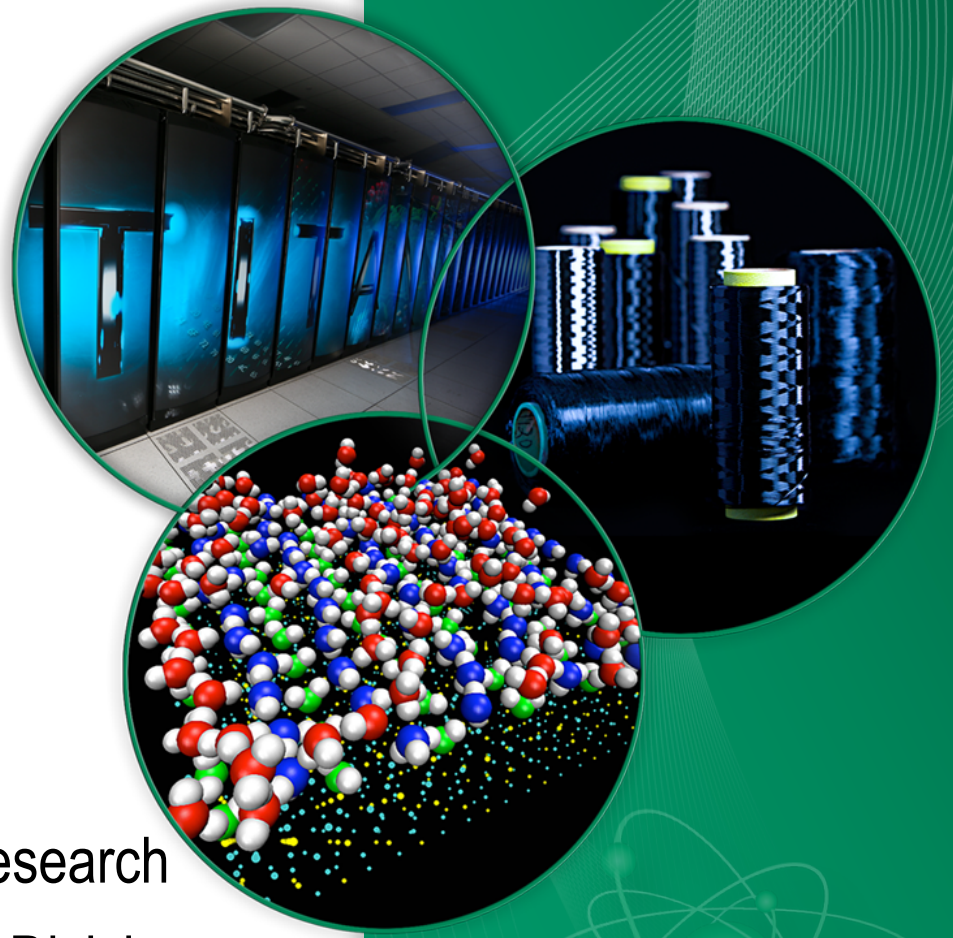
Some Assembly Required?

Thoughts on Programming at Extreme Scale



David E. Bernholdt
bernholdtde@ornl.gov

Group Leader, Computer Science Research

Computer Science and Mathematics Division
and National Center for Computational Sciences

Oak Ridge National Laboratory

Acknowledgements

Funding

- DOE
  - Advanced Scientific Computing Research
  - Fusion Energy Sciences
  - Nuclear Energy
  - Energy Efficiency and Renewable Energy
- DoD
  - Advanced Computing Initiative
  - DARPA
  - Extreme Scale Systems Center
- IARPA
- ORNL Directors R&D Fund
- National Science foundation

People & Organizations

- Pratul Agarwal, Matt Baker, Jay Billings, Swen Boehm, Wael Elwasif, Christian Engelmann, Samantha Foley, Al Geist, Robert Harrison, Oscar Hernandez, Chung-Hsing Hsu, Forest Hull, Christos Kartsaklis, Seyong Lee, Dong Li, Eric Lingerfelt, Qing Liu, Josh Lothian, Tiffany Mintz, Thomas Naughton, BH Park, Neeti Pokhriyal, Steve Poole, Nagi Rao, Stephen Scott, Aniruddha Shet, Geoffroy Vallee, Jean-Charles Vasnier, Sudharshan Vazhkudai, Jeff Vetter, Chao Wang, Matt Wolf
- ANL, LBNL, LLNL, PNNL, SNL, ITER
- Binghamton, Indiana, LSU, NCSU, OSU, Rutgers, Alaska, Carlos III de Madrid, Florida, Maryland, Tennessee, Waterloo, Virginia State
- Galois, Tech-X

This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Cavets

- I'll be talking about R&D projects at ORNL and elsewhere
- I'll be giving my opinions
- Don't blame good projects for my crazy opinions
  - Nor their sponsors

Challenges of Programming at Extreme Scale

- System level
  - Massive parallelism
  - Heterogenity
    - Different types of CPUs
    - Compute in memory, NIC, etc.
  - Data movement
  - Reliability
  - Power
- Ecosystem level
  - Diversity
  - Flux
- Programming languages
  - Expressiveness
  - Higher level abstractions
  - Mapping to resources
- Programming models
  - Performance portability
  - Asynchrony, task/event driven
  - One-sided communication
  - Resource Management
- Tools

OpenACC – Portable Accelerator Programming

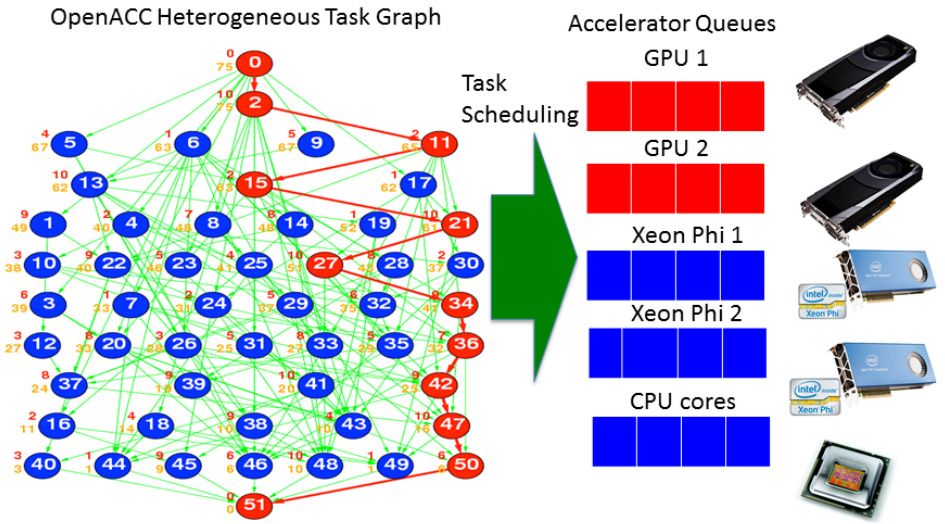
- What is it?
  - API and directives to allow regions of code to be offloaded to accelerators
  - Supports C, C++, Fortran
  - Industry standard, with multiple implementations
- Why is it important?
  - Unified approach to accelerator directives
  - Portability and simplicity
  - Same code base runs on platforms with and without accelerators
- Where is it going?
  - Aggressive cadence on standard releases
  - Eventual incorporation of concepts into OpenMP
    - OpenMP has a much slower release cadence

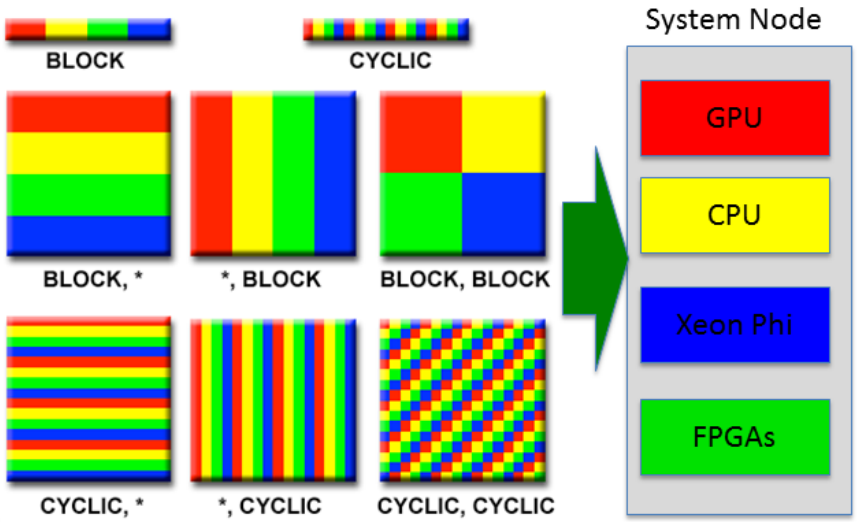
OpenACC
DIRECTIVES FOR ACCELERATORS

```
!$acc do vector(4)
  do k = k2begin,NZ_LOCAL
    kglobal = k + offset_k
!$acc do gang, vector(4)
    do j = 2,NY
!$acc do gang, vector(16)
      do i = 2,NX
```

Near-Term Focus Areas for OpenACC

- Improved support for C++
- Hybrid OpenACC+OpenMP programming
- A tasking model for OpenACC
  - Multiple accelerators
  - Heterogeneous nodes
- Heterogeneous data distributions
  - Placement within accelerator memory hierarchy
  - Distribution across multiple accelerators





Engaging the OpenACC and OpenMP Communities

What ORNL is doing…

- ORNL is a member of both the OpenACC and OpenMP standards organizations
- Pushing OpenACC to better address application needs
- Pushing to mature OpenACC experience for incorporation into OpenMP
- Looking to expand the community

OpenACC DIRECTIVES FOR ACCELERATORS

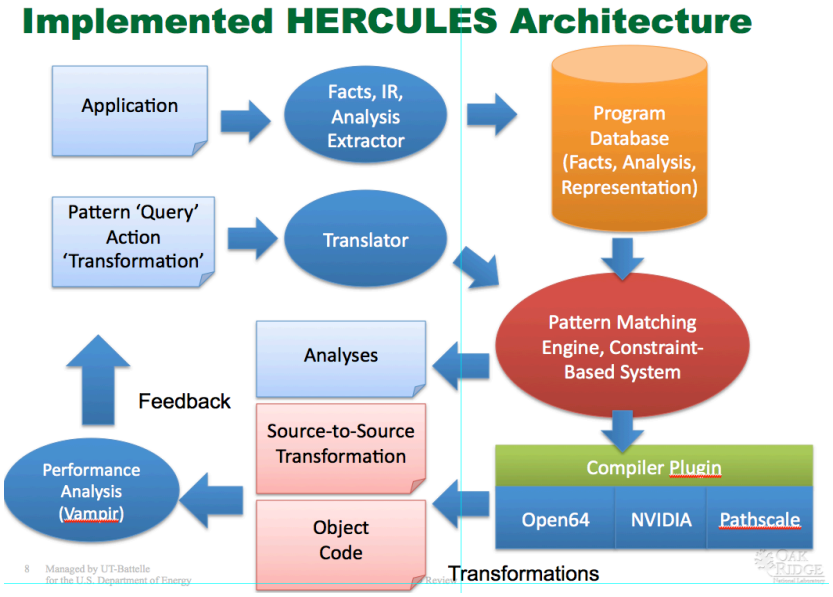
OpenMP

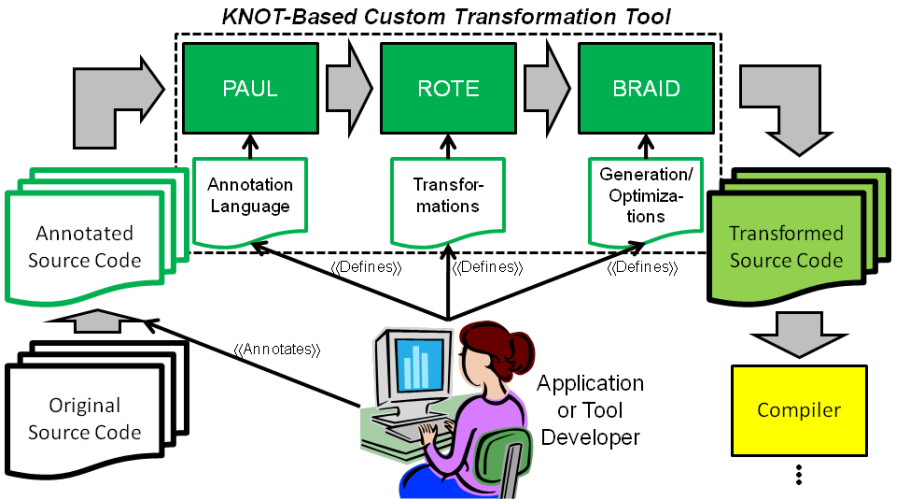
How you can help…

- Spend some time thinking about it *now*
- What are your key kernels to offload to accelerators?
  - Can you share them?
- What are the impediments to expressing them?
  - Getting performance?
- Explain your issues to the organization
  - Provide concrete examples
- Join up!

Code Transformation Tools

- Capturing and reusing the transformations required to migrate code
  - Source-to-source and back-end
  - Based on pattern matching and constraints
  - Database to collect program “facts”
- Giving programmers the tools to evolve their code systematically
  - Source-to-source and multi-language code generation
  - Based on term rewriting
  - Guidance via annotation user-defined annotation languages
  - Creation of transformations by example (semantic diff)
- *Early stages – much more to do*



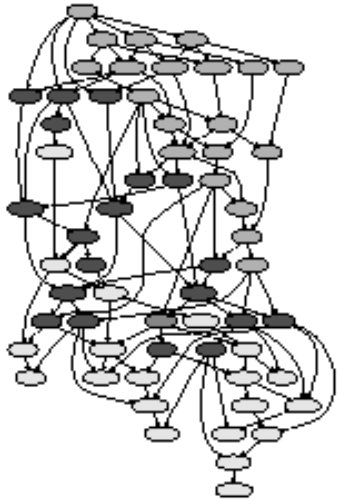


Programming Models Research in DOE

Features
- Lightweight threads
- One-sided communication and active messages
- Task/event driven

Runtimes
- Open Community Runtime
- GASNet-EX
- SEEC
- HPX

Programs
- 2012 X-Stack program
  - http://www.xstackwiki.com
- Fast Forward, Design Forward
  - https://sites.google.com/a/lbl.gov/exascale-initiative/



DAG representation of a Cholesky decomposition. From Asim YarKhan, Dynamic Task Execution on Shared and Distributed Memory Architectures

DEGAS (Kathy Yelick)
Hierarchical and resilient programming models, 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)
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 Schultz)
Tools for debugging and analysis of performance, power, and energy.

DynAX (Rishi Khan)
Novel programming models, dynamic adaptive execution models and runtime systems.

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.

New Programming Languages

X10

- Many to choose from; selected two exemplars
- Chapel (Cray), X10 (IBM)
  - Initiated by DARPA HPCS program; development continues
  - Designed for HPC technical computing
  - Open source with growing communities of outside users, contributors
- Asynchronous partitioned global address space (APGAS)
  - Address key exascale issues: concurrency, asynchrony, data distribution, synchronization, locality control
- "Multiresolution" approach (adjustable levels of abstraction)
  - Map to machine in lower SW layers, compiler/runtime

Domain-Specific Languages

- The ultimate in programming is probably direct translation of your equations into a working program
- The “program” is closer to the way the scientists think
  - Higher levels of abstraction, more compact expression
  - Preserve domain-specific information which would be lost in translation to a general-purpose language
- Higher-level specification of computation gives compiler more leeway in mapping to target platform
- Use domain-specific information to improve implementation
  - Knowledge & constraints may enable more/better/easier optimizations
- Relies significantly on compiler/runtime to do a good job!

Many-Body Methods in Quantum Chemistry

- Based on contractions of large, high-rank tensors
  - Many permutation operations and symmetries
- Hand coding is tedious, error-prone
- Naïve implementations orders of magnitude slower than optimized
- Would like to be able to explore variants, quickly and easily

$$
\begin{aligned}
0 = {} & \langle ab||ij\rangle + \sum_c \left(f_{bc} t_{ij}^{ac} - f_{ac} t_{ij}^{bc} \right) - \sum_k \left(f_{kj} t_{ik}^{ab} - f_{ki} t_{jk}^{ab} \right) + \\
& \frac{1}{2} \sum_{kl} \langle kl||ij\rangle t_{kl}^{ab} + \frac{1}{2} \sum_{cd} \langle ab||cd\rangle t_{ij}^{cd} + P(ij)P(ab) \sum_{kc} \langle kb||cj\rangle t_{ik}^{ac} + \\
& P(ij) \sum_c \langle ab||cj\rangle t_i^c - P(ab) \sum_k \langle kb||ij\rangle t_k^a + \\
& \frac{1}{2} P(ij)P(ab) \sum_{klcd} \langle kl||cd\rangle t_{ik}^{ac} t_{lj}^{db} + \frac{1}{4} \sum_{klcd} \langle kl||cd\rangle t_{ij}^{cd} t_{kl}^{ab} - \\
& P(ab) \frac{1}{2} \sum_{klcd} \langle kl||cd\rangle t_{ij}^{ac} t_{kl}^{bd} - P(ij) \frac{1}{2} \sum_{klcd} \langle kl||cd\rangle t_{ik}^{ab} t_{jl}^{cd} + \\
& P(ab) \frac{1}{2} \sum_{kl} \langle kl||ij\rangle t_k^a t_l^b + P(ij) \frac{1}{2} \sum_{cd} \langle ab||cd\rangle t_i^c t_j^d - P(ij)P(ab) \sum_{kc} \langle kb||ic\rangle t_k^a t_j^c + \\
& P(ab) \sum_{kc} f_{kc} t_k^a t_{ij}^{bc} + P(ij) \sum_{kc} f_{kc} t_i^c t_{jk}^{ab} - \\
& P(ij) \sum_{klc} \langle kl||ci\rangle t_k^c t_{lj}^{ab} + P(ab) \sum_{kcd} \langle ka||cd\rangle t_k^c t_{ij}^{db} + \\
& P(ij)P(ab) \sum_{kcd} \langle ak||dc\rangle t_i^d t_{jk}^{bc} + P(ij)P(ab) \sum_{klc} \langle kl||ic\rangle t_l^a t_{jk}^{bc} + \\
& P(ij) \frac{1}{2} \sum_{klc} \langle kl||cj\rangle t_i^c t_{kl}^{ab} - P(ab) \frac{1}{2} \sum_{kcd} \langle kb||cd\rangle t_k^a t_{ij}^{cd} - \\
& P(ij)P(ab) \frac{1}{2} \sum_{kcd} \langle kb||cd\rangle t_i^c t_k^a t_j^d + P(ij)P(ab) \frac{1}{2} \sum_{klc} \langle kl||cj\rangle t_i^c t_k^a t_l^b - \\
& P(ij) \sum_{klcd} \langle kl||cd\rangle t_k^c t_i^d t_{lj}^{ab} - P(ab) \sum_{klcd} \langle kl||cd\rangle t_k^c t_l^a t_{ij}^{db} + \\
& P(ij) \frac{1}{4} \sum_{klcd} \langle kl||cd\rangle t_i^c t_j^d t_{kl}^{ab} + P(ab) \frac{1}{4} \sum_{klcd} \langle kl||cd\rangle t_k^a t_l^b t_{ij}^{cd} + \\
& P(ij)P(ab) \sum_{klcd} \langle kl||cd\rangle t_i^c t_l^b t_{kj}^{ad} + P(ij)P(ab) \frac{1}{4} \sum_{klcd} \langle kl||cd\rangle t_i^c t_k^a t_j^d t_l^b .
\end{aligned}
$$

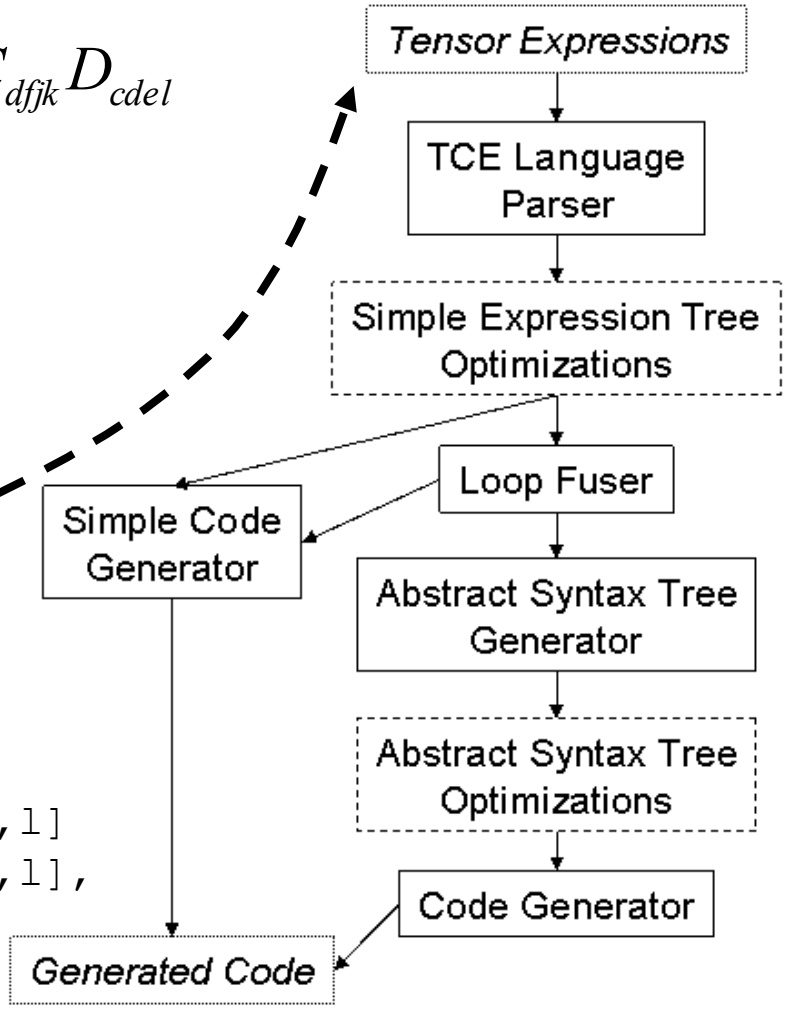
Theory	# Terms	F77 LOC	First Impl.
CCD	11	3,209	1978
CCSD	48	13,213	1982
CCSDT	102	33,932	1988
CCSDTQ	183	79,901	1992

Tensor Contraction Engine (TCE)

TCE Language and Software Architecture

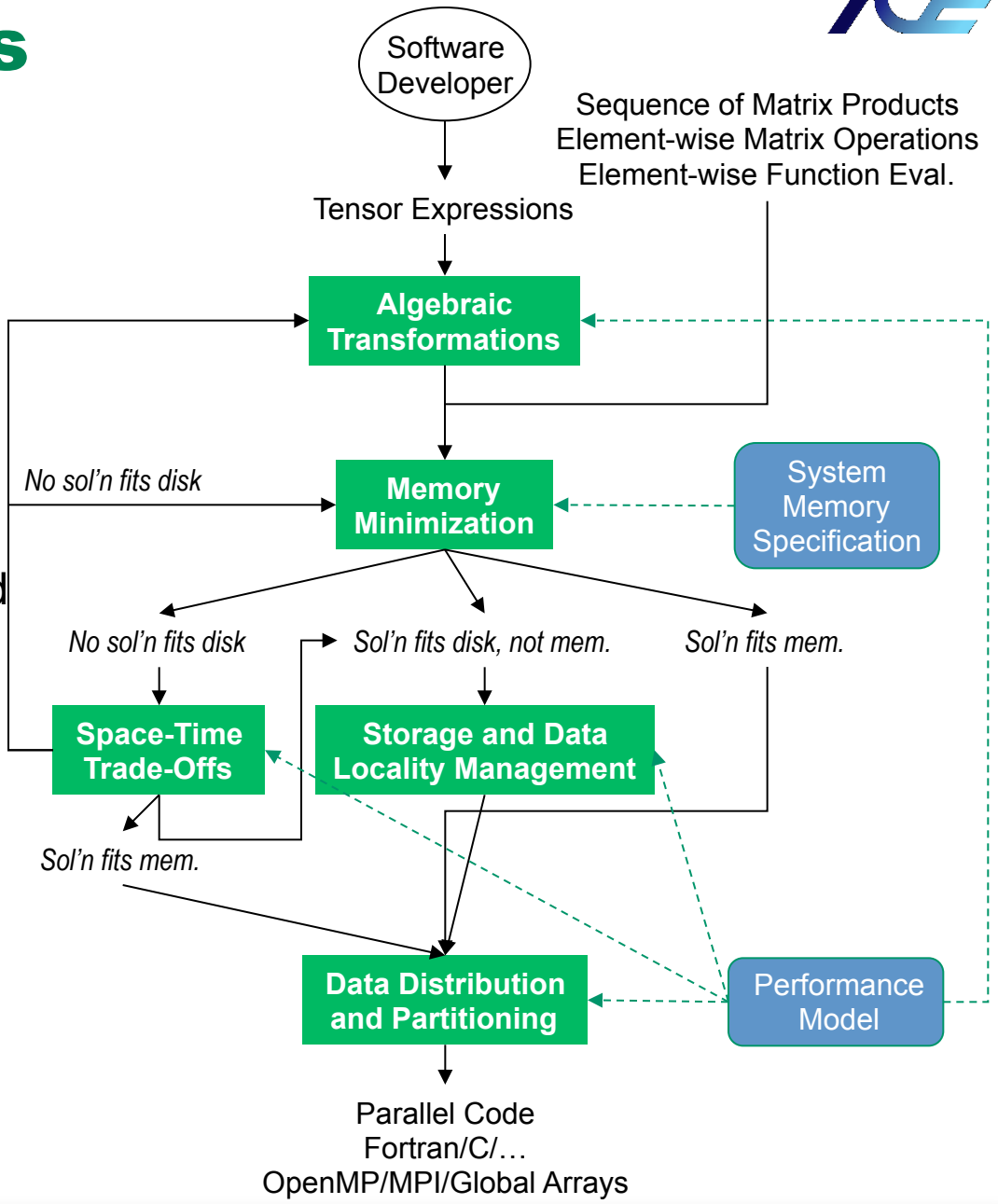
$$S_{abij} = \sum_{cefkl} A_{acik} B_{befl} C_{dfjk} D_{cdel}$$

```
range V = 3000;
range O = 100;

index a,b,c,d,e,f : V;
index i,j,k,l : O;

mlimit = 100GB;

procedure P(in A[V,V,O,O], in B[V,V,V,O],
            in C[V,V,O,O], in D[V,V,V,O],
            out S[V,V,O,O])=
begin
  S[a,b,i,j] == sum[ A[a,c,i,k] * B[b,e,f,l]
                   * C[d,f,j,k] * D[c,d,e,l],
                     {c,e,f,k,l}];
end
```



TCE Optimizations

- Algebraic Transformations
  - Minimize operation count (ICCS'05, ICCS'06)
- Memory Minimization
  - Reduce intermediate storage via loop fusion (LCPC'03)
- Space-Time Transformation
  - Trade-offs between storage and recomputation (PLDI'02)
- Data Locality Optimization
  - Optimize use of storage hierarchy via tiling (ICS'01, HiPC'03, IPDPS'04)
- Data Dist./Comm. Optimization
  - Optimize parallel data layout (IPDPS'03)
- Integrated System
  - (SC'02, Proc. IEEE 05)



Embedded DSLs and Expressiveness

- Convenient to be able to embed DSLs into general purpose languages

Simple TCE input

```
range V = 3000;
range O = 100;

index a,b,c,d,e,f : V;
index i,j,k,l : O;

mlimit = 100GB;

procedure P(in A[V,V,O,O], in B[V,V,V,O],
            in C[V,V,O,O], in D[V,V,V,O],
            out S[V,V,O,O])=
begin
  S[a,b,i,j] == sum[ A[a,c,i,k] * B[b,e,f,l]
                   * C[d,f,j,k] * D[c,d,e,l],
                     {c,d,e,f,k,l}];
end
```

Chapel version by Brad Chamberlain, Cray (working code!)

```
config const V = 3000,
             O = 100;
const DV = 1..V,
      DO = 1..O;
const DVVOO = [DV, DV, DO, DO],
      DVVVO = [DV, DV, DV, DO];
var A, C, S: [DVVOO] real,
       B, D: [DVVVO] real;
forall (a, b, i, j) in DVVOO do
   S(a,b,i,j) = + reduce [(c,d,e,f,k,l) in [DV,DV,DV,DV,DO,DO]]
                    (A(a,c,i,k) * B(b,e,f,l) * C(d,f,j,k) * D(c,d,e,l));
);
```

The Take-Away on DSLs

- Can provide a huge productivity boost to the domain scientists
  - If you identify the right abstractions
- Can facilitate mapping onto diverse hardware, optimizations
  - If you have a good implementation
- DSLs are a *lot* of work to do well
  - Need to be able to re-use the product (a lot)
  - Just beginning to see some tools to facilitate this
- Requires close collaboration of domain and computer scientists
- Even simpler code generators can be extremely powerful
  - Typically for domain scientist productivity; less so for performance
- TCE experience
  - Reduces impl. of new method from O(months to years) to O(hours)
  - ~3M LOC in NWChem (~4.5M LOC) is generated by a simplified TCE

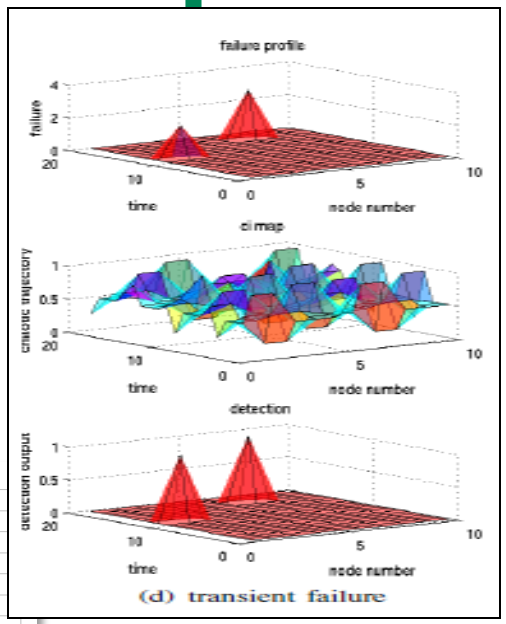
Resilience Concerns at Extreme Scale

- Every component of an HPC system can fail
- More components → more frequent failures
  - If node MTBF = 100,000 hrs (11.4 yrs!)
  - A 100,000 node system will experience failures hourly (on average)
- End of Dennard Scaling → scale out by adding cores/nodes
- More challenges at extreme scale
  - Shrinking lithographic feature size → greater vulnerability to upsets
  - Near-threshold voltage operation → greater vulnerability to upsets
  - Greater vulnerability to upsets → more transient errors (silent data corruption)
  - Power constraints → hardware resilience solutions less attractive
  - Hard (expensive) for storage systems to keep pace w/ system growth

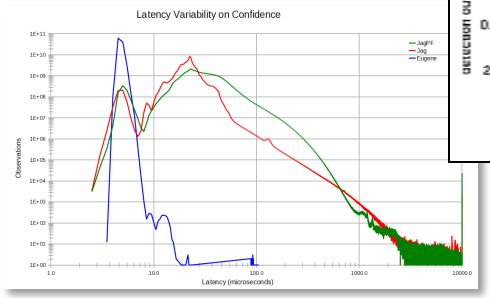
Understanding Faults and Their Impacts

- **Diagnostics**
  - Baselines, stress testing, runtime
  - Mathematical and empirical approaches
  - *Future:* fault models, statistical confidence in app. exeuction
- **Sensitivity analysis & propagation**
  - How different types of faults impact system sw and applications
  - How errors propagate once they occur
  - *Future:* differentiated protection
- **Performance under fault**
  - Times for propagation, detection, response
  - xSim framework for scalable modeling of MPI applications (100M+ procs) with fault injection
  - *Future:* evaluation of FT approaches

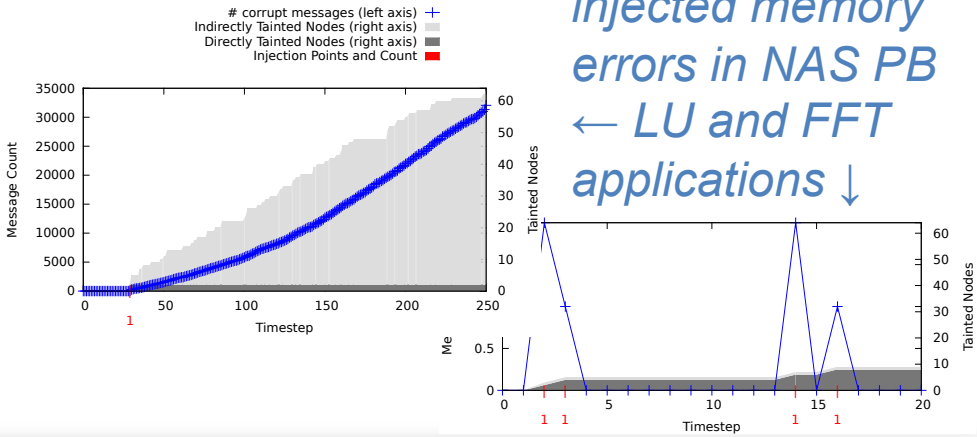
Using chaotic identity maps to detect failures →



SystemConfidence measurement of network latency variability ↓

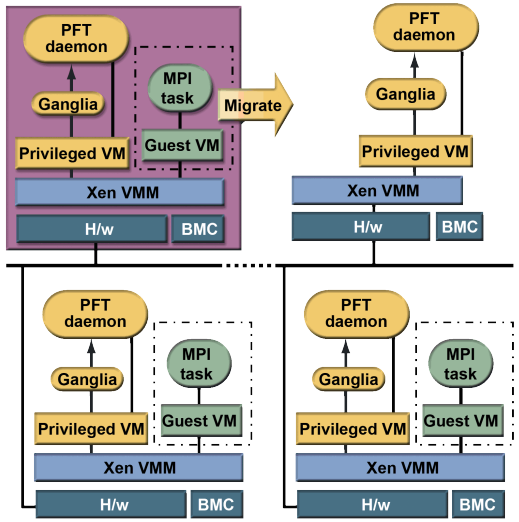


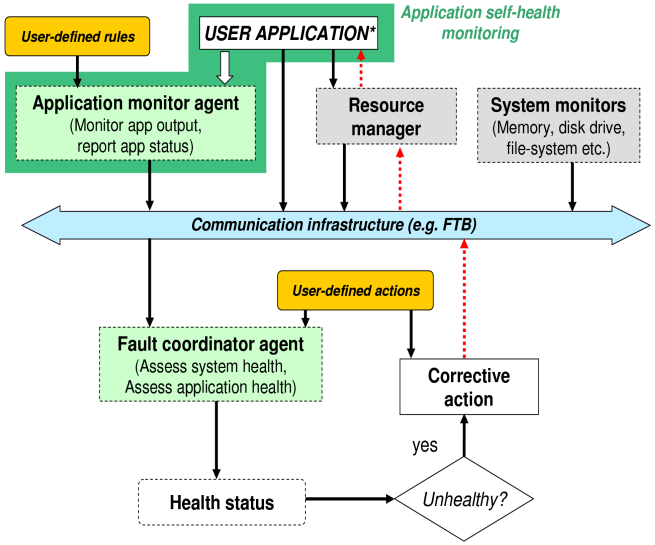
Propagation of injected memory errors in NAS PB ← LU and FFT applications ↓



Infrastructure for System and Application Resilience

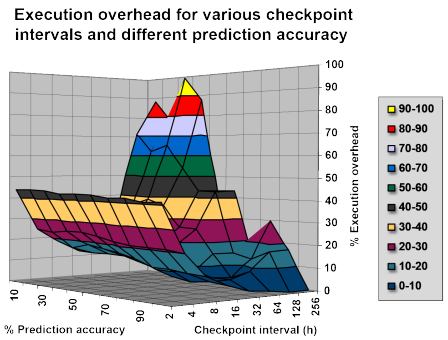
- **System-level:** proactive process migration
  - Monitoring and prediction
  - Virtualization to facilitate migration
- **Library-level:** MPI user-level fault mitigation
  - Robust implementation and demonstration of ULFM
  - Resilient runtime layer
- **Application-level**
  - Framework-integrated intelligent fault handling
  - Non-intrusive framework for user-guided fault detection and response
- Fault awareness and cross-layer coordination are key
- *Future:* integrated OS facilities for fault awareness, autonomic management infrastructure



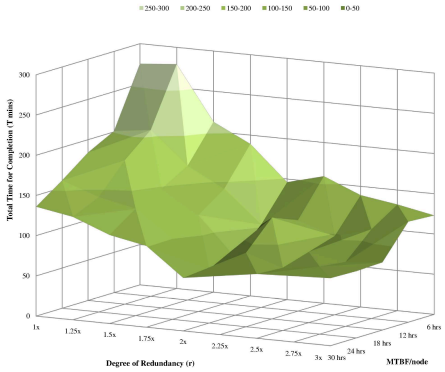


Trade-Offs in Responding to Faults

- Many ways to respond to a given fault
  - Different techniques
  - Different layers of the software stack
  - Hardware vs software
- How do we chose the best approach?
- *Today:* understanding trade-offs
- How to provide enough control to make it possible?
- *Future:* selecting and controlling responses

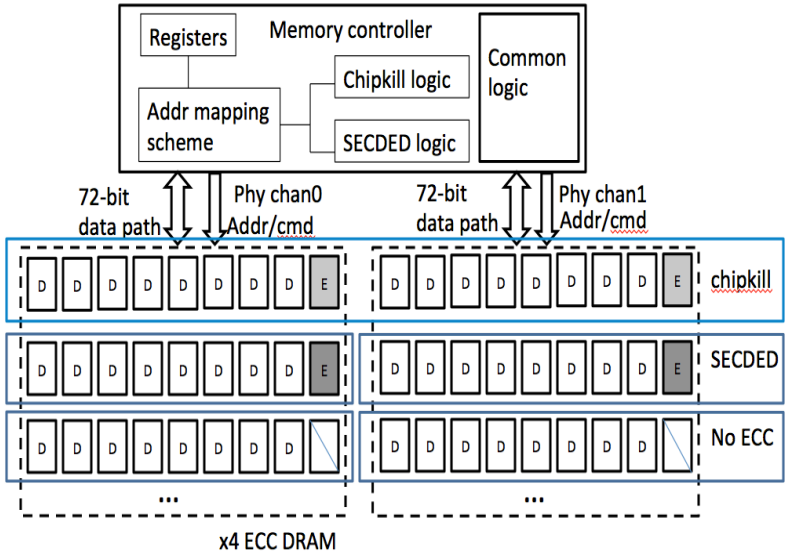


Overheads of proactive process migration and checkpoint/restart



Time to completion for redundant MPI vs node MTBF

Tailoring of ECC usage depending on algorithmic resilience

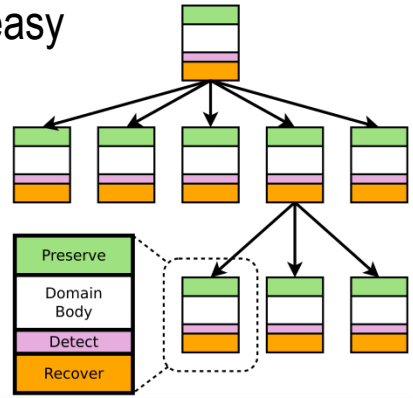


Today's Advice on Resilience for Tomorrow (Extreme-Scale)

- Resilience will require cooperation between hardware, system software, and application software
  - Think about where things can be done most effectively
  - The further down you push the response, the more general it must be
- Faults will no longer be rare events
  - Possibly encountering more faults during recovery
- Seek strategies with minimal fault-free overhead
  - Be willing to pay when faults occur
- Local failure → local recovery
  - Global coordinated checkpoint/restart is not sustainable
- Worry about transient/soft errors as well as hard
  - Resilience looks increasingly like good SW engineering and good verification
  - Add verification everywhere you can (even if not normally invoked)
- Don't assume you'll be notified when a fault occurs
  - Silent data corruption, other transient faults
- Use fault injection to understand your application

Resilience Strategies to Consider

- Fault prediction with process migration
  - Protects against hard errors
  - System level
  - Pros: works well if infrastructure is available
  - Cons: useful fault prediction is hard
- Uncoordinated checkpoint/restart
  - Protects against hard & soft errors
  - System level w/ user control
  - Pros: uncoordinated
  - Cons: message logs
- Containment domains
  - Protects against soft errors
  - User level w/ infrastructure support
  - Pros: conceptually easy
  - Cons: user level



Sullivan, et al. ***Containment Domains: A Full-System Approach to Computational Resiliency.*** *Technical report TR-LPH-2011-001, The University of Texas at Austin.*

- MPI user-level fault mitigation (ULFM)
  - Protects against hard errors
  - User level w/ infrastructure support
  - Pros: nearing standardization
  - Cons: small (but important) step
- Redundant MPI
  - Protects against hard and soft errors
  - System level
  - Pros: strong protection
  - Cons: high cost
- Programmer-guided reliability
  - Protects against soft errors
  - User level
  - Pros: easy to implement
  - Cons: effectiveness depends on programmer
- Algorithm-based fault tolerance (ABFT)
  - Protects against soft (and hard) errors
  - User level (w/ infrastructure support)
  - Pros: potentially very effective
  - Cons: can be hard to formulate

Programmer-Guided and Algorithm-Based Fault Tolerance

Programmer-Guided Reliability

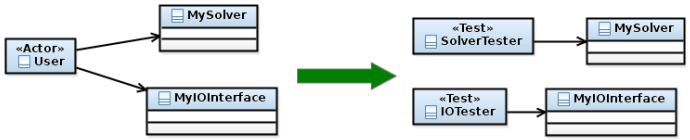
- Add "error detectors" to code
  - May also correct errors
  - Primarily based on properties of data/ algorithms
  - Alternate ways to compute qtys
  - Verify computed results
  - c.f. contracts and assertions
  - c.f. code verification
- R&D topics
  - Cost and efficacy of detectors may vary widely
  - Techniques to manage costs
- Examples
  - Verify matrix factors recover original
  - Output should be in the range [-1, 1]

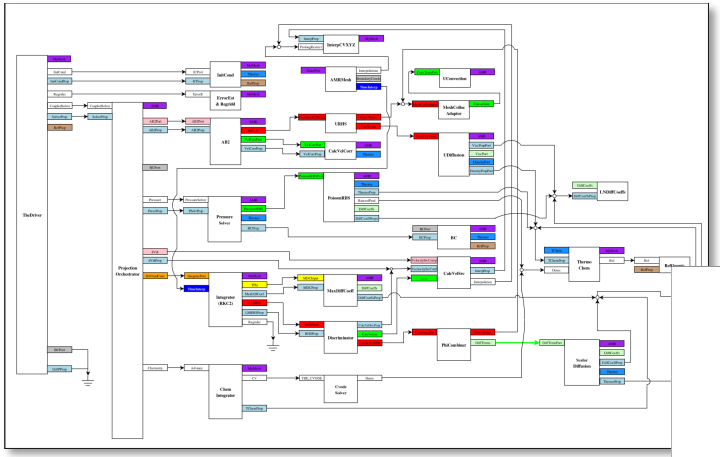
Algorithm-Based Fault Tolerance

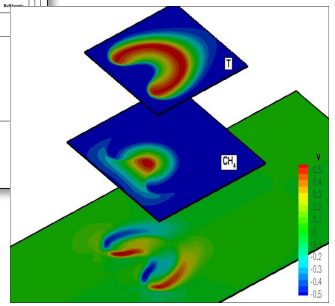
- Re-design algorithms to build in features to improve robustness
- May leverage innate resilience of algorithm
  - e.g. many iterative algorithms will still converge when data is perturbed
- Examples
  - Adding checksum rows/columns for dense linear algebra operations
  - Identify critical sections of algorithms or data structures and better protect those; tolerate errors elsewhere

Engineering of Scientific Software

- **Software engineering**
  - Understanding and improving real-world development practices
  - Test-driven modeling

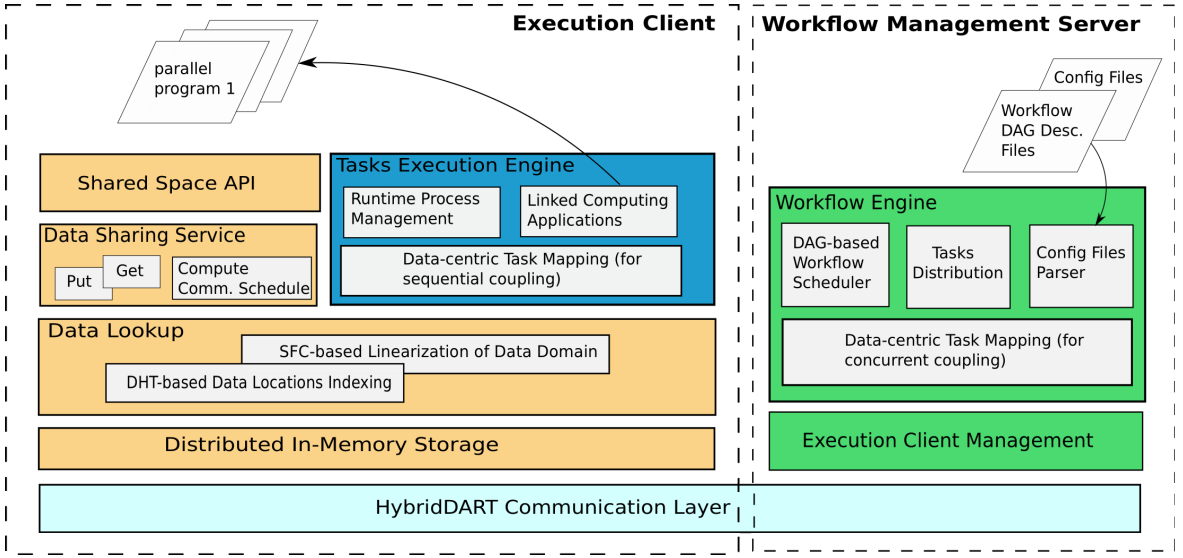






CCA-based combustion application "wiring diagram" and results
Courtesy Cosmin Safta, (SNL)

- **Software architecture**
  - Managing complexity
  - Components, service-oriented architecture
  - Enable scientists to do things previously thought intractable



Co-located DataSpaces with HybridDART transport layer

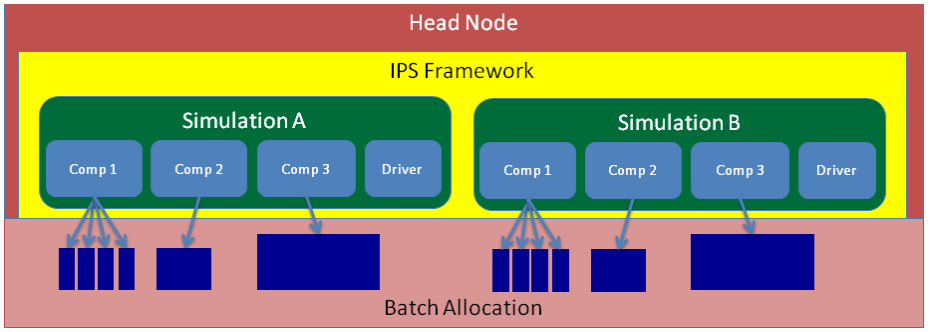
IPS Framework for Loosly-Coupled Simulation with Multi-Level Parallelism

Features

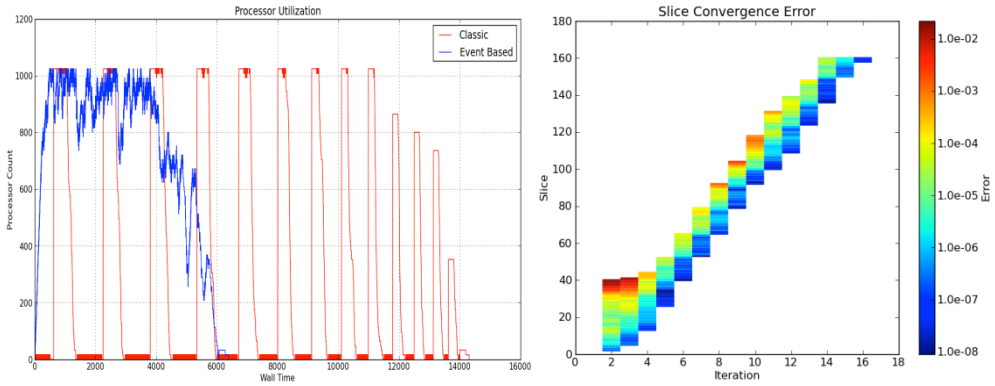
- Python-based component framework, based on CCA
- Wrap existing single-physics binaries as components (non-intrusive)
- Information exchange via files
- Asynchronous event model for inter-component information exchange

Impact

- Allowed physicists to easily design and carry out simulations they would not have considered feasible
- Novel multi-level parallelism has allowed new parallelization of existing applications, new algorithms
- Interest and adoption in international fusion community, battery modeling, structural modeling, …



Parallelism in the IPS: *(1) tasks are parallel applications, (2) a component can launch multiple tasks, (3) multiple components can run concurrently, (4) multiple simulations can execute concurrently*



Dependency-driven Parareal reduces wallclock time and improves resource utilization efficiency (left). Moving window implementation reduces overall resource utilization (right).

NiCE: Making HPC Modeling and Simulation Accessible

Challenge

- "Analysts" confronted with complex (HPC) M&S tools on complex (HPC) systems

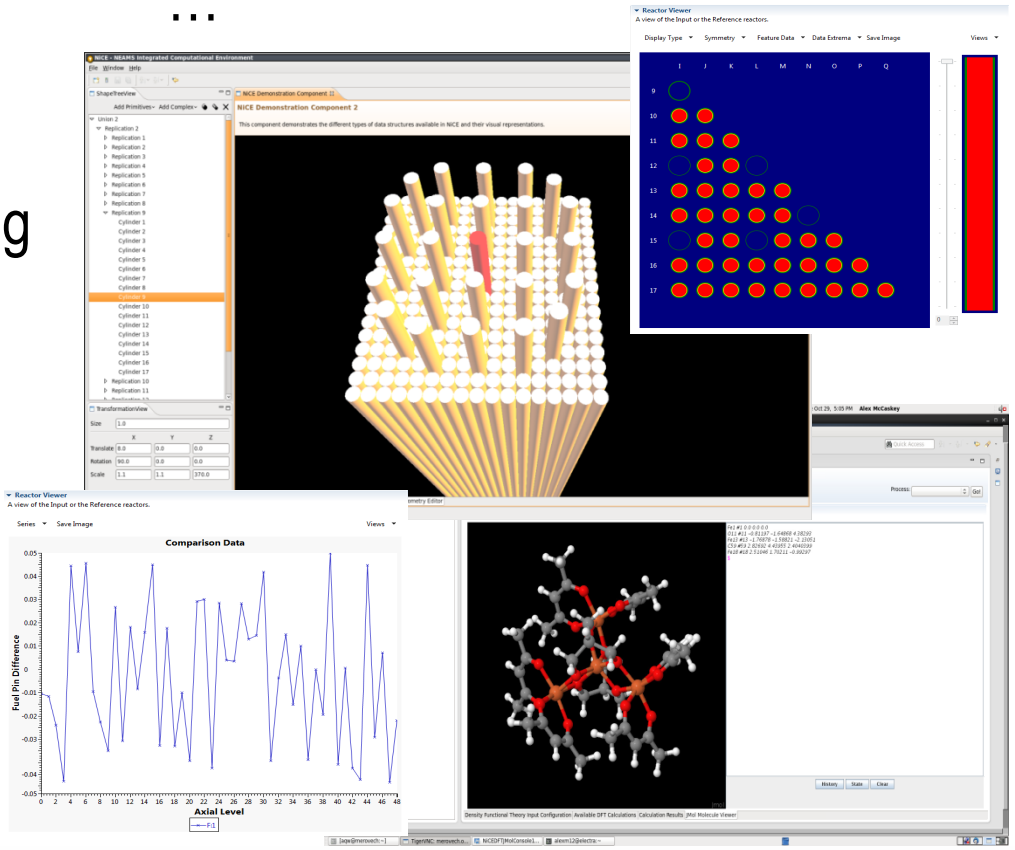
Response

- Plugin and component-based user "end station" for scientific computing
- Assist with job preparation, execution, analysis

Approach

- Leverage Open Services Gateway Initiative (OSGi) framework
- NiCE plug-ins described declaratively (XML, etc.) or in Java and C/C++

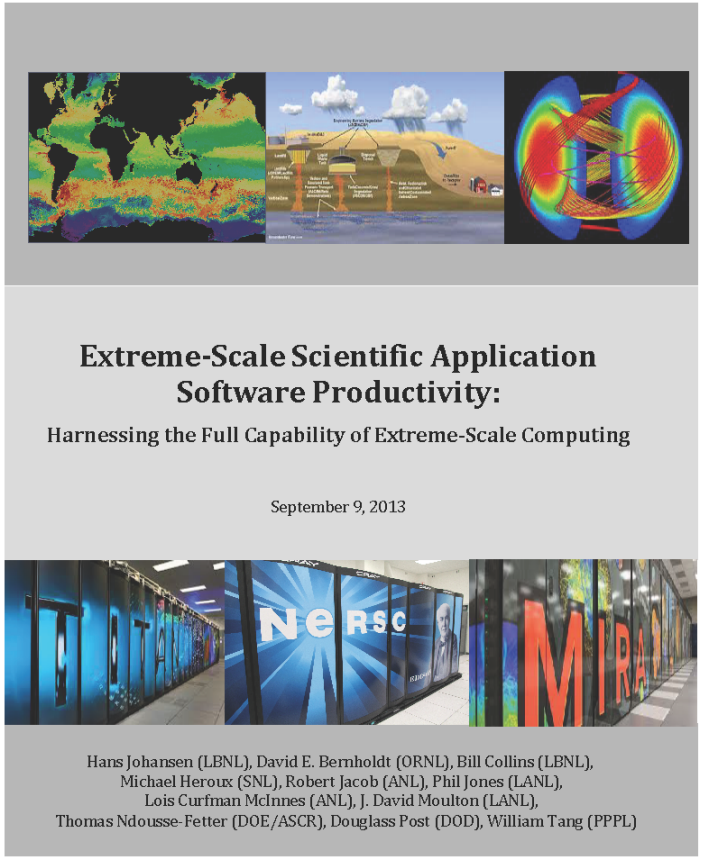
- Applications in reactor modeling, battery simulation, quantum chemistry, quantum computing, …



Software Productivity for Extreme-Scale Science (SWP4XS)

- Characterizing extreme-scale productivity impediments
- Developing performance-portable numerical software and applications
- Creating ecosystems of trusted high-performance libraries and tools
- Devising appropriate software productivity and engineering practices for HPC
- Programmatic needs
- Outreach needs

DOE ASCR Workshop
13-14 January 2014



Summary

- HPC hardware is entering a period of diversity and flux
- Programming environments for HPC are in flux
- But some trends are clear…
  - Asynchronous, task-based execution models
  - Higher levels of expression to facilitate (automatic) mapping to hardware
  - Silent data corruption will become a larger concern
  - Resilience will be handled (at least partly) at the user level
  - Agility will be required
- Still very much research, but tools & libraries exist to experiment with
- Engineering of scientific software is a long-standing concern that gets some attention from time to time in DOE

Reply hazy, try again

Questions