dorr1@llnl)

Molecular Dynamics (CoMD)

Molecular dynamics: particles interact via explicit interatomic potentials and evolve in time according to Newton's equations of motion:

$$\dot{\mathbf{r}}_i = \mathbf{p}_i / m_i \quad \dot{\mathbf{p}}_i = \mathbf{f}_i$$

$$\mathbf{f}_i = m_i \ddot{\mathbf{r}}_i = - \sum_j \nabla V_{ij}$$

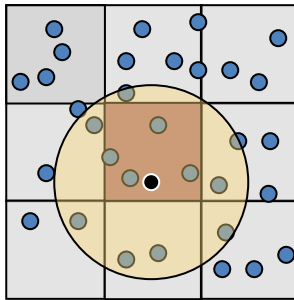


Interaction potentials determine both the physics and computer science

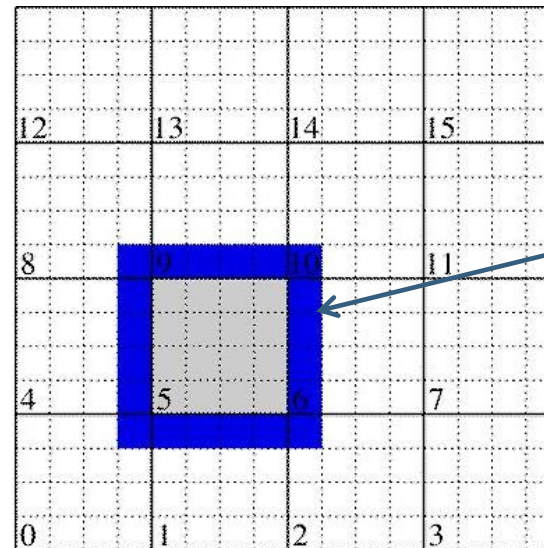
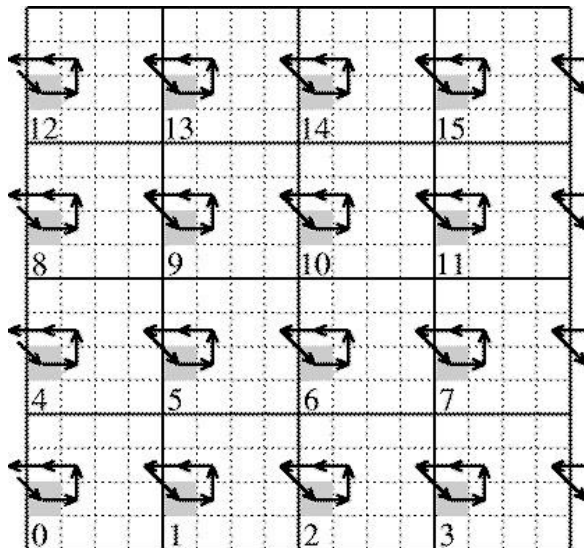
- Complex potentials are more accurate, but can require many more floating point operations.
- Locality of potential informs parallelization strategy, e.g. short-ranged potentials require only point to point communication.

Challenge Problem: Can you use an exascale computer with billion-way parallel parallelism to simulate longer in time? (not just more atoms)

How are forces calculated in a parallel MD code?



- ~ 20 atoms in each box
- ⇒ each atom interacts with 540 other atoms
- ⇒ However, only ~70 atoms lie within cutoff
- ⇒ Lots of wasted work
- ⇒ We need a means of rejecting atoms efficiently even within this reduced set

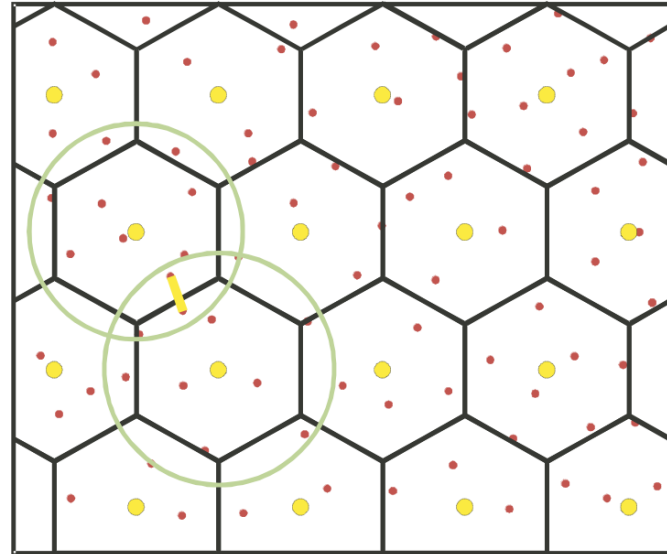


Fixed geometric domain decomposition limits scalability for any heterogeneous problem. Furthermore, statistical fluctuations in the force calculation between processors leads to an effective scalar term that also limits scaling (Amdahl's law).

Domain decomposition strategy for ddcMD

Design requirements:

- Run efficiently on arbitrary number of processors
- Excellent weak scaling to extend size of simulation
- Excellent strong scaling to extend MD time scale



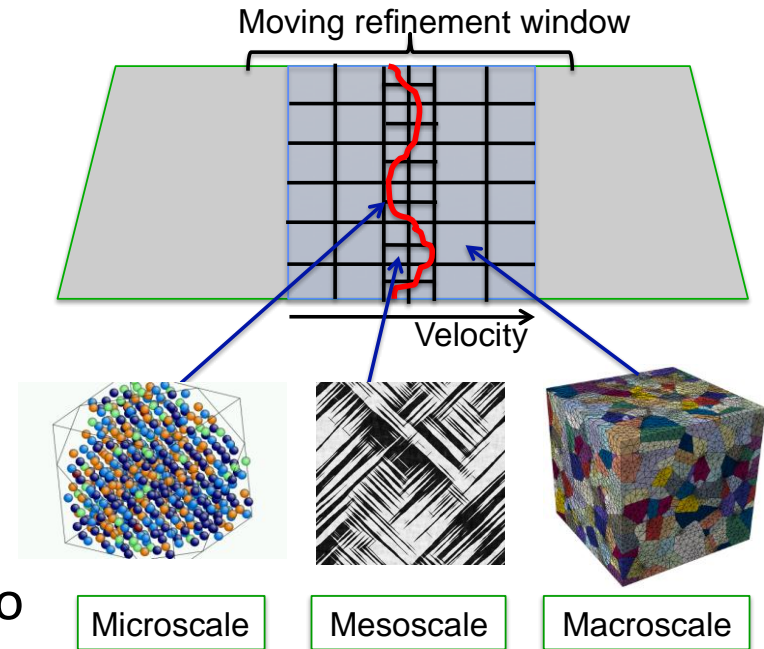
Solution:

- Particle-based domain decomposition - processors own particles, not regions - allows decomposition to persist through atom movement
- Maintain minimum communication list for given decomposition - allows extended range of "interaction"
- Arbitrary domain shape - allows minimal surface to volume ratio for communication



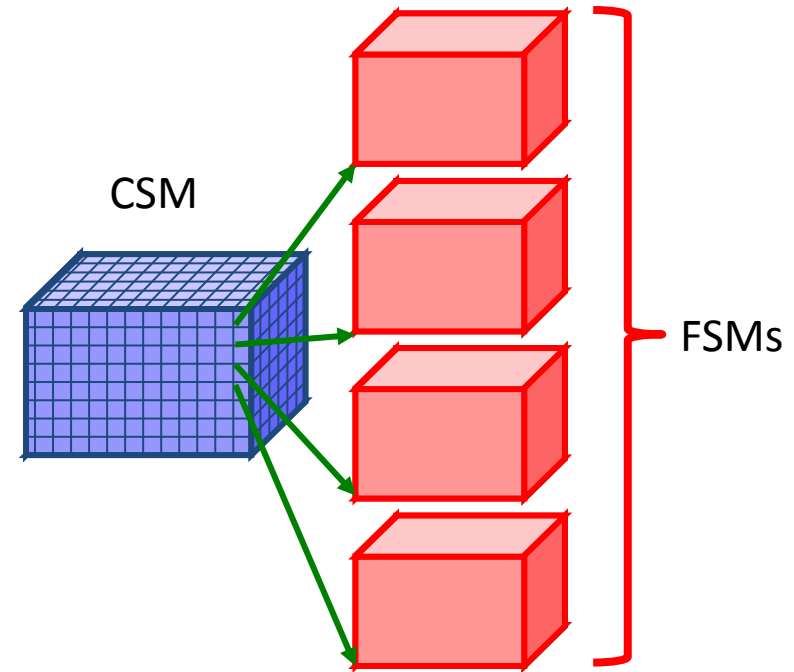
Embedded Scale-Bridging Algorithms

- Our goal is to introduce more detailed physics into computational materials science applications in a way which escapes the traditional synchronous SPMD paradigm and exploits the heterogeneity expected in exascale hardware.
- To achieve this, we are developing a UQ-driven *adaptive physics refinement* approach.
- Coarse-scale simulations dynamically spawn tightly coupled and self-consistent fine-scale simulations as needed.
- This *task-based* approach naturally maps to exascale heterogeneity, concurrency, and resiliency issues.



Direct multi-scale embedding requires full utilization of exascale concurrency and locality

- *Brute force multi-scale coupling:* Full fine scale model (FSM, e.g. a crystal plasticity model) run for every zone & time step of coarse scale mode (CSM, e.g. an ALE code)
- *Adaptive Sampling:*
 - Save FSM results in database
 - Before running another FSM, check database for FSM results similar enough to those needed that interpolation or extrapolation suffices
 - Only run full FSM when results in database not close enough



- Heterogeneous, hierarchical MPMD algorithms map naturally to anticipated heterogeneous, hierarchical architectures
- Escape the traditional bulk synchronous SPMD paradigm, improve scalability and reduce scheduling
- Task-based MPMD approach leverages concurrency and heterogeneity at exascale while enabling novel data models, power management, and fault tolerance strategies

Ref: Barton *et.al*, 'A call to arms for task parallelism in multi-scale materials modeling,' *Int. J. Numer. Meth. Engng* 2011; 86:744–764

Coarse Scale: LULESH (Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics)

- Represents coarse scale aspect of our scale bridging approach
- Initially created for DARPA UHPC work, now under LLNL LDRD, Tri-Lab mini-Apps
- 3k lines of code, including mesh and boundary conditions
 - Lagrangian hydrodynamics
 - Single simplified material model

Conservation of mass

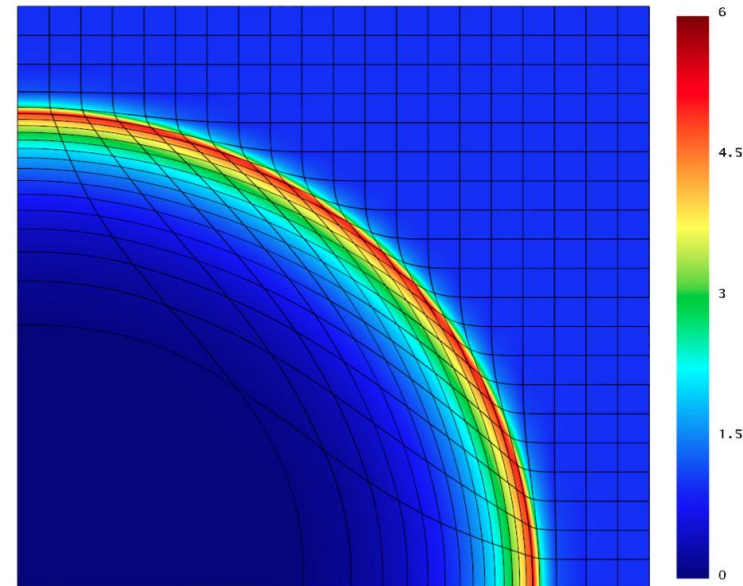
$$\frac{D\rho}{Dt} = -\rho \vec{\nabla} \cdot \vec{U}$$

Conservation of momentum

$$\rho \frac{D\vec{U}}{Dt} = -\vec{\nabla} \cdot p$$

Conservation of energy

$$\frac{De}{Dt} = -p \frac{DV}{Dt}$$

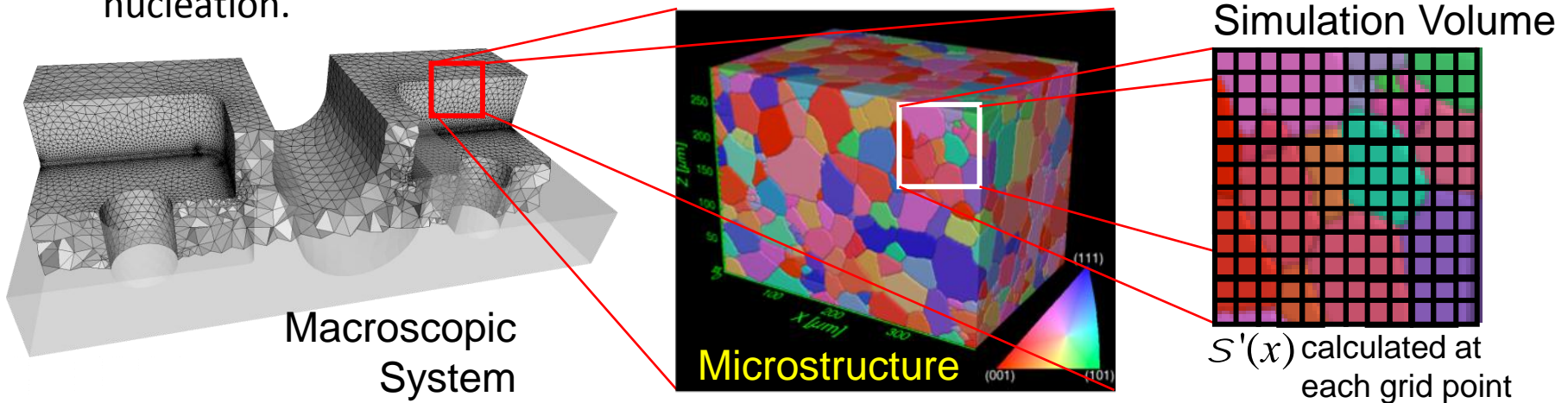


Sedov blast wave problem

Fine Scale: VP-FFT (Viscoplastic Fast Fourier Transform)

- Given input microstructure and applied deformation, compute full-field response in the form of stress states and anisotropic lattice reorientation due to polycrystal plasticity.
- Solve the set of non-linear constitutive equations iteratively for each grid point.
- Determine admissible stress and strain rate at each grid point that satisfies the equilibrium and compatibility condition.
- Spatial variation is crucial to prediction of failure, such as crack initiation and void nucleation.

$$\dot{\epsilon}(x) = \dot{\gamma}_o \sum_s \dot{a} m^s(x) \frac{\partial m^s(x) : S(x) \dot{\epsilon}^n}{t_o^s(x) \dot{\epsilon}}$$



R. A. Lebensohn, *Acta Mater.* **49**, 2723-2737 (2001); **56**, 3914-3926 (2008); A.D. Rollett et al., *MSMSE*, **18** 074005 (2010).

Frankie Lee (lee31@llnl.gov)

Bridging Scales: Adaptive Sampling proxy app

The Adaptive Sampling (AS) proxy application enables the standalone evaluation of fine-scale response data management options, addressing questions such as:

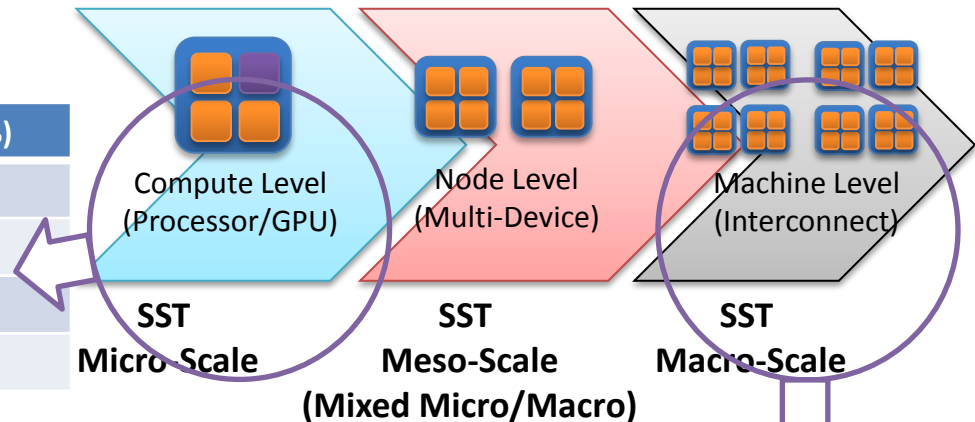
- How should the database be redistributed as the calculation proceeds?
- Should the fine-scale model sometimes be re-evaluated anyway to avoid communication?
- As different parts of the calculation “learn” the fine-scale response in their respective locales, how can that knowledge be shared globally?
- What are the load balancing and scheduling consequences of the fact that some elements will be performing new fine-scale evaluations while others are (more quickly) obtaining the fine-scale response from the database?
- Do existing programming models and system software provide sufficient capability to implement the “optimal” algorithm?
- Can the the task-based programming model inherent to adaptive sampling enable new paradigms for resiliency?

We need a predictive tool for mapping proxy apps to new architectures.

How well does SST represent the machines on the floor today?

SST Micro simulating serial LULESH on an AMD A8
2.90 GHz (2011) Core

Iterations	Actual (s)	Pred. (s)	Error (%)
10	1.5060	1.6200	7.57
20	2.9360	3.3771	15.02
30	4.2480	5.2400	23.35
40	5.9110	6.7500	14.19



Lessons learned:

SST is under active development and needs more development resources, it has not been fully validated.

SST simulations are themselves computationally intensive

10% error should be expected, Sims should be thought of as exploration of space, e.g. "what if?" kind of questions

SST needs better problem specifications

Compilers are a large source of variability

Cores	Actual (s)	Prediction (s)	Error (%)
64	105.95	96.57	8.86
125	106.23	97.40	8.32
216	108.74	98.32	9.59
343	106.46	99.52	6.52
512	107.04	102.97	3.80
729	112.44	99.71	11.32
1000	118.44	98.12	17.16
1331	107.66	96.65	10.22

SST Macro simulating LULESH MPI on a uBG/L
(VN-mode, 500 iterations, 45³ weak scaled)

Challenges facing SST

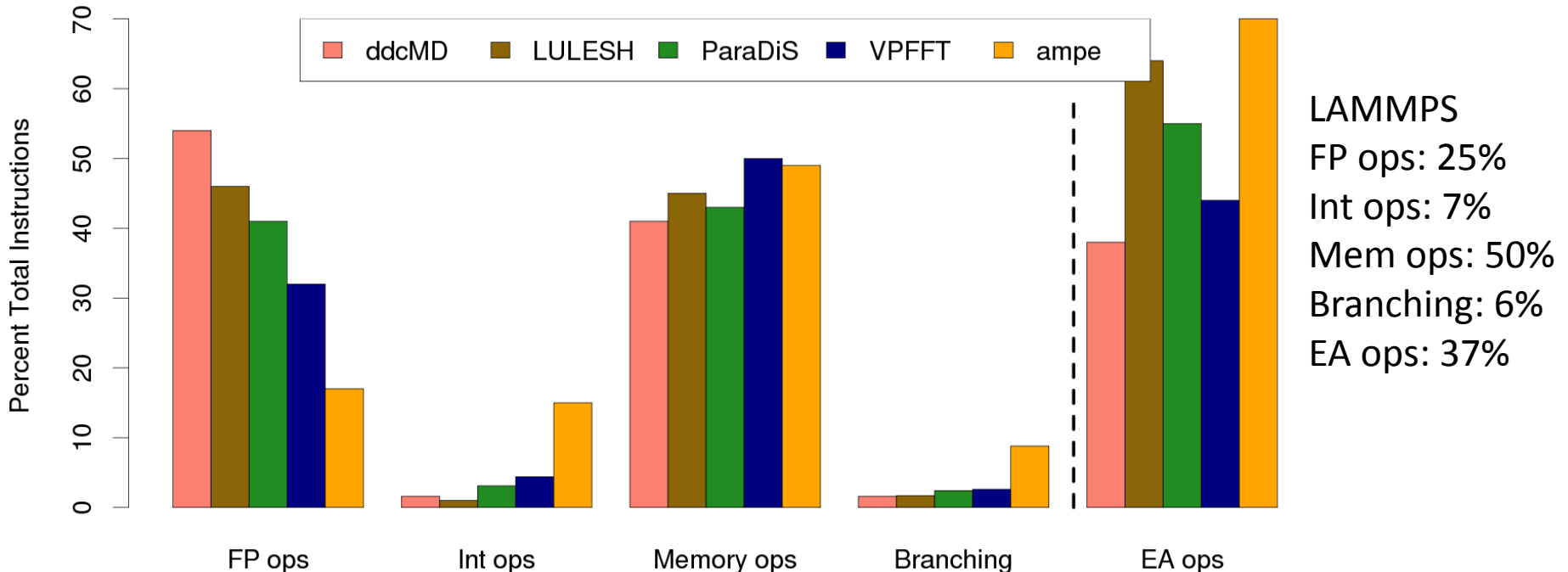
- Simulations are complex – need to replicate interactions within network, switches, cores, memory, *etc.*
 - Can take significant time to simulate large machines and scaling complexity is currently not well understood
 - Validating interaction of components is **very** challenging due to reproducibility and limited introspection in real systems
- Simulation of future architectures will require new components
 - GPU simulators currently being added to SST
 - MIC simulator will require new work
 - Novel memory architectures are beginning to be worked on
 - Working with vendors to introduce novel network architectures

Baseline Metrics (Application Signatures / Idioms – I really miss Allan)

- **How well does the proxy app suite represent the workload of the parent app?**
- **What really is the computational workload of the parent app? That is, the current implementation in the current computational ecosystem.**
- **Are these the right metrics to quantify the computational workload for an exascale computer? If not what are the right metrics?**
 - Fraction of peak performance
 - Data movement, during computation, across the network, between packages
 - Percentage of floating point that are SIMD
 - Working set size
 - Checkpoint fraction, frequency, and size
 - Time to checkpoint
 - Parallel FS (one big versus lots of little)
 - Flops / byte and flops / load
 - Integer / Floating point fraction
 - Cache reuse and utilization
 - TLB utilization
 - Message volume, frequency and size
 - MPI traffic, topology, hop distance, comm. / compute %, pending messages
 - Synchronization (network contention) and wait time
 - Metric for inherent quantity of application parallelism, threads / HW core
 - How well does the code scale (both weak and strong scaling)?

Metrics for computational work measure the behavior of the code within the computational ecosystem (e.g. HW/Stack/Compiler/etc.)

- Pin is a tool that measures utilization of specific functional units in the processor (e.g. floating point operations)
- Both ddcMD and LULESH are highly optimized codes. Pin analysis on entire code suite (see VG 2) in progress
- Analysis for Intel Sandy Bridge processor with Intel compiler (cab)
- LULESH percent vector utilization: Intel compiler = 8.7%, GCC = 0.15% (of FP)



Petascale (exascale) application developers must optimize for a **complex parallel machine**

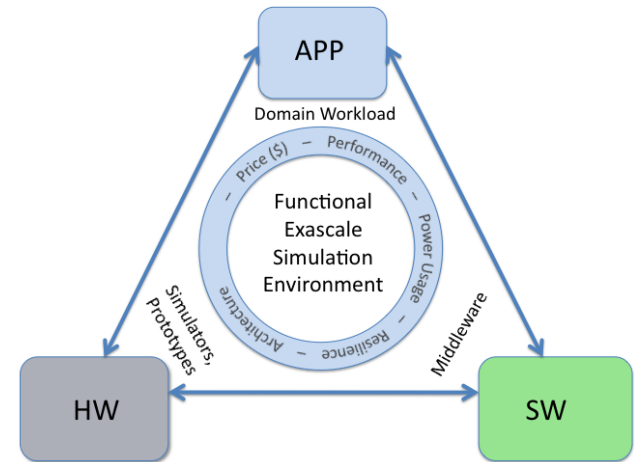
- **(Inter-Node) System-level.** No shared memory. Traditionally uses MPI to communicate data between disjoint address spaces.
- **(Intra-Node) Cores.** Modern nodes have **multiple CPU cores**. The work needs to be distributed across them. Old fashioned MPI is one option, but this increases surface to volume. Solutions such as OpenMP that acknowledge the shared address space (possibly NUMA) among the cores are probably preferred, especially in new code.
- **(Intra-Node) Threads.** Modern cores are supporting **multiple hardware threads** per node. Among other things, multiple threads per core cover latencies since some threads can typically proceed while others are stalled. Multiple threads may allow for better register usage, reduced pipeline stalls, etc.
- **(Intra-Node) SIMD.** We are now seeing quad-double SIMD units on intel and AMD hardware as well as BG/Q. Memory access need to be aligned to allow vector registers to be filled efficiently. **Throwing away SIMD instructions is instantly giving away a factor of 4 in performance.** On GPUs, warps are rather like SIMD instructions since all threads in a warp execute the same instruction.
- **(Intra-Node) Functional Units.** BG/Q has both an integer and a floating point unit for each core. **It is the integer unit that loads data so in order to do useful processing you need to keep both active** (to both load and process data). Note that any thread can only issue an instruction to one of the units per cycle so at least two threads are needed to fully exploit the units. It is also important to structure algorithms so that the use of the functional units is balanced.

What did we learn from creating petascale science apps and what does that mean for exascale?

- Problem: Fault tolerance is a problem at 10^5 and will be a much bigger problem at 10^8 :
 - Solution: Application assisted error recovery
 - Parity error triggers exception handler (like FPE)
 - Application knows what memory is “important” can catch exception and repair data
 - Exascale runtime will need to support task migration across nodes
- Problem: **Scaling** (absolutely crucial for exascale) requires very very good load balancing:
 - Solution: Decomposition based on Computational Work
 - Particle-based domain decomposition - processors own particles, not regions - allows decomposition to persist through atom movement
 - Maintain minimum communication list for given decomposition - allows extended range of “interaction”
 - Arbitrary domain shape - allows minimal surface to volume ratio for communication
 - Exascale: decomposition has to become dynamic and adaptive
- Problem: HW specific algorithms are crucial for performance but limit portability
 - E.g. Linked cells map better to current petascale systems than neighbor lists
 - Ordering neighbors within a cell exposes SIMD parallelism
- Problem: I/O does not work with too many files or one large file
 - Solution: Divide and concur, what is the optimal number of files?
 - Exascale: Dedicated checkpoint filesystem (flash?)

Productive Exascale Simulation requires the coordinated efforts of Domain Scientists, Computer Scientists and Hardware Developers

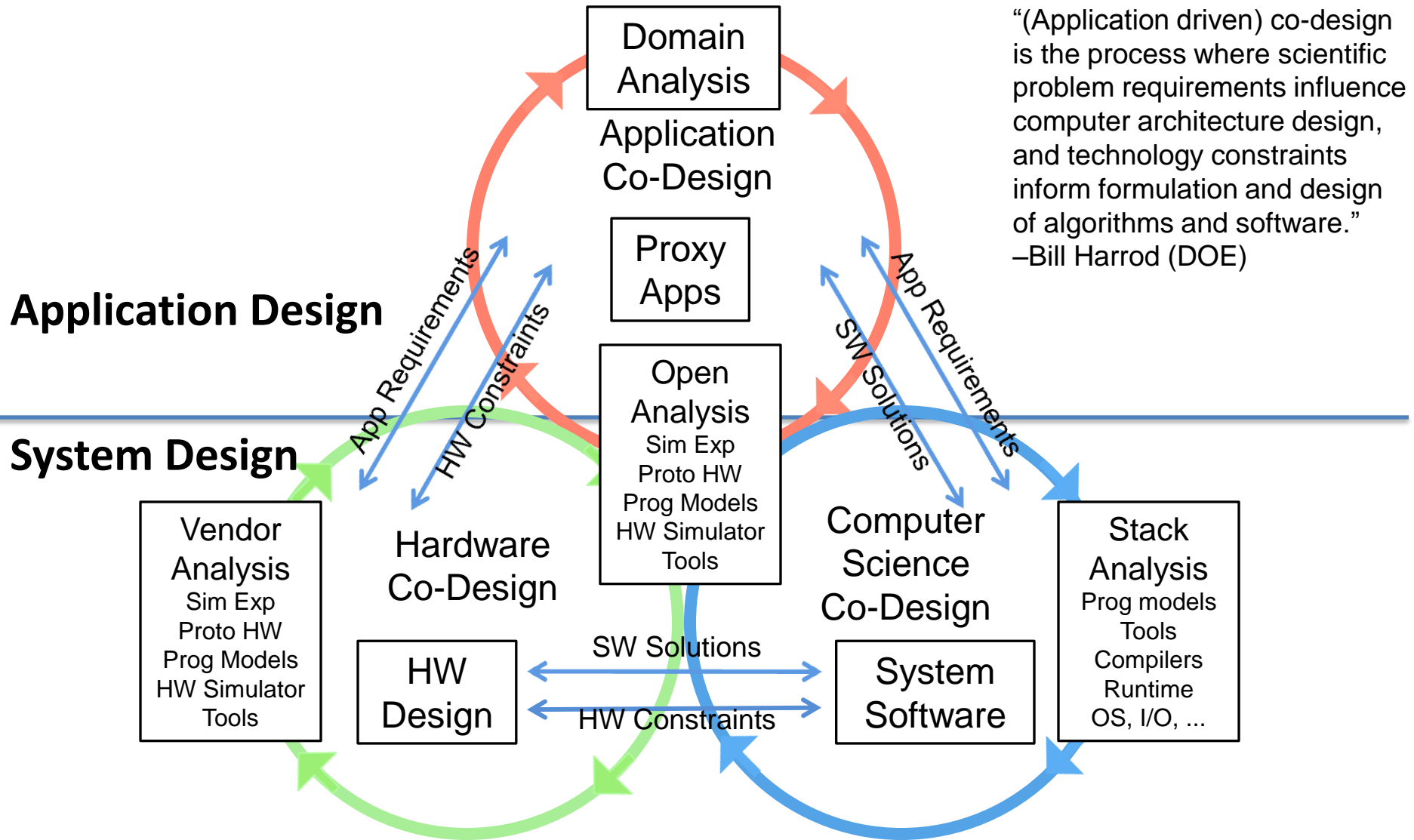
- Many, many-task coordination issues
 - Greater than one hundred million, more is different
 - Synchronization (essential for time evolution)
 - Stalls (keeping everyone working)
- Better exposure into hardware details for the exascale application developer
 - Compiler Interface
 - Simulators+Emulators+Tools measure code/ecosystem metrics
 - Are we defining the right metrics?
- Application developers need a better way to express (code) the computational work of the application into the exascale computational ecosystem
 - Better programming models (e.g. domain specific languages)
 - Runtime support for heterogeneous multi-program, multi-data (MPMD) applications
- The petascale science apps are NOT general apps. They have been painfully optimized for the petascale architecture by the app developer. How do we get exascale lessons learned into quotidian science applications (VASP, LAMMPS, ...)?
- The petascale codes already account for data movement, it is only going to get worse
 - Bandwidth to memory is scaling slower than compute
 - Memory access is dominating power
- The exascale codes will need to learn to adaptively respond to the system
 - Fault tolerance, process difference, power management, ...



Questions and Challenges for Performance Modeling and Simulation

- Need to include Emulation in the suite of tools
 - Current petascale systems enable us to “emulate” the behavior of exascale systems
 - Need to extend the metric of “performance” to include resilience and power management
 - Both computer system resilience and application resilience
 - What does a programming model for “error” tolerance look like?
 - Fault measurements and fault models are leading to “error” models
 - Can you model the behavior of a system (application) with an error model?
 - What is an error model for silent errors?
 - How do you measure silent errors? Bill Carlson does it for integer apps!
 - Need a concerted project including HW, SW (stack) and APP
 - Need better tools to analyze applications (and application codes) for the computational work as presented to the hardware – application signature
 - A given app code presents app work as seen through a given programming model, as compiled by a given compiler, within a given runtime, ...
 - Need better programming models and compilers to give the app developer both a better window into the hardware and a better way to expose the application work to the computer system
 - Need to go from developing tools, to asking “what ifs?” – trade off analysis – co-design!
 - Scaling matters! Complexity matters!
-

Model for the *Workflow of Co-design* between Application Co-design Centers, Vendors, and the broader Research Community



“(Application driven) co-design is the process where scientific problem requirements influence computer architecture design, and technology constraints inform formulation and design of algorithms and software.”
 –Bill Harrod (DOE)

Thank you for your attention!