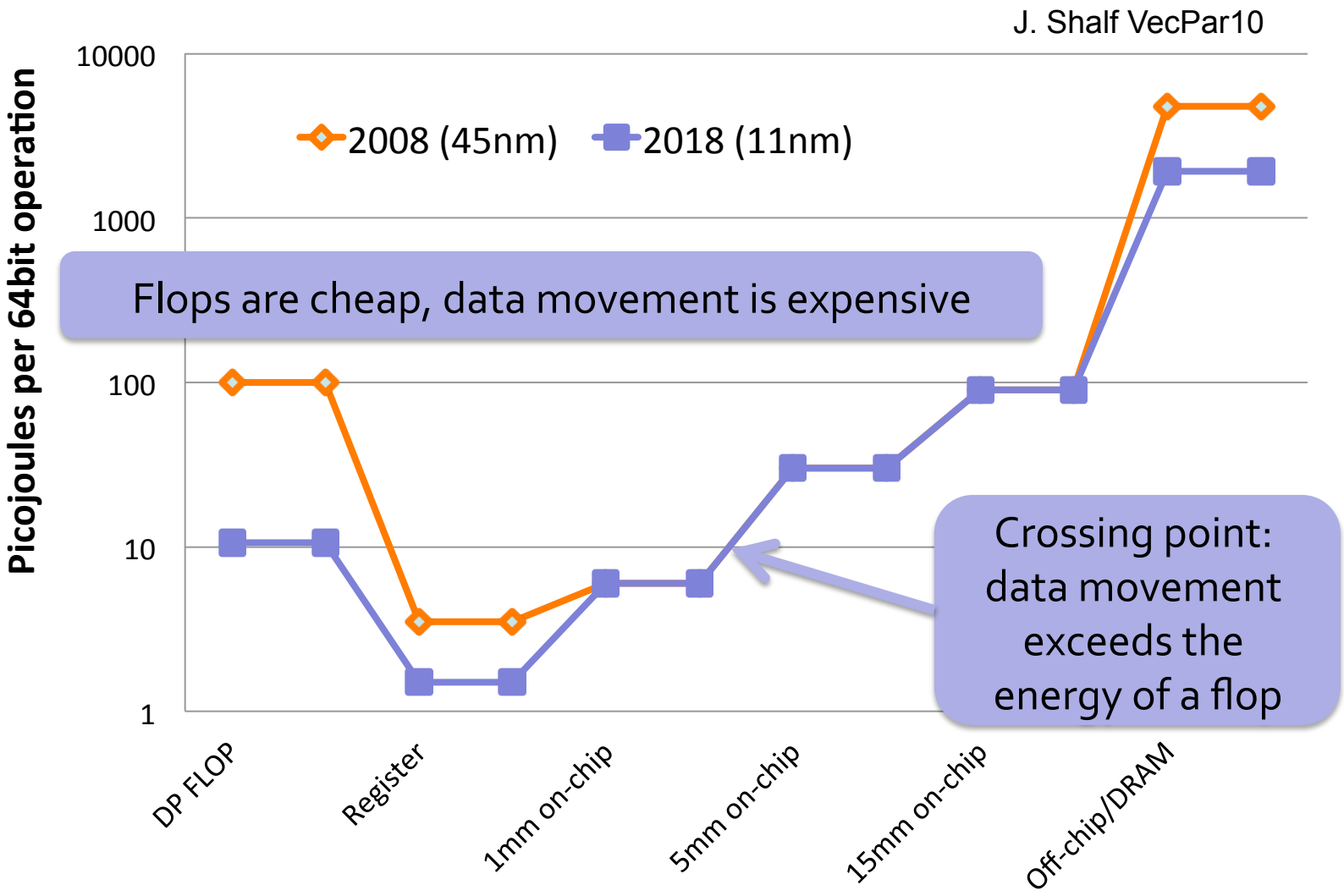
The Role of Modeling in Locality Optimizations

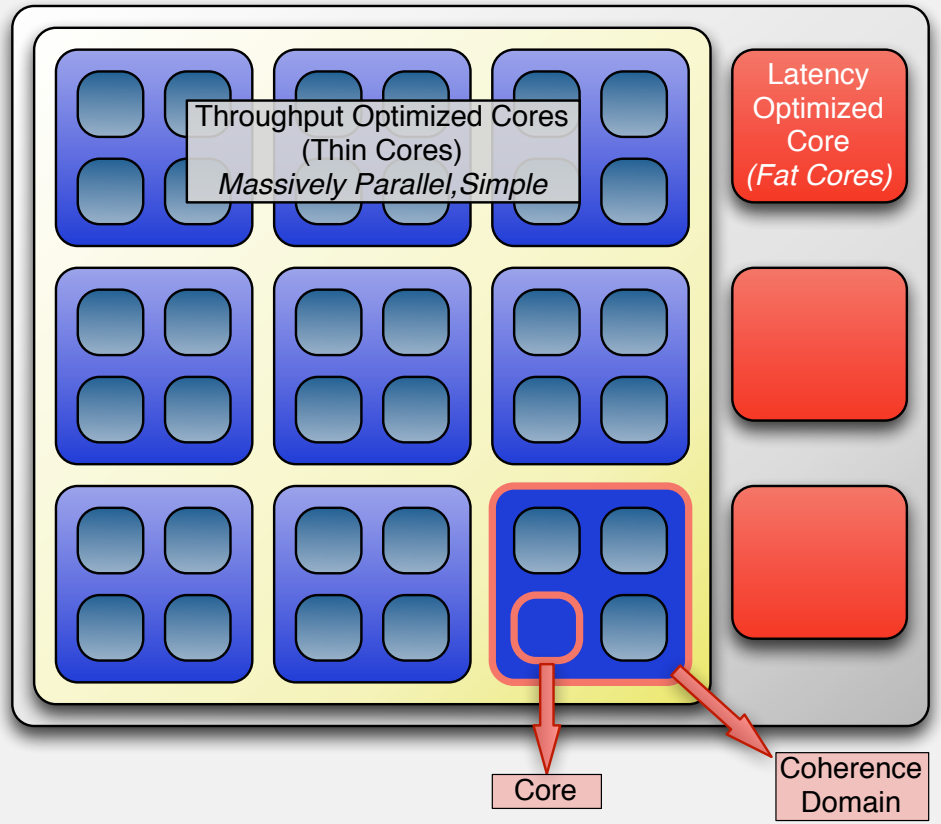
Didem Unat, **George Michelogiannakis**, John Shalf
Future Technologies Group
Lawrence Berkeley National Laboratory

MODSIM 2014

Trends in Computer Architecture



Abstract Machine Model (for emerging node architectures)



- The number of cores on a chip will be on the order of 1000s
- Maintaining cache coherence is NOT scalable
  - Expect coherence domains
- Flat and infinitely fast on-chip interconnect is NO longer practical
  - Expect complex NOCs
- Processing elements within a node are NOT equidistant.
  - Expect non-uniformity

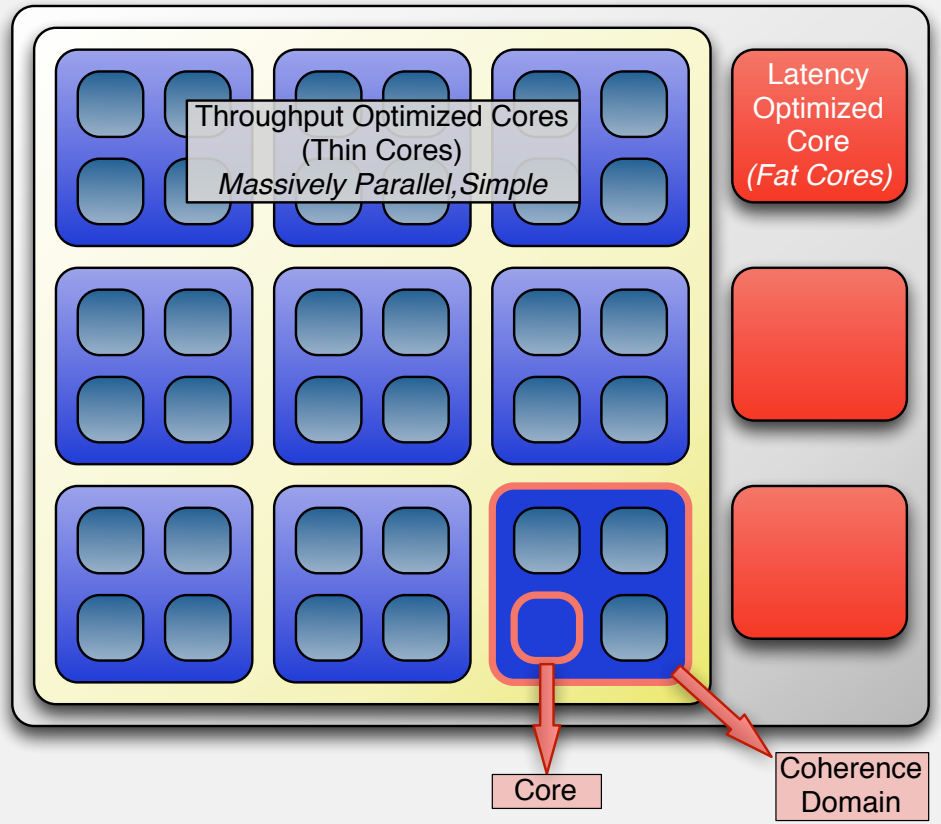
Download the CAL AMM doc from http://www.cal-design.org/

Non-Uniformity Trends

- Non-uniform communication
  - Larger-scale on-chip networks will have different distances and costs between different source-destination pairs
- Non-uniform memory access
  - NERSC's next system "Cori" will use Intel's next generation of Many Integrated Core [1]
  - Some cores are closer to memory controllers than others
- Therefore, data and task placement **matters**

[1] http://www.nersc.gov/users/computational-systems/nersc-8-system-cori/

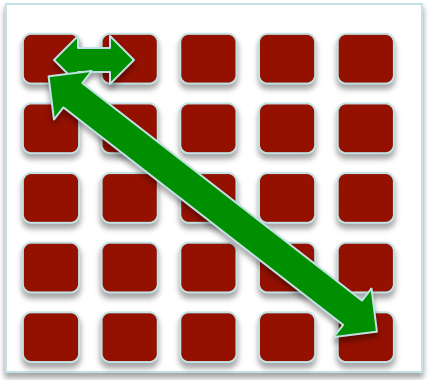
Data-Centric Programming Models



- Move away from compute-centric to data-centric programming model
- Minimize data movement by respecting the topology and hierarchy
- The new NERSC system, Cori, is already looking like this machine
  - Mesh Network w/quadrants
  - No coherence domains yet

Towards a Data Centric Computing Model

- **Old Model (OpenMP)**
  - Describe how to parallelize loop iterations
  - Parallel "DO" divides loop iterations evenly among processors
  - . . . but where is the data located?
  - Everything is equidistant



- **New Model (Data-Centric)**
  - Describe how data is laid out in memory
  - Changes to data layout applies to ALL Loop statements operate data where it is located (in-situ)
  - Similar to MapReduce, but need more sophisticated descriptions of data layout for scientific codes

TiDA: Tiling as a Durable Abstraction

Tiling is a well-known loop transformation for parallelism and data locality

- *Why not elevate it to the programming model?*

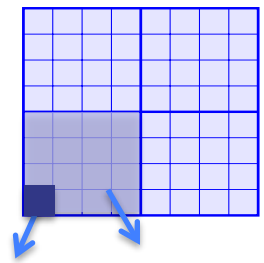
TiDA makes tiling part of data structure declaration by extending arrays with metadata

- Abstracts **layout, tile sizes and topology** to enable automated control of memory affinity
- Supports **parameterized** layout and tile size for easy tuning

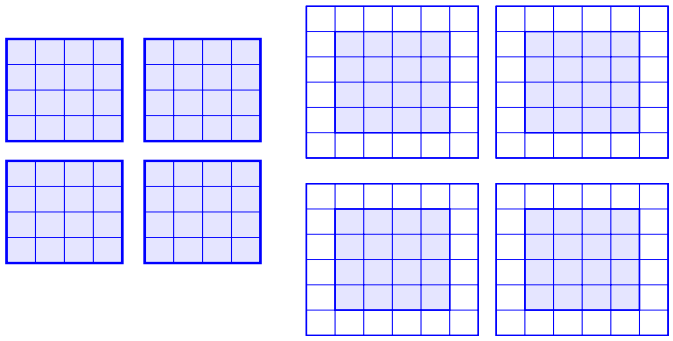
Benefits

- Support different layouts for various cache coherence scenarios
- *The solvers remain **unchanged** even when memory layout layout changes !!!*

a) Logical Tiles

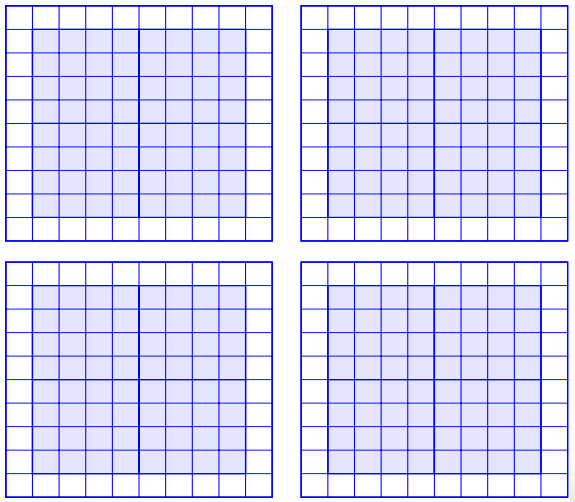


b) Separated Tiles



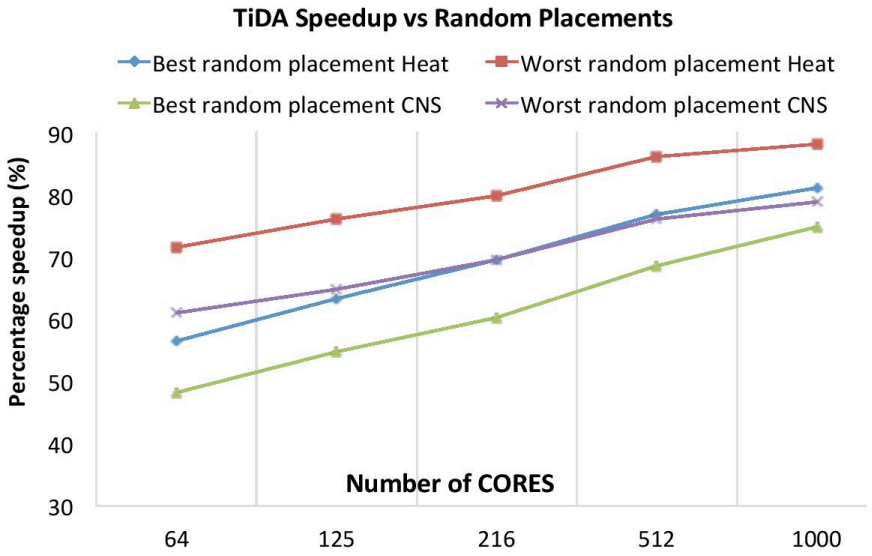
Separated tiles with halos

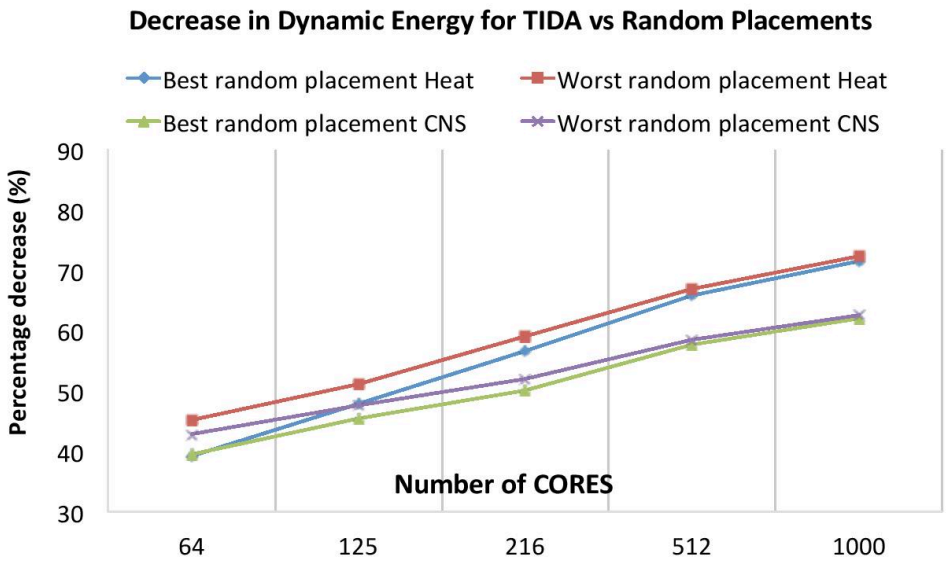
c) Regional Tiles



Modeling Tile Placement

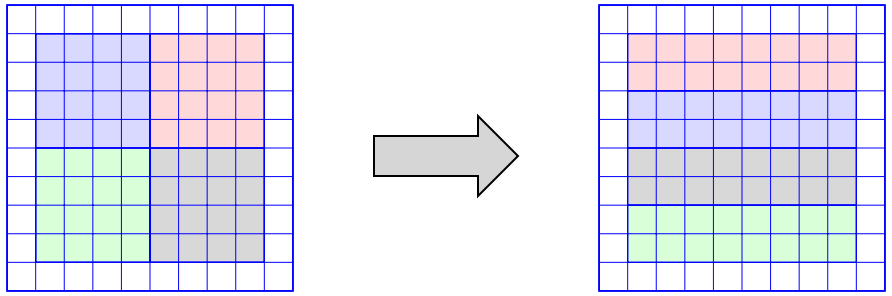
- An analytical or simulation model for the hardware guides the programmer or runtime to prune the exploration space
  - Such models need to account for non-uniformity and data movement
  - And many other machine energy and performance parameters





Modeling Tile Size and Layout

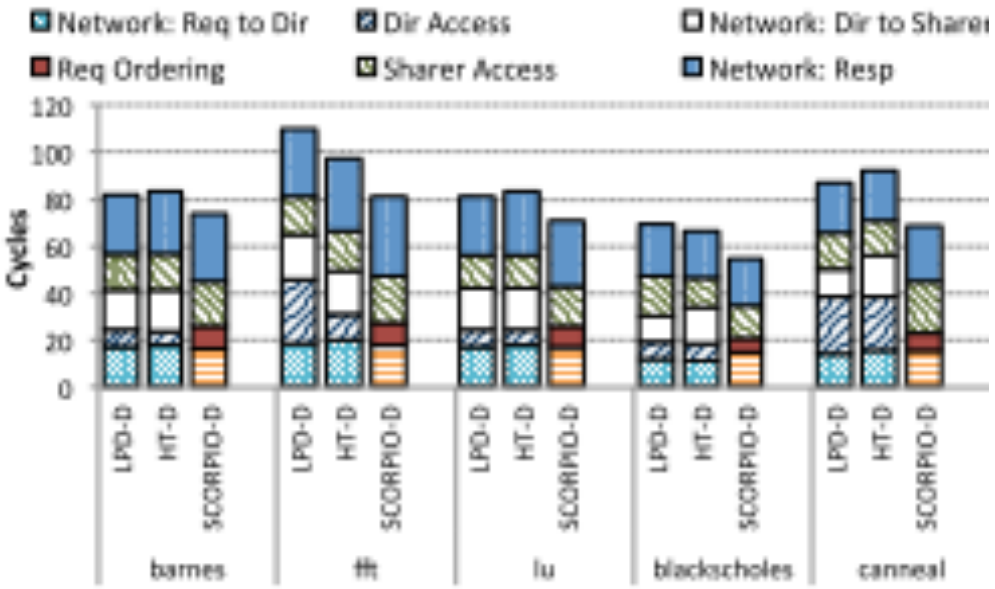
- Tile Size
  - Tile size is parameterized and global, can be set at runtime
  - Optimal tile size depends on the working set size and available cache
  - Modeling helps runtime decisions about tile size



- Memory layout
  - Layout is parameterized and global, can be set at the launch time
  - Different arrays can have different layouts
  - Different platforms have a different optimal layout
  - Assist programmer to estimate the best layout options

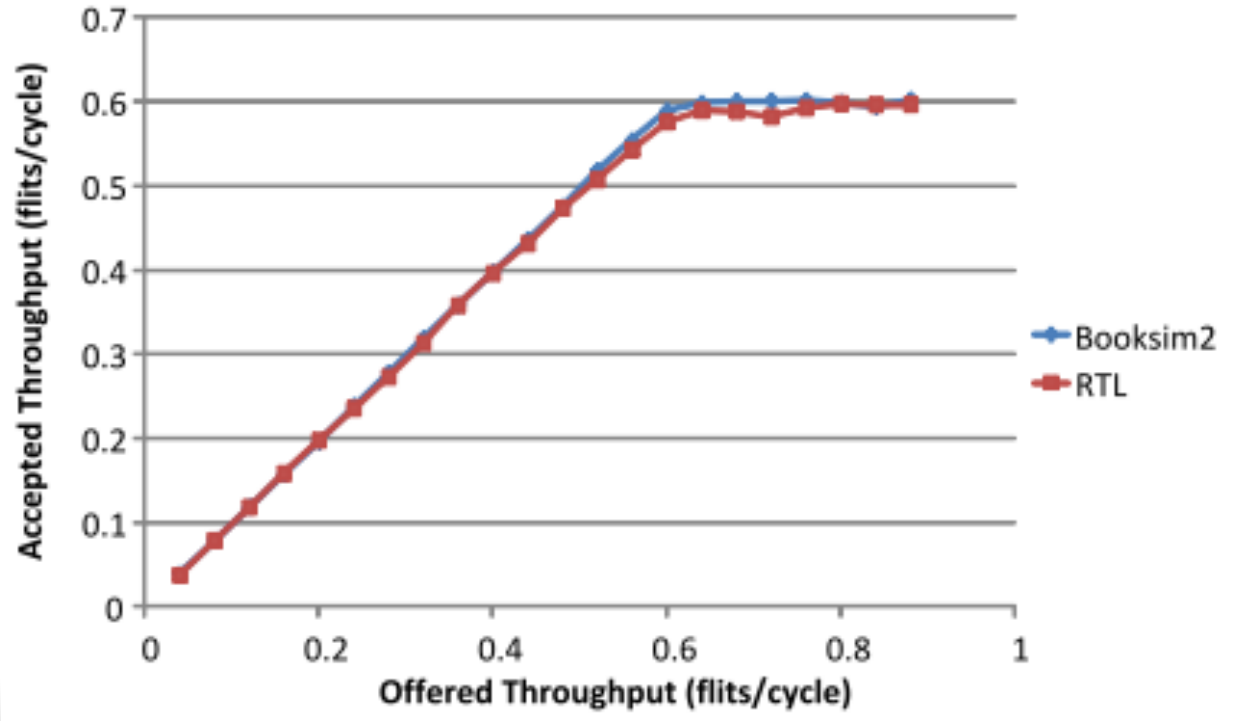
Generating a Model

- Simple analytical model suitable for run or compilation time
- Needs to capture all necessary information and costs for future chips
  - Data movement
  - NUMA effects
  - Core topology
  - Computation
  - Etc



Generating a Model

- Needs to be lightweight to be feasible at runtime or compile time
- Needs to be validated against real hardware or pre-validated software models
  - Such as OpenSoC NoC generator (next talk)



THANK YOU!

- Questions?