BigSim Tutorial

Presented by
Eric Bohm

Charm++ Workshop 2008
Parallel Programming Laboratory
University of Illinois at Urbana-Champaign

Outline

- Overview
- BigSim Emulator
- Charm++ on the Emulator
- Simulation framework
  - Post-mortem simulation
  - Trace log transformation
  - Network simulation
- Performance analysis/visualization

Simulation-based Performance Prediction

- Extremely large parallel machines are being built with enormous compute power
  - Very large number of processors with petaflops level peak performance
- Are existing software environments ready for these new machines?
  - How to write a peta-scale parallel application?
  - What will be the performance like? Can these applications scale?

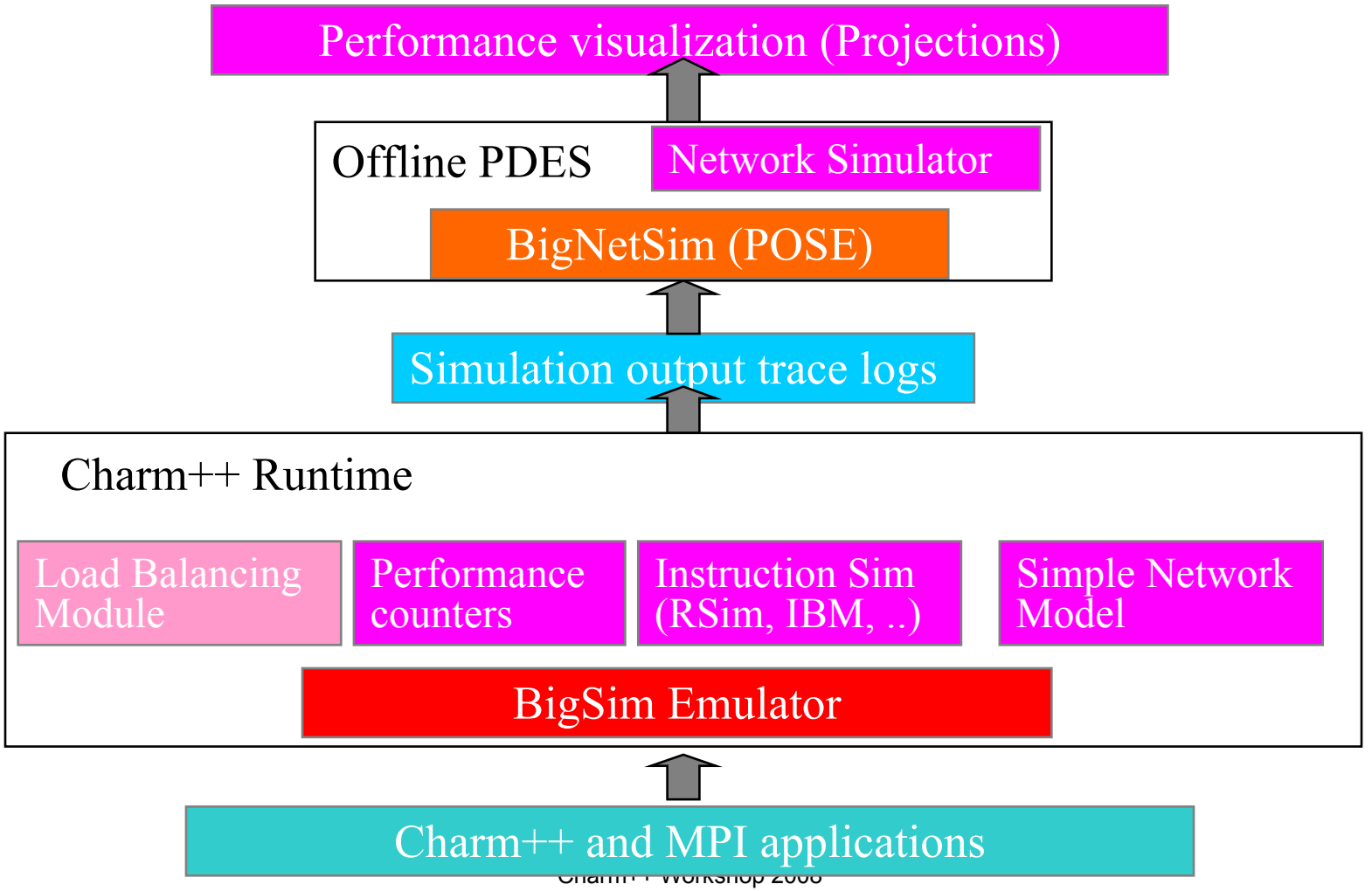
BigSim Simulation Toolkit

- BigSim emulator
  - Standalone emulator API
  - Charm++ on emulator
- BigSim Trace Interpolator
- BigSim simulator
  - Network simulator

Simulation-based Performance Prediction

- With focus on Charm++ and AMPI programming models
- Performance prediction is based on Parallel Discrete Event Simulation (PDES)
- Simulation is challenging, aims at different levels of fidelity
  - Processor prediction
  - Network prediction
- Two approaches
  - Direct execution (online mode)
  - Trace-driven (post-mortem mode)

Architecture of BigSim (postmortem mode)



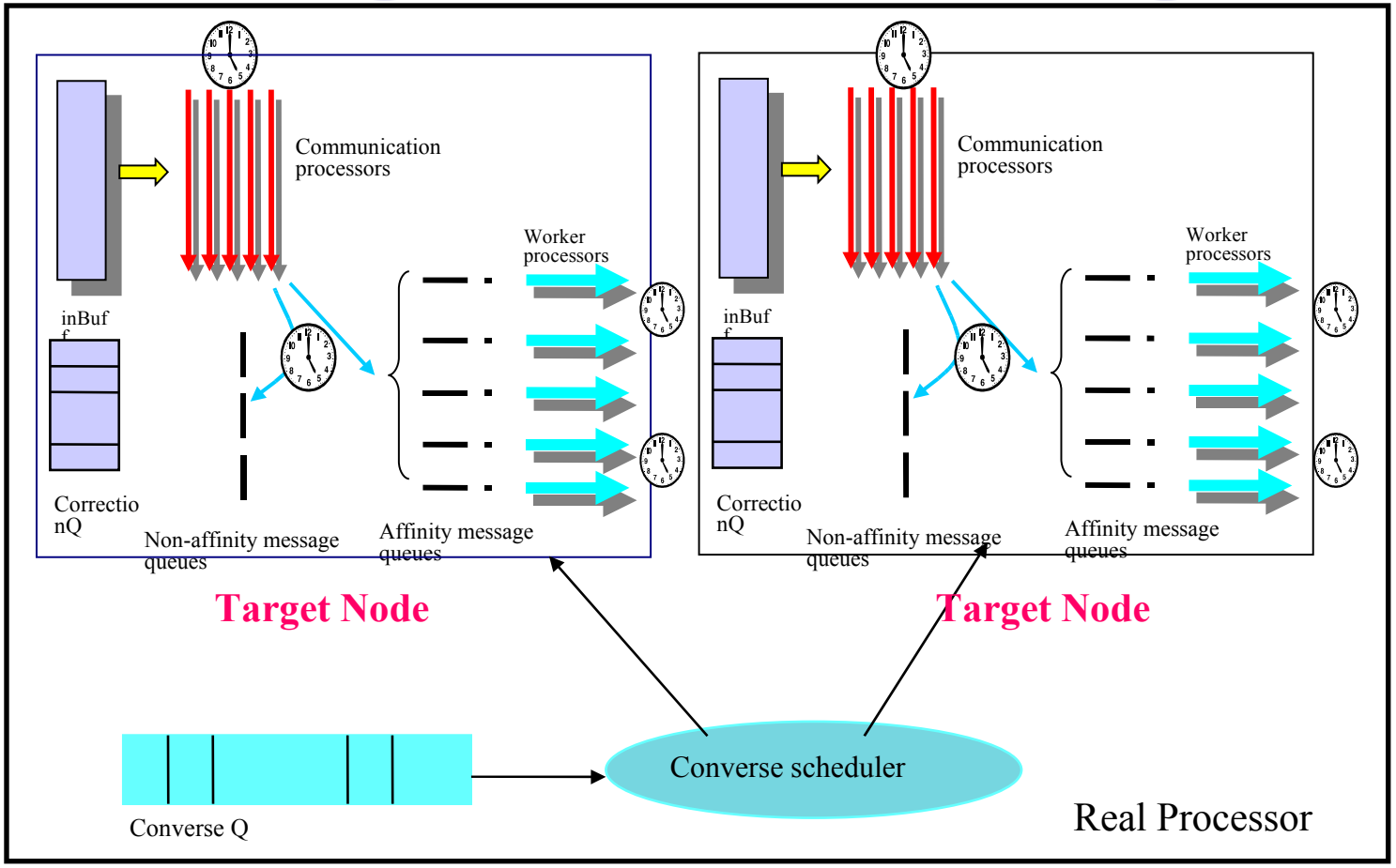
Outline

- Overview
- BigSim Emulator
- Charm++ on the Emulator
- Simulation framework
  - Online mode simulation
  - Post-mortem simulation
  - Network simulation
- Performance analysis/visualization

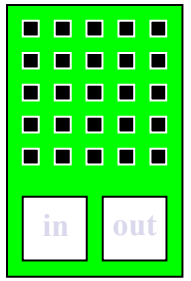
Emulator

- Emulate full machine on existing parallel machines
  - Actually run a parallel program with multi-million way parallelism
- Started with mimicking Blue Gene/C low level API
- Machine layer abstraction
  - Many multiprocessor (SMP) nodes connected via message passing

BigSim Emulator: functional view



BigSim Programming API



- Machine initialization
  - Set/get machine configuration
  - Get node ID: (x, y, z)
- Message passing
  - Register handler functions on node
  - Send packets to other nodes (x,y,z) with a handler ID

User's API

- BgEmulatorInit(), BgNodeStart()
- BgGetXYZ()
- BgGetSize(), BgSetSize()
- BgGetNumWorkThread(), BgSetNumWorkThread()
- BgGetNumCommThread(), BgSetNumCommThread()
- BgGetNodeData(), BgSetNodeData()
- BgGetThreadID(), BgGetGlobalThreadID()
- BgGetTime()
- BgRegisterHandler()
- BgSendPacket(), etc
- BgShutdown()

Examples

- charm/examples/bigsim/emulator
  - ring
  - jacobi3D
  - maxReduce
  - prime
  - octo
  - line
  - littleMD

BigSim application example - Ring

```
typedef struct {
  char core[CmiBlueGeneMsgHeaderSizeBytes];
  int data;
} RingMsg;

void BgNodeStart(int argc, char **argv)  {
     int x,y,z, nx, ny, nz;
    BgGetXYZ(&x, &y, &z);          nextxyz(x, y, z, &nx, &ny, &nz);
     if (x == 0 && y==0 && z==0)     {
          RingMsg msg = new RingMsg;                          msg->data =  888;
          BgSendPacket(nx, ny, nz, passRingID, LARGE_WORK, sizeof(RingMsg), (char *)msg);
     }
}

void passRing(char *msg)  {
     int x, y, z,  nx, ny, nz;
     BgGetXYZ(&x, &y, &z);           nextxyz(x, y, z, &nx, &ny, &nz);
     if (x==0 && y==0 && z==0)     if (++iter == MAXITER) BgShutdown();
     BgSendPacket(nx, ny, nz, passRingID, LARGE_WORK, sizeof(RingMsg), msg);
}
```

Emulator Compilation

- Emulator libraries implemented on top of Converse/machine layer:
  - libconv-bigsim.a
  - libconv-bigsim-logs.a
- Compile with normal Charm++ with "bigemulator" target
  - *./build bigemulator net-linux*
- Compile an application with emulator API
  - *charmc -o ring ring.C -language bigsim*

Execute Application on the Emulator

- Define machine configuration
  - Function API
    - BgSetSize(x, y, z), BgSetNumWorkThread(), BgSetNumCommThread()
  - Command line options
    - *+x +y +z*
    - *+cth +wth*
    - E.g.
      - *charmrun +p4 ring +x10 +y10 +z10 +cth2 +wth4*
  - Config file
    - *+bgconfig config*

Running with bgconfig file

- **+bgconfig ./bg_config**

```
x  10
y  10
z  10
cth 2
wth 4
stacksize  4000
timing  walltime
#timing  bgelapse
#timing  counter
#cpufactor   1.0
fpfactor    5e-7
traceroot     /tmp
log      yes
correct  no
network   bluegene
```

Ring Output

```
clarity>./ring 2 2 2 2 2
Charm++: standalone mode (not using charmrun)
BG info> Simulating 2x2x2 nodes with 2 comm + 2 work threads each.
BG info> Network type: bluegene.
alpha: 1.000000e-07     packetsize: 1024        CYCLE_TIME_FACTOR:1.000000e-03.
CYCLES_PER_HOP: 5       CYCLES_PER_CORNER: 75.
0 0 0 => 0 0 1
0 0 1 => 0 1 0
0 1 0 => 0 1 1
0 1 1 => 1 0 0
1 0 0 => 1 0 1
1 0 1 => 1 1 0
1 1 0 => 1 1 1
1 1 1 => 0 0 0

BG> BlueGene emulator shutdown gracefully!
BG> Emulation took 0.000265 seconds!
Program finished.
```

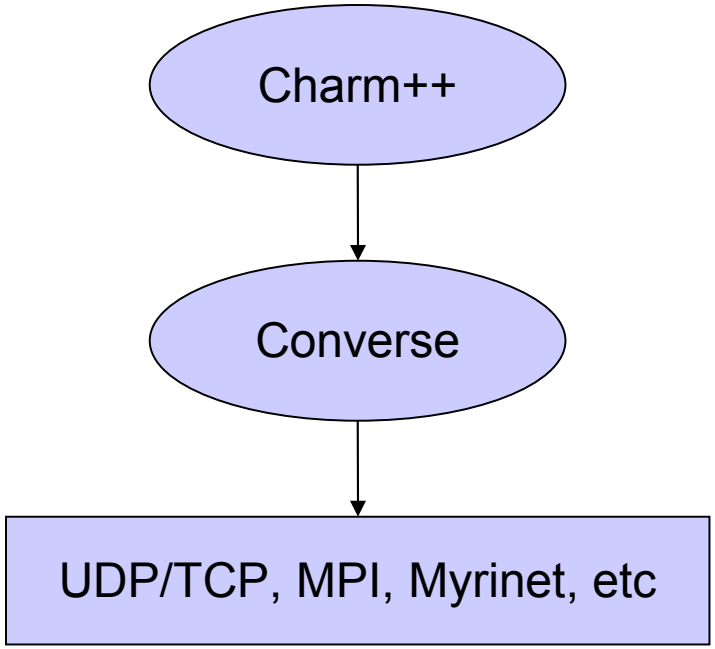
Outline

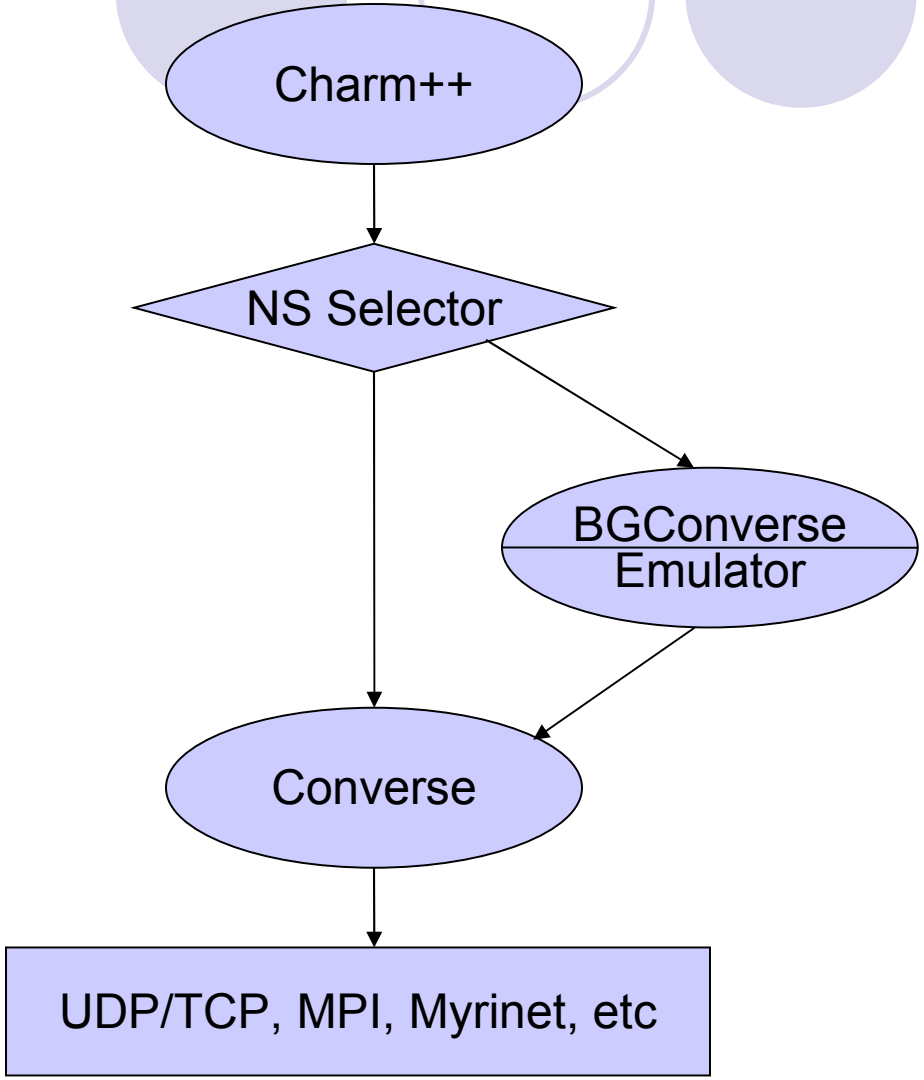
- Overview
- BigSim Emulator
- Charm++ on the Emulator
- Simulation framework
  - Online mode simulation
  - Post-mortem simulation
  - Network simulation
- Performance analysis/visualization

BigSim Charm++/AMPI

- Charm++/AMPI implemented on top of BigSim emulator, using it as another machine layer
- Support frameworks and libraries
  - Load balancing framework
  - Communication optimization library (comlib)
  - FEM
  - Multiphase Shared Array (MSA)

BigSim Charm++





Build Charm++ on BigSim

- Compile Charm++ on top of BigSim emulator
  - Build option "bigemulator"
  - E.g.
    - Charm++:

    ./build charm++ net-linux bigemulator
    - AMPI:

    ./build AMPI net-linux bigemulator

    (use net-linux-amd64 on opteron or x86_64)

Running Charm++/AMPI Applications

- Compile Charm++/AMPI applications
  - Same as normal Charm++/AMPI
  - Just use charm/net-linux-bigsim/bin/charmc
- Running BigSim Charm++ applications
  - Same as running on emulator
    - Use command line option, or
    - Use bgconfig file

Example – AMPI Cjacobi3D

- *cd charm/net-linux-bigemulator/examples/ampi/Cjacobi3D*
- Make
  - *charmc -o jacobi jacobi.o -language ampi -module EveryLB*

./charmrun +p2 ./jacobi 2 2 2 +vp8 +bgconfig ~/bg_config +balancer GreedyLB +LBDebug 1

```
[0] GreedyLB created
iter 1 time: 1.022634 maxerr: 2020.200000
iter 2 time: 0.814523 maxerr: 1696.968000
iter 3 time: 0.787009 maxerr: 1477.170240
iter 4 time: 0.825189 maxerr: 1319.433024
iter 5 time: 1.093839 maxerr: 1200.918072
iter 6 time: 0.791372 maxerr: 1108.425519
iter 7 time: 0.823002 maxerr: 1033.970839
iter 8 time: 0.818859 maxerr: 972.509242
iter 9 time: 0.826524 maxerr: 920.721889
iter 10 time: 0.832437 maxerr: 876.344030
[GreedyLB] Load balancing step 0 starting at 11.647364 in PE0
n_obj:8 migratable:8 ncom:24
GreedyLB: 5 objects migrating.
[GreedyLB] Load balancing step 0 finished at 11.777964
[GreedyLB] duration 0.130599s memUsage: LBManager:800KB CentralLB:0KB
iter 11 time: 1.627869 maxerr: 837.779089
iter 12 time: 0.951551 maxerr: 803.868831
iter 13 time: 0.960144 maxerr: 773.751705
iter 14 time: 0.952085 maxerr: 746.772667
iter 15 time: 0.956356 maxerr: 722.424056
iter 16 time: 0.965365 maxerr: 700.305763
iter 17 time: 0.947866 maxerr: 680.097726
iter 18 time: 0.957245 maxerr: 661.540528
iter 19 time: 0.961152 maxerr: 644.421422
iter 20 time: 0.960874 maxerr: 628.564089

BG> Bigsim mulator shutdown gracefully!
BG> Emulation took 36.762261 seconds!
```

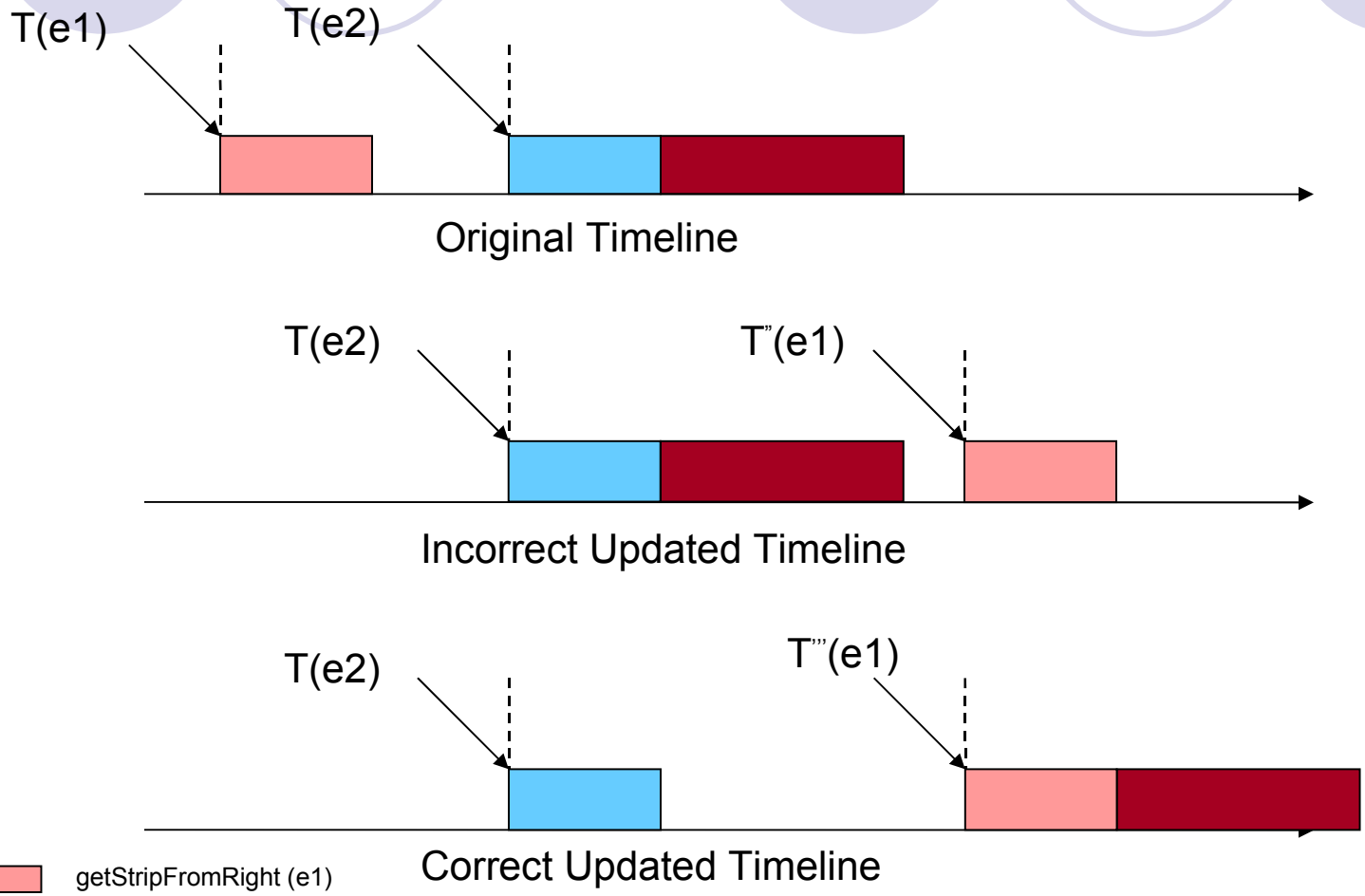
Performance Prediction

- How to predict performance?
  - Different levels of fidelity
  - Sequential portion:
    - User supplied timing expression
    - Wall clock time
    - Performance counters
    - Instruction level simulation
  - Message passing:
    - Simple latency-based network model
    - Contention-based network simulation

How to Ensure Simulation Accuracy

- The idea:
  - Take advantage of inherent determinacy of an application
  - Don't need rollback - same user function then is executed only once
  - In case of out of order delivery, only timestamps of events are adjusted

Timestamp Correction (Jacobi1D)



Structured Dagger (Jacobi1D)

```
entry void jacobiLifeCycle()
{
  for (i=0; i<MAX_ITER; i++)
  {
    atomic {sendStripToLeftAndRight();}
    overlap
    {
      when getStripFromLeft(Msg *leftMsg)
         { atomic { copyStripFromLeft(leftMsg); } }
      when getStripFromRight(Msg *rightMsg)
         { atomic { copyStripFromRight(rightMsg); } }
    }
    atomic{ doWork(); /* Jacobi Relaxation */ }
  }
}
```

Sequential time - BgElapse

- **BgElapse**

```
entry void jacobiLifeCycle()
{
  for (i=0; i<MAX_ITER; i++)
  {
    atomic {sendStripToLeftAndRight();}
    overlap
    {
      when getStripFromLeft(Msg *leftMsg)
         { atomic { copyStripFromLeft(leftMsg); } }
      when getStripFromRight(Msg *rightMsg)
         { atomic { copyStripFromRight(rightMsg); } }
    }
    atomic{ doWork(); BgElapse(10e-3);}
  }
}
```

Sequential Time – using Wallclock

- Wallclock measurement of the time can be used via a suitable multiplier (scale factor)
- Run application with +bgwalltime and +bgcpufactor, or
- +bgconfig ./bgconfig:

  timing walltime

  cpufactor 0.7
- Good for predicting a larger machine using a fraction of the machine

Sequential Time – performance counters

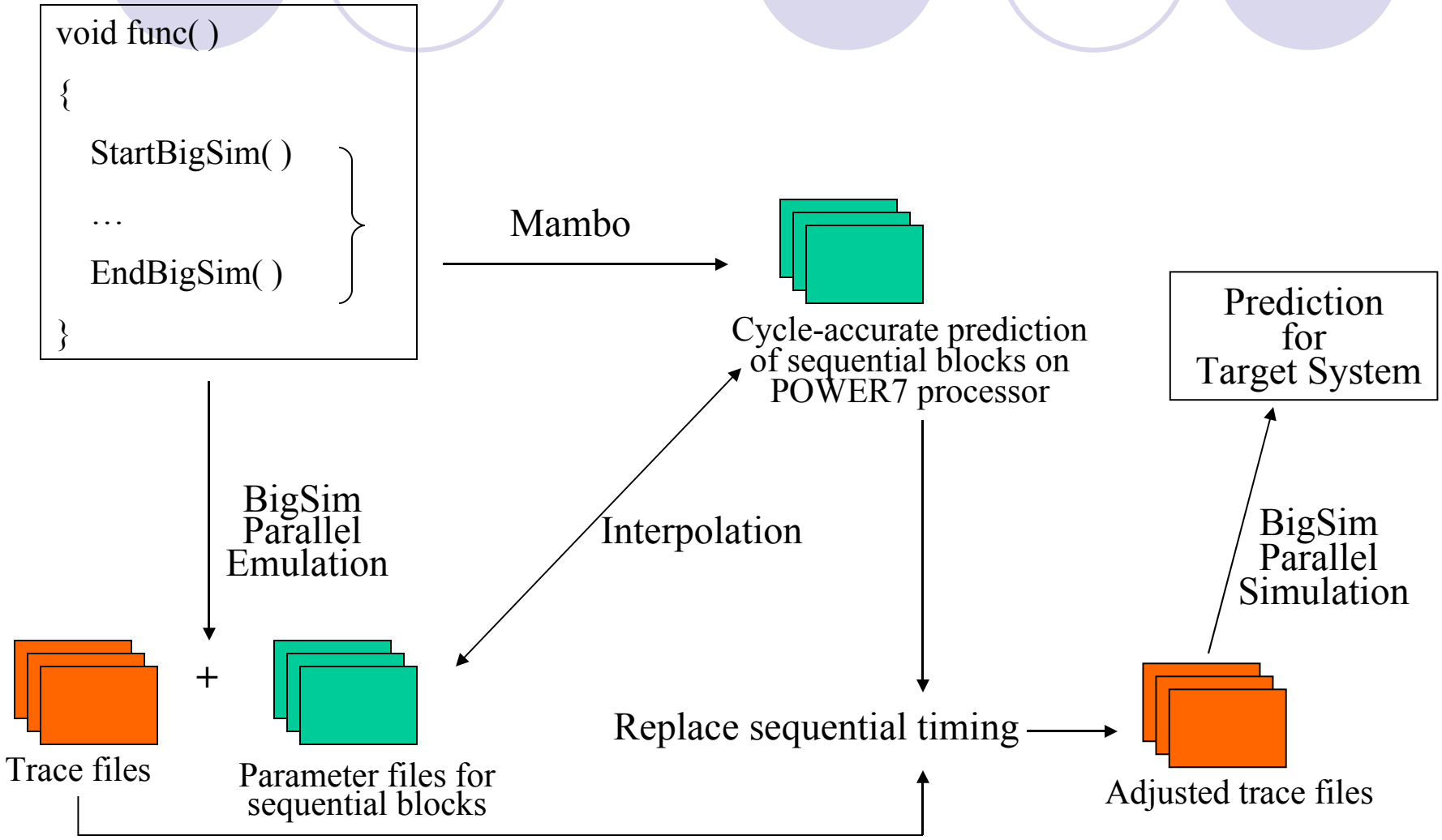
- Count floating-point, integer, memory and branch instructions (for example) with hardware counters
  - with a simple heuristic, use the expected time for each of these operations on the target machine to give the predicted total computation time.
- Cache performance and the memory footprint effects can be approximated by percentage of memory accesses and cache hit/miss ratio.
- Perfex and PAPI are supported
- Example of use, for a floating-point intensive code:

```
+bgconfig ./bg_config
 timing  counter
 fpfactor    5e-7
```

Sequential Time – Instruction level simulation

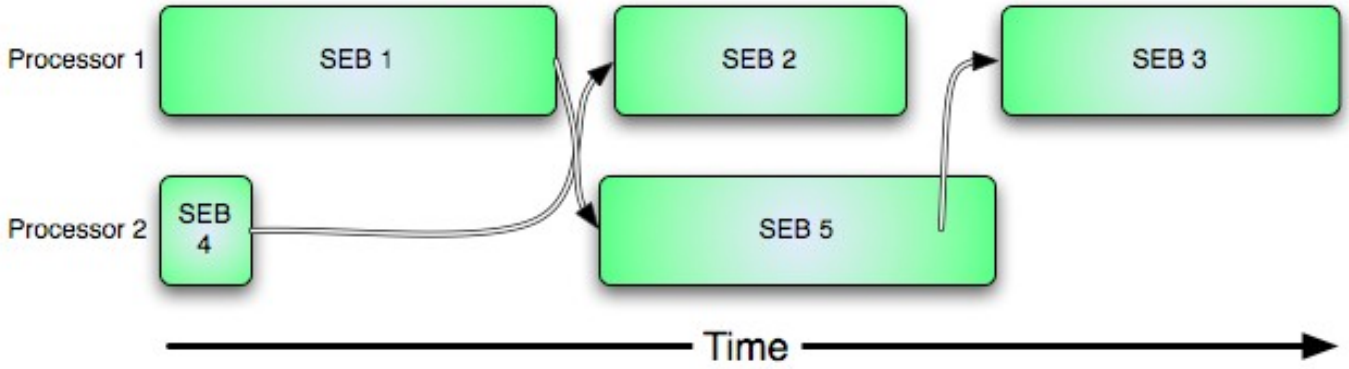
- Run instruction-level simulator separately to get accurate timing information (sampling)
- An interpolation-based scheme
  - Use result of a smaller scale instruction level simulation to interpolate for large dataset
    - do a least-squares fit to determine the coefficients of an approximation polynomial function

Case study: BigSim / Mambo

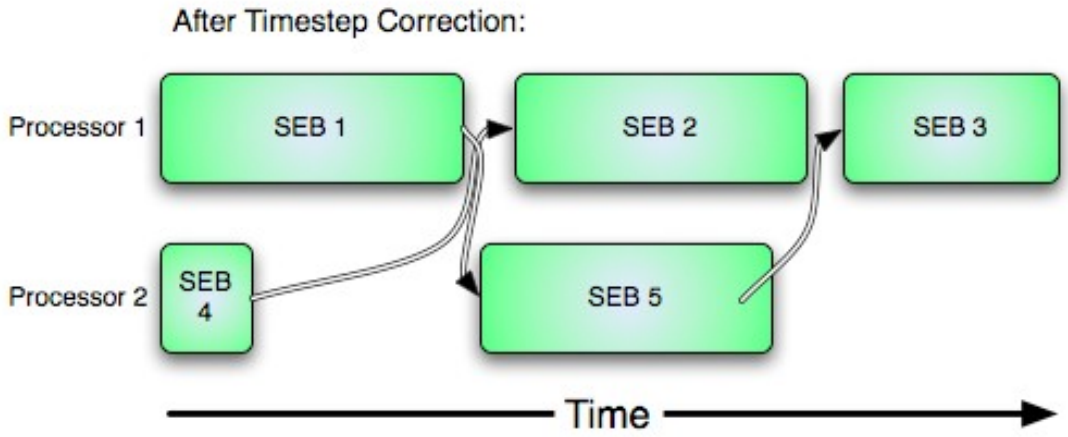


Interpolation Tool Rewrites SEB Durations

Traces from existing machine

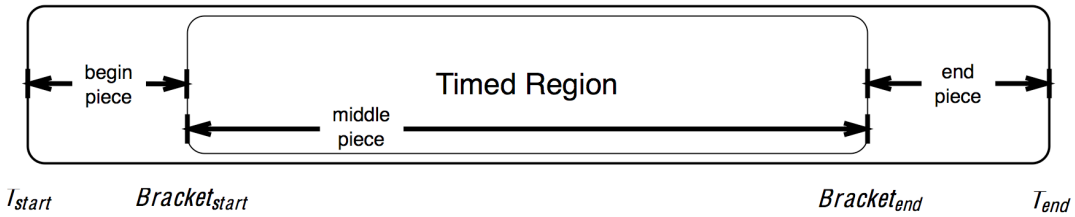


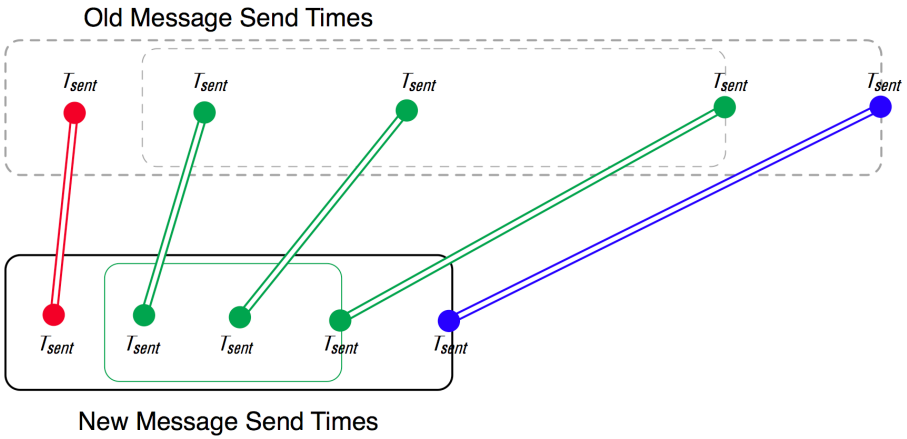
Traces adapted to match another machine



Interpolation Tool Rewrites SEB Durations

- Replace the duration of a portion of each SEB with known exact times recorded in a execution or cycle-accurate simulator
- Scale begin/end portions by a constant factor
- Message send points are linearly mapped into the new times





Using interpolation tool

- Compile interpolation tool
  - Install GSL, the GNU Scientific Library
  - cd charm/examples/bigsim/tools/rewritelog
  - Modify the file interpolatelog.C to match your particular tastes.
    OUTPUTDIR specifies a directory for the new logfiles
    CYCLE_TIMES_FILE specifies the file which contains accurate timing information
  - Make
- Modify source code
  - Insert startTraceBigSim() call before a compute kernel. Add an endTraceBigSim() call after the kernel. Currently the first call takes between 0 and 20 parameters describing the computation.

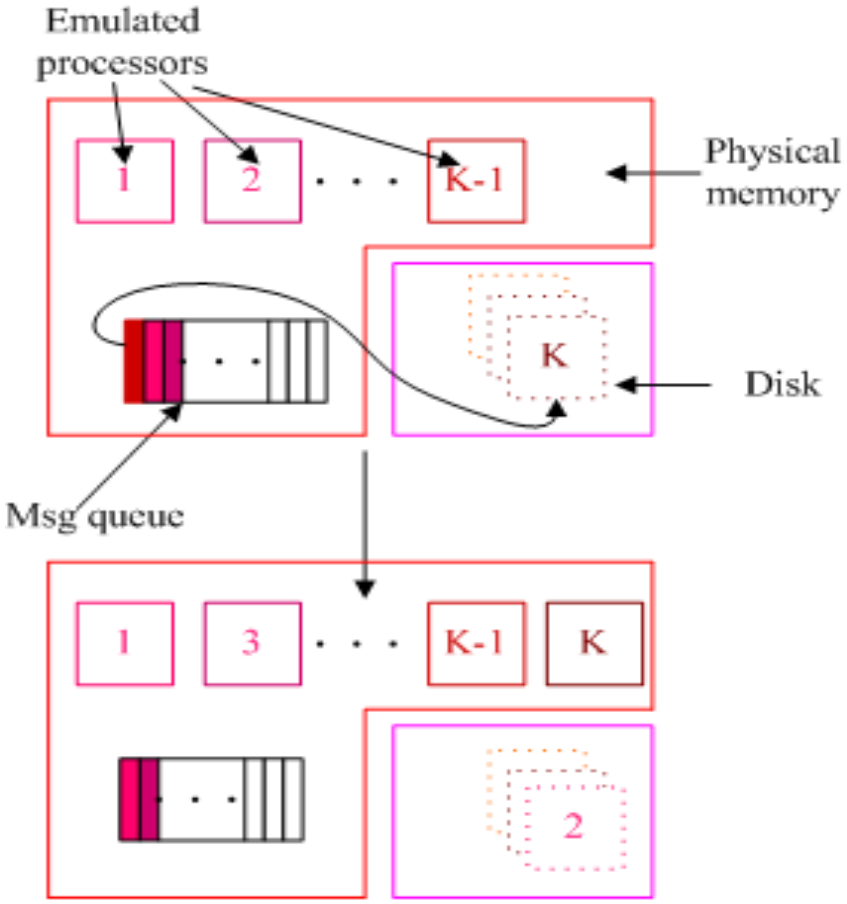
```
startTraceBigSim(param1, param2, param3, …);
// Some serial computational kernel goes here
endTraceBigSim("EventName");
```

Using interpolation tool (cont.)

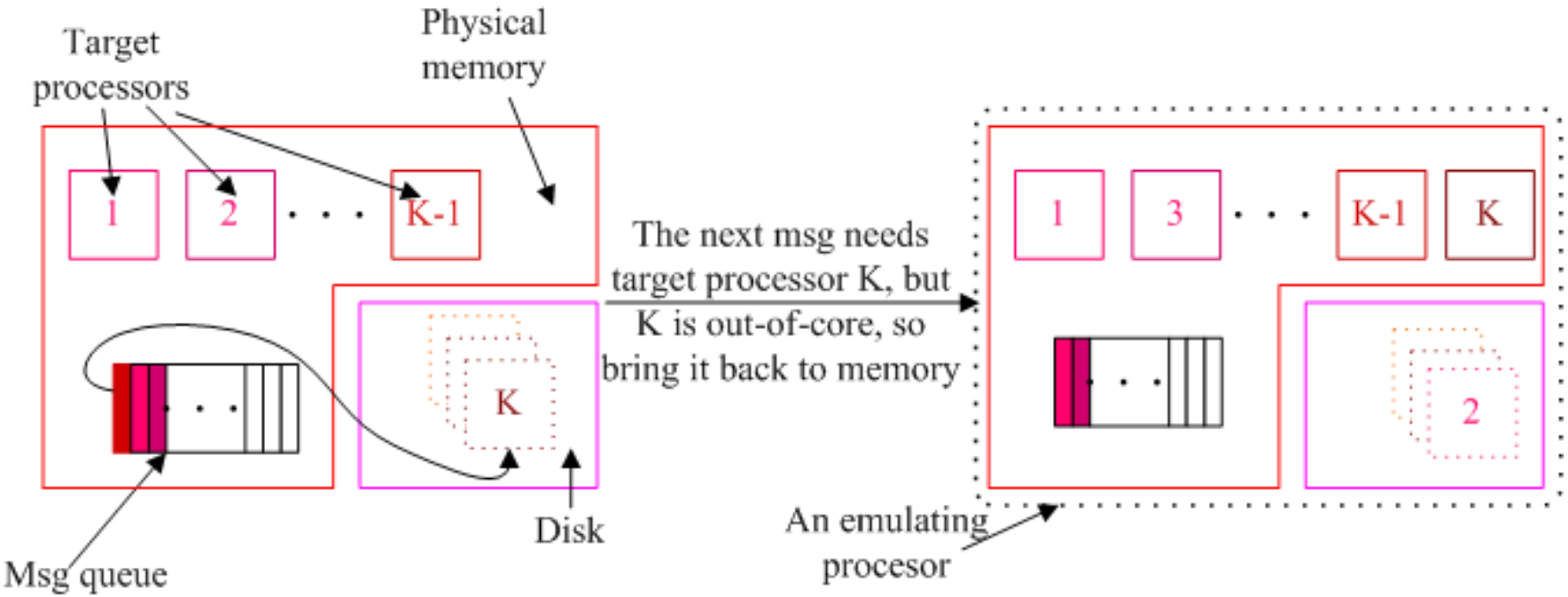
- Run the application through emulator, generating trace logs (bgTrace*)and parameter files (param.*)
- Run the same application with instruction-level simulator, get accurate timing indexed by parameters
- Run interpolation tool under bgTrace dir:
  - ./interpolatelog

Out-of-core Emulation

- Motivation
  - Physical memory is shared
  - VM system would not handle well
- Message driven execution
  - Peek msg queue => what execute next? (prefetch)



Overview of the idea



Options of basic schemes

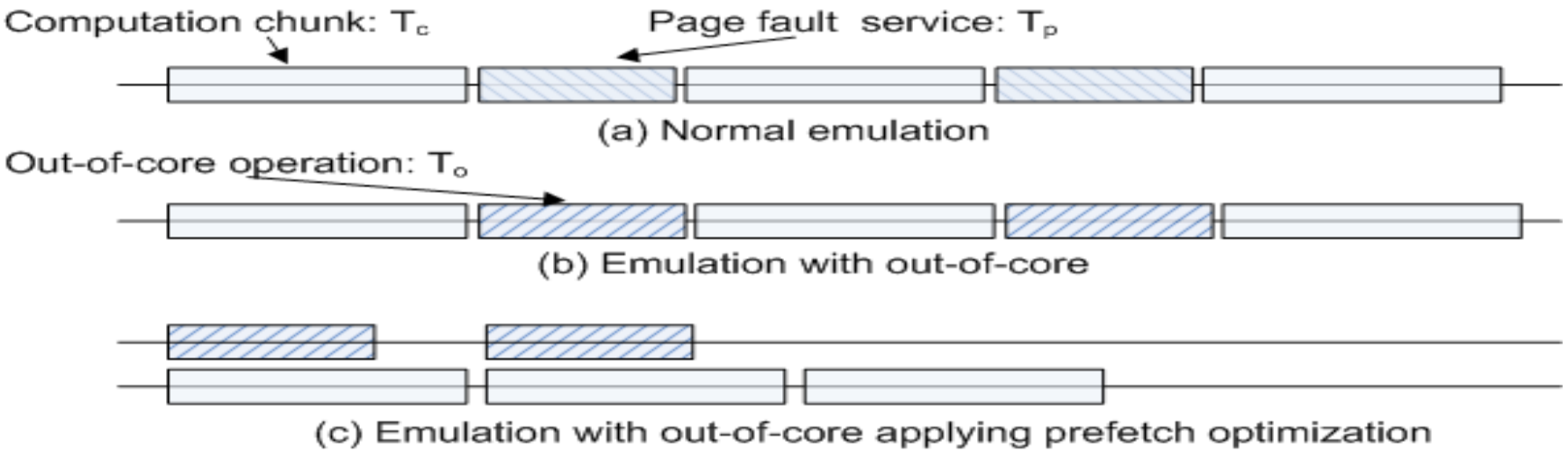
- Per message based
  - Swapping in/out a target processor for every message
- Multiple target processors based
  - Only allowing a fixed number of target processors in memory
- Memory based
  - Allowing as many target processors in memory as possible

Optimization for basic schemes

- Tuning eviction policy
  - Which processor to evict out?
- Applying prefetch
  - we know what will be the next message by peeking the message queue
  - How far we want to peek in the future?
  - Expected to gain most

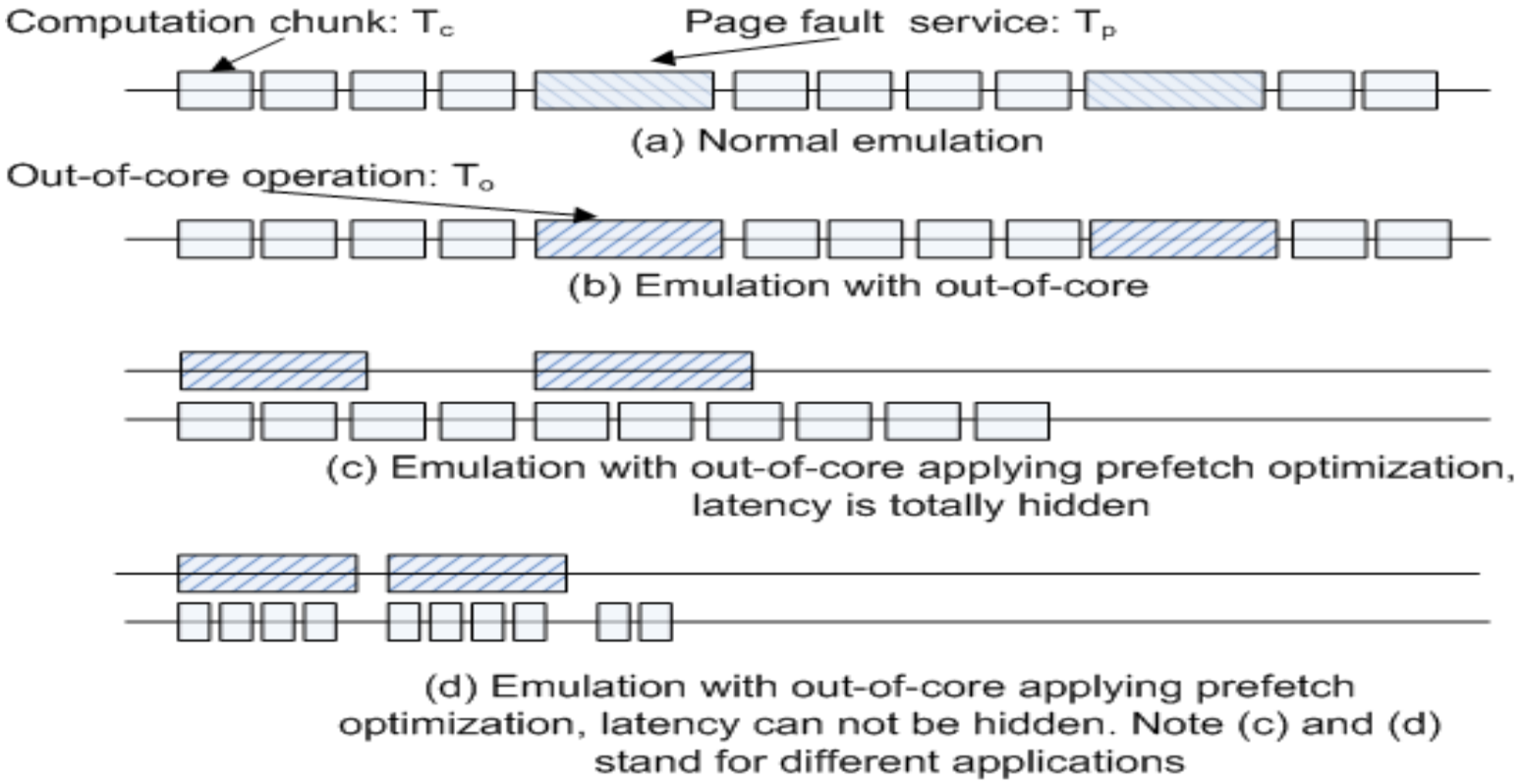
Two different scenarios (1)

- Per message triggers large chunk of computation



Two different scenarios (2)

- Per message triggers small chunk of computation



Using Out-of-core

- Compile an application with bigemulator
- Run the application through the emulator, and command line option:
  - *+ooc 512*

Simple Network Model

- No contention modeling
  - Latency and topology based
- Built-in network models for
  - Quadrics (Lemieux)
  - Blue Gene/C
  - Blue Gene/L

Choose Network Model at Run-time

- Command line option:
  - +*bgnetwork bluegenel*
- BigSim config file:
  - +*bgconfig ./bg_config*

    network bluegenel

How to Add a New Network Model

- Inherit from this base class defined in blue_network.h:

```
class BigSimNetwork
{
protected:
  double alpha;       // cpu overhead of sending a message
  char *myname;     // name of this network
public:
  inline double alphacost() { return alpha; }
  inline char *name() { return myname; }
  virtual double latency(int ox, int oy, int oz, int nx, int ny, int nz, int
    bytes) = 0;
  virtual void print() = 0;
};
```

How to Obtain Predicted Time

- BgGetTime()
  - Print to stdout is not useful actually
  - Because the printed time at execution time is not final.
  - Final timestamp can only be obtained after timestamp correction (simulation) finishes.

How to Obtain Predicted Time (cont.)

- BgPrint (char *)
  - Bookmarking events
  - E.g.

    BgPrint("start at %f\n");
  - Output to bgPrintFile.0 when simulation finishes
    - Look back these bookmarks
    - Replace "%f" with the committed time

Running Applications with Online Network Simulator

- Two modes
  - With simple network model (timestamp correction)
    - *+bgcorrect*
  - Partial prediction only (no timestamp correction)
    - *+bglog*
    - Generate trace logs for post-mortem simulation

With bgconfig

+bgconfig ./bg_config
x 64
y 32
z 32
cth 1
wth 1
stacksize 4000
timing walltime
#timing bgelapse
#timing counter
cpufactor 1.0
#fpfactor 5e-7
traceroot /tmp
log yes
correct no
network bluegene

BigSim Trace Log

- Execution of messages on each target processor is stored in trace logs (binary format)
  - named bgTrace[#], # is simulating processor number.
- Can be used for
  - Visualization/Performance study
  - Post-mortem simulation with different network models
- Loadlog tool
  - Binary to human readable ascii format conversion
  - *charm/examples/bigsim/tools/loadlog*

ASCII Log Sample

[22] 0x80a7a60 name:msgep (srcnode:0 msgID:21) ep:1
[[recvtime:0.000498 startTime:0.000498 endTime:0.000498]]
backward:
forward: [0x80a7af0 23]

[23] 0x80a7af0 name:Chunk_atomic_0 (srcnode:-1 msgID:-1) ep:0
[[recvtime:-1.000000 startTime:0.000498 endTime:0.000503]]
msgID:3 sent:0.000498 recvtime:0.000499 dstPe:7 size:208
msgID:4 sent:0.000500 recvtime:0.000501 dstPe:1 size:208
backward: [0x80a7a60 22]
forward: [0x80a7ca8 24]

[24] 0x80a7ca8 name:Chunk_overlap_0 (srcnode:-1 msgID:-1) ep:0
[[recvtime:-1.000000 startTime:0.000503 endTime:0.000503]]
backward: [0x80a7af0 23]
forward: [0x80a7dc8 25] [0x80a8170 28]

Postmortem Simulation

- Run application once, get trace logs, and run simulation with logs for a variety of network configurations
- Implemented on POSE simulation framework

Outline

- Overview
- BigSim Emulator
- Charm++ on the Emulator
- Simulation framework
  - Online mode simulation
  - Post-mortem simulation
  - Network simulation
- Performance analysis/visualization

How to Obtain Predicted Time

- Use BgPrint(char *) in similar way
  - Each BgPrint() called at execution time in online execution mode is stored in BgLog as a printing event
- In postmortem simulation, strings associated with BgPrint event is printed when the event is committed
- “%f” in the string will be replaced by committed time.

Compile Postmortem Simulator

- Compile Bigsim simulator
- Compile pose
  - Use normal charm++
  - *cd charm/net-linux/tmp*
  - *make pose*
- Obtain simulator
  - svn co https://charm.cs.uiuc.edu/svn/repos/BigNetSim
- Compile BigNetSim simulator
  - fix BigNetSim/trunk/Makefile.common
  - *cd BigNetSim/trunk/BlueGene*
  - *make*

Example (AMPI CJacobi3D cont.)

- **BigNetSim/trunk/tmp/bigsimulator 0 0**

```
bgtrace: totalBGProcs=4 X=2 Y=2 Z=1 #Cth=1 #Wth=1 #Pes=3
Opts: netsim on: 0
Initializing POSE...
POSE initialization complete.
Using Inactivity Detection for termination.
Starting simulation...
256 4 1024 1.750000 9 1000000 0 1 0 0 0 8 16 4
Info> timing factor 1.000000e+08 ...
Info> invoking startup task from proc 0 ...
[0:AMPI_Barrier_END] interation starts at 0.000217
[0:RECV_RESUME] interation starts at 0.000755
[0:RECV_RESUME] interation starts at 0.001292
[0:RECV_RESUME] interation starts at 0.001829
[0:RECV_RESUME] interation starts at 0.002367
[0:RECV_RESUME] interation starts at 0.002904
[0:RECV_RESUME] interation starts at 0.003441
[0:RECV_RESUME] interation starts at 0.003978
[0:RECV_RESUME] interation starts at 0.004516
[0:RECV_RESUME] interation starts at 0.005053
Simulation inactive at time: 587350
Final GVT = 587351
```

Outline

- Overview
- BigSim Emulator
- Charm++ on the Emulator
- Simulation framework
  - Online mode simulation
  - Post-mortem simulation
  - Network simulation
- Performance analysis/visualization

Big Network Simulator

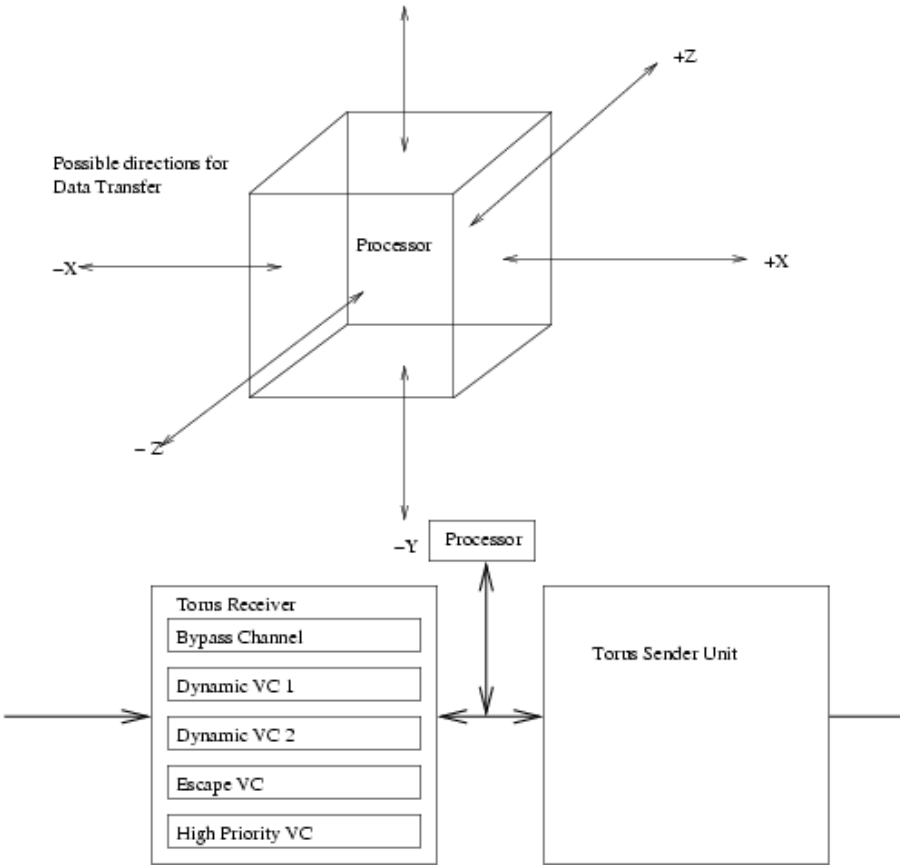
- When message passing performance is critical and strongly affected by network contention

BigNetSim Overview

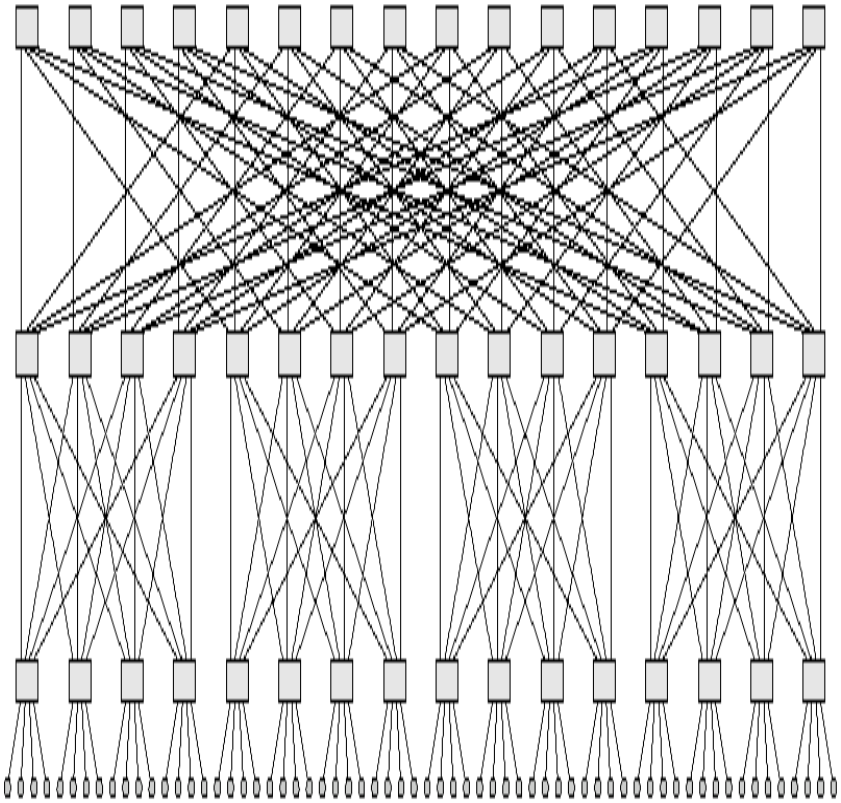
- Networks
- Design
- POSE
- Catalog of Network Simulations
- Building
- Running
- Configuration
- Modular NetSim
  - Mix and match architecture, topology, routing
- Using the Generator
- Extensibility

Networks

Direct Network



Indirect Network

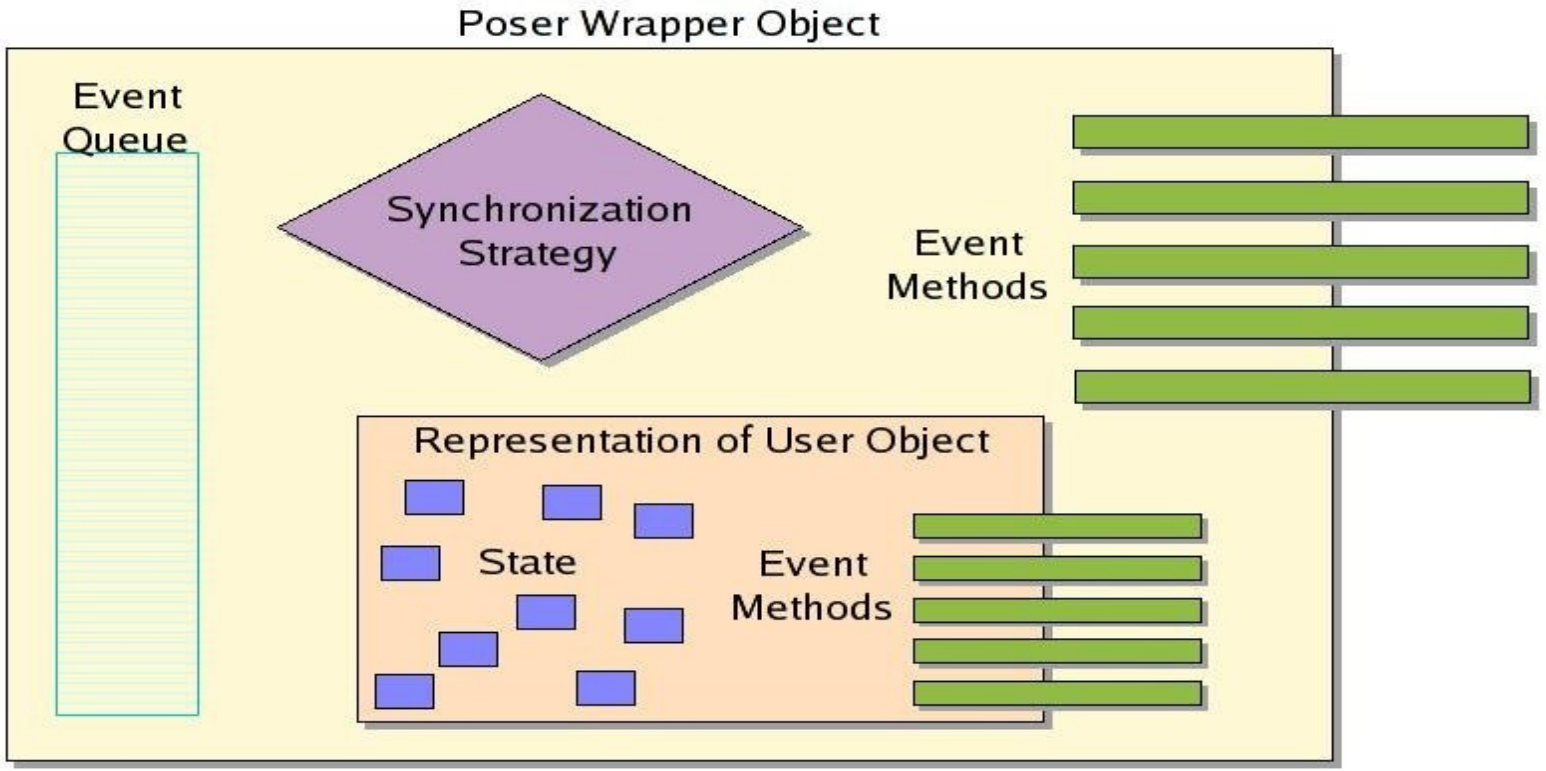


Implementation

- Post-Mortem Network simulators are Parallel Discrete Event Simulations
  - Parallel Object Simulation Environment (POSE)
  - Network layer constructs (NIC, Switch, Node, etc) implemented as poser simulation objects
  - Network data constructs (message, packet, etc) implemented as event methods on simulation objects

POSE

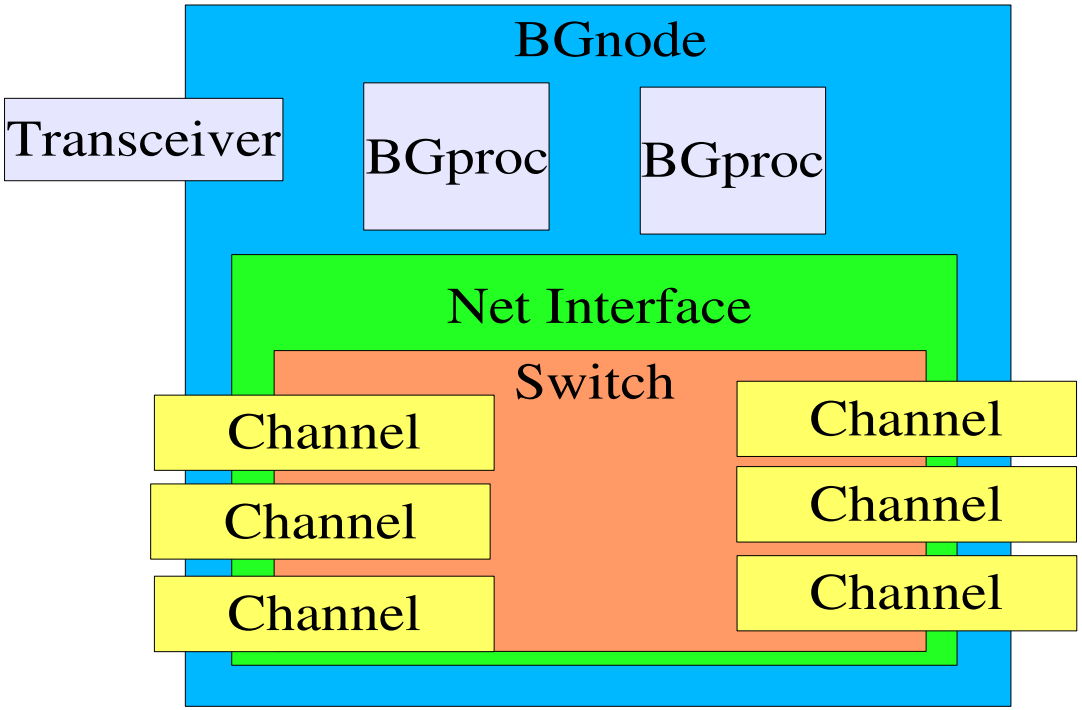
- Each poser is a tiny simulation



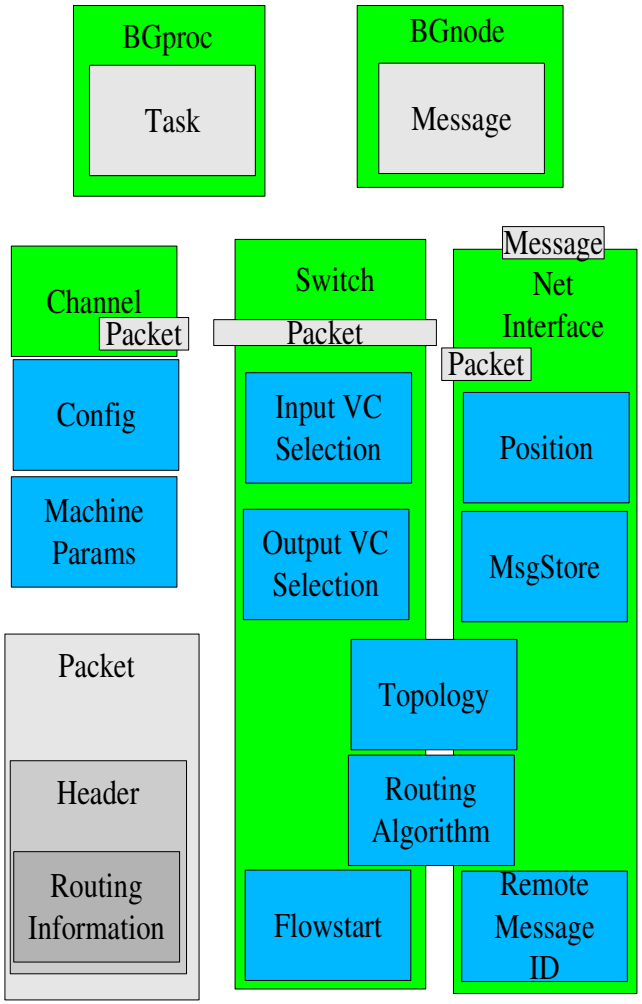
Interconnection Networks

- Flexible Interconnection Network modeling:
  - Choose from a variety of
    - Topologies
    - Routing Algorithms
    - Input Virtual Channel Selection strategies
    - Output Virtual Channel Selection strategies

BigNetSim Design



BigNetSim API: Extensibility



Topology

- Topologies available
  - HyperCube;
  - Mesh; generalized k-ary-n-mesh; n-mesh;
  - Torus; generalized k-ary-n-cube;
  - FatTree; generalized k-ary-n-tree;
  - Low Diameter Regular graphs(LDR)
  - Hybrid topologies
    - HyperCube-Fattree;
    - HyperCube-LDR;

Network Modeling

- Routing models
  - Virtual cut-through routing
- Contention Modeling
  - Port contention at a Switch
  - Load contention: available buffer at next layer of switches
- Adaptive and static Routing algorithms
  - Minimal deadlock-free
  - Non-minimal
  - Fault-tolerant

Routing Algorithms

- K-ary-N-mesh / N-mesh
  - Direction Ordered;
  - Planar Routing;
  - Static Direction Reversal Routing
  - Optimally Fully Adaptive Routing (modified too)
- K-ary-N-tree
  - UpDown (modified, non-minimal)
- HyperCube
  - Hamming
  - P-Cube (modified too)

Input/Output VC selection

- Input Virtual Channel Selection
  - Round Robin;
  - Shortest Length Queue
  - Output Buffer length
- Output Virtual Channel Selection
  - Max. available buffer length
  - Max. available buffer bubble VC
  - Output Buffer length

Building POSE

- POSE
  - cd charm
  - ./build pose net-linux
  - options are set in pose_config.h
    - stats enabled by POSE_STATS_ON=1
    - user event tracing TRACE_DETAIL=1
    - more advanced configuration options
      - speculation
      - checkpoints
      - load balancing

Building BigNetSim

- svn co https://charm.cs.uiuc.edu/svn/repos/BigNetSim
- Build BigNetSim/Bluegene
  - cd BigNetSim/trunk/Bluegene
  - make
  - for sequential simulator
    - make clean; make SEQUENTIAL=1
  - cd ../tmp

Running

- charmrun +p4 bigsimulator 1 1
- Parameters
  - First parameter controls detailed network simulation
    - 1 will use the detailed model
    - 0 will use simple latency
  - Second parameter controls simulation skip
    - 1 will skip forward to the time stamp set during trace creation
    - 0 if not set or network startup interesting

Configuring BigNetSim

USE_TRANSCEIVER 0 For network analysis ignore trace and generate random traffic
NUM_NODES 0 Number of nodes, taken from trace file or set for transceiver
MAX_PACKET_SIZE 256 Maximum packet size
SWITCH_VC 4 The number of switch virtual channels
SWITCH_PORT 8 Number of ports in switch, calculated automatically for direct networks
SWITCH_BUF 1024 Size in memory of each virtual channel
CHANNELBW 1.75 Bandwidth in 100 MB/s
CHANNELDELAY 9 Delay in 10 ns . So 9 => 90ns
RECEPTION_SERIAL 0 Used for direct networks where reception FIFO access has to be serialized
INPUT_SPEEDUP 8 Used to limit simultaneous access by VC in a port. Should be less than or equal to number of VC. Currently used only for bluegene.
ADAPTIVE_ROUTING 1 Additional flag to use adaptive/deterministic routing
COLLECTION_INTERVAL 1000000 Collection * 10ns gives statistics bin size
DISPLAY_LINK_STATS 1 Display statistics for each link
DISPLAY_MESSAGE_DELAY 1 Display message delay statistics

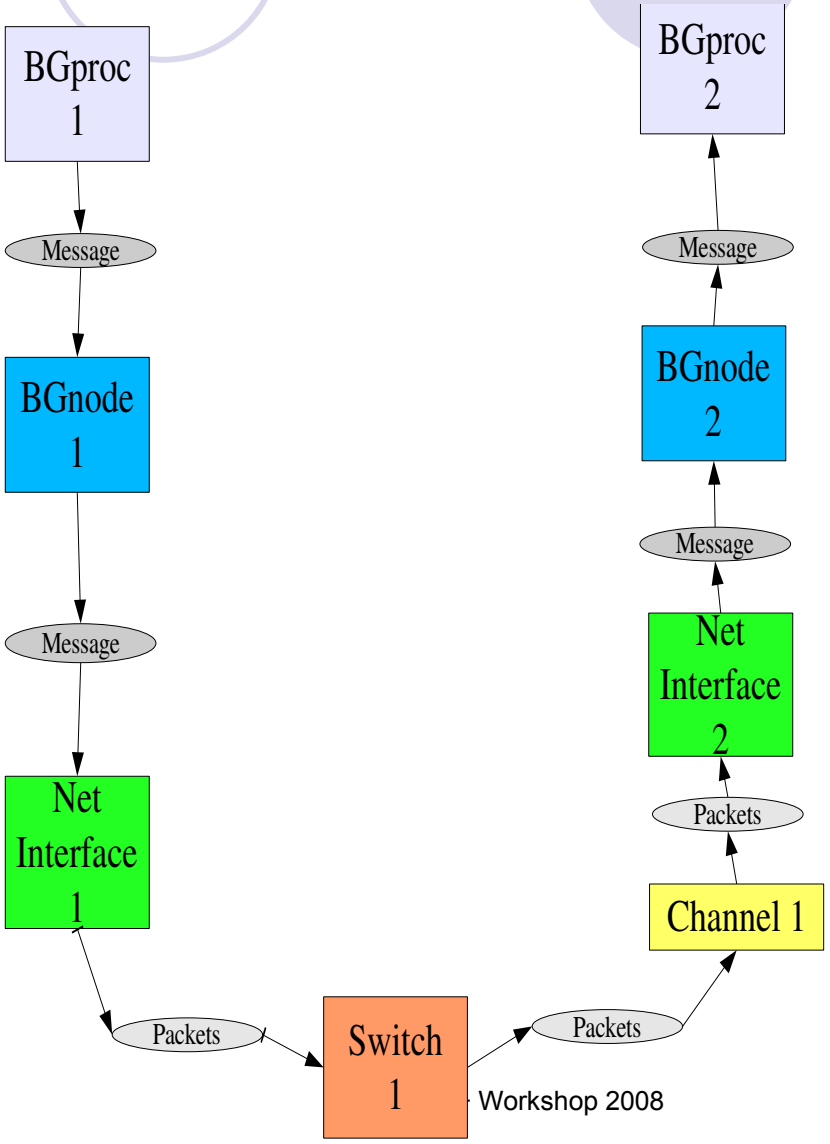
Output

- Completion time for trace run
- Per Link utilization, link contention high water marks
- If trace projections logs for the trace exist, an updated “corrected” copy is created.
- Turn on -tproj to get simple trace of network performance if projections traces from the emulator are not available
- Use -projname YOURAPPNAME to direct bignetsim to your existing tracelogs for updating.

Artificial Network Loads

- Generate traffic patterns instead of using trace files
  - additional command line parameters
    - Pattern
    - Frequency
- Pattern
  - 1 kshift
  - 2 ring
  - 3 bittranspose
  - 4 bitreversal
  - 5 bitcomplement
  - 6 poisson
- Frequency
  - 0 linear
  - 1 uniform
  - 2 exponential

BigNetSim: Data Flow



Adding a Network

- mkdir new subdir in trunk
- copy boilerplate InitNetwork.h
- copy boilerplate Makefile
  - change MACHINE make variable to your dirname
- new InitNetwork.C
  - Define switch, channel, nic mappings
  - Define how switches route and select virtual channels
  - Define topology and default routing

Adding a Topology

- New *.h *.C in trunk/Topology
  - constructor()
  - getNeighbours()
  - getNext()
  - getNextChannel()
  - getStartPort()
  - getStartVC()
  - getStartSwitch()
  - getStartNode()
  - getEndNode()

Adding a Routing Strategy

- New *.h *.C files in trunk/Routing
  - constructor()
  - selectRoute()
  - populateRoute()
  - loadTable()
  - getNextSwitch()
  - sourceToSwitchRoutes()

Adding a VC Selector

- Either Input or Output VC Selector
  - new *.h *C in [Input/Output]VCSelector
  - constructor()
  - select[Input/Output]VC()

Future

- Improved scalability
  - adaptive strategies
  - improved hardware collectives
  - out-of-core loading of tracefiles
  - load balancing
  - network fault simulation
- Ports to BG/L/P, Cray XT3/4, for hosting of simulator.
- Representative collection of netconfig files

Case Study - NAMD

- Molecular Dynamics Simulation Applications
- Compile BigSim Charm++:
  - *./build bigsim net-linux bigsim*
- Compile NAMD:
  - Get source code from:
    - *http://charm.cs.uiuc.edu/~gzheng/namd-bg.tar.gz*
  - *./config fftw Linux-i686-g++*

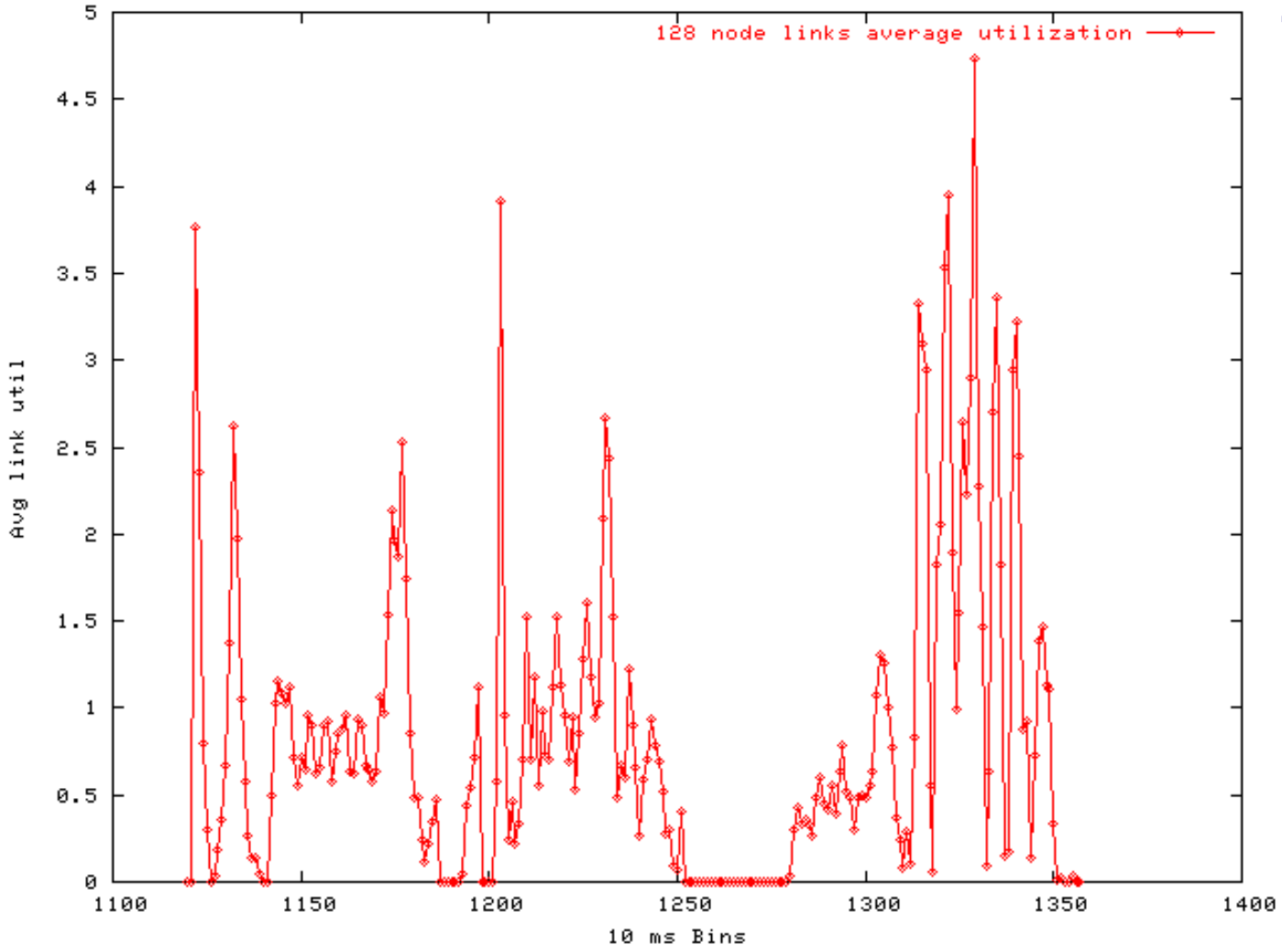
Validation with Simple Network Model

NAMD Apo-Lipoprotein A1 with 92K atom.

Performance simulation using 8 Lemieux processors

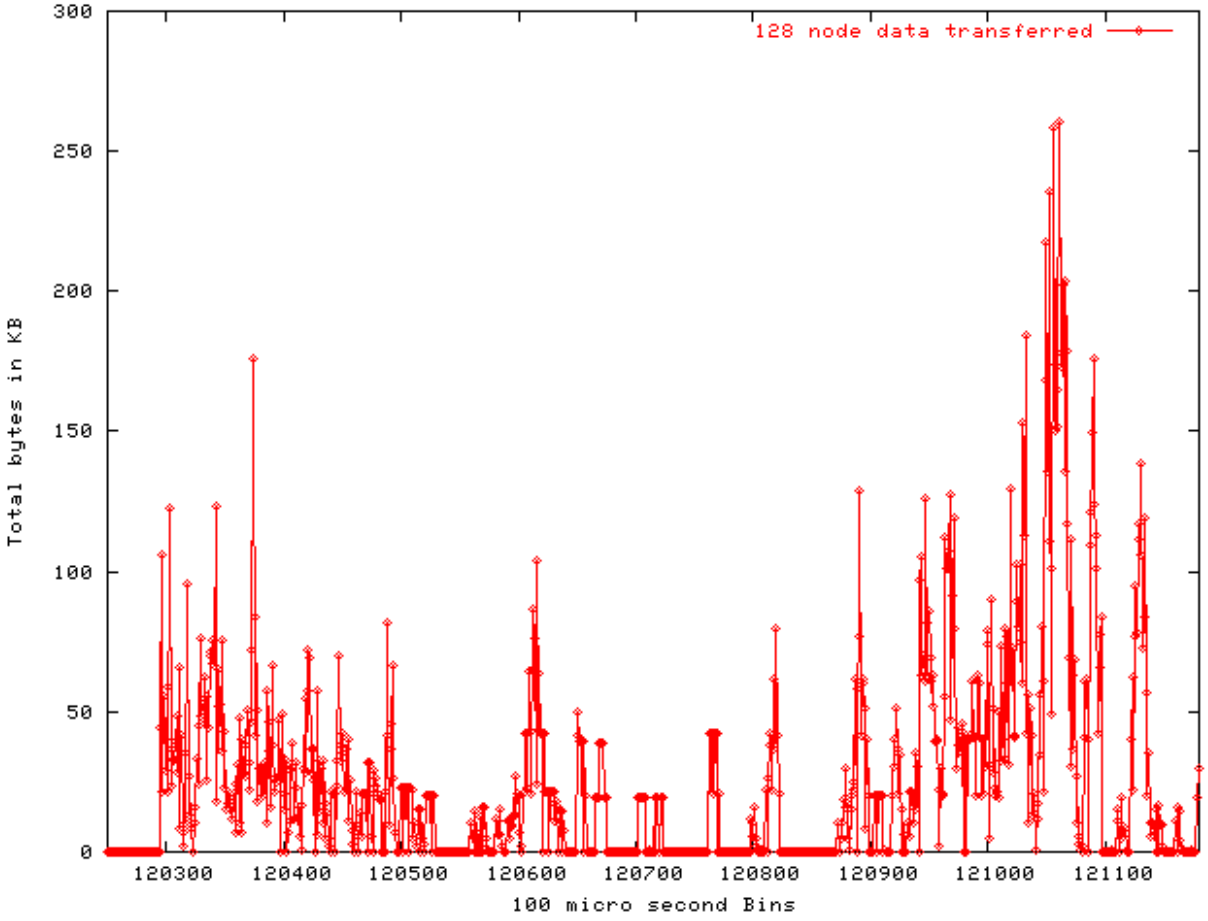
Processors	128	256	512	1024
Actual time (ms)	71.5	40.3	23.9	17.6
Predicted time (ms)	75.8	43.6	25.1	20.8

Network Communication Pattern Analysis



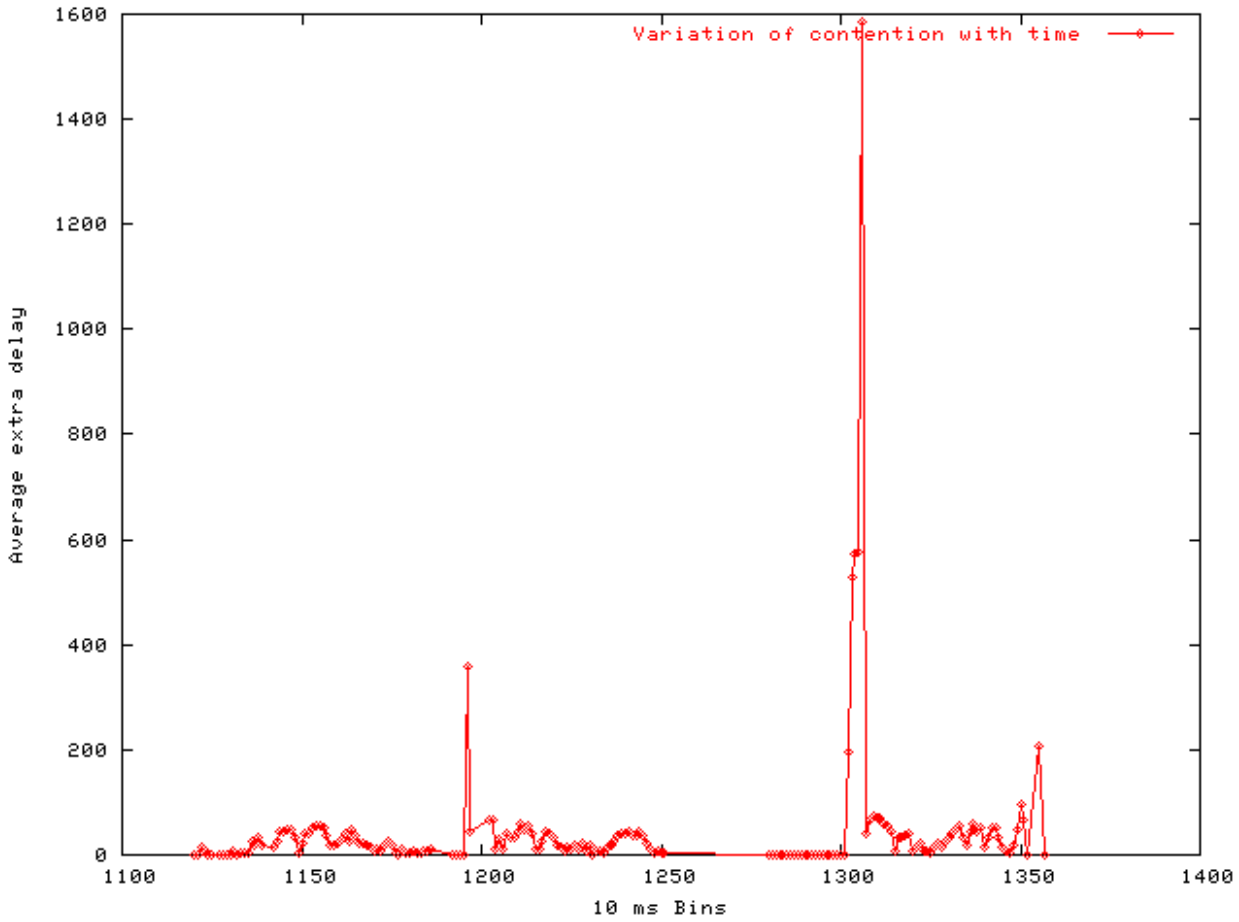
- NAMD with apoa1
- 15 timestep

Network Communication Pattern Analysis



Data transferred (KB) in a single time step

Contention Encountered by Messages



Outline

- Overview
- BigSim Emulator
- Charm++ on the Emulator
- Simulation framework
  - Online mode simulation
  - Post-mortem simulation
  - Network simulation
- Performance analysis/visualization

Performance Analysis/Visualization

- trace-projections is available for BigSim and BigNetSim
- One challenge:
  - Number of log files can be overwhelming

Generate Projections Logs

- Link application with
  - –tracemode projections
- Select subset of processors in bgconfig:

  projections 0-100,2000,3100-3200
- With timestamp correction, two sets of projections logs are generated
  - Before and after timestamp correction

Generate Projections Logs (the hideous secret)

- Problem:
  - Projections tracing function maintains a fix sized buffer for storing projections logs
  - Buffer is flushed to disk when it is filled up, disk I/O can effect predicted time
- Solution:
  - Use +logsize runtime option to provide large projections buffer size
- In fact, in online mode simulation, simulation aborts when disk I/O occurs.

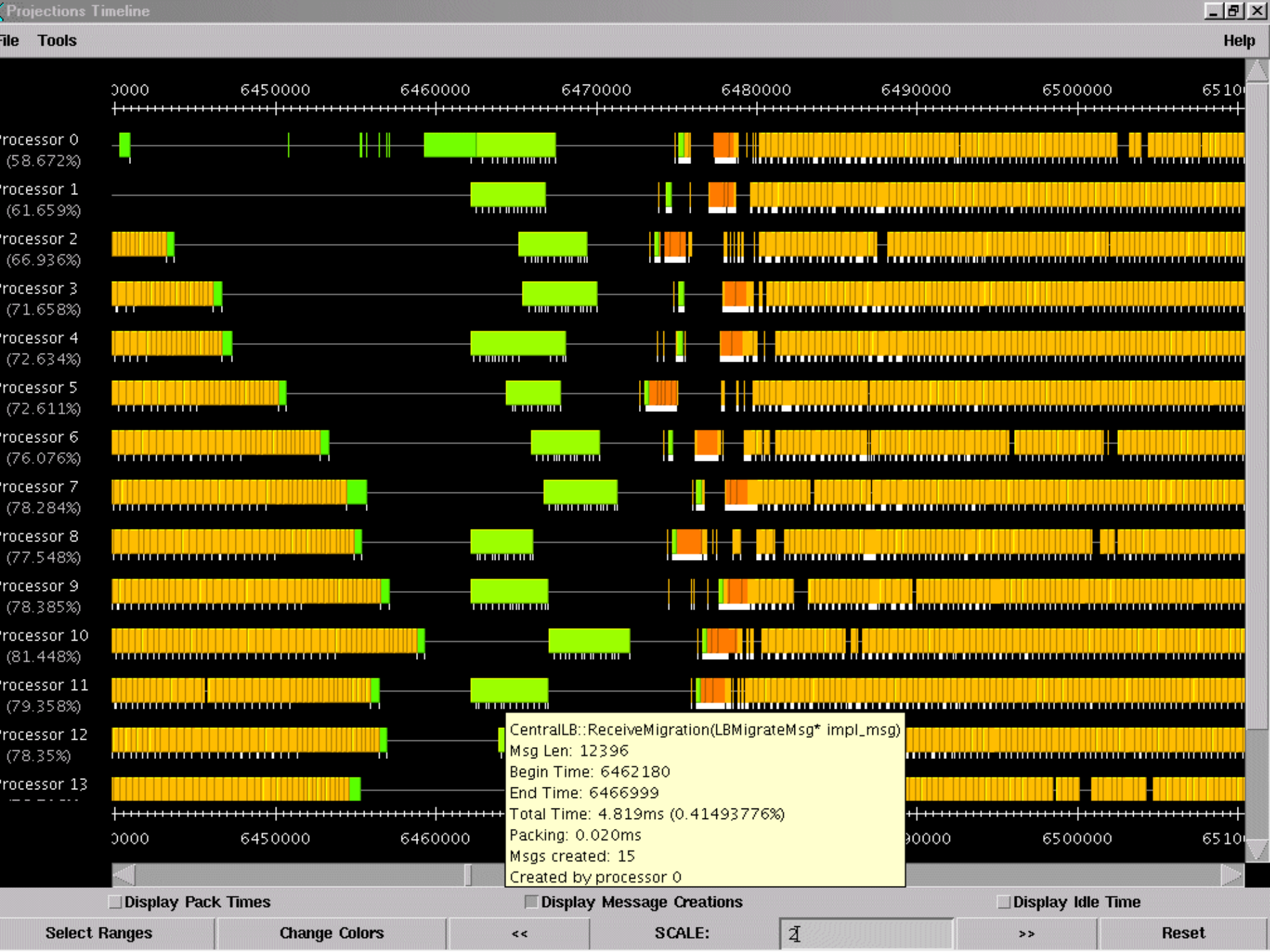
Projections with Jacobi

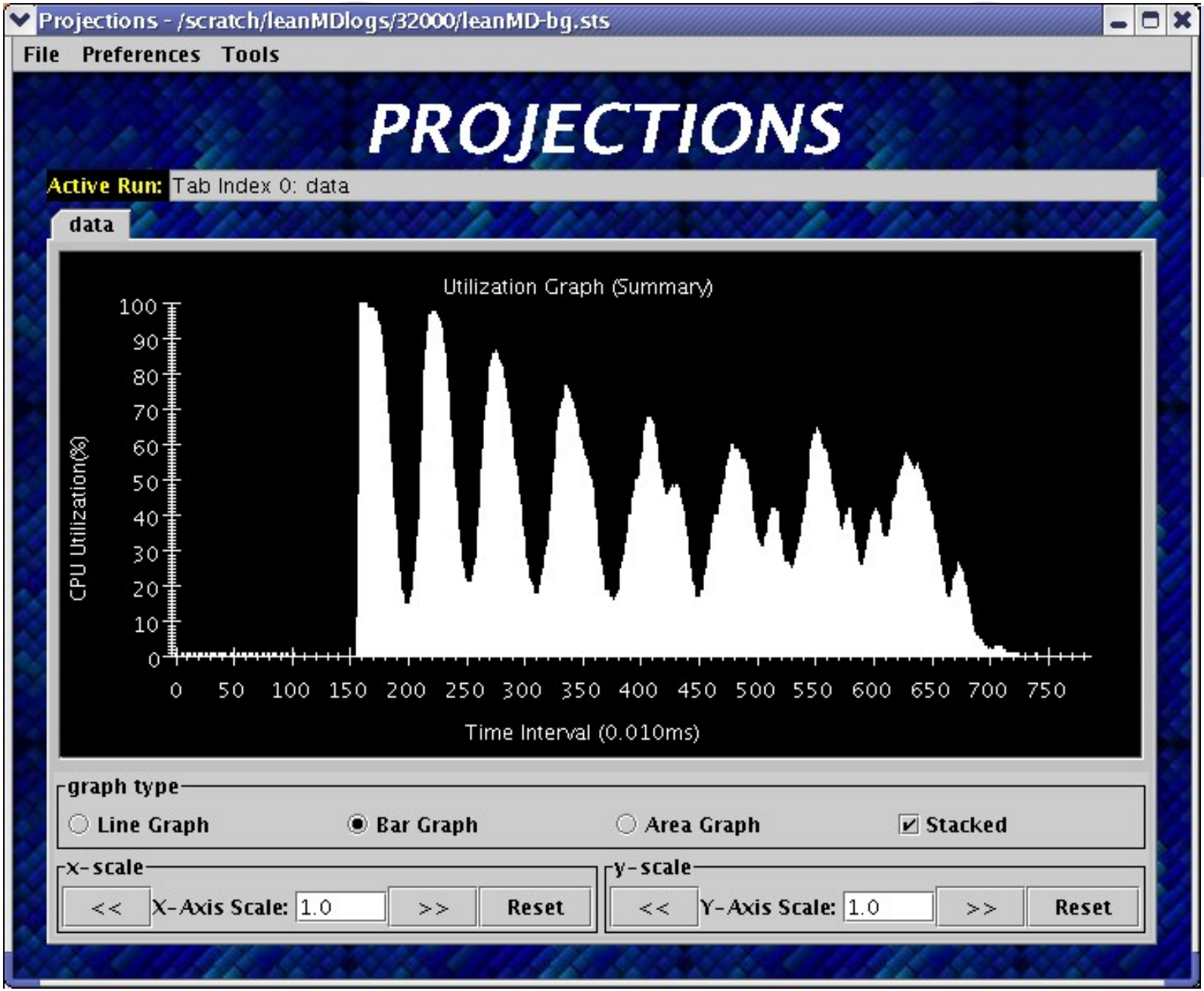
- cd charm/examples/bigsim/sdag/jacobi-no-redn
- ./charmrun +p4 ./jacobi 16384 10 8192 +bgconfig ./bg_config
- Config file:

```
x 32
y 16
z 16
cth 1
wth 1
stacksize  10000
#timing  walltime
timing  bgelapse
#timing  counter
cpufactor   1.0
fpfactor    5e-7
traceroot     .
log      yes
correct  yes
network   lemieux
projections 0,1000,8189-8191
```


Projections Timeline <3>
File Tools Colors
Help
0 50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950
PE 0 (85.3%)
PE 1000 (84.5%)
PE 8189 (83.7%)
PE 8190 (83.7%)
PE 8191 (83.7%)
0 50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950
Display Pack Times
Display Message Sends
Display Idle Time
View User Events (0)
Select Ranges
Change Colors
<<
SCALE:
1.0
>>
Reset
Highlight Time
Slection Begin Time
Selection End Time
Selection Length
Zoom Selected
Load Selected
Jump to graph:
Usage Profile
Graph
Histogram
Communication
Overview

Make bgtest
With 16 processors


Projections Timeline
File
Tools
Help
0000
6450000
6460000
6470000
6480000
6490000
6500000
6510
Processor 0
(58.672%)
Processor 1
(61.659%)
Processor 2
(66.936%)
Processor 3
(71.658%)
Processor 4
(72.634%)
Processor 5
(72.611%)
Processor 6
(76.076%)
Processor 7
(78.284%)
Processor 8
(77.548%)
Processor 9
(78.385%)
Processor 10
(81.448%)
Processor 11
(79.358%)
Processor 12
(78.35%)
Processor 13
CentralLB::ReceiveMigration(LBMigrateMsg* impl_msg)
Msg Len: 12396
Begin Time: 6462180
End Time: 6466999
Total Time: 4.819ms (0.41493776%)
Packing: 0.020ms
Msgs created: 15
Created by processor 0
Display Pack Times
Display Message Creations
Display Idle Time
Select Ranges
Change Colors
<<
SCALE:
1
>>
Reset


Projections - /scratch/leanMDlogs/32000/leanMD-bg.sts
File
Preferences
Tools
PROJECTIONS
Active Run:
Tab Index 0: data
data
Utilization Graph (Summary)
CPU Utilization(%)
100
90
80
70
60
50
40
30
20
10
0
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
Time Interval (0.010ms)
graph type
Line Graph
Bar Graph
Area Graph
Stacked
x-scale
<<
X-Axis Scale:
1.0
>>
Reset
y-scale
<<
Y-Axis Scale:
1.0
>>
Reset

Thank You!

Free download of Charm++ and BigSim at
http://charm.cs.uiuc.edu
Send comments to ppl@charm.cs.uiuc.edu