CCMT

Behavioral Emulation for Scalable Design-Space Exploration of Algorithms and Architectures

Nalini Kumar *(PhD Candidate),*

Carlo Pascoe, Chris Hajas, Herman Lam, Greg Stitt, and Alan George

PSAAP II Center for Compressible Multiphase Turbulence (CCMT)
NSF Center for High-Performance Reconfigurable Computing (CHREC)
ECE Department, University of Florida, Gainesville FL, USA

E-MuCoCoS 2016 (Co-located with ISC), Frankfurt, Germany

Outline

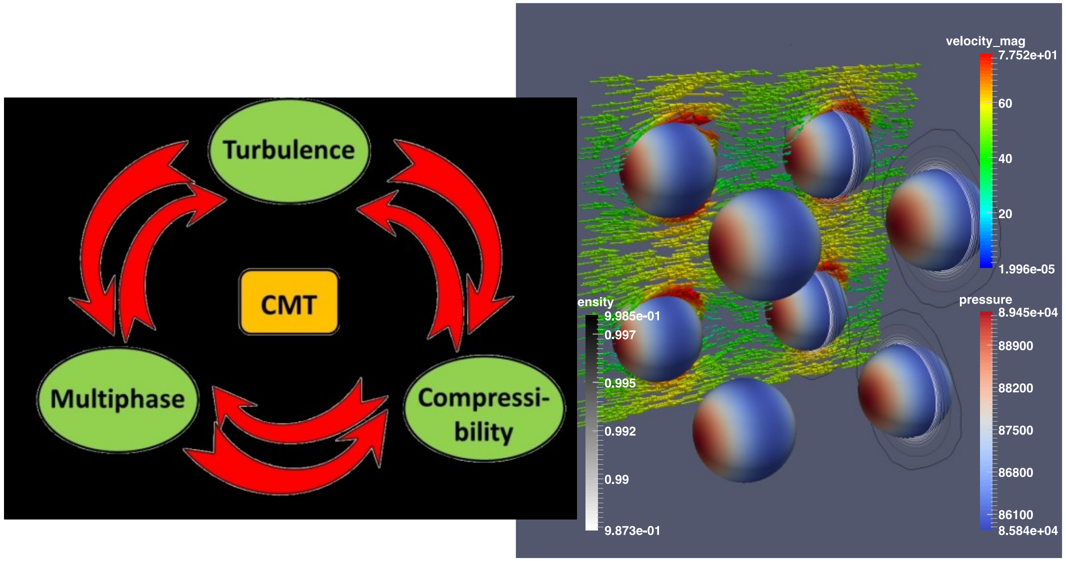
- The Big Picture – Modeling and Simulation for Co-design
- Our M&S approach – Behavioral Emulation
  - Overview and Workflow of Behavioral Emulation
- Modeling
  - What are we modeling? What are the independent parameters?
  - Building the models and model representations!
  - Measurements (what does our data look like?)
- Simulation
  - Step 1: Combining the models together
  - Step 2: Validation (not leave one out!) of individual block models
- Prediction: Finally what we wanted all along!
  - Design Space Exploration
  - Probabilistic simulations
- Conclusions & Future Directions

Outline

- The Big Picture – Modeling and Simulation for Co-design
- Our M&S approach – Behavioral Emulation
  - Overview and Workflow of Behavioral Emulation
- Modeling
  - What are we modeling? What are the independent parameters?
  - Building the models and model representations!
  - Measurements (what does our data look like?)
- Simulation
  - Combining the models together
  - Validation
- Prediction: Finally what we wanted all along!
  - Design Space Exploration
  - Probabilistic simulations
- Conclusions & Future Directions

The Big Picture

- CCMT Center Goals:
  - To radically advance the field of Compressible Multiphase Turbulence (CMT)
  - To advance predictive simulation science on current and near-future computing platforms with uncertainty budget as backbone
  - To advance a co-design strategy that combines exascale emulation, exascale algorithms, exascale CS



CMT-nek simulations

Our Co-design Problem

- Our challenge is to develop a scalable high-performance software
  - What are the most likely productive execution models?
  - What is the measurable benefit of switching from MPI-only to MPI+X?
  - Will it be considerable effort to optimize key kernels for each platform?
  - How can we better decompose the app to maximize the benefit from next-gen architectures and technologies (especially memories)?

- Also, pareto-optimization for high performance and low energy
  - We don't have the devices for experimentation

cycles of

- Need ^ simulation and emulation to help analyze different design tradeoffs – algorithm and architecture design space exploration (DSE)

Motivation: Large CMT-nek Design Space

Parametric Options – ***minimal changes to inputs & BE methods***

- h-refinement vs p-refinement of CMT-nek
- Number of computational particles per cell
- Order of accuracy of Euler-Lagrange interpolation/back-coupling

Algorithmic Options – ***require building models for new algorithms***

- Shock capturing methodology (hyperviscosity vs p-refinement)
- Euler-to-Lagrange interpolation algorithm (accuracy vs efficiency)
- Lagrange-to-Euler back-coupling algorithm
- Crystal router vs other data-communication for computational particles
- Immersed boundary vs immersed interface vs ghost fluid

Architectural Options – ***require models for each algorithm/arch. pair***

- GPU-CPU implementation of Lagrangian particles
- GPU-CPU workload partition

Other Design Space Options

- Domain partitioning (pencil vs sheets vs blocks)
- Focusing computational power to where needed

Developed in collaboration with CMT-nek development team

Our M&S Approach – Behavioral Emulation

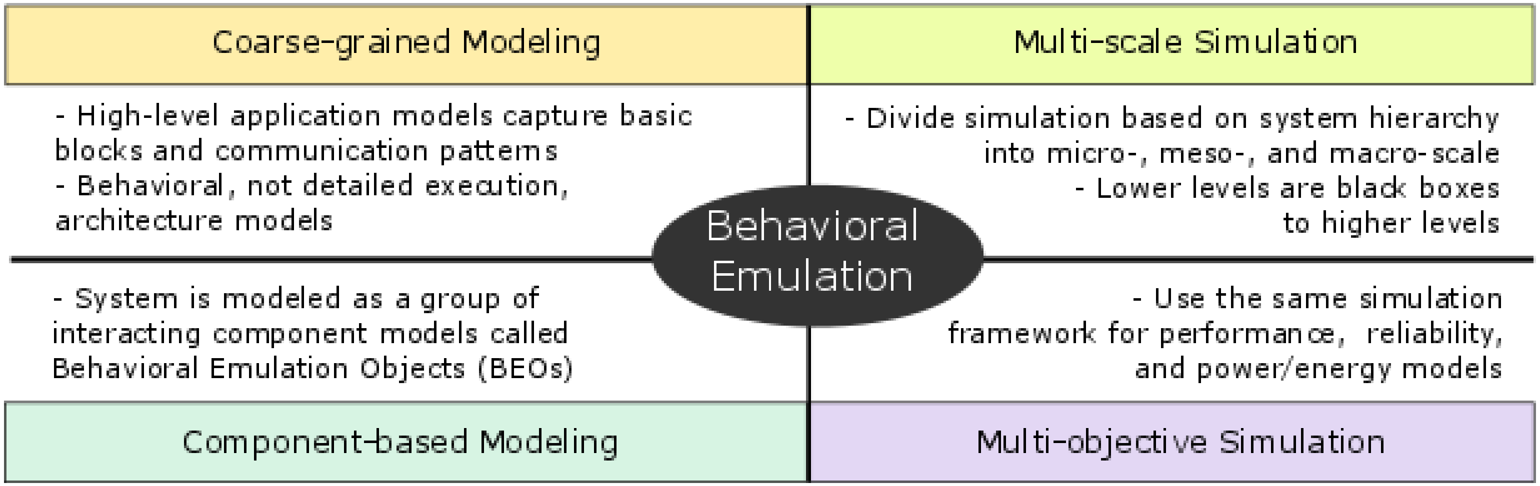
- How may we study Exascale before the age of Exascale?
  - Analytical studies – systems are too complicated
  - Software simulation – simulations are too slow at scale
  - Functional emulation – systems too massive and complex
  - Prototype device – future technology, does not exist
  - Prototype system – future technology, does not exist
- Many pros and cons with various methods
  - We believe behavioral emulation is most promising in terms of balance of DSE goals (accuracy, speed, and scalability, as well as versatility)

- Scope and contribution of this paper:
  - Develop methods and confidence in BE
    - Prototype and validate BEO models and simulation framework which is essential before optimizing framework for speed and scale
  - Gain insight into abstraction and representation of application behavior
  - Demonstrate the use of BE for early design space exploration

Outline

- The Big Picture – Modeling and Simulation for Co-design
- Our M&S approach – Behavioral Emulation
  - Overview and Workflow of Behavioral Emulation
- Modeling
  - What are we modeling? What are the independent parameters?
  - Building the models and model representations!
  - Measurements (what does our data look like?)
- Simulation
  - Combining the models together
  - Validation
- Prediction: Finally what we wanted all along!
  - Design Space Exploration
  - Probabilistic simulations
- Conclusions & Future Directions

Key Features of Behavioral Emulation (BE)



Coarse-grained Modeling

- High-level application models capture basic blocks and communication patterns
- Behavioral, not detailed execution, architecture models

Multi-scale Simulation

- Divide simulation based on system hierarchy into micro-, meso-, and macro-scale
- Lower levels are black boxes to higher levels

Behavioral Emulation

Component-based Modeling

- System is modeled as a group of interacting component models called Behavioral Emulation Objects (BEOs)

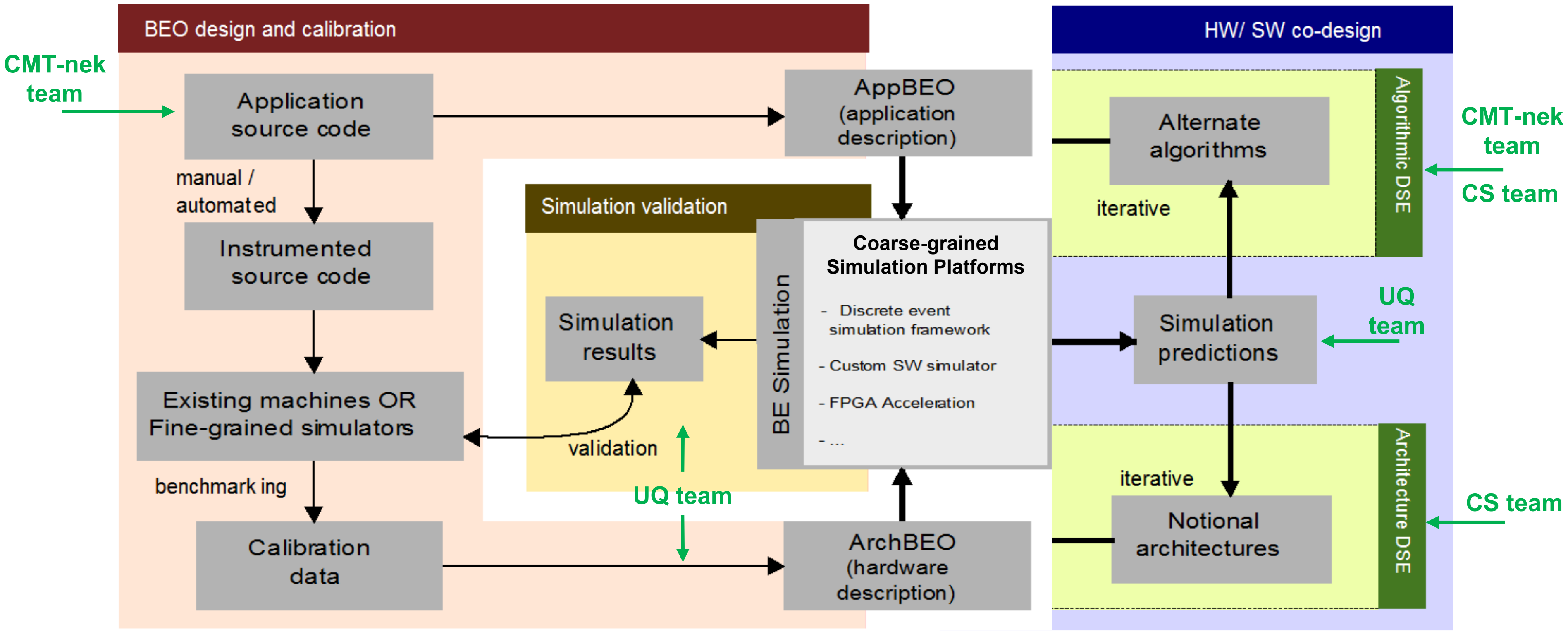
Multi-objective Simulation

- Use the same simulation framework for performance, reliability, and power/energy models

- Component-based simulation
  - Fundamental constructs called BE Objects (BEOs) act as surrogates
  - BEOs characterize & represent behavior of app, device, node, & system objects as fabrics of interconnected ArchBEOs (with AppBEOs)
- Multi-scale simulation
  - Hierarchical method based upon experimentation, abstraction, exploration
- Multi-objective simulation
  - Performance, power, reliability, and other environmental factors
  - Our challenge is to develop a scalable high-performance software

N. Kumar, A. George, H. Lam, G. Stitt, S. Hammond, "Understanding Performance and Reliability Trade-offs for Extreme-scale Systems using Behavioral Emulation", Workshop on Modeling & Simulation of Systems and Applications (ModSim 2015)

Co-Design Using *Behavioral Emulation*



* BEO – Behavioral Emulation Object

Outline

- The Big Picture – Modeling and Simulation for Co-design
- Our M&S approach – Behavioral Emulation
  - Overview and Workflow of Behavioral Emulation
- Modeling
  - What are we modeling? What are the independent parameters?
  - Building the models and model representations!
  - Measurements (what does our data look like?)
- Simulation
  - Combining the models together
  - Validation
- Prediction: Finally what we wanted all along!
  - Design Space Exploration
  - Probabilistic simulations
- Conclusions & Future Directions

Application Models: AppBEOs

- Representation of applications that simulator can understand
  - AppBEOs are list of instructions processed by ProcBEOs
  - Small and simple description allows easy development
    - Developer does not need to worry about creating working application code
  - Intermediate format is compiled separately for each simulation platform

AppBEO (high-level description)

```
// Define group as nodes 0-3
VAR commGrp=0:3
// Broadcast matrix A
(dataSize=64*64/2) to group
Bcast(int32,2048,0,commGrp)
// Barrier sync
Barrier(commGrp)
// Scatter 1/4 of matrix B
(dataSize=(64*64)/(4*2)) to each node
Scatter(int32,512,0,commGrp)
// Perform dot product of vector size 64
of int32
DotProduct(int32,64)
// Gather solutions from matrices
(dataSize=(64*64)/(4*2))
Gather(int32,512,commGrp)
Done
```

Intermediate format

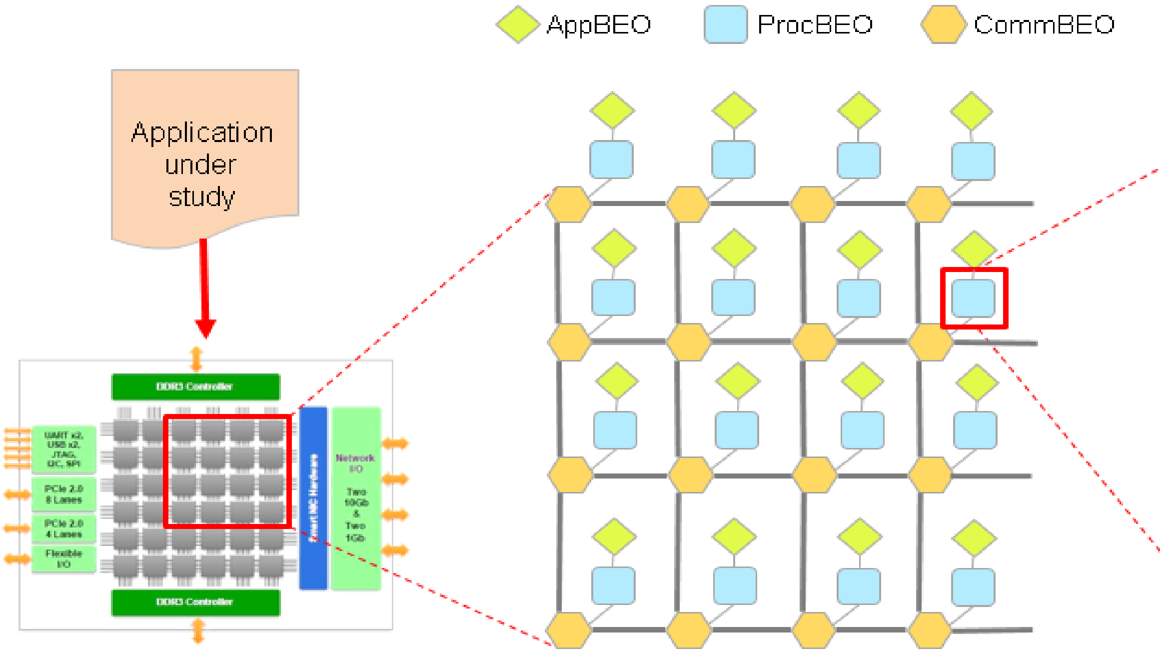
```
send 1 1 129971 1
recv 4
send 2 2 129971 1
recv 8
send 13 1 381 1
recv 12
send 16 1 32420 1
recv 17
send 18 2 32420 1
recv 19
send 20 3 32420 1
recv 21
advt 5753856
```

Human Readable Intermediate Format (debug mode)

```
// Bcast(int32,2048,0,commGrp)
send 1 1 129971 1       Send broadcast to node 1
recv 4              Receive acknowledgement for broadcast from node
1
send 2 2 129971 1       Send broadcast to node 2
recv 8              Receive acknowledgement for broadcast from node
2
// Barrier(commGrp)
send 13 1 381 1        Send barrier to node 1
recv 12              Received barrier from node 0
// Scatter(int32,512,0,commGrp)
send 16 1 32420 1       Scatter from master to node 1
recv 17              Receive acknowledgement for scatter from 1
send 18 2 32420 1       Scatter from master to node 2
recv 19              Receive acknowledgement for scatter from 2
send 20 3 32420 1       Scatter from master to node 3
recv 21              Receive acknowledgement for scatter from 3
// DotProduct(int32,64)
advt 5753856        Advance timer for compute time in dot product
```

Device Case Study: TILE-Gx36

- Many-core processor from Tilera (then EZchip, now Mellanox)
  - 36 64-bit cores or tiles with local L1 and shared L2 caches
  - 6x6 2D mesh interconnect called iMesh
    - Non-blocking switches
    - One out of five networks is user accessible (User Dynamic Network)



Device under study
(TILE-Gx36)

Behavioral Emulation with
interacting BEOs

*Spectral Element Solver

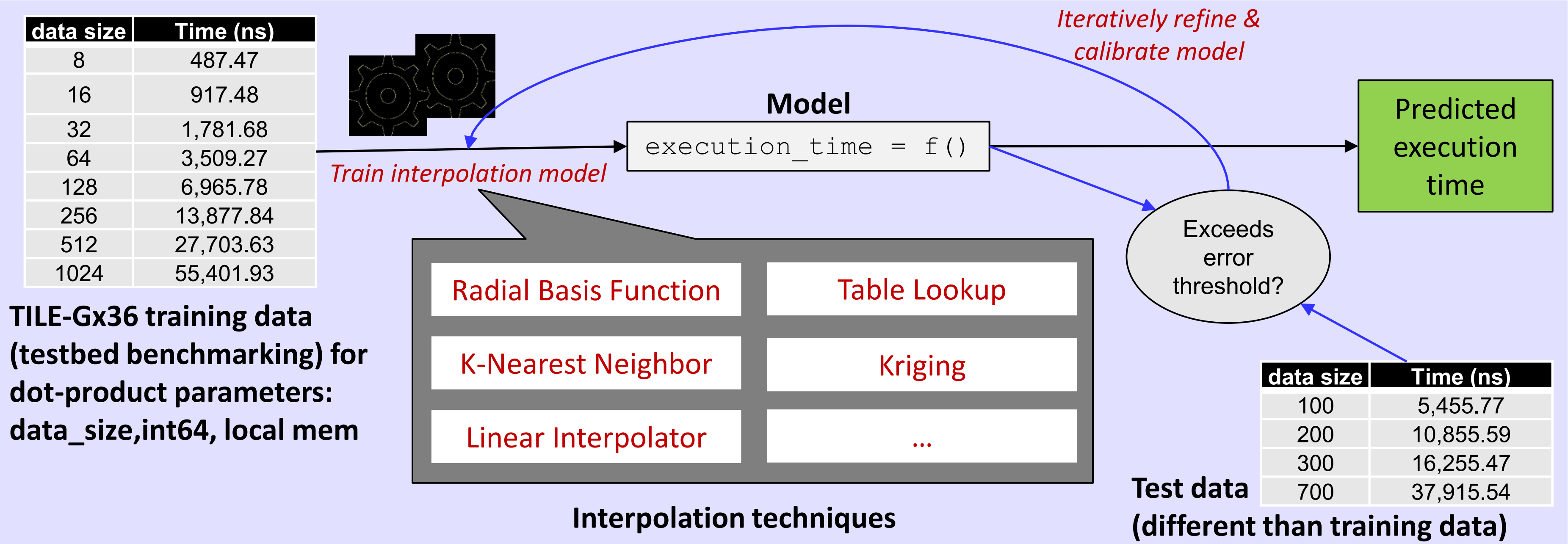
CCMT

Example: ProcBEO for TILE-Gx36*

- Mimic behavior of TILE-GX36 device
  - Read and decode AppBEO instructions
  - Resolve computes (determine performance)
  - Update local clock
  - Assign communication instructions to CommBEO

Pseudo-code for ProcBEO

```
if (init) {
  clock=clock+t_init}
if (mem_init){…}
if(compute_dot_product){…}
if(scatter){…}

           ...
```



data size	Time (ns)
8	487.47
16	917.48
32	1,781.68
64	3,509.27
128	6,965.78
256	13,877.84
512	27,703.63
1024	55,401.93

TILE-Gx36 training data (testbed benchmarking) for dot-product parameters: data_size,int64, local mem

Train interpolation model

Model

`execution_time = f()`

Iteratively refine & calibrate model

Exceeds error threshold?

Predicted execution time

Radial Basis Function | Table Lookup
K-Nearest Neighbor | Kriging
Linear Interpolator | …

Interpolation techniques

data size	Time (ns)
100	5,455.77
200	10,855.59
300	16,255.47
700	37,915.54

Test data (different than training data)

D. Rudolph and G. Stitt. "An interpolation-based approach to multi-parameter performance modeling for heterogeneous systems". In IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP), July 2015

ProcBEO Calibration (Tile-Gx36)

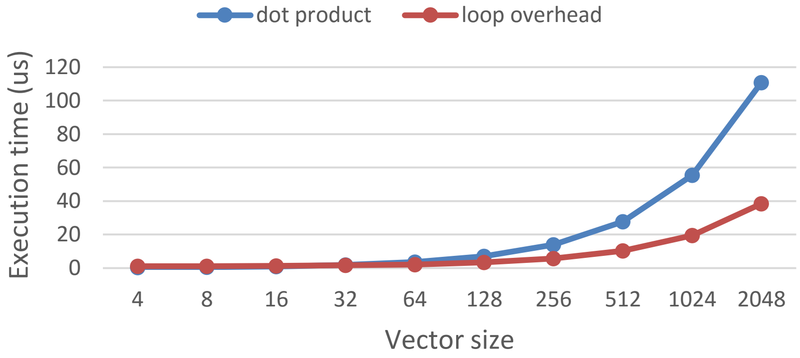
- Example data from Tilera testbed
- Data have varying dimension
  - Zero-dimensional: Pixel Gradient
  - One-dimensional: Dot Product
  - Multi-dimensional: Matrix Multiply

Gradient calculation of one pixel

x-gradient computation time = 931ns

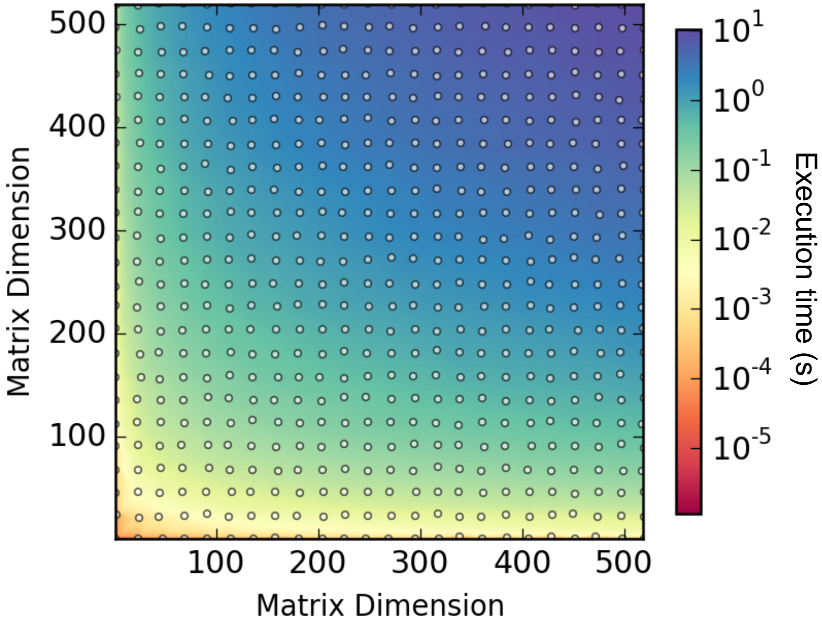
y-gradient computation time = 952ns

Dot product (int32) and Loop Overhead



2D Matrix Multiply

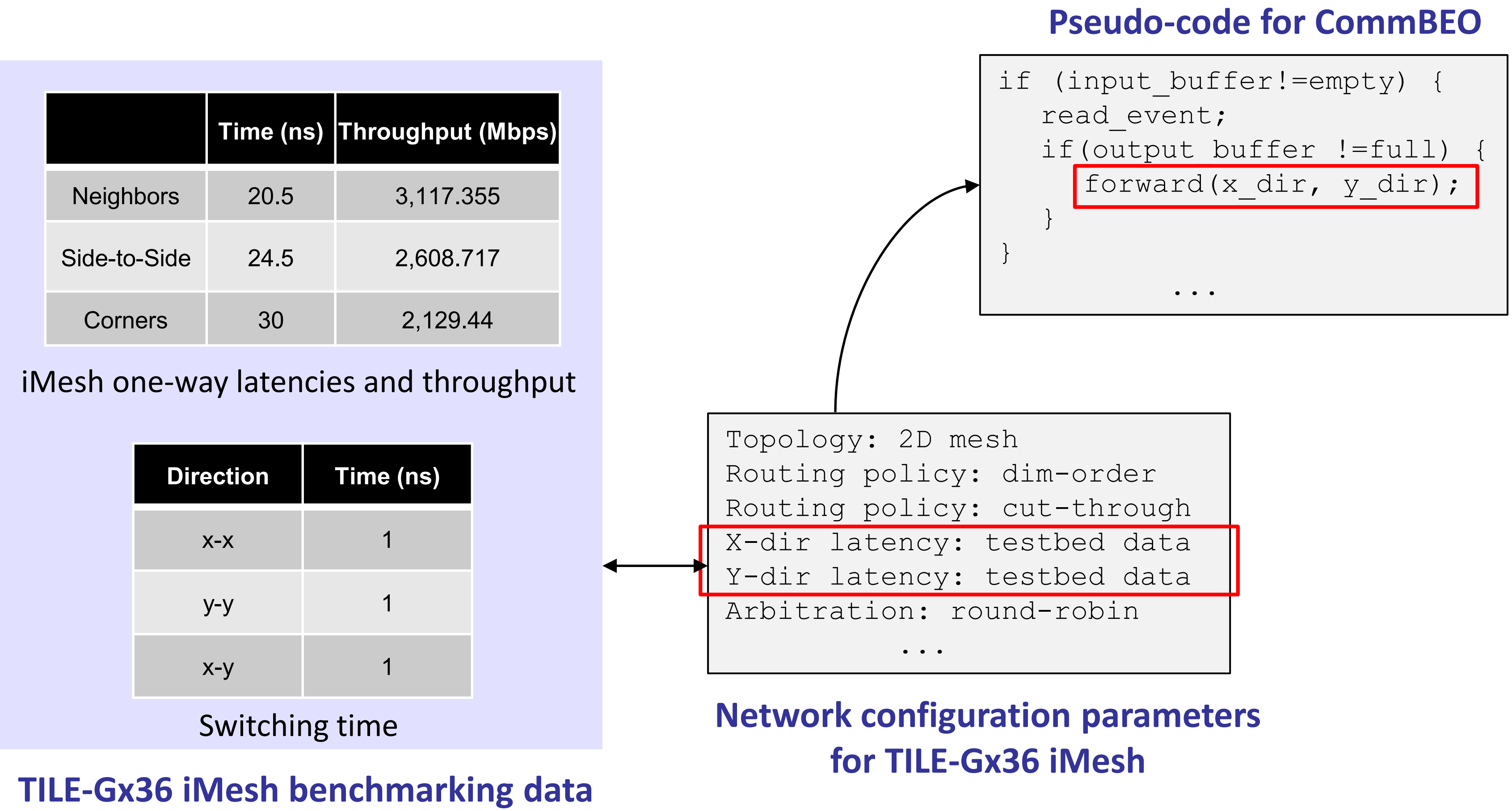
(MxN and NxN)



CCMT

Example: CommBEO for iMesh

- Mimic Tilera iMesh network behavior
  - Topology, routing policy, arbitration, etc.



	Time (ns)	Throughput (Mbps)
Neighbors	20.5	3,117.355
Side-to-Side	24.5	2,608.717
Corners	30	2,129.44

iMesh one-way latencies and throughput

Direction	Time (ns)
x-x	1
y-y	1
x-y	1

Switching time

TILE-Gx36 iMesh benchmarking data

```
Topology: 2D mesh
Routing policy: dim-order
Routing policy: cut-through
X-dir latency: testbed data
Y-dir latency: testbed data
Arbitration: round-robin
        ...
```

Network configuration parameters for TILE-Gx36 iMesh

Pseudo-code for CommBEO

```
if (input_buffer!=empty) {
  read_event;
  if(output buffer !=full) {
    forward(x_dir, y_dir);
  }
}
          ...
```

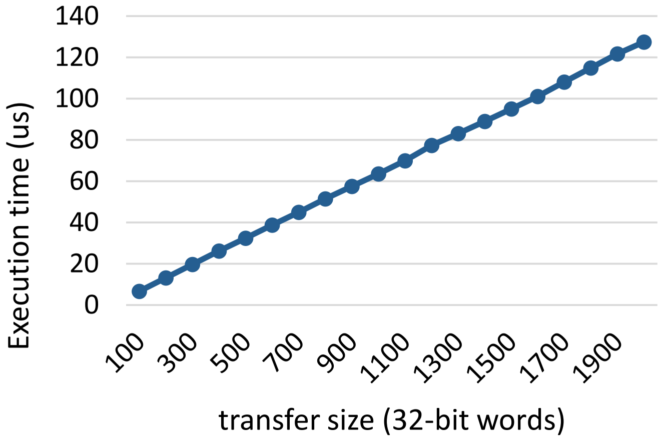
CommBEO Calibration (iMesh)

- CommBEOs require both quantitative and qualitative parameter values
  - Qualitative parameters (left) are used to mimic movement of packets in network
  - Quantitative parameters (right) help in estimating communication time
    - Some Quantitative parameters are functions of independent variables (e.g., latency)
    - Others are fixed information about the network (e.g., hop time)

Network configuration parameters

- Topology: 2D mesh
- Mesh size: 6x6
- Routing policy: dim-order
- Routing policy: store and forward
- Arbitration: round-robin

Round-trip latency



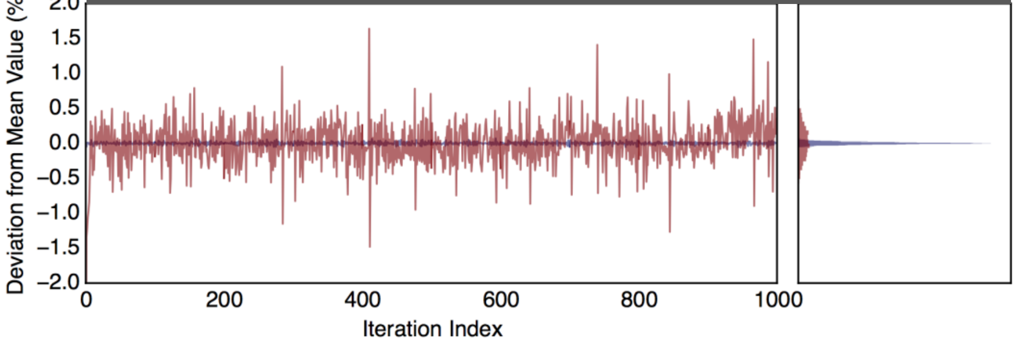
Switching Time

Direction	Time (ns)
x-x	1
y-y	1
x-y	1

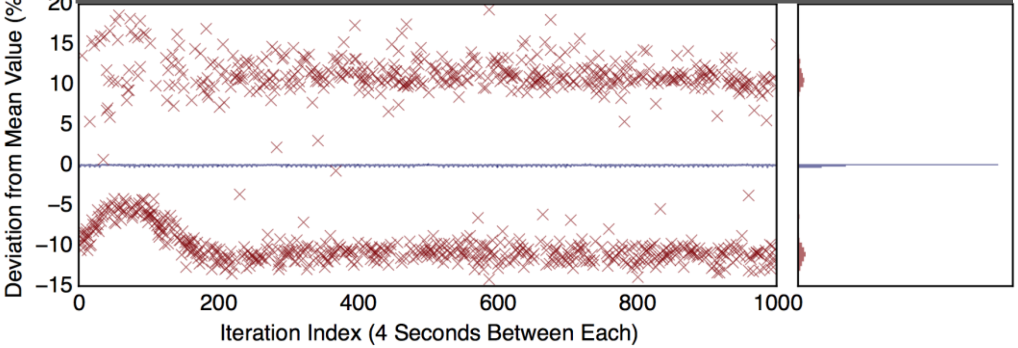
Hop Time: 1ns

Additional notes on Modeling Data

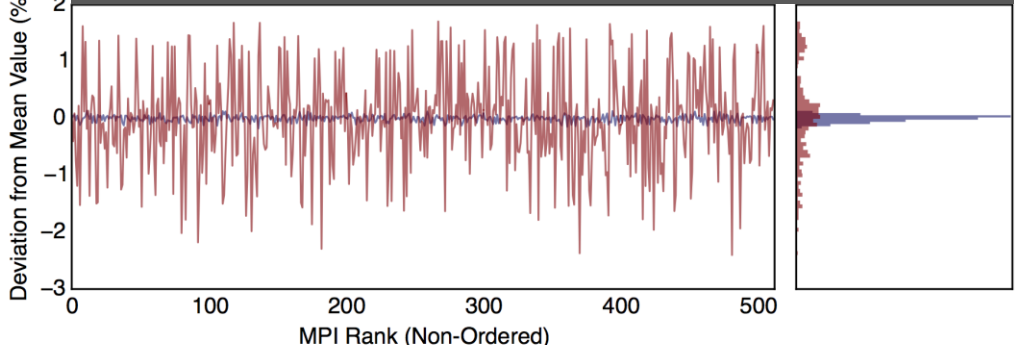
- Potentially some factors to account for in collecting source data to build BE models
- Vulcan & Cab are two large machines at LLNL
- Observations:
  - Vulcan is much more consistent than Cab for each of these cases
  - Vulcan has less variation across different allocations compared to Cab for 10 random node allocations (0.106% vs 2.66%) (Not plotted on right)
- Issues manifest on a per-machine basis; needs
  - Careful benchmarking practices
  - UQ input to improve models



Temporal difference over 1 minute (Rank 0)



Temporal difference over 1 hour (Rank 0)



Spatial difference across 512 MPI ranks (1 timestep)

Red: Cab
Blue: Vulcan

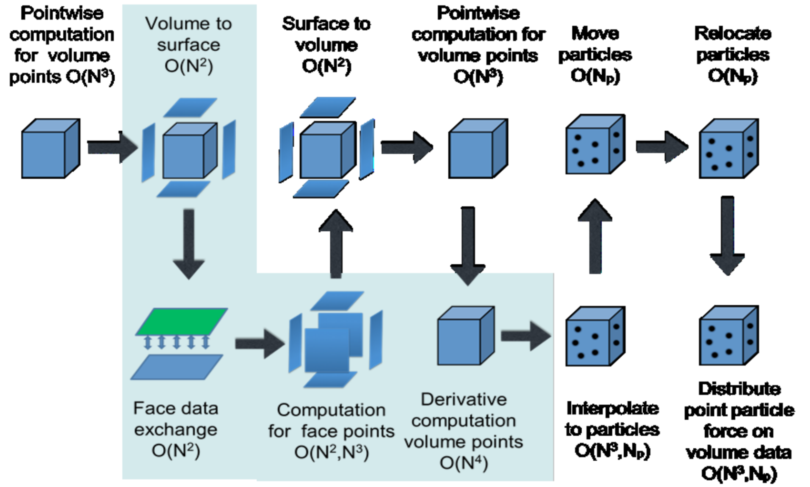
Outline

- The Big Picture – Modeling and Simulation for Co-design
- Our M&S approach – Behavioral Emulation
  - Overview and Workflow of Behavioral Emulation
- Modeling
  - What are we modeling? What are the independent parameters?
  - Building the models and model representations!
  - Measurements (what does our data look like?)
- Simulation
  - Combining the models together
  - Validation
- Prediction: Finally what we wanted all along!
  - Design Space Exploration
  - Probabilistic simulations
- Conclusions & Future Directions

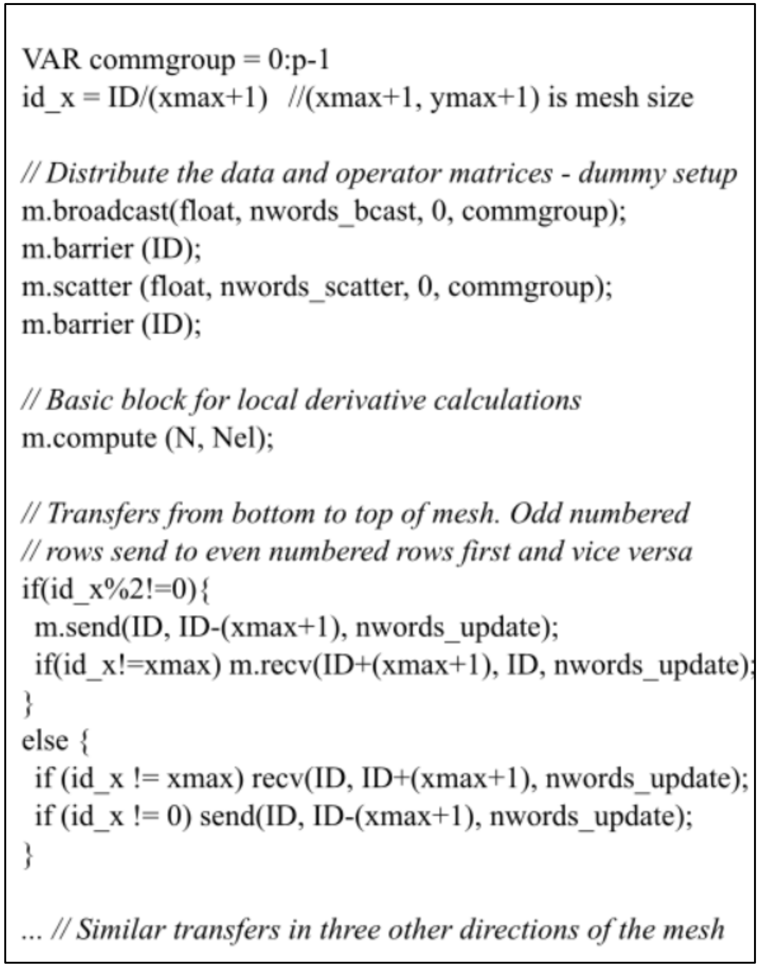
Our Capstone Application: CMT-nek SES*

- CMT-nek is an code being developed to solve an exascale problem
  - It is a moving target – not well suited for early-stage in-depth analysis
- Most computationally expensive and most prominent communication routines evolved into a “mini-app” – CMT-bone
  - Mini-app development is a joint effort between CS & Physics groups

CMT-nek Workflow



*Spectral Element Solver

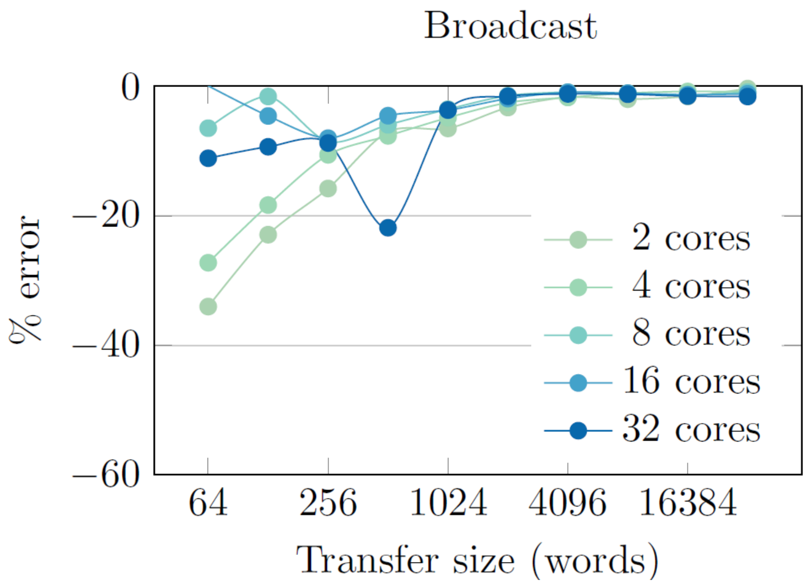


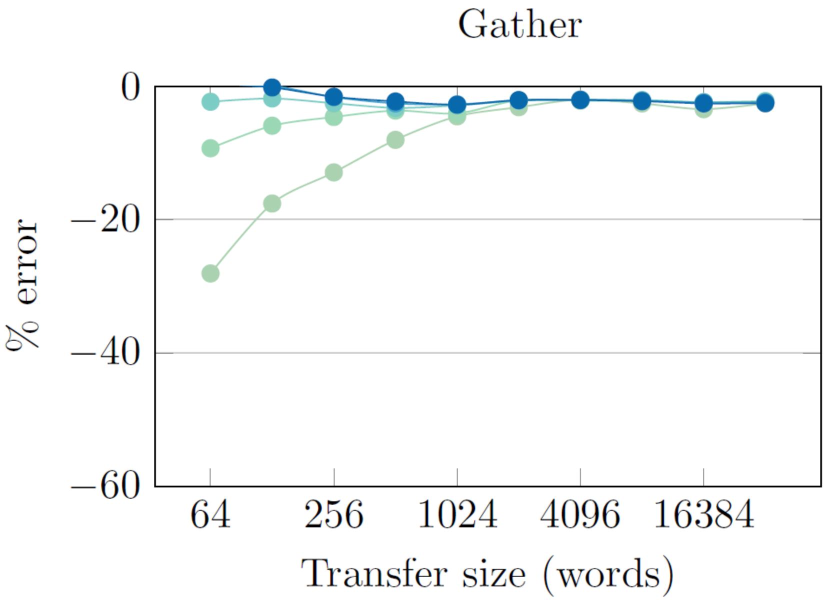
```
VAR commgroup = 0:p-1
id_x = ID/(xmax+1)   //(xmax+1, ymax+1) is mesh size

// Distribute the data and operator matrices - dummy setup
m.broadcast(float, nwords_bcast, 0, commgroup);
m.barrier (ID);
m.scatter (float, nwords_scatter, 0, commgroup);
m.barrier (ID);

// Basic block for local derivative calculations
m.compute (N, Nel);

// Transfers from bottom to top of mesh. Odd numbered
// rows send to even numbered rows first and vice versa
if(id_x%2!=0){
  m.send(ID, ID-(xmax+1), nwords_update);
  if(id_x!=xmax) m.recv(ID+(xmax+1), ID, nwords_update);
}
else {
  if (id_x != xmax) recv(ID, ID+(xmax+1), nwords_update);
  if (id_x != 0) send(ID, ID-(xmax+1), nwords_update);
}

... // Similar transfers in three other directions of the mesh
```

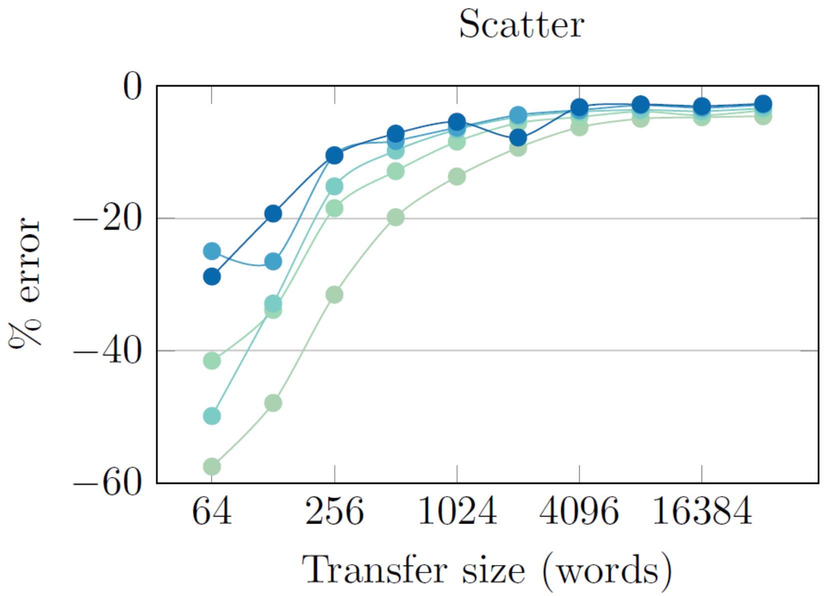
N. Kumar, M. Sringarpure, T. Banerjee, J. Hackl, S. Balachandar, H. Lam, A. George, and S. Ranka, "CMT-bone: A Mini-app for Compressible Multiphase Turbulence Simulation Software", WRAp 2015

Communication Microbenchmarks

- Setup: Tilera iMesh network CommBEOs
- Observation:
  - Simulations under-predict execution time in most cases, can improve calibration to account for setup overhead
  - Accuracy broadly improves with increase in number of cores and transfer size (large message sizes)
  - Need better latency models


Broadcast

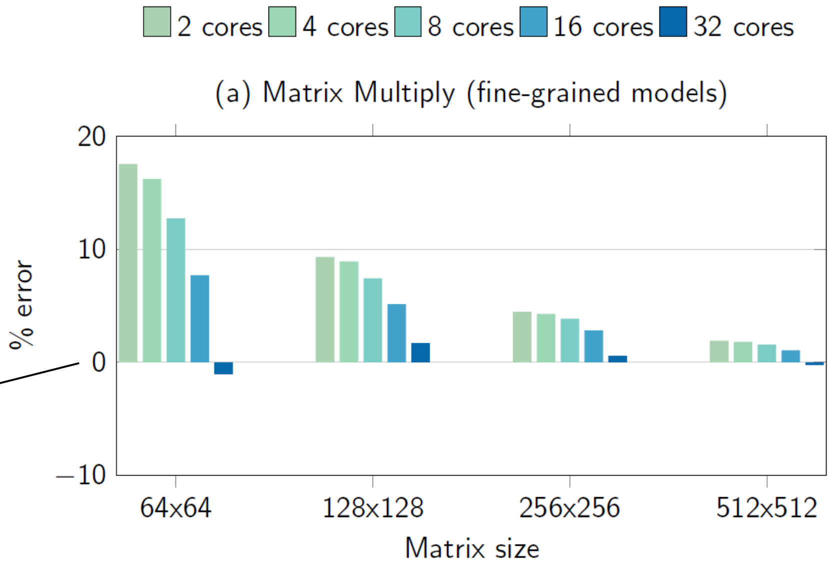

Gather


Scatter

Parallel 2D Matrix Multiply

Simulation setup:

- Calibration: compute models for dot product, loop overhead, & network parameters
- Application: Row-decomposition with data sharing by explicit transfers



Fewer cores means more share of work performed by each processor. For fine-grained decomposition, more error incurred.

- Computation dominates communication, resulting in high total error
- Error in dot-product model gets multiplied several times over

2 cores (% error)

matrix size	Bcast	Scatter	Compute	Gather	Total
64x64	-2.91	-0.94	18.79	-2.61	17.51
128x128	-2.93	-0.58	10.04	-2.92	9.30
256x256	-3.23	-1.07	5.08	-3.19	4.47
512x512	-5.04	-6.22	2.47	-6.66	1.90
1024x1024	-3.90	-5.75	1.32	-5.69	0.76

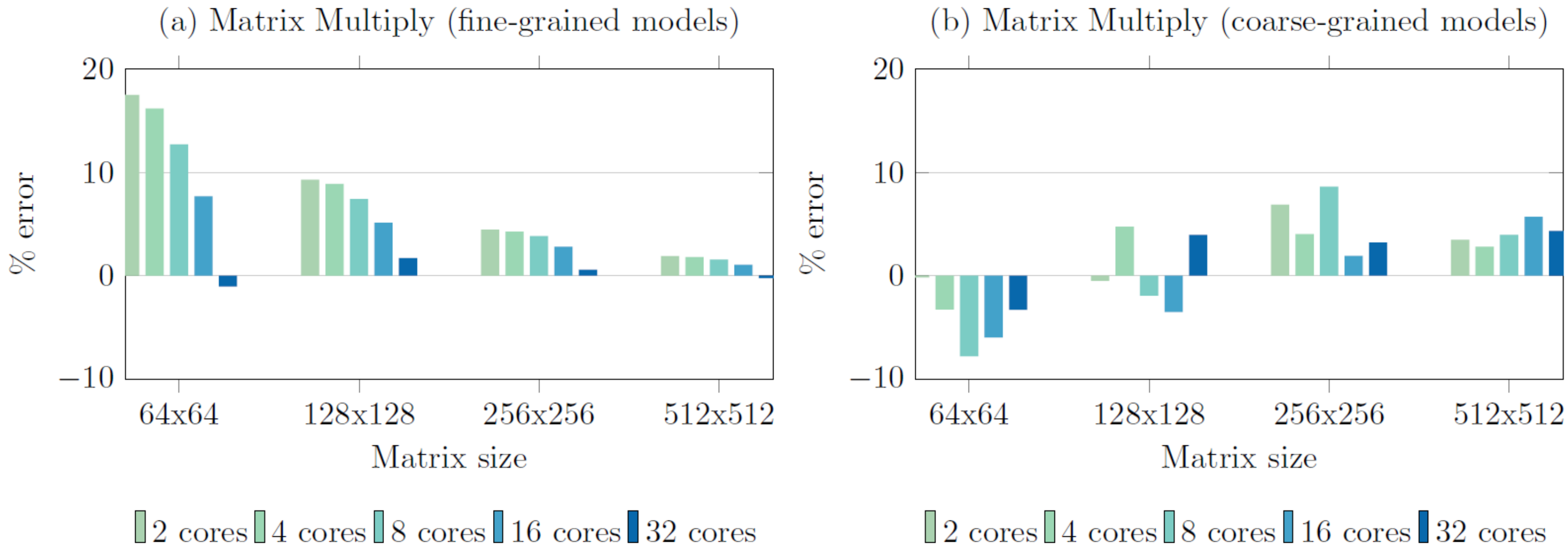
Observations:

- Accuracy of simulations improves with increase in number of cores and matrix size
- Large error values due to fine-grained decomposition of computes (dot products)
- Possible solution: Coarse-grained timing models of compute operations

Parallel 2D Matrix Multiply

Simulation setup:

- Calibration: compute models for dot product, loop overhead, & network parameters
- Application: Row-decomposition with data sharing by explicit transfers

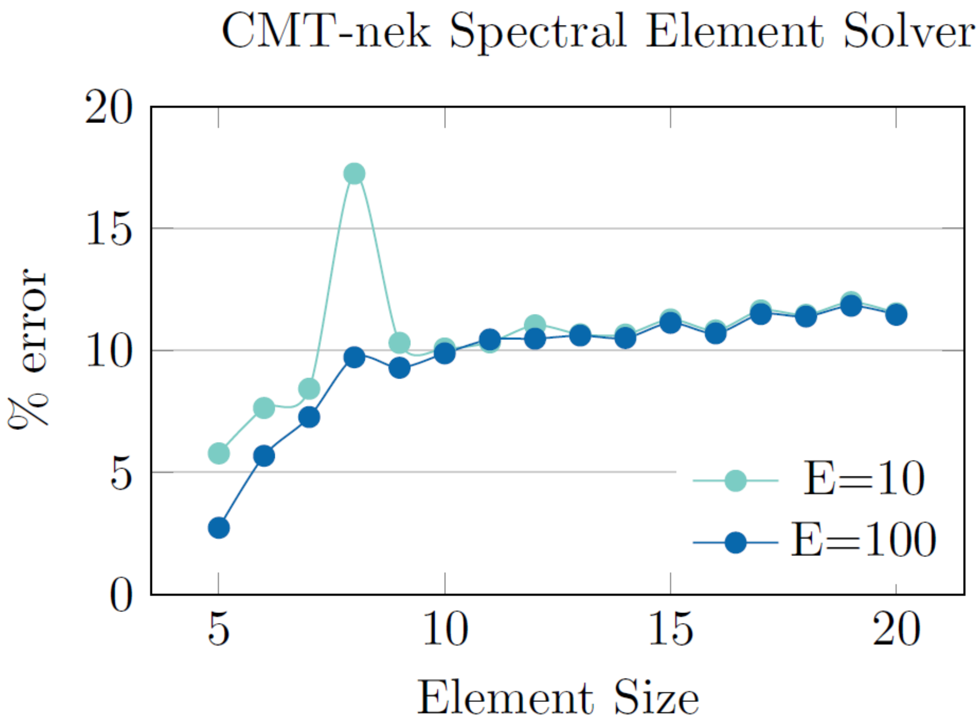


Simulation setup: compute models for matrix multiply, loop overhead, & network parameters

Observations:

- Abstraction improves simulation accuracy at a one-time cost of training effort
- Accuracy is a function of domain, no. of samples, & other kriging parameters

CMT-nek Spectral Element Solver



Simulation setup: compute models for matrix multiply, loop overhead, & network parameters

Observations:

- Abstraction improves simulation accuracy at a one-time cost of training effort
- Accuracy is a function of domain, no. of samples, & other kriging parameters

System-scale experiments on Vulcan

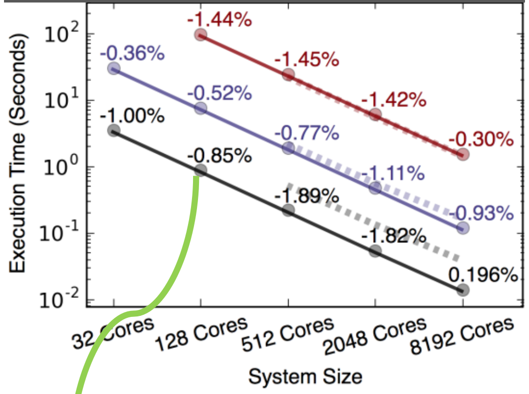
Predictions made from information from only a subset of nodes

- Foundation for simulating Exascale from Petascale systems
- Performance very well predicted, *as expected*, since:
  - Vulcan architecture is well structured and well behaved
  - CMT-bone-BE is overwhelmingly computational intensive
- Predictions closely follow the CMT-nek execution time trend

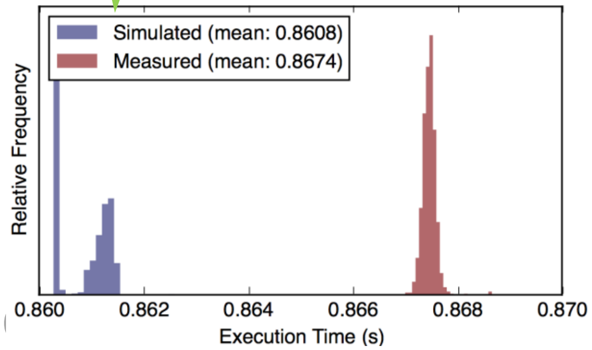
Element size: 15
Element size: 9
Element size: 5

● Measured
— Simulated
Text: Discrepancy %

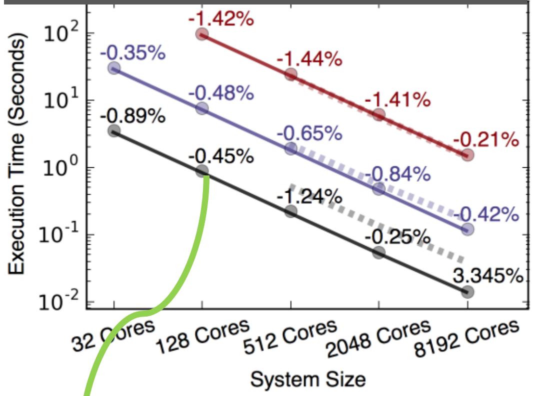
Models built at Compute Card Scale
Predicted at Midplane & Rack Scale



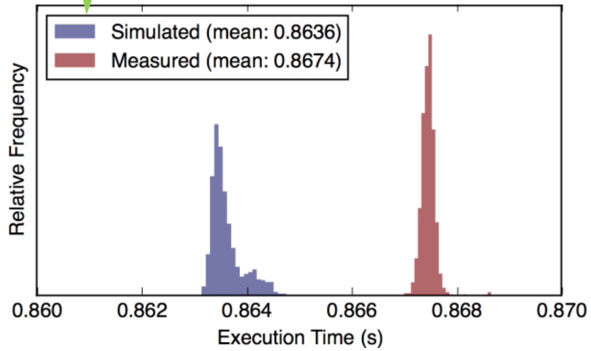
(Mean error~1.0%)



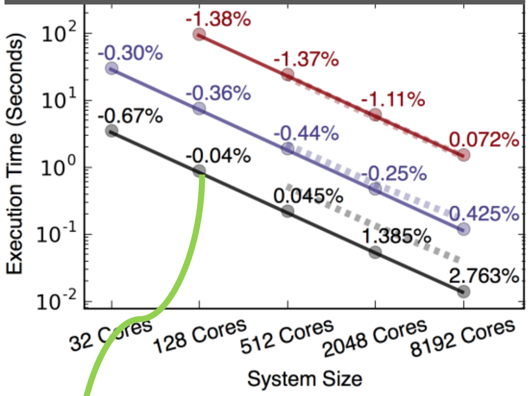
Models built at Node Card Scale
Predicted at Midplane & Rack Scale



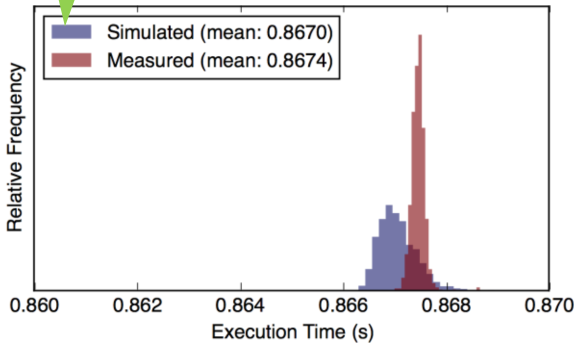
(Mean error ~0.8%)



Models built at Midplane Scale
Predicted at Rack Scale



(Mean error ~0.7%)



Outline

- The Big Picture – Modeling and Simulation for Co-design
- Our M&S approach – Behavioral Emulation
  - Overview and Workflow of Behavioral Emulation
- Modeling
  - What are we modeling? What are the independent parameters?
  - Building the models and model representations!
  - Measurements (what does our data look like?)
- Simulation
  - Step 1: Combining the models together
  - Step 2: Validation (not leave one out!) of individual block models
- Prediction: Finally what we wanted!
  - Design Space Exploration
  - Probabilistic simulations
- Conclusions & Future Directions

Case Studies for Architecture DSE

With some confidence in Behavioral Emulation approach we can proceed to study *next-generation devices*

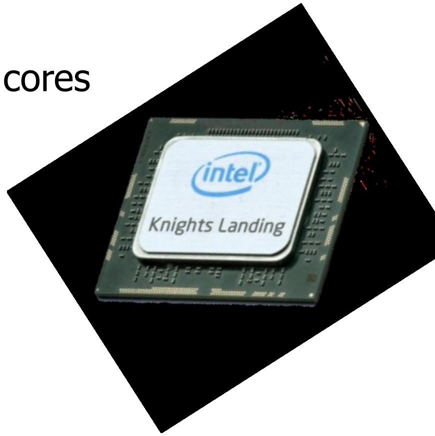
- DSE: Ability to evaluate what-if scenarios by changing BEOs parameters

Tile-Gx72: Many-core processor from Tilera (EZchip, then Mellanox)

- One of the largest device made by Tilera: 72 cores
- Cores in Tile-Gx72 are identical to cores in Tile-Gx36
- To simulate Tile-Gx72, we scale simulation to 72 Proc & CommBEOs

Mesh-based Intel processor*: Notional Intel-based many-core processor

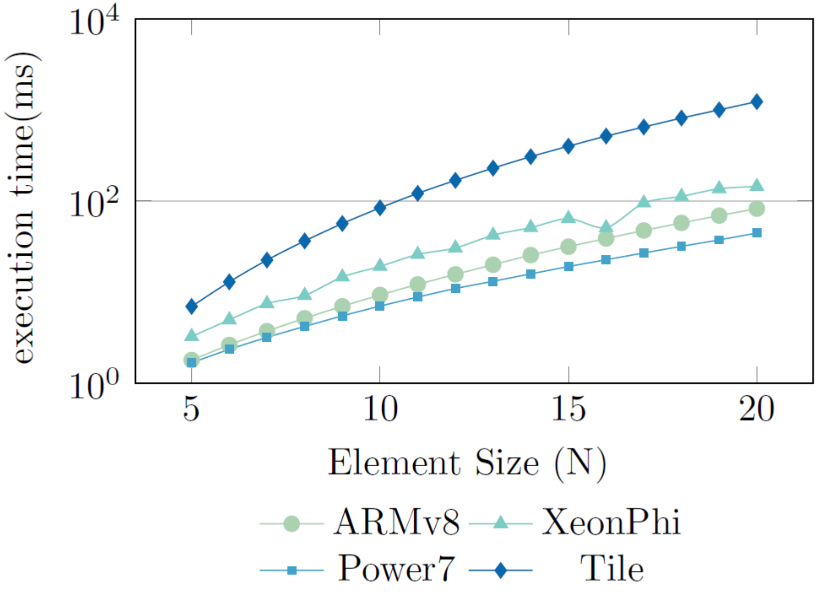
- Xeon Phi-type cores with Mesh network
- To simulate anticipated Knight's Landing
  - Calibrate ProcBEOs based on existing XeonPhi (KNC) processor cores
  - Use validated CommBEOs developed for iMesh network
- 64-core device: similar in size to existing Xeon Phi
- 100-core device: probable size; larger than existing devices

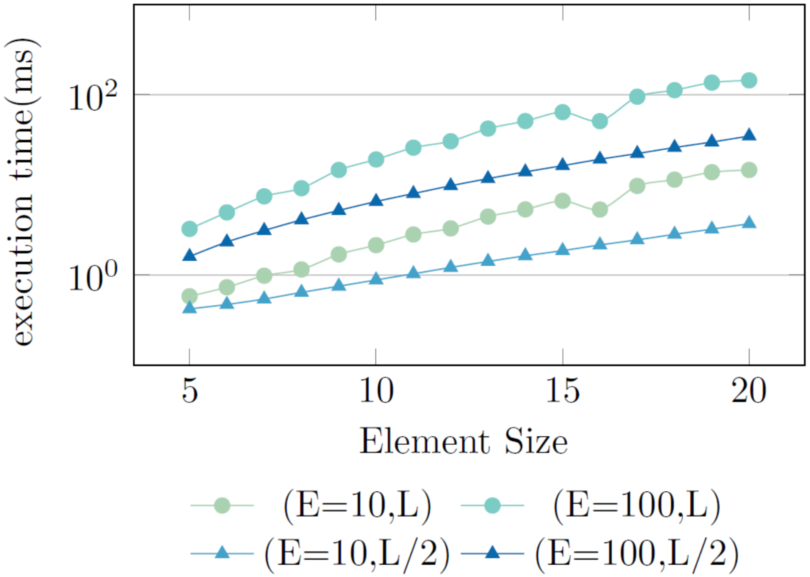


… and other notional processors with mesh-based architecture

**These simulations were conducted in 2014, before Intel confirmed details of KNL architecture*

Selected DSE Simulation Results


(a) SES on mesh devices with various ProcBEOs


(b) SES on mesh devices with different network latencies

Can evaluate many more what-if scenarios: *More processors, Faster processors, Faster network, Network configuration*

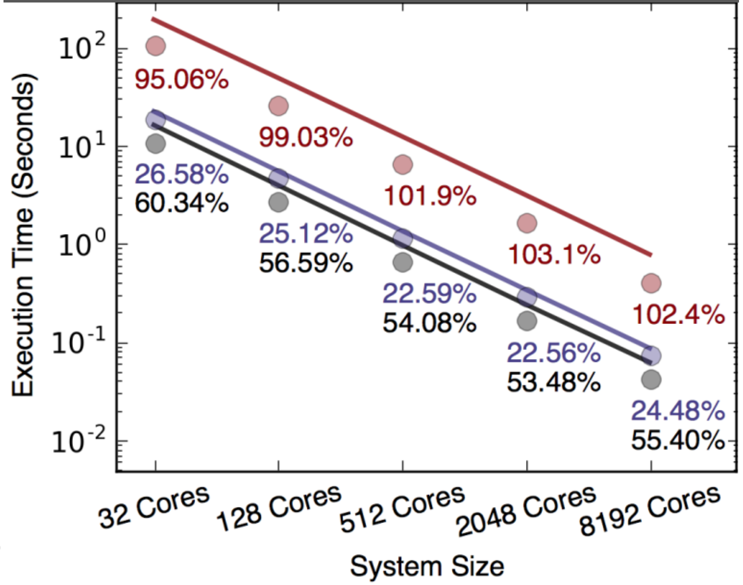
Vulcan Blind Predictions: Different Element Size

- With a very large sampling space, it is not feasible to collect a dense sample set for all model parameter values
  - Predictions for element sizes (7,8,12) made from models for element sizes (5,9,15) using interpolation
- Accuracy of predictions at off-collection-points is affected strongly by choice of interpolation technique

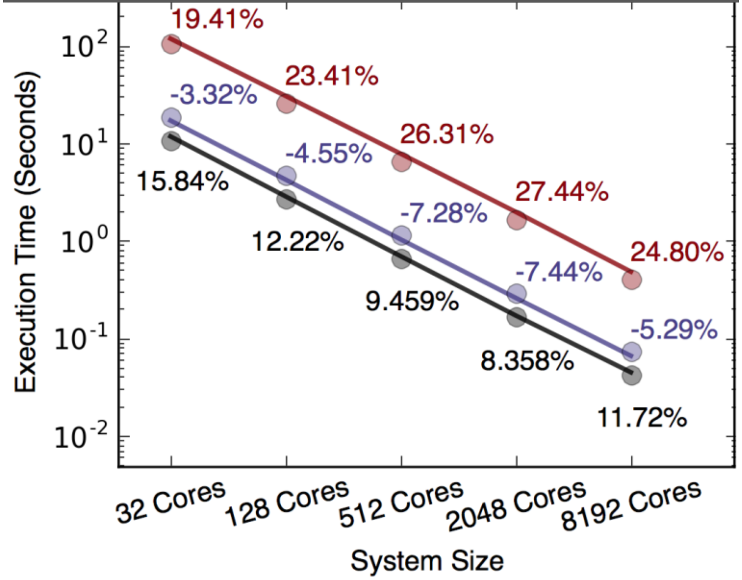
Element size: 12
Element size 8
Element size: 7

● Measured
— Simulated
Text: Discrepancy %

Predictions for various element sizes using Linear Interpolation



Predictions for various element sizes using Polynomial Interpolation



Outline

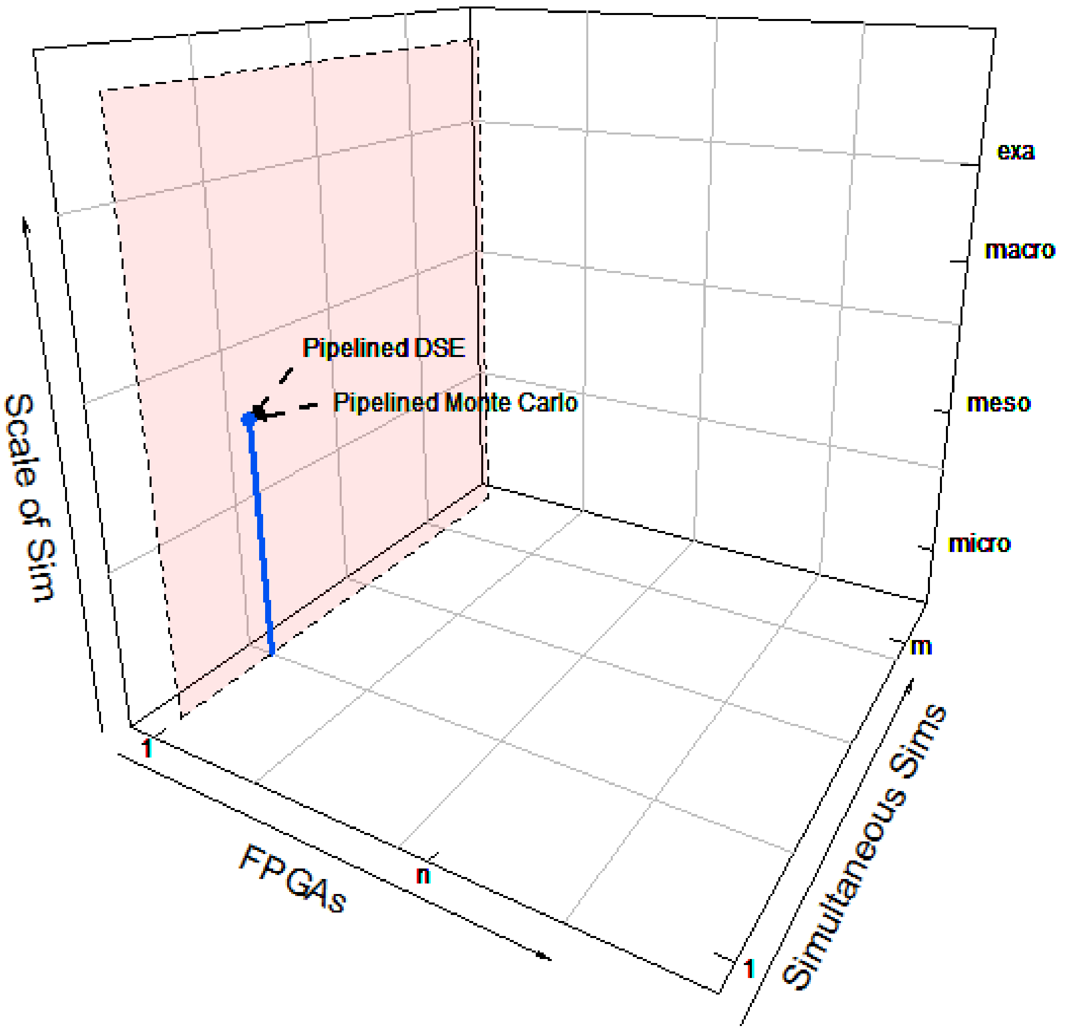
- The Big Picture – Modeling and Simulation for Co-design
- Our M&S approach – Behavioral Emulation
  - Overview and Workflow of Behavioral Emulation
- Modeling
  - What are we modeling? What are the independent parameters?
  - Building the models and model representations!
  - Measurements (what does our data look like?)
- Simulation
  - Step 1: Combining the models together
  - Step 2: Validation (not leave one out!) of individual block models
- Prediction: Finally what we wanted all along!
  - Design Space Exploration
  - Probabilistic simulations
- Conclusions & Future Directions

Future Directions

Lots of things in the works!

- Integration into a popular simulator is well underway – Structural Simulation Toolkit from Sandia National Laboratories
- Making BE easier to use:
  - Automate application modeling for broader adoption in the community
  - Systematic data collection and repeatable experiments
- Methods & practical techniques for interpolation on multi-dimensional data
- Using FPGAs for accelerating BE simulations for pruning the design space

Landscape of FPGA-acceleration Studies



Original Project Target

- *1 large, Exascale sim distributed over many FPGAs*

NGEEv1* Progress

- *1 small, microscale sim limited to a single FPGA*

NGEEv1 Enhancements

- *Ongoing improvements to allow for sims at larger scale*

NGEEv1 Parameter Sweeps

- *Multi-FPGA DSE$^+$ limited to a single simulation per device*

(NEW) Pipelined Simulations: start simulation every cycle

- *Rapid design-space exploration*
- *Monte Carlo simulation for UQ*

CCMT

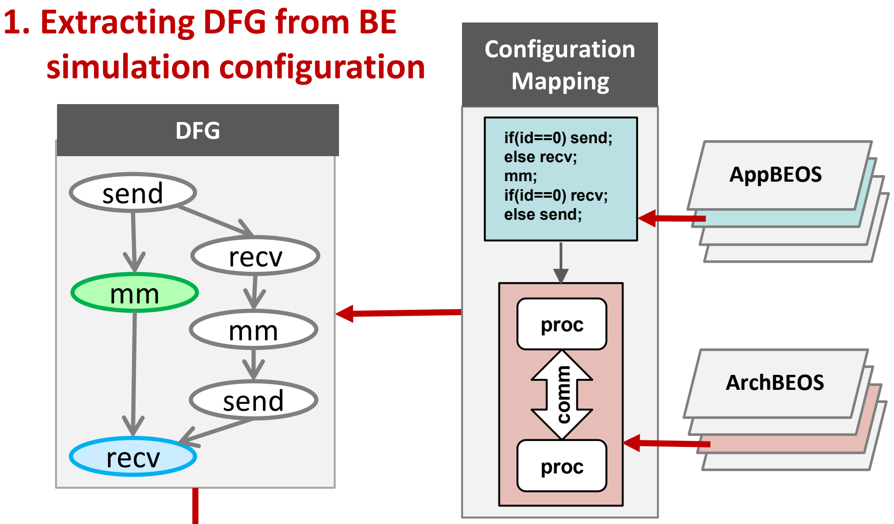
---

*NGEE: Novo-G Exascale Emulation $^+$DSE: Design-Space Exploration

Pipelined Simulations: Concept & Approach

1. Construct Data Flow Graph (DFG) from simulation configuration

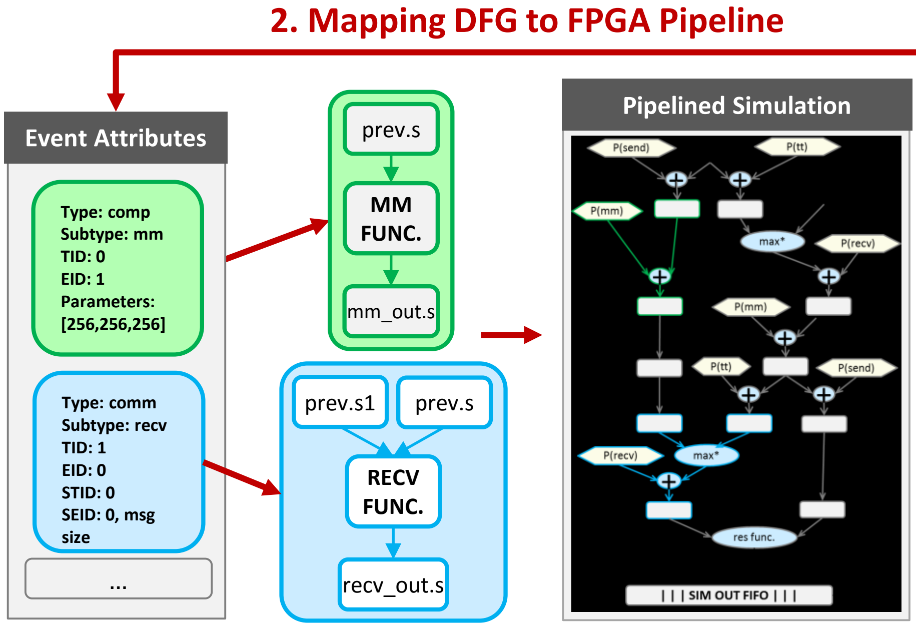
- AppBEO+ArchBEO define instructions and operand/output dependencies
- Instructions map to vertices and dependencies map to edges in DFG
- Various opportunities for graph-level optimizations



1. Extracting DFG from BE simulation configuration

```
if(id==0) send;
else recv;
mm;
if(id==0) recv;
else send;
```

2. Mapping DFG to FPGA Pipeline



Type: comp
Subtype: mm
TID: 0
EID: 1
Parameters: [256,256,256]

Type: comm
Subtype: recv
TID: 1
EID: 0
STID: 0
SEID: 0, msg size

...

2. Map DFG to pipeline circuit

- Vertex attributes define operations and instantiate dedicated HW
- Edge attributes (e.g., src/dst) instantiate pipeline register between src/dst pair
- Various opportunities for circuit-level optimizations

Because each instruction (from sim) mapped to independent HW (no resource sharing), each vertex able to *start next sim 1 cycle after current sim*

Conclusions

- Investigated and validated basic concepts and methods of BE
  - Developed prototype BEOs for benchmarks and many-core processors
  - Validated performance (simulation vs. testbed) and mostly observed modest error that can be useful for DSE
  - Demonstrated applicability of BE beyond device-level
  - Identified aspects of benchmarking & modeling which require UQ

- Laid foundation for design-space exploration
  - Predictions for Spectral Element Solver on some notional architectures
  - Blind prediction using architectural and application parameters

Questions?

Nalini Kumar

nkumar@hcs.ufl.edu

References

System (macro-scale) Simulators

- C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, “A simulator for large-scale parallel architectures” International Journal of Parallel and Distributed Systems, vol. 1, no. 2, pp. 57-73, 2010. **SST MACRO**
- E. Grobelny, D. Bueno, I. Troxel, A.D. George, and J.S. Vetter, “FASE: A Framework for Scalable Performance Prediction of HPC Systems and Applications, Simulation”, Simulation, Vol. 83, No. 10, pp. 721-745, Oct. 2007. **FASE**
- G. Zheng, G. Kakulapati, L. V. Kale, “Bigsim: A parallel simulator for performance prediction of extremely large parallel machines”, 18th IPDPS, pp. 78, 2004. **BIGSIM**
- A. D. George, R. B. Fogarty, J. S. Markwell, and M. D. Miars, “An Integrated Simulation Environment for Parallel and Distributed System Prototyping”, Simulation, vol. 72, pp. 283-294, May 1999. **ISE**
- A. Symons, V. L. Narasimhan, "Parsim-message PAssing computeR SIMulator," IEEE First International Conference on Algorithms and Architectures for Parallel Processing, vol. 2, pp. 621, 630, 19-20, ICAPP, 1995. **PARSIM**

References

Device (micro-scale) & Node (meso-scale) Simulators

- Z. Dong, J. Wang, G. Riley, and S. Yalamanchili, “An Efficient Front-End for Timing-Directed Parallel Simulation of Multi-Core System”, 7th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2014), March 2014. **MANIFOLD**
- J. Wang, J. Beu, S. Yalamanchili, and T. Conte. “Designing Configurable, Modifiable and Reusable Components for Simulation of Multicore Systems”, 3rd International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, November 2012. **MANIFOLD**
- M. Hseih, R. Riesen, K. Thompson,W. Song, A. Rodrigues, “SST: A Scalable Parallel Framework for Architecture-Level Performance, Power, Area and Thermal Simulation”, Computer Journal, vol. 55, no. 2, pp. 181-191, 2012. **SST MICRO**
- M. Hseih, A. Rodrigues, R. Riesen, K. Thompson,W. Song, “A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration”, SIGMETRICS, Performance Evaluation Review, vol. 38, no. 4, pp. 63-68 2011. **SST MICRO**

Object-oriented System Modeling

- J. C. Browne, E. Houstis, and J. R. Purdue, “POEMS – End to End Performance Models for Dynamic Parallel and Distributed Systems”

References

Hardware Emulation

- Z. Tan, A. Waterman, H. Cook, S. Bird, K. Asanovi, and D. Patterson, “A Case for FAME : FPGA Architecture Model Execution”, ISCA’10, June 19–23, 2010, Saint-Malo, France, 290–301.
- J. Wawrzynek, D. A. Patterson, S. Lu, and J. C. Hoe, “RAMP: A Research Accelerator for Multiple Processors”, 2006.

Supercomputer-specific Modeling & Simulation

- S. R. Alam, R.F. Barrett, M. R. Fahey, J. M. Larkin, and P.H. Worley, “Cray XT4 : An Early Evaluation for Petascale Scientific Simulation”, 2007.
- A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, and S. Pakin, “A Performance Comparison Through Benchmarking and Modeling of Three Leading Supercomputers : Blue Gene / L , Red Storm , and Purple”, (November), 1–10, 2006.

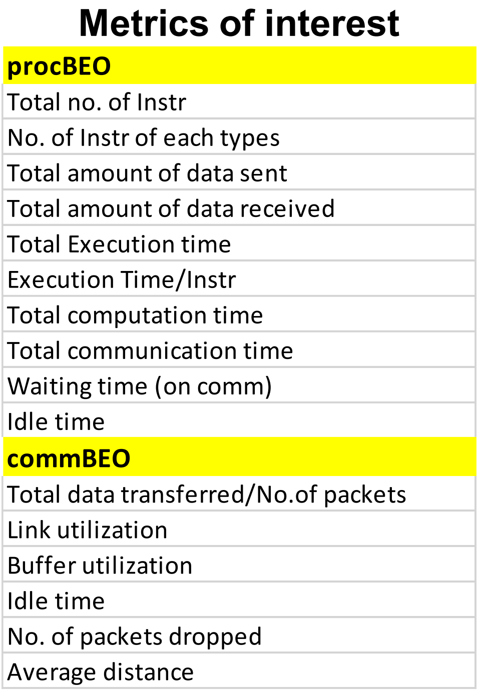
Analytical Modeling

- L. Carrington, A. Snavely, and N. Wolter, “A performance prediction framework for scientific applications”. Future Generation Computer Systems, 22(3), 336–346.
- N. Jindal, V. Lotrich, E. Deumens, B.A. Sanders, and I. Sci, “ SIPMaP : A Tool for Modeling Irregular Parallel Computations in the Super Instruction Architecture”, IPDPS 2013

APPENDIX

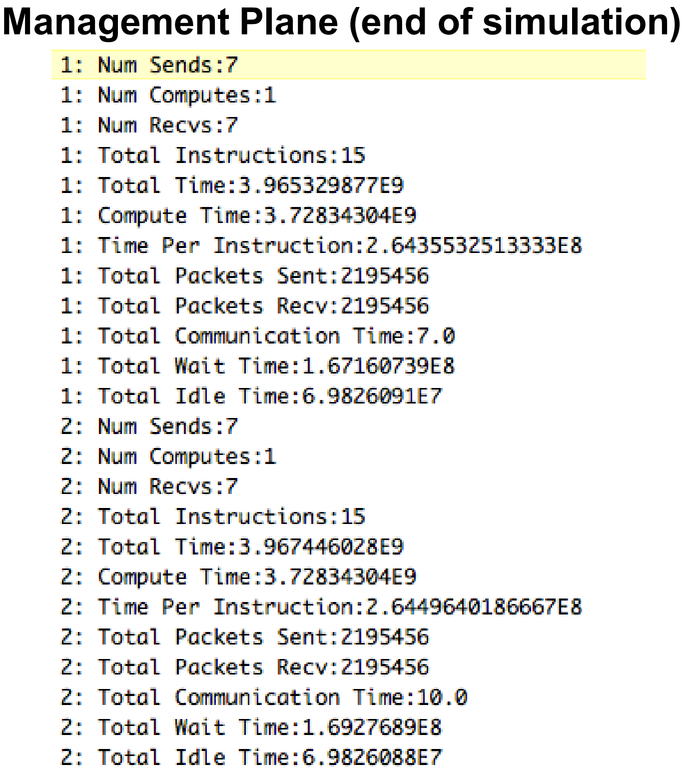
Emulation Output

- Management plane of BEOs collects various metrics of interest during simulation run



Metrics of interest

procBEO
Total no. of Instr
No. of Instr of each types
Total amount of data sent
Total amount of data received
Total Execution time
Execution Time/Instr
Total computation time
Total communication time
Waiting time (on comm)
Idle time
commBEO
Total data transferred/No.of packets
Link utilization
Buffer utilization
Idle time
No. of packets dropped
Average distance

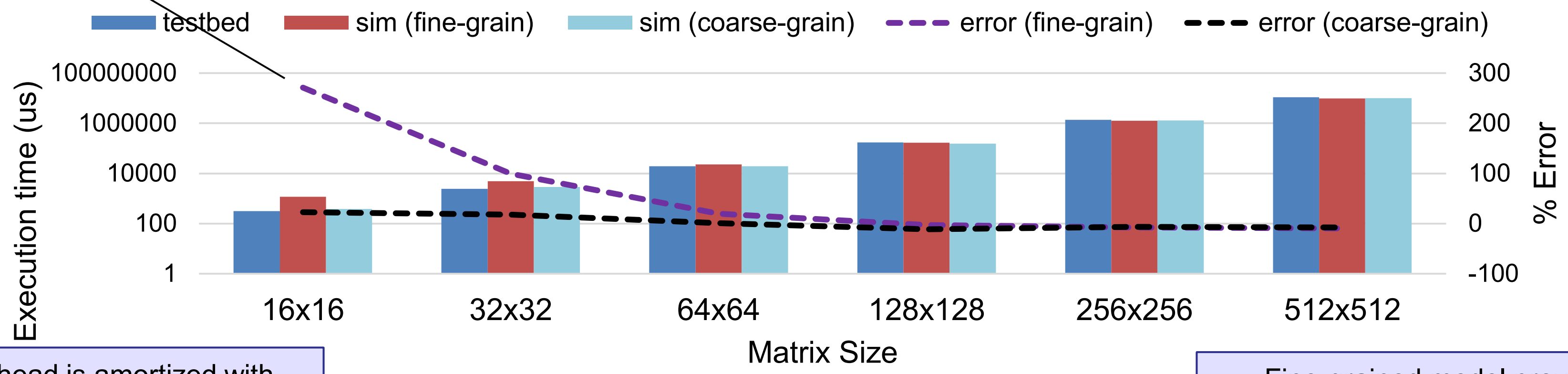


Management Plane (end of simulation)

```
1: Num Sends:7
1: Num Computes:1
1: Num Recvs:7
1: Total Instructions:15
1: Total Time:3.965329877E9
1: Compute Time:3.72834304E9
1: Time Per Instruction:2.6435532513333E8
1: Total Packets Sent:2195456
1: Total Packets Recv:2195456
1: Total Communication Time:7.0
1: Total Wait Time:1.67160739E8
1: Total Idle Time:6.9826091E7
2: Num Sends:7
2: Num Computes:1
2: Num Recvs:7
2: Total Instructions:15
2: Total Time:3.967446028E9
2: Compute Time:3.72834304E9
2: Time Per Instruction:2.6449640186667E8
2: Total Packets Sent:2195456
2: Total Packets Recv:2195456
2: Total Communication Time:10.0
2: Total Wait Time:1.6927689E8
2: Total Idle Time:6.9826088E7
```

Compute Microbenchmarks

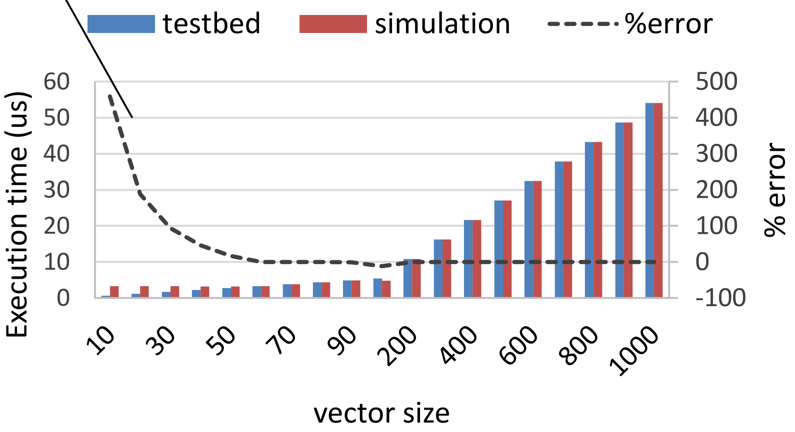
Granularity of problem decomposition has significant effect on accuracy



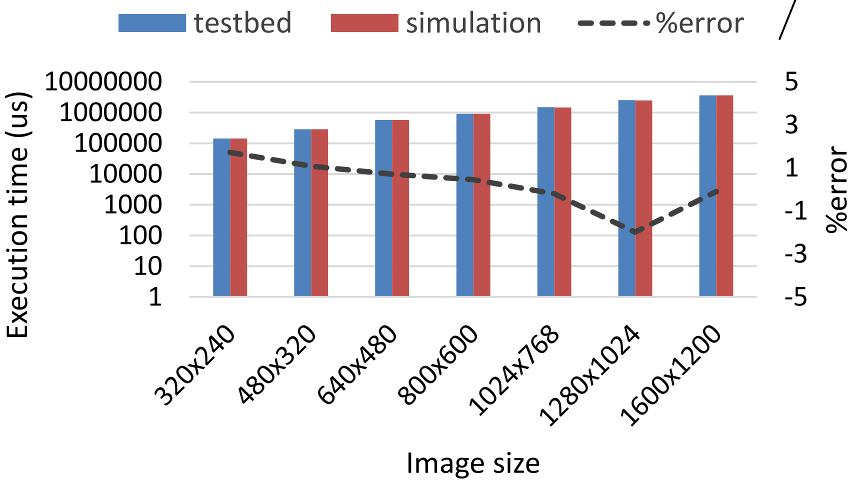
Prediction Error in single-core Matrix Multiply

Overhead is amortized with increase in problem size

Fine-grained model provides desirable accuracy for this algorithm



Prediction Error in single-core **Dot Product**



Prediction Error in single-core **Sobel Filtering**

Parallel 2D Matrix Multiply

(Breakdown: Fine-grained compute model)

% Error in predicting different portions of kernel

	2 cores					4 cores				
matrix size	Bcast	Scatter	Compute	Gather	Total	Bcast	Scatter	Compute	Gather	Total
64x64	-2.91	-0.94	18.79	-2.61	17.51	-2.41	-2.82	19.00	-2.98	16.19
128x128	-2.93	-0.58	10.04	-2.92	9.30	-2.58	0.45	10.06	-2.41	8.90
256x256	-3.23	-1.07	5.08	-3.19	4.47	-3.10	-1.63	5.08	-3.05	4.28
512x512	-5.04	-6.22	2.47	-6.66	1.90	-4.70	-4.62	2.49	-4.10	1.81
1024x1024	-3.90	-5.75	1.32	-5.69	0.76	-5.10	-6.93	1.32	-5.76	0.65

	8 cores					16 cores				
matrix size	Bcast	Scatter	Compute	Gather	Total	Bcast	Scatter	Compute	Gather	Total
64x64	-1.92	-3.35	18.79	-2.47	12.71	-1.52	-3.83	18.65	-2.08	7.70
128x128	-2.61	-0.52	9.73	-2.70	7.42	-2.72	-2.05	9.36	-2.55	5.14
256x256	-3.10	-2.91	5.05	-2.55	3.85	-3.04	-2.66	4.90	-3.10	2.82
512x512	-4.28	-5.14	2.45	-3.10	1.57	-4.04	-5.55	2.34	-2.74	1.06
1024x1024	-5.67	-8.77	1.28	-5.34	0.57	-6.81	-12.21	1.18	-4.70	0.13

	32 cores				
matrix size	Bcast	Scatter	Compute	Gather	Total
64x64	-1.10	-4.30	15.47	-1.75	-1.05
128x128	-1.78	-2.37	8.87	-3.55	1.71
256x256	-3.27	-6.80	4.68	-4.55	0.58
512x512	-4.02	-7.98	2.22	-3.04	-0.23
1024x1024	-5.86	-13.21	1.06	-4.23	-0.35

Observations:

- Under-prediction of communication time & over-prediction of compute time results in errors canceling out
- Worst-case error: 17.51%
- Best-case error: 0.13%

Parallel 2D Matrix Multiply

(Breakdown: Coarse-grained compute model)

% Error in predicting different portions of kernel

	2 cores					4 cores				
matrix size	Bcast	Scatter	Compute	Gather	Total	Bcast	Scatter	Compute	Gather	Total
64x64	-2.91	-0.94	0.52	-2.61	-0.15	-2.41	-2.82	-2.53	-2.98	-3.26
128x128	-2.93	-0.58	0.05	-2.92	-0.50	-2.58	0.45	5.70	-2.41	4.76
256x256	-3.23	-1.07	7.51	-3.19	6.87	-3.10	-1.63	4.83	-3.05	4.03
512x512	-5.04	-6.22	4.06	-6.66	3.47	-4.70	-4.62	3.51	-4.10	2.81

	8 cores					16 cores				
matrix size	Bcast	Scatter	Compute	Gather	Total	Bcast	Scatter	Compute	Gather	Total
64x64	-1.92	-3.35	-8.58	-2.47	-7.78	-1.52	-3.83	-7.64	-2.08	-5.97
128x128	-2.61	-0.52	-1.18	-2.70	-1.92	-2.72	-2.05	-3.17	-2.55	-3.51
256x256	-3.10	-2.91	10.24	-2.55	8.63	-3.04	-2.66	3.81	-3.10	1.93
512x512	-4.28	-5.14	4.95	-3.10	3.96	-4.04	-5.55	7.54	-2.74	5.70

	32 cores				
matrix size	Bcast	Scatter	Compute	Gather	Total
64x64	-1.10	-4.30	7.37	-1.75	-3.29
128x128	-1.78	-2.37	13.91	-3.55	3.95
256x256	-3.27	-6.80	8.99	-4.55	3.21
512x512	-4.02	-7.98	8.28	-3.04	4.35

Observations:

- Under-predicting communication time as before
- Compute predictions improve for small cores & problem sizes
- Worst-case error: 8.63%
- Best-case error: -0.15%

Parallel Sobel Filtering

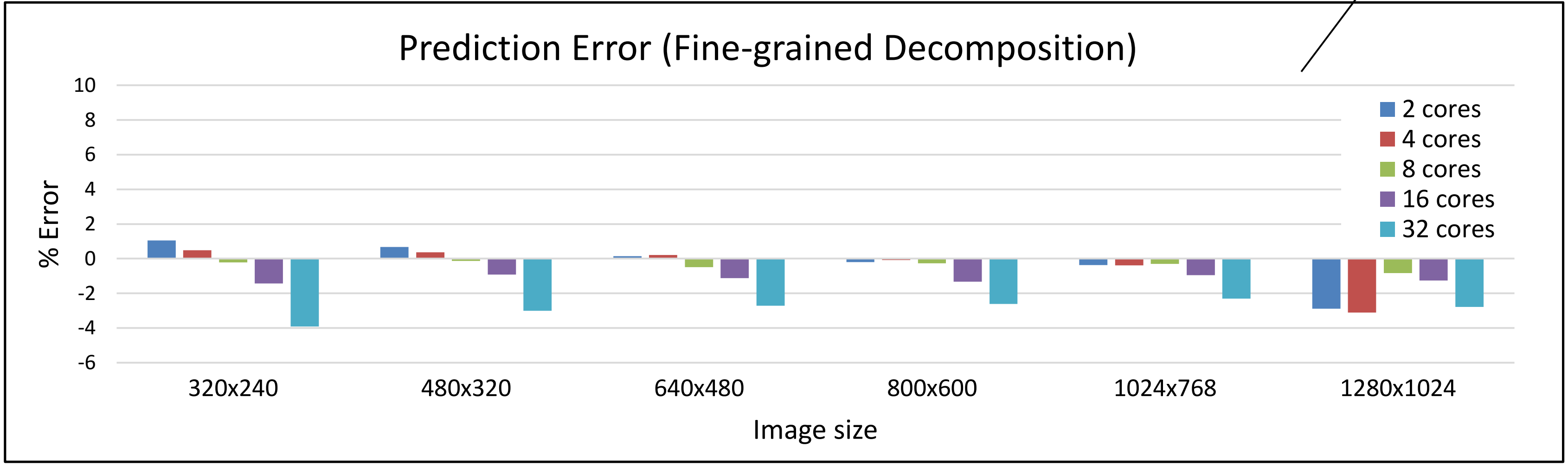
Simplation Setup:

- Calibration parameters: *Sobel gradient computation time per-pixel*
- Application: Row-decomposition of image, fixed filter size, & transfers over iMesh

Observations:

- Less than ±5% error for all tested image sizes
- Does not require coarse-grained models for computation

Fine-grained models provide fairly good accuracy in simulations



Raw data available in Appendix

Parallel Sobel Filtering

(Breakdown)

% Error in predicting different portions of kernel

	2 cores					4 cores				
Image size	Scatter	Compute_Gx	Compute_Gy	Gather	Total	Scatter	Compute_Gx	Compute_Gy	Gather	Total
320x240	-0.58	0.24	1.04	-4.11	1.05	-3.69	0.15	0.38	-4.18	0.48
480x320	-1.67	-0.16	0.64	-4.31	0.68	-3.78	0.03	0.17	-3.69	0.37
640x480	-2.13	0.02	-0.11	-4.72	0.15	-3.94	-0.19	-0.13	-3.97	0.22
800x600	-2.43	0.08	-0.65	-4.57	-0.20	-3.88	-0.30	-0.31	-4.77	-0.09
1024x768	-3.50	0.04	-0.83	-4.44	-0.37	-3.72	-0.39	-1.18	-4.05	-0.39
1280x1024	-4.23	-0.19	-5.69	-4.52	-2.88	-3.72	-0.49	-6.99	-3.93	-3.11

	8 cores					16 cores				
Image size	Scatter	Compute_Gx	Compute_Gy	Gather	Total	Scatter	Compute_Gx	Compute_Gy	Gather	Total
320x240	-4.63	0.16	0.09	-4.79	-0.21	-7.49	0.20	-0.16	-7.46	-1.42
480x320	-4.46	0.10	0.08	-3.58	-0.12	-5.93	-0.19	-0.08	-5.81	-0.92
640x480	-4.69	-0.12	-0.16	-3.62	-0.48	-5.53	-0.08	-0.33	-4.83	-1.11
800x600	-4.39	-0.30	-0.21	-3.64	-0.26	-5.31	-0.29	-0.41	-4.52	-1.33
1024x768	-4.25	-0.46	-0.29	-4.32	-0.30	-5.18	-0.50	-0.27	-4.39	-0.95
1280x1024	-4.11	-0.53	-2.42	-4.28	-0.83	-4.99	-0.63	-2.19	-5.49	-1.27

	32 cores				
Image size	Scatter	Compute_Gx	Compute_Gy	Gather	Total
320x240	-11.14	0.07	-0.13	-13.77	-3.91
480x320	-9.11	-0.03	-0.35	-9.00	-3.01
640x480	-7.61	-0.37	-0.86	-7.07	-2.71
800x600	-7.06	-0.34	-0.74	-6.81	-2.61
1024x768	-6.22	-0.61	-0.41	-5.97	-2.31
1280x1024	-5.98	-0.79	-1.90	-6.90	-2.78

Observations:

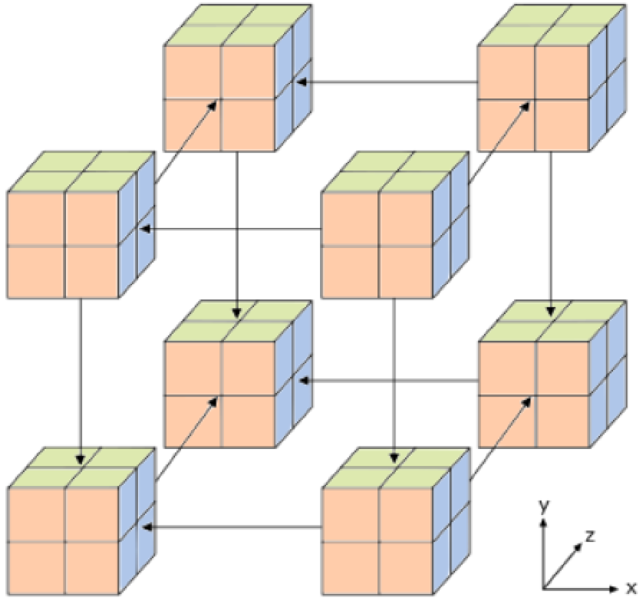
- Worst-case error: -3.91%
- Best-case error: -0.09%

More on CMT-nek SES Case Study

```
for ie=0 to ie = Nel
  for k=0 to N-1
    for j=0 to N-1
      for i=0 to N-1
        for i=0 to N-1
          dudr(i,j,k,ie) += a(i,l) x u(l,j,k,ie)
```

(a)

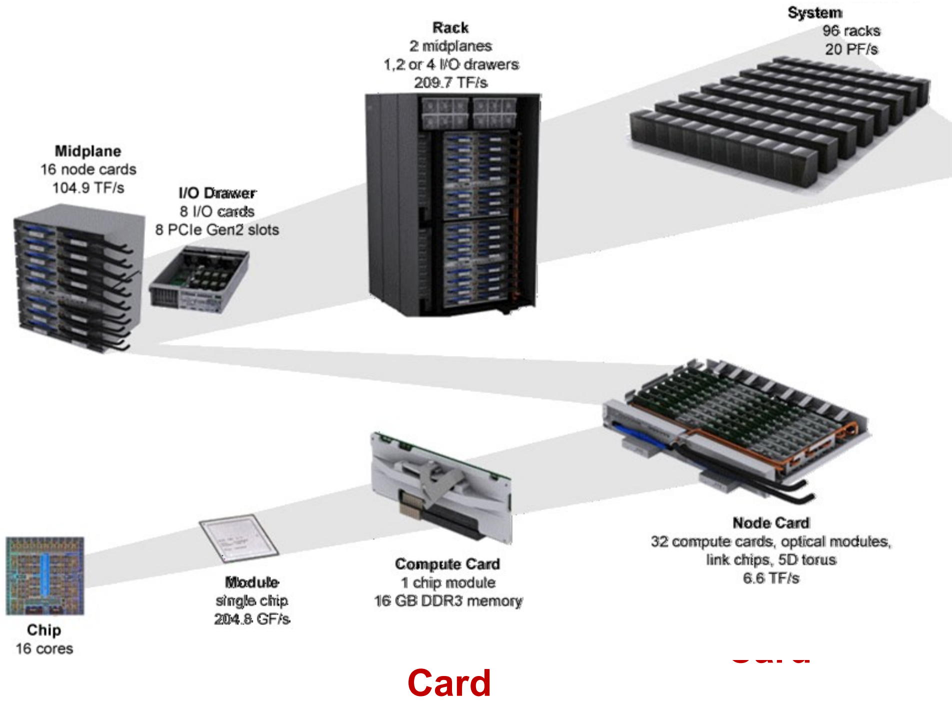
Surface data exchange between elements



```
VAR commgroup = 0:p-1
id_x = ID/(xmax+1)   //(xmax+1, ymax+1) is mesh size

// Distribute the data and operator matrices - dummy setup
m.broadcast(float, nwords_bcast, 0, commgroup);
m.barrier (ID);
m.scatter (float, nwords_scatter, 0, commgroup);
m.barrier (ID);

// Basic block for local derivative calculations
m.compute (N, Nel);

// Transfers from bottom to top of mesh. Odd numbered
// rows send to even numbered rows first and vice versa
if(id_x%2!=0){
 m.send(ID, ID-(xmax+1), nwords_update);
 if(id_x!=xmax) m.recv(ID+(xmax+1), ID, nwords_update);
}
else {
 if (id_x != xmax) recv(ID, ID+(xmax+1), nwords_update);
 if (id_x != 0) send(ID, ID-(xmax+1), nwords_update);
}

... // Similar transfers in three other directions of the mesh
```

Scaling Experiment on Vulcan: Architecture

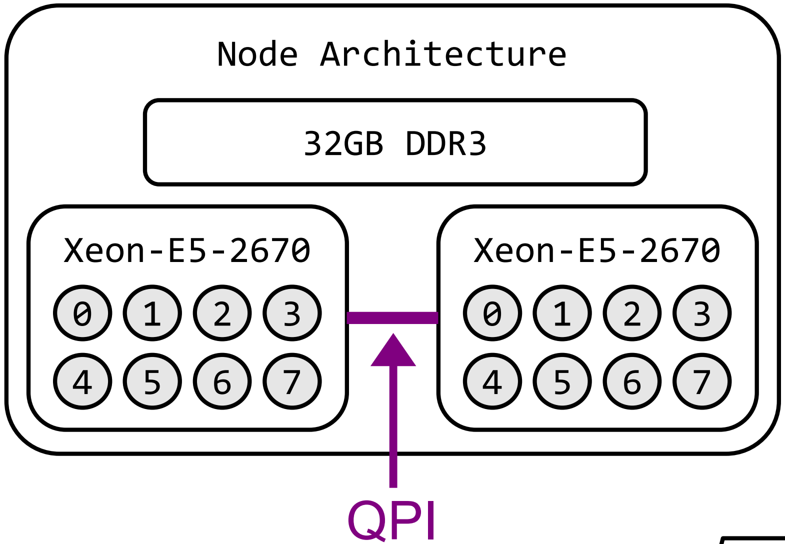
- Platform: Vulcan@LLNL
  - IBM BG/Q system
  - *24,576* nodes, *16* cores/node
  - *5D-torus* interconnect



- Vulcan is a very well-behave
  - Homogenous machine typically into *small or large blocks*
    - Large: Multiples of 512 nodes
    - Small: Multiples of 32 nodes
  - Within *a block network is isolated* and without interference

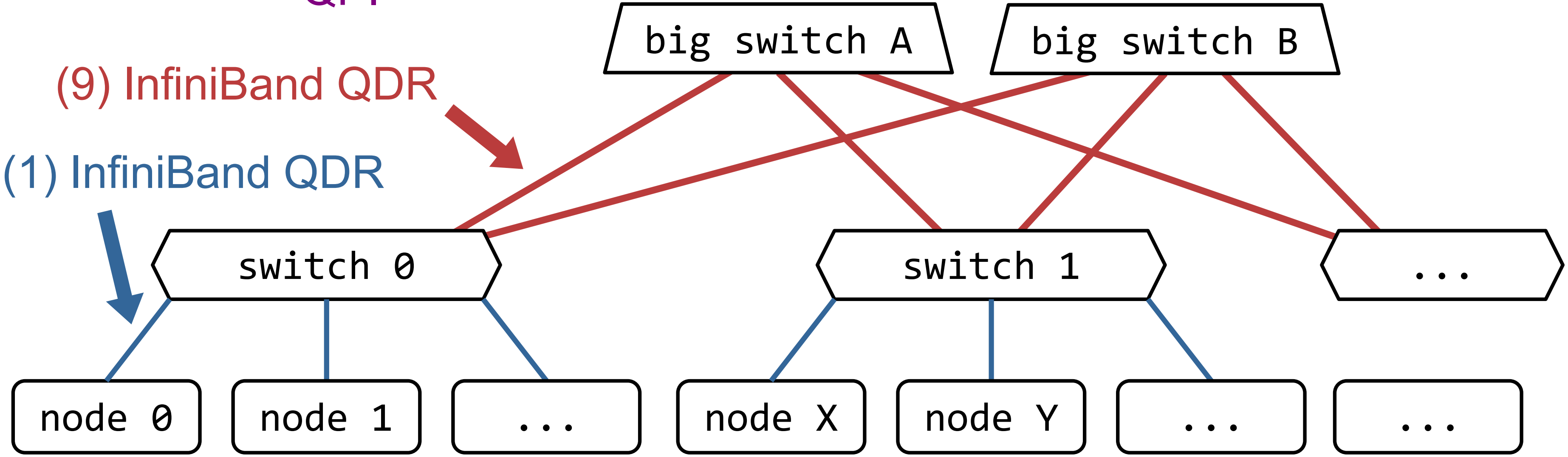
- Modeling method
  - Network is modeled as a *single switch* – simplifying assumption for Vulcan
    - Networking is a small portion of total application run-time
    - Not true for typical BE simulations
  - "Nodes" are *node cards* composed of 32 *compute cards*, each with 16 cores

Full-Scale Experiment: Architecture



Cab: Computing cluster at LLNL

- 1296 nodes, 40 TB memory, 2.6Ghz Cores
- Two-level switch InfiniBand QDR network
- Fat-tree-like layout
- Microsecond latencies



Full-Scale Experiment: Setup

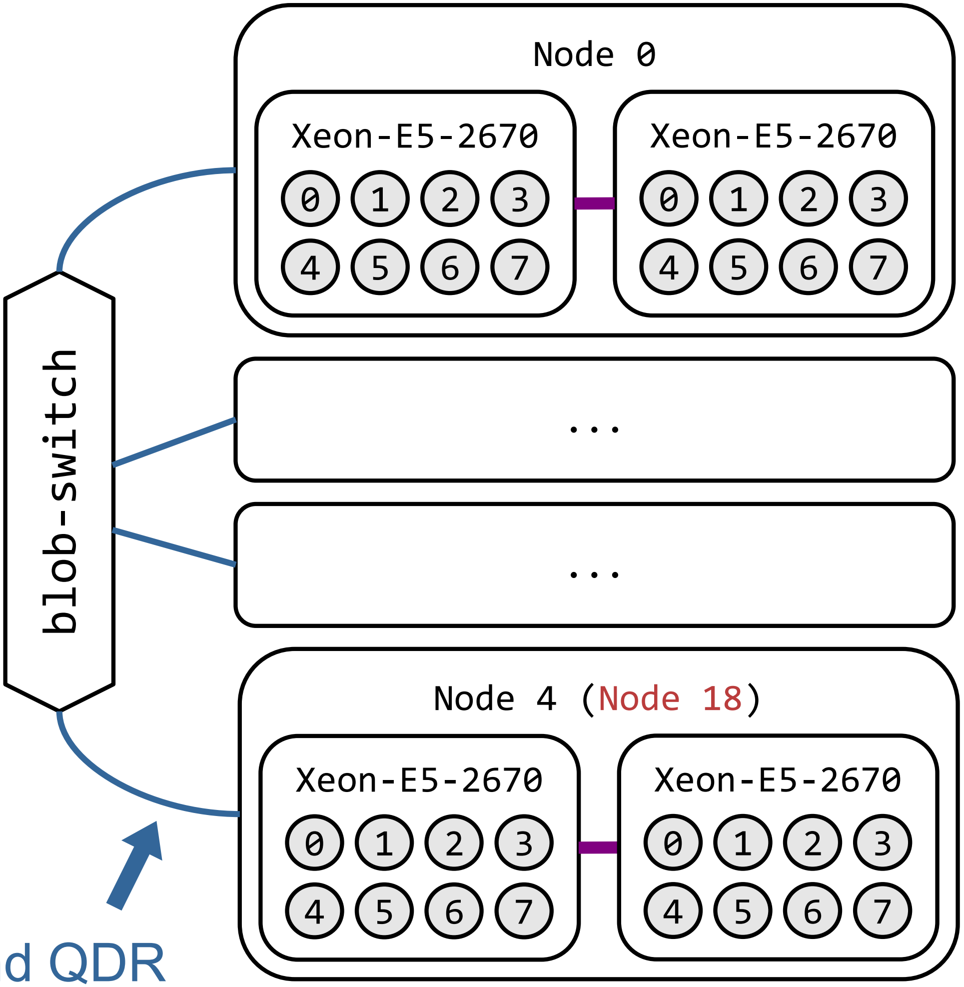
We simulate the test application on three different subsets of Cab.

The sizes of the modeled subsets are driven by 3D Cartesian mesh sizes:

- Tiny: 2^3 mesh (8 processes)
- Small: 4^3 mesh (64 processes)
- Medium: 6^3 mesh (216 processes)

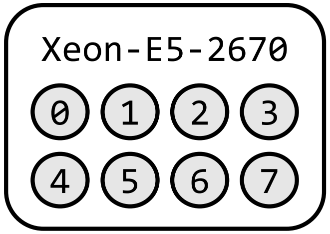
We then run the test application on the real Cab machine, and compare simulated versus real execution time.

Small (4^3), Medium (6^3) Tests:



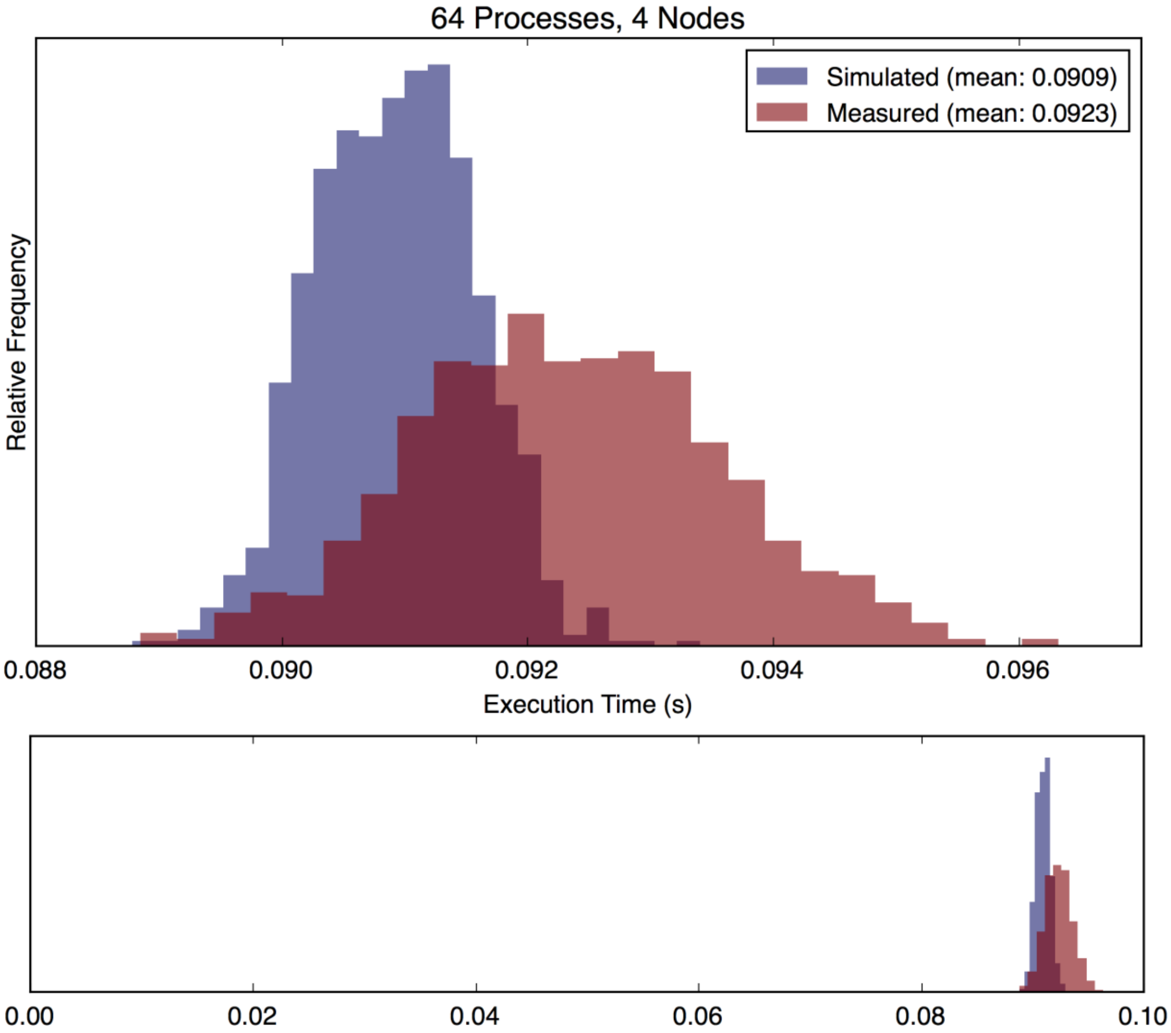
InfiniBand QDR

Tiny (2^3) Test:



Experiment Results: Accuracy (4^3)

Small Example: Comparison of simulated and real execution time (histogram of 1000 runs of each)

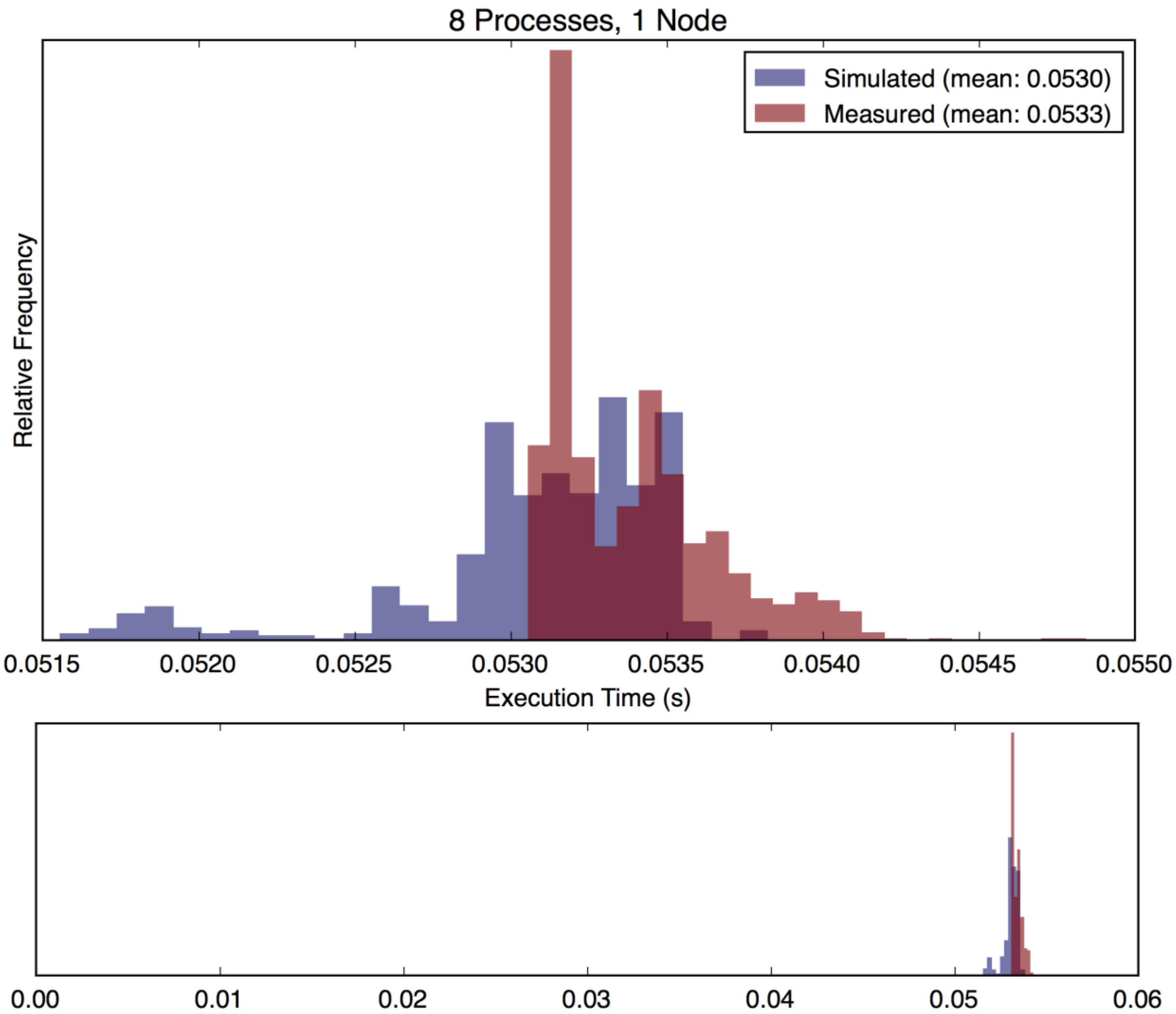


Observations:

- Mean error of roughly 1%
- Measured distribution is comparatively wide due to unrelated system load
- Measured distribution has higher mean due to unrelated system load
- Cab network appears to be well-characterized by a single-switch model

Experiment Results: Accuracy (2^3)

Tiny Example: Comparison of simulated and real execution time (histogram of 1000 runs of each)

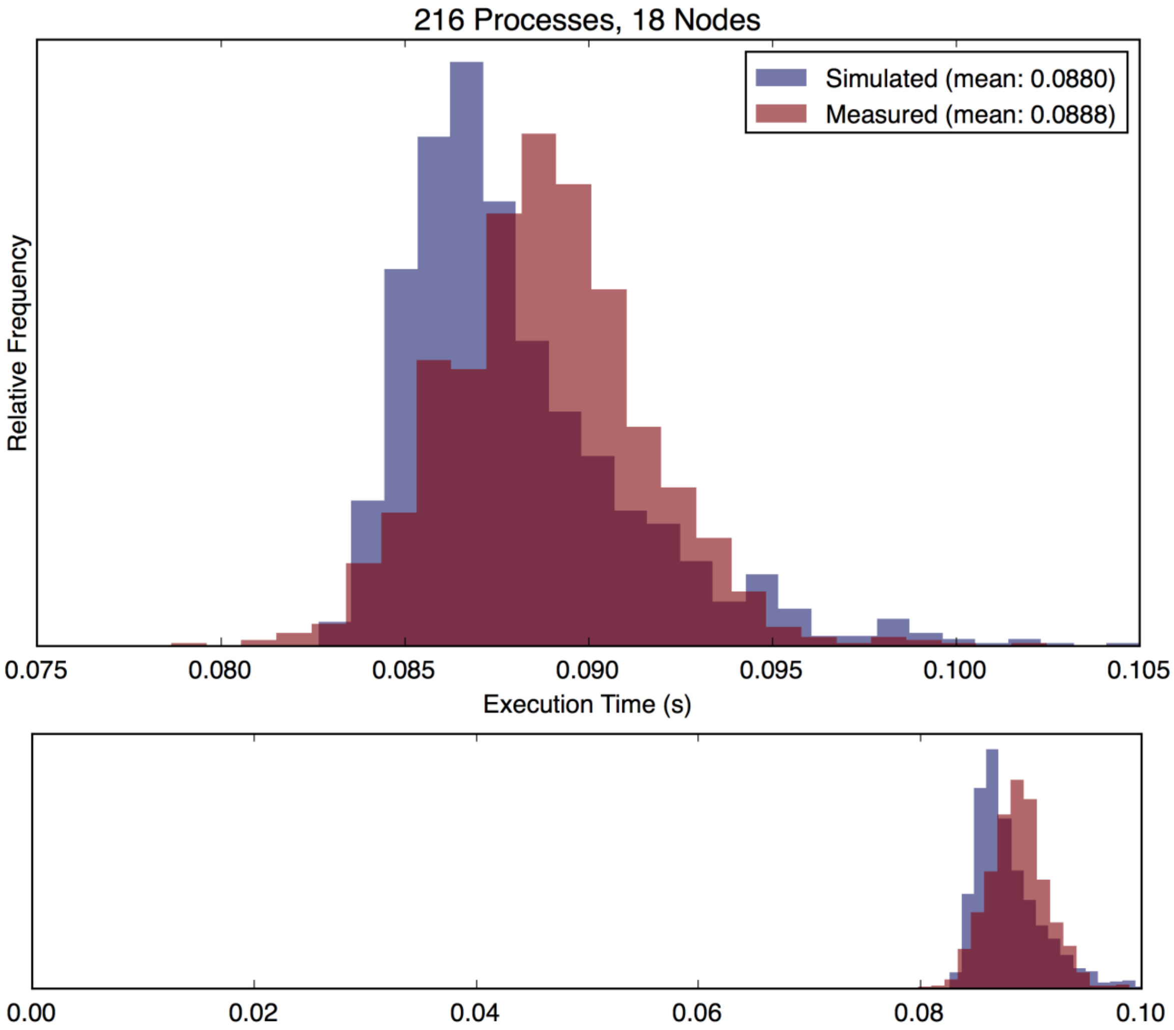


Observations:

- Mean error of roughly 1%
- Measured distribution has higher mean due to unrelated system load
- Assorted software and hardware state parameters affect result distributions
- Distribution is not well simulated, but we are not targeting network-less simulations

Experiment Results: Accuracy (6^3)

Medium Example: Comparison of simulated and real execution time (histogram of 1000 runs of each)



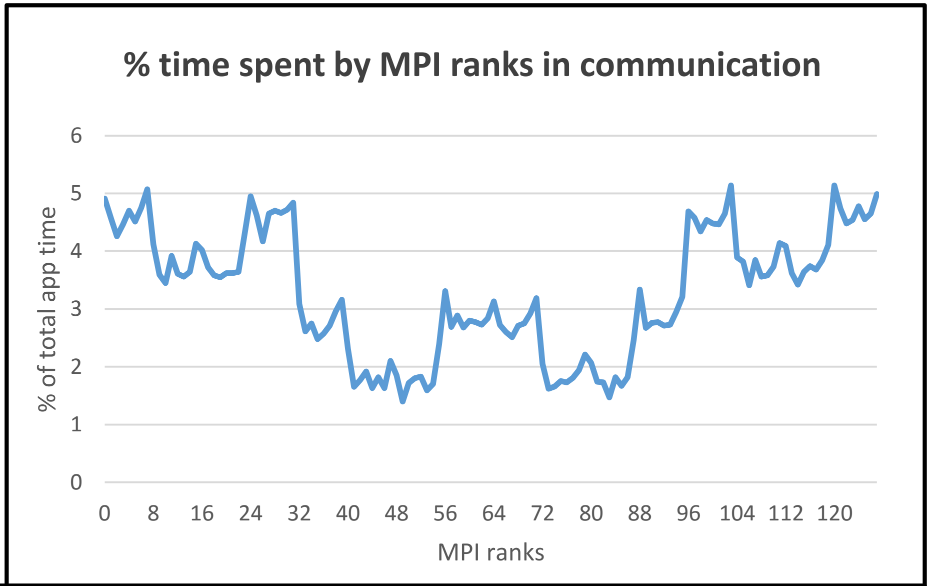
Observations:

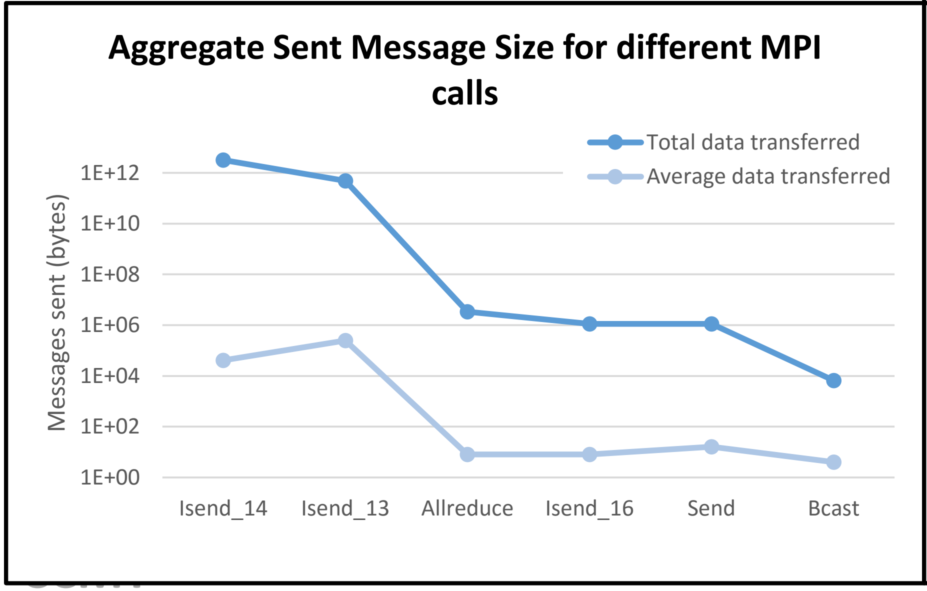
- Mean error of roughly 1%
- Measured distribution is comparatively wide due to unrelated system load
- Measured distribution has higher mean due to unrelated system load
- Network (compared to small example) is faster and less consistent

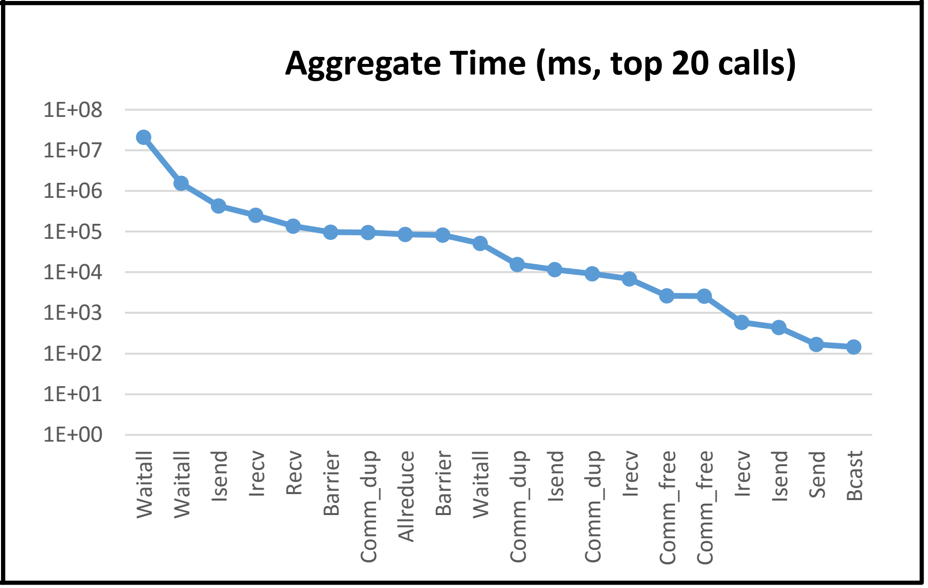
CMT-Bone MPI Profiling Data

- Experimental setup:
  - 128 MPI ranks, 1 rank/node
  - mpiP profiling data
  - Best-case, all exchanges across all MPI ranks occur in parallel

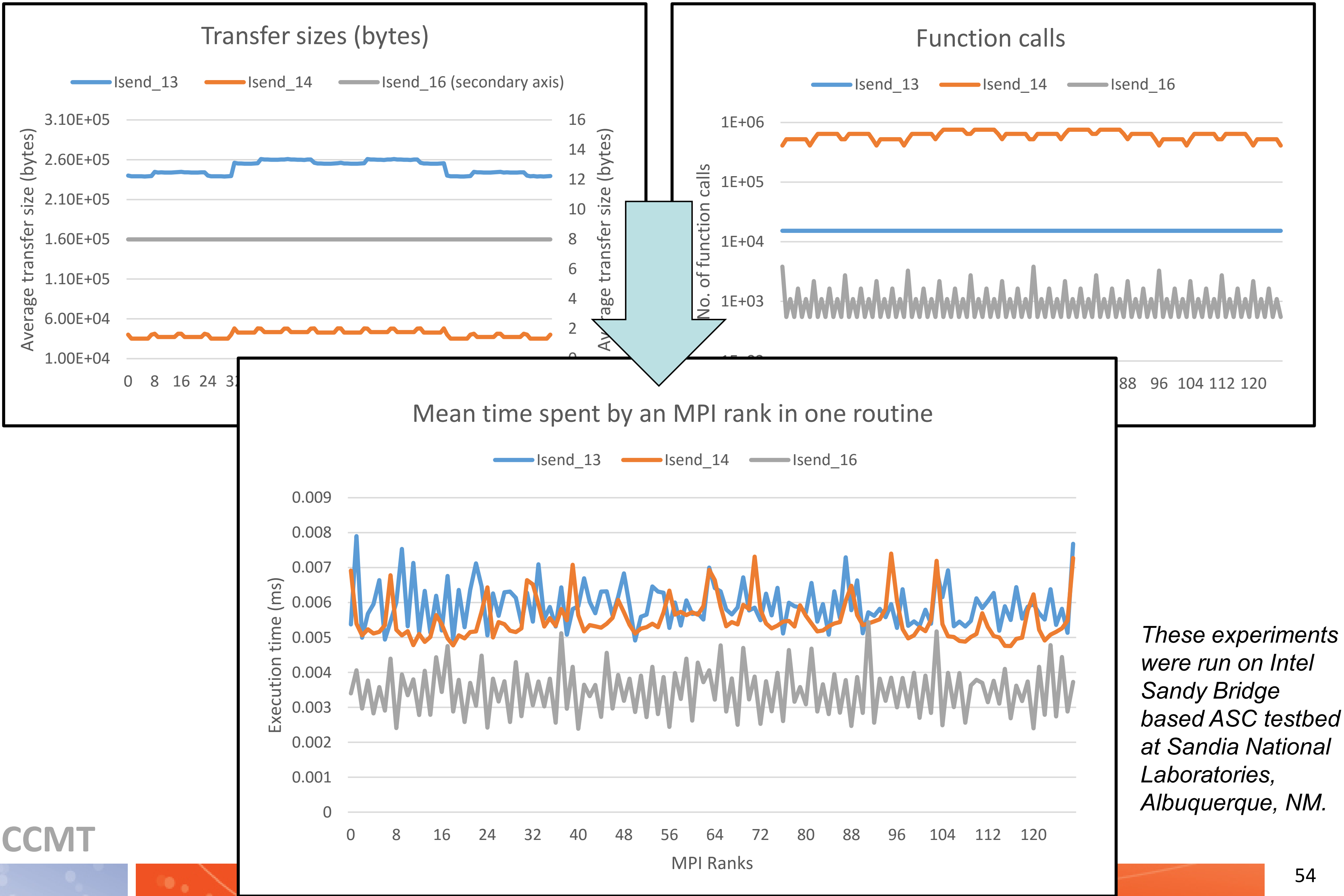
These experiments were run on Intel Sandy Bridge based ASC testbed at Sandia National Laboratories, Albuquerque, NM.


% time spent by MPI ranks in communication


Aggregate Sent Message Size for different MPI calls

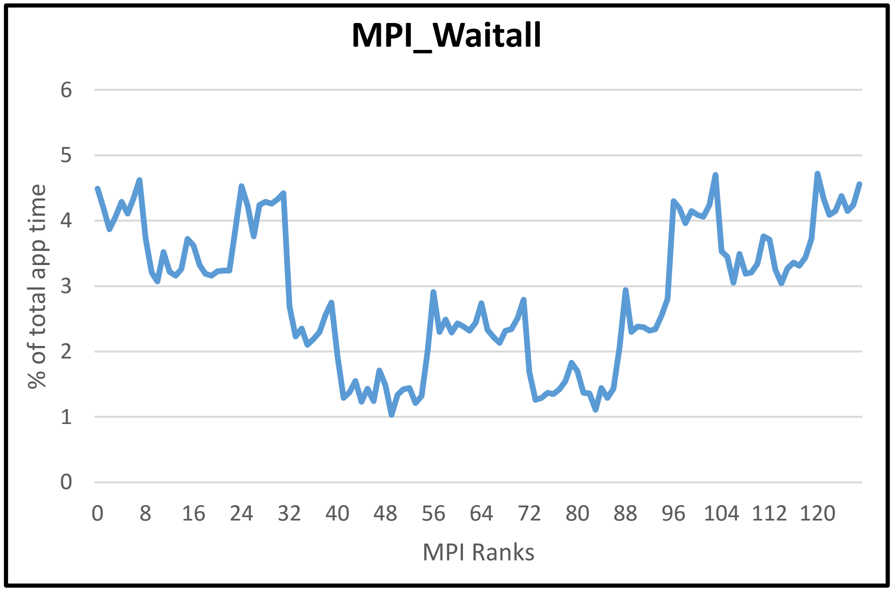

Aggregate Time (ms, top 20 calls)

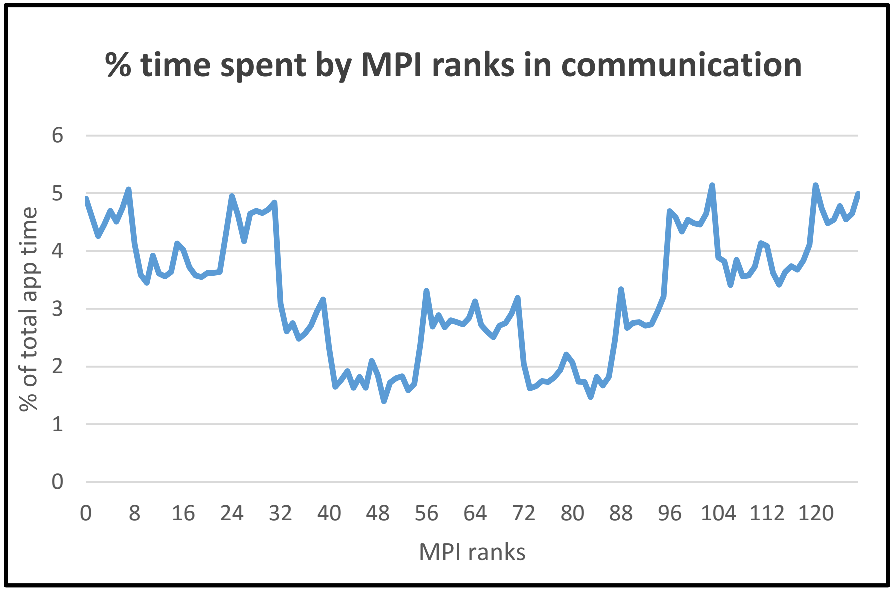
Data for Estimation of Transfer Times



These experiments were run on Intel Sandy Bridge based ASC testbed at Sandia National Laboratories, Albuquerque, NM.

Overall Communication Time Estimation





These experiments were run on Intel Sandy Bridge based ASC testbed at Sandia National Laboratories, Albuquerque, NM.

- Most of the time is spent in MPI_Waitall
  - Need timed simulations to look at these effects
  - It may still be possible to use coarse models for actual transfer time estimations

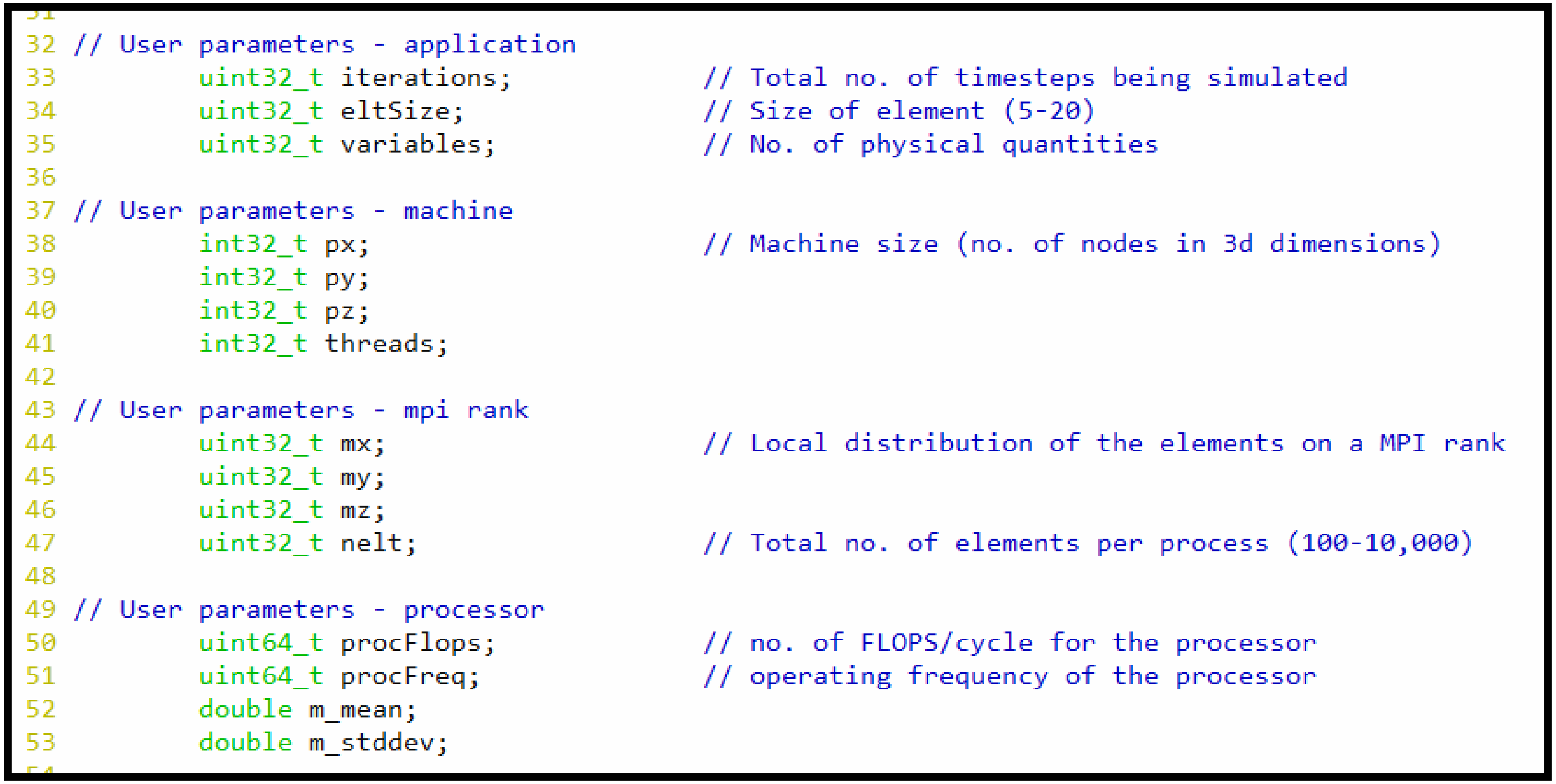
Sandia National Laboratories

Application Modeling in SST (Motifs)

- Motifs are coarse-grained representations of app behavior, similar to AppBEOs, that capture interactions between network endpoints
  - Look very much like an MPI program (serial flow)
  - Network endpoints can be cores, devices, nodes, etc.
  - Compute blocks or local operations are delay blocks used to pace the simulation similar to our ProcBEOs
- Ember contains motifs for several commonly used comm. patterns
  - e.g., halo exchanges, MPI collectives, sweeps, etc.
  - We extended motifs library by adding models for CMT-nek comm routines

CMT-bone Simulations using SST (1 of 5)

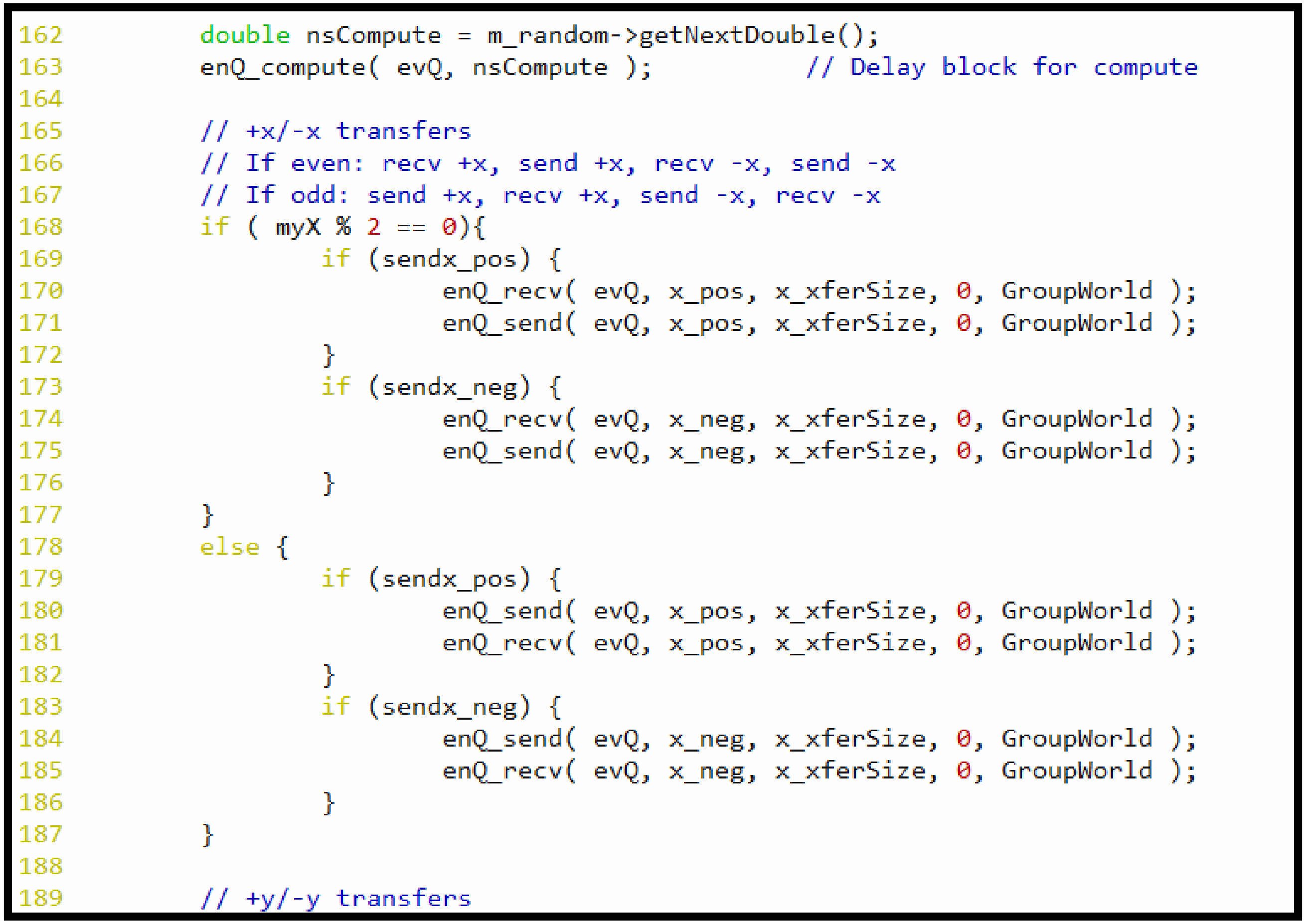
- For simulations we need:
  1. Motif/abstract application description for CMT-bone
  2. Modeling parameters to describe system
  3. SST configuration file specifying motif parameters



```
// User parameters - application
        uint32_t iterations;            // Total no. of timesteps being simulated
        uint32_t eltSize;               // Size of element (5-20)
        uint32_t variables;             // No. of physical quantities

// User parameters - machine
        int32_t px;                     // Machine size (no. of nodes in 3d dimensions)
        int32_t py;
        int32_t pz;
        int32_t threads;

// User parameters - mpi rank
        uint32_t mx;                    // Local distribution of the elements on a MPI rank
        uint32_t my;
        uint32_t mz;
        uint32_t nelt;                  // Total no. of elements per process (100-10,000)

// User parameters - processor
        uint64_t procFlops;             // no. of FLOPS/cycle for the processor
        uint64_t procFreq;              // operating frequency of the processor
        double m_mean;
        double m_stddev;
```

CMT-bone Simulations using SST (2 of 5)

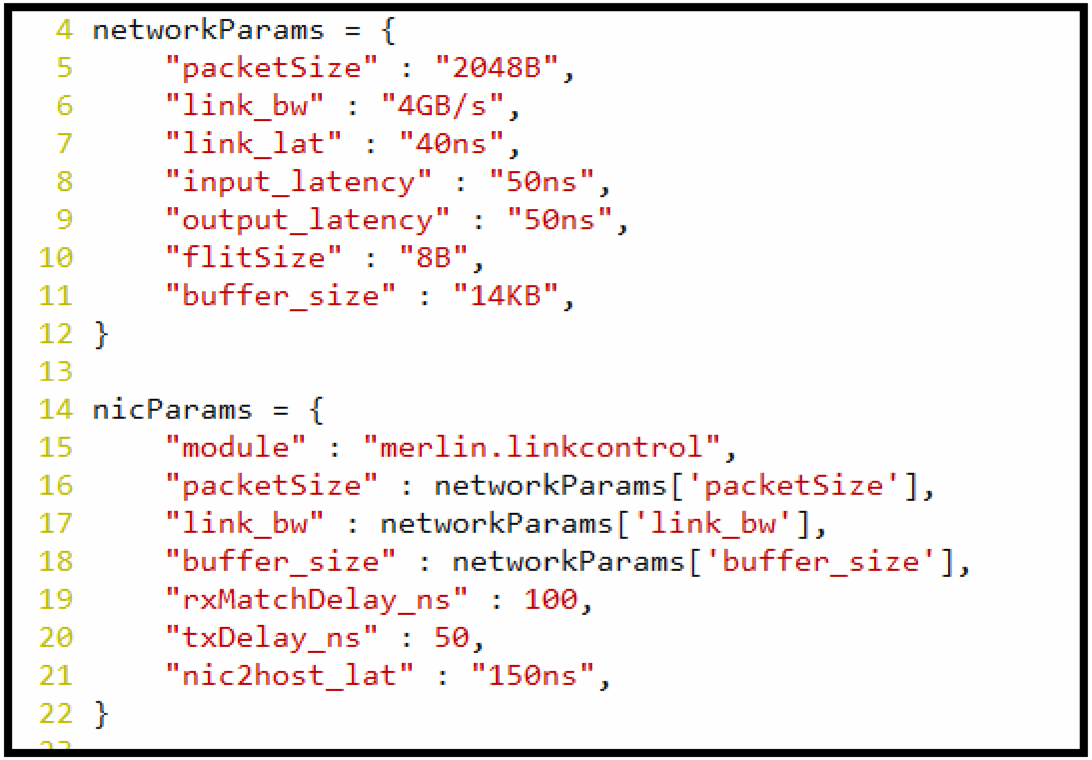
- For simulations we need:
  1. Motif/abstract application description for CMT-bone



```
double nsCompute = m_random->getNextDouble();
enQ_compute( evQ, nsCompute );          // Delay block for compute

// +x/-x transfers
// If even: recv +x, send +x, recv -x, send -x
// If odd: send +x, recv +x, send -x, recv -x
if ( myX % 2 == 0){
        if (sendx_pos) {
                enQ_recv( evQ, x_pos, x_xferSize, 0, GroupWorld );
                enQ_send( evQ, x_pos, x_xferSize, 0, GroupWorld );
        }
        if (sendx_neg) {
                enQ_recv( evQ, x_neg, x_xferSize, 0, GroupWorld );
                enQ_send( evQ, x_neg, x_xferSize, 0, GroupWorld );
        }
}
else {
        if (sendx_pos) {
                enQ_send( evQ, x_pos, x_xferSize, 0, GroupWorld );
                enQ_recv( evQ, x_pos, x_xferSize, 0, GroupWorld );
        }
        if (sendx_neg) {
                enQ_send( evQ, x_neg, x_xferSize, 0, GroupWorld );
                enQ_recv( evQ, x_neg, x_xferSize, 0, GroupWorld );
        }
}

// +y/-y transfers
```

CMT-bone Simulations using SST (3 of 5)

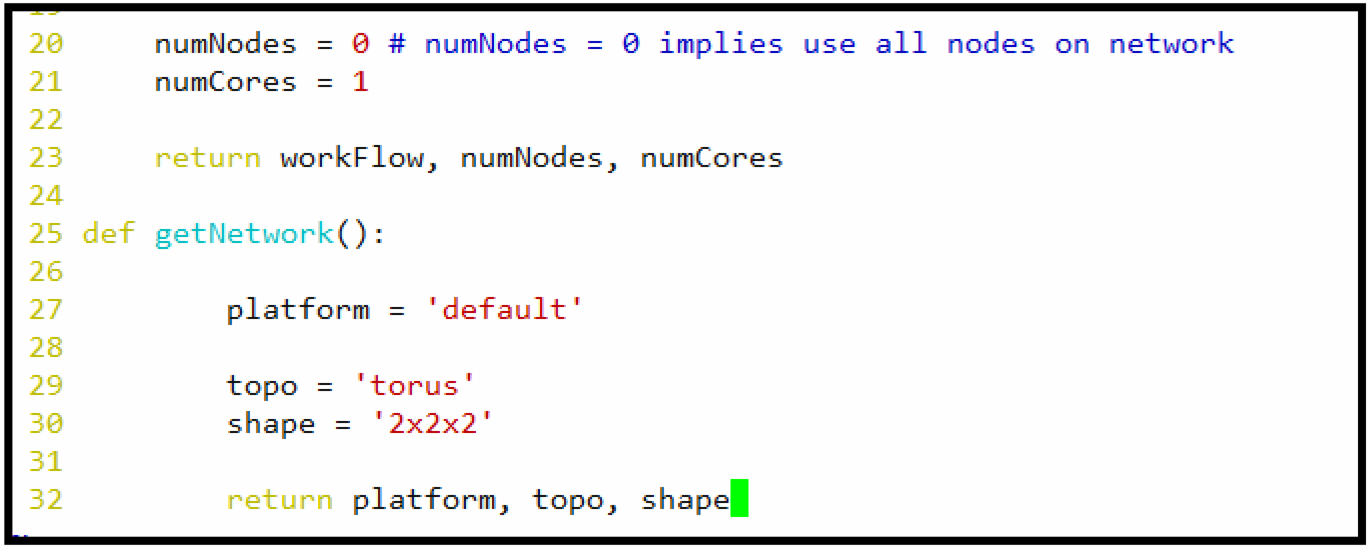
- For simulations we need:
  1. Motif/abstract application description for CMT-bone
  2. Modeling parameters to describe network
  3. SST configuration file specifying motif parameters



```
networkParams = {
    "packetSize" : "2048B",
    "link_bw" : "4GB/s",
    "link_lat" : "40ns",
    "input_latency" : "50ns",
    "output_latency" : "50ns",
    "flitSize" : "8B",
    "buffer_size" : "14KB",
}

nicParams = {
    "module" : "merlin.linkcontrol",
    "packetSize" : networkParams['packetSize'],
    "link_bw" : networkParams['link_bw'],
    "buffer_size" : networkParams['buffer_size'],
    "rxMatchDelay_ns" : 100,
    "txDelay_ns" : 50,
    "nic2host_lat" : "150ns",
}
```

CMT-bone Simulations using SST (4 of 5)

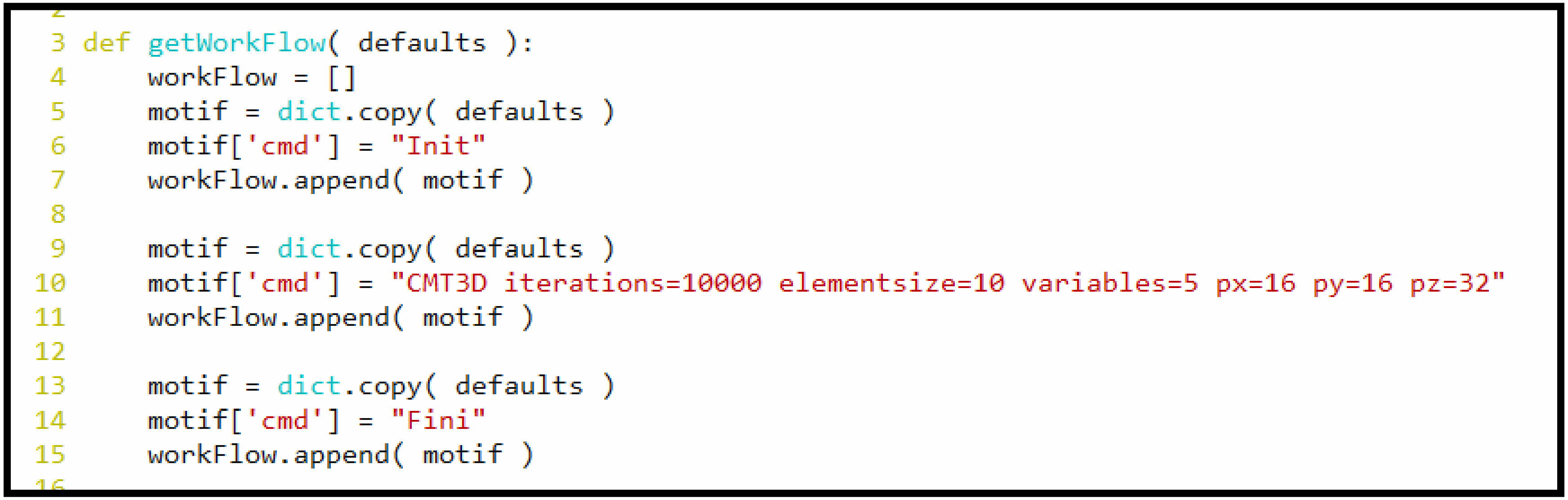
- For simulations we need:
  1. Motif/abstract application description for CMT-bone
  2. Modeling parameters to describe network
  3. SST configuration file specifying motif parameters



```
    numNodes = 0 # numNodes = 0 implies use all nodes on network
    numCores = 1

    return workFlow, numNodes, numCores

def getNetwork():

        platform = 'default'

        topo = 'torus'
        shape = '2x2x2'

        return platform, topo, shape
```

CMT-bone Simulations using SST (5 of 5)

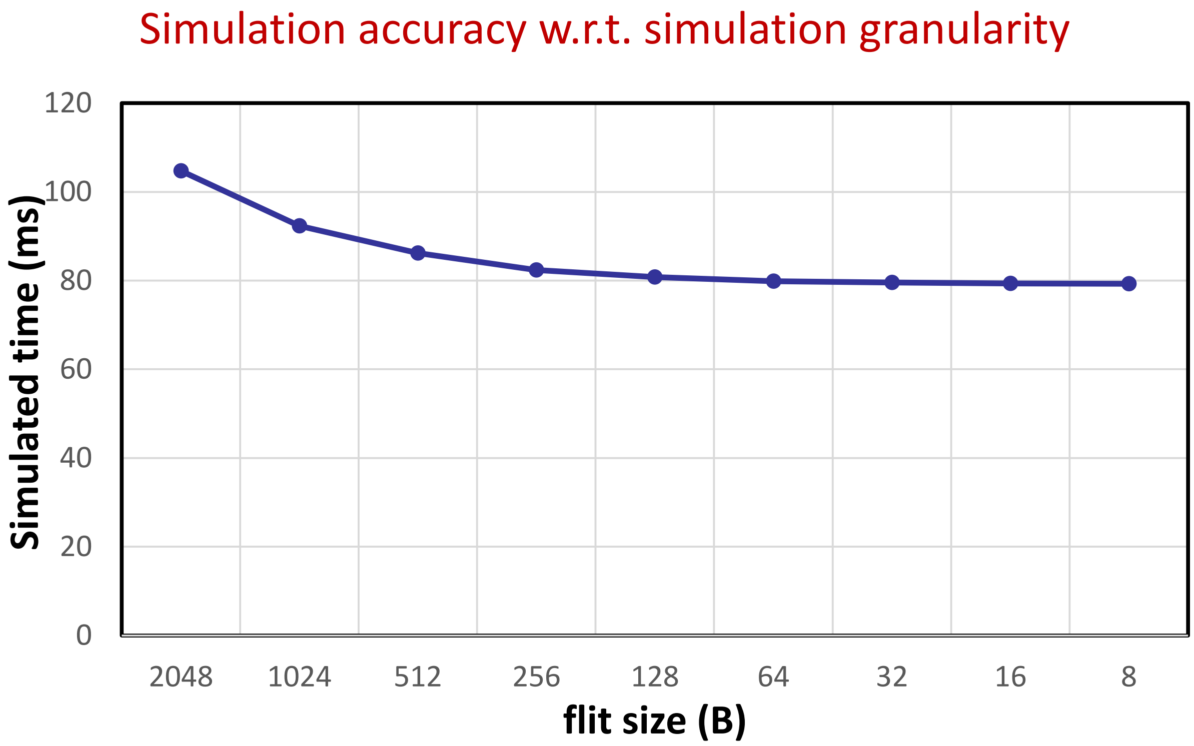
- For simulations we need:
  1. Motif/abstract application description for CMT-bone
  2. Modeling parameters to describe network
  3. Ember configuration file specifying motif parameters



```
def getWorkFlow( defaults ):
    workFlow = []
    motif = dict.copy( defaults )
    motif['cmd'] = "Init"
    workFlow.append( motif )

    motif = dict.copy( defaults )
    motif['cmd'] = "CMT3D iterations=10000 elementsize=10 variables=5 px=16 py=16 pz=32"
    workFlow.append( motif )

    motif = dict.copy( defaults )
    motif['cmd'] = "Fini"
    workFlow.append( motif )
```

Sensitivity to Model Parameters

- Estimating effect of granularity on simulation accuracy

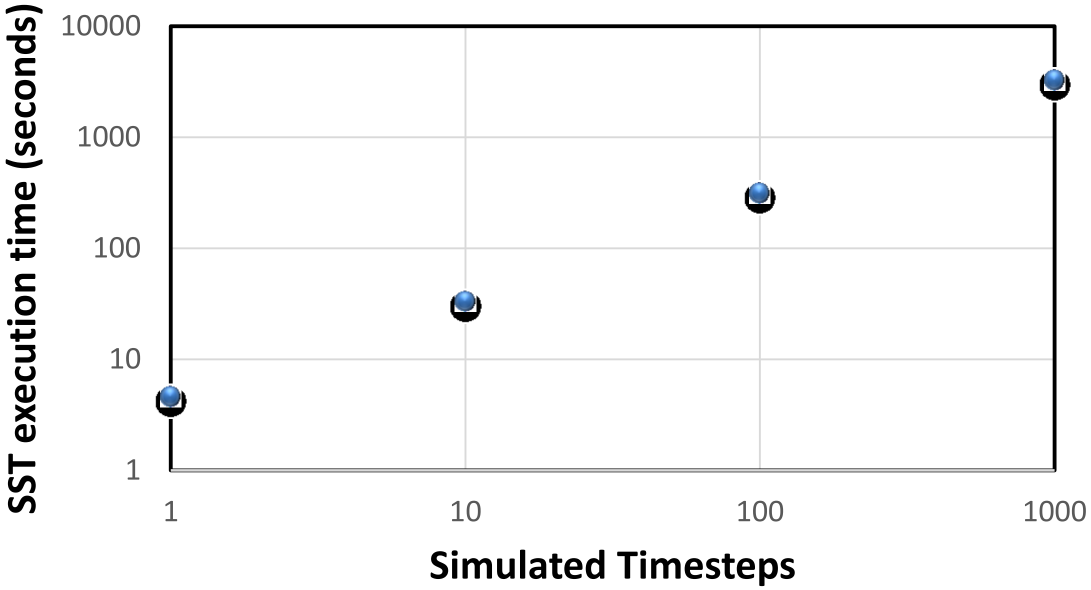


- **Application setup**: element size=10, iterations=1000
- **Machine setup**: 8x8x8 3D torus, pkt size=2048 B
- **Observations:** As flit size approaches pkt size, simulation estimations become increasingly more inaccurate (~30%)

Scaling SST Simulations

- Speed of SST simulations as size of application grows

Single-threaded SST simulations

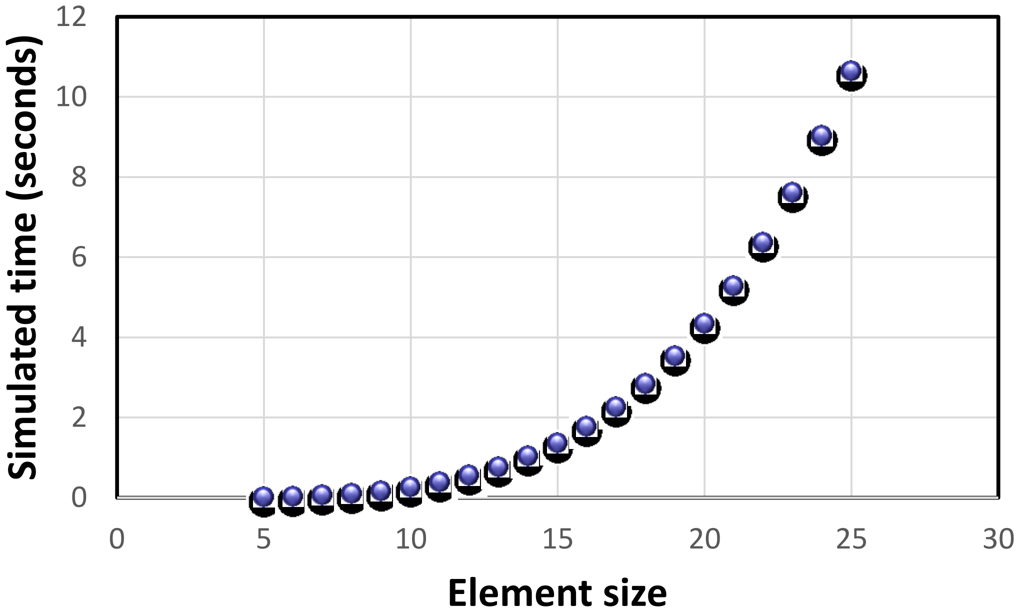


- **Application setup:** 1000 elements/processor, element size=10
- **Machine setup:** 512 nodes (8x8x8 torus), bw= 4GB/s ,pkt size= 2048B, flit size = 8B
- **Observations:** SST execution time increases linearly with an increase in problem size

Design-Space Exploration (1 of 3)

- Effect of varying element size on application execution time

Parameter = Element size

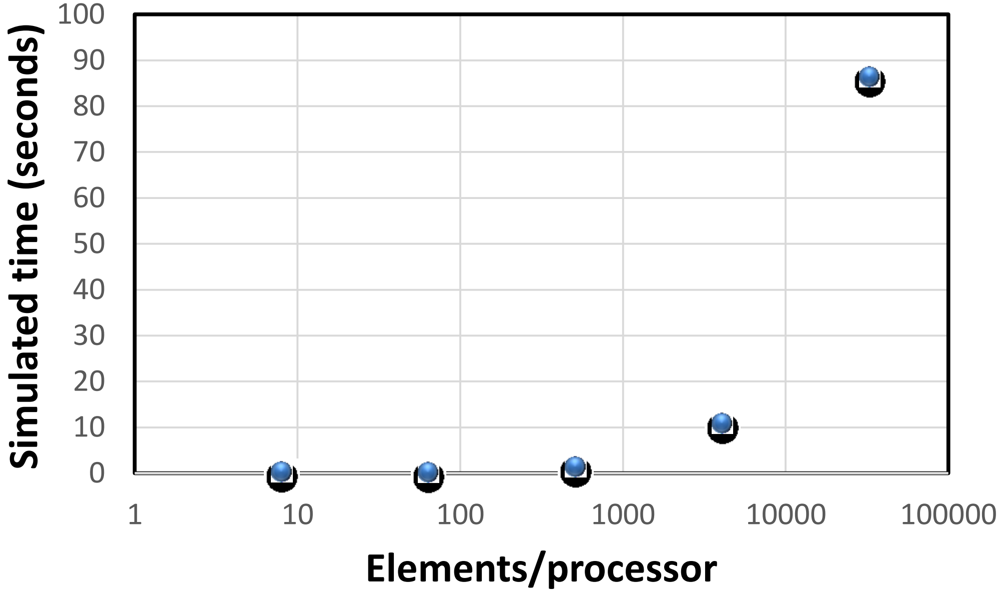


- **Application setup:** 1000 elements/process, 1000 timesteps (iterations)
- **System setup:** 4x4x4 torus with 1 process per node, bw=4GB/s, pkt size=2048B, flit size=8B
- **Observations:** As expected, app execution time (estimated) increases exponentially with increase in element size

Design-Space Exploration (2 of 3)

- Effect of varying elements on application execution time

Parameter = Elements/processor

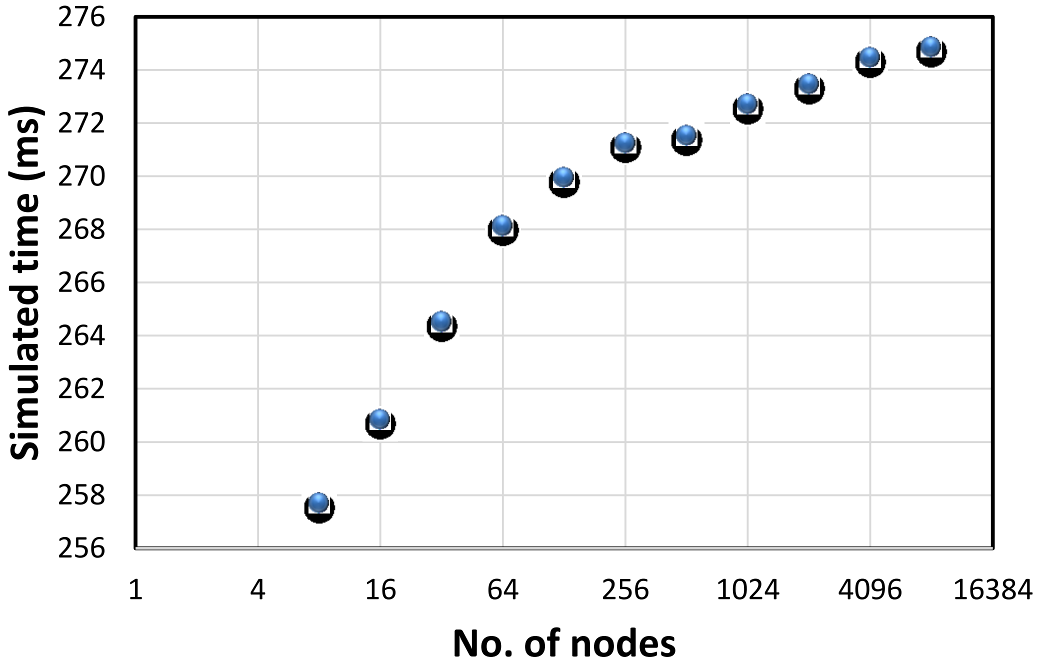


- **Application setup:** element size=10, 1000 timesteps (iterations)
- **System setup:** 4x4x4 torus with 1 process per node, bw=4GB/s, pkt size=2048B, flit size=8B
- **Observation:** Execution time increases almost linearly with an increase in processor load. Computation is the major contributor to this increase.

Design-Space Exploration (3 of 3)

- Weak scaling

parameter = machine size & problem size



- **Application setup:** element size=10, 100 timesteps (iterations)
- **System setup:** 3d torus with 1 process per node, bw=4GB/s, pkt size=2048B, flit size=8B
- **Observation:** As problem size and system size increase, the amount of computation per processor remains the same. Communication time grows fast in the beginning before stabilizing.