Simulating Large-Scale Systems with MVAPICH2 and SST

9th MVAPICH User Group (MUG) Conference '21

Presentation at MUG '21

Kaushik Kandadi Suresh, Tu Tran

Network Based Computing Laboratory (NBCL)

Dept. of Computer Science and Engineering , The Ohio State University

kandadisuresh.1@osu.edu

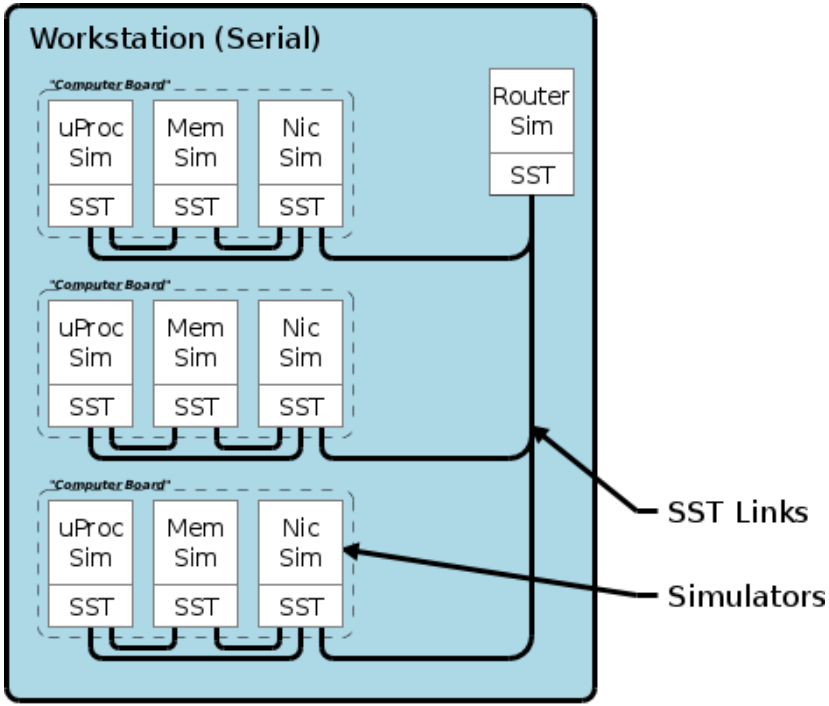
Outline

- Introduction
- Example
- Architecture
- SST Macro
- MVAPICH2 + SST Macro Integration
- SST+MVAPICH2 : Performance prediction of collective algorithms
- SST+MVAPICH2 : Performance of MPI_Alltoall on different topologies

Introduction

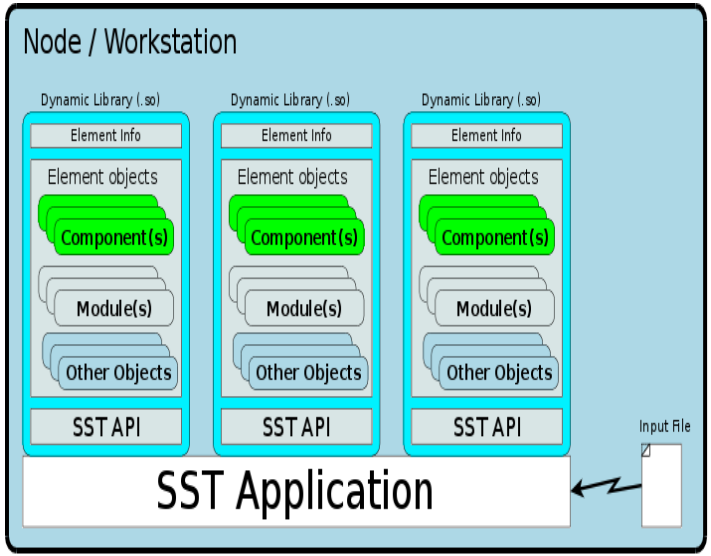
- The Structural Simulation Toolkit (SST) is an open-source software framework used for simulating large-scale High Performance Computing (HPC) systems
- The simulations can be run on one or many nodes.
- SST lets multiple disparate/discrete simulators to share data with each other in order to perform very large scale simulations.

SST-Example



- Simulation of 3 “computer boards”
- Each “computer board” simulations are comprised of 3 different simulators sharing data using SST
- There will be 10 separate concurrent simulations running on this single workstation all sharing data using SST

SST- Architecture



- SST is a standalone application
- When run, it loads set of Dynamic Libraries (called Element Libraries) using a config file
- These Element Libraries contain several internal objects (Components, Modules, etc) that represent the simulators and support objects to perform a simulation.
- The Element Libraries are developed in C++ using SST API

SST-Macroscale Element Library

- The macroscale element library provides functionality for simulating extreme-scale applications

```
1  int size = atoi(argv[1]);
2  if (rank == 0){
3   int partner = 1;
4    MPI_Send(buffer, size, MPI_INT, partner, tag, MPI_COMM_WORLD);
5  } else {
6    int partner = 0;
7    MPI_Recv(buffer, size, MPI_INT, partner, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
8  }
9  MPI_Barrier(MPI_COMM_WORLD);
10
11 if (rank == 0){
12   printf("Rank 0 finished at t=%8.4f ms\n", MPI_Wtime()*1e3);
13 }
```

```
node {
 app1 {
  name = send_recv
  launch_cmd = aprun -n 2
  argv = 20
 }
}
```

SST-Macro Working

- Discrete Event Simulation (DES)
  - Components and Events
  - Component send events to other components
  - Each Event has Arrival time
  - Components process events in time-order (stored in sorted queue of events)
  - Time advances as components pop and process events from the queue
- Components:
  - NIC, Router, Memory Models, Node
- SST macro has ‘Application’ which is just a stack and a heap without event queue
- SST macro intercepts MPI calls when compiled with sst++, which defines macro to replace the MPI calls

SST macro parameter file

- Network and Application launch configuration
- Network parameters
  - Switch
  - Topology
  - Communication model
  - NIC
- CPU
  - Processor
  - Memory

```
app1 {
 exe = ./run
 argv =
 launch_cmd = aprun -n 256 -N 1
 env {
   SLURM_NPROCS = 256
 }
}
```

```
topology {
  name = torus
  geometry = [32, 32, 32]
  redundant = [32,32,32]
}
```

```
memory {
 name = snappr
 channel_bandwidth = 7GB/s
 num_channels = 10
 latency = 10ns
}
proc {
 ncores = 56
 frequency = 2GHz
}
name = simple
}
```

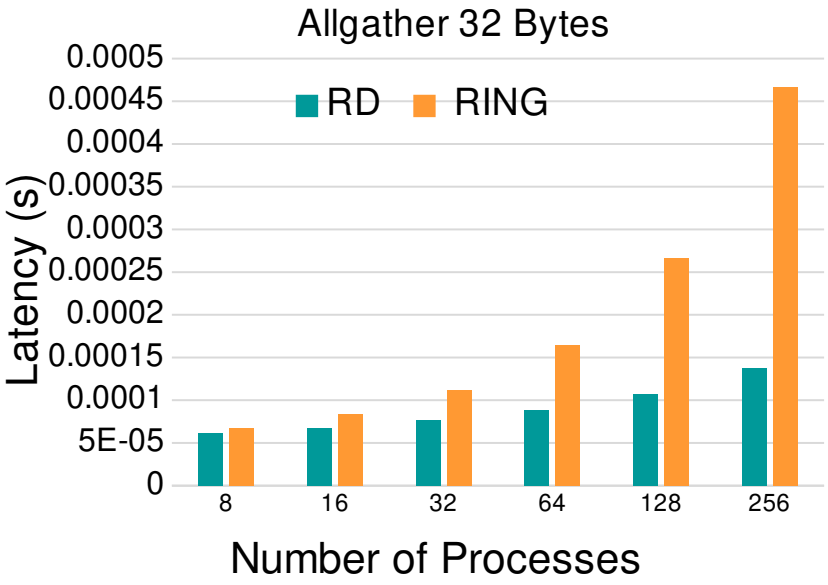
```
switch {
 name = snappr
 link {
  bandwidth = 10.0GB/s
  latency = 100ns
  credits = 8KB
  xmit_active {
   group = test
   type = accumulator
  }
```

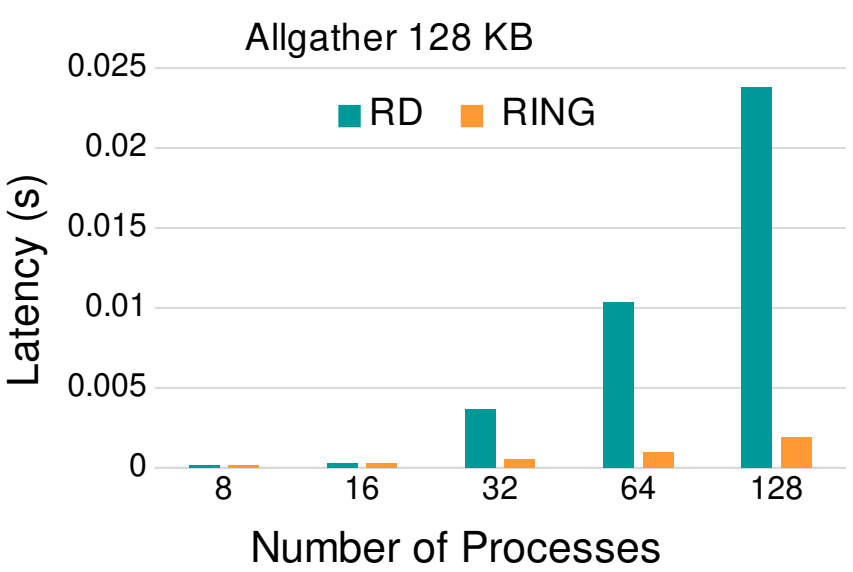
SST macro + MVAPICH2

- SST Transport:
  - Emulates transport library (eg: libfabric)
- Steps to run MPI Application with SST macro + MVAPICH2
  - Install Clang
  - Build SST macro using clang. This generates sstcc (a wrapper for compiling with sst)
  - Build SST transport using clang, SST macro
  - Build MVAPICH2 with sstcc and link it with SST-transport library [provides libfabric, pmi cache emulation]
  - Compile MPI program with mpicc generated in the previous step
  - Add run time params to config file : parameter.ini
  - Run as follows:
    - sstmac –f parameter.ini ./app.exe

Estimate relative performance of algorithms with SST

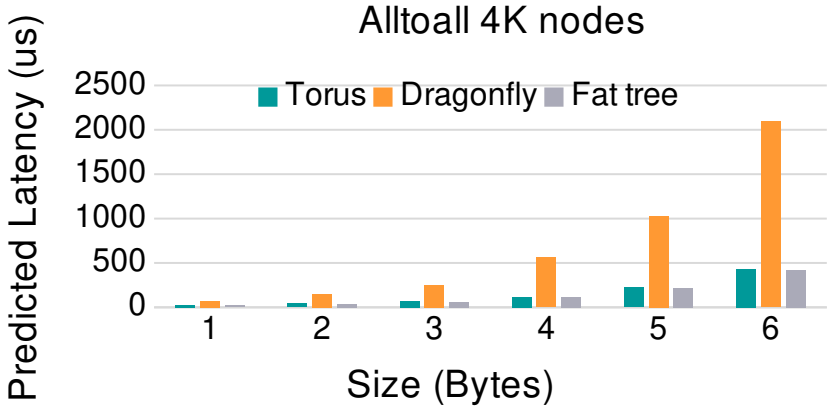
- Ring algorithm good for large messages
- Recursive doubling good for small messages
- Useful for estimating any new algorithm implemented in MVAPICH2





Understanding the impact of topology

- Large scale simulation of MPI_Alltoall using OMB
- Impact of topology on Alltoall algorithm implemented in MVAPICH2



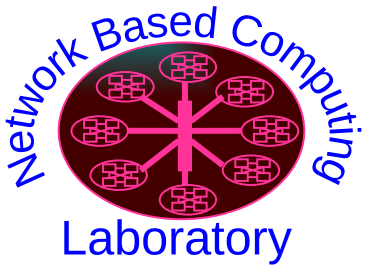

```
topology {
  name = torus
  geometry = [32, 32, 32]
  redundant = [32,32,32]
}
```

```
topology {
  name = fat_tree
  leaf_switches_per_subtree = 64
  agg_switches_per_subtree = 64
  concentration = 64
  up_ports_per_leaf_switch = 64
  down_ports_per_agg_switch = 64
  num_agg_subtrees = 2
  num_core_switches = 64
  up_ports_per_agg_switch = 64
  down_ports_per_core_switch = 128
}
```

```
topology {
  name = dragonfly
  geometry = [32,9]
  h = 16
  inter_group = circulant
  concentration = 16
}
```

Thank You!

Kandadisuresh.1@osu.edu



Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

Follow us on

https://twitter.com/mvapich

MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/