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
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 diagram](image)
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
- 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)

- 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

Co-Design Using Behavioral Emulation

**BEO design and calibration**
- Application source code
  - manual / automated
  - Instrumented source code
- Existing machines OR Fine-grained simulators
  - benchmarking
- Calibration data

**Simulation validation**
- Simulation results
  - validation
- BE Simulation
- AppBEO (application description)

**Coarse-grained Simulation Platforms**
- Discrete event simulation framework
- Custom SW simulator
- FPGA Acceleration
- ...

**BEO design and calibration**
- AppBEO (application description)

**HW/ SW co-design**
- Alternate algorithms
- Algorithmic DSE
  - iterative
- Simulation predictions
- Notional architectures
- Architecture DSE

**BEO – Behavioral Emulation Object**

* UQ team
* CS team
* CMT-nek team
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)**

```plaintext
// Define group as nodes 0-3
VAR commGrp=0:3
// Broadcast matrix A
(d dataSize=64*64/2) to group
Bcast(int32,2048,0,commGrp)
// Barrier sync
Barrier(commGrp)
// Scatter 1/4 of matrix B
(d 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
(d dataSize=(64*64)/(4*2))
Gather(int32,512,commGrp)
Done
```

**Intermediate format**

```plaintext
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)**

```plaintext
// 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)

*Spectral Element Solver*
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

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

<table>
<thead>
<tr>
<th>data size</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>487.47</td>
</tr>
<tr>
<td>16</td>
<td>917.48</td>
</tr>
<tr>
<td>32</td>
<td>1,781.68</td>
</tr>
<tr>
<td>64</td>
<td>3,509.27</td>
</tr>
<tr>
<td>128</td>
<td>6,965.78</td>
</tr>
<tr>
<td>256</td>
<td>13,877.84</td>
</tr>
<tr>
<td>512</td>
<td>27,703.63</td>
</tr>
<tr>
<td>1024</td>
<td>55,401.93</td>
</tr>
</tbody>
</table>

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

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

Dot product (int32) and Loop Overhead

2D Matrix Multiply
(MxN and NxN)
Example: CommBEO for iMesh

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

### Network configuration parameters for TILE-Gx36 iMesh

- **Topology**: 2D mesh
- **Routing policy**: dim-order
- **Routing policy**: cut-through
- **X-dir latency**: testbed data
- **Y-dir latency**: testbed data
- **Arbitration**: round-robin
  
### Pseudo-code for CommBEO

```plaintext
if (input_buffer!=empty) {
  read_event;
  if(output_buffer!=full) {
    forward(x_dir, y_dir);
  }
}
```

### iMesh one-way latencies and throughput

<table>
<thead>
<tr>
<th>Direction</th>
<th>Time (ns)</th>
<th>Throughput (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighbors</td>
<td>20.5</td>
<td>3,117.355</td>
</tr>
<tr>
<td>Side-to-Side</td>
<td>24.5</td>
<td>2,608.717</td>
</tr>
<tr>
<td>Corners</td>
<td>30</td>
<td>2,129.44</td>
</tr>
</tbody>
</table>

### Switching time

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

<table>
<thead>
<tr>
<th>Direction</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-x</td>
<td>1</td>
</tr>
<tr>
<td>y-y</td>
<td>1</td>
</tr>
<tr>
<td>x-y</td>
<td>1</td>
</tr>
</tbody>
</table>

Hop Time: 1ns

![Graph showing round-trip latency and switching time]
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

---

**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 Diagram]

```c
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);
}
...
```

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

*Spectral Element Solver*
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
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

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

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

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

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

![Graphs showing execution time predictions and discrepancies](attachment:graphs.png)

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

Element size:
- 15
- 9
- 5

Text: Discrepancy %
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

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

![Graphs showing predictions for various element sizes using Linear Interpolation and Polynomial Interpolation.](image)
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
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

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

References

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

Object-oriented System Modeling
References

Hardware Emulation


Supercomputer-specific Modeling & Simulation


Analytical Modeling

APPENDIX
Emulation Output

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

<table>
<thead>
<tr>
<th>Metrics of interest</th>
<th>Management Plane (end of simulation)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>procBEO</strong></td>
<td>1: Num Sends: 7</td>
</tr>
<tr>
<td>Total no. of Instr</td>
<td>1: Num Computes: 1</td>
</tr>
<tr>
<td>No. of Instr of each types</td>
<td>1: Num Recvs: 7</td>
</tr>
<tr>
<td>Total amount of data sent</td>
<td>1: Total Instructions: 15</td>
</tr>
<tr>
<td>Total amount of data received</td>
<td>1: Total Time: 3.965329877E9</td>
</tr>
<tr>
<td>Total Execution time</td>
<td>1: Compute Time: 3.72834304E9</td>
</tr>
<tr>
<td>Execution Time/Instr</td>
<td>1: Time Per Instruction: 2.6435532513333E8</td>
</tr>
<tr>
<td>Total computation time</td>
<td>1: Total Packets Sent: 2195456</td>
</tr>
<tr>
<td>Total communication time</td>
<td>1: Total Packets Recv: 2195456</td>
</tr>
<tr>
<td>Waiting time (on comm)</td>
<td>1: Total Communication Time: 7.0</td>
</tr>
<tr>
<td>Idle time</td>
<td>1: Total Wait Time: 1.67160739E8</td>
</tr>
<tr>
<td></td>
<td>1: Total Idle Time: 6.9826091E7</td>
</tr>
<tr>
<td><strong>commBEO</strong></td>
<td>2: Num Sends: 7</td>
</tr>
<tr>
<td>Total data transferred/No.of packets</td>
<td>2: Num Computes: 1</td>
</tr>
<tr>
<td>Link utilization</td>
<td>2: Num Recvs: 7</td>
</tr>
<tr>
<td>Buffer utilization</td>
<td>2: Total Instructions: 15</td>
</tr>
<tr>
<td>Idle time</td>
<td>2: Total Time: 3.967446028E9</td>
</tr>
<tr>
<td>No. of packets dropped</td>
<td>2: Compute Time: 3.72834304E9</td>
</tr>
<tr>
<td>Average distance</td>
<td>2: Time Per Instruction: 2.6449540186667E8</td>
</tr>
<tr>
<td></td>
<td>2: Total Packets Sent: 2195456</td>
</tr>
<tr>
<td></td>
<td>2: Total Packets Recv: 2195456</td>
</tr>
<tr>
<td></td>
<td>2: Total Communication Time: 10.0</td>
</tr>
<tr>
<td></td>
<td>2: Total Wait Time: 1.6927689E8</td>
</tr>
<tr>
<td></td>
<td>2: Total Idle Time: 6.9826088E7</td>
</tr>
</tbody>
</table>
Granularity of problem decomposition has significant effect on accuracy.

Overhead is amortized with increase in problem size.

Fine-grained model provides desirable accuracy for this algorithm.

**Compute Microbenchmarks**

**Prediction Error in single-core Matrix Multiply**

- **Testbed**
- **Sim (Fine-grain)**
- **Sim (Coarse-grain)**
- **Error (Fine-grain)**
- **Error (Coarse-grain)**

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

- **Testbed**
- **Simulation**
- **% Error**

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

- **Testbed**
- **Simulation**
- **% Error**
Parallel 2D Matrix Multiply  
(Breakdown: Fine-grained compute model)

% Error in predicting different portions of kernel

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

<table>
<thead>
<tr>
<th>matrix size</th>
<th>Bcast</th>
<th>Scatter</th>
<th>Compute</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 cores</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64x64</td>
<td>-1.92</td>
<td>-3.35</td>
<td>18.79</td>
<td>-2.47</td>
<td>12.71</td>
</tr>
<tr>
<td>128x128</td>
<td>-2.61</td>
<td>-0.52</td>
<td>9.73</td>
<td>-2.70</td>
<td>7.42</td>
</tr>
<tr>
<td>256x256</td>
<td>-3.10</td>
<td>-2.91</td>
<td>5.05</td>
<td>-2.55</td>
<td>3.85</td>
</tr>
<tr>
<td>512x512</td>
<td>-4.28</td>
<td>-5.14</td>
<td>2.45</td>
<td>-3.10</td>
<td>1.57</td>
</tr>
<tr>
<td>1024x1024</td>
<td>-5.67</td>
<td>-8.77</td>
<td>1.28</td>
<td>-5.34</td>
<td>0.57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>matrix size</th>
<th>Bcast</th>
<th>Scatter</th>
<th>Compute</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 cores</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64x64</td>
<td>-1.52</td>
<td>-3.83</td>
<td>18.65</td>
<td>12.07</td>
<td>6.56</td>
</tr>
<tr>
<td>128x128</td>
<td>-2.72</td>
<td>-2.05</td>
<td>9.36</td>
<td>4.40</td>
<td>4.96</td>
</tr>
<tr>
<td>256x256</td>
<td>-3.04</td>
<td>-2.66</td>
<td>4.90</td>
<td>2.34</td>
<td>2.20</td>
</tr>
<tr>
<td>512x512</td>
<td>-4.04</td>
<td>-5.55</td>
<td>2.34</td>
<td>-2.74</td>
<td>1.06</td>
</tr>
<tr>
<td>1024x1024</td>
<td>-6.81</td>
<td>-12.21</td>
<td>1.18</td>
<td>-4.70</td>
<td>0.13</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

Simulation 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

Prediction Error (Fine-grained Decomposition)

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

<table>
<thead>
<tr>
<th>Image size</th>
<th>2 cores</th>
<th>4 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scatter</td>
<td>Compute_Gx</td>
</tr>
<tr>
<td>320x240</td>
<td>-0.58</td>
<td>0.24</td>
</tr>
<tr>
<td>480x320</td>
<td>-1.67</td>
<td>-0.16</td>
</tr>
<tr>
<td>640x480</td>
<td>-2.13</td>
<td>0.02</td>
</tr>
<tr>
<td>800x600</td>
<td>-2.43</td>
<td>0.08</td>
</tr>
<tr>
<td>1024x768</td>
<td>-3.50</td>
<td>0.04</td>
</tr>
<tr>
<td>1280x1024</td>
<td>-4.23</td>
<td>-0.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image size</th>
<th>8 cores</th>
<th>16 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scatter</td>
<td>Compute_Gx</td>
</tr>
<tr>
<td>320x240</td>
<td>-4.63</td>
<td>0.16</td>
</tr>
<tr>
<td>480x320</td>
<td>-4.46</td>
<td>0.10</td>
</tr>
<tr>
<td>640x480</td>
<td>-4.69</td>
<td>-0.12</td>
</tr>
<tr>
<td>800x600</td>
<td>-4.39</td>
<td>-0.30</td>
</tr>
<tr>
<td>1024x768</td>
<td>-4.25</td>
<td>-0.46</td>
</tr>
<tr>
<td>1280x1024</td>
<td>-4.11</td>
<td>-0.53</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image size</th>
<th>32 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scatter</td>
</tr>
<tr>
<td>320x240</td>
<td>-11.14</td>
</tr>
<tr>
<td>480x320</td>
<td>-9.11</td>
</tr>
<tr>
<td>640x480</td>
<td>-7.61</td>
</tr>
<tr>
<td>800x600</td>
<td>-7.06</td>
</tr>
<tr>
<td>1024x768</td>
<td>-6.22</td>
</tr>
<tr>
<td>1280x1024</td>
<td>-5.98</td>
</tr>
</tbody>
</table>

Observations:
- Worst-case error: -3.91%
- Best-case error: -0.09%
for \( ie=0 \) to \( ie = Nel \)
for \( k=0 \) to \( N-1 \)
for \( j=0 \) to \( N-1 \)
for \( i=0 \) to \( N-1 \)
\[ dudr(i,j,k,ie) += a(i,l) \times u(l,j,k,ie) \]

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);
for i=0 to N-1
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-behaved machine**
  - Homogenous machine typically partitioned 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

Node Architecture

- Xeon-E5-2670 nodes
- Two-level switch
- InfiniBand QDR network
- Fat-tree-like layout
- Microsecond latencies

Diagram of the computing cluster showing
- 1296 nodes
- 40 TB memory
- 2.6Ghz Cores
- Two-level switch
- InfiniBand QDR network
- Fat-tree-like layout
- Microsecond latencies

Diagram of the node architecture showing
- Xeon-E5-2670 processors
- 32GB DDR3 memory
- QPI interconnects

Diagram of the network showing
- Big switch A
- Big switch B
- Node 0
- Node 1
- Node X
- Node Y
- Switch 0
- Switch 1
- Microsecond latencies

Diagram of the connectivity showing
- InfiniBand QDR connections
- Microsecond latencies
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.
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.

---

**Aggregate Sent Message Size for different MPI calls**

- Total data transferred
- Average data transferred

---

**Aggregate Time (ms, top 20 calls)**

- Waitall
- Isend
- Irecv
- Barrier
- Allreduce
Data for Estimation of Transfer Times

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

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

```c
// 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

```c
162   double nsCompute = m_random->getNextDouble();
163   enQ_compute( evQ, nsCompute );    // Delay block for compute
164
165   // +x/-x transfers
166   // If even: recv +x, send +x, recv -x, send -x
167   // If odd: send +x, recv +x, send -x, recv -x
168   if ( myX % 2 == 0 ) {
169     if ( sendx_pos ) {
170       enQ_recv( evQ, x_pos, x_xferSize, 0, GroupWorld );
171       enQ_send( evQ, x_pos, x_xferSize, 0, GroupWorld );
172     }
173     if ( sendx_neg ) {
174       enQ_recv( evQ, x_neg, x_xferSize, 0, GroupWorld );
175       enQ_send( evQ, x_neg, x_xferSize, 0, GroupWorld );
176     }
177   } else {
178     if ( sendx_pos ) {
179       enQ_send( evQ, x_pos, x_xferSize, 0, GroupWorld );
180       enQ_recv( evQ, x_pos, x_xferSize, 0, GroupWorld );
181     }
182     if ( sendx_neg ) {
183       enQ_send( evQ, x_neg, x_xferSize, 0, GroupWorld );
184       enQ_recv( evQ, x_neg, x_xferSize, 0, GroupWorld );
185     }
186   }
187
188   // +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

```plaintext
4 networkParams = {
5     "packetSize" : "2048B",
6     "link_bw" : "4GB/s",
7     "link_lat" : "40ns",
8     "input_latency" : "50ns",
9     "output_latency" : "50ns",
10    "flitSize" : "8B",
11    "buffer_size" : "14KB",
12 }
13
14 nicParams = {
15     "module" : "merlin.linkcontrol",
16     "packetSize" : networkParams['packetSize'],
17     "link_bw" : networkParams['link_bw'],
18     "buffer_size" : networkParams['buffer_size'],
19     "rxMatchDelay_ns" : 100,
20     "txDelay_ns" : 50,
21     "nic2host_lat" : "150ns",
22 }
23 ```
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

```python
20    numNodes = 0  # numNodes = 0 implies use all nodes on network
21    numCores = 1
22
23    return workFlow, numNodes, numCores
24
25    def getNetwork():
26
27        platform = 'default'
28
29        topo = 'torus'
30        shape = '2x2x2'
31
32        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

```python
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 )
```

CCMT
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

![Graph showing SST execution time vs. simulated timesteps](image)

- **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
Effect of varying element size on application execution time

- **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

- **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

- **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.