SAND2011-6111C

Challenges in the utilization of extreme-scale high performance computers by quantum chemistry applications

Pushing the Envelope: Computational Chemistry at the Petascale and Beyond

242nd ACS National Meeting

Aug. 28 – Sep. 1, 2011

Denver, CA

Curtis Janssen, Sandia National Laboratories

cljanss@sandia.gov

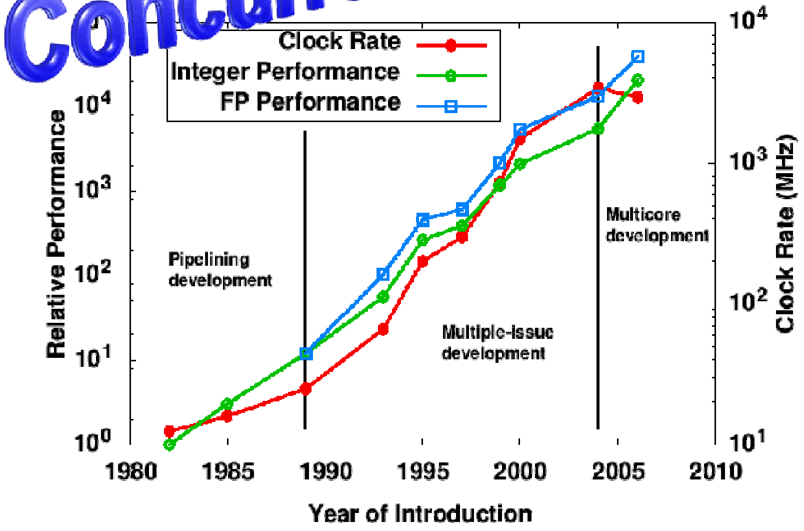
Unlimited Release

SAND 2011-XXXXX

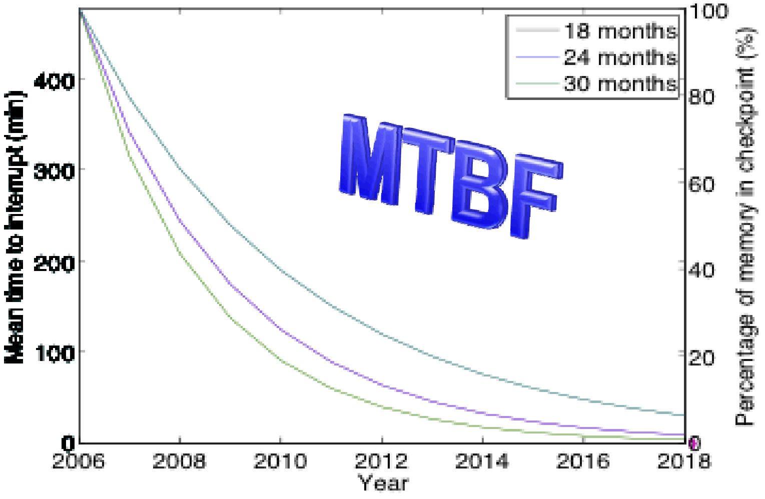
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Challenges impacting exascale application performance

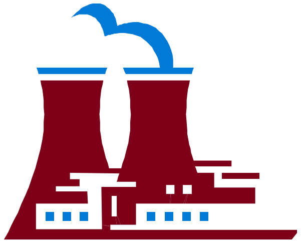
Concurrency



MTBF

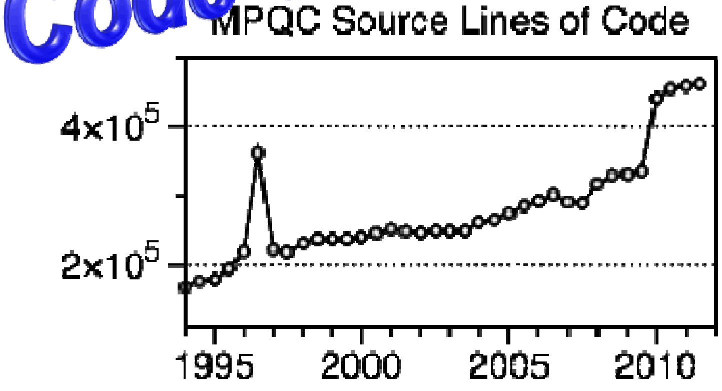


Schroeder and Gibson, Journal of Physics: Conference Series, **78** (2007) 012022, SciDAC 2007 Proceedings.



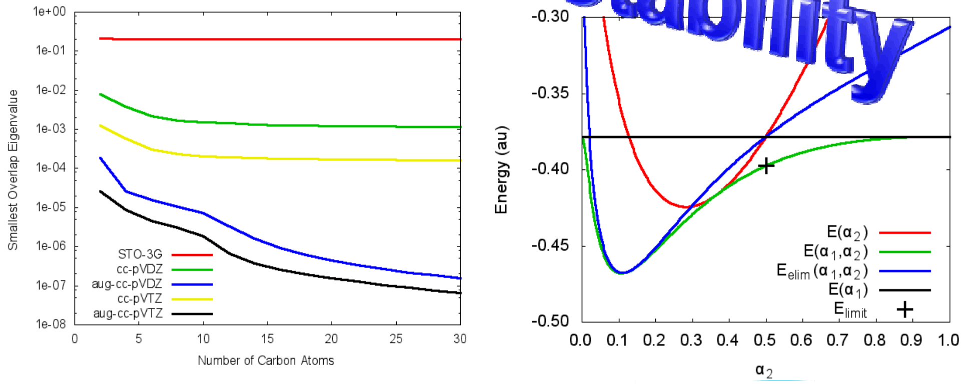
Power

Code Complexity



Generated using David A. Wheeler's 'SLOCCount'

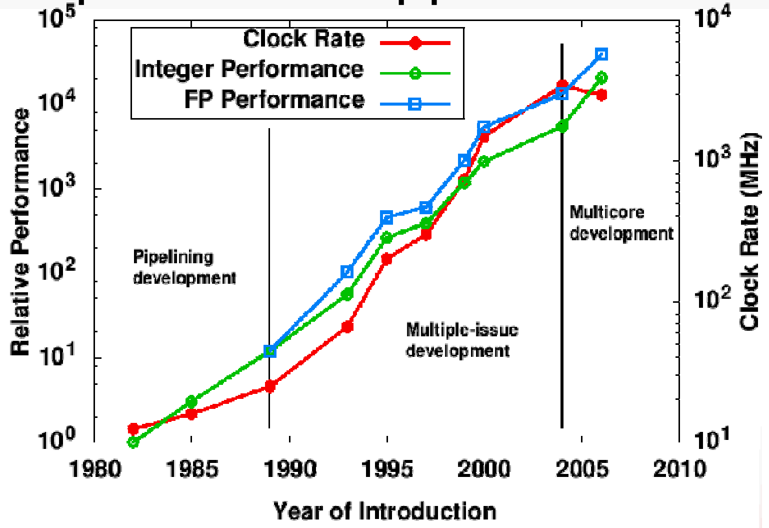
Numerical Stability



Motivation: complexity of parallel machines is accelerating, but tools to manage this are not

- Several complexity issues affect apps:
  - Extreme parallelism
  - More computation power enables more complex/higher fidelity simulations
    - More complex software
    - Numerical issues
  - Dropping mean time between failure
  - Energy enters optimization objective function

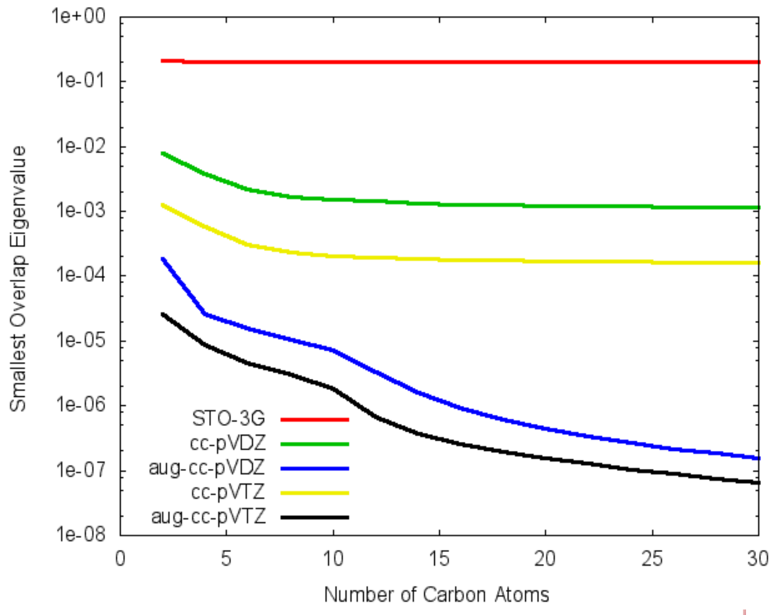
Improvements to chip performance over the



- Human effort does not scale easily to such a complex environment
  - Can another approach to programming solve some of these problems?
- Outline of current work:
  - Hartree-Fock theory selected due to its expense and scaling issues
    - Basis for many other electronic structure methods
  - Examine traditional implementation of Hartree-Fock theory
  - Show preliminary results of applying an alternative programming approach to Hartree-Fock and compare this to traditional implementations.

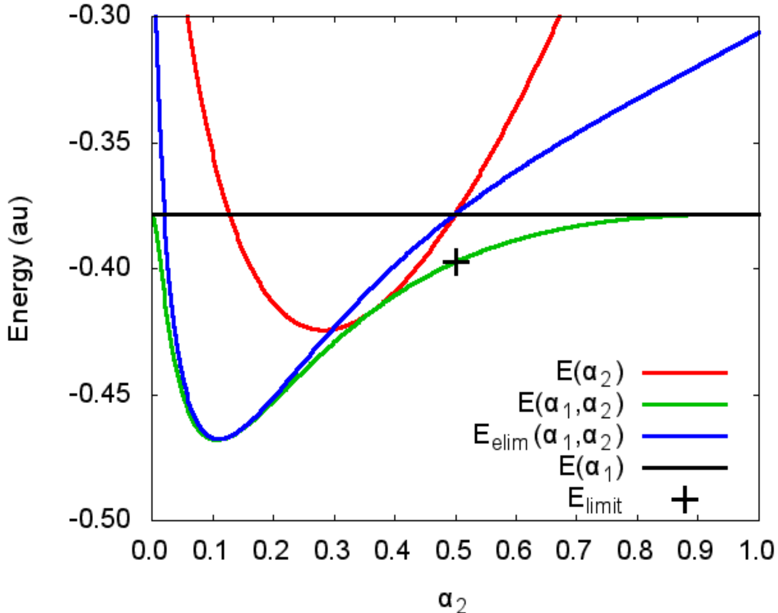
Illustration of numerical issues using Hartree-Fock theory as an example

Large systems are ill-conditioned: smallest overlap eigenvalue for linear alkane rapidly decrease as system grows for diffuse basis sets



Eliminating near linear dependencies can change energies—even in the limit of an exact linear dep.

Errors due to keeping the nearly linear dep. functions grow like s_1^{-3}, and we need the difference between large numbers:



Example application: Hartree-Fock theory

- Approximate solution to Schrödinger's equation

$$H = \frac{1}{2}\sum_{i}^{n}\nabla_i^2 - \sum_{i}^{n}\sum_{a}^{N}\frac{q_a}{r_{ia}} + \sum_{i<j}^{n}\frac{1}{r_{ij}} + \sum_{a<b}^{N}\frac{q_a q_b}{r_{ab}}$$

- Electron interact with average field of other electrons, giving rise to a generalized eigenvalue problem
- Major steps (assuming spin restricted closed shell):
  - Integral computation:

$$S_{pq} = \int \chi_p(\mathbf{r})\chi_q(\mathbf{r})\,d\mathbf{r} \qquad H_{pq} = \int \chi_p(\mathbf{r})\left(\nabla^2 - \sum_{a}^{N}\frac{q_a}{r_A}\right)\chi_q(\mathbf{r})\,d\mathbf{r}$$

$$G_{pqrs} = \int \chi_p(\mathbf{r}_1)\chi_q(\mathbf{r}_1)\frac{1}{r_{12}}\chi_r(\mathbf{r}_2)\chi_s(\mathbf{r}_2)\,d\mathbf{r}_1\,d\mathbf{r}_2$$

  - Fock matrix formation:

$$F_{pq} = H_{pq} + P_{rs}\left(v_{pqrs} + \frac{1}{2}v_{prqs}\right)$$

  - Diagonalization:

$$\mathbf{FC} = \mathbf{SCe} \qquad \mathbf{CSC}^T = \mathbf{1}$$

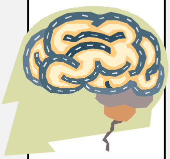
  - Density computation:

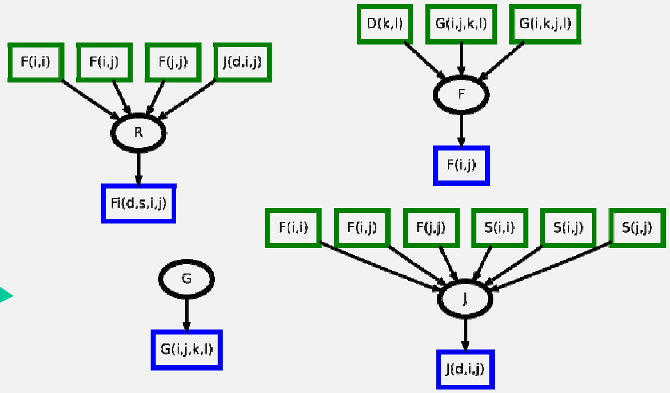
$$P_{pq} = 2\sum_{a}^{N/2} C_{pa}C_{qa}$$

Unteasing concurrency from applications

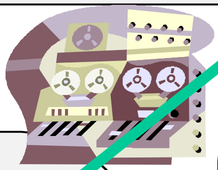
```
Form the atomic orbital Fock, F, and overlap, S
Synchronize so that F is complete on all nodes
Begin iterative eigensolver
  For each set of independent shell pairs
    Compute the rotation matrix
    Synchronize so rotation matrix is complete
    Rotate F and S
    Synchronize so that F and S are complete
  End loop over independent shell pairs
End eigensolver iterations
```

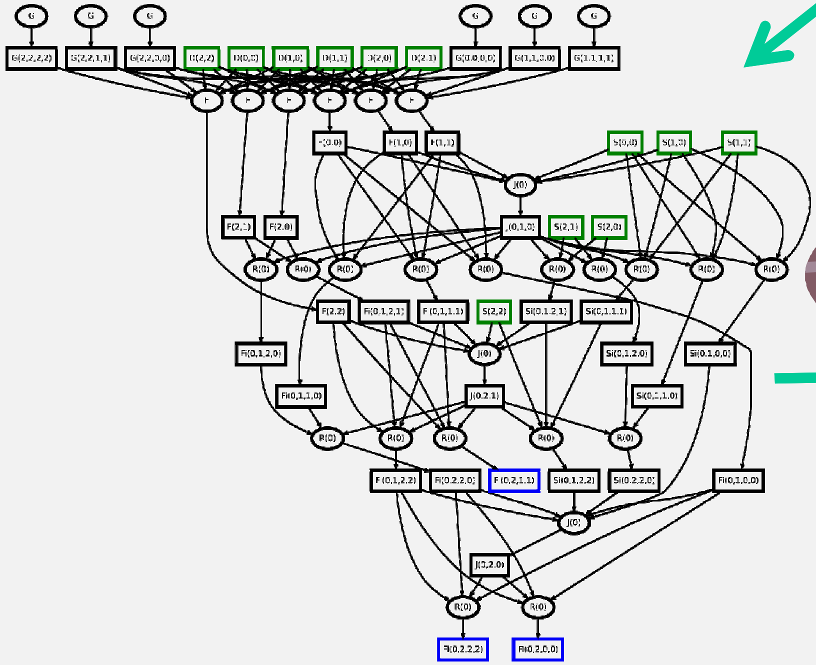
Traditional imperative formulation



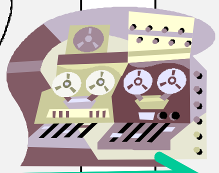


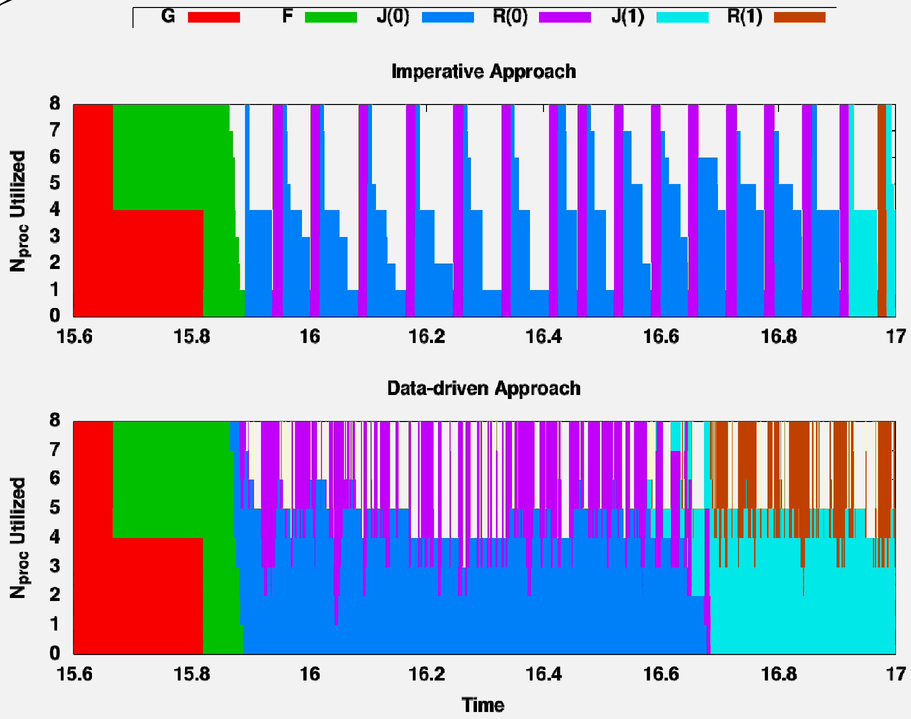
Elementary operations





Dataflow graph



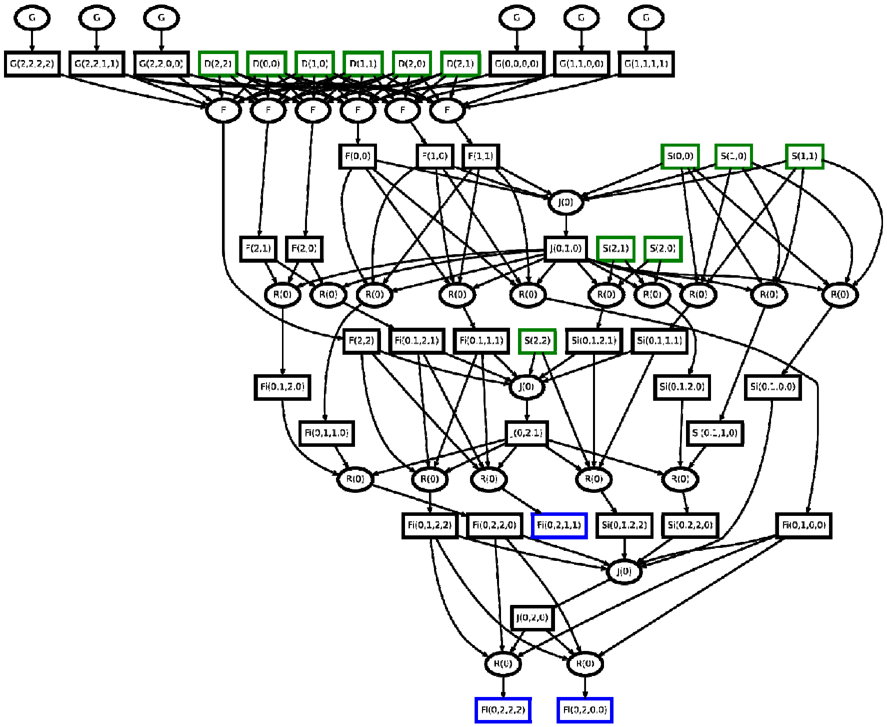


Simulated timings

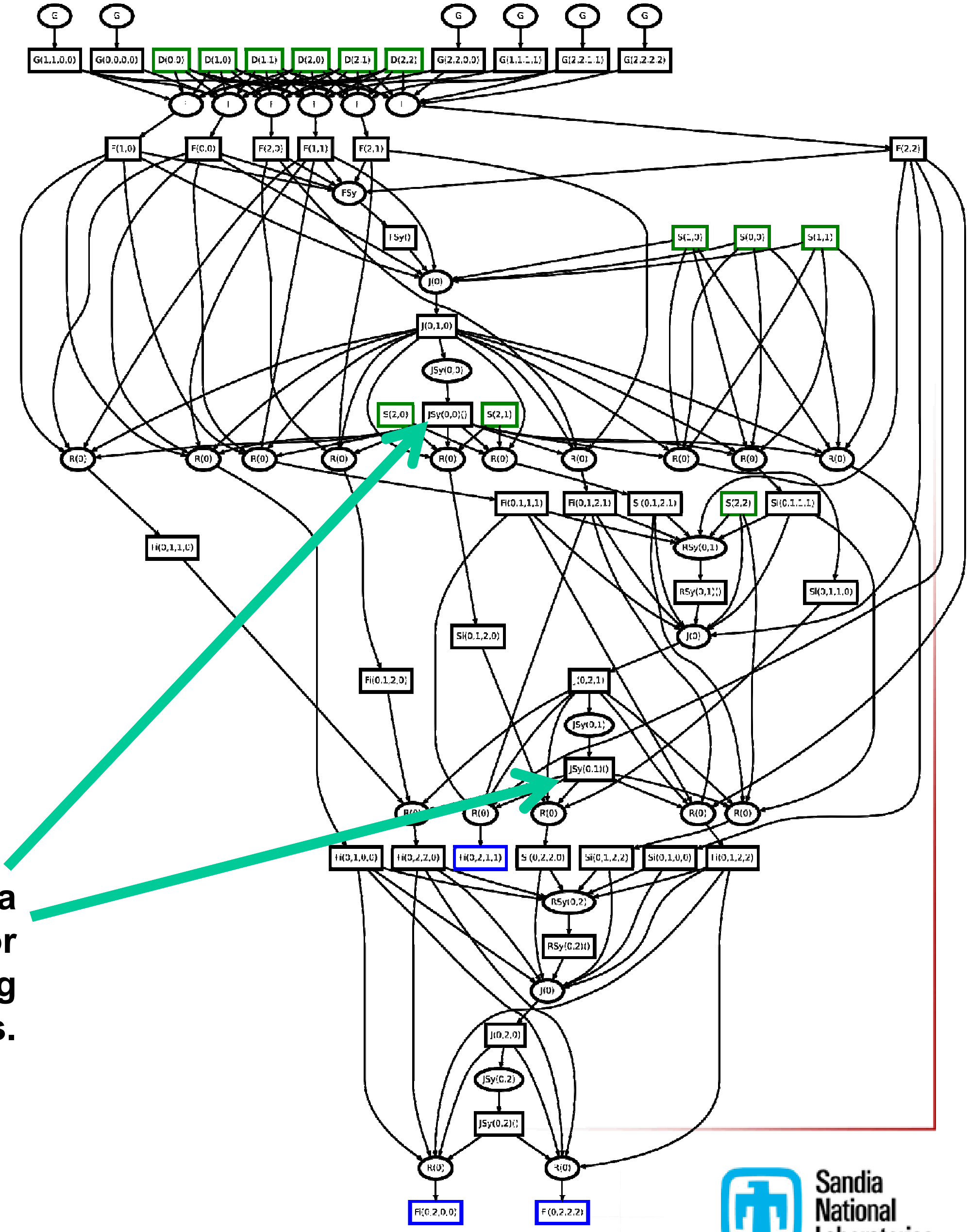
Sandia National Laboratories

Comparison of data dependencies with and without synchronization

Without synchronization:



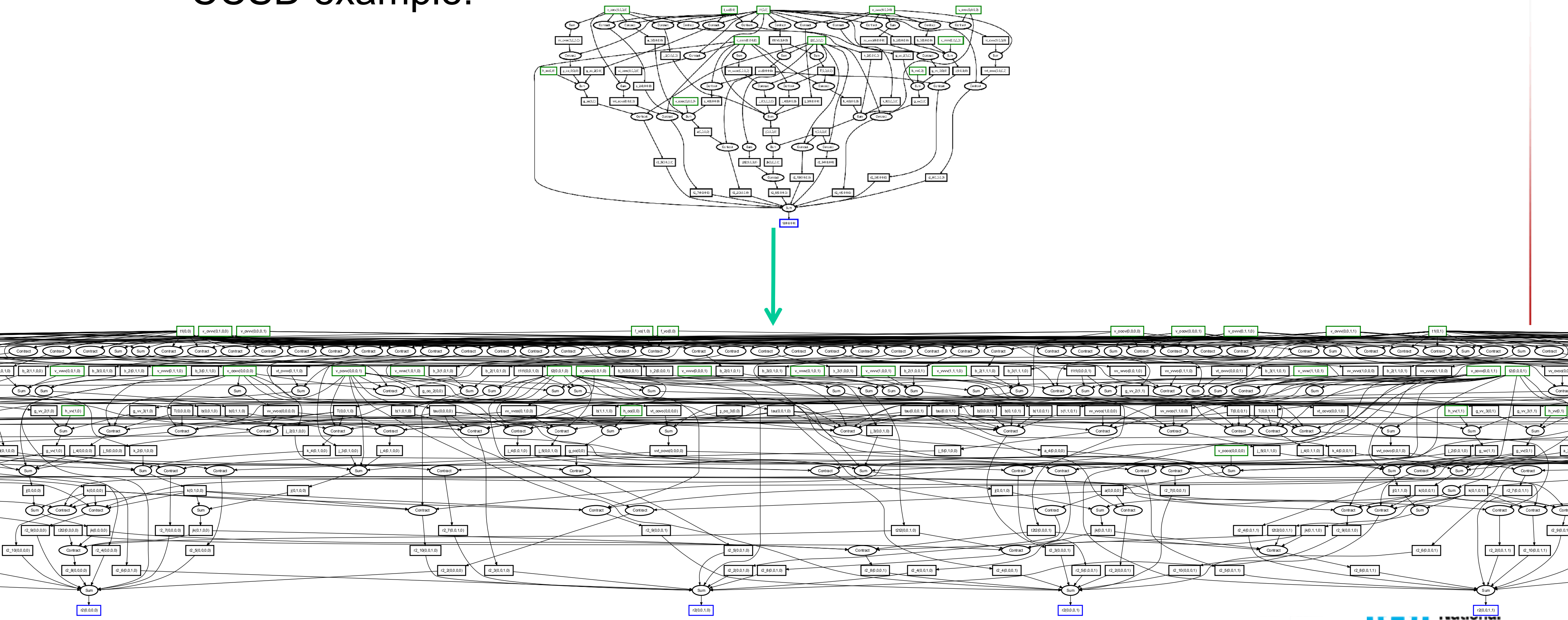
With synchronization:



Synchronization increases the number of data dependencies. Thus, the overall potential for parallelization is reduced by synchronizing operations such as barriers and collectives.

Hierarchical decomposition needed for locality and scalabilty

- Hierarchical in terms of operations
  - Eigenvectors constructed from Fock matrix constructed from integrals
- Hierarchical in terms of data
  - Large blocks containing small blocks, etc.
  - Map data hierarchy to memory hierarchy
  - CCSD example:



Be careful for what you ask …

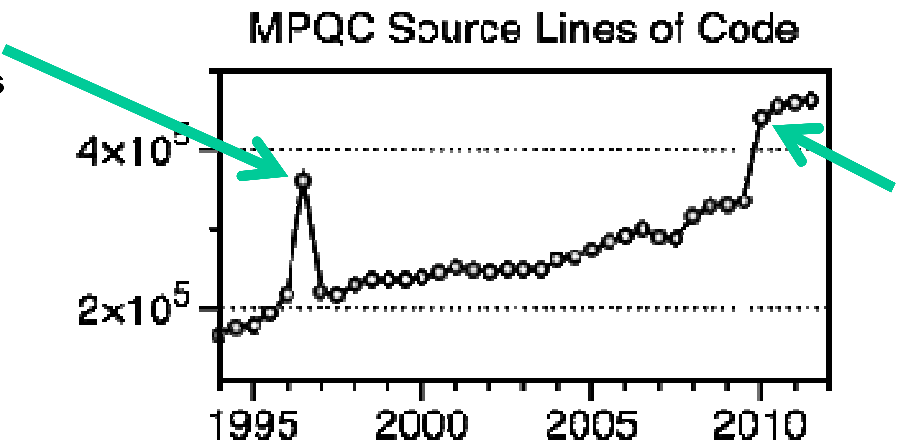
Am I asking for a monolithic runtime system?

- No – this is the problem with MPI. Need a lightweight, portable, and low-level interface for fast messaging. Includes active messages and fault notification primitives.
- Varying levels of sophistication can be built upon this low-lying interface.

Am I asking for new languages?

- Yes and no – general purpose languages spoken and developed by a wide community will always play a role. Libraries, DSLs (to generate the underlying code), and embedded DSLs (to supplement the underlying language) will be essential to hide machine complexity.

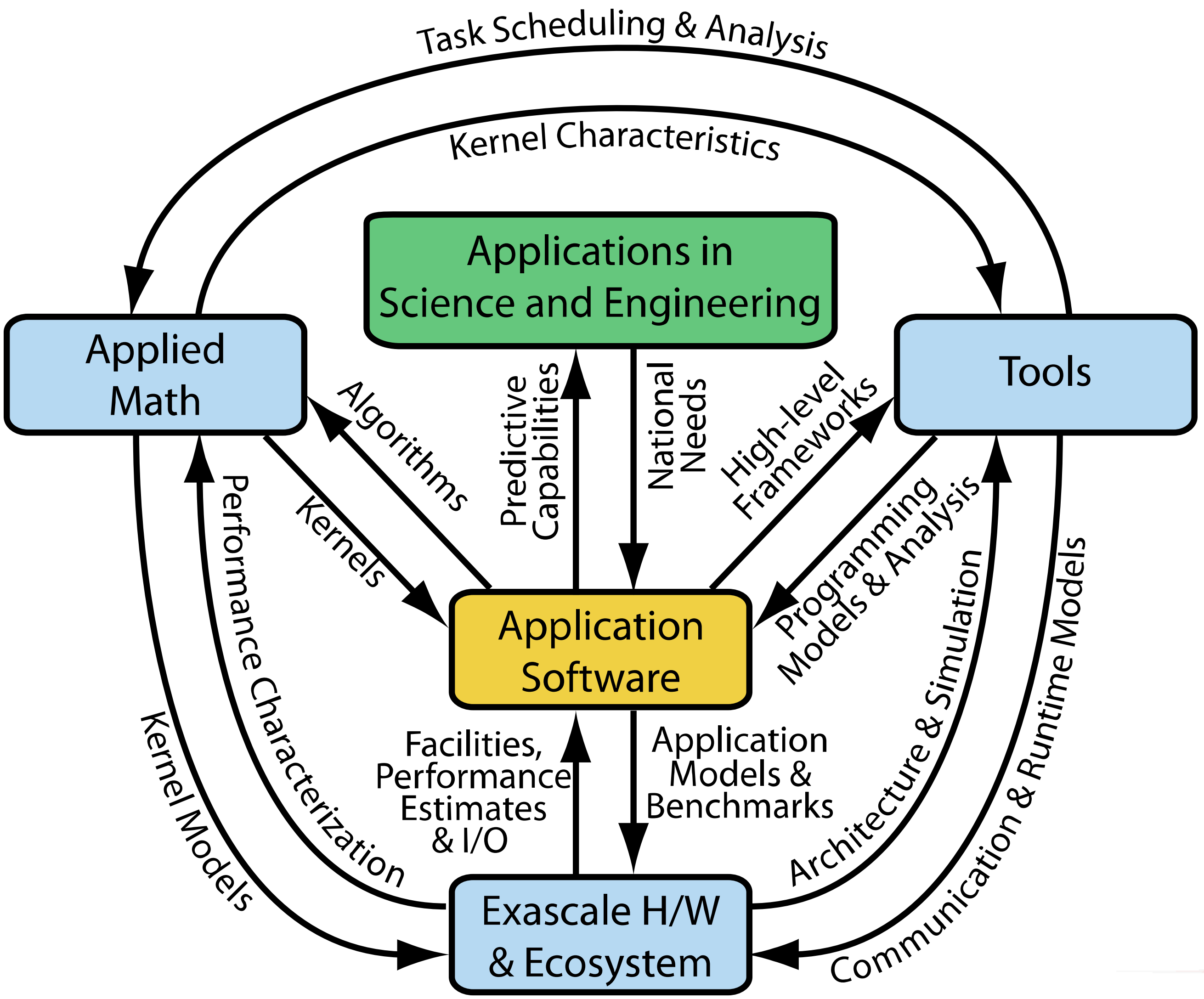
Introduction of DSL
for two electron integrals
(code too difficult for
compilers of the era was
subsequently removed)



Introduction of DSL
for many-body terms

Software isn't everything …

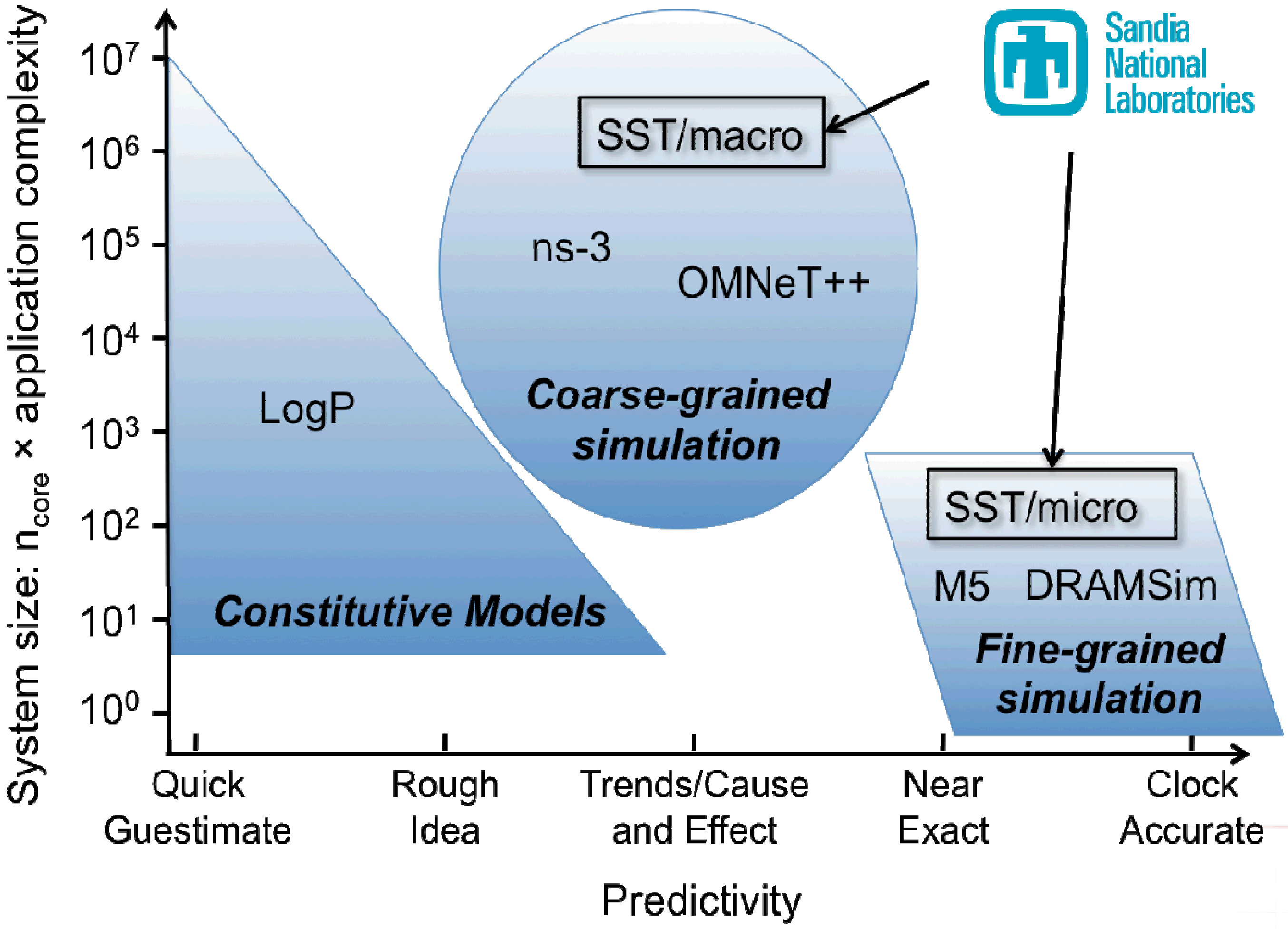
Drive hardware design in tandem with software: co-design


Task Scheduling & Analysis
Kernel Characteristics
Applications in Science and Engineering
Applied Math
Tools
Algorithms
Kernels
Predictive Capabilities
National Needs
High-level Frameworks
Programming Models & Analysis
Application Software
Performance Characterization
Kernel Models
Facilities, Performance Estimates & I/O
Application Models & Benchmarks
Architecture & Simulation
Communication & Runtime Models
Exascale H/W & Ecosystem

Sandia National Laboratories

Simulation permits study of future HPC systems

Structural Simulation Toolkit (SST) – create a multi-scale computer architecture for design and procurement of large-scale parallel machines as well as in the design of algorithms for these machines.

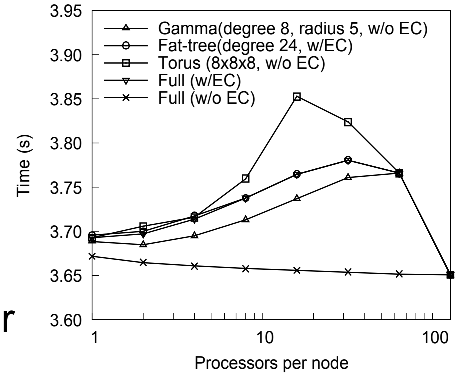


Sandia National Laboratories

Functionality, validation, and quality are key to SST/macro

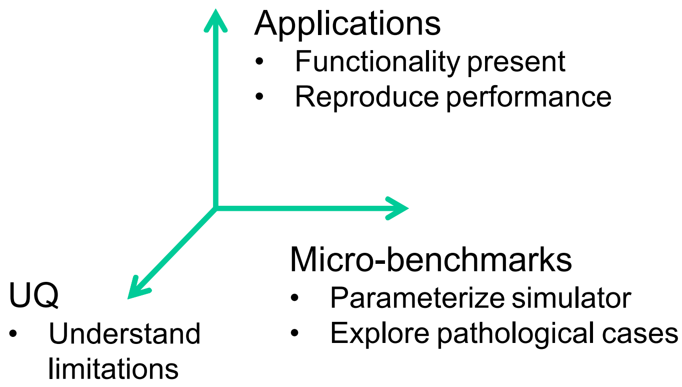
Functionality: Permit co-design

- Correctly identify causal relationships
  - Network topology
  - Node configuration
  - Noise/imbalance
  - Bandwidth
  - Latency
  - Resource contention



- Play "what if" games
  - Implementation effects for communication routines
  - Infinite performance in some components to stress others.
- Test changes to application, middleware, or resource management
  - Reordering code blocks, scheduling effects, etc.
- Test novel programming models
  - Fault-tolerant or fault-oblivious execution models
  - Alternatives to MPI, parallel runtime designs
  - Mixed programming models

Validation

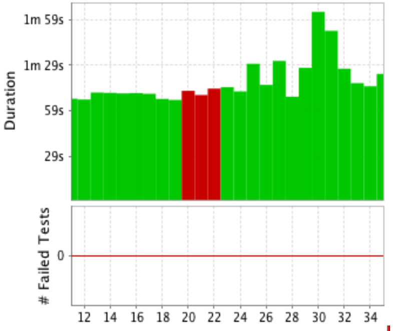


Applications
- Functionality present
- Reproduce performance

Micro-benchmarks
- Parameterize simulator
- Explore pathological cases

UQ
- Understand limitations

Software Quality Assurance

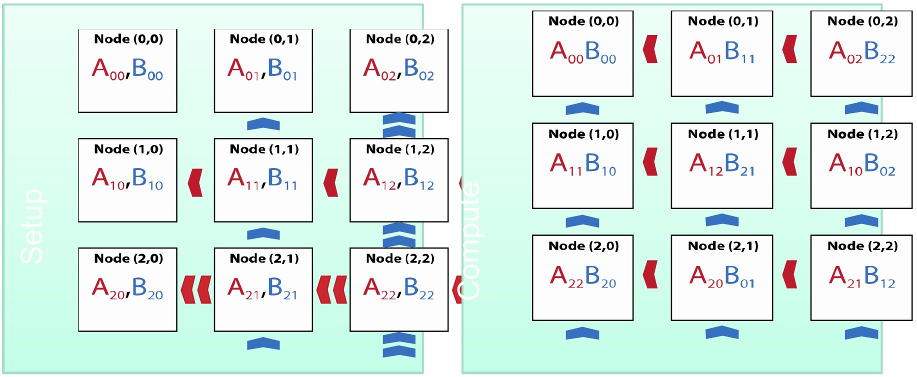


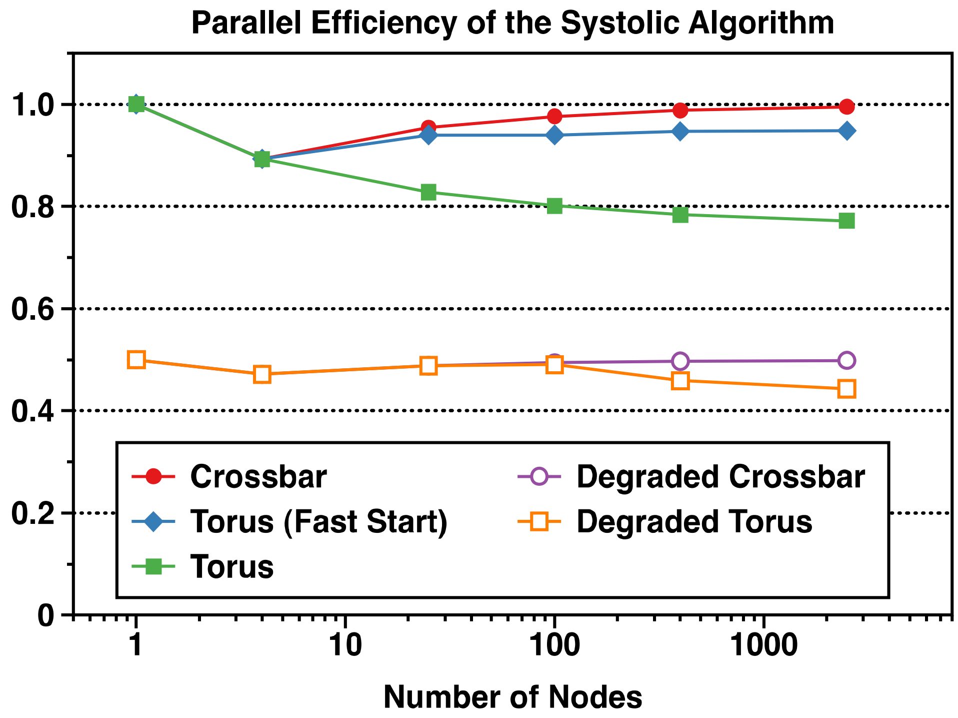
- Issue tracking / DVCS
- Continuous integration
- Automatic generation of
- documentation and
- distribution artifacts
- Software Development Plan
- Coordination with Sandia Software Quality Implementation Group (SQIG) on lab-wide Software Quality efforts

Multiple instruments are needed for co-design

Co-design Instrument	SST/macro role
Applications	• Use SST’s DUMPI to collect traces. Use SST/macro to simulate performance on different architectures and validate simulator. • Use to guide development of skeleton app, whether manually or automatically.
Compact Apps: small program capturing some aspect of full app. Generates a result.	
Mini Apps: small program capturing simplified aspect of full app. Perhaps no meaningful result.	
Skeleton Apps: captures control flow and communication pattern of app. Runs in simulator.	• SST/macro simulations for machines and algorithms not yet available.
Kernels: Capture node-level aspects of an algorithm.	• Generate parameterizations for coarse-grained models.

Programming model exploration: MPI application


Setup
Node (0,0) A_{00},B_{00}
Node (0,1) A_{01},B_{01}
Node (0,2) A_{02},B_{02}
Node (1,0) A_{10},B_{10}
Node (1,1) A_{11},B_{11}
Node (1,2) A_{12},B_{12}
Node (2,0) A_{20},B_{20}
Node (2,1) A_{21},B_{21}
Node (2,2) A_{22},B_{22}
Compute
Node (0,0) $A_{00}B_{00}$
Node (0,1) $A_{01}B_{11}$
Node (0,2) $A_{02}B_{22}$
Node (1,0) $A_{11}B_{10}$
Node (1,1) $A_{12}B_{21}$
Node (1,2) $A_{10}B_{02}$
Node (2,0) $A_{22}B_{20}$
Node (2,1) $A_{20}B_{01}$
Node (2,2) $A_{21}B_{12}$

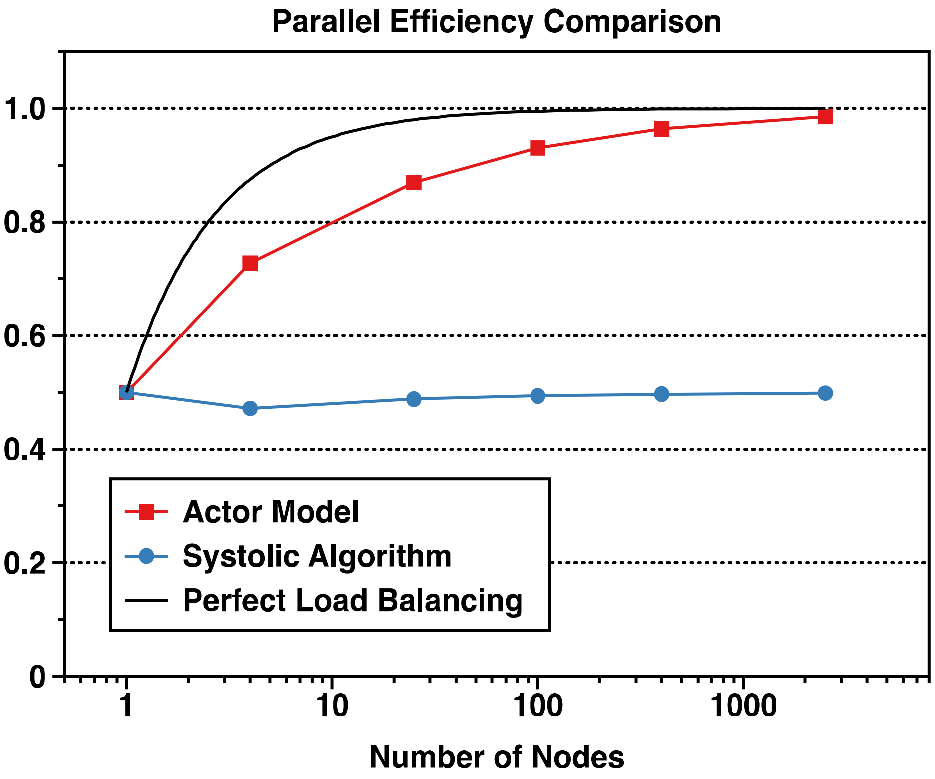


Skeleton Code Fragment

```
for (int i=0; i<nblocks-1; i++) {
  std::vector<sstmac::mpiapi::mpirequest_t> reqs;
  // Begin non-blocking left shift of A blocks
  sstmac::mpiapi::mpirequest_t req;
  mpi()->isend(blocksize, datatype, myleft,
              tag, world, req);
  reqs.push_back(req);
  mpi()->irecv(blocksize, datatype, myright,
              tag, world, req);
  reqs.push_back(req);
  // Likewise for B shifting down ...
  // Simulate computation with current blocks
  compute_api()->compute(instructions);
  mpi()->waitall(reqs, statuses);}
// Finish last block
compute_api()->compute(instructions);
```

- The implicitly synchronous systolic algorithm cannot recover from node degradation

Programming model exploration: actor model app.



Skeleton Code Fragments

```
// actormatmul run loop body
simplembox::recvresult_t reply =
mbox()->recv(actorid::any(),
          actorpattern::any());
actorid id = reply.first;
shared_ptr<base> msg
 = dynamic_pointer_cast<base>(reply.second);
msg->handle(id, self_.lock());

// actormatmul::compute run loop body
simplembox::recvresult_t res =
mbox()->recv(actorid::any(),actorpattern::any());
boost::shared_ptr<work> msg
 = boost::dynamic_pointer_cast<work>(res.second);
compute(msg);
mbox()->send(msg->store_to(),
          store::construct(msg->iteration()),
          msg->matdim()*msg->matdim());
```

- Simulation permits straightforward investigation of alternative programming models
- Work-stealing approaches will play a role in dealing with large-scale machines lacking perfect homogeneity

Summary

- Supercomputers in 10 years' time will provide fundamental challenges to both software and hardware designers
- Software and hardware must evolve together, in a co-design process, to meet this challenges
- Specialized tools, such as application surrogates and architecture simulators, are needed to implement this process

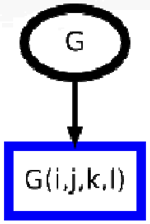
- More information:

http://sst.sandia.gov

cljanss@sandia.gov

Supplemental Slides

Elementary operations for Hartree-Fock in terms of data dependencies



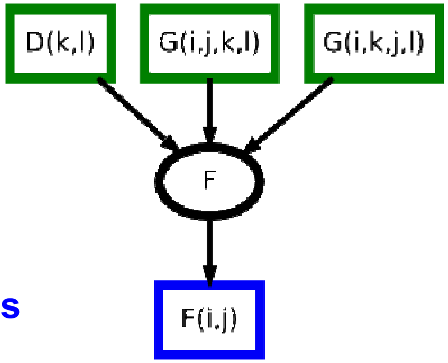
Two electron integrals formation, G:

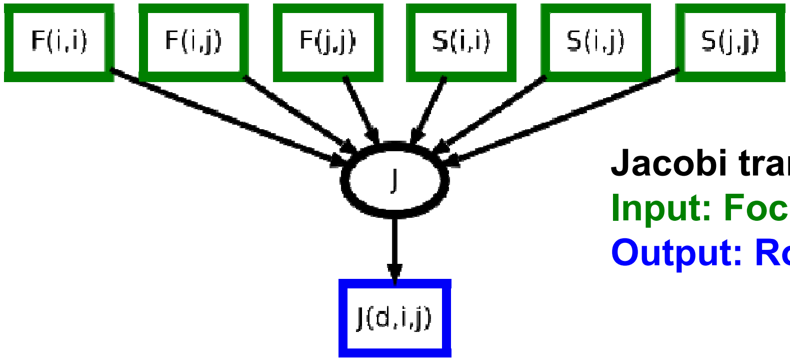
Output: (ij|kl) for a shell quartet

Fock matrix formation, F:

Input: Two electron integrals and density matrix

Output: Fock matrix elements for a shell pair





Jacobi transform formation, J:

Input: Fock and overlap matrix elements

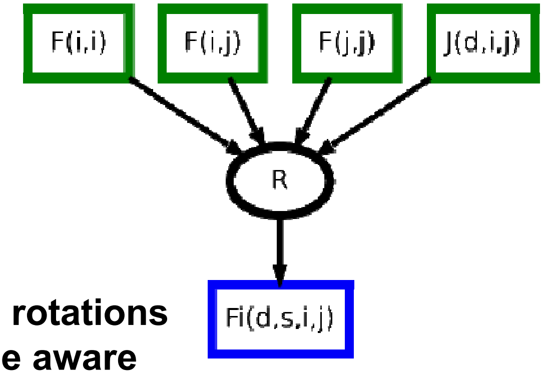
Output: Rotation matrix diagonalizing the sub-block

Matrix transformation, R:

Input: Fock or overlap matrix elements and Jacobi transform

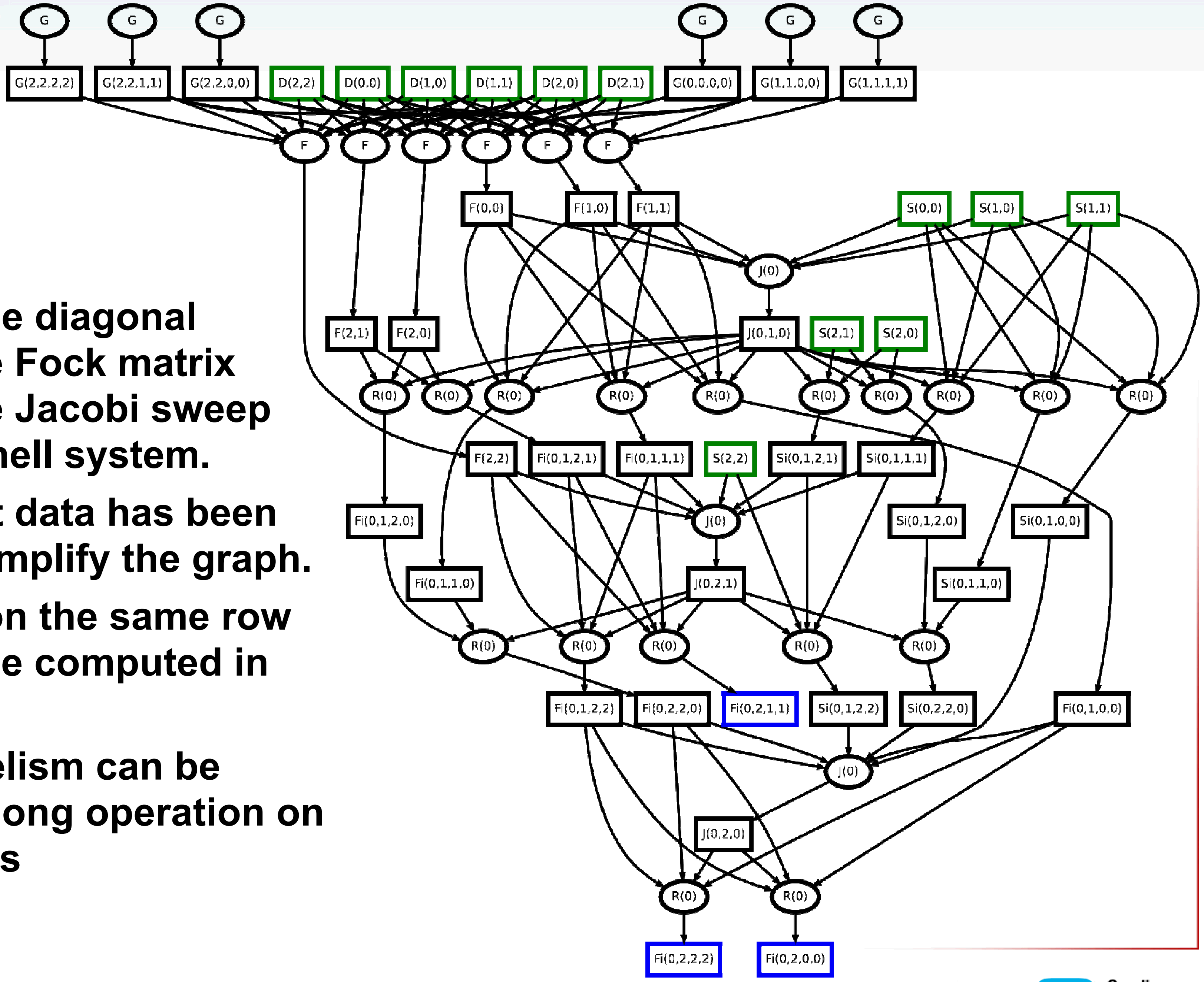
Output: Transformed matrix elements

Note: output has a sequence number that ensures rotations are done in the correct order. Both J and R must be aware of sequence number



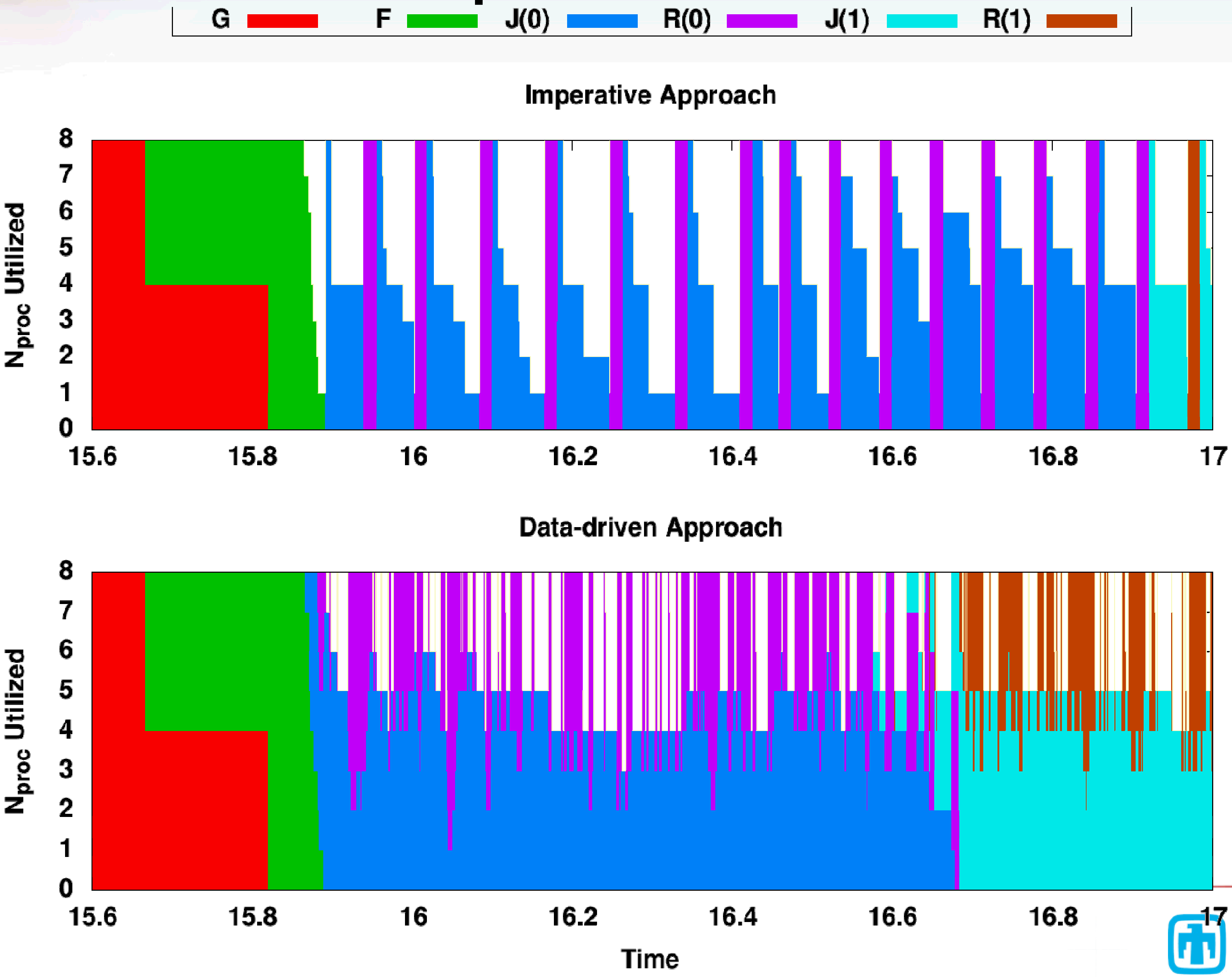
Hartree-Fock data dependencies

- Computes the diagonal blocks of the Fock matrix after a single Jacobi sweep for a three shell system.
- Certain input data has been omitted to simplify the graph.
- Operations on the same row (ovals) can be computed in parallel
- Some parallelism can be exploited among operation on different rows



Sandia National Laboratories

Simulated timings for 16 shells on 8 processors


G
F
J(0)
R(0)
J(1)
R(1)
Imperative Approach
N_{proc} Utilized
Data-driven Approach
N_{proc} Utilized
Time