ParaLearn: A Massively Parallel, Scalable System for Learning Interaction Networks on FPGAs

Christopher W. Fletcher
Greg Gibeling
Dan Burke
John Wawrzynek

UC Berkeley

Narges B. Asadi
Eric Glass
Karen Sachs
Zoey Zhou
Wing Wong
Garry Nolan

Stanford

Outline

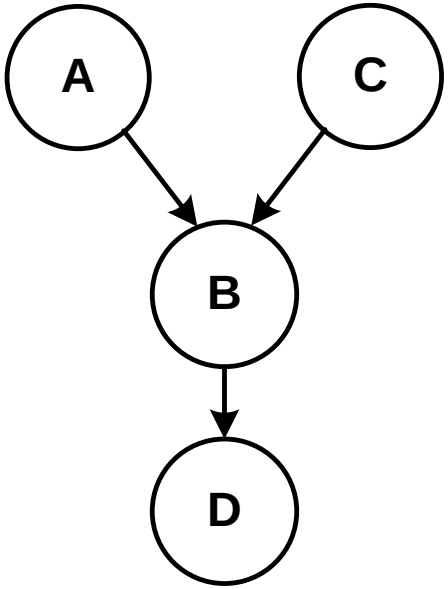
- What is ParaLearn?
  - Introduction: Terminology and objective
  - Motivation: Learning the structure of cell signaling networks
  - Algorithm and architectural overview
- Results
  - Design {scalability, flexibility}
  - End-to-End runtime
- Sources of speedup
- Closing

Introduction

ParaLearn is a specialized computer for conducting research on interaction networks

- We use software for:
  1. Control infrastructure
  2. Less computationally intensive steps
- We use hardware (FPGAs) for:

  Accelerating the algorithm kernel

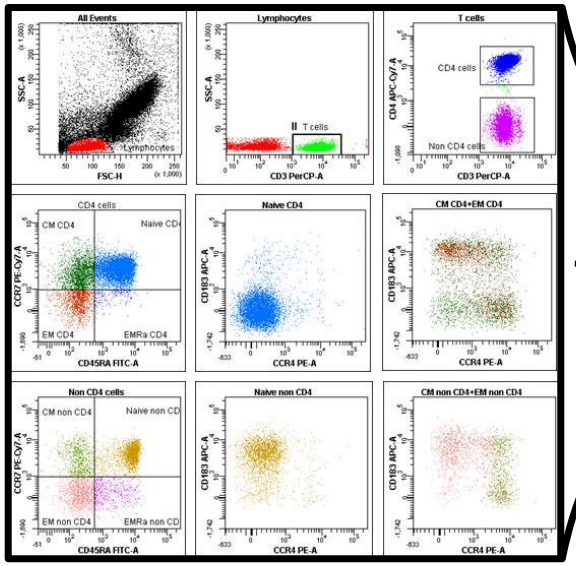


4 node interaction network:
A, B, C, D are **nodes**
A and C are **parents** of B
{A, C} is the **parent set** of B

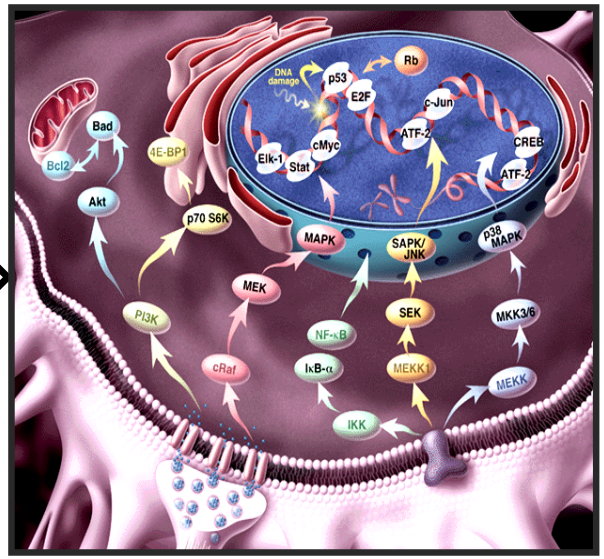
Cell Signaling Networks

Goal: Given flow cytometry 'CyTof' data, learn the structure of cell signaling networks

- Flow Cytometry
  - Data in the form of "raw" quantitative observations
  - Measurement of proteins & other components inside cells
- Cell Signaling Networks
  - Structures that model protein signaling pathways
  - Modeling perturbations to a network can help uncover the cause of human disease

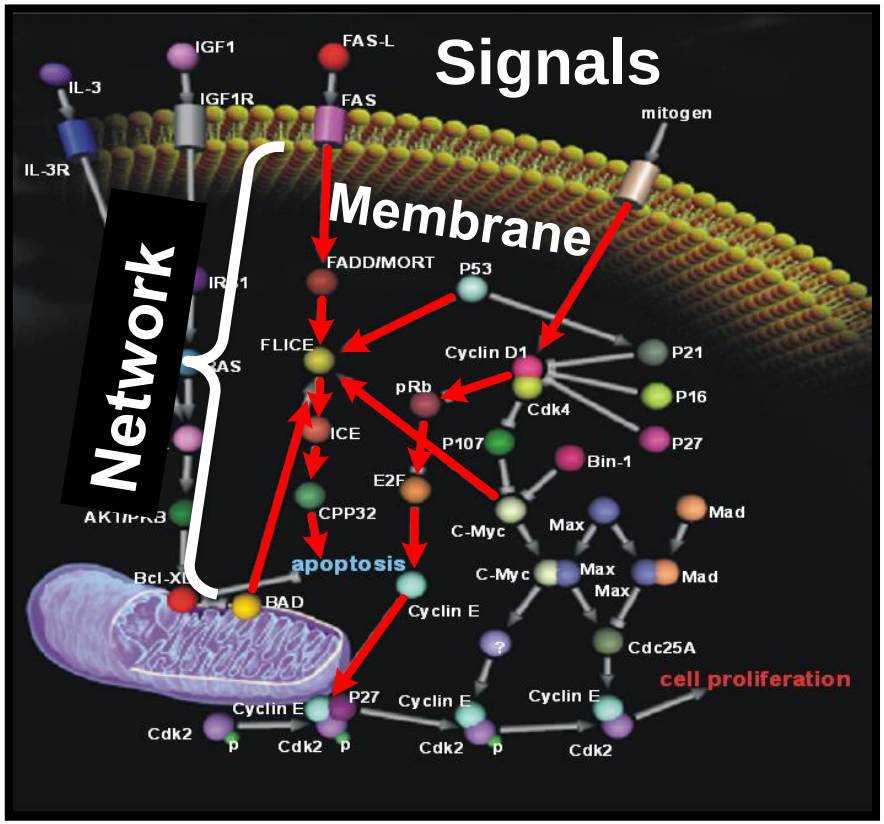


This talk



Signal Transduction Networks

'STN's: The cell's communication medium

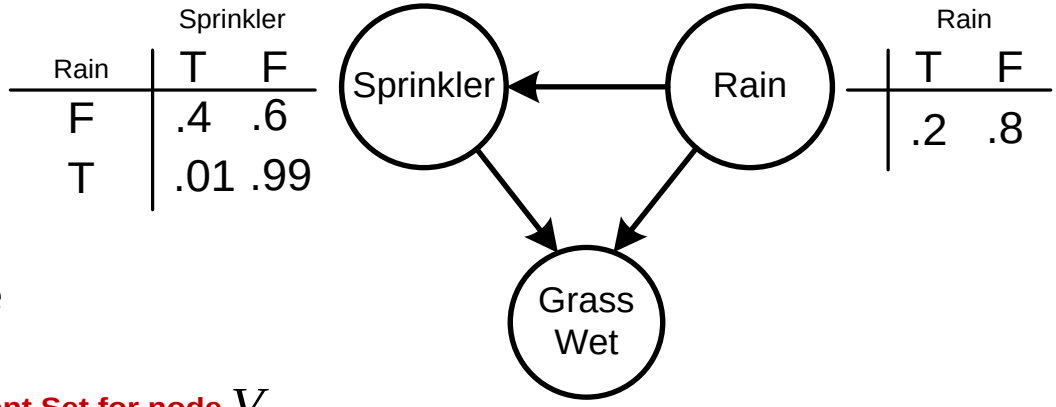


- Carry extra-cellular signals (heat, cold, pressure, etc) throughout the cell
- Span from membrane → nucleus
- Traditional model: Linear/independent protein chains
- Modern model:

  All proteins interact in a complex network

Bayesian Networks

- "Belief Network"
  - Directed acyclic graph
  - Structure encodes…
    - Conditional independence
    - Causal relationships



Rain	Sprinkler T	Sprinkler F
F	.4	.6
T	.01	.99

Rain T	Rain F
.2	.8

Parent Set for node V_i

$$P(V_1,\ldots,V_N)=\prod_{i=1}^{N}P(V_i\mid\Pi_i)$$

Sprinker	Rain	Grass Wet T	Grass Wet F
F	F	0	1
F	T	.8	.2
T	F	.9	.1
T	T	.99	.01

Courtesy of Tom Griffiths (U.C. Berkeley)

- Bayesian Score
  - A basis for comparing Bayesian Structures
  - Based on prior belief and observations

Experimental data

$$P(D,G)=P(G)P(D\mid G)$$

Graph

Prior probability

Macro Approach

Goal: Determine which network best explains the data

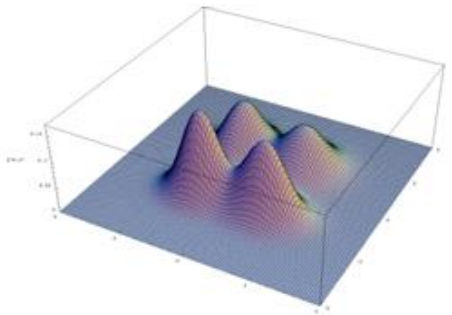
- End-to-end computation
  - (1) Pre-processor: Calculate local scores per parent set
  - (2) "Order Sampler": Determine the high **scoring** orders (**algorithm kernel**)
  - (3) "Graph Sampler": Extract graphs from high scoring orders
  - (4) Post-processing: high-level analysis and normalization

- Strategy
  - Implement steps (1) and (4) in software
  - Parallelize (and merge) steps (2) and (3) in hardware

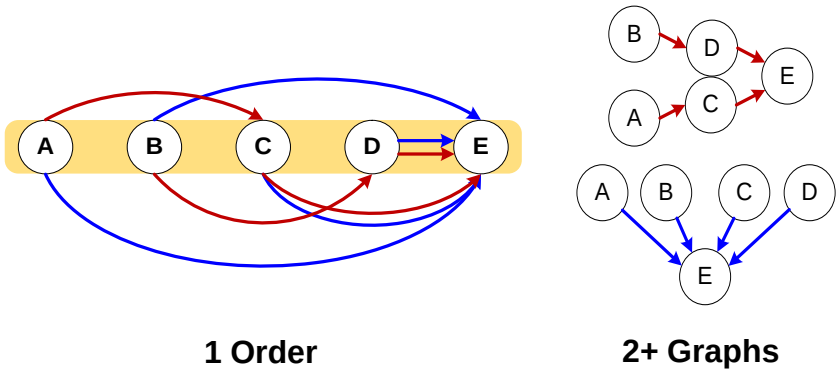
Kernel Considerations

- Learning graph structure is an NP-hard problem
  - Search space grows super-exponentially with the graph's node count
  - Multiple local optima, encoding best-solutions, may exist

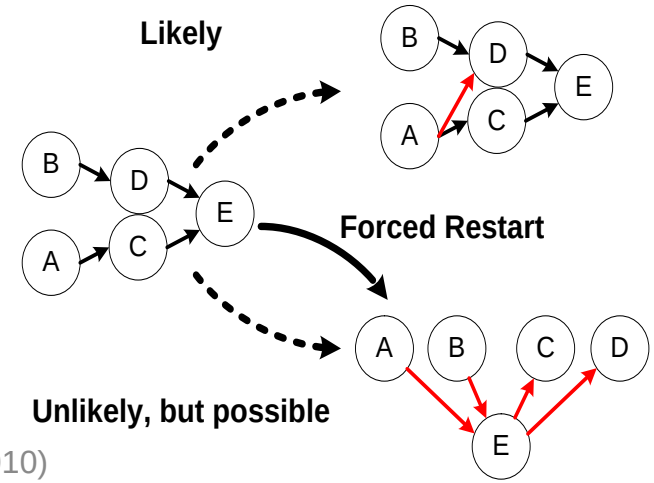
Nodes	Graphs
4	453
5	29281
10	4.7×10^{17}
20	2.34×10^{72}



- Order sampler: |order space| < |graph space|



- MCMC sampling, "**Restarts**": jump out of peaks



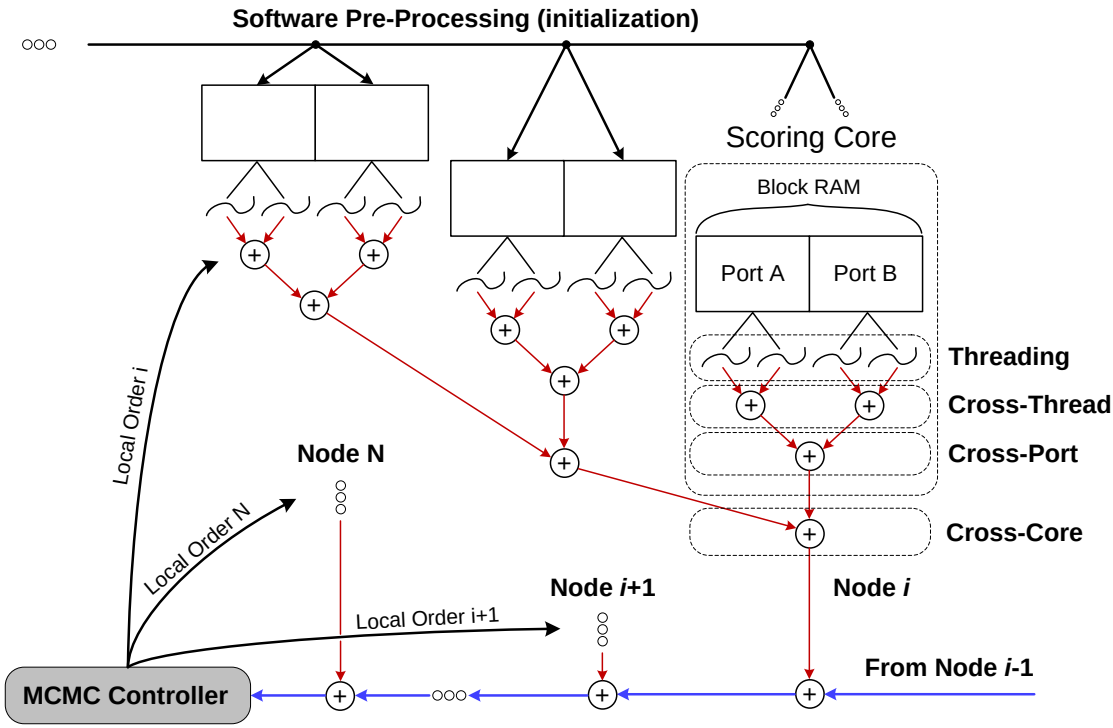
Micro Approach

Algorithm

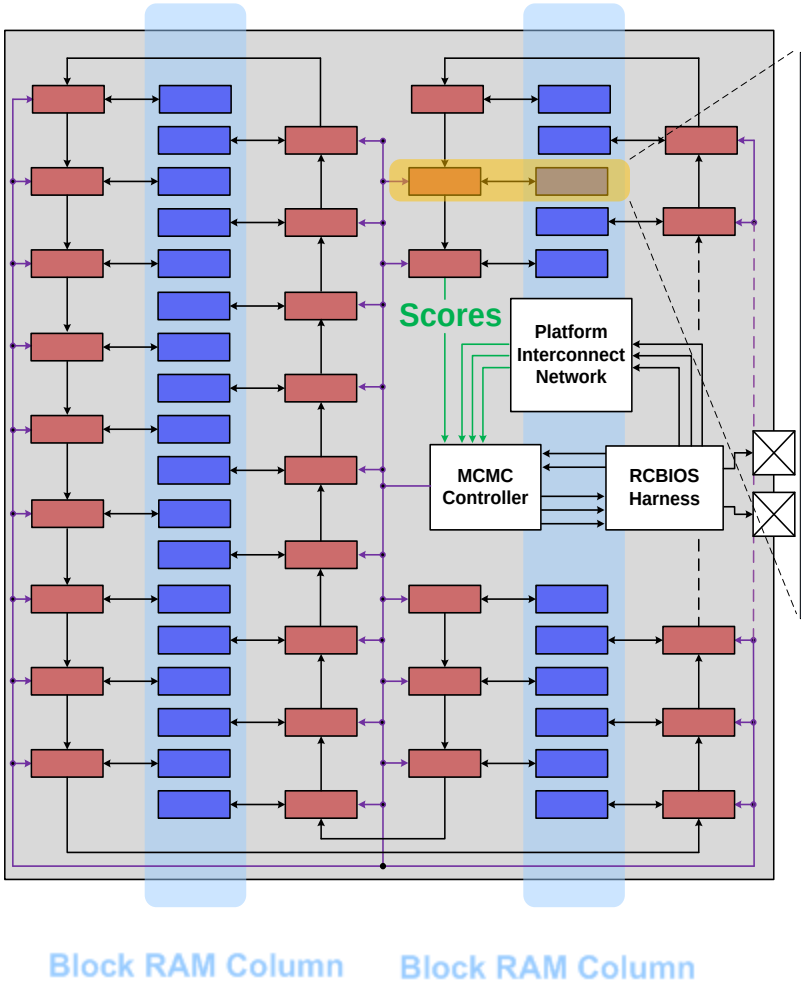
$$score(\prec | D) = \prod_{i=1}^{N} \sum_{\Pi_i \in \Pi_{\prec}} LocalScore(V_i, \Pi_i; D, G)$$

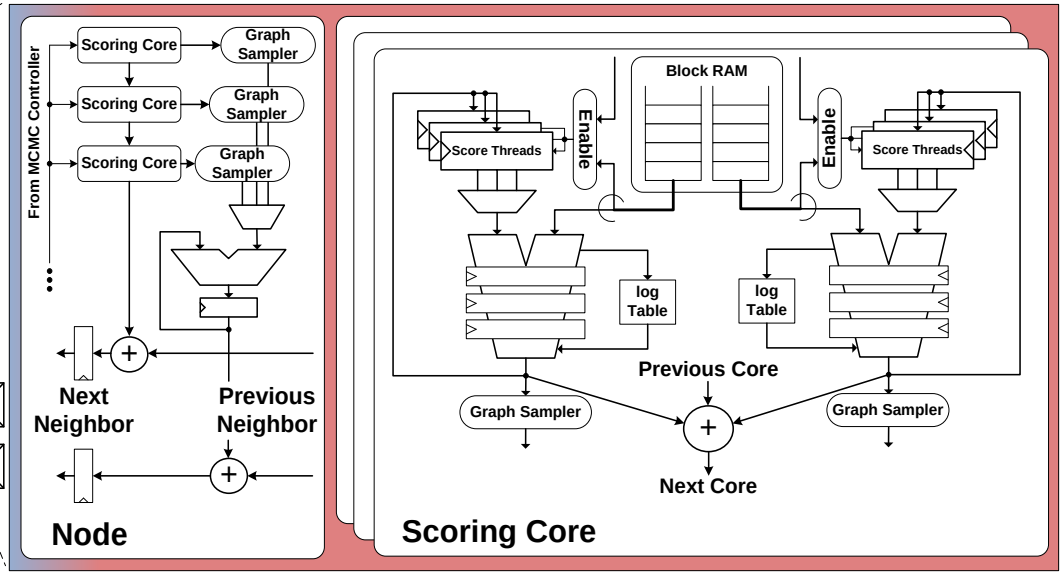
```
score(Order order):

orderScore = 0
nScore = Integer.MIN_VALUE
For each node (n):
  For each parent set (ps):
    If ps is "compatible" with order:
      nScore = log(e^ls[n][ps] + e^nScore)
  orderScore = orderScore + nScore
  nScore = Integer.MIN_VALUE
return {order, orderScore}
```

FPGA Implementation



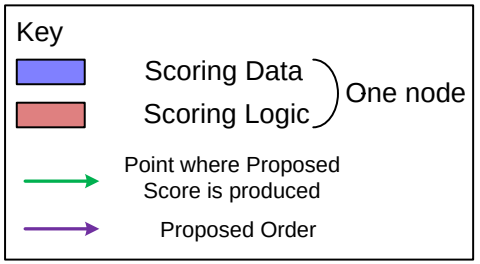
FPGA Core





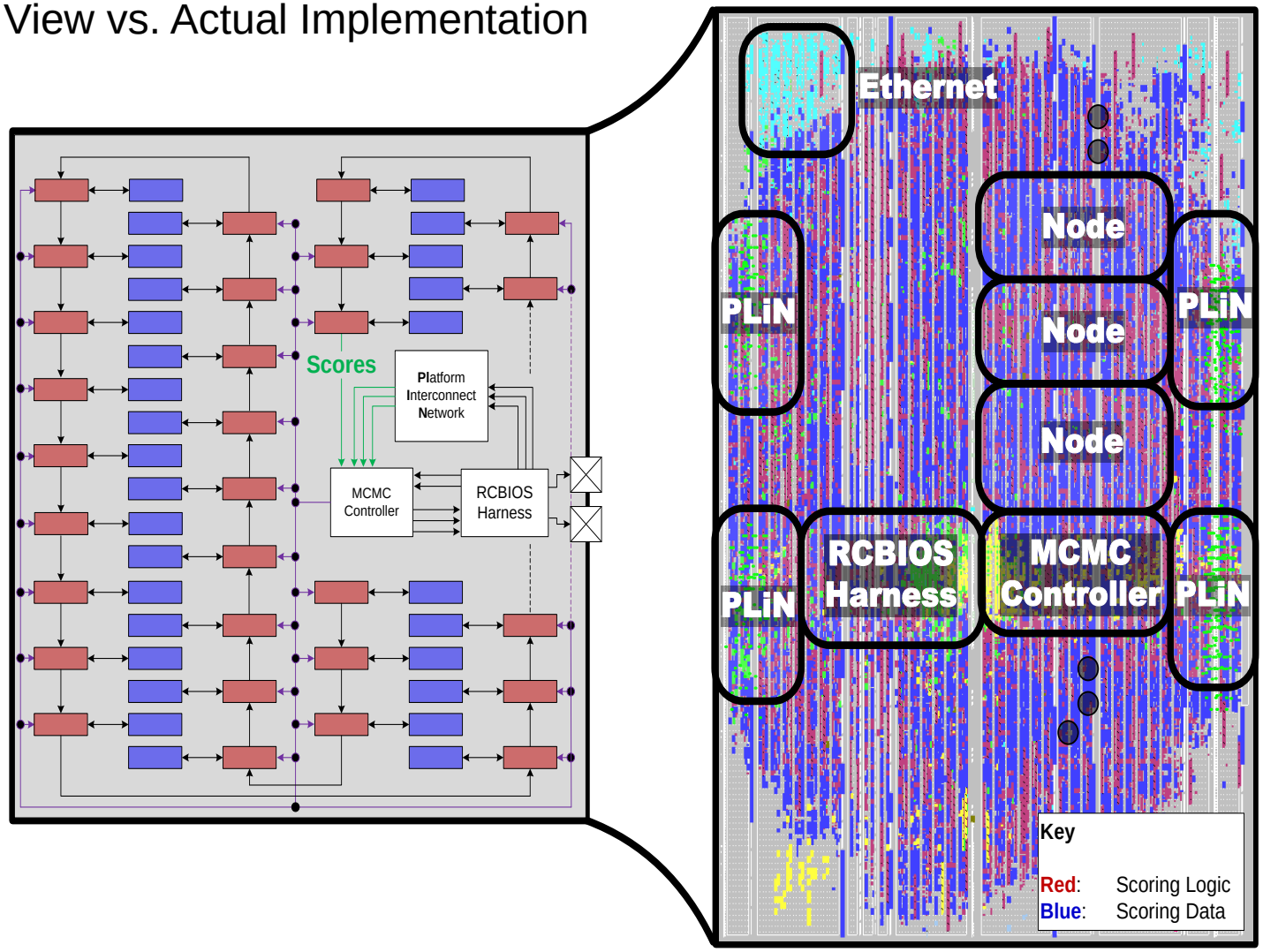
Xilinx Virtex-5 LX155T FPGA

29 node system
3 scoring cores per node



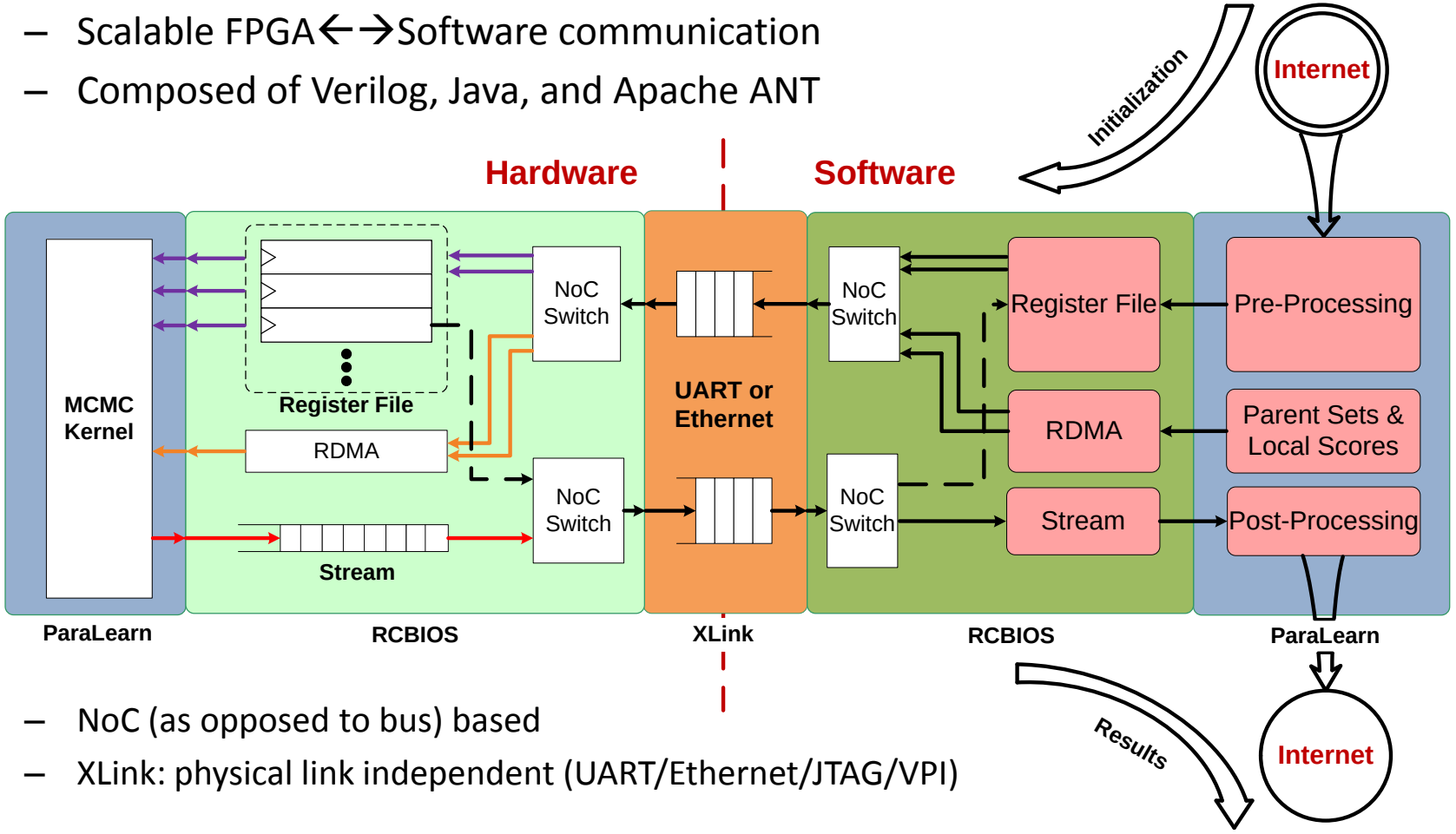
Floorplanner

Abstract View vs. Actual Implementation



System Infrastructure

- RCBIOS – Part of GateLib
  - Scalable FPGA←→Software communication
  - Composed of Verilog, Java, and Apache ANT



  - NoC (as opposed to bus) based
  - XLink: physical link independent (UART/Ethernet/JTAG/VPI)

FPGA Array

"MCMC Mesh"

Idea: Split larger problems across multiple FPGAs

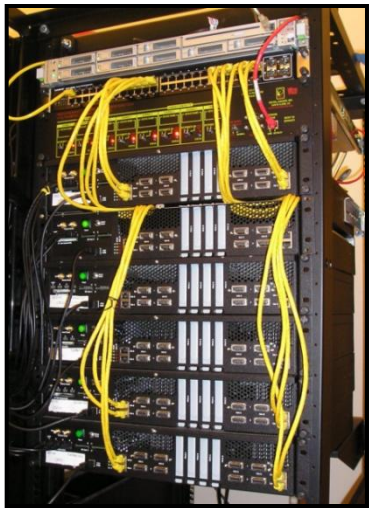
* While maintaining the base design

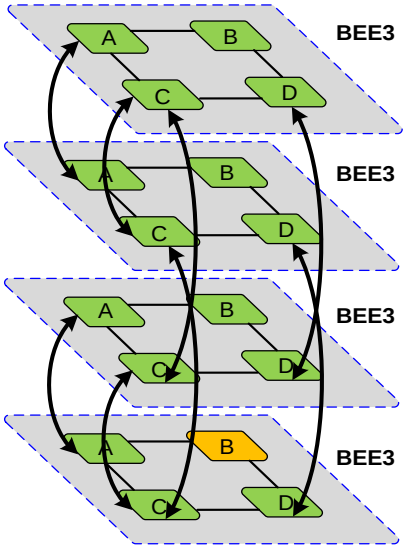
- Additional Infrastructure

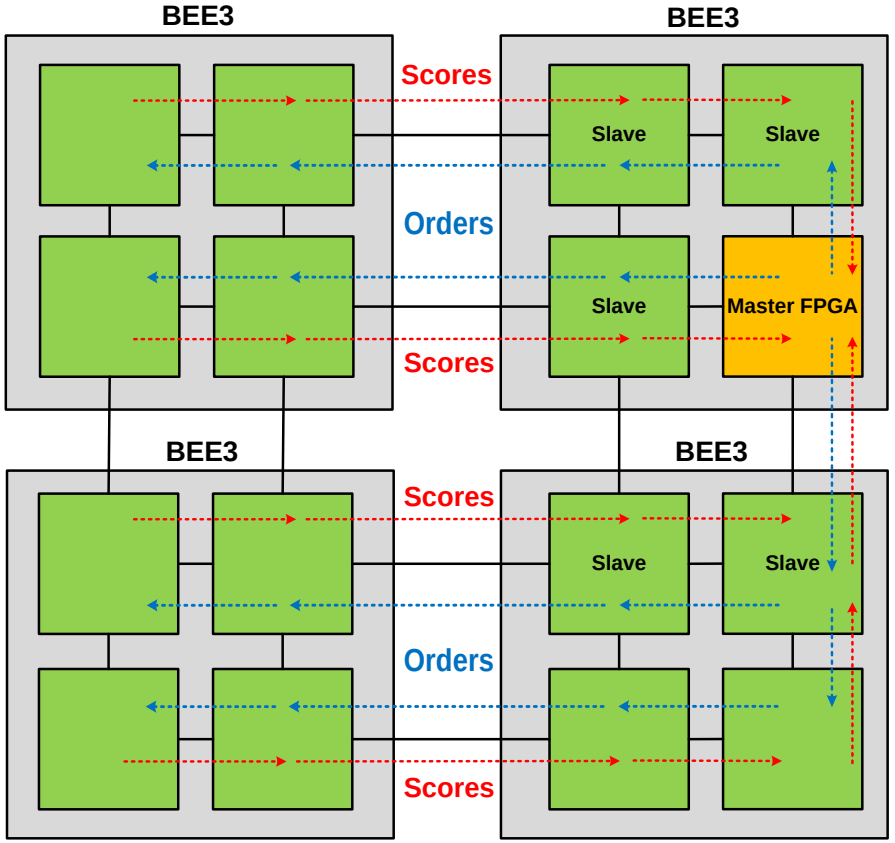
  (1) Inter-chip ring connections

  (2) Inter-board Aurora high-speed links

  (3) **Pl**atform **I**nterconnect **N**etwork (PLiN) built on (1) and (2)



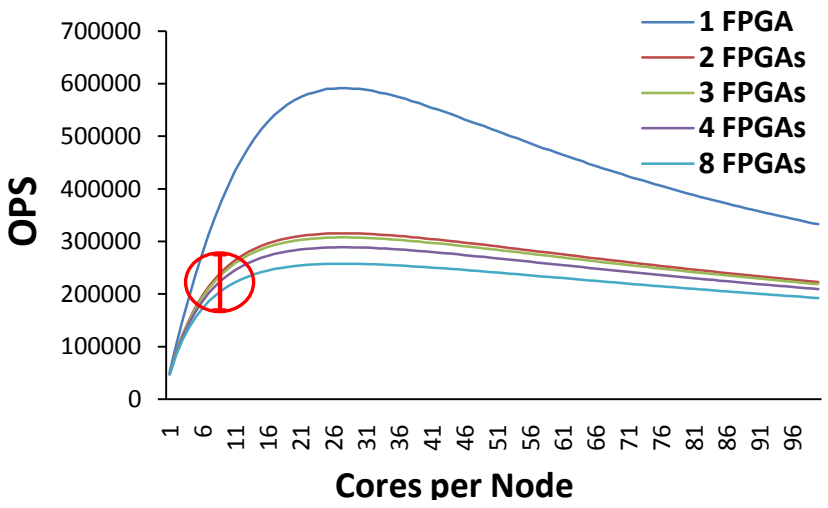


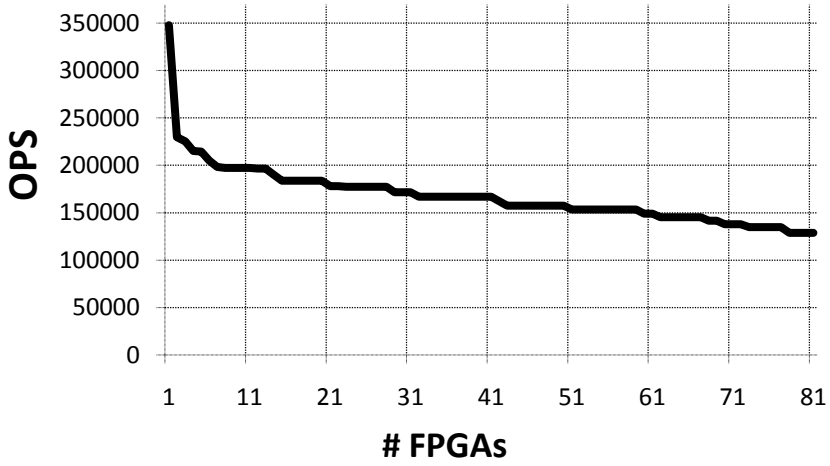


Scalability

Theme: Given a fixed network, vary hardware resources

Performance



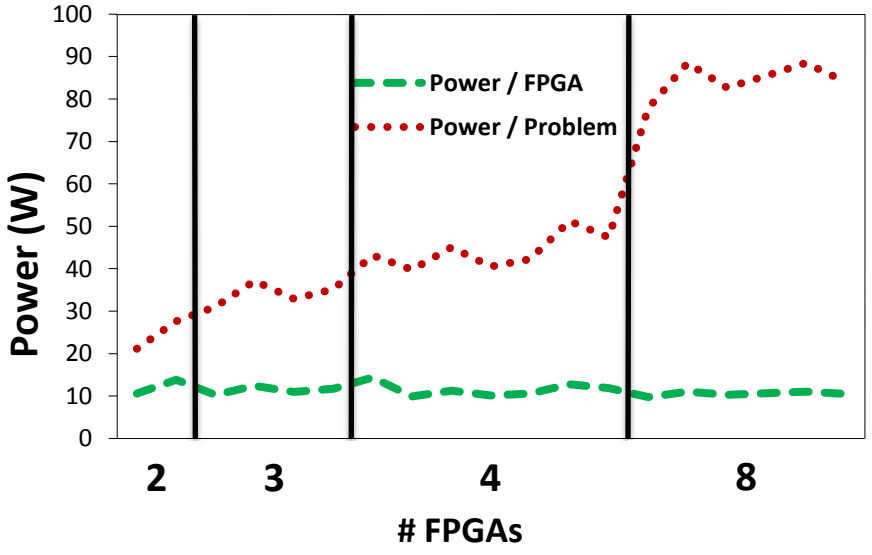


CyTof Network

- 22 nodes
- 4 indegree → 7547 parent sets per node

*OPS: “Orders per second”

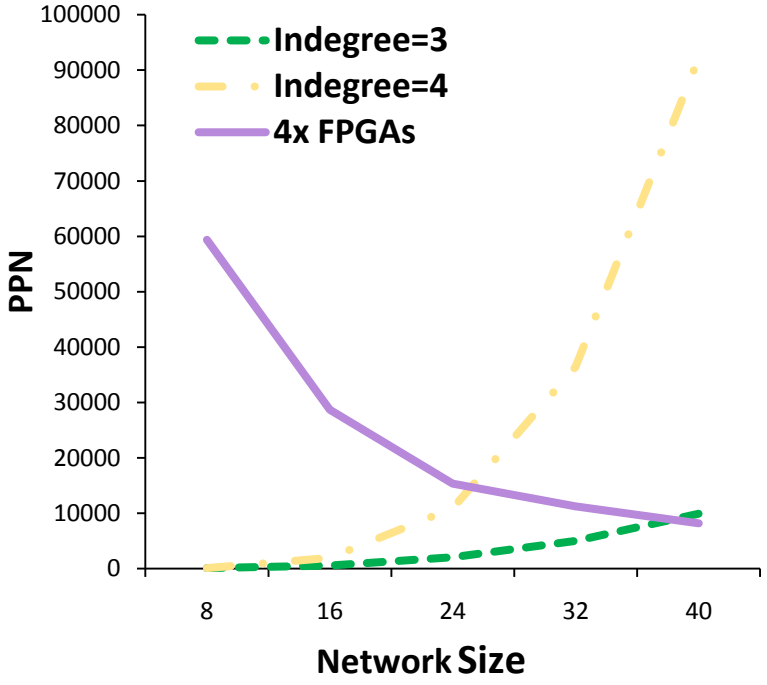
Power

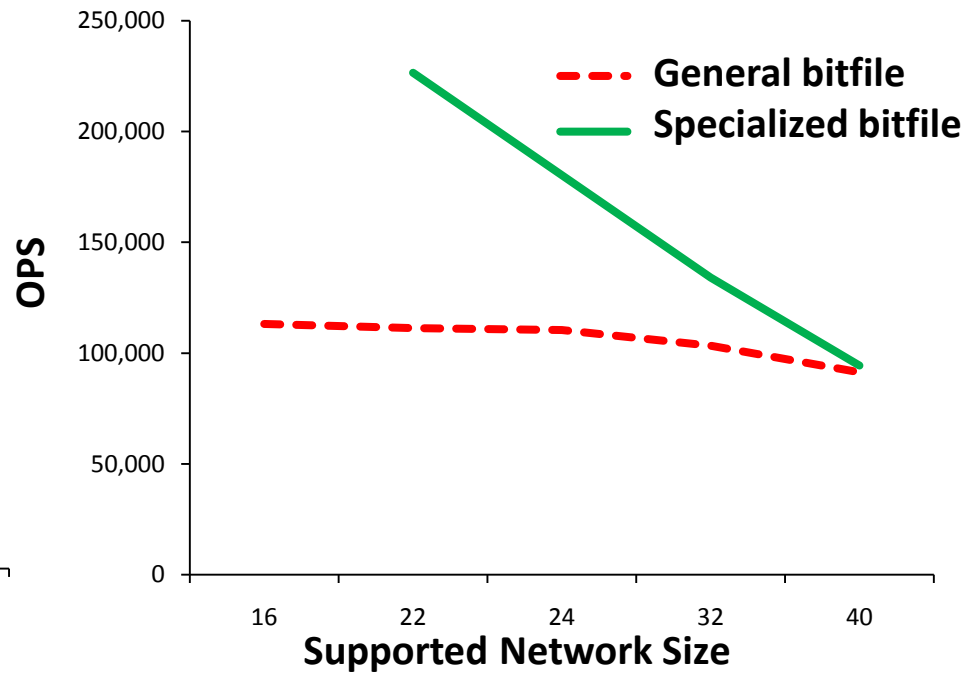


Flexibility

Theme: Given fixed hardware, vary the network

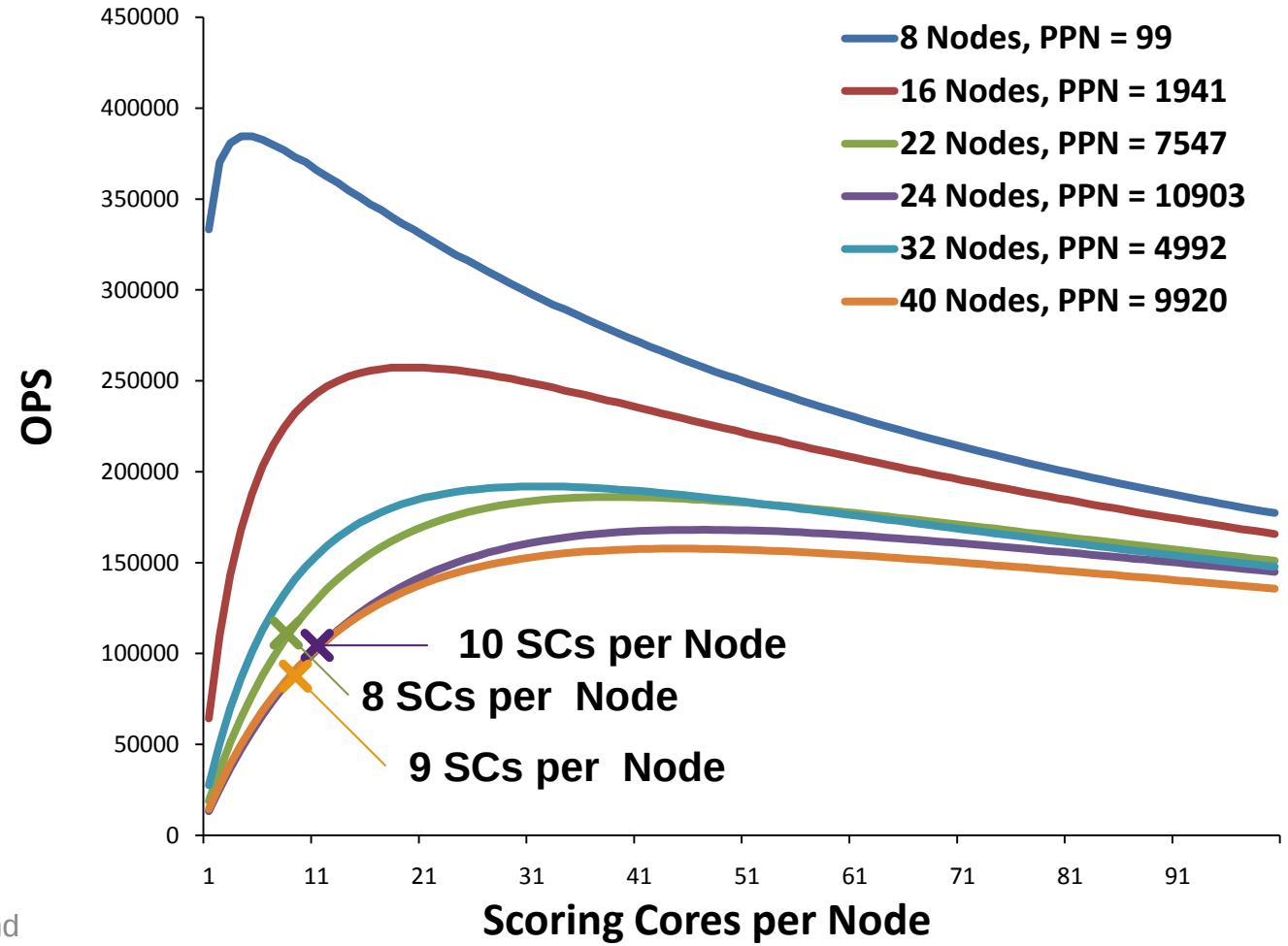
Problems	FPGA block RAM is limited	FPGA CAD tools are slow
Solutions	Limit the indegree of the explored graphs	Add hardware overhead to accommodate any network **up to** a given size





Scalability + Flexibility

Theme: Given different networks, vary hardware resources



End-to-End Runtime

- "Time to graphs" – this includes:
  1. Pre-processing
  2. FPGA load time
  3. FPGA run time (includes order and graph sampling)
  4. Post-processor (overhead is hidden)
- Over the 22 node CyTof data, varying the number of **restarts**

	0 Restarts		50 Restarts		100 Restarts	
Computation Step	GPP	FPGA	GPP	FPGA	GPP	FPGA
Pre-processing (Multinomial)	185	""				
Pre-processing (Linear Gaussian)	50	""				
Load time	**0**	4.5	""			
Order sampler	5.2	**0.44**	44.6	**22.1**	78.8	**44.15**
Graph sampler	0.85	**0**	18.8	**0**	31.3	**0**
Total	6.05	**4.94**	63.4	**26.6**	110.1	**48.65**

* Times in seconds

Sources of Speedup & Caching

- Considering FPGA and GPP implementations, which architecture benefits in what ways?
- We have considered…

  bitwise optimizations, threading/parallelism, fixed vs. floating point, and caching
- Caching

  Insight: order N → score M
  - Used by both GPU & GPP implementations
  - Can be made at an order or node granularity
- Caching analytic model
  - DRAM-based order and node caches, modeling: hash function, DRAM latency, hit rate, **all-or-nothing node cache hits**

Platform	OS Time (s)
Baseline GPP	48500
Optimized GPP	44.6
FPGA 4x	22.1

Setup	Time (s)
Baseline FPGA	22.1
Order cache	21.98
Node cache	8.63
Both caches	8.53
Speedup	2.59x

Performance Optimizations

- Pre-processing on FPGA
  - (1) “Pre-processing” has become new bottleneck
  - Map “Local score” generation to each FPGA in network
  - Transport “observations” data to FPGAs

  Insight: Observation files are small, score files are large
- Additional parallelism
  - FPGA load time can overlap with the pre-processing step (hides load time)
  - The MCMC controller can be ‘threaded’ with multiple restarts (hides result accumulation overhead)

Closing

This work is an example of how a hand optimized, low- level FPGA design can lead to significant performance and power benefits over conventional processors and GPUs

Algorithm performance is having immediate impact on work done by Biologists who are studying STNs

Our approach's cost: large development and debug time

Our group is currently working on high-level programming abstractions that will ease development effort, ideally without losing efficiency

Acknowledgements

For making this work possible, the authors thank:

- NIH / Cancer Research Institute Grant #130826-02
- M. Linderman et al.
- Stanford Nolan Lab
- Members of the Berkeley Reconfigurable Computing Group
- UC Berkeley Wireless Research Center (BWRC)
- Contributors to the GateLib research library