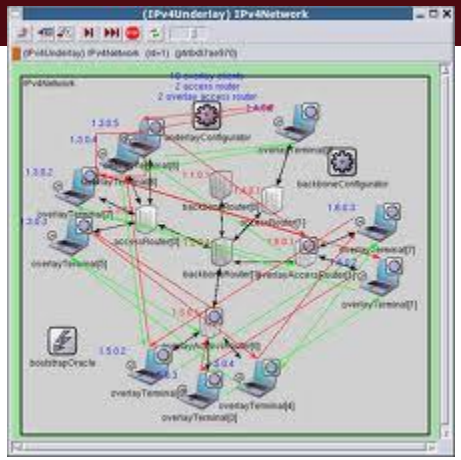
Enabling Scalable Parallel Processing of Venus/OMNeT++ Network Models on the IBM Blue Gene/Q Supercomputer

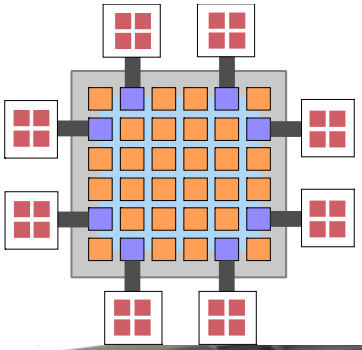
Chris Carothers,
Elsa Gonsiorowski and
Justin LaPre
Center for Computational Innovations
Rensselaer Polytechnic Institute

Philip Heidelberger,
German Herrera,
Cyriel Minkenberg and
Bogdan Prisacari
IBM Research, TJ Watson and Zurich

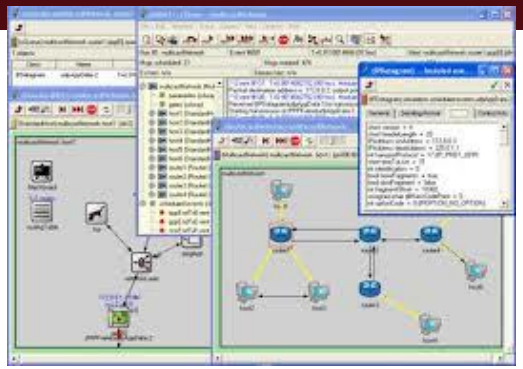
Outline

- **Motivation and Goals**
- **IBM Blue Gene/Q**
- **PDES & YAWNS**
- **YAWNS Implementation**
- **Porting Venus/OMNeT++**
- **Performance Results**
- **Plans for the Future**

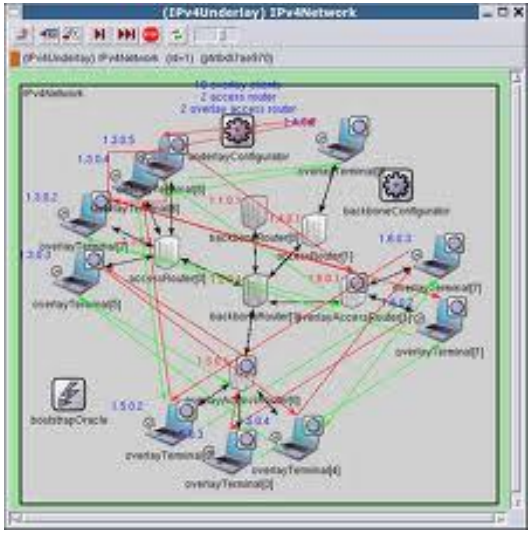




Motivation: Need for Parallel Network Simulation

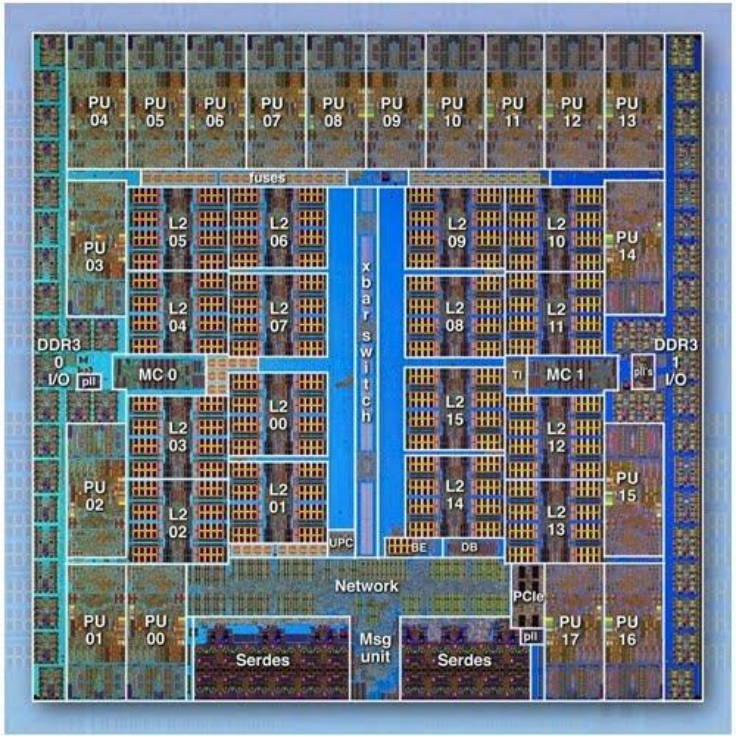


- **IBM's Venus HPC Network Simulator** is built on **OMNeT++**
  - Significant IBM investment over the last 5 years
- **OMNeT++** provides basic building blocks and tools to develop sequential event-driven models
- Written in C++ with a rich class library that provides:
  - Sim kernel, RNG, stats, topology discovery
  - "Modules" and "channels" abstractions
  - NED language for easy model configuration
- **Challenge:** sequential simulation execution times of days to weeks depending on the traffic load and topology size
- **Solution: Enable scalable parallel network simulation for Venus network models on the Blue Gene/Q and MPI clusters**



Goal: 50 to 100x speedup using BG/Q

IBM Blue Gene/Q Architecture



- 1.6 GHz IBM A2 processor
- 16 cores (4-way threaded) + 17th core for OS to avoid jitter and an 18th to improve yield
- 204.8 GFLOPS (peak)
- 16 GB DDR3 per node
- 42.6 GB/s bandwidth
- 32 MB L2 cache @ 563 GB/s
- 55 watts of power
- 5D Torus @ 2 GB/s per link for all P2P and **collective comms**

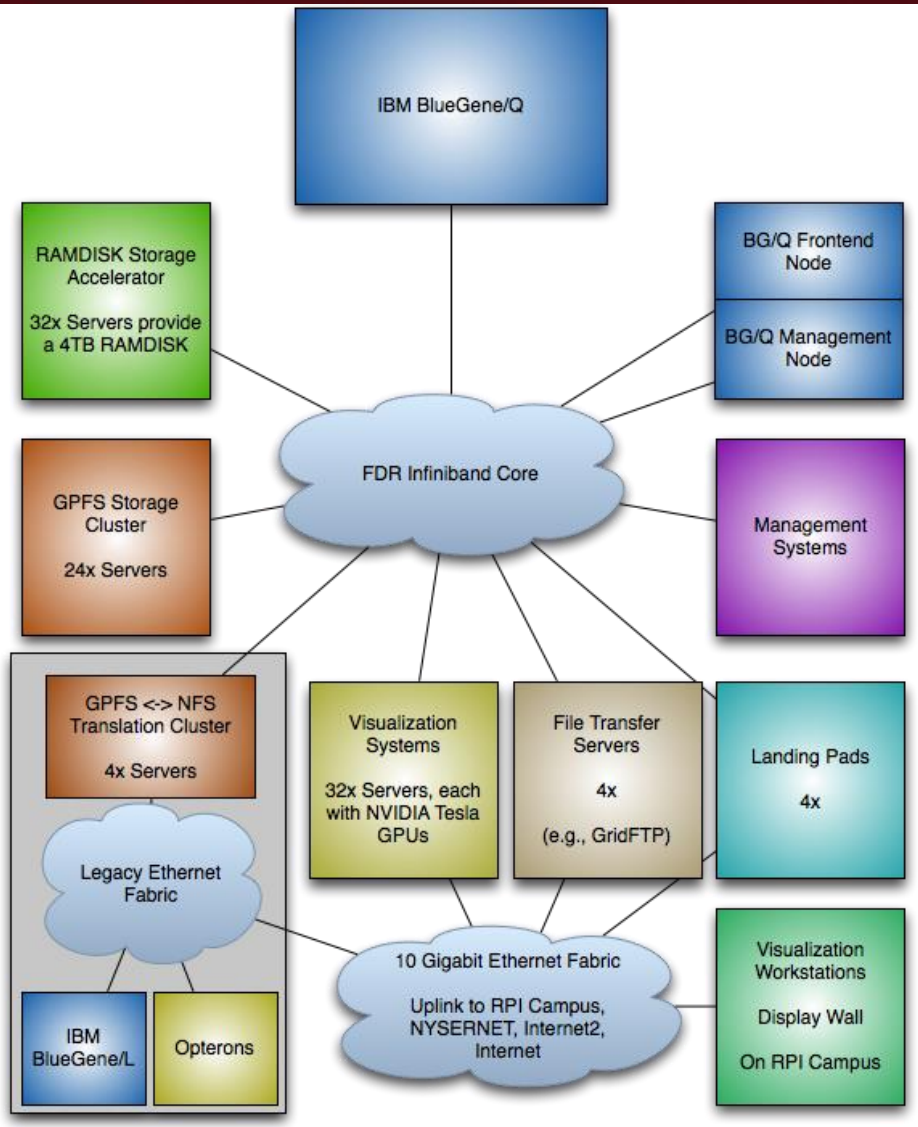
1 Rack =

- **1024 Nodes, or**
- **16,384 Cores, or**
- **Up to 65,536 threads or MPI tasks**

“Balanced” Supercomputer @ CCI

RPI CCNI / MRI Architecture - 2012

- IBM Blue Gene/Q
  - 5120 nodes / 81920 cores
  - *1* teraFLOPS @ 2+ GF/watt
  - **10PF and 20PF DOE systems**
  - *Exec Model: MPI + threads*
  - 80 TB RAM
  - **160 I/O nodes (4x over other BG/Qs)**
- **Clusters**
  - 64 Intel nodes @ 128 GB RAM each
  - 32 Intel nodes @ 256 GB each
- **Disk storage: ~2 Petabytes**
  - IBM ESS w/ GPFS
  - Bandwidth: 5 to ~20 GB/sec
- **FDR 56 Gbit/sec Infiniband core network**



OMNeT++: Null Message Protocol (NMP)

Null Message Protocol (executed by each MPI rank):

Goal: Ensure events are processed in time stamp order and avoid deadlock

WHILE (simulation is not over)

wait until each FIFO contains at least one message

remove smallest time stamped event from its FIFO

process that event

send null messages to neighboring LPs with time stamp indicating a lower bound on future messages sent to that LP (current time plus minimum transit time between cModules or cSimpleModules)

END-LOOP

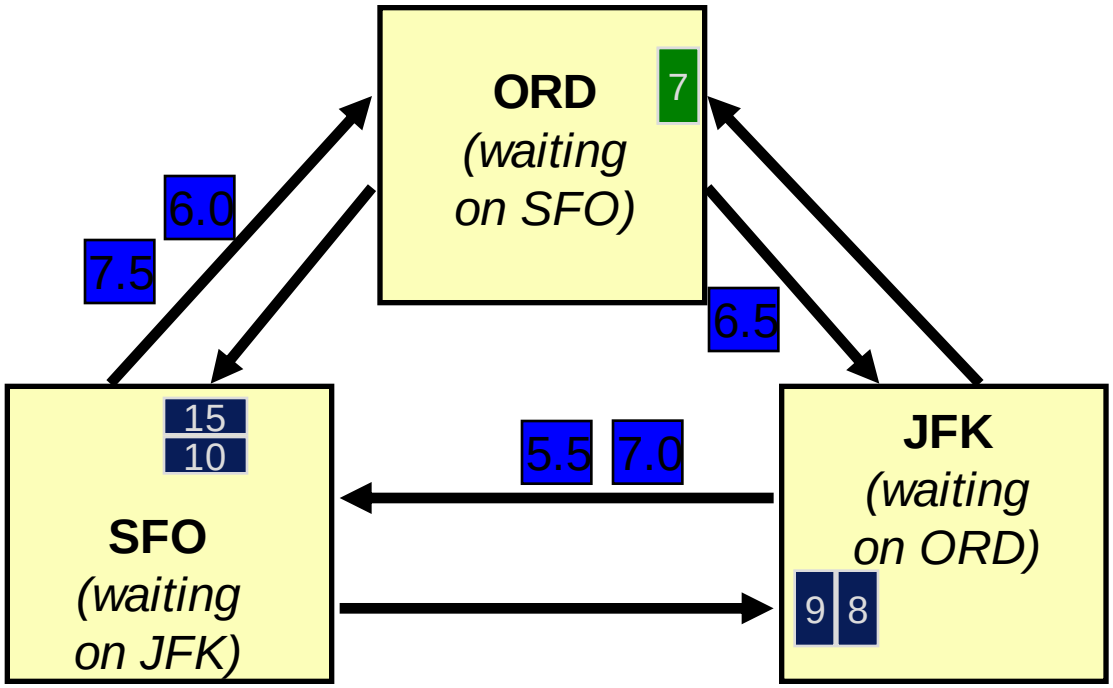
Variation: LP requests null message when FIFO becomes empty

- Fewer null messages
- Delay to get time stamp information

NMP and Lookahead Constraint

- The Null Message Protocol relies on a "prediction" ability referred to as ***lookahead***
- **Airport example:** "ORD at simulation time 5, minimum transit time between airports is 3, so the next message sent by ORD must have a time stamp of *at least* 8"
- **Link lookahead:** If an LP is at simulation time T, and an outgoing link has lookahead L_i, then any message sent on that link must have a time stamp of at least $T+L_i$
- **LP Lookahead**: If an LP is at simulation time T, and has a lookahead of L, then any message sent by that LP must will have a time stamp of at least T+L
  - Equivalent to link lookahead where the lookahead on each outgoing link is the same

NMP: The Time Creep Problem



Null messages:

JFK: timestamp = 5.5

SFO: timestamp = 6.0

ORD: timestamp = 6.5

JFK: timestamp = 7.0

SFO: timestamp = 7.5

ORD: process time stamp 7 message

Five null messages to process a single event!

Assume minimum delay between airports is ~~3~~ 0.5 units of time

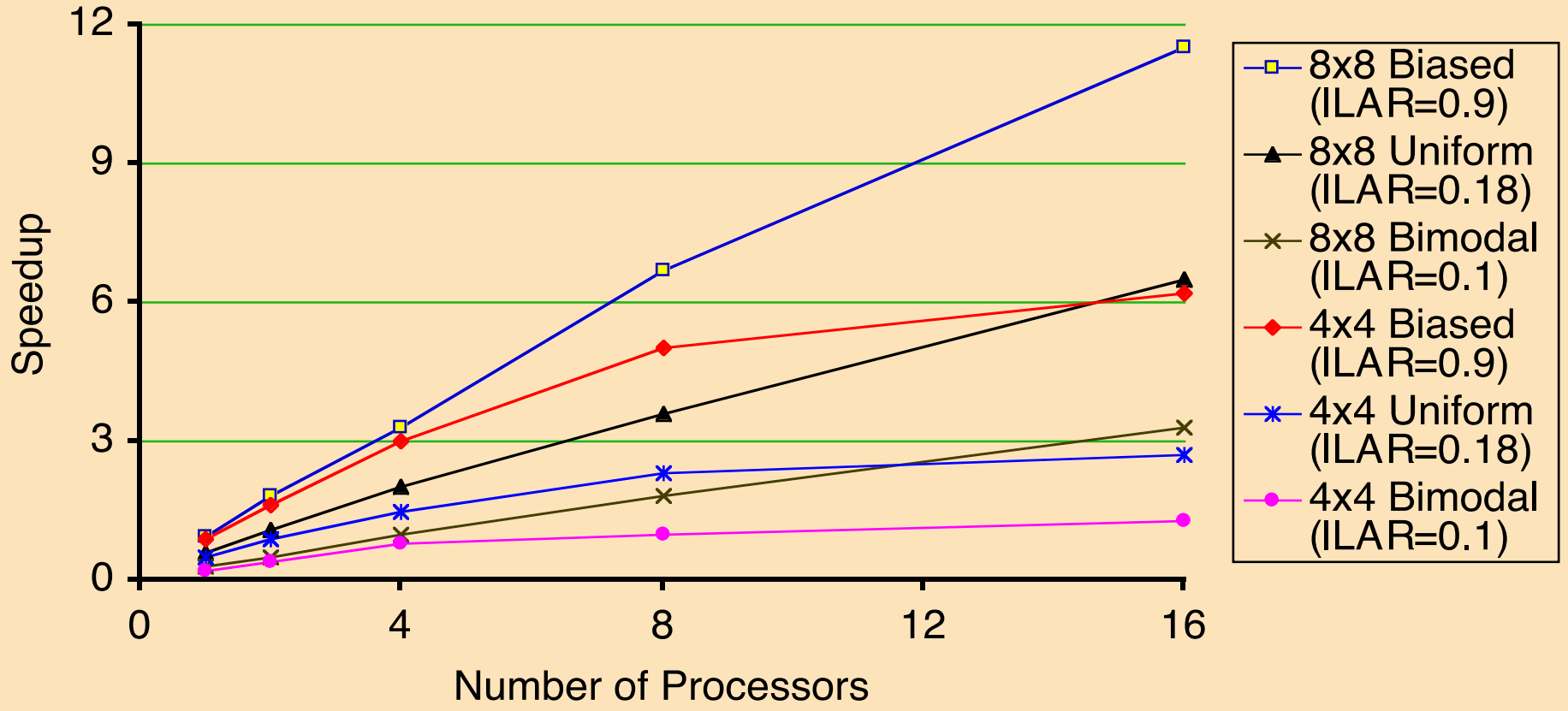
JFK initially at time 5

Many null messages if minimum flight time is small!

Null Message Algorithm: Speed Up

- toroid topology
- message density: 4 per LP
- 1 millisecond computation per event

- vary time stamp increment distribution
- **ILAR=lookahead / average time stamp increment**



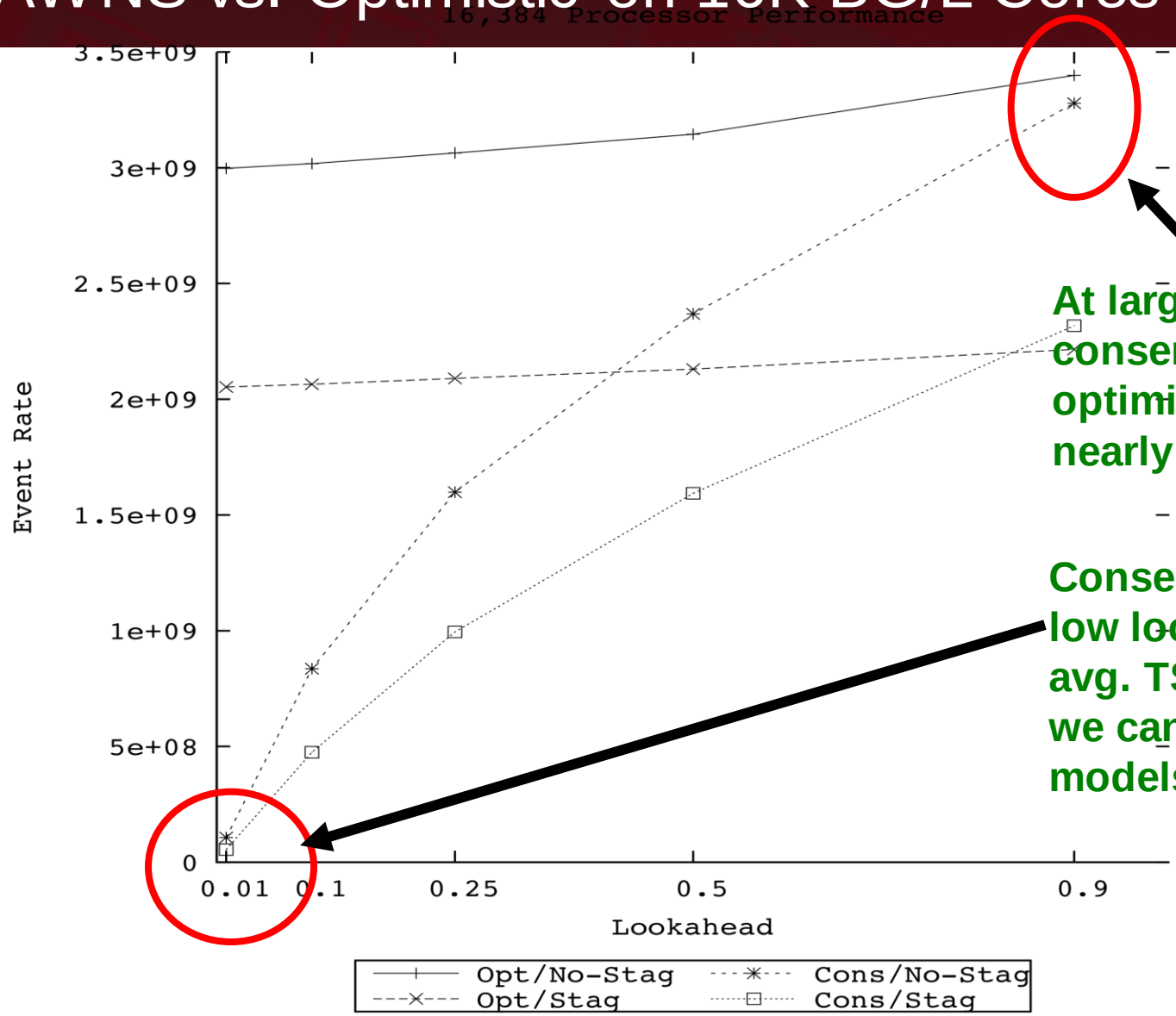
Conservative algorithms live or die by their lookahead!

Overview of YAWNs Into OMNeT++

```
YAWNS_Event_Processing()
// This is a windowing type protocol
//  to avoid NULL messages!!
while true do
    process network queues
    process inbound event queue
    if smallest event >= GVT +
      Lookahead then
     compute new GVT
    end if
    if simulation end time then
     break
    end if
    process events  subject to:
      event.ts < GVT + Lookahead
end while
```

- Must use OMNeT++ existing parallel simulation framework due to object ownership rules
- Migrated YAWNS implementation from ROSS into OMNeT++
  - ROSS has shown great performance out to 16K cores
  - Translated iterative scheduler into a re-entrant one using API
- Uses a single global model "lookahead" value
- Allows zero timestamp increment messages to "self"
- Can switch from NullMessage or YAWNS w/i OMNeT++ model config.

YAWNS vs. Optimistic on 16K BG/L Cores Using ROSS



At large lookaheads, conservative and optimistic performance are nearly equal

Conservative very poor at low lookahead relative to avg. TS increment which we can have in system models

GVT: Global Control Implementation

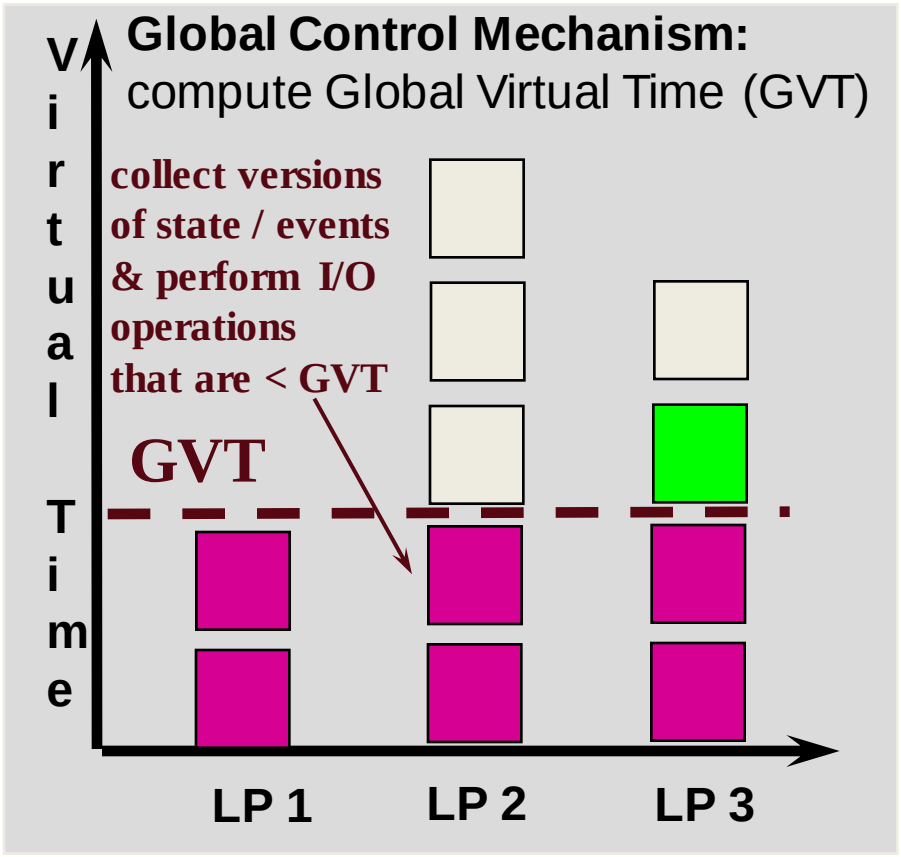
GVT (kicks off when memory is low):

1. Each core counts #sent, #recv
2. Recv all pending MPI msgs.
3. MPI_Allreduce Sum on (#sent - #recv)
4. If #sent - #recv != 0 goto 2
5. Compute local core's lower bound time-stamp (LVT).
6. GVT = MPI_Allreduce Min on LVTs

An interval parameter or lack of local events controls when GVT is done.

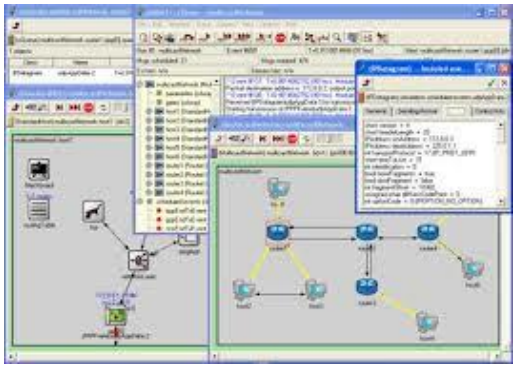
Repurposed GVT to implement conservative YAWNS algorithm!

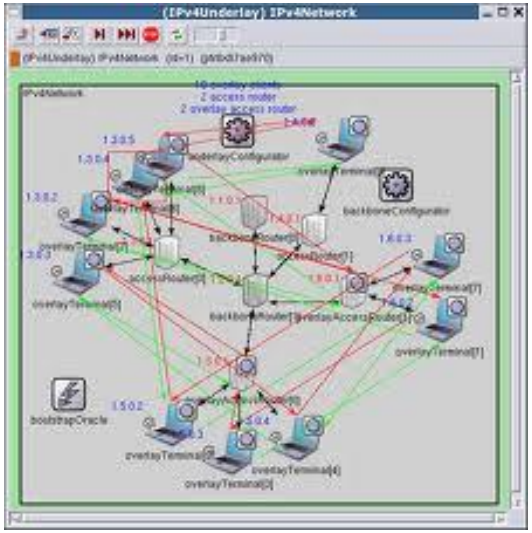
GVT is typically used by Time Warp/Optimistic synchronization



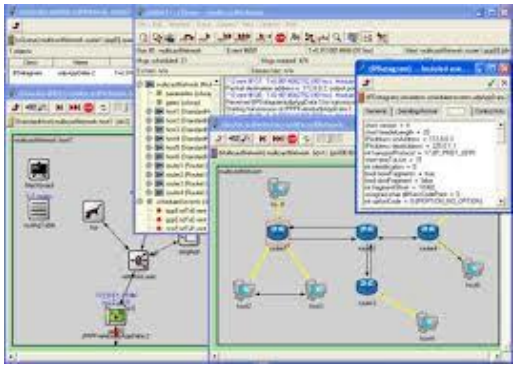
OMNeT++ Parsim API

- OMNeT++ Parsim API supports new conservative parallel algorithms
- NMP and “ideal” supported
- New algorithm must write the following methods:
  - class constructor and destructor
  - **`startRun()`:**
  - `setContext()`:
  - **`endRun()`:**
  - **`processOutgoingMessage()`:**
  - **`processReceivedBuffer()`:**
  - **`getNextEvent()`:**
  - `reScheduleEvent();`



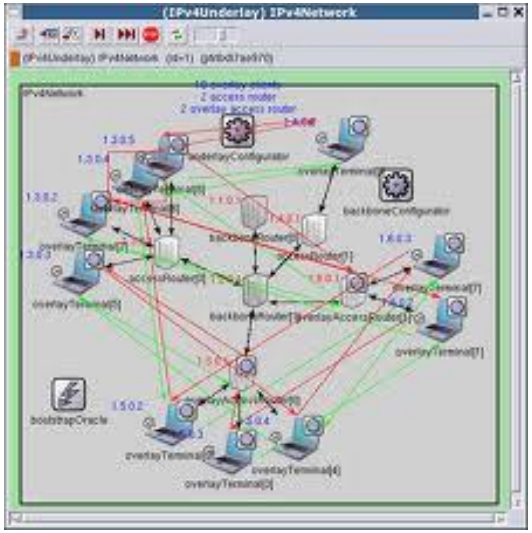


OMNeT++ YAWNs: startRun() & endRun()



cYAWNS::startRun()

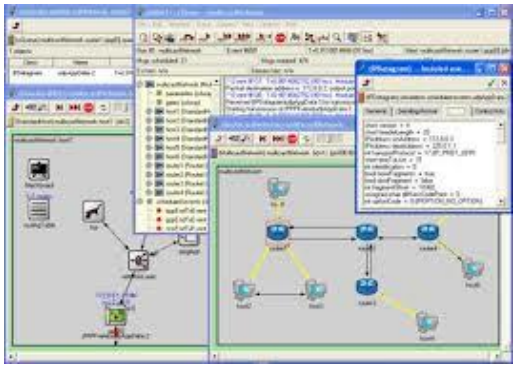
- Init segment and partition information
- Exec correct lookahead calculation method using segment/partition information
- Note, OPP::SimTime::getMaxTime() does not work on Blue Gene/Q.
  - MaxTime hardwired to 10 seconds
-



cYAWNS::endRun()

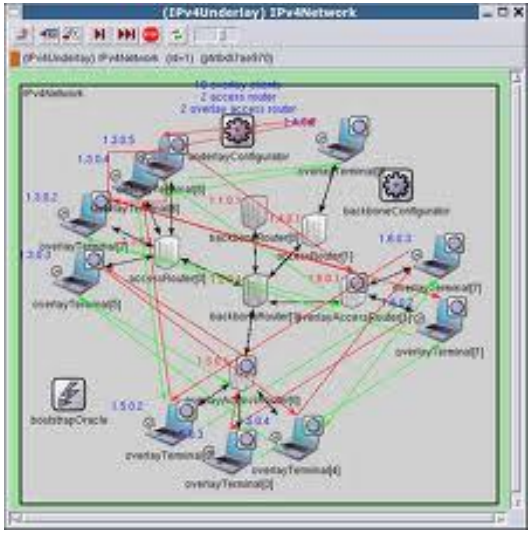
- Computes one last GVT if needed
- Cleans-up the lookahead calc
- Need to more fully understand OMNeT's exception generation and handling mechanisms

OMNeT++ YAWNs: Processing Messages



cYAWNS::processOutGoingMessages()

- All remote messages sent using “blocking” MPI operations
- Message data is “packed” into a single block of memory
  - Records destination module ID and gate ID information
  - Model messages tagged as CMESSAGE
- Increments message sent counter used by GVT



cYAWNS::processRecievedBuffer()

- “Unpacks” MPI message into a cMesssage class
- Increments message recv’ed counter used by GVT

OMNeT++ YAWNs: getNextEvent()

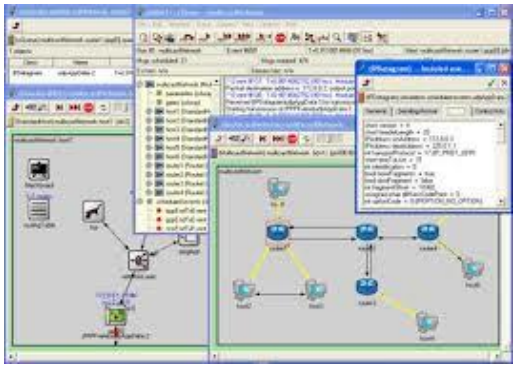
cMessage *cYAWNS::getNextEvent()

```
static unsigned batch = 0;
cMessage *msg;
while (true)
   {
    batch++;
    if(batch == YAWNS_BATCH) {
       batch = 0;
       tw_gvt_step1(); //ROSS
       tw_gvt_step2(); //ROSS
    }
    if(GVT == YAWNS_ENDRUN)
       return NULL;
    if( GVT > endOfTime )
       return NULL;
```

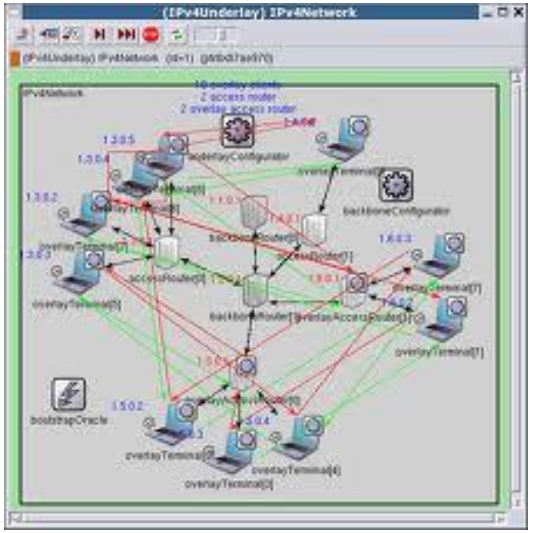
```
msg=sim->msgQueue.peekFirst();
if (!msg) continue;
if (msg->getArrivalTime() >
     GVT + LA)
   {
    batch = YAWNS_BATCH - 1;
    continue;
    }
 return msg;
} // end while

return msg;
```

Porting OMNeT++ to IBM Blue Gene/Q

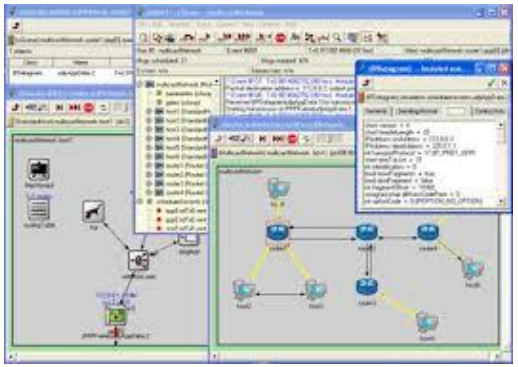


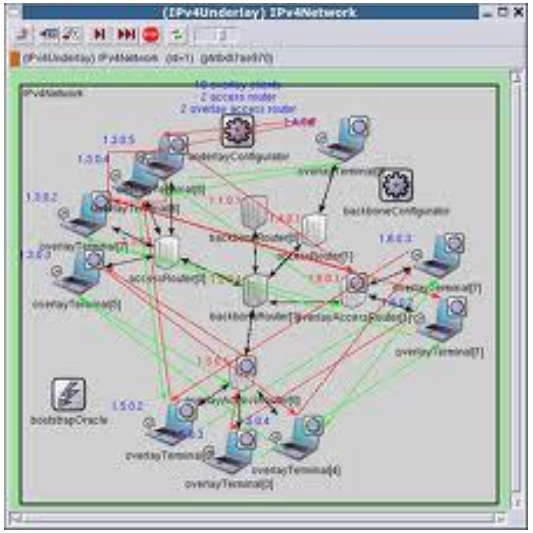
- Run ./configure on standard Linux system
  - OMNeT ./configure will not complete on BG/Q
- Move OMNeT++ repo to Blue Gene/Q front end
- Build flex, bison, libXML, Sqllite3 and zlib for BG/Q.
- Turn off TCL/TK
- Edit Makefile.in for BG/Q
  - Switch to IBM XLC compiler from GCC
  - Flags: `-O3 -qhot -qpic=large -qstrict -qarch=qp -qtune=qp -qmaxmem=-1 -DHAVE_SWAPCONTEXT -DHAVE_PCAP -DWITH_PARSIM -DWITH_MPI -DWITH_NETBUILDER`
- Discovered connection of remote gates create MPI failure at > 256 cores



Re-write of cParsimPartition::connectRemoteGates()

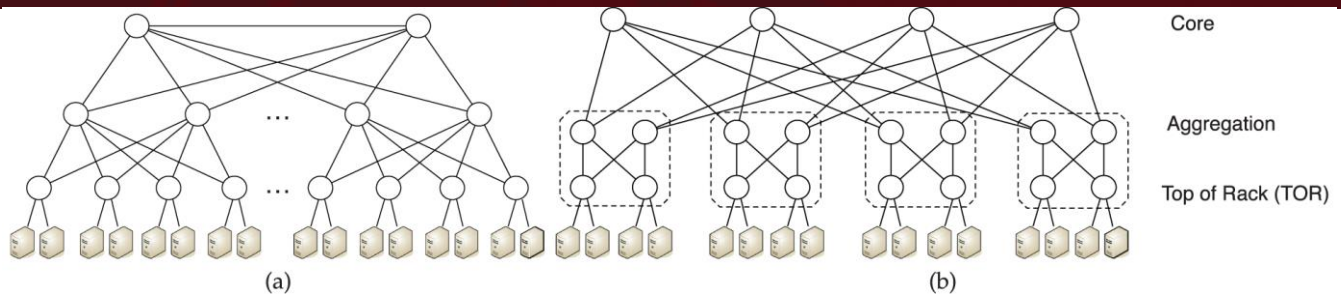
- **Original Algorithm:** each MPI rank would send a point-2-point message to all other ranks with list of cGate objects
- **Failure mode:** would result in each MPI rank needing to dedicated GBs of RAM to MPI internal memory for message data handling.
- **MPI on BG/Q** was not intended to be used this way @ larger rank counts
- **Re-write approach:** Let each MPI rank use MPI_Bcast to send it's cGate object data to all other rank.
- **Other mod:** use gate index and not name to look-up gate object on receivers sides.
  - **Improved Performance by 6x**





At 2K MPI ranks, takes about ~30 mins to init a 64K node network model

Venus Network Model Configuration



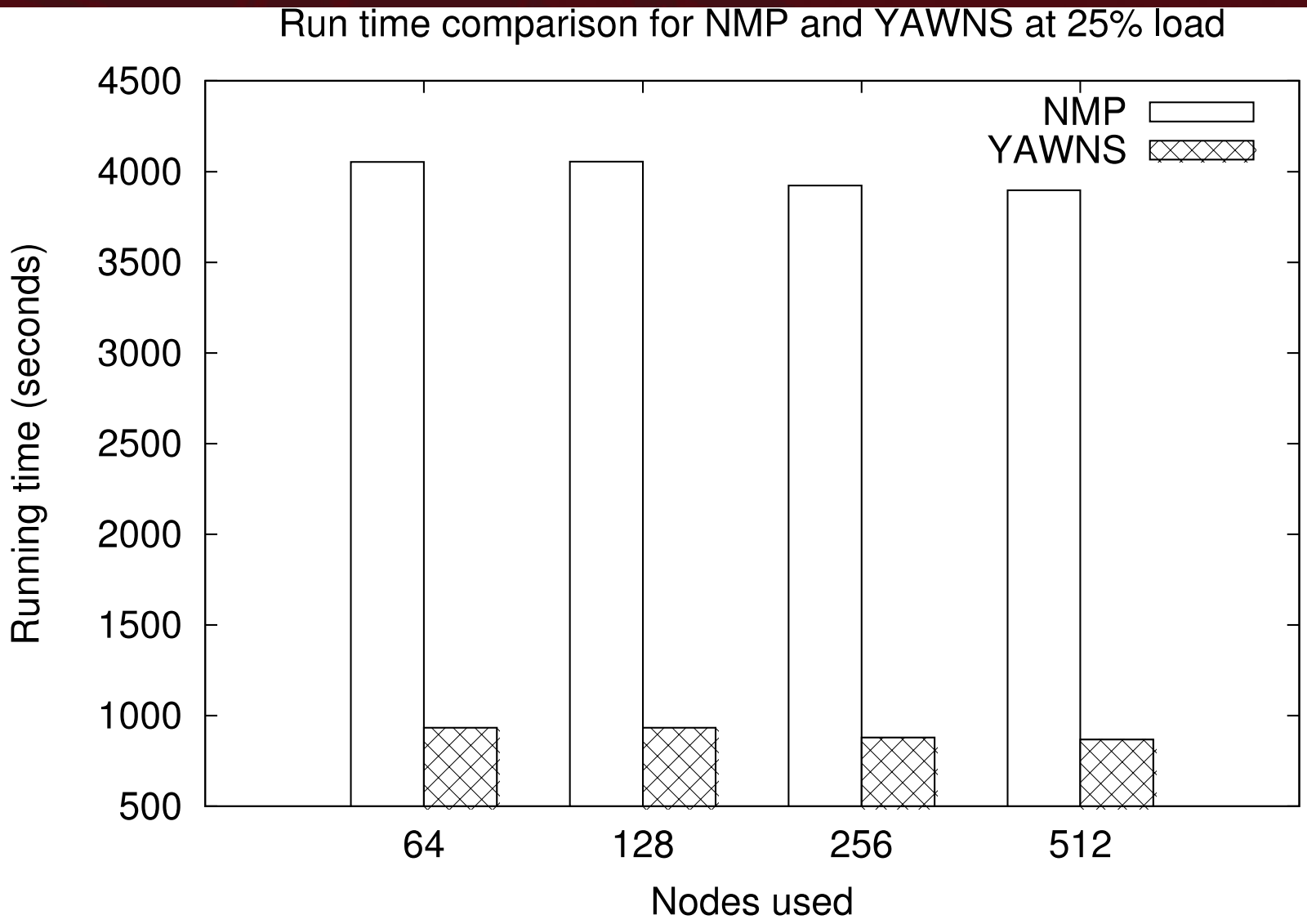
- 65,536 node Fat Tree, 3 levels, double sided
  - 64 ports switches
  - 2K switches @ L1 and L2, 1K @ L3 , 5120 switches total
- Random nearest neighbor traffic
  - 25%, 50%, 80% max injection workload
- **Link bandwidth:** 50 GB/sec
- **Link delay:** 10.2 ns
- Network adaptor and switch delays: 100 ns
- Sim time: 120 us
- **Routing:** DModK
- **Serial Platform:** AMD Opteron 6272, 2.1 GHz , 512 GB RAM
- **Parallel Platform:** "AMOS" 5-rack BG/Q system, 1K cores used

Validation of Venus Model in Parallel

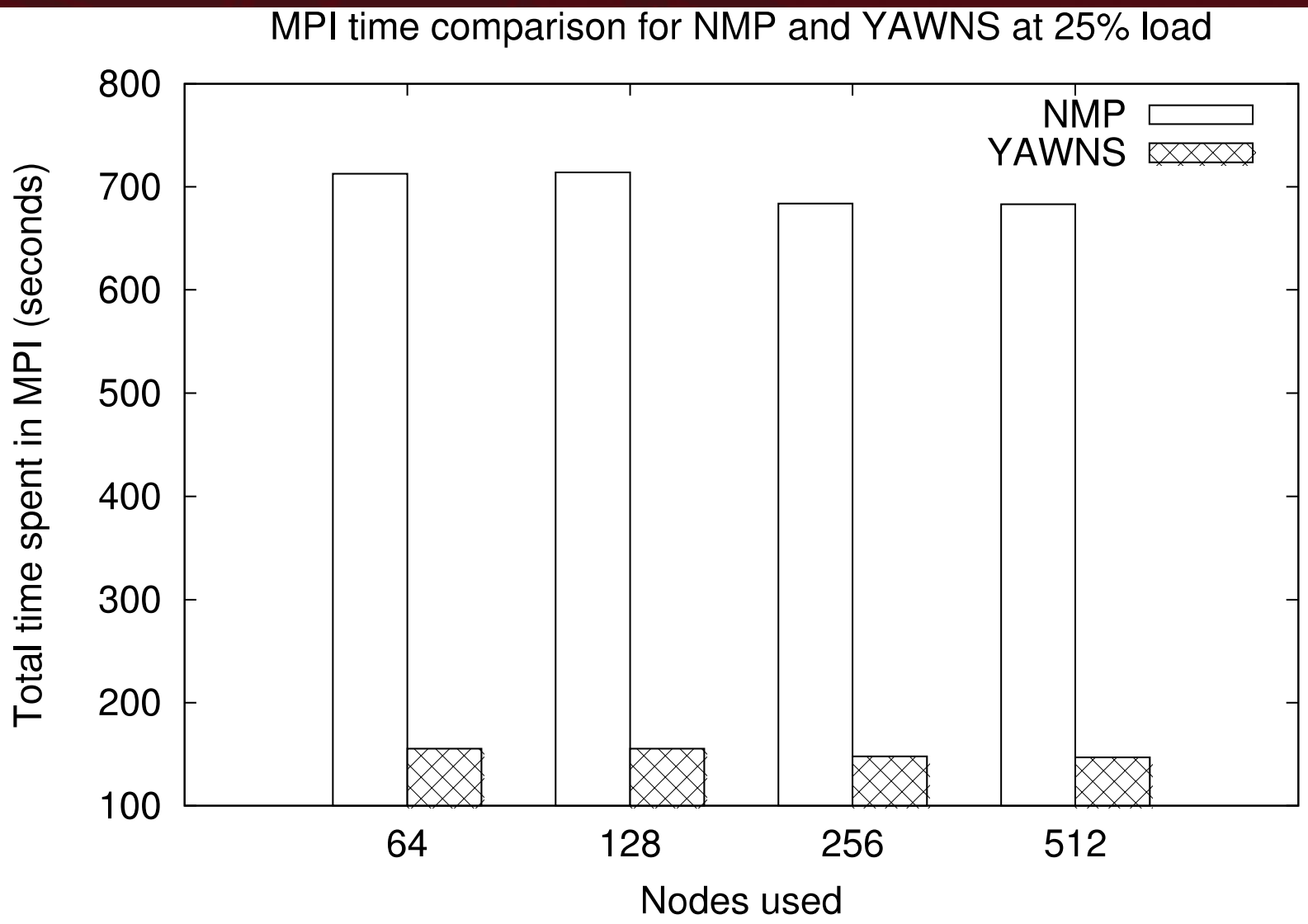
Model Output Stats	Serial	NMP	YAWNS
Total packets processed	507,666,409	507,664,860	507,665,150
Throughput (pkts/sec)	$5.09353e+12$	$5.09351e+12$	$5.09351e+12$
Max injection rate estimate	79.3229	79.3226	79.3227
Num. of pkts with 1 hop	241,246	241,753	241,755
Num. of pkts with 3 hops	7,702,089	7,707,681	7,707,676
Num. of pkts with 5 hops	499,723,074	499,715,426	499,715,719

Table 2: Comparison of serial and parallel run key output statistics for 80% traffic load configuration. The serial column denotes output statistics from the single processor execution of the Venus network model. The NMP column denotes output statistics from a parallel execution of the Venus network model using the Null Message Protocol (NMP). The YAWNS column denotes output statistics from a parallel execution of the Venus network model using the YAWNS protocol. All parallel runs were made using 128 BG/Q nodes, 8 cores/MPI ranks per node.

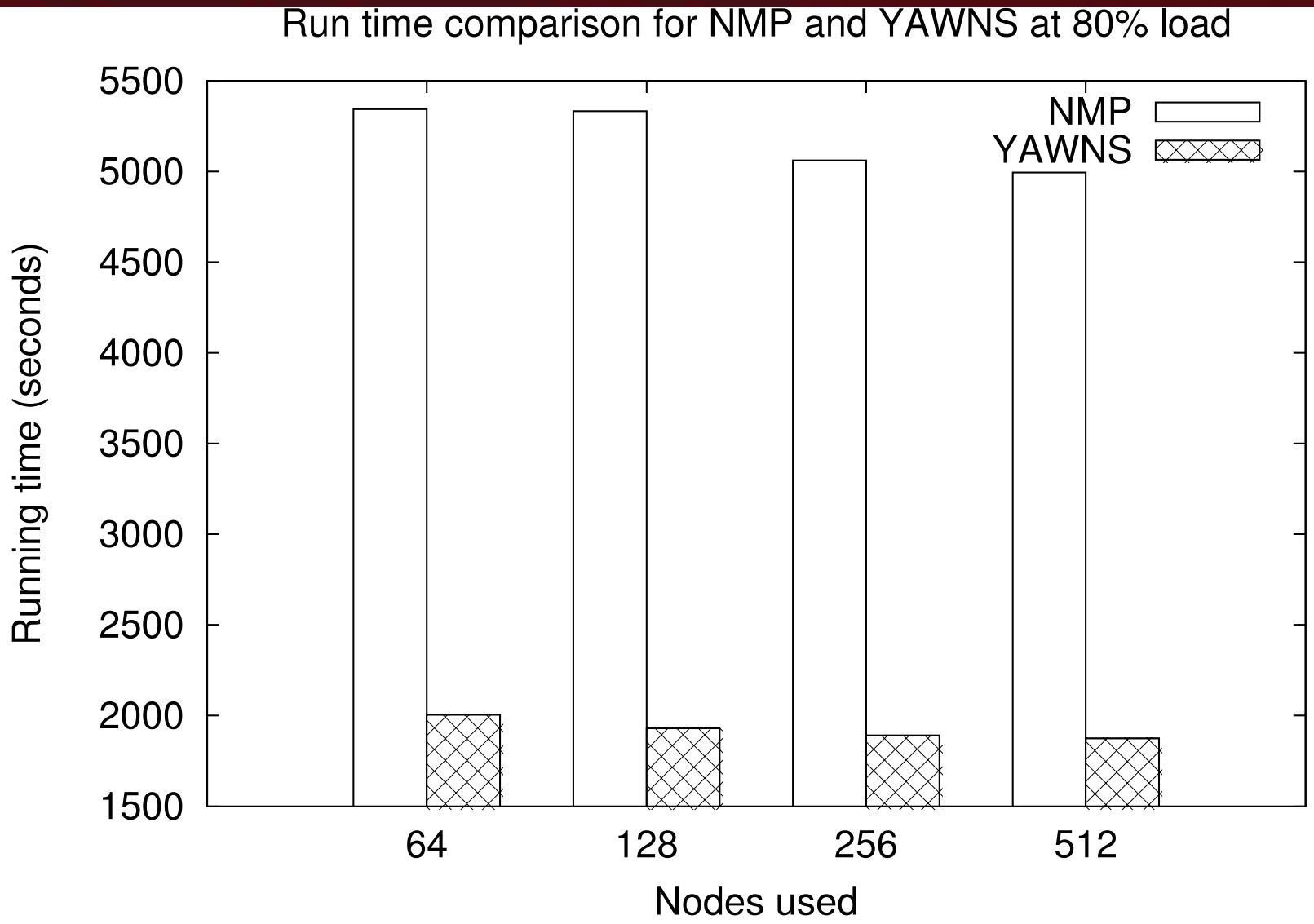
Run Time: YAWNS vs. NMP @ 25% Workload



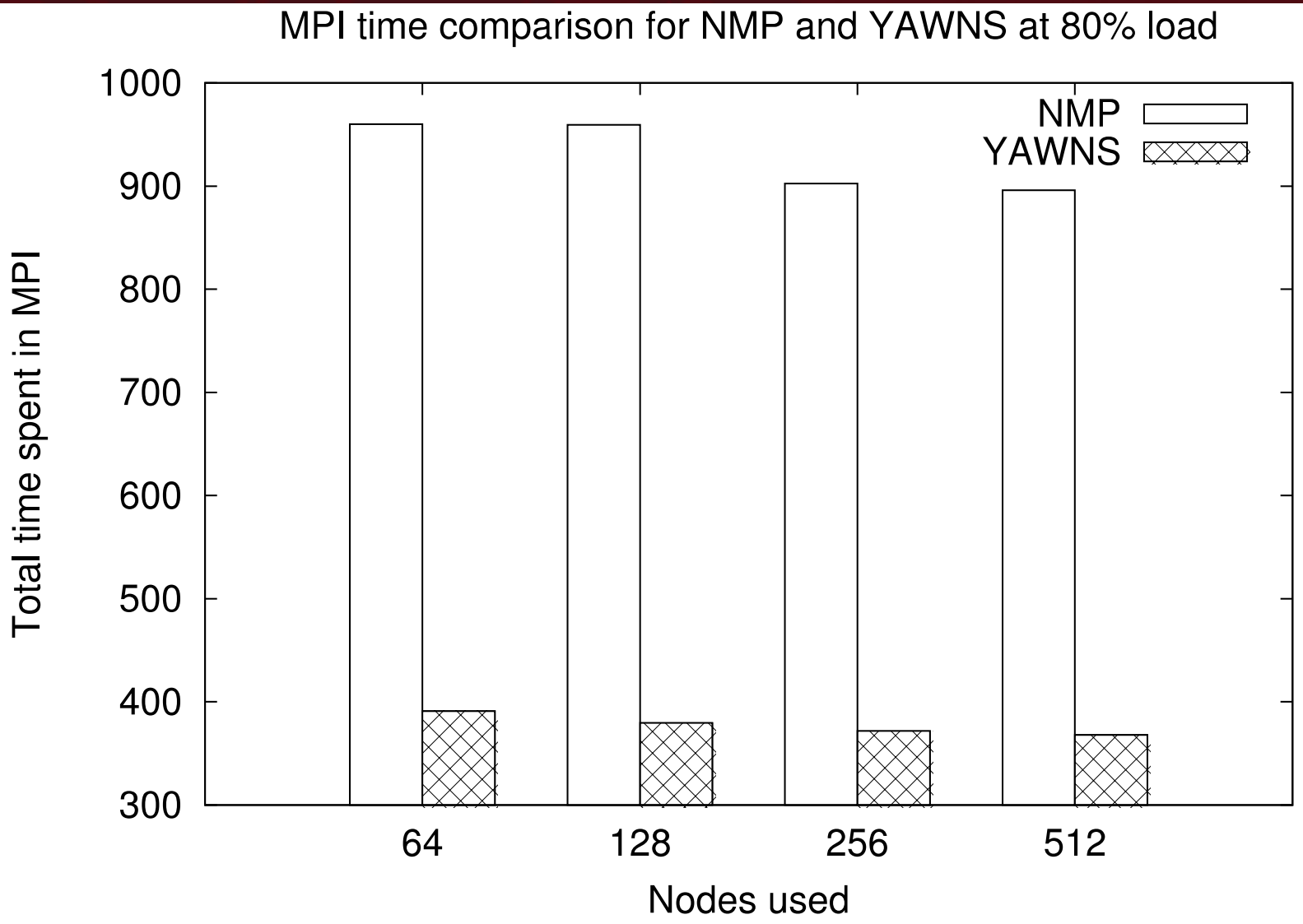
MPI Time: YAWNS vs. NMP @ 25% Workload



Run Time: YAWNS vs. NMP @ 80% Workload



MPI Time: YAWNS vs. NMP @ 80% Workload



Speedup of YAWNS on BG/Q vs. AMD Server

Traffic Load	Serial Exec. Time	YAWNS Exec. Time	Speedup
50%	70,038 secs	1,340 secs on 256 nodes	52.27
80%	119,926 secs	1,875 secs on 512 nodes	63.96

Table 3: Overall parallel speedup for YAWNS on 1024 cores for 50% and 80% traffic load scenarios. The YAWNS execution time is the shortest execution time among 1024 MPI rank runs using 64, 128, 256 and 512 nodes. All times are shown in seconds and reports the pure simulation execution times not including any time spent in model initialization or setup.

Future Work

- Ensure YAWNS works with all uses of OMNeT++ exceptions
  - Still a work-in-progress
- Modify OMNeT++ MPI layer to use non-blocking MPI send/recv operations
- Enable MPI ranks to be “idle” processes to support wider range of network configurations and parallel partitions (e.g. a 13824 node network does not map well to 1024 BG/Q cores)
- Conduct detailed performance study of YAWNS on:
  - Changes in topology
  - Changes in topology size/scale
  - Changes in network partitioning
  - Changes in model lookahead
- Release YAWNS implementation as open source