Interconnection Network Models for Large-Scale Performance Prediction

Kishwar Ahmed, Mohammad Obaida, Jason Liu, Florida International University, FL, USA

Stephan Eidenbenz, Nandakishore Santhi, Joe Zerr, Los Alamos National Laboratory, NM, USA

4th Summer of CODES
July 17-18, 2018, Argonne National Laboratory, IL, USA

Outline

- Motivation
- Performance Prediction Toolkit (PPT)
- Automatic Performance Prediction
- Conclusion

Motivation

- Rapid changes in HPC architecture
  - Multi-core and many-core architecture
  - Accelerator technologies
  - Complex memory hierarchies
- HPC software adaptation is a constant theme:
  - *No code is left behind*: must guarantee good performance
  - Need high-skilled software architects and computational physicists
- **Need modeling and simulation of large-scale HPC systems and applications**
  - And the systems are getting larger (exascale systems around the corner)

HPC Performance Prediction

- HPC performance prediction provides **insight** about
  - Applications (e.g., scalability, performance variability)
  - Hardware/software (e.g., better design)
  - Workload behavior (present and future)
- Which is **useful** for –
  - Understanding application performance issues
  - Improving application and system
  - Budgeting, designing efficiency systems (present and future)

Our Goals for Rapid Performance Prediction

- Easy integration with other models of varying abstraction
- Easy integration with applications (e.g., physics code)
- Short development cycles
- Performance and scale

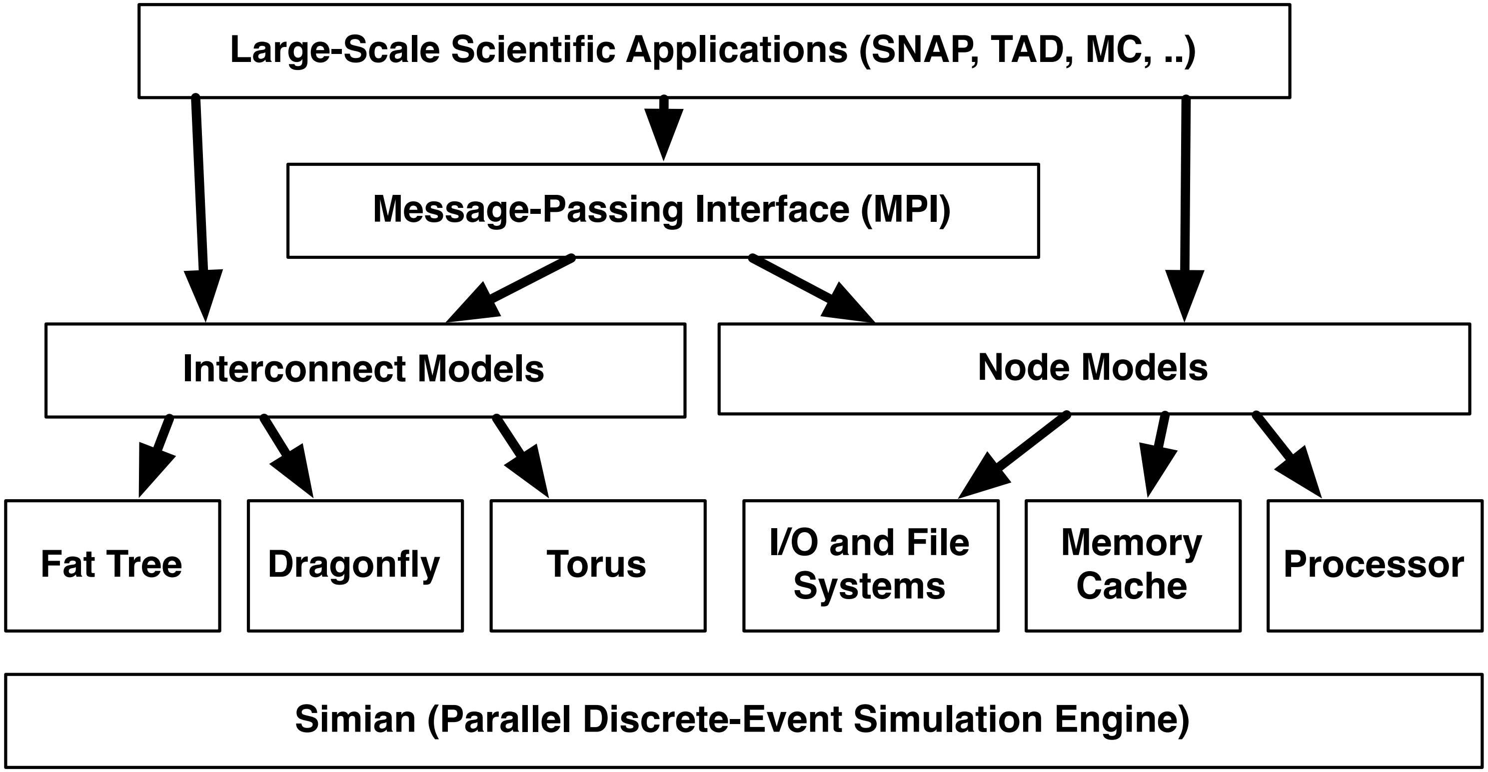
Outline

- Motivation
- Performance Prediction Toolkit (PPT)
- Automatic Performance Prediction
- Conclusion

Performance Prediction Toolkit (PPT)

- **Make it simple, fast, and most of all useful**
  - Designed to allow rapid assessment and performance prediction of large-scale applications on existing and future HPC platforms
  - PPT is a library of models of computational physics applications, middleware, and hardware
    - That allows users to predict execution time by running pseudo-code implementations of physics applications
- "Scalable Codesign Performance Prediction for Computational Physics" project

PPT Architecture



Simian: PDES using Interpreted Languages

- Open-source general purpose parallel discrete-event library
- Independent implementation in three interpreted languages: Python, LUA, and JavaScript
- Minimalistic design: LOC = 500 with 8 common methods (python implementation)
- Simulation code can be Just-In-Time (JIT) compiled to achieve very competitive event-rates
- Support process-oriented world-view (using Python greenlets and LUA coroutines)

Integrated MPI Model

- Developed based on Simian (entities, processes, services)
- Include all common MPI functions
  - Point-to-point and collective operations
  - Blocking and non-blocking operations
  - Sub-communicators and sub-groups
- Packet-oriented model
  - Large messages are broken down into packets (say, 64B)
- Reliable data transfer
  - Acknowledgement, retransmission, etc.

Table 1: Implemented MPI Functions

MPI_Send	blocking send (until message delivered to destination)
MPI_Recv	blocking receive
MPI_Sendrecv	send and receive messages at the same time
MPI_Isend	non-blocking send, return a request handle
MPI_Irecv	non-blocking receive, return a request handle
MPI_Wait	wait until given non-blocking operation has completed
MPI_Waitall	wait for a set of non-blocking operations
MPI_Reduce	reduce values from all processes, root has final result
MPI_Allreduce	reduce values from all, everyone has final result
MPI_Bcast	broadcast a message from root to all processes
MPI_Barrier	block until all processes have called this function
MPI_Gather	gather values form all processes at root
MPI_Allgather	gather values from all processes and give to everyone
MPI_Scatter	send individual messages from root to all processes
MPI_Alltoall	send individual messages from all to all processes
MPI_Alltoallv	same as above, but each can send different amount
MPI_Comm_split	create sub-communicators
MPI_Comm_dup	duplicate an existing communicator
MPI_Comm_free	deallocate a communicator
MPI_Comm_group	return group associated with communicator
MPI_Group_size	return group size
MPI_Group_rank	return process rank in group
MPI_Group_incl	create new group including all listed
MPI_Group_excl	create new group excluding all listed
MPI_Group_free	reclaim the group
MPI_Cart_create	add cartesian coordinates to communicator
MPI_Cart_coords	return cartesian coordinates of given rank
MPI_Cart_rank	return rank of given cartesian coordinates
MPI_Cart_shift	return shifted source and destination ranks

MPI Example

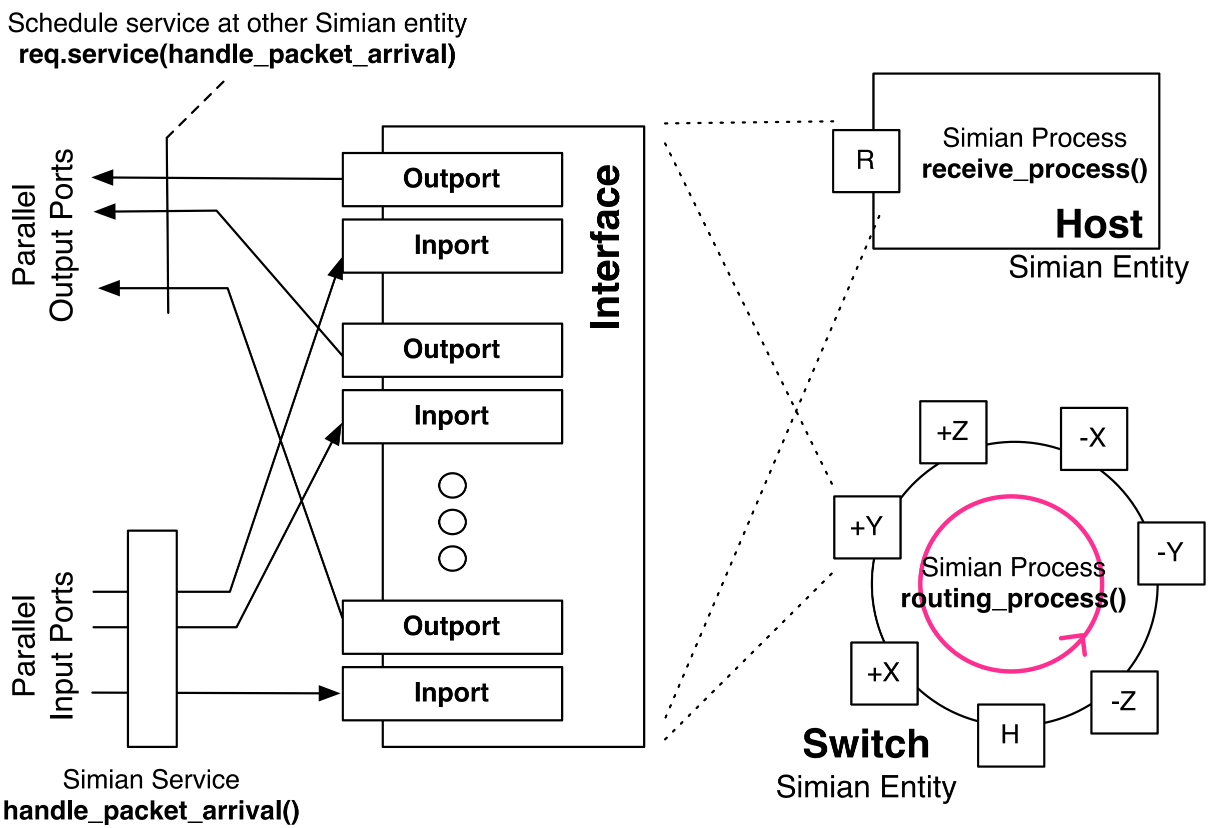
```
from ppt import *

# config hopper (17x8x24 gemini interconnect)
model_cfg = { # a dictionary
  "intercon_type" : "gemini",
  "host_type" : "mpihost",
  "torus" : configs.hopper_intercon,
  "mpiopt" : configs.gemini_mpiopt,
}
model = HPCSim(model_cfg, ..)

# mpi main function, n is matrix dimension
def cannon(mpi_comm_world, n):
  ... # we describe this later

# start 16 mpi ranks, pass matrix dimension
model.start_mpi(range(16), cannon, 10000)

# simulation starts
model.run()
```

Hardware configuration

MPI application

Run MPI

Interconnect Model



Interconnect model using Simian entities, processes, and services

Interconnect Model (Contd.)

- Common interconnect topologies
  - Torus (Gemini, Blue Gene/Q)
  - Dragonfly (Aries)
  - Fat-tree (Infiniband)
- Some properties:
  - Emphasis on production systems
    - Cielo, Darter, Edison, Hopper, Mira, Sequoia, Stampede, Titan, Vulcan, …
  - Seamlessly integrated with MPI
  - Scalable to large number of nodes
  - Detailed congestion modeling

3D Torus – Cray's Gemini Interconnect

- 3D torus direct topology
- Each building block
  - 2 compute nodes
  - 10 torus connections
    - ±X*2, ±Y, ±Z*2
- Examples: Jaguar (ORNL), Hopper (NERSC), Cielo (LANL)

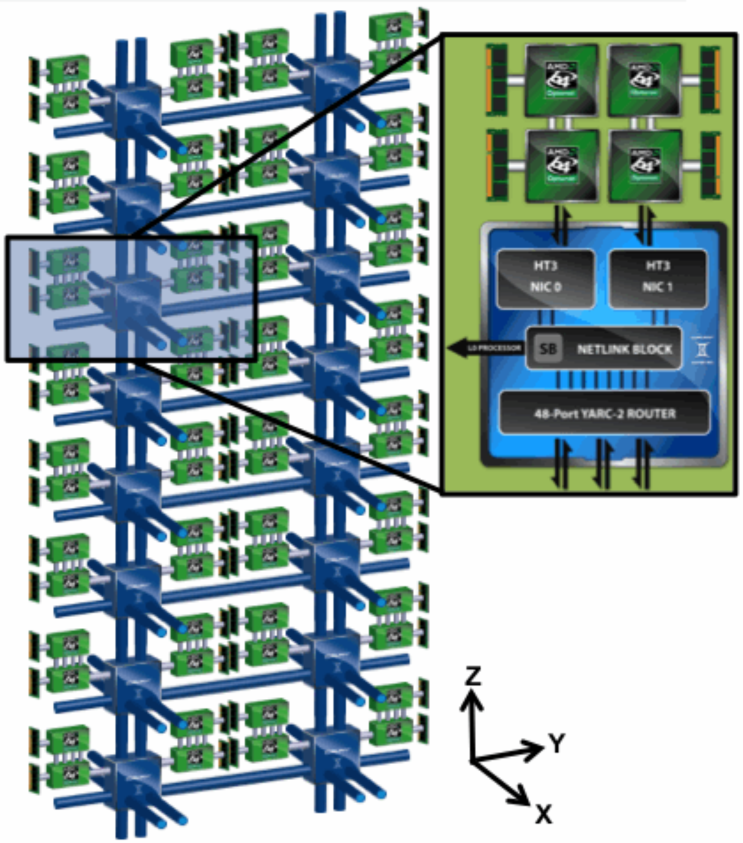
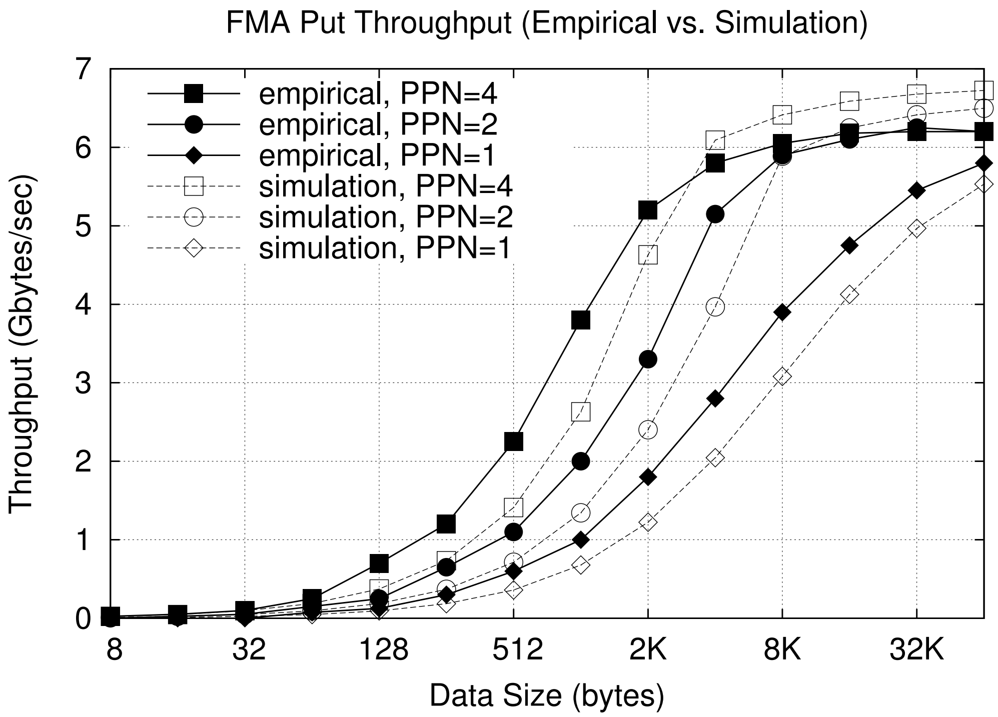


Image courtesy of Cray, Inc.

Gemini Validation

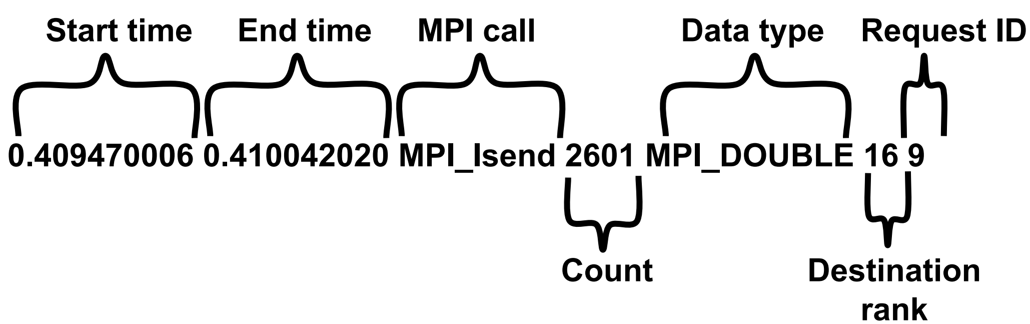
Compared against empirical results from Hopper @ NERSC



Gemini FMA put throughput (as reported in [2]) versus simulated throughput as a function of transfer size for 1, 2, and 4 processes per node.

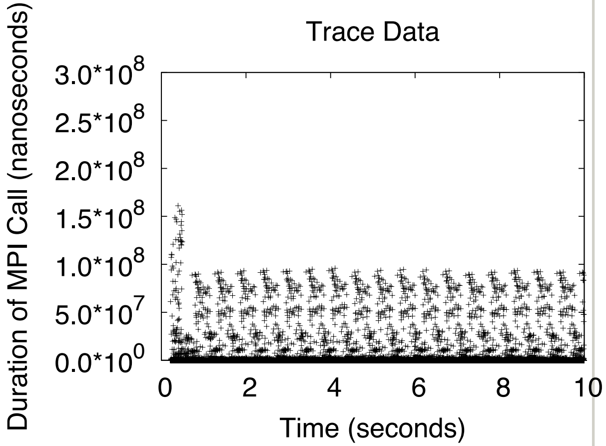
Trace-Driven Simulation

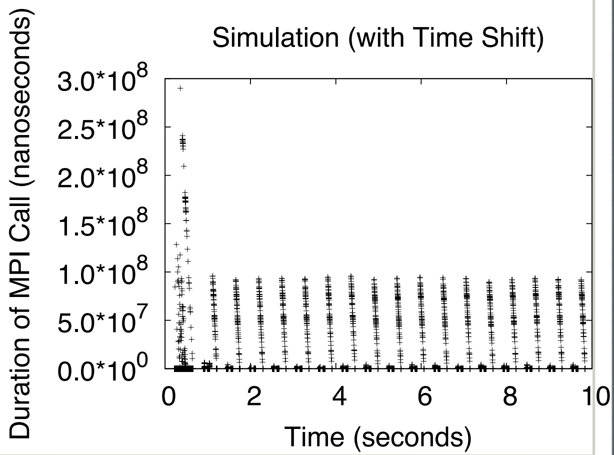
- Mini-app MPI traces:
  - Trace generated when running mini - apps on NERSC Hopper (Cray XE06) with <=1024 cores
  - Trace contains information of the MPI calls (including timing, source/destination ranks, data size, …)



Trace-Driven Simulation (Contd.)

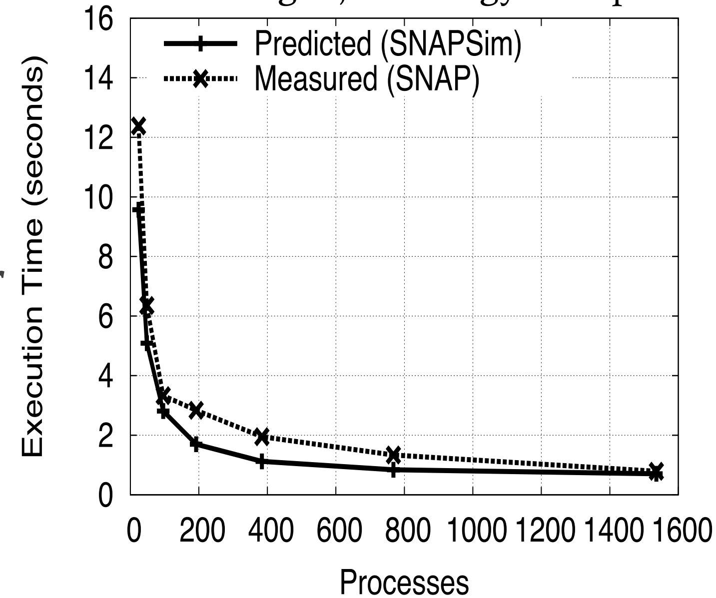
- For this experiment, we use:
  - LULESH mini-app from ExMatEx
  - 64 MPI processes
- Run trace for each MPI rank
  - Start MPI call at exactly same time indicated in trace file
  - Store completion time of MPI call
  - Compare it with the completion time in trace file





Case Study: SN Application Proxy

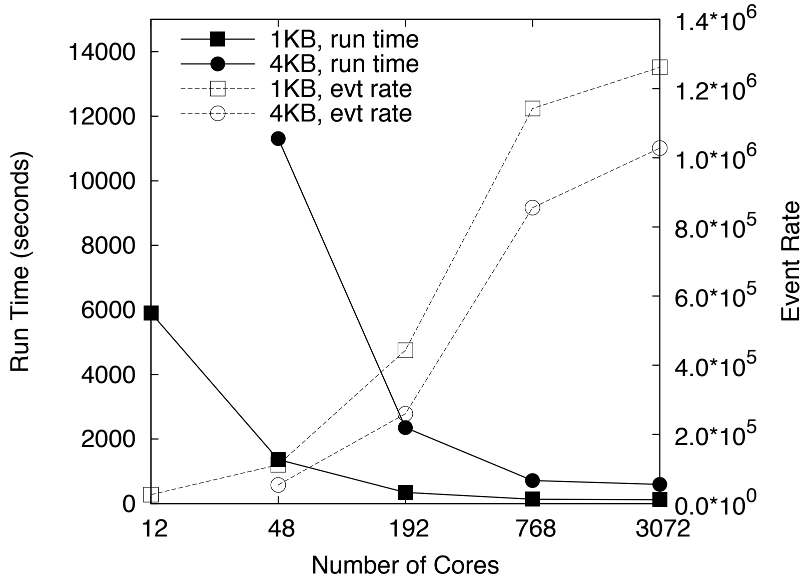
- SNAP is a "mini-app" for PARTISN
- PARTISN is code for solving radiation transport equation for neutron and gamma transport
- Use MPI to facilitate communication
- Use node model to compute time



NERSC's Edison supercomputer, which is Cray XC30 system with Aries interconnect

Parallel Performance

- 1500-node cluster at LANL, connected by an Infiniband QDR interconnect
- MPI_Allreduce, with different data size (1K or 4K)
- Three times event-rate (C++ parallel simulator: MiniSSF)



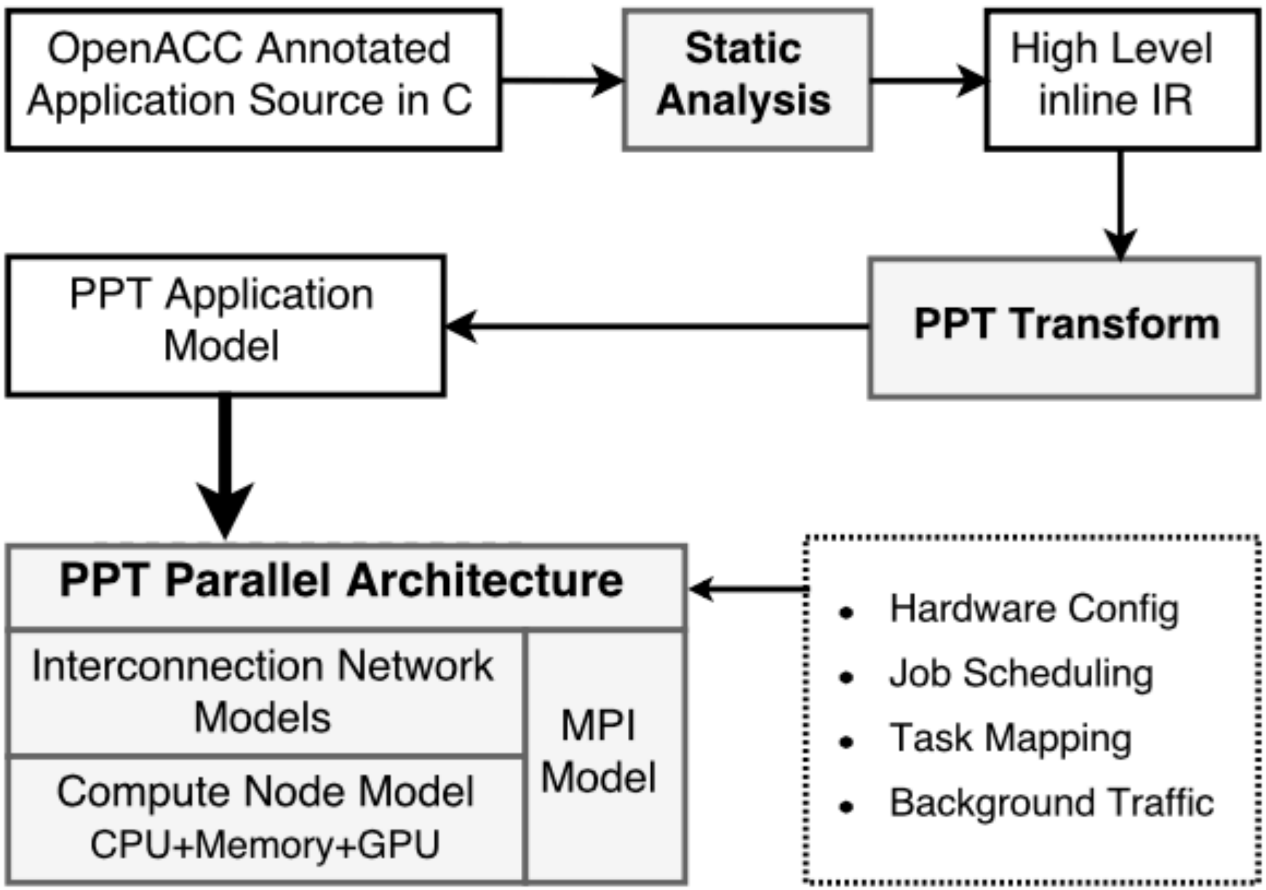
Outline

- Motivation
- Performance Prediction Toolkit (PPT)
- Automatic Performance Prediction
- Conclusion

The Framework

- Purpose is to maintain **accuracy** and **performance**, **flexibility**, and **scalability**; so as to do studies of **large-scale applications**
- Steps of an application performance analysis
  - Start with an application program
  - Statically analyze the program to build an abstract model
  - Transform into an executable model (encompassing CPU, GPU, and communication)
  - Run model with HPC simulation (for performance prediction)

The Framework (Contd.)



Static Analysis

- Derive an abstract model
  - GPU computation
    - Identify GPU kernels
    - Based on COMPASS
    - Obtain workload (flops and memory loads/stores)
  - CPU computation
    - Transform source code to IR using LLVM
    - Using Analytical Memory Model (AMM) model resource-specific operations (e.g., loads, stores)

GPU Model Building

- OpenARC provides
  - Memory-GPU transfers and vice versa, loads, stores, flops, etc.
- Build GPU-warp task-list from OpenARC-generated IR

GLOB_MEM_ACCESS	access GPU on-chip global memory
iALU	integer operations
diALU	double precision integer operations
fALU	floating point operations(flops)
dfALU	double precision flop
SFU	special function calls
L1_ACCESS	direct GPU L1 accesses
L2_ACCESS	direct GPU L1 accesses
DEVICE_SYNC	synchronize GPU threads
THREAD_SYNC	synchronize GPU threads w/ CPU

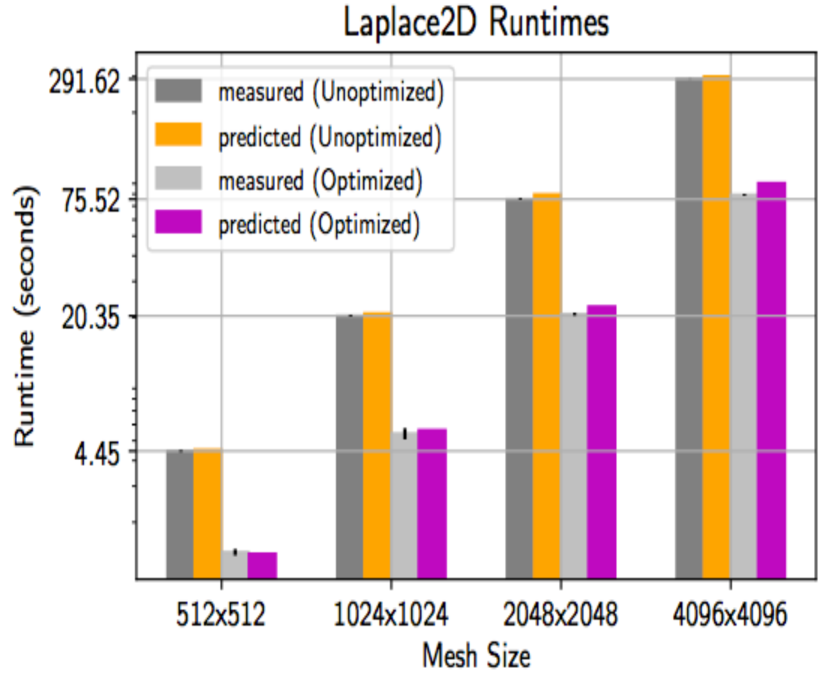
alloc	[host] memory alloc. in # of bytes
unalloc	[host] memory de-allocate
DEVICE_ALLOC	device allocations
DEVICE_TRANSFER	device transfers
KERNEL_CALL	call a GPU kernel with block/grid

Execution Model

- Launch application model on PPT
- PPT features
  - Hardware models (processor, memory, GPU)
  - Full-fledged MPI model
  - Detailed interconnect models
  - Large-scale workload model

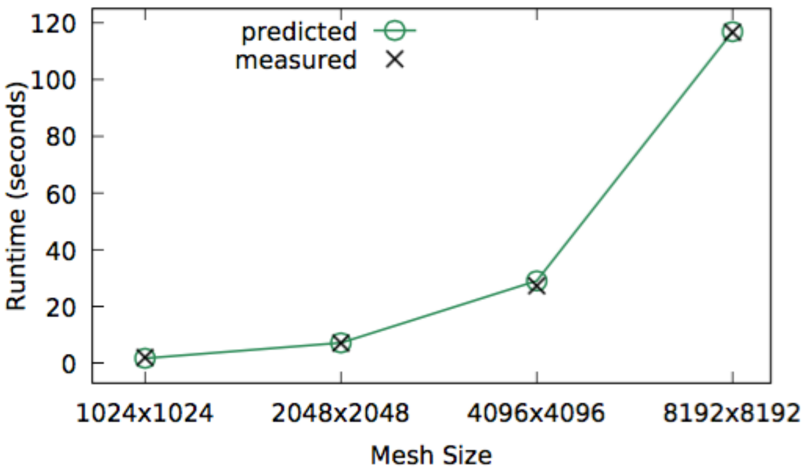
Experiment: Runtime Prediction (CPU)

- Laplace 2D benchmark
  - Compute-intensive application
  - Four different mesh sizes
  - With and without compiler optimizations
- Two Intel Xeon processors running at 2.4GHz frequency
- Observations
  - 7.08% error (with optimizations)
  - 3.12% error (without optimizations)



Experiment: Runtime Prediction (GPU)

- Application: Laplace 2D MM
- Two 8-core Xeon E5-5645 @2.1 GHz
- NVIDIA Geforce GM 204
- Observations:
  - 13.8% error for 1024 X 1024
  - 0.16% error for 8192 X 8192



Outline

- Motivation
- Performance Prediction Toolkit (PPT)
- Automatic Performance Prediction
- Conclusion

Conclusion

- Building a full HPC performance prediction model
- PPT – Performance Prediction Toolkit
- MPI model and interconnection network models (torus, dragonfly, fat-tree)
- Automatic application performance prediction
- Future work:
    - Apply dynamic analysis and ML for irregular applications
    - Automatic application optimization framework

References

- An Integrated Interconnection Network Model for Large-Scale Performance Prediction, Kishwar Ahmed, Mohammad Obaida, Jason Liu, Stephan Eidenbenz, Nandakishore Santhi, and Guillaume Chapuis. 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS 2016), May 2016.
- Scalable Interconnection Network Models for Rapid Performance Prediction of HPC Applications, Kishwar Ahmed, Jason Liu, Stephan Eidenbenz, and Joe Zerr. 18th International Conference on High Performance Computing and Communications (HPCC 2016), December 2016.
- The Simian Concept: Parallel Discrete Event Simulation with Interpreted Languages and Just-in-Time Compilation, Nandakishore Santhi, Stephan Eidenbenz, and Jason Liu. 2015 Winter Simulation Conference (WSC 2015), December 2015.
- Parallel Application Performance Prediction Using Analysis Based Models and HPC Simulations, Mohammad Abu Obaida, Jason Liu, Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. 2018 SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS'18), May 2018.

Thank you! Questions?