SAND2013-2250P

Execution Models for Exascale – A Bottom-Up Approach – (EMBU)

Sandia National Laboratories

Sandia National Laboratories
Robert Clay
Mike Heroux
Gilbert Hendry (ghendry@sandia.gov)

BERKELEY LAB

Lawrence Berkeley National Laboratory
John Shalf
Nick Wright

CENTER FOR RESEARCH IN EXTREME SCALE TECHNOLOGIES
INDIANA UNIVERSITY
Pervasive Technology Institute

Indiana University
Thomas Sterling
Matt Anderson

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Project Goals

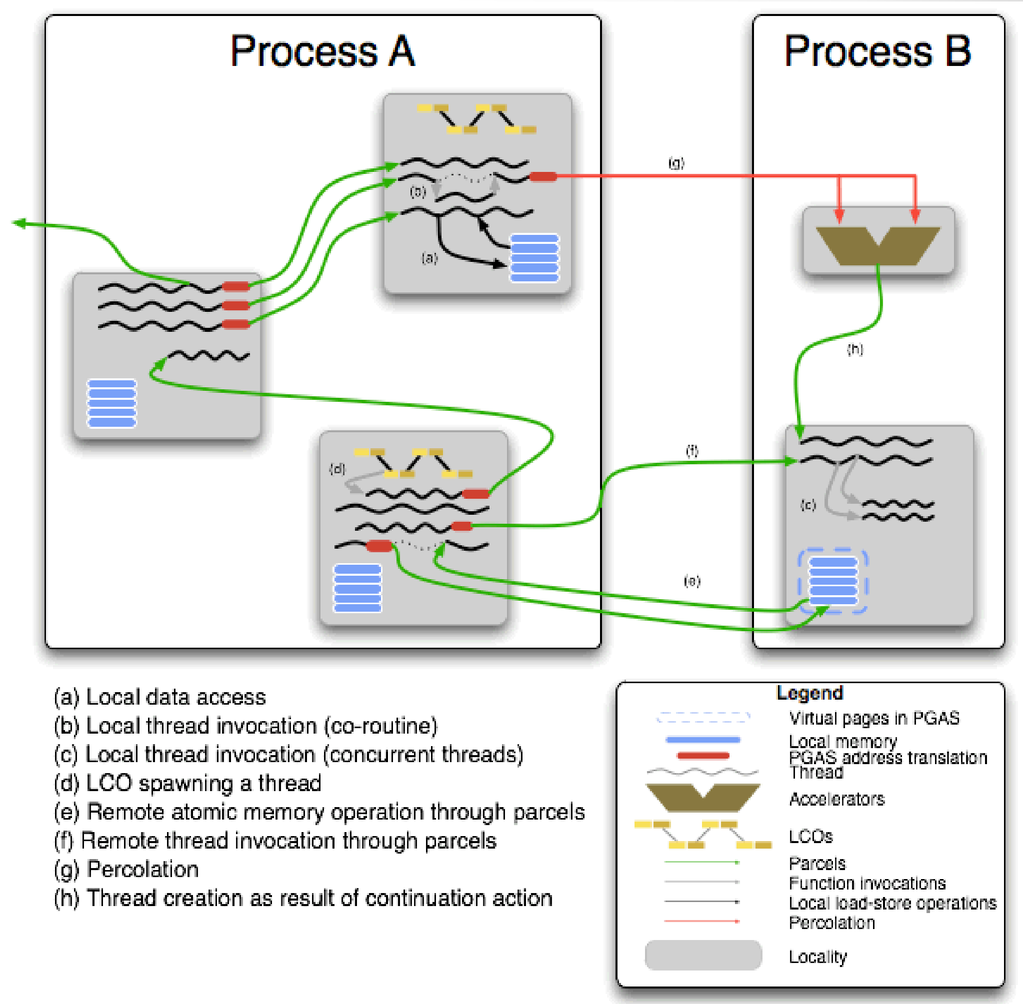
1. Develop the simulation capability to answer questions involving exascale execution models:
    1. What are the characteristics of applications using implementations of different execution models as they scale?
    2. What system-level architectural features are critical, given implementations of different execution models
    3. What software features are critical in implementations of different execution models related to parallel scalability

Simulator-based Approach

- Implement rough implementations of different execution models in SST/macro
  - capture what we want to observe, nothing more
  - is easier, faster than doing the real thing
  - allows us to easily scale, and introspect
  - SST/macro and HPX implementation can be found at:
    - bitbucket.org/ghendry/sstmacro
- Port applications to SST/macro execution model implementations
  - APIs are the same (between SST/mac and real thing)
  - skeletonize, for faster execution and rapid prototyping
  - Have GTC, HPCCG

ParalleX Execution Model

- Lightweight multi-threading
  - Divides work into smaller tasks
  - Increases concurrency
- Message-driven computation
  - Move work to data
  - Keeps work local, stops blocking
- Constraint-based synchronization
  - Declarative criteria for work
  - Event driven
  - Eliminates global barriers
- Data-directed execution
  - Merger of flow control and data structure
- Shared name space
  - Global address space
  - Simplifies random gathers



Slide Credit: Thomas Sterling, Indiana University

About SST/macro HPX Implementation

- Almost identical API, so resulting application code is the nearly the same as real HPX
- However, the API is not as filled out as the real thing. Only more commonly used functions/components are implemented
- Default AGAS implemented as distributed hash.
- Do have a model for on-node thread manager for oversubscription, though focus is on the parallel scalability aspects

Using HPCCG: conjugant gradient solver from Mantevo

- Use it because it is extremely simple
  - it's not very predictive of any app, lacks real context around the conjugate gradient solver
  - but it's easy to understand, and easy to work with
- We have GTC around, but it takes a while to simulate.

Some small experiments investigating some aspects of Execution Models

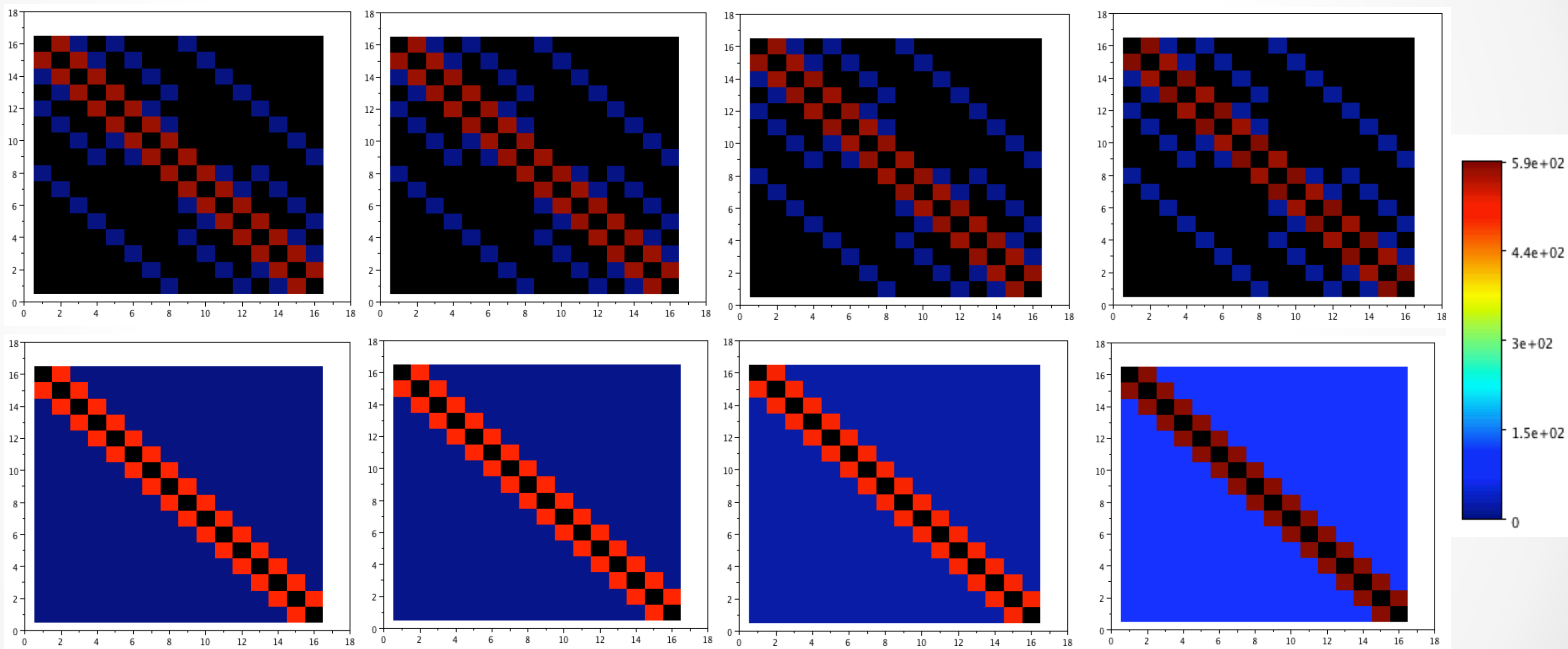
- Compare general application characteristics using MPI and HPX
- Explore sensitivity of EM to architectural parameters
  - Figure out which ones are important for each EM
  - Some key parameters:
    - network latency, bandwidth, topology
    - memory bandwidth

Communication characteristics for MPI and HPX are different, namely in collectives

Ranks = 16 nodes = 16 | Ranks = 32 nodes = 16 | Ranks = 64 nodes = 16 | Ranks = 128 nodes = 16

MPI

HPX

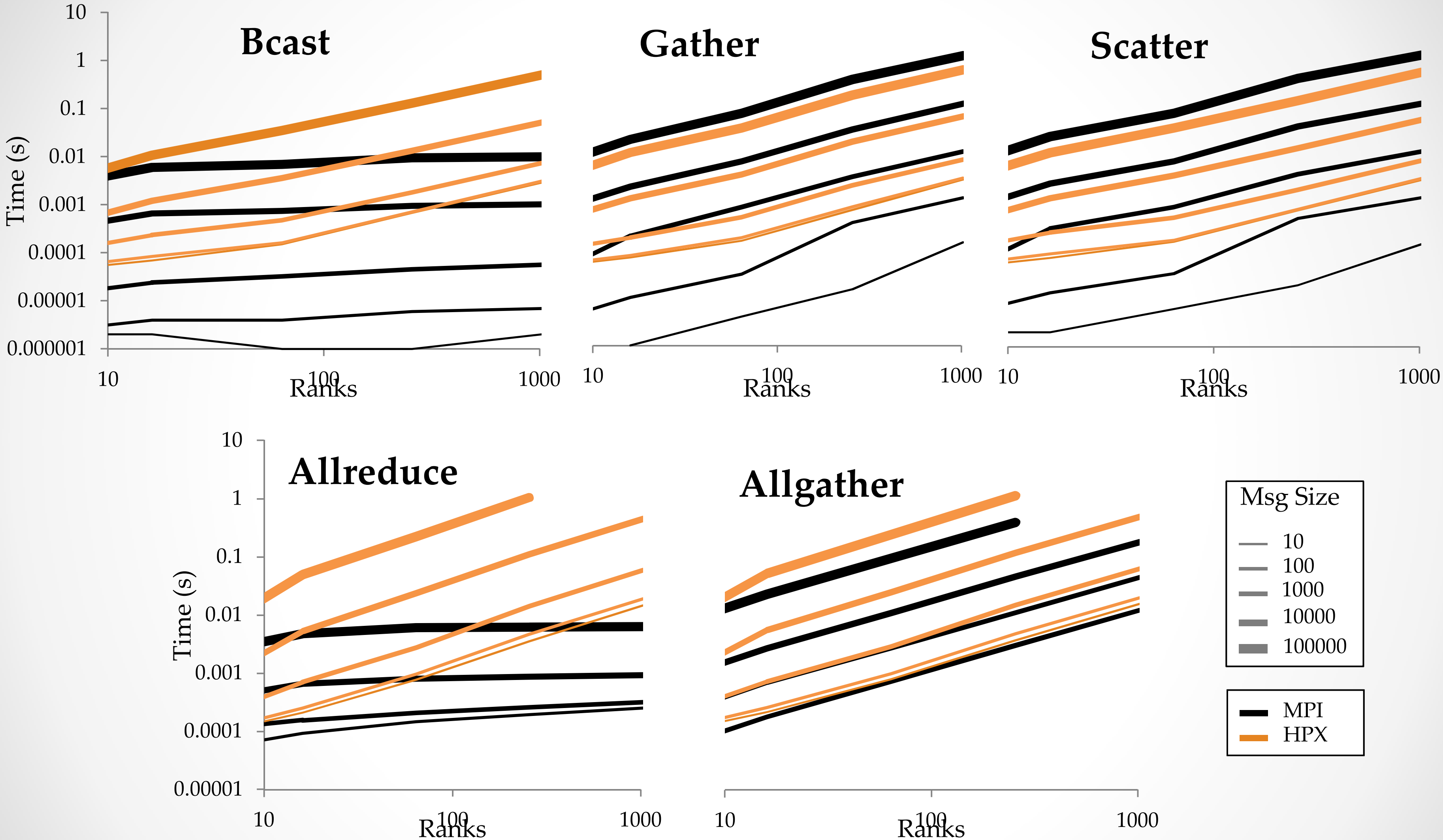


- HPCCG is mostly nearest-neighbor in 1D
- HPX collective implementation is all-to-all to exploit asynchrony
    - produces a lot of traffic

Let’s ask a fundamental question

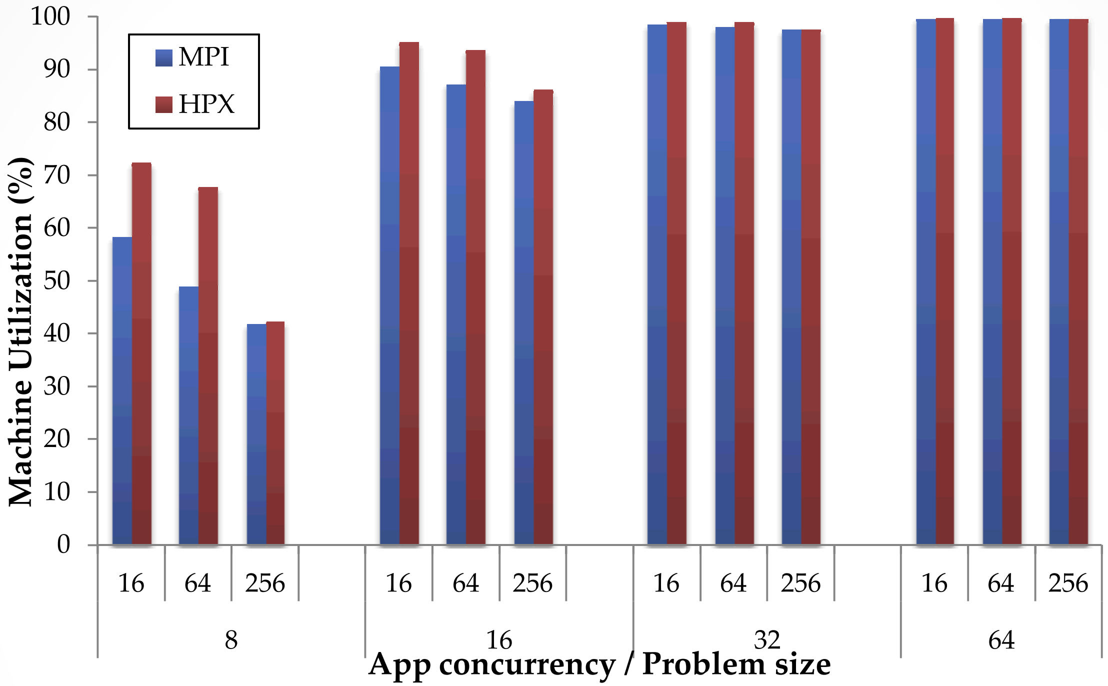
- MPI collectives are optimized for reducing network traffic
- HPX collectives are implemented to maximize asynchrony/bandwidth
- What are the performance implications of these two as they scale?
- Simple experiment: blocking collectives
    - If we can’t reformulate algorithms so that collectives aren’t blocking, how will HPX perform?

Collective comparisons



- 10 blocking collectives with 10ms of compute time each
- Bcast and Allreduce don't scale well in HPX
- Gather and Scatter do scale well in HPX
- Allgather is ok, some overhead

back to HPCCG: HPX can exploit asynchrony

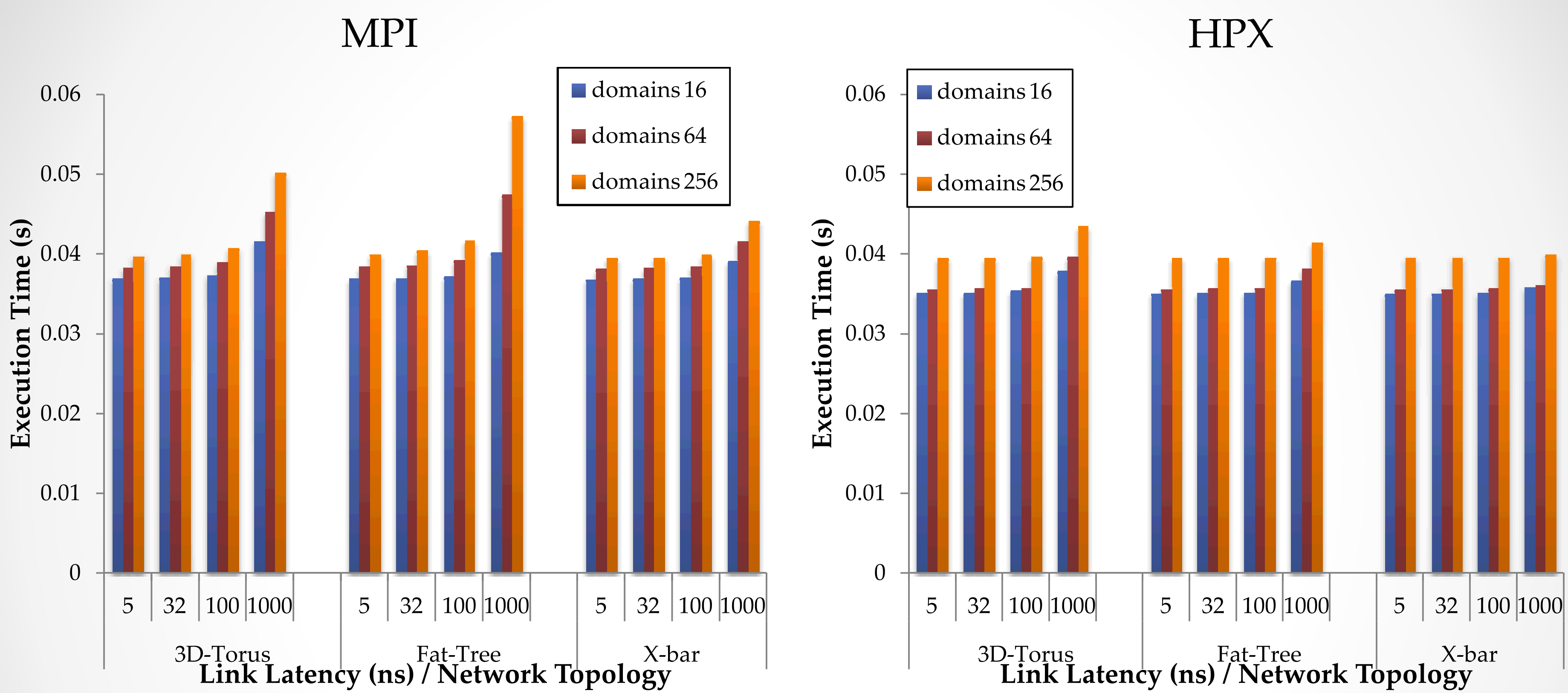


- Measuring machine utilization (avg compute time / avg total time) for different problem sizes and application concurrencies
- HPCCG has no problem weak scaling
  - we knew that, we're just playing with it because it's easy
- HPX does exploit some asynchrony for smaller problem sizes
- Larger problem sizes are compute-bound anyway

Some small experiments investigating some aspects of Execution Models

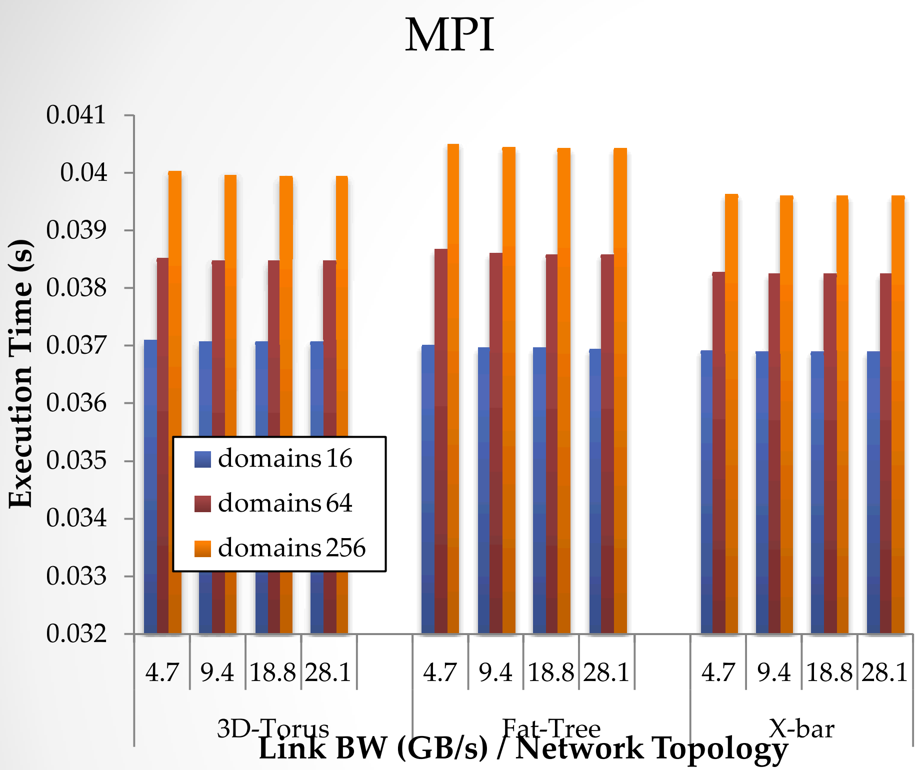
- Compare general application characteristics using MPI and HPX
- Explore sensitivity of EM to architectural parameters
  - Figure out which ones are important for each EM
  - Some key parameters:
    - network latency, bandwidth, topology
    - memory bandwidth

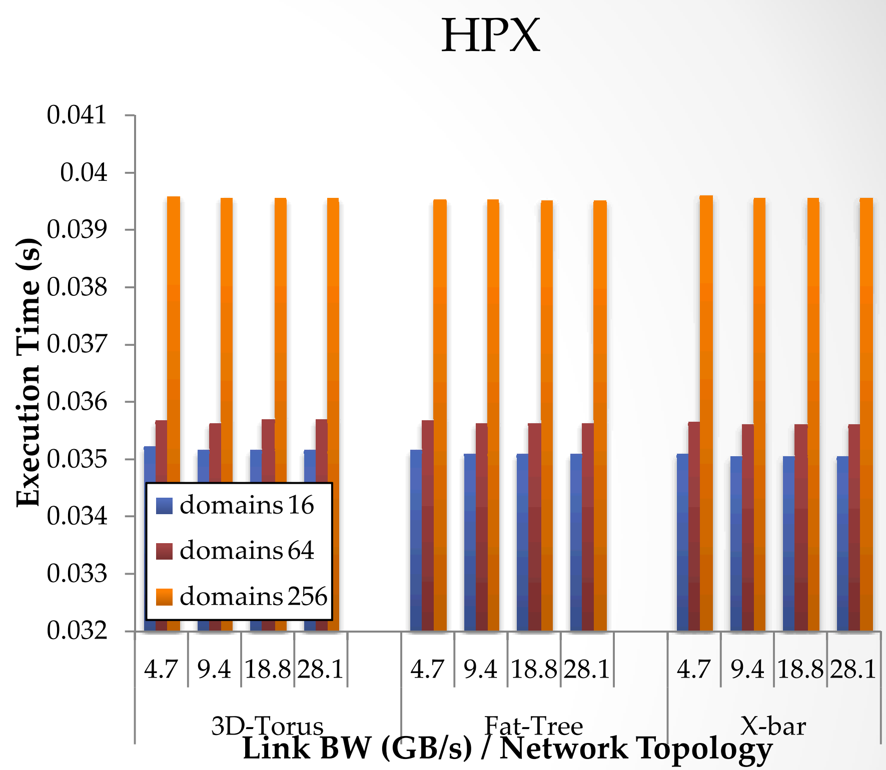
HPCCG sensitivity to network link latency



- HPX looks less sensitive to link latency
  - this is expected from asynchronous collectives, what HPX is after
- Also less sensitive to topology

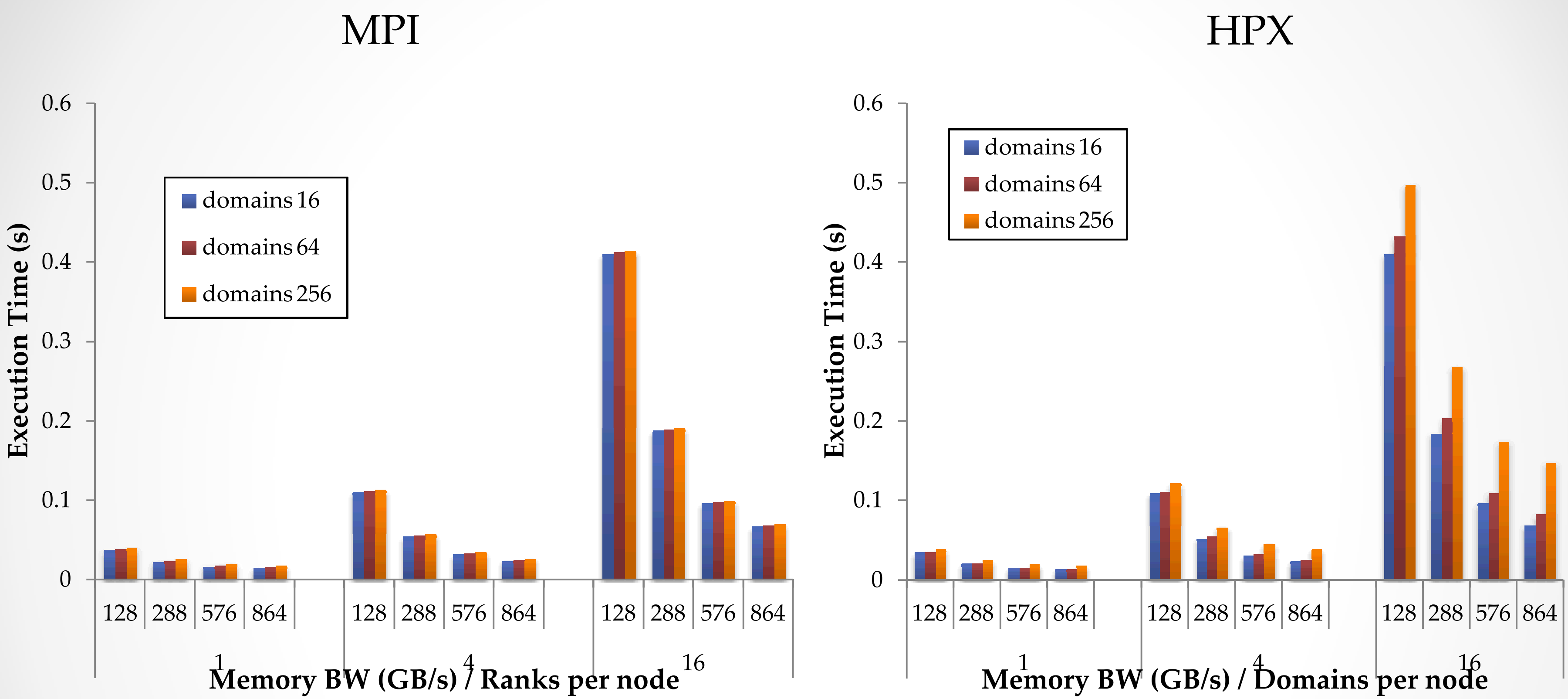
HPCCG sensitivity to Link BW





- HPCCG not really sensitive to link bw
  - could be limited by something else

HPCCG sensitivity to Memory BW



- More sensitive as you put more ranks/domains on a node
- HPX more sensitive than MPI

Technical Summary

- HPX is capable of exploiting global asynchrony to the extent possible
    - but avoid Allreduces and Bcasts or scalability will kill you
- HPX can be less sensitive to network parameters through latency hiding, which may be good considering the unknown exascale landscape

Future Work

- validate HPX implementation
- more apps, reformulation of *something*
- complete PGAS implementation in simulator (and validate) as a baseline alternative
    - PGAS is like an in-between point between MPI and HPX in that it has an asynchrony feature (one-sided communication) and a global naming feature (symmetric data). But it's still structured, and porting is pretty easy.
- Formal UQ parameter sensitivity analysis