SAND2013-0201P

Sandia National Laboratories

Exceptional service in the national interest

Fault tolerant programming models

Work by Janine Bennett, John Floren, Nicole Slattengren, Yevgeniy Vorobeychik

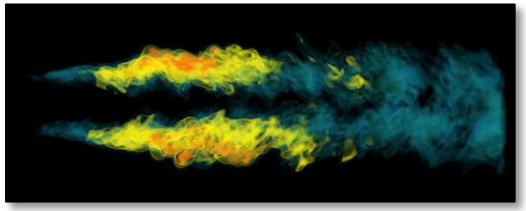
U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

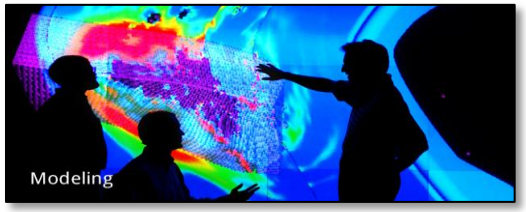
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Leadership-class HPC compute capabilities are required for DOE policy and decision making

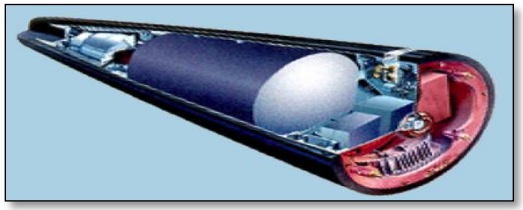
Sandia National Laboratories



Energy: Reduce U.S. reliance on foreign energy, reduce carbon footprint



Climate change: Understand, mitigate, and adapt to the effects of global warming



National Nuclear Security: Maintain a safe, secure, and reliable nuclear stockpile

Exascale computing and beyond is required to simulate complex phenomena that characterize the DOE mission space

Resilience is one of the many research challenges posed by the shift to exascale computing

Sandia National Laboratories

Exascale systems will experience errors/faults much more frequently than petascale systems*

Cause: There is a significant increase in the number of components with insufficient improvements in mean time to failure for each one.

Solution: True exascale resilience requires advancements in

- Fault detection, propagation, and understanding
- Fault recovery
- Fault-oblivious algorithms
- Stress testing of proposed fault-tolerance solutions

*Towards Exascale Resilience, Cappello et al., Intl. Journal of High Performance Computing Applications Nov 2009 vol. 23 no. 4 374-388

Our goal: Discover the right approach for extreme-scale, fault-resilient programming

Sandia National Laboratories

The community needs to understand:

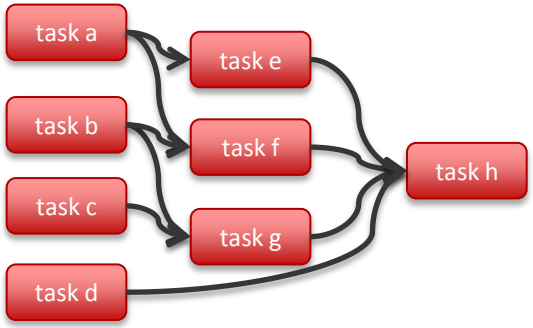
- Can MPI+X offer high scalability at exascale even in the face of faults?
- If not, which programming models can reach which scales?
- If no programming model can reach scales of interest for a given application without algorithmic changes, how might algorithms be adapted?
- What are co-design implications for tradeoffs between memory, I/O, power, resilience, application performance, and development effort?

Existing programming models are inherently not fault-resilient	Asynchronous many-task (AMT) programming models can be fault-resilient
Single Program Multiple Data (SPMD), implicitly synchronous algorithms cannot recover from failure nor adapt well to node degradation	Asynchronous execution and redundancy minimize the impact of node degradation/failure and benefit scalability even without failure
Global check-points no longer feasible	Synergistic with local check-pointing

To achieve our goal, we must explore challenges impeding the use of AMT programming models

Sandia National Laboratories

- How does one intuitively express tasks to achieve asynchronous execution?
  - Task-Directed Acyclic Graphs (task-DAGs) can depict data dependencies and flow; however may not be enumerable until run-time



- What is the best approach for resilient decentralized scheduling of tasks?
  - How can missing task subgraphs be regenerated after failure using asynchronous local check-points?
  - Can our approach to resilience be leveraged to improve scalability and avoid additive cascading delays?

Challenges will be explored via tests on hardware simulators as well as tests on current system architectures

Deliverables for FY13

Sandia National Laboratories

- Creation of initial metrics for development effort and performance profile analysis for non-MPI programming models
- Validation of simulator performance predictions with realistic application workloads using Cielo and/or other current ASC platforms
- Exploration of the programming model design space to include
  1. Applications that are difficult to load-balance
  2. New failure response strategies
  3. Models of failure histories
  4. Task-DAG scheduling frameworks

Specific FY13 efforts

Sandia National Laboratories

- Understand past and current PM and resilience efforts
- Develop metrics for fair comparisons between MPI and non-MPI programming models both with and without faults
- Validation of simulator performance predictions of the cellular automaton on Cielo
- Develop a shared-memory task-DAG API/runtime and add in-memory redundancy and local check-pointing capabilities
- Port a simple conjugate gradient mini-app to our model
- Start extending the task-DAG API/runtime to work in a distributed-memory environment
- Concurrently, starting porting more realistic mini-apps, starting with mini-FE

Sandia National Laboratories

FY 13 RESULTS

Specific FY13 efforts

Sandia National Laboratories

- Understand past and current PM and resilience efforts
- Develop metrics for fair comparisons between MPI and non-MPI programming models both with and without faults
- Validation of simulator performance predictions of the cellular automaton on Cielo
- Develop a shared-memory task-DAG API/runtime and add in-memory redundancy and local check-pointing capabilities
- Port a simple conjugate gradient mini-app to our model
- Start extending the task-DAG API/runtime to work in a distributed-memory environment
- Concurrently, starting porting more realistic mini-apps, starting with mini-FE

Some alternative programming models efforts

Sandia National Laboratories

- DAGuE/TBB:
  - C++ libraries that include Task-DAG scheduling
  - Key limitations: resilience not directly supported; explicit dependencies are only among tasks
- Chapel/X10/Fortress:
  - New parallel programming languages
  - Key limitation: no direct path forward from currently implemented codes
- UPC:
  - Extension of C rather than new programming language
  - Key limitations: SPMD; number of threads fixed at start time
- Charm++:
  - Very generic; no direct support for Task-DAG model or resilience
- ParalleX/HPX:
  - Message-driven work-queue model using global address space
  - Scheduler and AGAS server are single points of failure

Specific FY13 efforts

Sandia National Laboratories

- Understand past and current PM and resilience efforts
- Develop metrics for fair comparisons between MPI and non-MPI programming models both with and without faults
- Validation of simulator performance predictions of the cellular automaton on Cielo
- Develop a shared-memory task-DAG API/runtime and add in-memory redundancy and local check-pointing capabilities
- Port a simple conjugate gradient mini-app to our model
- Start extending the task-DAG API/runtime to work in a distributed-memory environment
- Concurrently, starting porting more realistic mini-apps, starting with mini-FE

Performance metrics for comparisons between MPI and non-MPI programming models

Sandia National Laboratories

- For different classes of applications (e.g., CPU-bound, memory-bound, I/O-bound), measure scalability and performance in terms of:
  - Time to completion (including checkpoint/restart)
  - Progress made (e.g. iterations completed) in a period of time
  - Processor utilization
  - Communication cost
- Perform comparisons both in the absence of failures and under different rates of failure
- Use failure models where failed nodes either leave the computation permanently (fail-stop) or rejoin after some delay (fail-delay)

Specific FY13 efforts

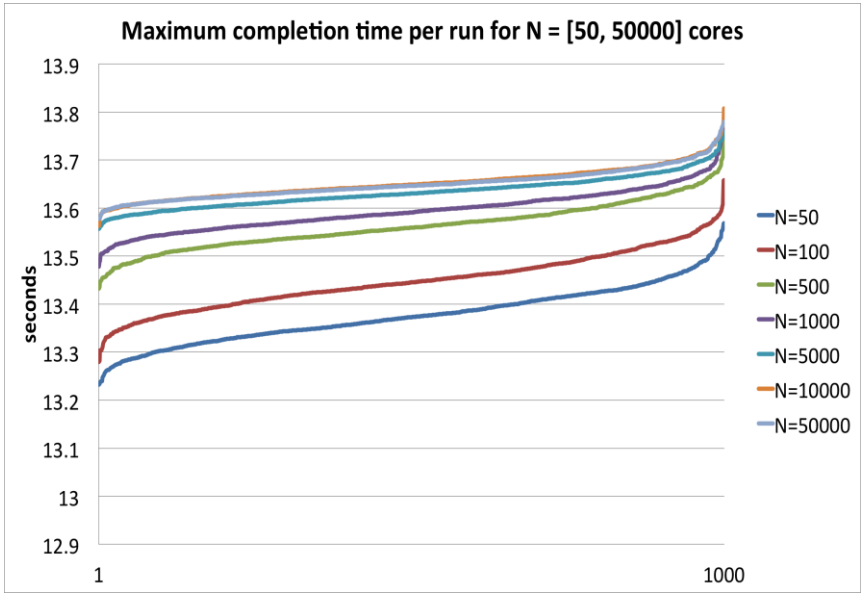
Sandia National Laboratories

- Understand past and current PM and resilience efforts
- Develop metrics for fair comparisons between MPI and non-MPI programming models both with and without faults
- Validation of simulator performance predictions of the cellular automaton on Cielo
- Develop a shared-memory task-DAG API/runtime and add in-memory redundancy and local check-pointing capabilities
- Port a simple conjugate gradient mini-app to our model
- Start extending the task-DAG API/runtime to work in a distributed-memory environment
- Concurrently, starting porting more realistic mini-apps, starting with mini-FE

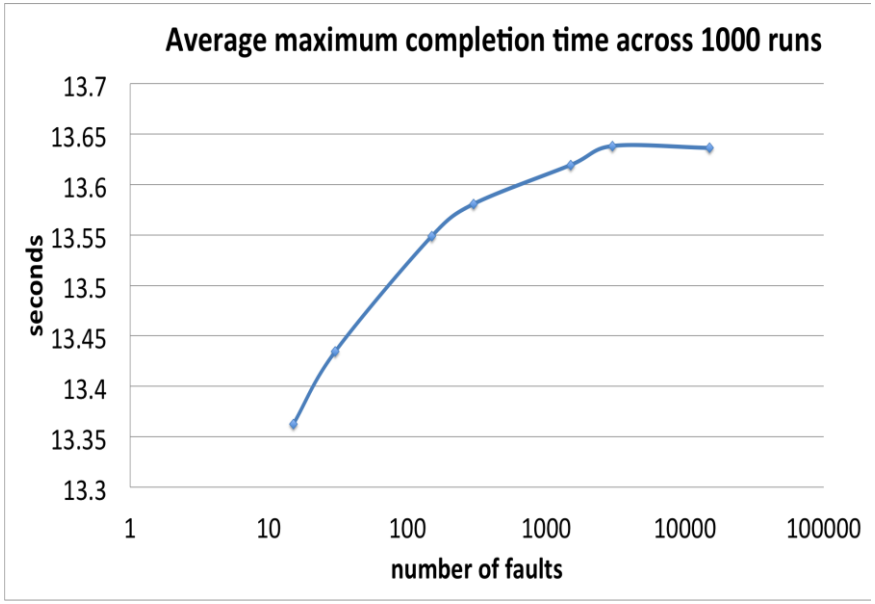
Validation of simulator performance predictions for the cellular automaton on Cielo

Sandia National Laboratories

- Task-driven cellular automaton code scales up to 50,000 cores
- In the process of performing analogous scaling studies using SST Macro to validate performance predictions



Run number (sorted by maximum completion time)



Poisson fault model: # of faults increases with number of cores

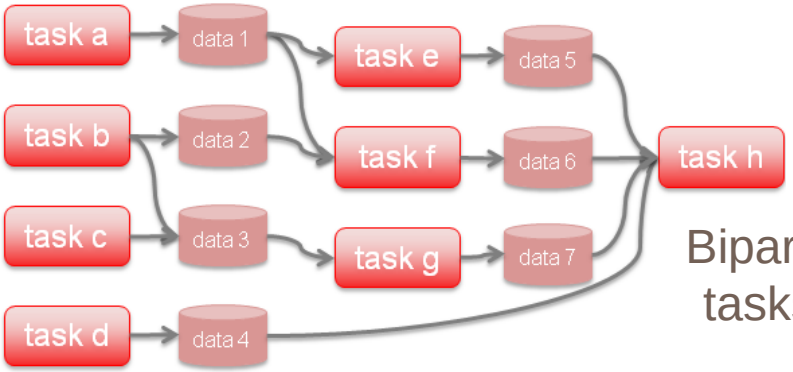
Specific FY13 efforts

Sandia National Laboratories

- Understand past and current PM and resilience efforts
- Develop metrics for fair comparisons between MPI and non-MPI programming models both with and without faults
- Validation of simulator performance predictions of the cellular automaton on Cielo
- Develop a shared-memory task-DAG API/runtime and add in-memory redundancy and local check-pointing capabilities
- Port a simple conjugate gradient mini-app to our model
- Start extending the task-DAG API/runtime to work in a distributed-memory environment
- Concurrently, starting porting more realistic mini-apps, starting with mini-FE

Our approach: tasks depend only on data; dependencies among tasks are implicit

Sandia National Laboratories



Bipartite DAG of tasks and data

- Explicit dependencies of tasks on data allows automation of data replication, check-pointing, and other FT mechanisms
  - Leverage resilience work on local check-pointing (ASC)
  - Differs from approaches for DAGuE, TBB
- Dynamic scheduling may allow us to make more intelligent scheduling decisions based on the time needed to retrieve a dependency from a remote node
  - Leverage data stores from FOX (DOE ASCR X-Stack) and/or Nessie

Our approach: tasks depend only on data

Sandia National Laboratories

- Transaction-like semantics of tasks allow them to safely be replayed in the event of failure
  - Tasks can modify state *only* by producing defined, write-once results, unlike in ParalleX/HPX
  - Leveraging the resilience API from Bob Lucas's group would allow us to respond to failures instead of abort
- Fully dynamic scheduling may allow us to adapt to dynamically-changing resources as nodes drop out due to failure or are added when no longer needed by other jobs
  - Resources not fixed at execution time like in UPC
- The task-DAG API/runtime can be provided as a library, potentially allowing it to integrate with legacy codes
  - Not an entirely new language like Chapel, X10, Fortress
  - We can evaluate our approach at exascale both with and without faults using SST/macro simulation

Specific FY13 efforts

Sandia National Laboratories

- Understand past and current PM and resilience efforts
- Develop metrics for fair comparisons between MPI and non-MPI programming models both with and without faults
- Validation of simulator performance predictions of the cellular automaton on Cielo
- Develop a shared-memory task-DAG API/runtime and add in-memory redundancy and local check-pointing capabilities
- Port a simple conjugate gradient mini-app to our model
- Start extending the task-DAG API/runtime to work in a distributed-memory environment
- Concurrently, starting porting more realistic mini-apps, starting with mini-FE