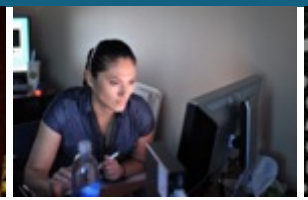
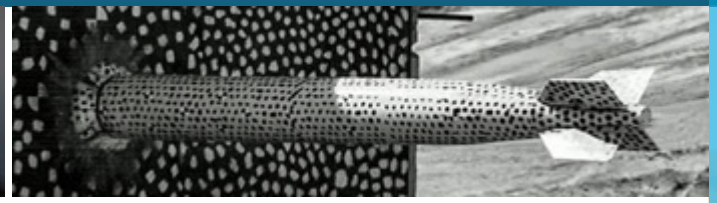
A Data-driven Ember Motif for Breadth-First Search





Collaborators

Patrick Lavin, Heidi Komkov, Joseph Kenny,
Scott Hemmert, Arun Rodrigues

March 1, 2023

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2023-00258PE

Outline

- **Motivation**
- **Ember Overview**
  - Example Ember SDL and Motifs
- **Data Collection**
  - A-mpiP
- **Modeling**
  - Control Flow
  - Computation and Communication
- **Results and Discussion**
  - Scaling on Cori
  - Future modeling opportunities

Motivation

Talk Overview

Motivation

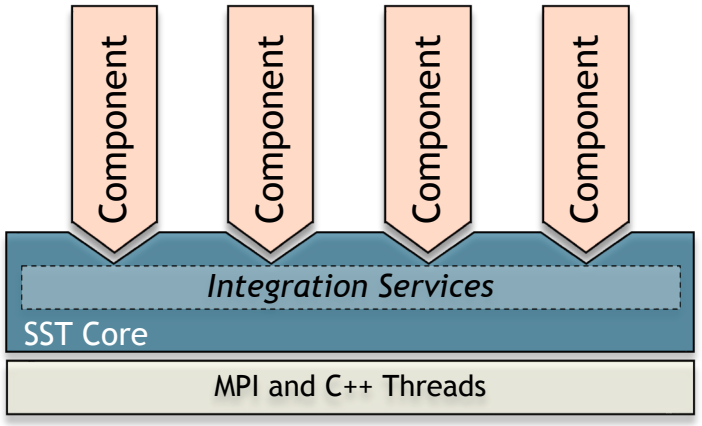
- We want to predict the performance of kernels
  - on larger inputs than we can currently run, and
  - on larger machines than currently exist.
- Kernels with data-dependent access can be hard to model, but with certain constraints, we can simplify the problem for BFS
  - Only R-MAT input graphs
  - Modeled node similar to what currently exists
- These slides cover our approach for generating a an Ember motif from MPI traces.
- But first, we'll talk a bit about what how Ember works.

Ember Overview

High-Level View

SST Architecture

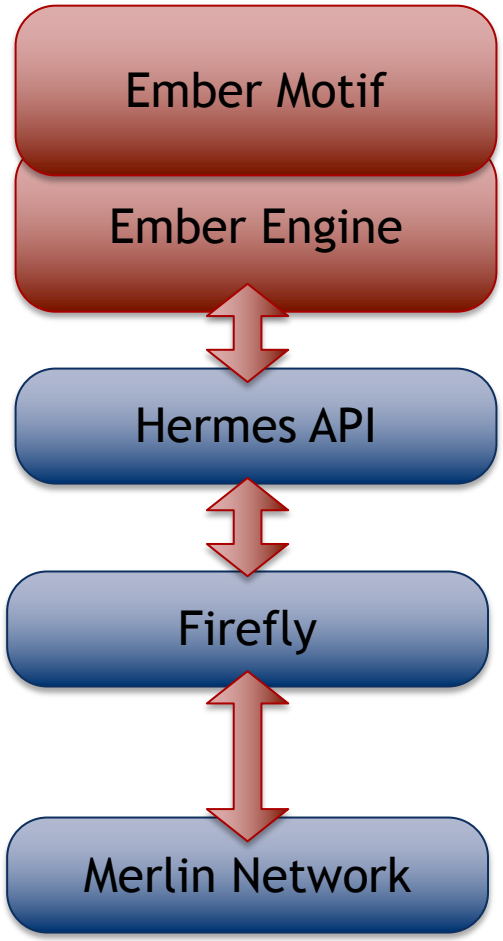
- SST **Core** framework
  - The backbone of simulation
  - Provides utilities and interfaces for simulation components (models)
    - Clocks, event exchange, statistics and parameter management, parallelism support, etc.
- SST **Element** libraries
  - Libraries of components that perform the actual simulation
  - Elements include processors, memory, network, etc.
    - Includes many existing simulators: DRAMSim3, Spike, HMCSim, Ramulator, etc.



Ember: Network Traffic Generator

- Light-weight endpoint for modeling network traffic
  - Enables large-scale simulation of networks where detailed modeling of endpoints would be expensive
- Packages patterns as *motifs*
  - Can encode a high level of complexity in the patterns
  - Generic method for users to extend SST with additional communication patterns
- Intended to be a driver for the Hermes, Firefly, and Merlin communication modeling stack
  - Uses Hermes message API to create communications
  - Abstracted from low-level, allowing modular reuse of additional hardware models

Ember: Overview



High Level Communication Pattern and Logic
Generates communication events

Event to Message Call, Motif Management
Handles the tracking of the motif

Message Passing Semantics
Collectives, Matching, etc.

Packetization and Byte Movement Engine
Generates packets and coordinates with network

Flit Level Movement, Routing, Delivery
Moves flits across network, timing, etc.

Ember: Motifs

- Motifs are lightweight patterns of communication
  - Tend to have very small state
  - Extracted from parent applications
  - Modeled as an MPI program (serial flow of control)
    - Many motifs acting in the simulation create the parallel behavior

- Example motifs
  - Halo exchanges (1, 2, and 3D)
  - MPI collections – reductions, all-reduce, gather, barrier
  - Communication sweeping (Sweep3D, LU, etc.)
  - BFS

Ember: Motifs (continued)

- The EmberEngine creates and manages the motif
  - Creates an event queue which the motif adds events to when probed
  - The Engine executes the queued events in order, converting them to message semantic calls as needed
  - When the queue is empty, the motif is probed again for events

- Events correspond to a specific action
  - E.g., send, recv, allreduce, compute-for-a-period, wait, etc.

Ember: Example Motif - Allreduce

Excerpt from:

sst-elements/src/sst/elements/ember/tests/dragon_128_allreduce.py

```
ep = EmberMPIJob(0,topo.getNumNodes())
ep.network_interface = networkif
ep.addMotif("Init")
ep.addMotif("Allreduce")
ep.addMotif("Fini")
```

Excerpt from:

sst-elements/src/sst/elements/ember/mpi/motifs/emberallreduce.cc

```
bool EmberAllreduceGenerator::generate(…) {

    if ( m_loopIndex == m_iterations ) {
        return true; // This generator is empty
    }
    …

    enQ_compute( evQ, m_compute );
    enQ_allreduce( evQ, m_sendBuf, m_recvBuf, m_count,
    DOUBLE, m_op, GroupWorld );

    if ( ++m_loopIndex == m_iterations ) {
        enQ_getTime( evQ, &m_stopTime );
    }

    return false; // This generator is not empty
}
```

Data Collection

High-Level View

Data Collection: A-mpiP

- We are using a data-driven approach to modeling BFS
- The inputs required for the BFS ISB are limited
  - Only R-mat graphs specified in T&E
- We have extended LLNL's mpiP to capture MPI traces
  - Wrapper around MPI calls using PMPI
  - Our version is available as A-mpiP: https://github.com/plavin/A-mpiP
  - Every rank is traced
- Data Collected from NERSC's Cori
  - `nodes:  [1 4 9 16 25 36 49 64]`
  - `nruns:  [1]`
  - `exe:  [dobfs]`
  - `scales:  [16-24]`
  - `threads:  [1]`
- Average the data across ranks and across 5 runs

Data Collection: Trace files

- A-mpiP produces two types of files:
  - A trace file for each rank
  - An .mpiP file with information about each *callsite*

Example trace file:

```
TRACE src -> dst delta_t Function Comm Ranks

TRACE 1 -> 2  501.0 MPI_Allreduce coll 0xc4000009 8 to 0 1
TRACE 2 -> 3  62.0  MPI_Barrier   coll 0x44000000 0 to 0 1 2 3
TRACE 3 -> 4  31.0  MPI_Allreduce coll 0xc4000008 8 to 0 1 2 3
TRACE 4 -> 5  15.0  MPI_Allreduce coll 0xc4000009 8 to 0 1
TRACE 5 -> 6  37.0  MPI_Bcast     coll 0xc4000008 8 to 0 1 2 3
TRACE 6 -> 7  17.0  MPI_Bcast     coll 0xc4000008 8 to 0 1 2 3
TRACE 7 -> 8  15.0  MPI_Bcast     coll 0xc4000008 8 to 0 1 2 3
TRACE 8 -> 9  14.0  MPI_Bcast     coll 0xc4000008 8 to 0 1 2 3
TRACE 9 -> 10 41.0  MPI_Comm_dup
```

Data Collection: Callsite information

- Callsites are identified as the top three functions on the call stack
- The same MPI call may be in multiple callsites if it is called by different functions, or with different template parameters

Example .mpiP file:

```
ID Lev File/Address        Line Parent_Funct MPI_Call
  1   0 SpParMat.cpp         795 combblas::SpParMat<…>::getncol() const Allreduce
  1   1 BFSFriends.h         330 combblas::FullyDistSpVec<…> combblas::SpMV<…>(…)
  1   2 DirOptBFS.cpp        427 main

  2   0 BitMapFringe.h       148 combblas::BitMapFringe<long, long>::TransposeGather() Allgatherv
  2   1 DirOptBFS.cpp        404 main

  3   0 CommGrid.h            63 combblas::CommGrid::CommGrid(…) Comm_dup
  3   1 DirOptBFS.cpp        378 main

  4   0 CommGrid.h            52 combblas::CommGrid::~CommGrid() Comm_free
  4   1 shared_ptr_base.h    154 std::_Sp_counted_base<…>::_M_release()
  4   2 DirOptBFS.cpp        469 main
```

Modeling BFS

High-Level View

Modeling BFS

- Machine Model
  - Not covered here
  - A model of Cori has been released on the SST github:
    - https://github.com/sstsimulator/a-sst/blob/main/ISB-BFS/cori-simple.py

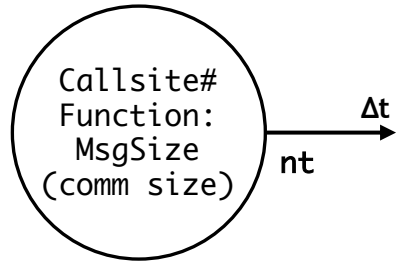
- We need three models to fully represent the ISB
  - Control flow
    - Which MPI routine runs next?
  - Communication volume
    - How large are the messages being sent?
  - Computation time
    - How long does the compute between MPI calls take?

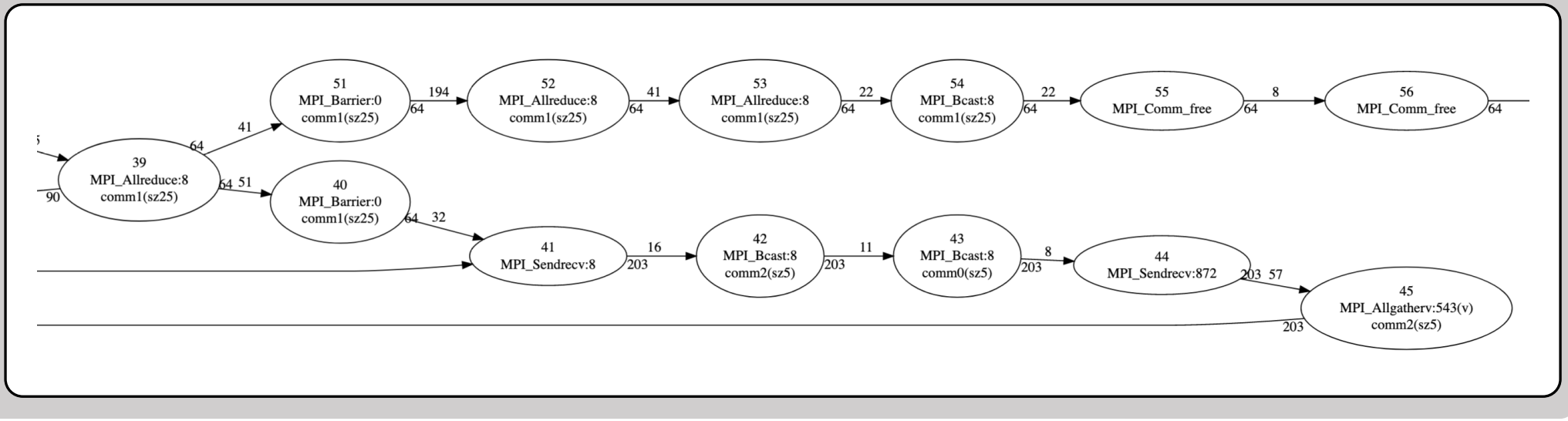
Modeling BFS: Control Flow

Which MPI routine runs next?

- This was simple for BFS – the control flow graph hardly changes for different input sizes and numbers of ranks.
  - 62 callsites
- A trace is turned into a large switch statement in the motif code (see code fragment)
- Manual inspection of traces is done to identify subcommunicators and branching flow

```
switch (callsite) {
case 1:
     MPI_Allreduce(…)
     callsite=2
     break;
case 2:
     MPI_Scatter(…)
     callsite=1
     break;
}
```





Modeling BFS: Motif Excerpt

```
case 3: // MPI_Barrier
    {
        int next_state = 4;
        auto exec_time = exec_model.at(std::make_tuple(state, next_state, threads))->eval(nodes, sz);

        enQ_barrier( evQ, comm1 );
        enQ_compute(evQ, exec_time);

        state = next_state;
    }
    break;

case 4: // MPI_Allreduce
    {
        int next_state = 5;
        auto msg_size  = msg_model_nd.at(std::make_tuple(state,threads))->eval(nodes, sz);
        auto exec_time = exec_model.at(std::make_tuple(state,next_state,threads))->eval(nodes, sz);

        enQ_allreduce( evQ, nullBuf, nullBuf, msg_size, CHAR, MP::MAX, comm1 );
        enQ_compute(evQ, exec_time);

        state = next_state;
    }
    break;
```

Modeling BFS: Model Types for Communication and Computations

- The BFS motif takes two model files as input:
  - A message size model
  - A communication time model
- The message size model is per-callsite, meaning we have a separate model for each MPI routine
- The communication time model is per-callsite transition, meaning that the computation between any two callsite is modeled separately
- The models are parameterized on number of nodes and input size:
  - `msg_size  = msg_model[callsite](nodes, size)`
  - `exec_time = exec_model[callsite_transition](nodes, size)`

```
Available model types:

Constant    : A
Exponential : exp(A*nodes*size + B*nodes + C*size + D)
Polynomial  : a_{0,0} + a_{1,0}*size^1 + a_{0,1}*nodes + a_{1,1}*nodes*size + ... // up to 4th order
Trace       : (get value from MPI trace)
```

Modeling BFS: Trace and Constant Models

- Trace models:
  - To generate a trace model, we take a rank 0 trace file, and transform it into the following format:
  - This line indicates that callsite 28 sent messages of sizes 0, 32, 0, 0, and 4 bytes the first 5 times it was called. The rest of the line is omitted for brevity.
  - The "1" following the callsite number is the number of threads the model was created for simulating.

```
28 1 TRACE 0.0 32.0 0.0 0.0 4.0
```

- Constant models:
  - A number of callsites send the same size message very time, independent of the number of nodes or the graph size, such as callsite 15.

```
15 1 CONSTANT 4.0
```

Modeling BFS: Polynomial Models

- The polynomial models are 4th-order
  - The include terms for exponents summing to up to 4.
- To create a models, create the Vandermode matrix for two variables

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \end{pmatrix} \qquad n = 4,\ x_1 = size,\ x_2 = nodes$$

- Then, find the coefficients using a least-squares fit
- An example model is below. Higher-order terms are omitted for brevity.
- A column labeled X, Y means that column contains the coefficients for the term $size^X nodes^Y$.

src	dst	threads	0,0	1,0	0,1	2,0	1,1	0,2	3,0	2,1	...
1	2	1	-6890.8582	1593.4272	-17.5702	-136.3303	2.3121	0.0652	5.0886	-0.0351	...

Modeling BFS: Creating the Comm and Comp Models

- The computation and communication scales linearly with the number of edges, but the number of edges scales exponentially with the R-MAT size
- To create the exponential models, we take the log of the data, then do a least squares fit to the following model:

$$A * nodes * size + B * nodes + C * size + D$$

- To use the model, evaluate the polynomial and take the exponential.

```
41 1 EXPONENTIAL -0.0001 -0.03205 0.65496 -4.1217
```

Modeling BFS: Model Code

How are the models implemented

```
struct
{
    double scale = 1.0;
    virtual double eval(int nodes, int size) = 0;
};
```

```
struct ConstModel : public Model
{
    double val;
    ConstModel(double val) : val(val) {}

    double eval(int nodes, int size) override {
        return val*scale;
    }

};
```

```
// Create model for callsite `cs`
msg_model[cs] = unique_ptr<Model>(new
ConstModel(constant));

// Use model
msg_size = msg_model_new.at(cs)->
eval(nodes,size);
```

- A generic Model class contains only the eval() function interface
- Specialized classes exist for our 5 model types, Constant, Bilinear, Polynomial, Exponential, and Trace
- Polymorphism ensures the correct model is called for each callsite

Modeling BFS: Summary

- Control flow modeling is largely manual
  - Inspection of subcommunicators, path choice, required
- Computation and communication use
  - 4rd-order, 2-dimensional polynomials,
  - Exponential models
  - Constant models or
  - Trace-driven models
- There is plenty of opportunity to extend the number of models. We'll discuss that after looking at some results.

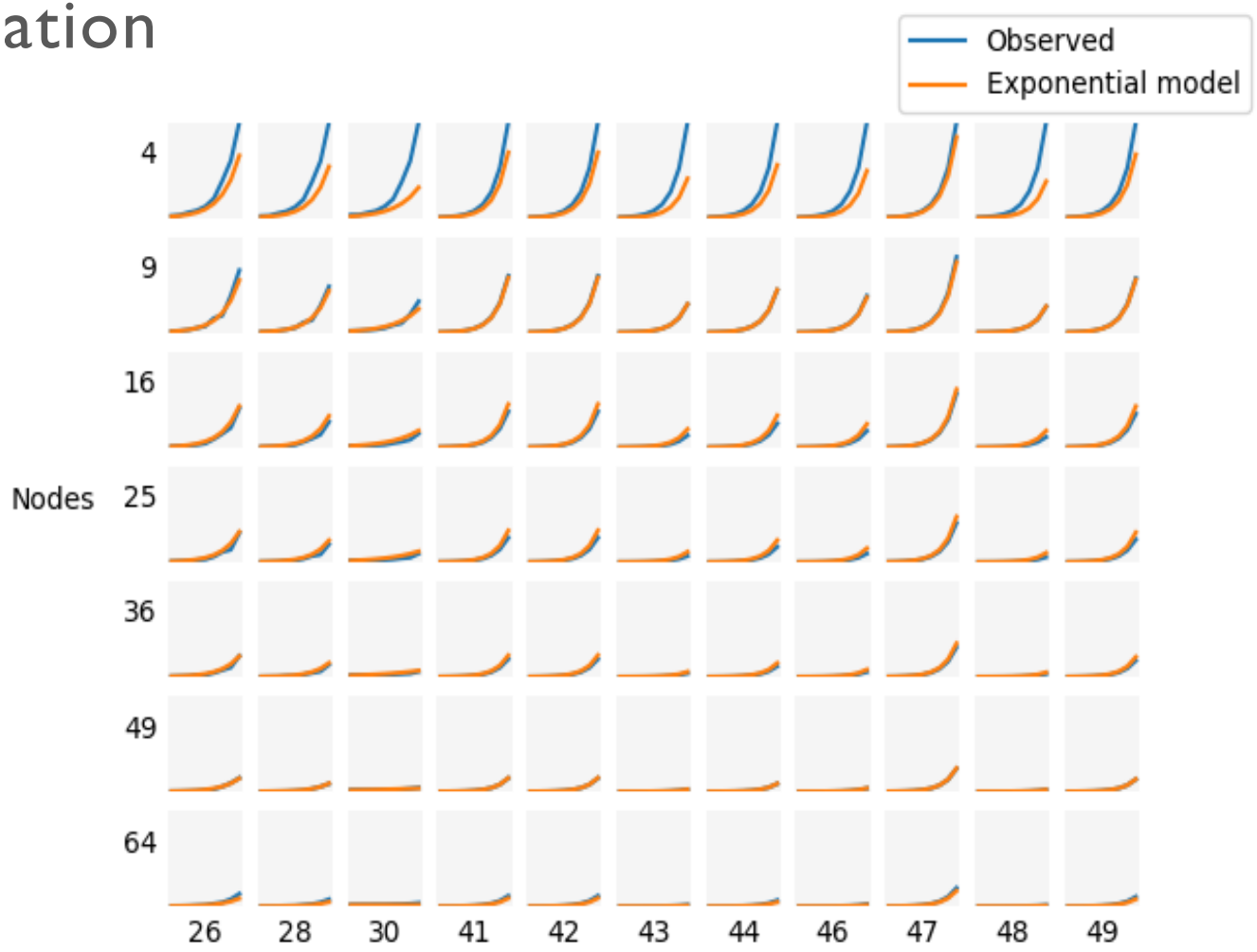
Results and Discussion

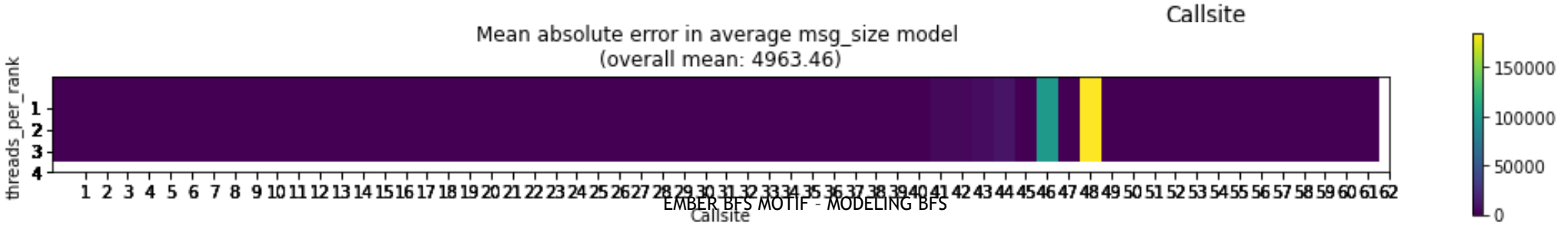
High-Level View

Modeling BFS: Communication

Results

- Only 11 callsites looked exponential, with the rest being constant
- Each small plot shows how the exponential model scales as a function of input size.
- There is a small plot for each callsite and each number of nodes
- As the number of nodes is a model parameter, each column to the right is a single model
- Note: graphs in the same column have the same scale, but different columns have different scales.

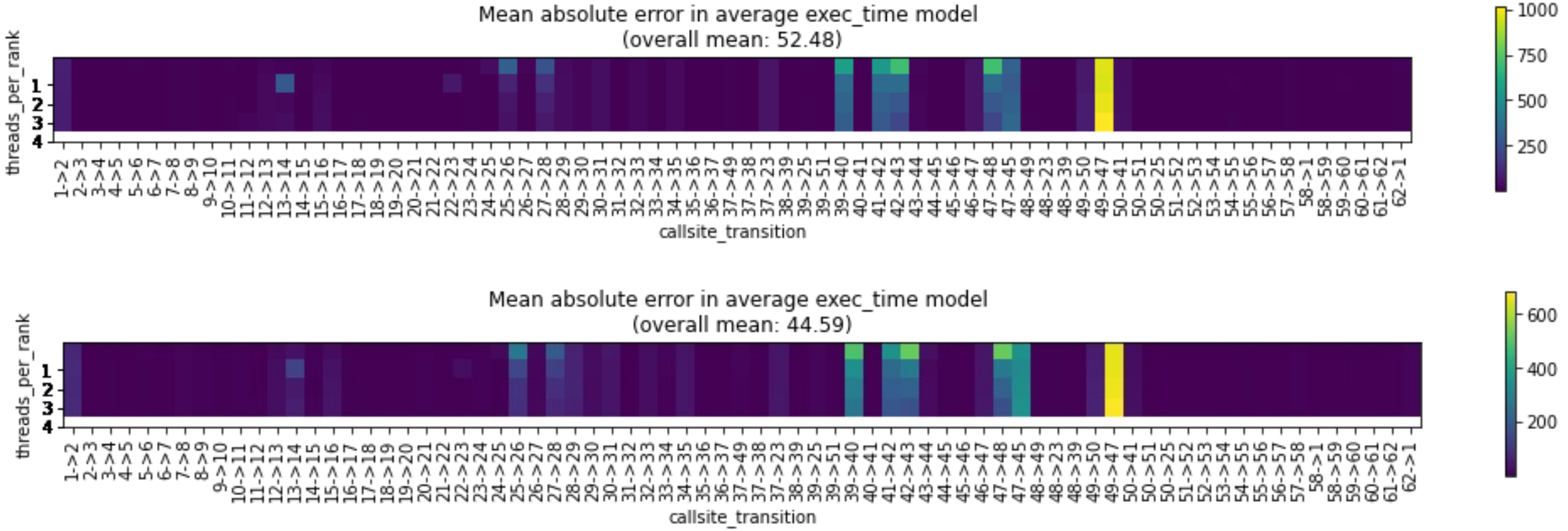




Modeling BFS: Computation

Results

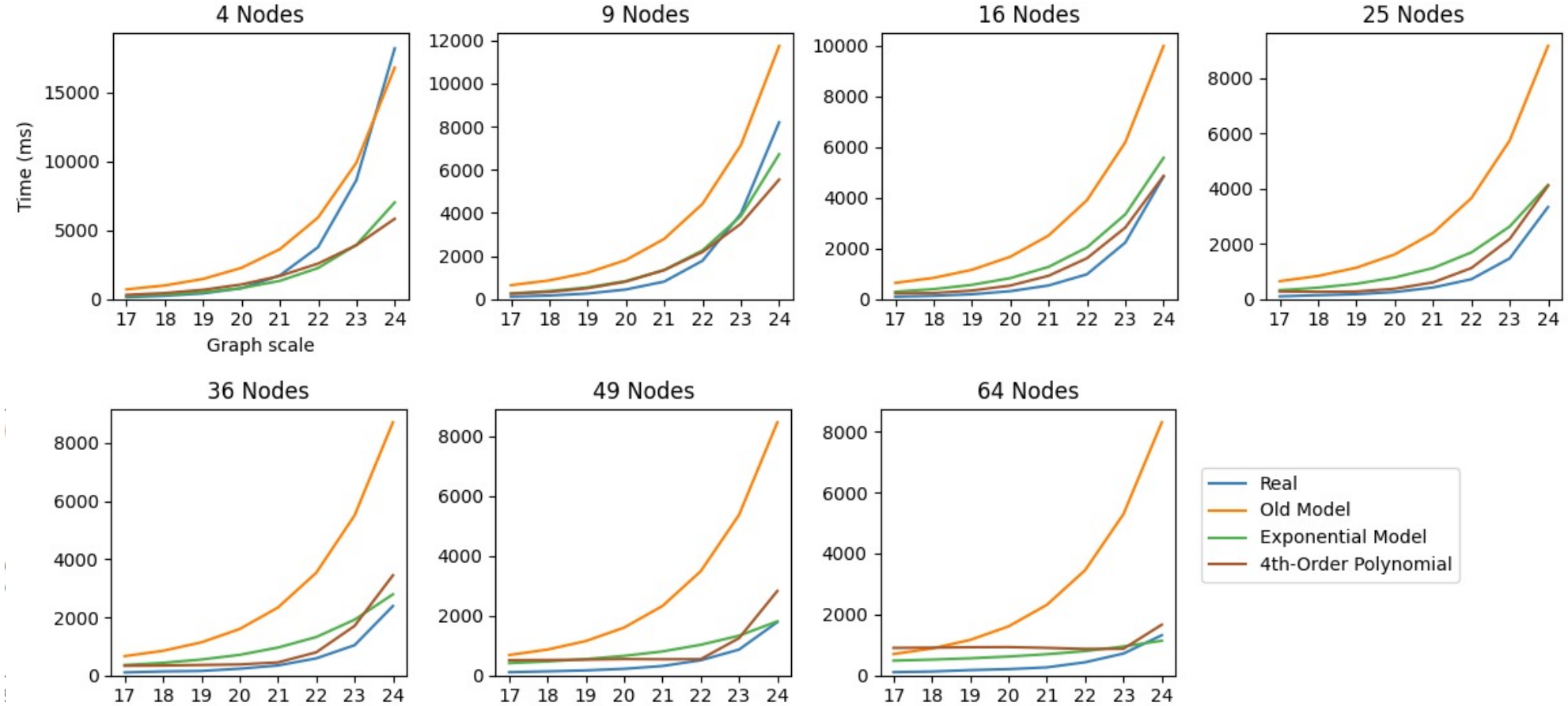
- The exponential models slightly outperformed the polynomial models for computation (compare the scales).



Modeling BFS: Overall

Results

- The polynomial and exponential models both perform better than the old model that was created by manually examining each callsite and transition.



Future modeling opportunities

- We implemented 4 types of models in this work, representing both communication and computation
- Limitations:
  - Limited set of models
  - Choosing between models is heuristic
  - Time dependency is not modeled
  - Each rank uses the same model
- We modeled control flow by manual examination

- Opportunities:
  - Automate control flow modeling with Markov models
  - Use ML models for communication and computation
  - Use techniques from dynamical systems such as SiNDY
  - Integrate data on multi-threaded runs into the model, instead of making a separate model per number of threads
  - Collect and integrate data on runs that use multiple ranks per node
  - Integrate a more detailed node-level model
    - Large changes to Ember required

References

- Websites
  - http://www.sst-simulator.org/
  - https://github.com/sstsimulator
- A-mpiP (MPI Tracing code):
  - https://github.com/plavin/A-mpiP
- Motif Code (included with SST-Elements):
  - https://github.com/sstsimulator/sst-elements/blob/master/src/sst/elements/ember/mpi/motifs/emberBFS.cc
- SDL and Model Parameter Files:
  - https://github.com/sstsimulator/a-sst/tree/main/ISB-BFS


Sandia
National
Laboratories