Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Assessing the Impact of Execution Model Selection on Data Locality

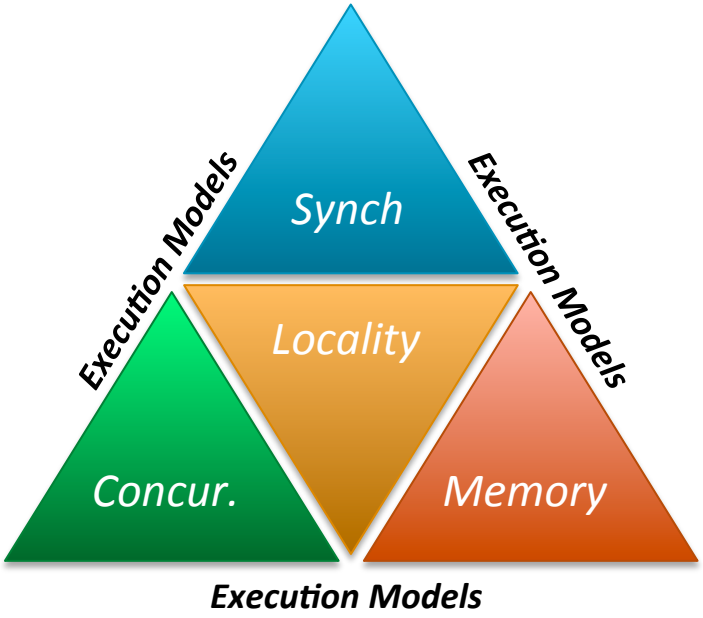
KEVIN J. BARKER, VINAY AMATYA, DANIEL CHAVARRÍA, ADOLFY HOISIE, SRIRAM KRISHNAMOORTHY, GOKCEN KESTOR, JOSEPH MANZANO

PACIFIC NORTHWEST NATIONAL LABORATORY

Workshop on Modeling and Simulation of Systems and Applications
August, 2016

Motivation: Execution Model Attributes

- Execution Models can be classified according to several *attributes*
  - These are not linearly independent
  - We have described the *SCaLeM* methodology:
    - Synchronization, Concurrency, Locality, and Memory
- These attributes trade off against one another in practice



Synch	Coordination between Concurrency Units
Locality	Differentiation between local and remote regions or units
Concurrency	Creation, management, and destruction of concurrency units
Memory	Availability of address ranges and operations on such ranges

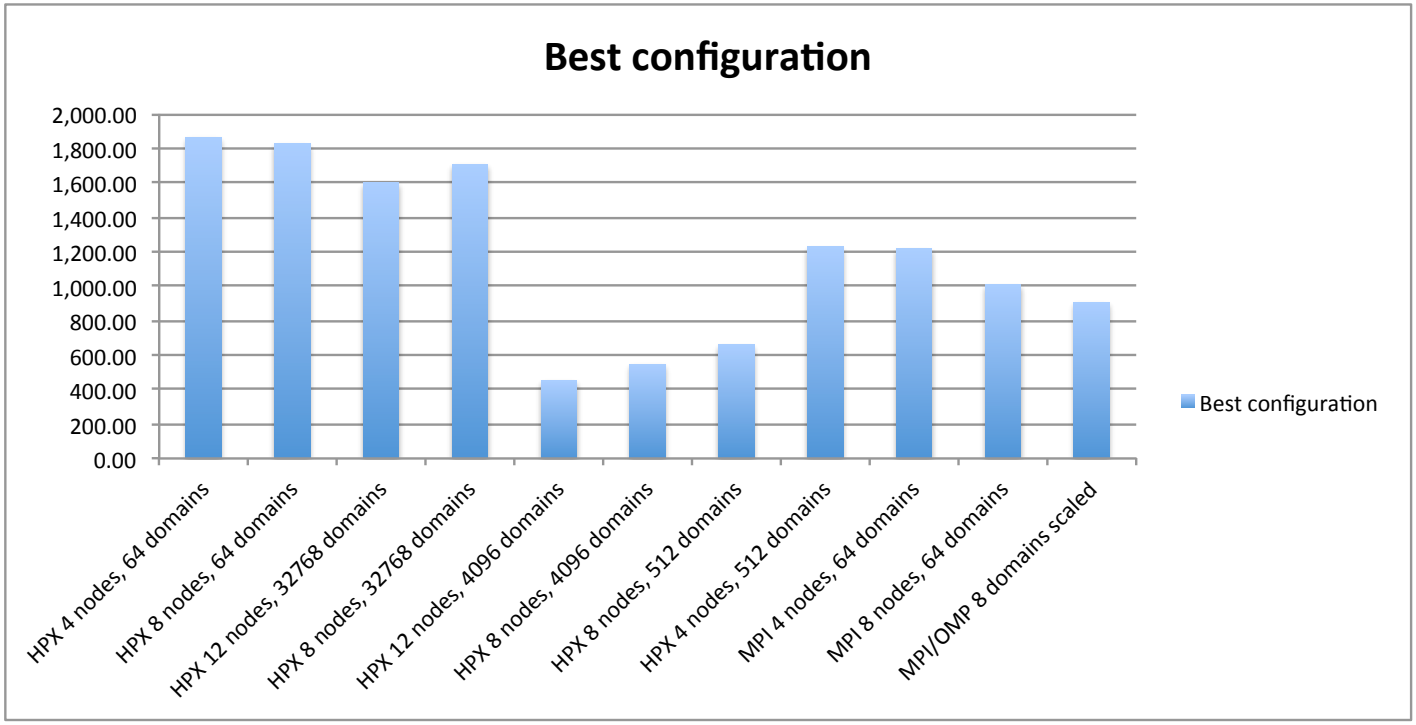
Motivation: Interplay between Attributes

- Application behavior may be sensitive to this interplay
  - Inherent in their algorithms
    - Fine-grained algorithms may incur large synchronization costs
  - Inherent in their data sets
    - Locality impacted by sparseness of data structures
  - Inherent in execution model implementation
    - High overheads for certain operations
    - Management of layouts for important (i.e., heavily used) data structures

- Objective: characterize this complex interplay for the attributes of Concurrency and Locality
  - I.e., how is data locality impacted within execution models that support fine-grained task decomposition?

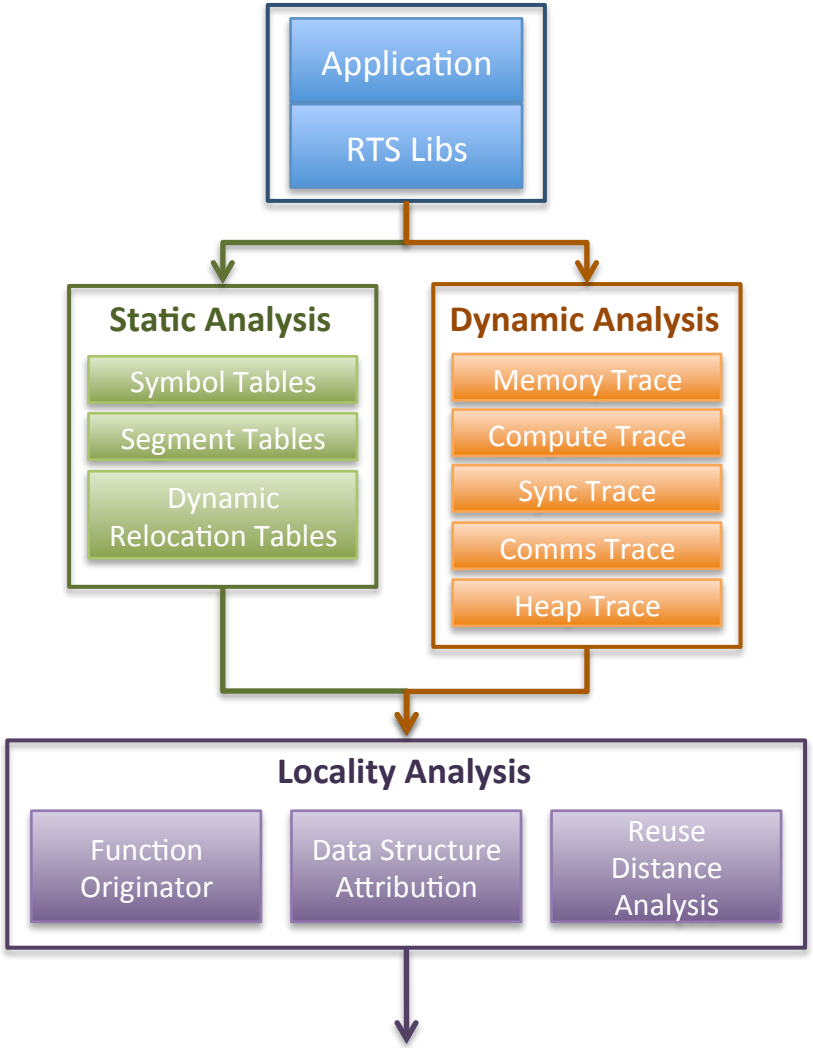
Overdecomposition: Impact on Performance

- Implemented using the ParalleX or CSP (using MPI+OpenMP) EMs
- Increased levels of over-decomposition yields improved performance
- Because of the cost of data movement, we want to understand the interaction between *concurrency* and *locality* EM attributes



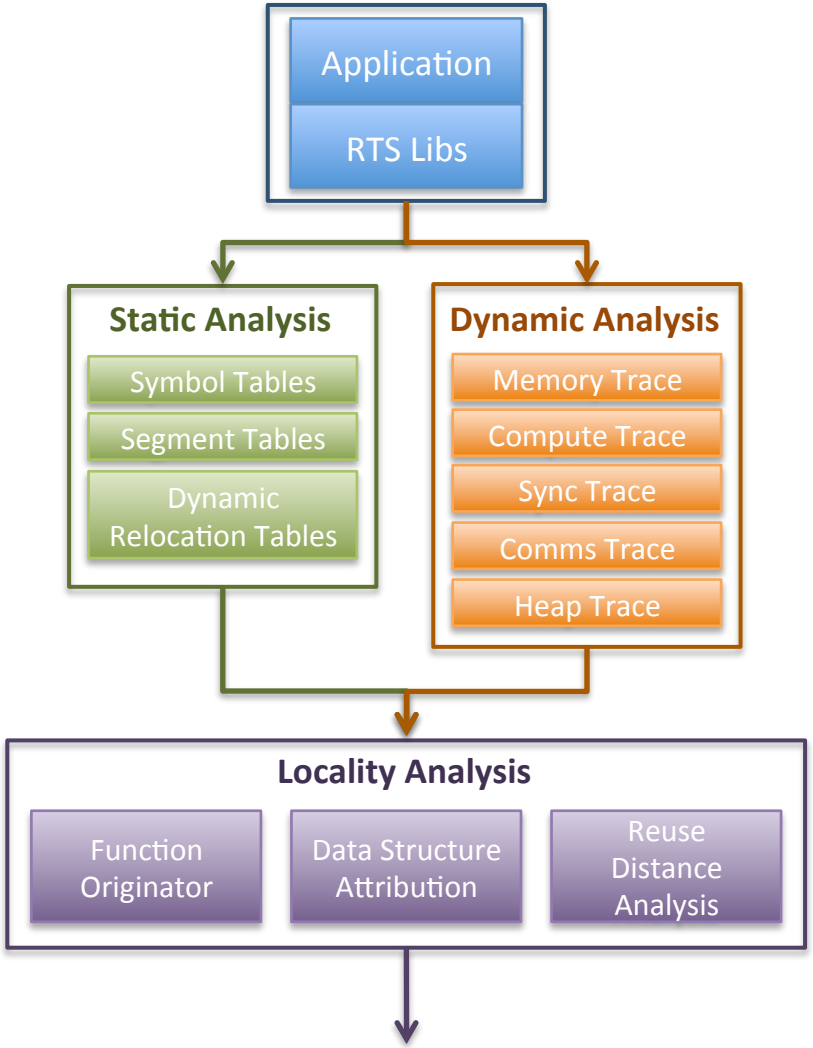
Application Analysis Framework

- **Assumption:** Algorithms and applications are written to target a specific Execution Model
- **Assumption:** Applications are expressed through programming models and runtime primitives
- Characterizing an application and its underlying runtime serve to showcase their inherent attributes
- **Challenge:** Capturing application behavior and attributing it to different components



Data Collection: Dynamic Instrumentation

- Dynamic Instrumentation Features:
  - Partial order of operations between different concurrent actors
  - Complete observability of events and operations regardless of origin
  - Frequency of operations
  - Usable to run “What-if” scenarios
- Shortcomings:
  - No performance or complete ordering
  - Overheads in both space and time
- Tools:
  - Intel PIN tool for x86 code
  - Collect operations associated with a particular attribute (e.g., load/store)
  - Reusable across Execution Models

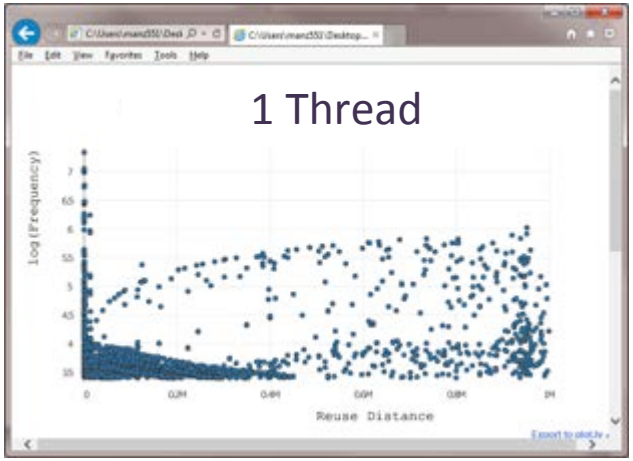


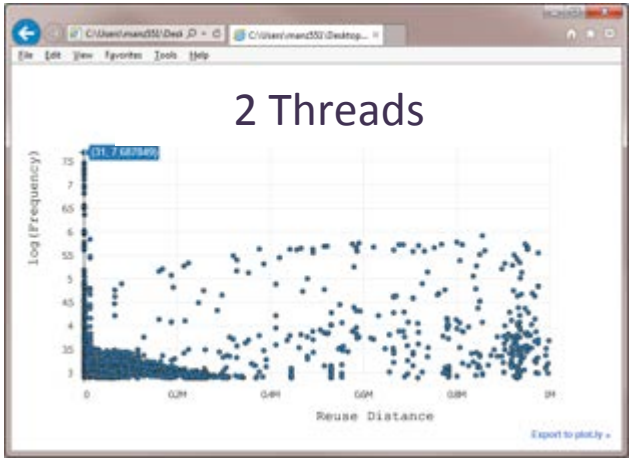
Case Study: Reuse Distance for HPCG

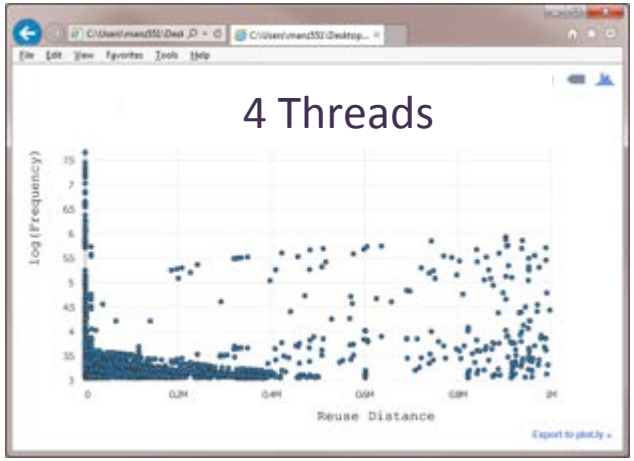
- Implementations using MPI and HPX runtimes
- Our metric for locality is *reuse distance*
  - Defined to be the number of unique addresses accessed between successive accesses to the same location
  - Metric is designed to be architecture independent
    - Does not consider cache sizes or number of cache levels
    - Does not consider accesses to the same page or cache line, but only to the same address
- Case study workload: HPCG
  - Pre-conditioned CG solver with 3D rectangular grid domain
  - Sparse matrix with 27 non-zero entities per matrix row
  - Majority of time spent in sparse matrix-vector multiply

HPCG Under MPI+OpenMP and ParalleX Execution Models

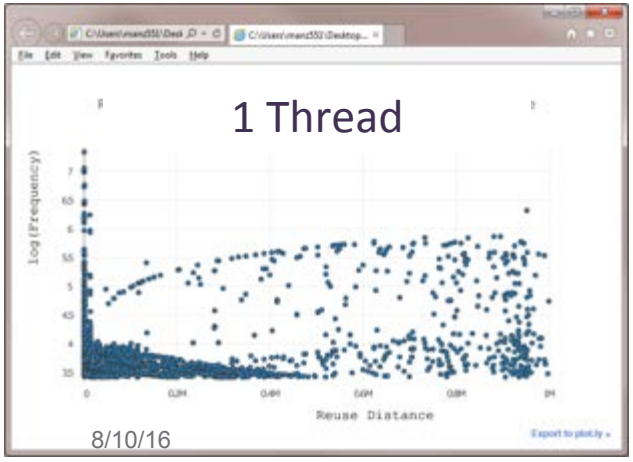
HPCG: HPX – Multiple Threads

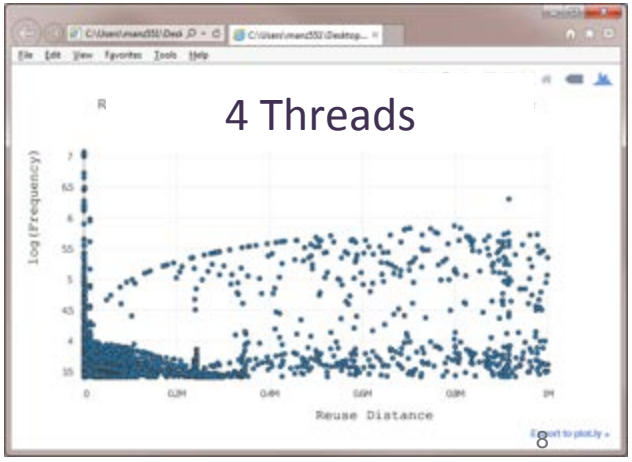




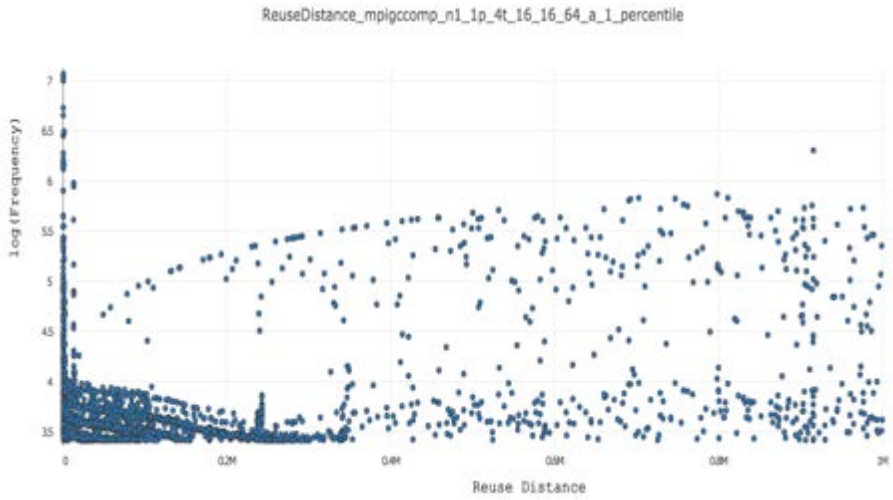


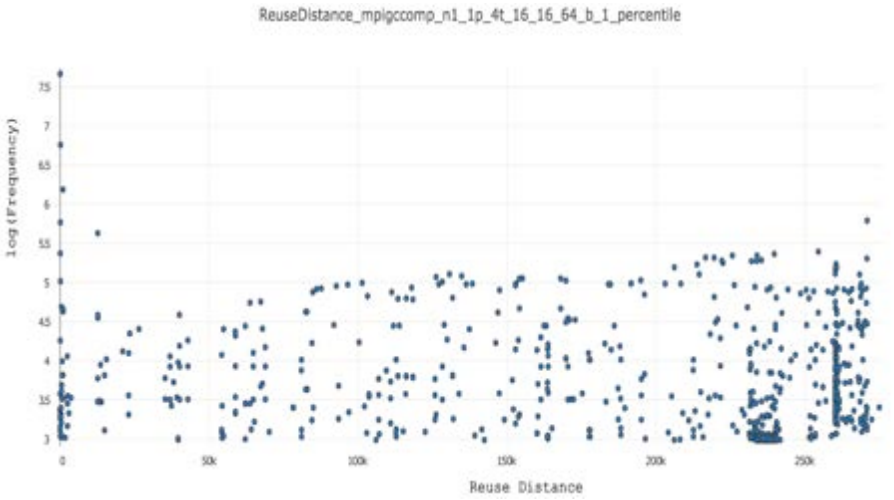
HPCG: MPI+OpenMP – Multiple Threads



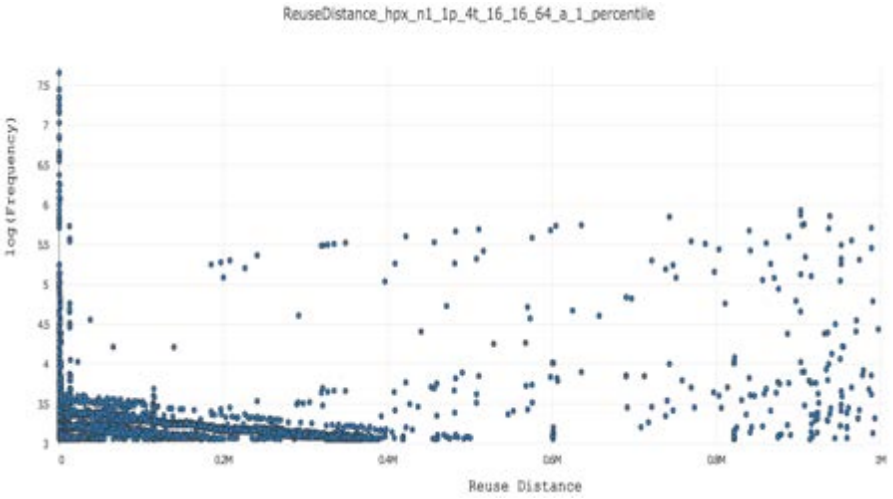


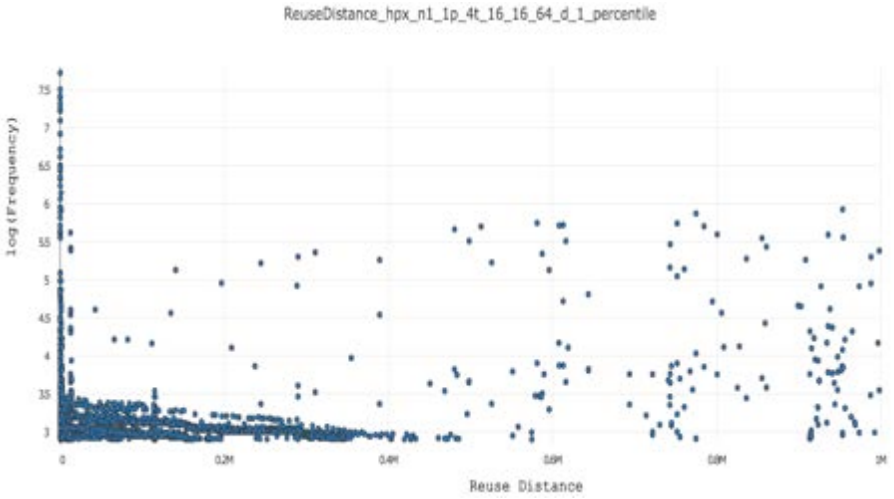
Reuse Distance for 4 OpenMP Threads





Reuse Distance for 4 HPX Worker Threads





Conclusions and Next Steps

- Improved reuse distance (and improved data locality) results from over-decomposition
- Some execution models easily allow for data over-decomposition, while others do not

- Future plans:
  - Data access attribution (e.g., runtime vs. application)