

Facilitating Co-Design for Extreme-Scale Systems Through Lightweight Simulation

Christian Engelmann¹ and Frank Lauer^{1,2}

**¹ Computer Science Research Group
Computer Science and Mathematics Division
Oak Ridge National Laboratory, USA**

² The University of Reading, UK

Motivation (1/2)

- **Ongoing trends in HPC system design:**
 - Increasing core counts (overall and SMP concurrency)
 - Increasing node counts (OS instances)
 - Heterogeneity (CPU+GPGPU at large scale)
- **Emerging technology influencing HPC system design:**
 - Stacked (3D) memory
 - Non-volatile memory (SSD and phase change)
 - Network-on-chip
- **Additional forces influencing HPC system design:**
 - Power consumption ceiling (overall and per-chip)
- **How to design HPC systems to fit application needs?**
- **How to design applications to efficiently use HPC systems?**

Proposed Exascale Initiative Road Map

Many design factors are driven by the power ceiling of 20MW

Systems	2009	2011	2015	2018
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa
System memory	0.3 PB	1.6 PB	5 PB	10 PB
Node performance	125 GF	200GF	200-400 GF	1-10TF
Node memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	22 GB/s	25 GB/s	50 GB/s
System size (nodes)	18,700	100,000	500,000	O(million)
Total concurrency	225,000	3,200,000	O(50,000,000)	O(billion)
Storage	15 PB	30 PB	150 PB	300 PB
IO	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	days	days	days	O(1 day)
Power	6 MW	~10MW	~10 MW	~20 MW

My Exascale Resilience Scenario: MTTI Scales with Node Count

Systems	2009	2011	2015	2018
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa

System size (nodes)	5x	5x	2x
---------------------	----	----	----

Vendors are able to maintain current node MTTI

MTTI	4 days	19 h 4 min	3 h 52 min	1 h 56 min
------	--------	------------	------------	------------

My Scary Scenario: Current MTTI of 1 Day

Systems	2009	2011	2015	2018
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa

System size (nodes)	5x	5x	2x
---------------------	----	----	----

Current system MTTI is actually lower

MTTI	1 day	4 h 48 min	58 min	29 min
------	-------	------------	--------	--------

My Really Scary Scenario: Component MTTI drops 3% Each Year

Systems	2009	2011	2015	2018
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa

System size (nodes)	5x	5x	2x
---------------------	----	----	----

Vendors are not able to maintain current node MTTI

MTTI	1 day	4 h 31 min	48 min	22 min
------	-------	------------	--------	--------

Motivation (2/2)

- **Today's applications can not handle a MTTI of 22 minutes**
 - **Checkpoint/restart to/from a parallel file system is not feasible at this failure rate**
 - **MPI applications are generally not fault tolerant**
- **An exascale system may simply become unusable if only one out of many design factors is off**
- **HPC system/application co-design is necessary to:**
 - **Identify early on system and application constraints**
 - **Continuously track these constraints as they change**
 - **Efficiently match system and application properties**

HPC Application/System Co-Design Through Simulation

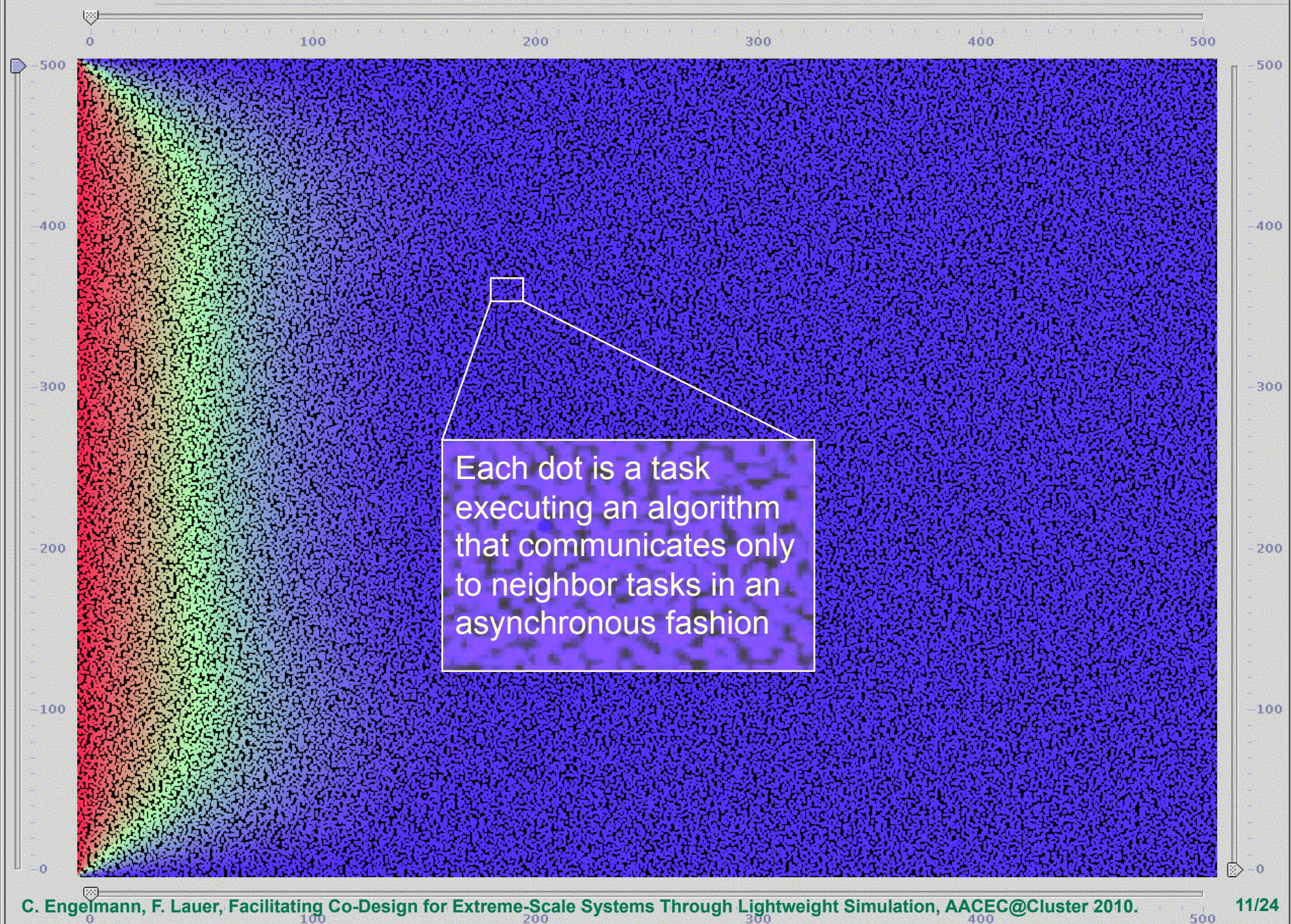
- **Parallel discrete event simulation (PDES) to emulate the behavior of future architecture choices**
- **Execution of real applications, algorithms or their models atop an emulated environment for:**
 - **Performance evaluation, including identification of resource contention and underutilization issues**
 - **Resilience evaluation, including dependability benchmarking (performance under failure) and robustness testing (error propagation and masking)**
- **The presented work focuses on performance evaluation through simulation as a first step**

Related Work

- **Efficient parallel discrete event simulation (PDES) is obviously a research field of its own**
- **Java Cellular Architecture Simulator (JCAS) from Oak Ridge National Laboratory**
- **BigSim project at University of Illinois at Urbana-Champaign**
- **Other trace-driven PDES solutions, such as DIMEMAS**
- **$\mu\pi$ from Oak Ridge National Laboratory**
- **SST project at Sandia National Laboratories**

Java Cellular Architecture Simulator (JCAS)

- **Developed in Java with native C/Fortran support (2002-04)**
- **Runs as standalone or distributed application**
- **Lightweight framework that simulates up to 1,000,000 lightweight virtual processes on 9 real processors**
- **Standard and experimental network interconnects:**
 - **Multi-dimensional mesh/torus**
 - **Nearest/Random neighbors**
- **Message driven simulation without notion of time**
 - **Not in real-time, no virtual time**
- **Primitive fault-tolerant MPI support**
 - **No collectives, no MPI 2**



Targeted Applications/Algorithms (Natural Fault Tolerance and Super-Scalable)

- **Local information exchange algorithms:**
 - Mesh-free chaotic relaxation (Laplace/Poisson)
 - Finite difference/element methods
 - Dynamic adaptive refinement at runtime
 - Asynchronous multi-grid methods
 - Monte Carlo method
 - Peer-to-peer diskless checkpointing
- **Global information exchange algorithms:**
 - Global peer-to-peer broadcasts of values
 - Global maximum/optimum search
- **Applications:**
 - Locally self-consistent multiple scattering (LSMS) method
 - Molecular dynamics simulation for computational biology

BigSim, $\mu\pi$ and SST

BigSim

- Initiated in 2001 by the IBM Blue Gene/C project
- BigSim Emulator
 - Atop Charm++ and Adaptive MPI
 - Similar to JCAS, but scales worse
 - Meant for testing and debugging at scale
- BigSim Simulator:
 - Trace-driven PDES
 - Meant for performance estimation

$\mu\pi$

- Currently under development
- Similar to presented effort
- Based on more advanced, heavier PDES engine

SST

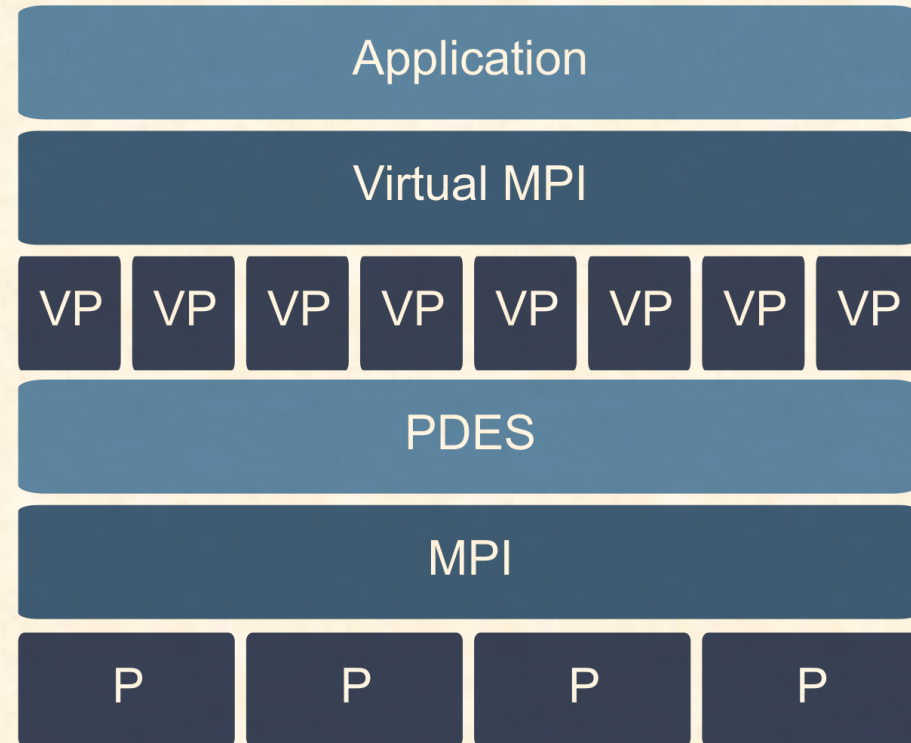
- Simulation toolkit for performance estimation
- Focused on accurate simulation of a few nodes
- *See Ron Brightwell's talk*

Objectives

- **Facilitating co-design of extreme scale systems through simulation of future architecture choices**
- **Performance and resilience evaluation of applications/ algorithms at extreme scale (million-to-billion concurrency)**
- **Lightweight simulation of system properties to enable investigations at extreme scale**
- **Bridging the gap between:**
 - **Time-inaccurate execution at extreme scale (JCAS and BigSim Emulator)**
 - **Time-accurate trace-driven simulation at extreme scale (BigSim Simulator and others)**

Technical Approach

- Combining highly oversubscribed execution, a virtual MPI, and a time-accurate PDES
- PDES uses the native MPI and simulates virtual processors
- The virtual processors expose a virtual MPI to applications
- Applications run within the context of virtual processors:
 - Global and local virtual time
 - Execution on native proc.
 - Local or native MPI comm.
 - Processor/network model

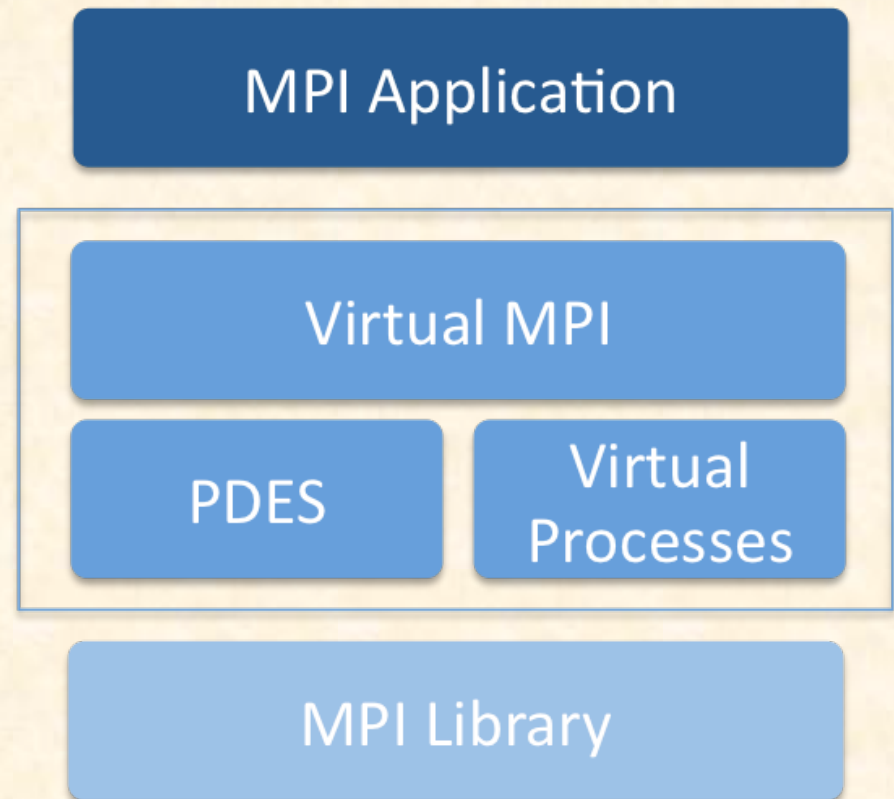


Implementation: Overview

- The simulator is a library
- Utilizes PMPI to intercept MPI calls and to hide the PDES
- Easy to use:
 - Replace the MPI header
 - Compile and link with the simulator library
 - Run the MPI program:

`mpirun -np <np> <prog> -xsim-np <vp>`

- C/C++ with 2 threads/nproc.
- Support for C and Fortran MPI applications



Implementation Details (1/2)

- **PDES**

- Maintains virtual time for each VP equivalent to execution time and scaled by the processor model
- Virtual MPI message latency is defined by the network model and maintained by the PDES
- PDES bootstrap sends a message to each VP to invoke the programs main
- Conservative execution without deadlock detection (not needed at this point)

- **Virtual Processes**

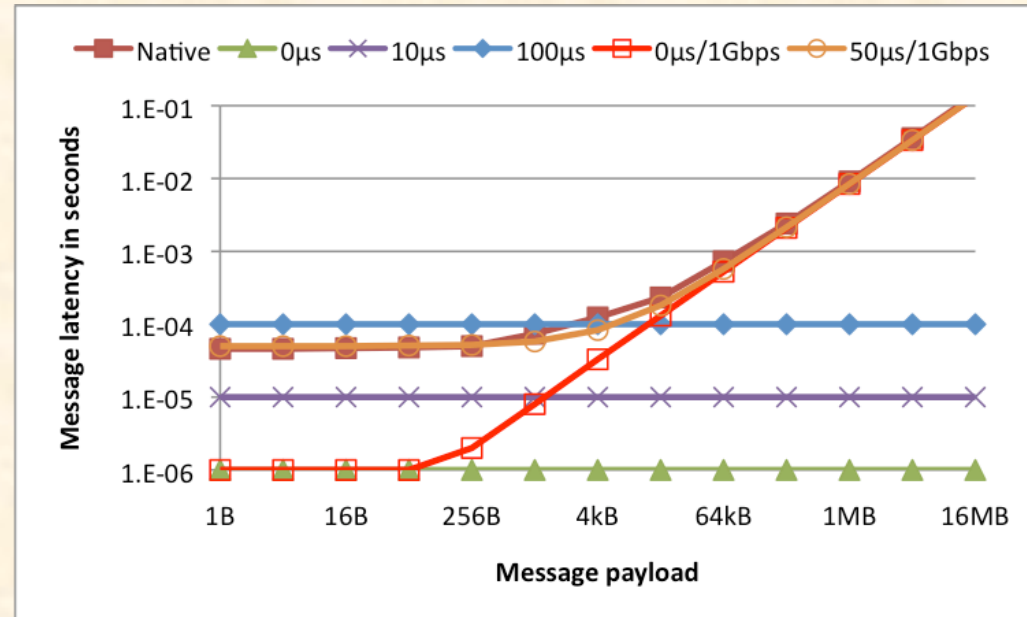
- Encapsulated in a user-space thread for efficient execution at extreme scale (100,000s VPs/OS)
- User-space (pthread) stack frame management
- User-space context switch
- Customizable stack size (via command line option)
- Intel 32/64-bit (x86/x86_64)
- Risk of stack overflow

Implementation Details (2/2)

- **Virtual MPI**
 - Reimplementation of MPI atop VP messaging
 - Basic MPI functionality
 - MPI_COMM_WORLD|SELF supported communicators
 - Non-/blocking P2P comm., broadcast and barrier
 - MPI_Wtime and MPI_Wtick
 - Causality mechanism for MPI_Waitsome
 - No MPI_ANY_SOURCE
- **Virtual MPI & PDES**
 - Virtual clock is off during virtual MPI calls
 - Message sends occur immediately (add to queue for local VP or native MPI_Send for remote VP)
 - Message receive time is defined by network model
 - Queued messages are ordered by receive time
 - Virtual MPI msg. receive performs a context switch to the addressee of the first message in the queue

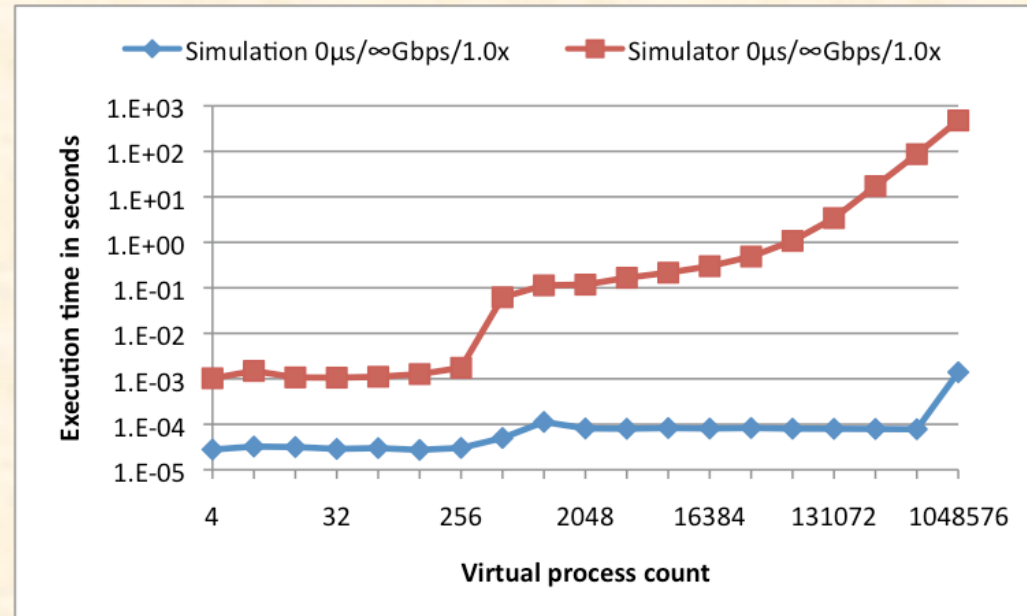
Experimental Results: Network Model

- Model allows to define network architecture, latency and bandwidth
- Basic star network at the time of writing this paper
- Model can be set to $0\mu\text{s}$ and ∞Gbps as baseline
- $50\mu\text{s}$ and 1Gbps roughly represented the native test environment
 - 4 Intel dual-core 2.13GHz nodes with 2GB of memory each
 - Ubuntu 8.04 64-bit Linux
 - Open MPI 1.4.2 with multi-threading support



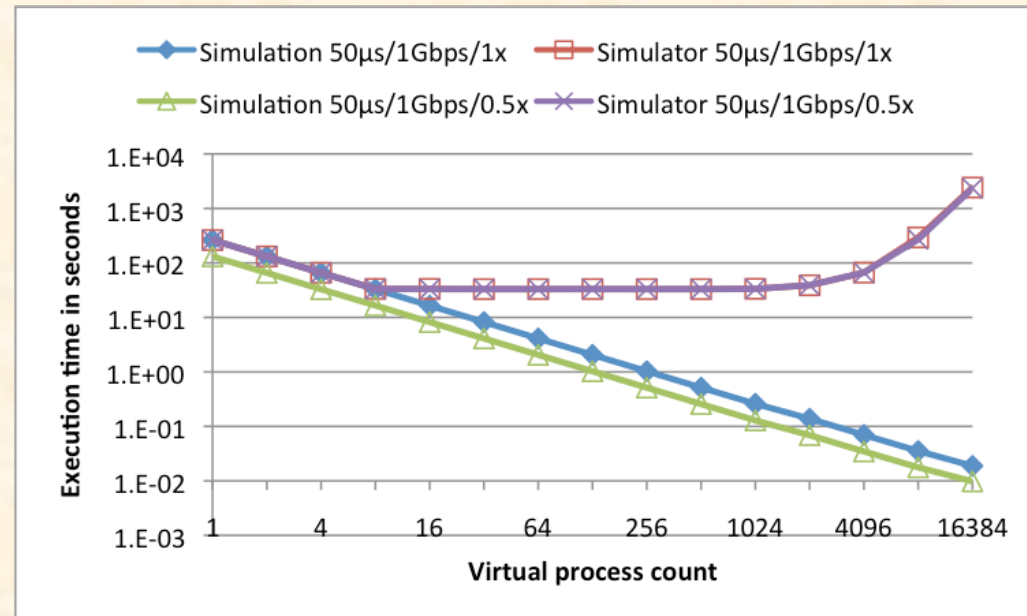
Experimental Results: Processor Model

- Model allows to set relative speed to a future processor
- Basic scaling model
- Model can be set to 1.0x for baseline numbers
- MPI hello world scales to 1M+ VPs on 4 nodes with 4GB total stack (4kB/VP)
- Simulation (application)
 - Constant execution time
 - <1024 VPs: Noisy clock
- Simulator
 - >256 VPs: Output buffer issues



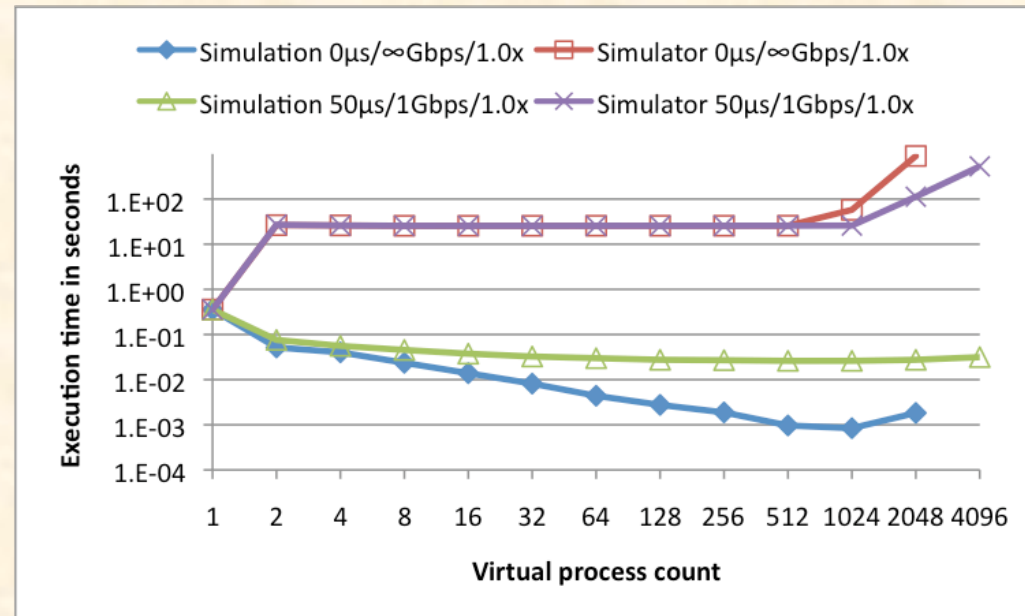
Experimental Results: Scaling up a computation-intensive application

- Basic PI Monte Carlo solver
- Network model:
 - Star, 50 μ s and 1Gbps
- Processor model
 - 1x (32kB stack/VP)
 - 0.5x (32kB stack/VP)
- Simulation (application)
 - Perfect scaling
- Simulator
 - ≤ 8 VPs: 0% overhead on the 8 processor cores
 - ≥ 4096 VPs: comm. load dominates



Experimental Results: Scaling up a communication-intensive application

- Basic 1-D Heat Eq. solver
- Network model:
 - Star, 50 μ s and 1Gbps
 - Star, 0 μ s and ∞ Gbps
- Processor model
 - 1x (32kB stack/VP)
- Simulation (application)
 - Limited scaling
- Simulator
 - 1 VP: no communication, therefore no overhead
 - ≥ 1024 VPs: comm. load dominates



Ongoing and Future Work

Ongoing Work:

- Re-implementation in pure C
- Sophisticated network model:
 - Different architectures
 - Multi-core multi-processor NUMA nodes
 - Network-on-chip
- Better MPI support
 - Groups/Communicators
 - Collectives
 - MPI_ANY_SOURCE
- Stack overflow protection

Future Work:

- Better processor model:
 - Interpolation using performance counters
- Resilience evaluation using fault injection:
 - Process faults
 - Communication link faults
 - Silent data corruption
- Dependability benchmarking
- Robustness testing

Questions?