BigSim: Simulating PetaFLOPS Supercomputers

Gengbin Zheng

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

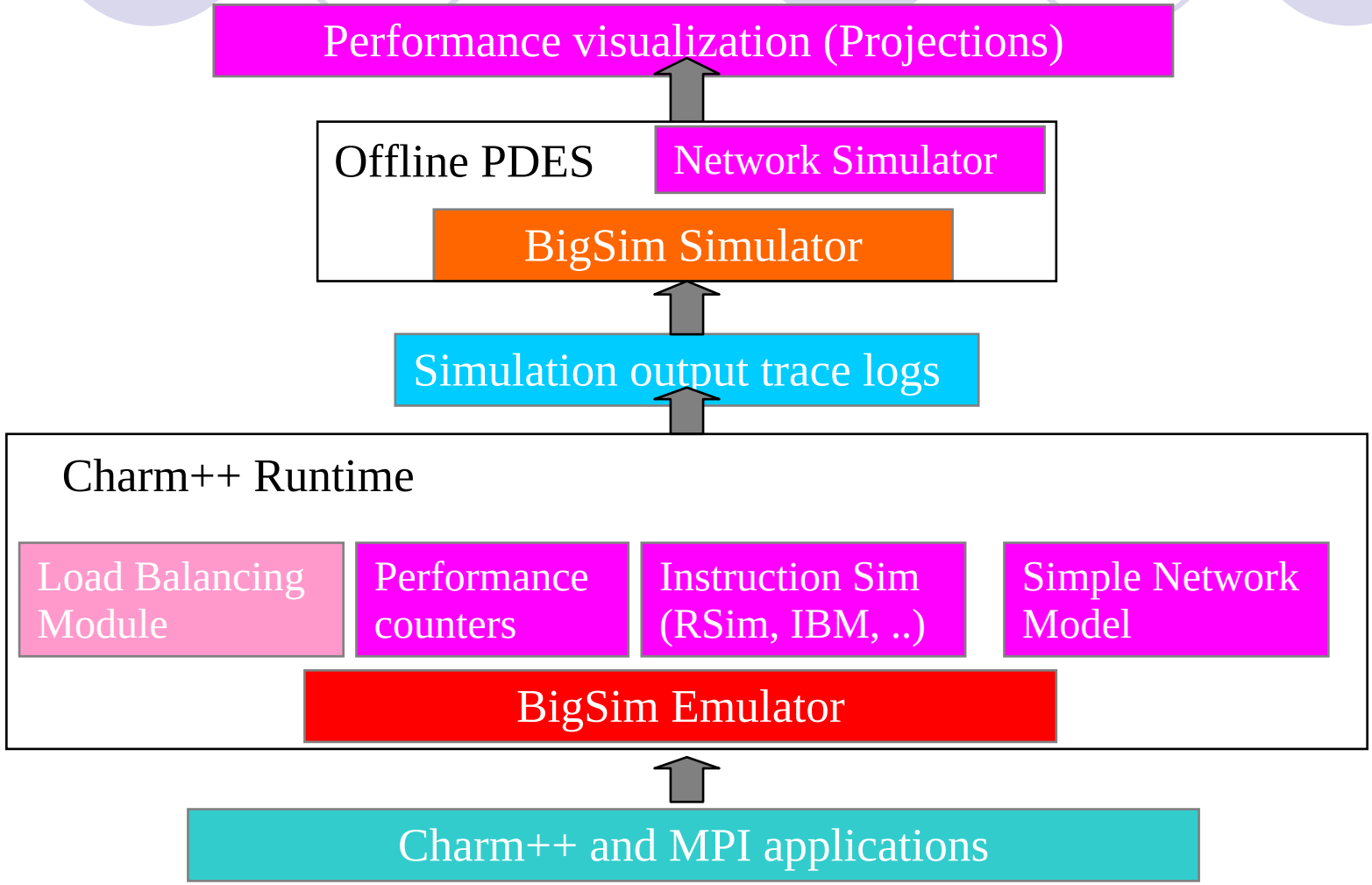
Introduction

- Extremely large scale machines are being built
  - To use the full machine is going to be difficult
  - Access may not be easy
  - Applications are not necessarily ready for them
- It is hard to understand parallel application without actually running it
- Simulation-based approach
  - Tools that allow one to predict the performance of applications on future machines, also
  - help debugging and tuning, finding performance bottleneck
- Allows easier "offline" experimentation
- Provides a feedback for machine designers

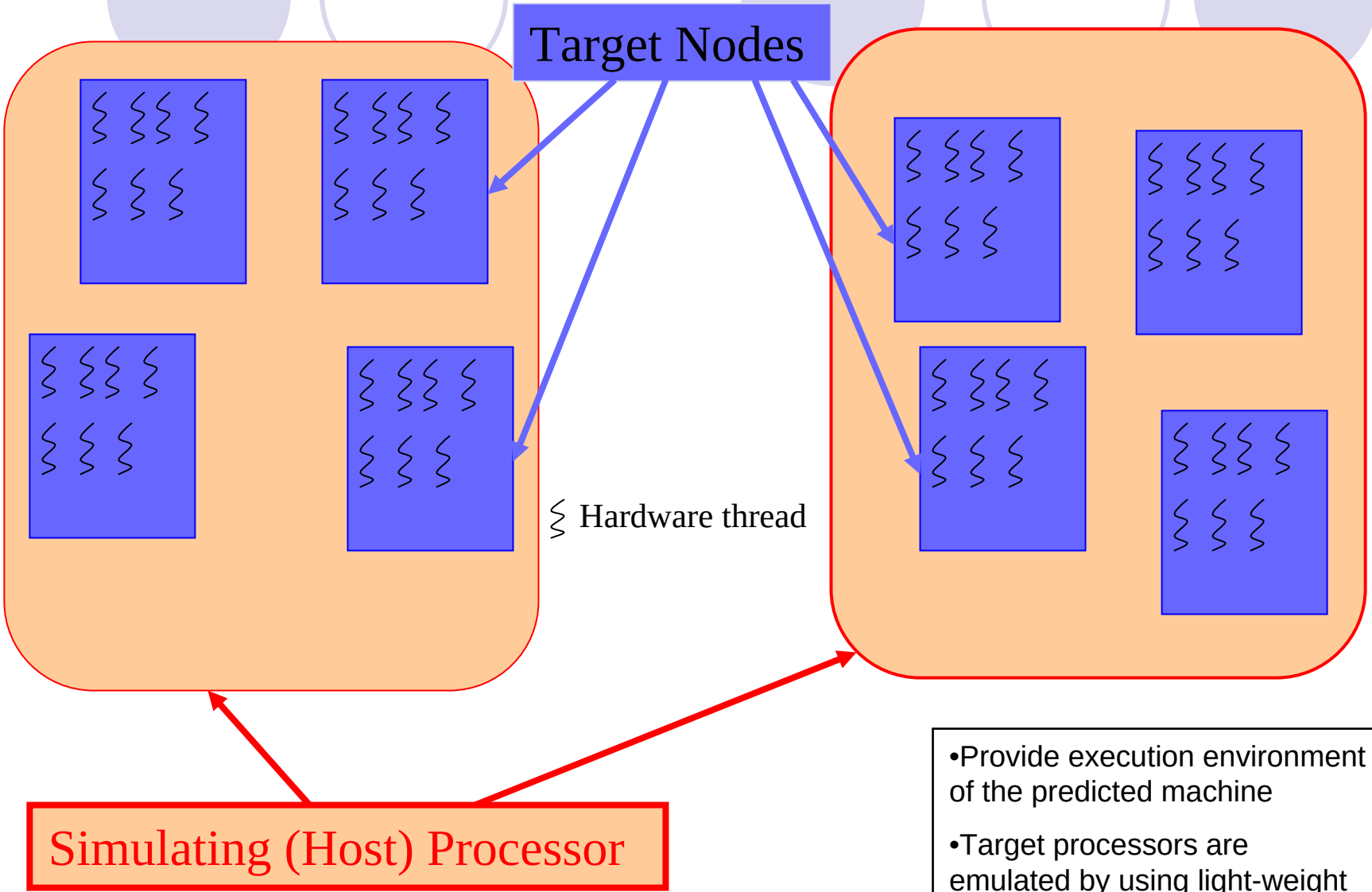
Summary of Our Approach

- Trace-driven simulation method
  - Run application once, working on traces varying hardware parameters
- Two step simulation
  - BigSim Emulator, capable of running application
    - Charm++, Adaptive MPI / MPI
    - Predict sequential execution blocks
    - Generate trace logs
  - Parallel Discrete Event BigSim Simulator (POSE-based)
    - Predict network performance
- Implemented on Charm++

BigSim System Architecture



BigSim Emulator



- Provide execution environment of the predicted machine
- Target processors are emulated by using light-weight threads in Charm++

Predicting Sequential Performance

- Different level of fidelity:
  - Direct mapping of CPU frequency
  - Performance counter
  - Instruction-level simulator
- Provide trade-offs in accuracy and simulation time

Trace Logs

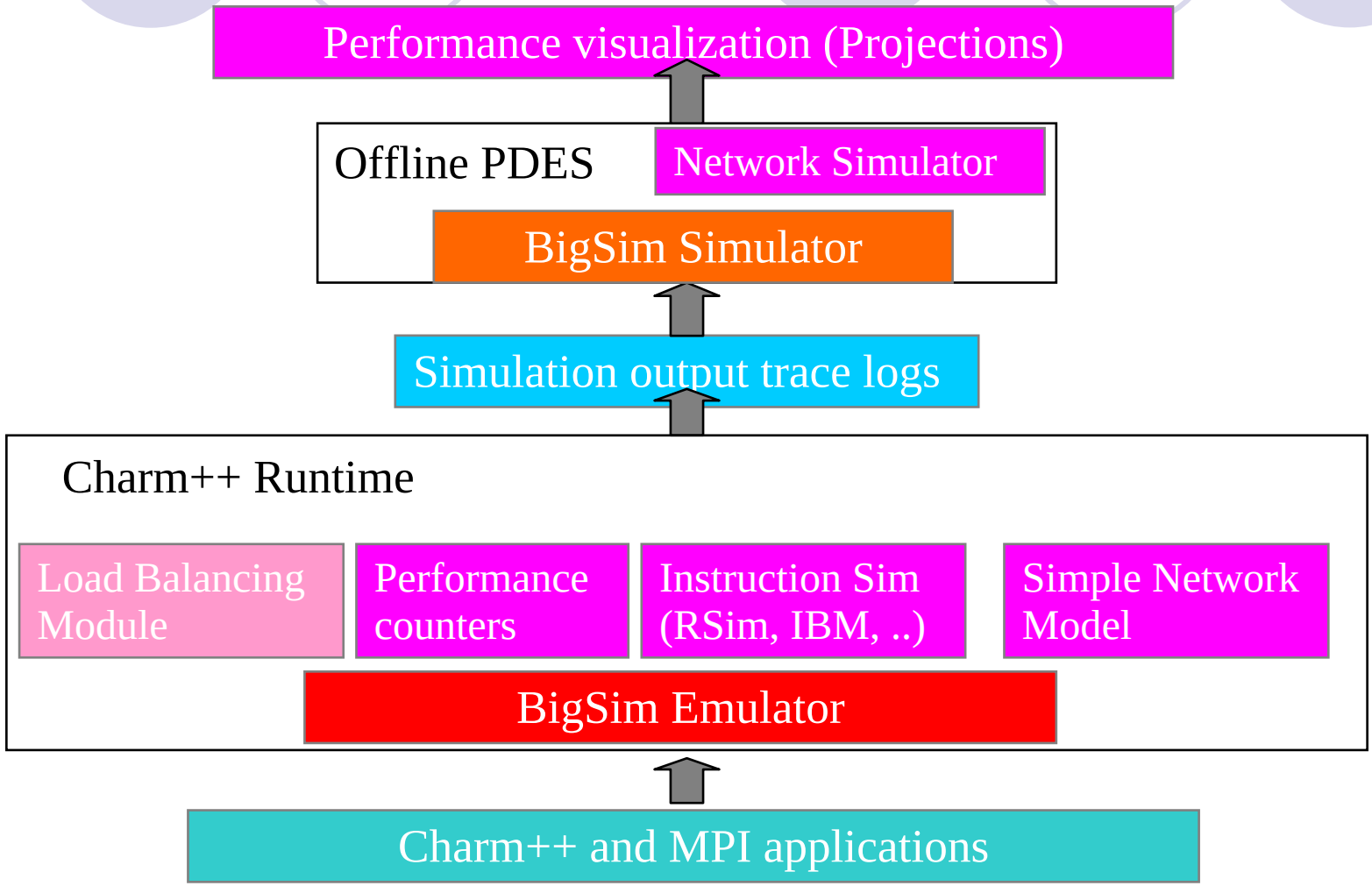
- Event logs are generated for each target processor
  - Predicted time on execution blocks with timestamps
  - Event dependencies
  - Message sending events

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

[23] 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: [22]
forward: [24]

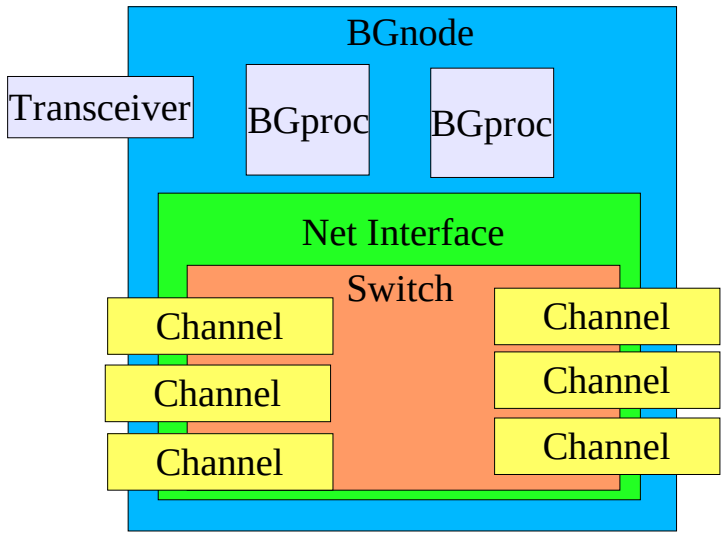
[24] name:Chunk_overlap_0 (srcnode:-1 msgID:-1) ep:0
[[recvtime:-1.000000 startTime:0.000503 endTime:0.000503]]
backward: [0x80a7af0 23]
forward: [25] [28]

BigSim System Architecture

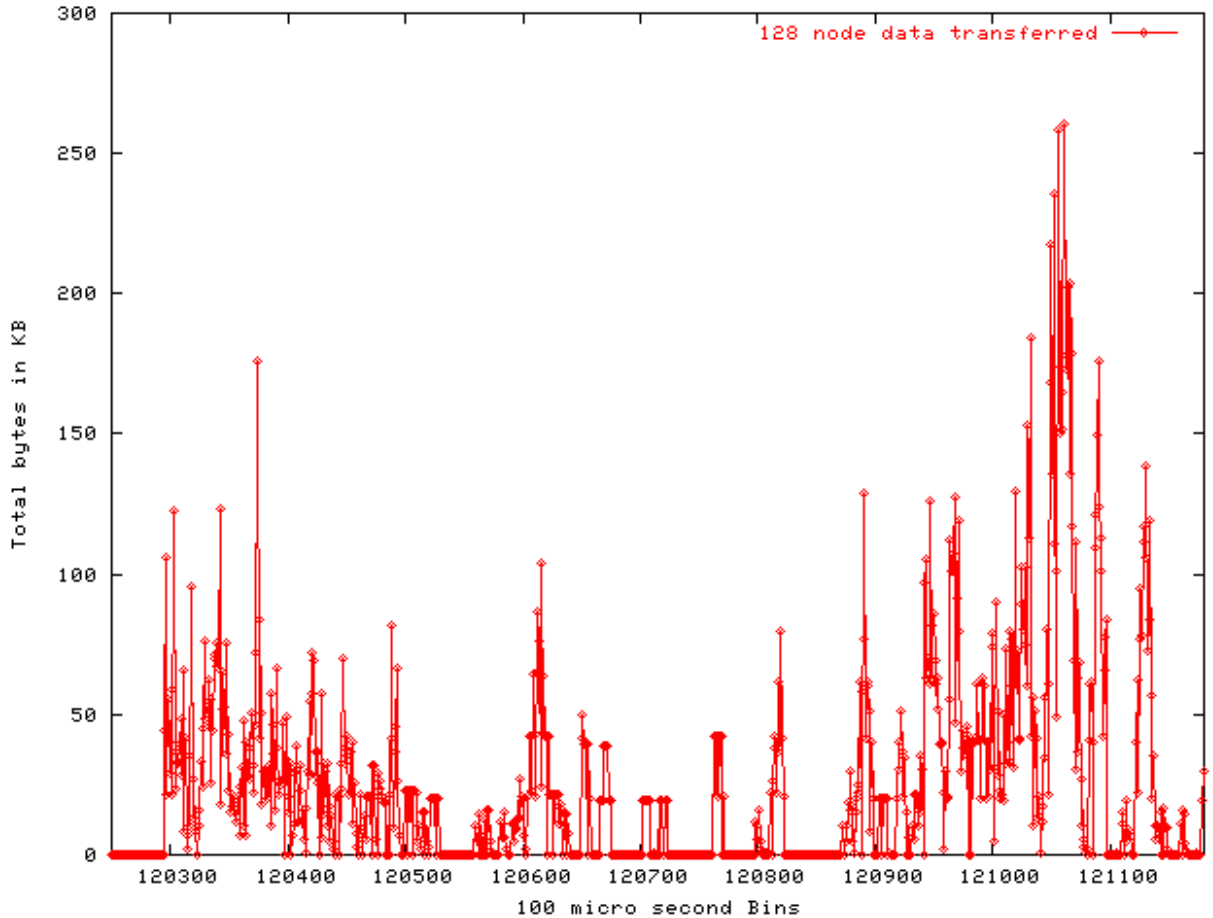


Predicting network performance – second step

- Parallel Discrete Event Simulation (PDES)
  - Needed for correcting causality errors due to out-of-order messages
  - Timestamps are re-adjusted without rerunning the actual application
- Simulate network behaviours: packetization, routing, contention, etc
  - simple latency based network model, and contention-based network model



Network Communication Pattern Analysis

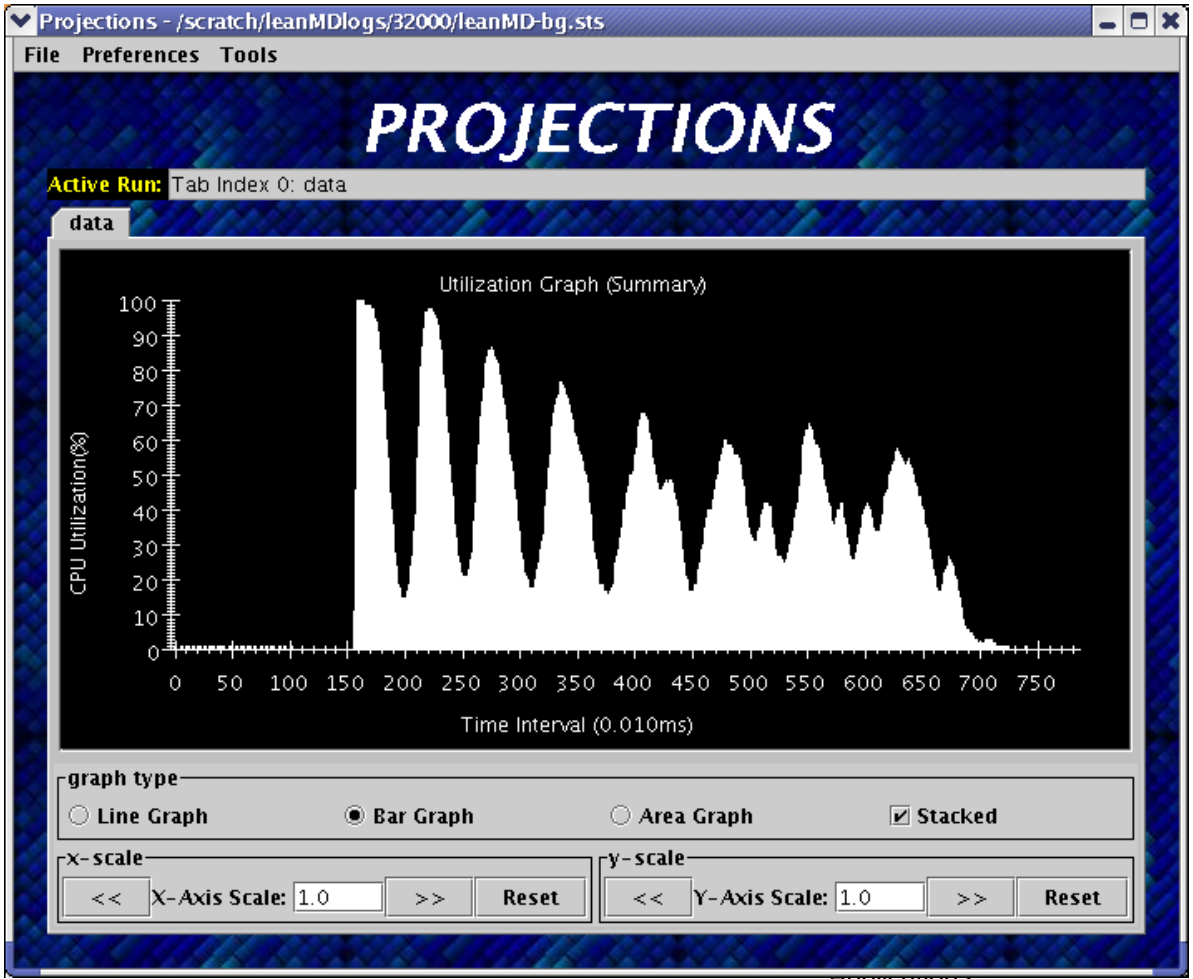


Data transferred (KB) in a single time step

Why Charm++?

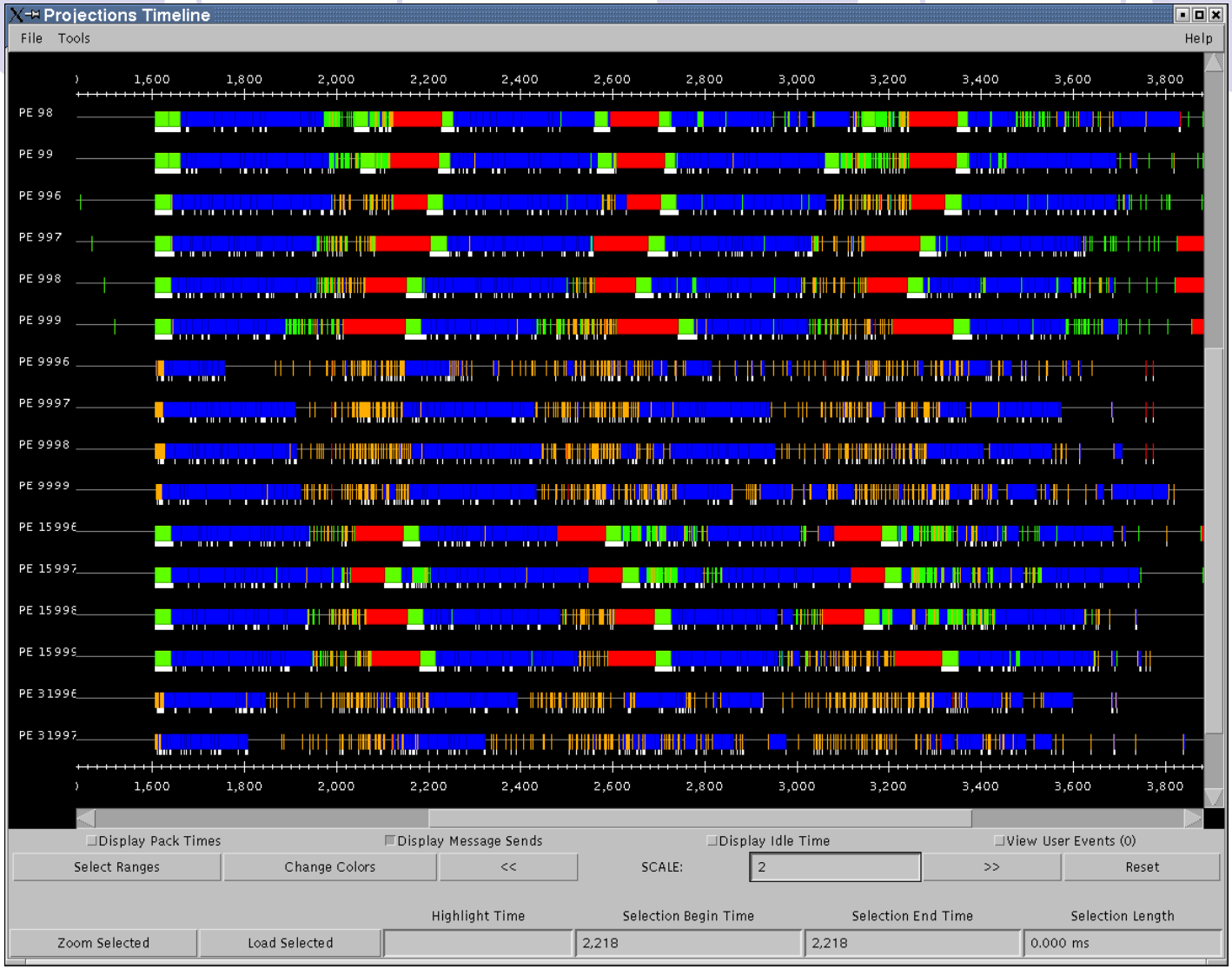
- Highly portable
- PDES simulation is naturally message-driven
- Large number of entities naturally maps to virtual processors
- Fine grained simulation and migratable objects allow dynamic load balancing
- Performance analysis and visualization tools

LeanMD Performance Analysis



- Benchmark 3-away ER-GRE
- 36573 atoms
- 1.6 million objects
- 8 step simulation
- 64k BG processors
- Running on PSC Lemieux

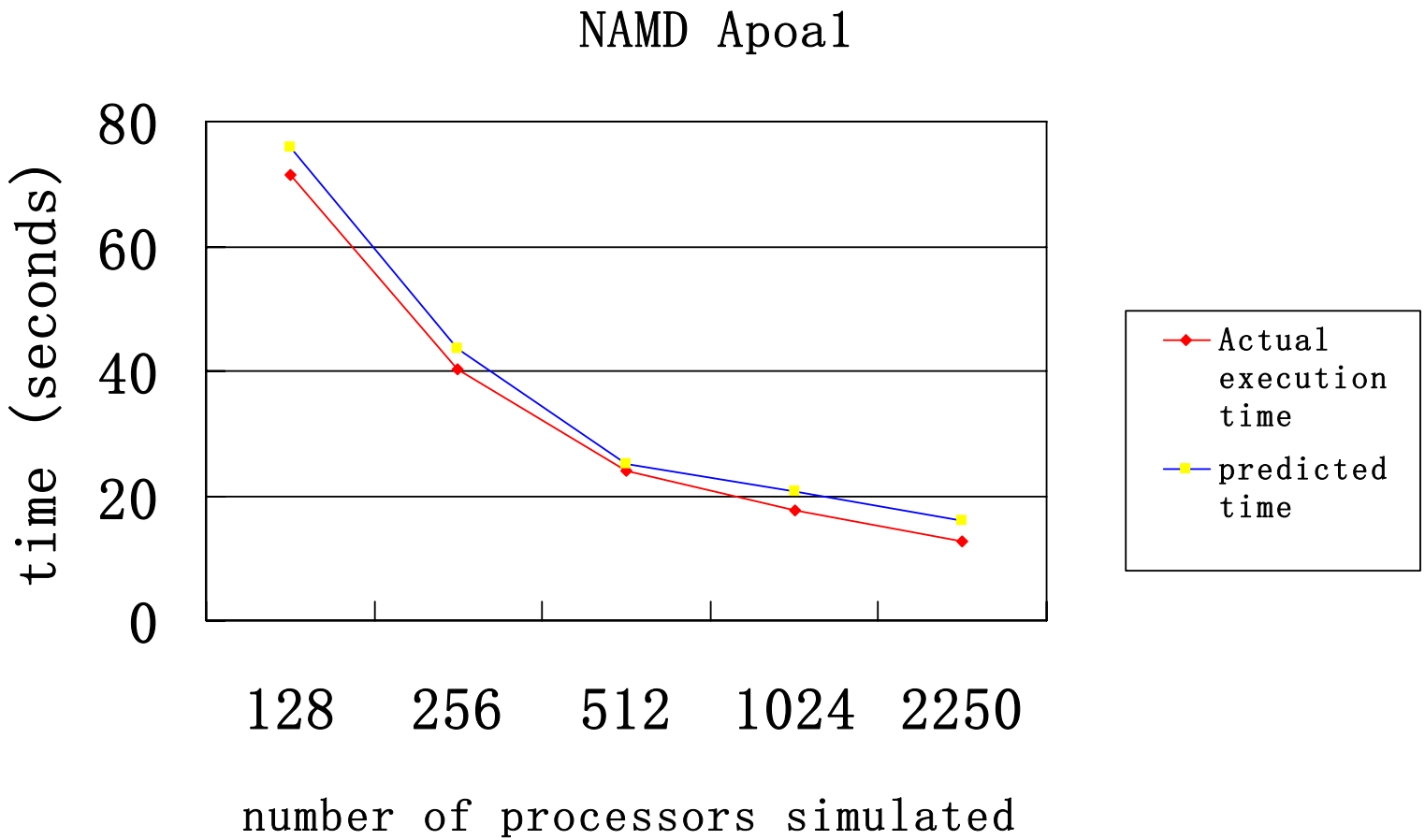
Performance visualization - Timeline



BigSim trace log API

- Provide API to generate bigsim trace logs by hand
  - Standalone libraries that are independent of Charm++
- Allow other non-Charm++ applications to generate trace logs and use our simulator and performance analysis and visualization tool

Validation


NAMD Apoa1
time (seconds)
80
60
40
20
0
128
256
512
1024
2250
number of processors simulated
Actual execution time
predicted time

BigSim New Challenges

- How to work with a Instruction-level simulator for cycle accurate prediction
- How to get away with the memory constraints when running memory-bound applications

Integration with Instruction Level Simulators

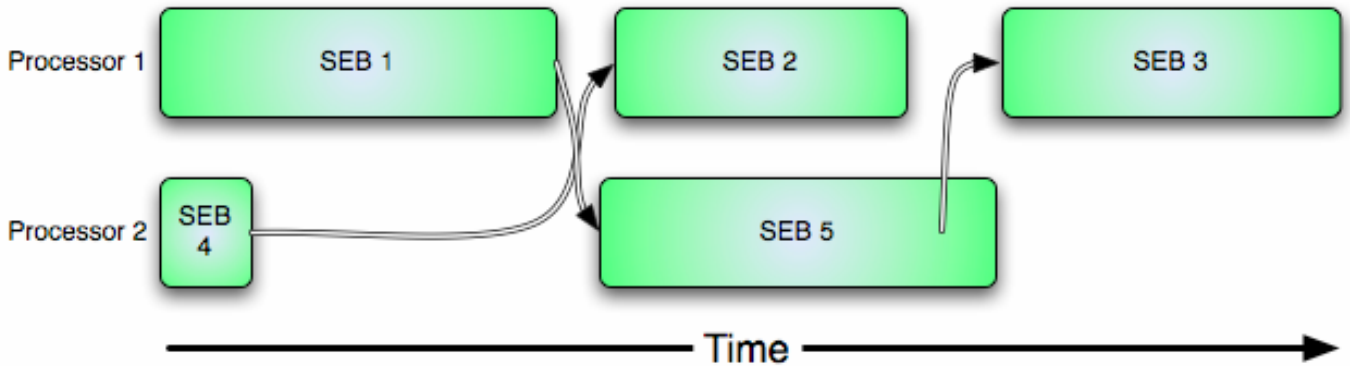
- Instruction level simulators typically are
  - Standalone applications, only simulate a sequential program
    - Difficult to integrate
  - Cycle accurate simulation is very slow
    - Difficult to simulate the whole problem
- One observation for Charm++ application
  - Each SEB takes same amount of execution time in sequential and parallel cases
    - Partition of the problem is independent of number of processors

Two-phase simulation

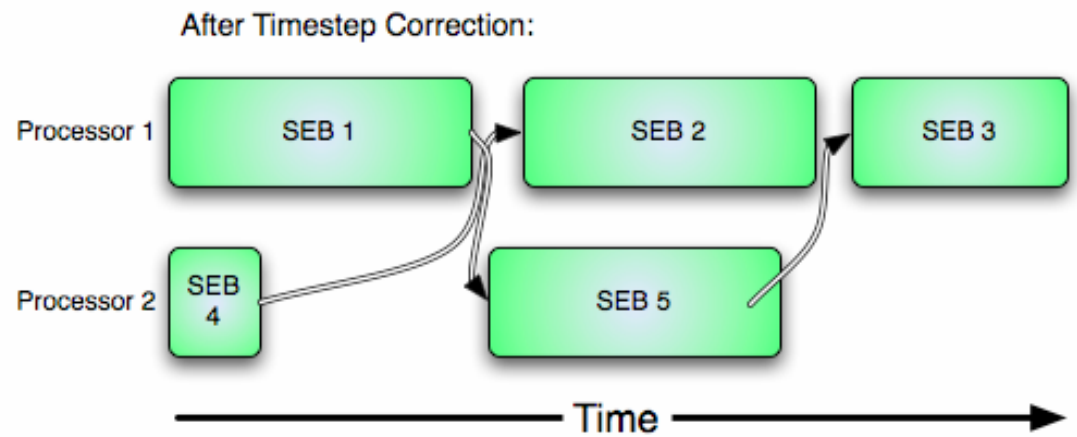
- Run cycle accurate simulation on a dataset
  - Generate cycle accurate timings for each SEB
- Run BigSim emulation on an existing machine on the same dataset
  - SEB timings can be wrong in the trace logs
- Rewrite the SEB in trace logs using cycle accurate data

Interpolation Tool Rewrites SEB Durations

Traces from existing machine

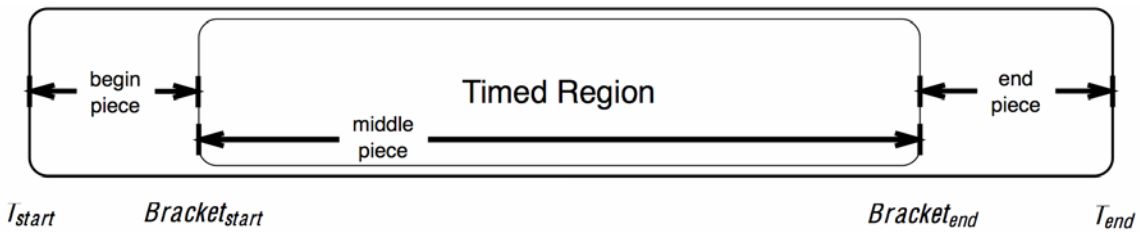


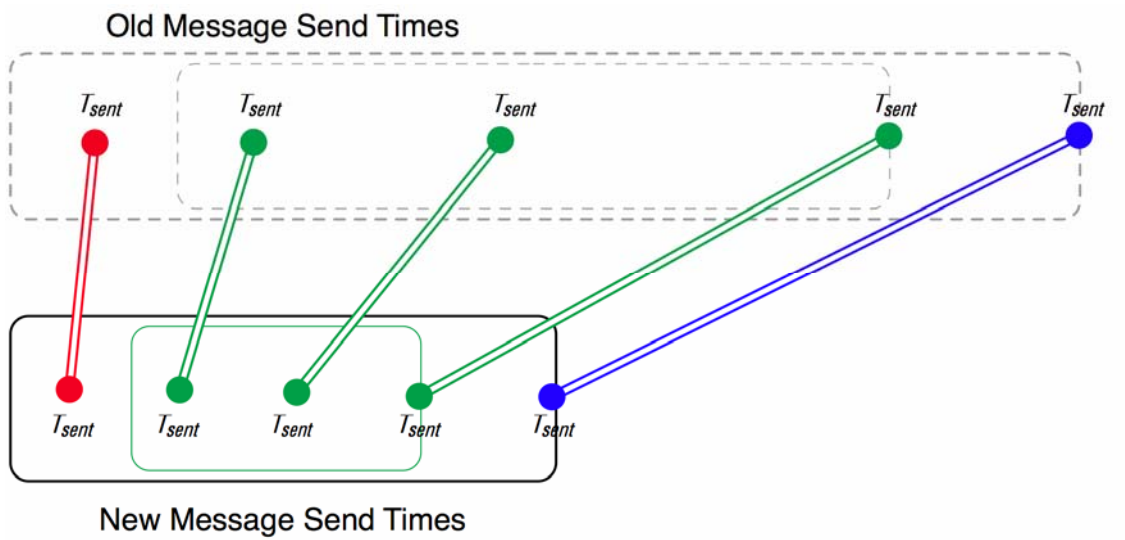
Traces adapted to match another machine



Interpolation Tool Rewrites SEB Durations

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

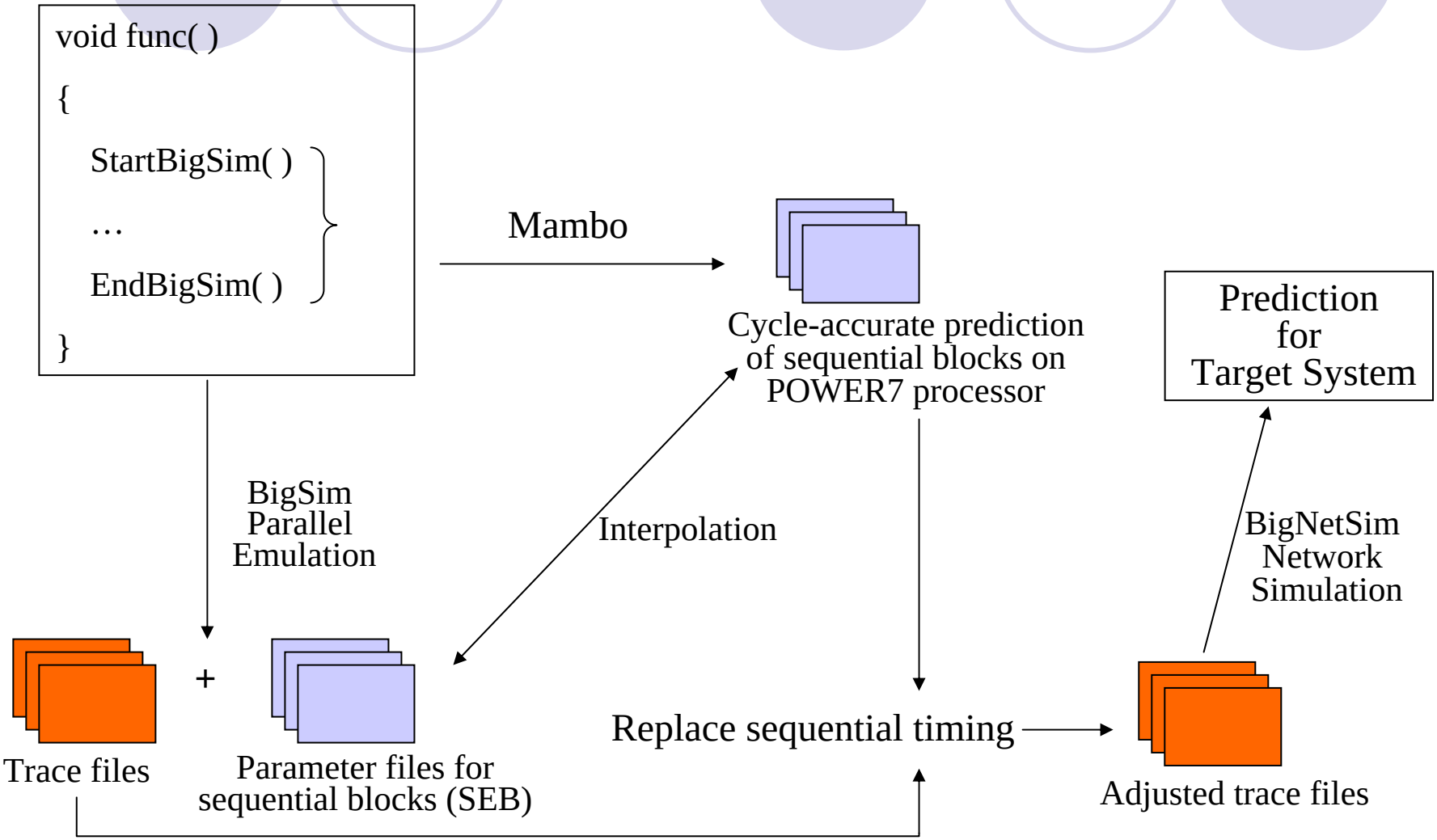




More Complicated Scenario

- When it is impossible to simulate the whole problem size in cycle accurate mode
- Use a small run on a smaller dataset to predict the final large problem
  - Define a set of parameters that best describe the performance of a SEB
    - $T_{SEB}(x_1, x_2, \ldots, x_n) = A_1x_1 + A_2x_2 + .. A_nX_n + C$
  - Based on the sample data from the small size run, do a least-squares fit to determine the coefficients of an approximation polynomial function
  - Use $T_{SEB}(x_1, x_2, \ldots, x_n)$ to predict large dataset

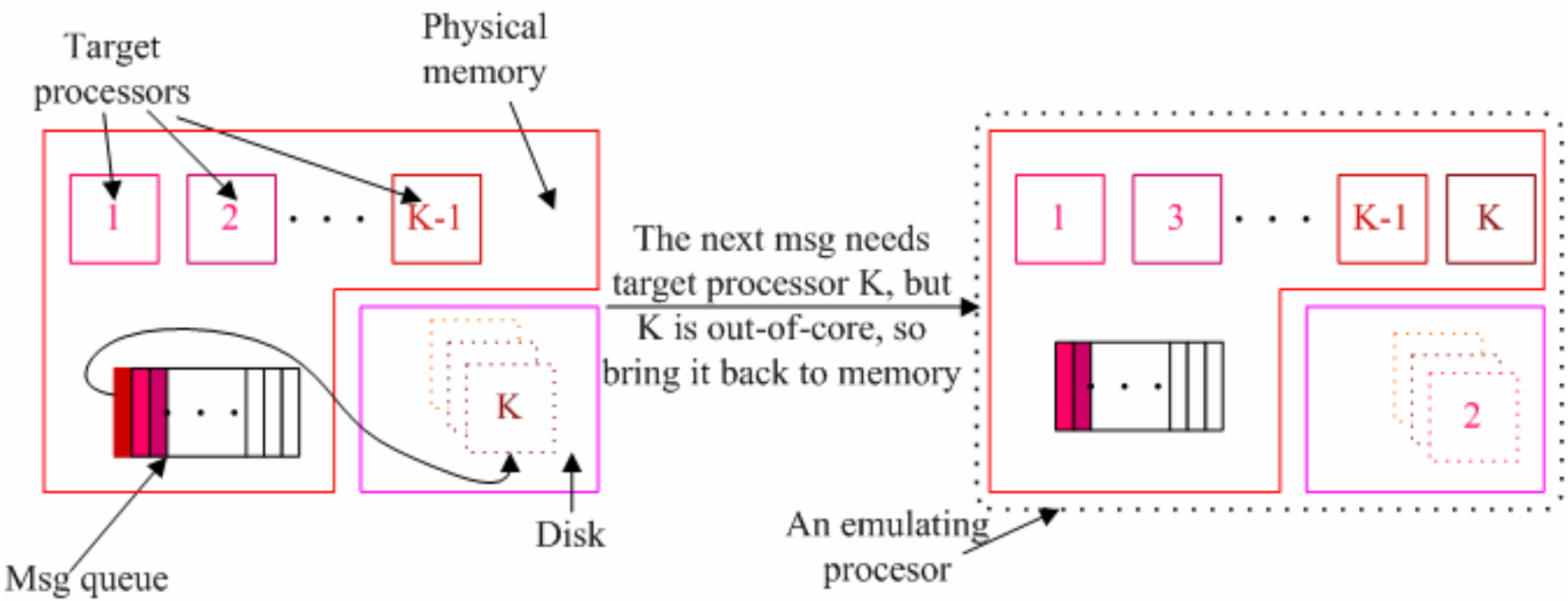
Workflow


void func( )
{
StartBigSim( )
…
EndBigSim( )
}
Mambo
Cycle-accurate prediction of sequential blocks on POWER7 processor
Prediction for Target System
BigSim Parallel Emulation
Interpolation
BigNetSim Network Simulation
+
Trace files
Parameter files for sequential blocks (SEB)
Replace sequential timing
Adjusted trace files

Out-of-core Emulation

- Memory constraint when running many copies of an application on a single node
  - Physical memory is shared
  - VM system would not handle well
- A straightforward technique: out-of-core

Overview of the idea



- Out-of-core
  - Restore a processor from checkpoint
  - Invoke entry on that processor
  - Processor writes checkpoint
  - remove all array elements and user data to free up memory

Initial results for basic schemes

- Environment
  - A Jacobi3D problem in MPI
  - A linux workstation with 4GB memory/4GB swap
  - Two sets of tests
    - 1 emulating processor, 8 targeting processors, 1 MPI process/targeting processor
    - 1 emulating processor, 512 targeting processors, 1 MPI process/targeting processor

Preliminary Results

- Normal test, allowing 1 target processor in memory

problem size per MPI process	image size of target proc (MB)	emulation time s/step		slowdown ratio
		w/o out-of-core	with out-of-core	
10^3	1.03	0.024	0.755	31.46
50^3	2.31	0.146	1.716	11.75
100^3	10.01	1.826	25.110	13.75
200^3	67.55	15.728	184.816	18.11

- Stress test
  - problem size: 200^3 ➔ total memory footprint is slightly over 4GB memory (on swap space)
  - About 1.77 times slowdown

Options of basic schemes

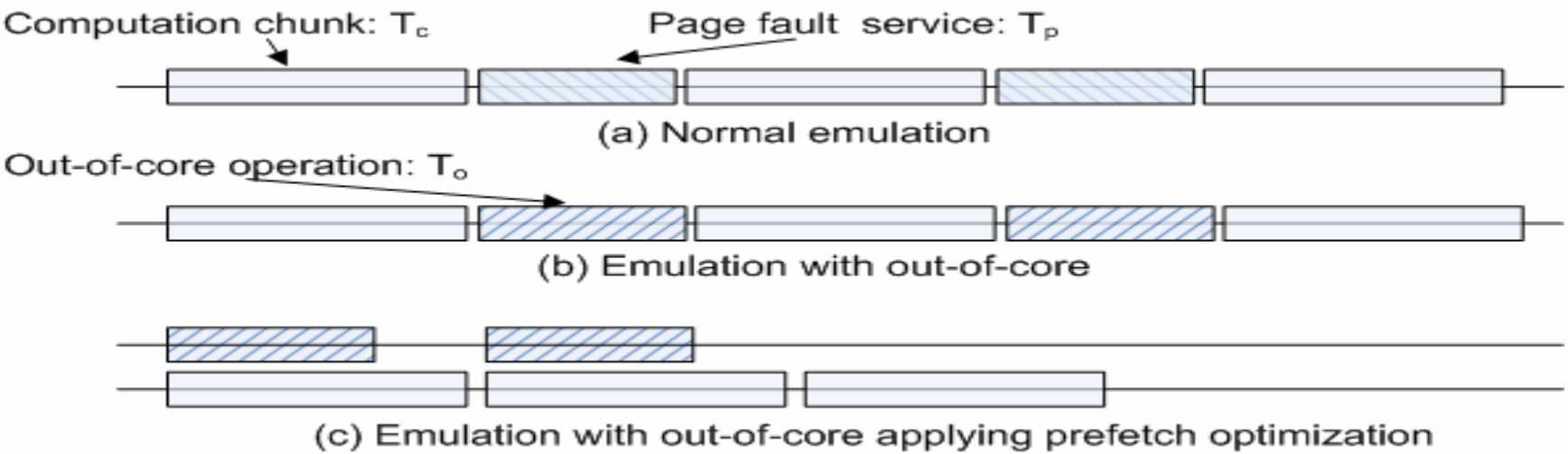
- Per message based
  - Swapping in/out a target processor for every message
- Multiple target processors based
  - Only allowing a fixed number of target processors in memory
- Actual memory based
  - Allowing as many target processors in memory if possible

Future work on optimization

- Tuning replacement policy
  - Which processor to swap out?
- Using prefetch
  - Overlap simulation with asynchronous I/O
  - Good predictability
    - Messages in the queue
    - We know what will be the next message by peeking the message queue

Two different scenarios (1)

- Per message triggers large chunk of computation



Thank you

BigSim software can be downloaded from http://charm.cs.uiuc.edu