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

Narges Bani Asadi[†]
Electrical Engineering
Department
Stanford University
Stanford, CA, USA 94305
nargesb@stanford.edu

Christopher W. Fletcher[†]
Electrical Engineering and
Computer Sciences
Department
University of California
Berkeley, CA, USA 94720
cwfletcher@berkeley.edu

Greg Gibeling
Electrical Engineering and
Computer Sciences
Department
University of California
Berkeley, CA, USA 94720
gdgib@berkeley.edu

John Wawrzynek
Electrical Engineering and
Computer Sciences
Department
University of California
Berkeley, CA, USA 94720
johnw@eecs.berkeley.edu

Wing H. Wong
Statistics and Health
Research and Policy
Departments
Stanford University
Stanford, CA, USA 94305
whwong@stanford.edu

Garry P. Nolan
Microbiology and Immunology
Department
Stanford University
Stanford, CA, USA 94305
gnolan@stanford.edu

[†] Lead authors.

ABSTRACT

ParaLearn is a scalable, parallel FPGA-based system for learning interaction networks using Bayesian statistics. ParaLearn includes problem specific parallel/scalable algorithms, system software and hardware architecture to address this complex problem.

Learning interaction networks from data uncovers causal relationships and allows scientists to predict and explain a system's behavior. Interaction networks have applications in many fields, though we will discuss them particularly in the field of personalized medicine where state of the art high-throughput experiments generate extensive data on gene expression, DNA sequencing and protein abundance. In this paper we demonstrate how ParaLearn models Signaling Networks in human T-Cells.

We show greater than 2000 fold speedup on a Field Programmable Gate Array when compared to a baseline conventional implementation on a General Purpose Processor (GPP), a 2.38 fold speedup compared to a heavily optimized parallel GPP implementation, and between 2.74 and 6.15 fold power savings over the optimized GPP. Through using current generation FPGA technology and caching optimizations, we further project speedups of up to 8.15 fold, relative to the optimized GPP. Compared to software approaches, ParaLearn is faster, more power efficient, and can support novel learning algorithms that substantially improve the precision and robustness of the results.

ICS'10, June 2–4, 2010, Tsukuba, Ibaraki, Japan.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems

General Terms

Design, Performance, Algorithms

Keywords

FPGA, Bayesian Networks, Markov Chain Monte Carlo, Reconfigurable Computing, Signal Transduction Networks

1. INTRODUCTION

ParaLearn is a scalable, parallel FPGA-based system for learning interaction networks using Bayesian statistics. Interaction networks and graphical models have various applications in bioinformatics, finance, signal processing, and computer vision. They have found use particularly in systems biology and personalized medicine in recent years, as improvements in biological high-throughput experiments provide scientists with massive amounts of data.

ParaLearn algorithms are based on a Bayesian network (BN) [19] statistical framework. BNs' probabilistic nature allows them to model uncertainty in real life systems as well as the noise that is inherent in many sources of data. Unlike other graphical models such as Markov Random Fields and undirected graphs, BNs are easily capable of learning sparse and causal structures that are interpretable by scientists [9, 14, 19].

Learning BNs from experimental data helps scientists predict and explain systems' outputs and learn causal relations that lead to new discoveries. Discovering unknown BNs from data, however, has been a computationally challenging problem and despite the significant recent improvements in algorithms is still computationally infeasible, except for networks with few variables [8].

ParaLearn accelerates BN discovery through exploiting parallelism in BN learning algorithms. Furthermore, ParaLearn uses Markov Chain Monte Carlo (MCMC) optimization methods to search for the BNs that can best explain experimental data. As it was designed for high-performance, ParaLearn is able to use novel algorithms to address robustness and precision issues, which have traditionally been sacrificed to decrease runtime on general purpose processors (GPPs).

BN modeling has motivated previous studies targeting single chip FPGA implementations [5, 20]. Unlike these approaches, ParaLearn's focus is to explore different avenues for system scalability. In order to scale to larger problems and networks, ParaLearn supports a general mesh of interconnected FPGAs. To increase robustness and find the true underlying model behind the data, ParaLearn coordinates an ensemble of scoring methods in what we call the *MetaSearch* approach.

We implemented ParaLearn on multi-core GPPs, GPUs [17] and FPGAs. The FPGA implementation achieves the highest performance and power savings through best exploiting fine-grained parallelism inherent in the algorithm, but required the most development effort. Moreover, FPGA multi-chip systems are interconnected through a topology customized to ParaLearn algorithms. This allows ParaLearn to scale, while maintaining performance, using additional FPGAs.

We demonstrate ParaLearn's performance, power and scalability in modeling the Signal Transduction Networks (STN) in human T-Cells. STNs include hundreds of interacting proteins and small molecules in the cell and regulate numerous cellular functions in response to changes in the cell's chemical or physical environment. STNs are important subjects of study in systems biology and alterations in their structures are profoundly linked to increased risks of human diseases like cancer [15].

Single cell high-throughput measurement techniques such as flow cytometry have made it possible to measure and monitor the activity of small molecules and proteins involved in STNs. BNs have been successfully applied to reconstruct possible STN pathways from this data [21]. BN inference helps determine the causal structure of STNs as well as linking different structures to clinical outcomes. This can lead to new diagnostics, drug target and biomarker discoveries and novel prognosis tools [15]. While the previous computational studies on reconstructing STNs from data involved few variables (about 10 proteins), a new high-throughput measurement technology, “CyTof” [4], has been developed based on mass spectrometry that enables scientists to measure up to a hundred molecules simultaneously. CyTof creates the opportunity and the need for more sophisticated analysis of STNs in cancer cells. We use the data that has been produced using CyTof technology [4] in our analysis.

2. ALGORITHMS & MAPPING TO FPGAS

BNs [19] are a class of probabilistic graphical models that are useful in modeling and learning complex stochastic relationships between interacting variables. A BN is a directed acyclic graph $\mathcal{G}$, the nodes of which represent multivariate random variables $V = \{V_1, \ldots, V_n\}$. The structure of the graph encodes conditional and marginal independence of the variables and their causal relations by the local Markov property: that every node is independent of its non-descendants given its parents. The dependence of nodes on their parents is encoded in local Conditional Probability Distributions (CPDs) at each node [14, 19]. The joint probability distribution is decomposed to the product of the probability of each variable V_i conditioned on its parents Π_i:

$$P(V_1, \ldots, V_n) = \prod_{i=1}^{n} P(V_i|\Pi_i)$$

The CPDs that model $P(V_i|\Pi_i)$ can have different forms based on the data characteristics. The Multinomial distribution, Linear Gaussian or Mixture of Gaussians are a few examples of possible CPD formulations. Figure 1.a depicts an example of a BN with binomial CPDs.

The algorithms that have been developed for learning BNs usually assign a score to candidate graph structures (based on how well each graph structure explains the data) and then search for the best scoring graph structures. The score of a graph structure $\mathcal{G}$ given the data $\mathcal{D}$ can be computed in many different ways. The most popular scoring metrics are based on Maximum Likelihood (ML), Bayesian Information Criterion (BIC) or Bayesian methods [14, 16]. The graph scores also depend on the CPD formulation that is assumed for the BN. The important observation here is that the graph score can always be decomposed into a local score at each node. As we explain later, this enables a general purpose parallel framework that can accommodate all the different BN learning algorithms.

While learning CPD parameters for a given graph structure is a relatively easy task, the computational inference of the graph structures is an NP-hard problem [6]. Despite significant recent progress in algorithm development, this is currently an open challenge in computational statistics that has remained infeasible except for cases with a small number of variables [8].

MCMC optimization methods have been used to perform the search algorithm [5, 8] in this super-exponentially growing space. MCMC methods are capable of skipping the local optima in the search space and are preferred over heuristic search methods in multi-modal distributions. Following [5, 8, 22] we use MCMC in the order space instead of directly searching the graph space. The order space is smaller than the graph space ($2^{O(n\log(n))}$ vs. $2^{\Omega(n^2)}$, where n is the number of variables or nodes of the BN). Moreover, the order space enables powerful search operations (such as swapping the position of two nodes) leading to better exploration of the search space.

For any BN there exists at least one total ordering of the vertices, denoted as $\prec$, such that $V_i \prec V_j$ if $V_i \in \Pi_j$. An example of a BN and a compatible order is depicted in Figure 1. To employ MCMC in the order space, one performs a random walk in the space of possible orders and at each step accepts or rejects the proposed order based on the current and proposed orders' scores (according to the Metropolis-Hastings rule [13]). The score of a given order is decomposed into the scores of the graphs consistent with the order: $Score(\prec|\mathcal{D}) = \sum_{\mathcal{G}\in\prec} Score(\mathcal{G}|\mathcal{D})$, which can be efficiently calculated as shown in [10]:

$$\begin{aligned} Score(\prec|\mathcal{D}) &= \sum_{\mathcal{G}\in\prec} \prod_{i=1}^{n} LocalScore(V_i, \Pi_i; \mathcal{D}, \mathcal{G}) \\ &= \prod_{i=1}^{n} \sum_{\Pi_i \in \Pi_\prec} LocalScore(V_i, \Pi_i; \mathcal{D}, \mathcal{G}) \qquad (1) \end{aligned}$$

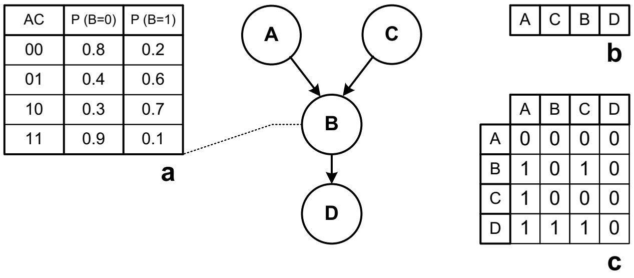


Figure 1: A Bayesian Network with 4 variables and Binomial CPDs. a: The CPD at node B, which encodes the distribution of B given different states of its parents. b: An order that is compatible with the graph. c: Two dimensional one-hot encoding of the same order.

After MCMC converges, each order is sampled with a frequency proportional to its posterior probability. The high scoring orders include the high scoring graphs and these graph structures can be extracted from the orders in parallel.

Local score generation must finish before the MCMC Order Sampler can start walking the order space. The computation of local scores in Equation 1 depends on the scoring metric as well as CPD formulations. Local scores are calculated from the raw data for the possible parent sets of that node and only once for each node. The MCMC Order Sampler then scores randomly generated orders by traversing the list of possible parent sets for each node and accumulating the local score of each parent set that is compatible with the current order. Each node's score is then multiplied together to form the order score, according to Equation 1. The MCMC Order Sampler usually needs to perform hundreds of thousands or even millions of iterations before convergence and this is where more than 90% of computation takes place [5].

Therefore BN inference is a three step computation problem:

1. **Preprocessing**: local scores are generated according to the CPD formulation and scoring method for possible parent sets.
2. **Order and Graph Sampling ("MCMC Kernel")**: the high scoring order structures are learned from data and the high scoring graph structures are sampled from these orders. This step usually takes 90% of execution time on general purpose processors.
3. **Postprocessing**: higher level analysis processes such as visualization and other tools for comparing and using the learned graph structures.

To achieve high-throughput as well as flexibility, ParaLearn uses general purpose computing tools (GPPs) to perform the first and third steps and optimizes the MCMC Kernel (second step) by using a customized hardware implementation. The key observation that leads to this efficient and flexible design is that while the different BN scoring and search methods differ in the first and third steps, all feature the same kernel (second step). In the next section we will explain how the computations needed for the MCMC Kernel are mapped and optimized to FPGA platforms.

3. IMPLEMENTATION

In this section, we discuss how the MCMC Kernel is implemented on a single FPGA and scaled to multiple FPGAs.

3.1 Mapping the MCMC Kernel to an FPGA

The MCMC Kernel's computation is performed by an MCMC Controller, Scoring Unit and Graph Sampler Unit, shown in Figure 2.

3.1.1 MCMC Controller

The MCMC Controller Unit (MCU) performs the sample (order) generation as well as deciding whether to keep or discard a sample based on its score. The MCU uses the two dimensional encoding to represent orders as in [5]. In the two dimensional encoding of an order, each row represents a "local order" and encodes the possible parents of the node corresponding to that row, as is shown in Figure 1. The MCU performs the random walk by swapping the position of two nodes (corresponding rows and columns in the two dimensional encoding) to create new samples, and accepts the new sample with probability A based on the Metropolis-Hastings rule:

$$A(\prec \to \prec^{'}) = \min\left(1, \frac{Score(\prec^{'}|D)}{Score(\prec|D)}\right)$$

Since the scores represent very small probabilities, all computations are done in logarithmic (*log*) space. This also allows the reduction step on the FPGA to be implemented as simple additions rather than multiplications. Therefore, the MCU decides whether to keep the new sample by subtracting the old score from the new score and compares this number to the *log* of a random number. A hardware LFSR and a *log* look-up table are designed to generate the required random numbers.

3.1.2 MCMC Scoring Unit

When the MCU generates an order, the MCMC scoring unit (MSU) is responsible for calculating and returning the resulting score to the MCU. As introduced in Section 2, the scoring process for each local order involves traversing over a set of parent sets (stored in memory) and accumulating the local score of each compatible parent set. The accumulation process is associative and commutative and can be decomposed into as many smaller, parallel accumulations as the implementation platform can accommodate.

On Virtex-5 FPGAs, each accumulation circuit (called a scoring core or 'SC') requires Block RAM (BRAM) for parent set and local score lookups, regular FPGA logic to implement the accumulator itself, and either BRAM or distributed/LUT RAM (LRAM) to implement a *log* lookup operation. Score look-ups are made with BRAMs because of their high bandwidth and single cycle access. Using LRAM for *log* look-ups saves BRAM resources for parent sets and scores but constrains routing high-utilization/frequency designs.

At the MSU's top level, the logic responsible for scoring one node (or 'SN') is replicated for each node that the system must support. Nodes are then divided into SCs, which are

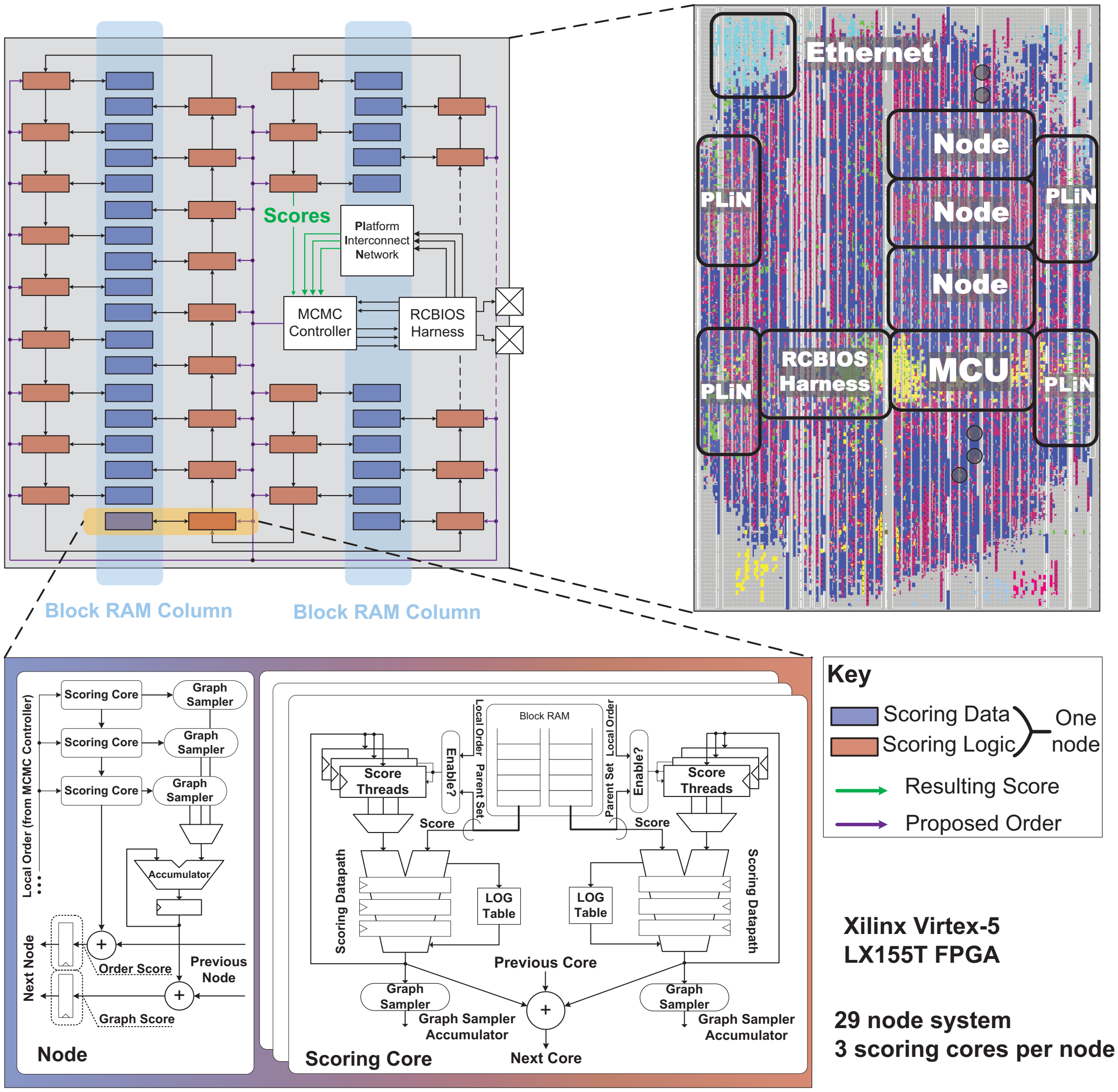


Figure 2: A ParaLearn MCMC Kernel on a single FPGA, supporting 29 nodes.

assigned *BRAMperSC* 18Kbit BRAM, given by:

$$\text{BRAMperSC} = \left\lceil \frac{\text{Network Size}}{32 \times \frac{1}{\text{Ports}}} \right\rceil + \left\lceil \frac{\text{FP Precision}}{32 \times \frac{1}{\text{Ports}}} \right\rceil$$

Ports is either 1 or 2 for Virtex-5 FPGA BRAMs, meaning that each BRAM can be dual-ported to increase performance. Within each SC, the score accumulation datapath is pipelined and time-multiplexed between hardware scoring threads in order to maintain a one-score-per-cycle accumulation in the steady state, regardless of clock frequency. Each SN is assigned as many SCs as is necessary to support the problem specified parent-sets-per-node (PPN) constraint. This architecture strives to minimize BRAM depth and maximize throughput per SC, which in turn maximizes performance.

When the MCU broadcasts an order, the order is split into local orders and, through a pipeline that fans out across the FPGA, sent to each node's SCs in parallel. Each SC starts and finishes scoring its local order at the same time since the scoring process is a data independent memory traversal. Once scoring is complete, each SC's result is accumulated as shown in Figure 3. Linear accumulation is used to accumulate in nearest-neighbor fashion across SCs and SNs because the Virtex-5 FPGA's BRAMs are arranged in columns.

To avoid rerunning FPGA CAD tools for different networks, each node and SC in the MSU can be enabled or disabled at runtime. Each SC that does not receive any parent sets from software is considered disabled and is bypassed during the score accumulation process. If all SCs within an SN are bypassed in this way, the entire SN is bypassed. This means that an MCMC Kernel that is designed to accommodate an N node and P PPN system can accommodate any number of nodes $\leq N$ and any number of PPN $\leq P$.

When the system is synthesized to support more PPN than it needs to support a given network, the system can

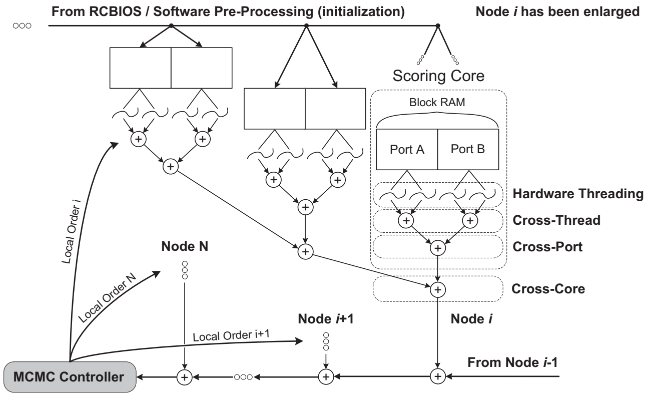


Figure 3: Abstract connectivity and reduction tree for the MSU. This example shows two threads per port, two ports per SC, three SCs per node, and N nodes.

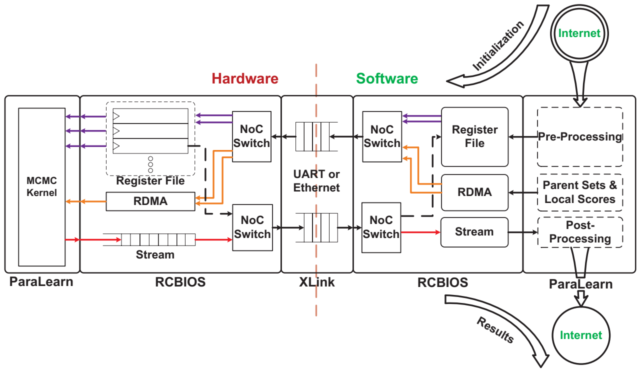


Figure 4: RCBIOS infrastructure and hardware/software blocks supporting MCMC. From right to left, the system is initialized. From left to right, results are collected.

achieve a greater speedup. This is because every parent set BRAM is designed to behave like a restartable FIFO. Thus, each SC only scores as many parent sets as have been loaded at initialization. By spreading out parent sets across the SCs as evenly as possible, the kernel can optimize at runtime for arbitrary problems.

3.1.3 MCMC Graph Sampler Unit

Determining which graph will produce the highest score from each order is a process typically undertaken by software after order sampling is complete [17]. ParaLearn's FPGA Kernel determines graphs and their scores in parallel with the order scoring process.

The highest scoring graph consists of the highest scoring parent set for each node. Thus, each SC must keep track of the highest scoring parent set it has seen throughout each iteration. When order scoring is complete, the graph score is accumulated separately across SCs and the graph itself is assembled and sent to software (see the "Graph-Samplers" in Figure 2). Assembling the final graph takes more clock cycles than accumulating the order score, so the i^{th} order's graph is assembled while the $(i+1)^{st}$ order is issued by the MCU. Through integrating the graph sampling step into the order sampler, graph sampling costs zero time overhead.

3.1.4 GateLib & RCBIOS

GateLib [12] is a standard library of hardware and software code with an integrated build and test framework. GateLib was developed at U.C. Berkeley and includes everything from standard registers, to DRAM controllers, to build and test tools which ensure that both the library and designs such as ParaLearn are operating correctly.

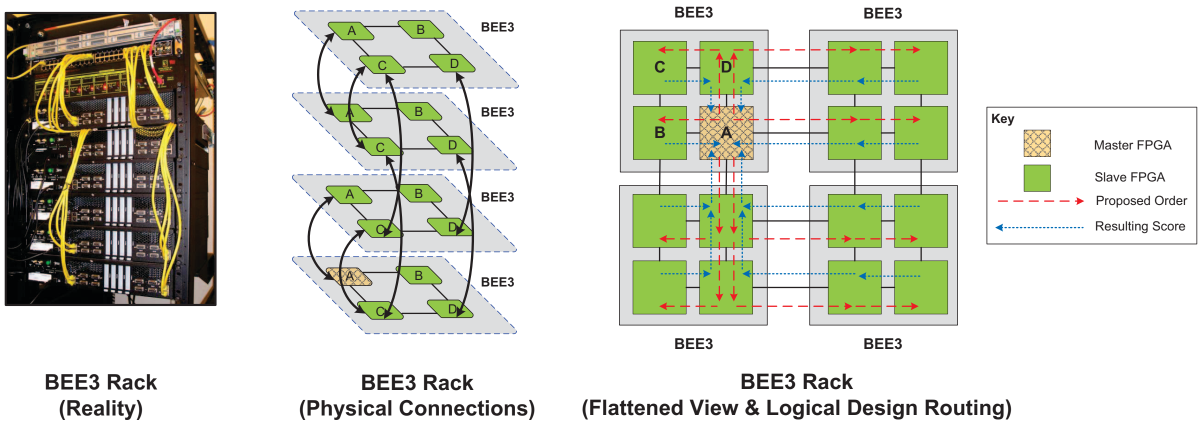
For ParaLearn the most significant component of GateLib is the sub-library called RCBIOS, which provides a Reconfigurable Cluster Basic Input/Output System for the kind of FPGA computing platforms used in this work. RCBIOS provides three primary interfaces between hardware and software: remote memory access (RDMA), control and status registers and data streams. RCBIOS is built on top of a flexible Network-on-a-Chip interconnect and XLink, another sub-library of GateLib which provides simple hardware/software communication. Implemented directly in RTL Verilog (i.e. without any service processors), the RCBIOS modules provide high-performance, low cost and easy-to-use communications between the FPGAs and the front-end system.

ParaLearn uses RCBIOS to initialize system state and to collect resulting graphs and their scores as shown in Figure 4. After the pre-processing step, ParaLearn software uses RCBIOS to send parent sets, local scores, a node count, an iteration count, and an initial order to the reconfigurable cluster (referred to as *Load Time*). As each order is scored, its highest scoring graph and that graph's score is streamed back to software and post-processed.

4. SCALABILITY

When the problem becomes too large for a single FPGA, the MCMC Kernel must be spread across multiple FPGAs. ParaLearn leverages the BEE3 [2] platform's mesh network (see Figure 5), composed of both interchip links between the four FPGAs, and CX4 links between BEE3's. A multi-FPGA MCMC Kernel is composed of a master FPGA and one or more slave FPGAs. The master FPGA contains MCU and MSU logic. The slave FPGAs are used for their MSUs only and score any local order which arrives, returning the partial score to the master FPGA. In all cases, since the logic which differentiates master and slave is relatively small compared to total FPGA area, a single FPGA bit-file is used for both master and slave, and each is configured at runtime through software. This simplifies the process of reconfiguring and managing the system as more FPGAs need to be introduced to support larger problems.

ParaLearn augments the BEE3 mesh network with a general cross-chip router to support additional FPGAs without having to modify the pre-existing system. The router, a dedicated circuit called the "Platform Interconnect Network" or PIN [18], interfaces with firmware designed to support both interchip tuning and interboard protocols. Furthermore, PIN uses dimension order routing to channel packets both to and from the master FPGA. When the MCU broadcasts an order, a subset of the order is sent directly to the master's MSU, and the remainder is packetized and sent to the slave FPGAs in the system. As the scoring process takes



BEE3 Rack (Reality)

BEE3 Rack (Physical Connections)

BEE3 Rack (Flattened View & Logical Design Routing)

Figure 5: Different views of a multi-FPGA ParaLearn Kernel.

the same amount of time for every node, the master FPGA decreases total scoring time by sending local orders to slave FPGAs whose hop latency to the master[1] is greatest, first. The hop latency is made up of hops across interchip *and* interboard links, and therefore is not completely determined by the manhattan distance between two FPGAs.

4.1 FPGA Platforms

ParaLearn's scalability, coupled with the dearth of industry standards, makes it important for us to address the tradeoffs between different multi-FPGA or reconfigurable cluster platforms. Reconfigurable cluster platforms differ in the number and type of FPGAs per board, the connection topology, as well as supported peripherals and interfaces. Platforms range from the Dini Group DN9000K10 which consists of a mesh of FPGAs, to the Xilinx ACP that uses an Intel front-side-bus (FSB) to connect a smaller number of FPGAs to a CPU.

ParaLearn targets the BEE3 reconfigurable cluster platform from BEEcube [2]. The BEE3 consists of four Virtex-5 LX155T FPGAs, two channels of DDR2 SDRAM per-FPGA, and a point-to-point interconnect. The interconnect for each FPGA consists of two "interchip" links, which form a ring on the board, and two 10GBase-CX4 Ethernet interfaces for interboard connections. ParaLearn as discussed in section 4, uses these connections to assemble a mesh network of MCMC cores.

Unlike platforms that embed a full network of FPGAs in a single PCB, such as Dini Group boards, the CX4 connections on the BEE3 are cable-based, which allows the system to be reconfigured to meet the application's needs. This allows different FPGAs in the system to be connected directly together in order to reduce inter-FPGA hop count, or to increase the size of the overall system. These CX4 connections come at a price, having latency on the order of several dozen cycles, as opposed to the 5 cycles between FPGAs on one board.

When comparing the BEE3 against bus-connected platforms like the Xilinx ACP, there is a tradeoff between cluster size and front-end communication. Bus-connected platforms like the ACP are limited by bus sharing and score accumulation will scale linearly in time with the number of FPGAs. By contrast the BEE3 can take advantage of score reductions at each hop, accumulating multiple FPGA's score at a single time. Furthermore, the cost of the ACP systems must include a processor which is of no particular use to ParaLearn and adds to purchase, complexity and maintenance costs.

While ParaLearn is currently implemented on a BEE3 system due to the characteristics of the algorithm, this is not the only compatible implementation platform, and the system can be efficiently implemented on other platforms. In particular the system was developed in part on the Xilinx ML505 demonstration boards, which are comparable to a quarter of a BEE3, and provide a low-cost entry level FPGA platform.

5. RESULTS AND ANALYSIS

Our primary study analyzes 22 proteins in human cancer T-Cells with data from CyTof technology. We use 10,000 single cell measurements of the 22 proteins and limit the search indegree for the graph search to 4. Therefore, we have 7547 possible parent sets for each node and that we need at least two Virtex-5 LX155T FPGAs to implement one MCMC Kernel.

5.1 Quality of Results

To assess the correctness of the FPGA design, we tested the software and FPGA versions of the algorithms on synthetic data simulated from known BN structures such as the ALARM [1] network and we were able to reconstruct the network on both implementations. The only approximation in the FPGA implementation compared to the software version is the fixed point conversion of the local scores and lookup table entries. We used 32 bit fixed-point precision and while this results in a small change in the graph scores (less than 0.1 - that is about 0.1% of the score), the relative orderings of the graph structures do not change and the best graph structures found by the two implementations are identical.

With simulated data like the ALARM network, the CPD formulations are known, while with real data like CyTof, we do not know which CPD is the closest representative of the underlying interactions in the system. Therefore we applied two different CPD formulations to learn the interaction in this system—shown in Sections 5.1.1 and 5.1.2.

[1] Measured in clock cycles across the mesh.

As we expected the final results of the two kernels are significantly different. The Multinomial encoding results in a sparser model with 14 edges while the Linear Gaussian results in a denser model with 63 edges. There are 6 edges that appear in both models. These different graphs need to be further validated by scientists and domain experts and their true quality should be measured by their predictive power on new data sets from the STN.

5.1.1 Multinomial CPD

For the Multinomial representation we need to discritize raw data that is continuous. We used Biolearn software [3] to perform the discritization to a three-level discrete data set. The local score of a given parent set for each node using Bayesian formulation (with Dirichelet priors on parameters) is calculated as: $BayesianLocalScore(V_i, \Pi_i; \mathcal{D})$

$$LS_{V_i,\Pi_i} = \log\left(\prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ik})}{\Gamma(\alpha_{ik} + N_{ik})} \prod_{j=1}^{|V_i|} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})}\right)$$

where $r_i = \prod_{V_j \in \Pi_i} |V_j|$ [7]. α is the BDe prior parameter introduced in [14]. N_{ik} and N_{ijk} are sufficient statistics (counts) that are calculated from experimental data $\mathcal{D}$.

5.1.2 Linear Gaussian CPD

The joint probability distribution of variables in a Gaussian network is a multivariate Gaussian distribution. We used the BIC scoring method and, as shown in [11], the local scores can be calculated as: $BICLocalScore(V_i, \Pi_i; \mathcal{D})$

$$LS_{V_i,\Pi_i} = \frac{-mn}{2}\log(2\pi e) - n\sum_{i=1}^{m}\log\left(\frac{detS^{\pi_i X}}{detS^{\pi_i}}\right) - \gamma\frac{\log(n)}{2}\sum_i |\Pi_i|$$

where m is the number of variables and n is the number of observations. $S^{\pi_i X}$ is the covariance matrix of the node and its parents and S^{π_i} is the covariance matrix of the parent variables. γ is the penalty parameter that is used to adjust the BIC score to penalize the more complex networks in the search algorithm.

5.2 Design Space Exploration

Sections 5.2.2 through 5.2.5 present different studies used to evaluate ParaLearn in terms of scalability, performance, area and power. In our analysis, we use an "orders per second" (OPS) metric to determine performance. Unless otherwise noted, all tests are run on the following hardware:

FPGA: Xilinx Virtex-5 XC5VLX155T (-2 speed) FPGAs

GPP: Intel(R) Core(TM) i7 CPU QuadCore running at 3.07 GHz, with 12 GB of Memory and with an advertised TDP of 130 W.

For perspective, the Virtex-5 is a generation-old FPGA family and the XC5VLX155T is a mid-sized chip in the family.

5.2.1 Software Generator

To help conduct performance studies, we use software to generate MCMC Kernels that are optimized for different networks. The generator takes as input the desired network size, PPN, and parameters describing the implementation platform (such as BRAM dimensions and the number of FPGAs available to solve the problem). From this information, the generator sets parameters used by the FPGA CAD tools that describe how many SCs should be instantiated in each node, and also how each of the SCs should be implemented.

To build the optimal configuration, the software generator first determines the theoretical maximum performance through fixing the network size, PPN, and hardware resources, while scaling the number of SCs per node. Generally, performance will increase with the number of SCs to the point where the result accumulation step overwhelms the SC's execution time. Figures 7 and 9 exemplify the pare-to optimum curves as a function of SCs per node.

The generator varies the number of SCs per node by adding more single-ported SCs or dual-porting existing SCs. Adding single-ported SCs costs the most BRAM and FPGA logic, but can be done incrementally until the system runs out of FPGA fabric. Using dual-ported SCs is "all or nothing"—if one SC is dual-ported, the rest have to be as well or the performance benefit is masked by the slower SCs. Adding single-ported SCs also increases the maximum PPN that the system can support, while dual-porting does not. ParaLearn dual-ports SCs when the required PPN is low or the system is BRAM constrained.

5.2.2 Current Scalability

In this section we compare the 22 parameter[2] CyTof data across currently attainable[3] FPGA configurations in order to show how speedup and power consumption is affected by the hardware configuration used to support the problem.

BRAM port mode	SCs / FPGA	Clock (Mhz)	OPS	Power / FPGA (W)	Power / Problem (W)
Two FPGAs (nodes per FPGA = 11)					
Single	88	100	86,730	10.56	21.12
		150	**122,951**	13.82	27.64
Three FPGAs (nodes per FPGA = 8)					
Single	64	100	88,106	10.33	30.99
		150	124,792	12.38	37.14
		200	159,109	10.96	32.88
Double	128	100	144,092	11.72	35.16
		150	**195,567**	14.39	43.17
Four FPGAs (nodes per FPGA = 6)					
Single	48	100	84,962	9.94	39.76
		150	119,048	11.26	45.04
		200	153,022	10.13	40.52
Double	96	100	137,363	10.56	42.24
		150	184,729	12.80	51.20
		200	**226,501**⊕	11.85	47.40
Eight FPGAs (nodes per FPGA = 3)					
Single	24	100	77,640	9.73	77.84
		150	109,649	11.06	88.48
		200	152,091	10.33	82.64
Double	48	100	114,679	10.68	85.44
		150	160,600	11.06	88.48
		200	**201,005**	10.51	84.08

Table 1: MCMC Kernel study on the 22 parameter CyTof data ($PPN = 7547$) on one and two BEE3 boards. The highest performance "attainable" configuration for each system composed of N FPGAs is shown in bold.

The OPS column in Table 1 shows the performance impact of spreading the 22 parameter problem across different

[2] One parameter corresponds to one node.

[3] An "attainable" configuration can fit into the allotted hardware and meet timing at the specified frequency.

arrangements of FPGAs. For each number of FPGAs, trials are run for {100, 150, 200} Mhz clock frequencies and for both single and dual-ported SCs. To isolate the effect of scaling across FPGAs, all configurations support exactly 8 single or dual-ported SCs per node. If a configuration in this permutation isn't listed in Table 1, it was not attainable. As more FPGAs are used to support the problem, configurations using the same number of SC ports and the same clock frequency tend to degrade in performance due to an increased hop latency in the FPGA mesh. With additional hardware, however, more aggressive configurations (in terms of SC porting and clock frequency) become "attainable."

The Power columns in Table 1 show power utilization for different configurations. All power results are gathered through the Xilinx XPower Analyzer after simulating traces through the system running the 22 parameter CyTof data set. As we have fixed the number of SCs per node in each experiment, the power consumption per FPGA drops in sparser FPGA configurations however total system power (taking into account the number of FPGAs) tends to increase. We found that this is due to fixed cross-FPGA communication overhead, namely the interchip links in each sample (which produced ≈ 3.8 W) and the GTP connections (responsible for ≈ 1 W) in the 8 FPGA experiment. Furthermore, Table 1 shows that in general, 150 Mhz configurations require more power than 200 Mhz configurations. We attribute this to there being an extra clock in 150 Mhz configurations (interchip requires a 200 Mhz clock and RCBIOS requires a 100 Mhz clock—thus the MCMC Kernel can use those existing clocks when running at 100 or 200 Mhz).

Figure 6 shows the FPGA resource utilization for all attainable 2–4 FPGA configurations in Table 1. In general, using dual-ported SCs increases performance improves by ≈ 1.6× and more evenly uses FPGA resources. In practice, we observed that designs that utilize a majority of FPGA logic resources could not route at higher frequencies when LRAM was also heavily utilized. To get around this problem for 200 MHz configurations, we more heavily relied on BRAM rather than LRAM to implement *log* tables, relative to 150 MHz configurations.

5.2.3 Projected Scalability

In this section, we explore how larger FPGAs can be used to increase system speedup. For this experiment, we used the software generator (Section 5.2.1) to project performance over a range of SC counts per node (shown in Figure 7). We verified these trends through:

1. Direct comparison with hardware performance for "attainable" configurations.

2. Gate-level simulation at fixed intervals on the curve, and at the projected upper-bound, for configurations requiring larger FPGAs.

Extrapolated from Figure 7, larger FPGAs will provide between 1.5× and 1.7× speedup over current configurations assuming a comparison across the same number of FPGAs. If the problem is able to fit onto a single FPGA, power will be minimized[4] and speedup can increase by up to 2.61×.

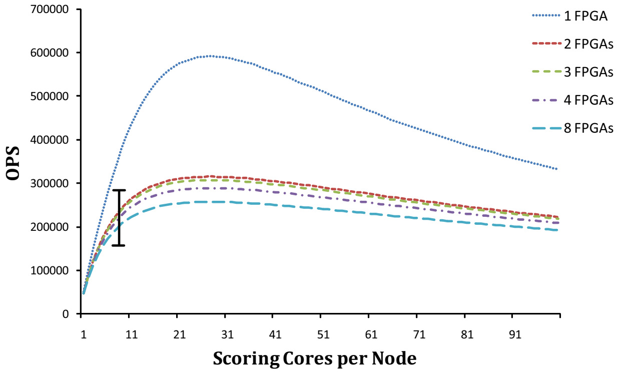


Figure 7: Scaling the number of SCs per node for the 22 parameter CyTof data. All experiments are carried out with dual-ported SC BRAMs, a 200 Mhz clock, and assuming that an FPGA chip can support any number of SCs. The vertical bar indicates the point that our current hardware allows us to achieve.

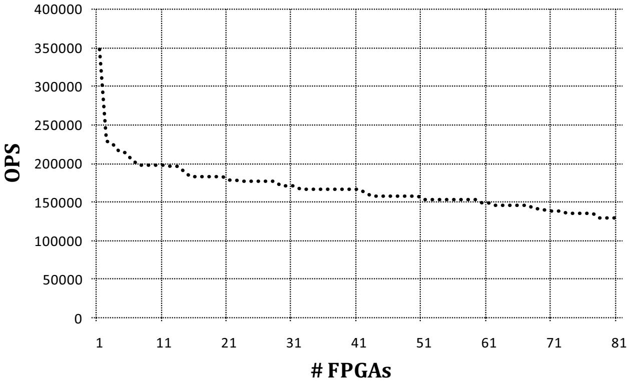


Figure 8: Scaling the number of FPGAs for the 22 parameter CyTof data. Samples are taken at 200 Mhz with dual-ported SCs.

It is also important to consider how performance degrades as the number of FPGAs in the system increases. To isolate this effect, we constrain the software generator to 8 dual-ported SCs per node while increasing the number of FPGAs, as shown in Figure 8. In order to maximize performance, each FPGA mesh is arranged so that the hop latency from the master FPGA to any given slave FPGA is minimized. The occasional kinks in the graph are due to adding an FPGA with a new greatest hop latency. As can be seen, the greatest performance hit is moving from one to two FPGAs, with the performance decreasing linearly as the number of FPGAs increases.

5.2.4 Current Flexibility

This section studies how ParaLearn can scale to handle different networks.

As the FPGA CAD tools take an appreciable amount of time to run, it is important to study how a general FPGA bitfile, configured for up to N nodes and P PPN performs

[4] In addition to saving power with less FPGAs, single FPGA configurations do not use interchip and GTP connections, which we showed to be a large contributor to total power consumption.

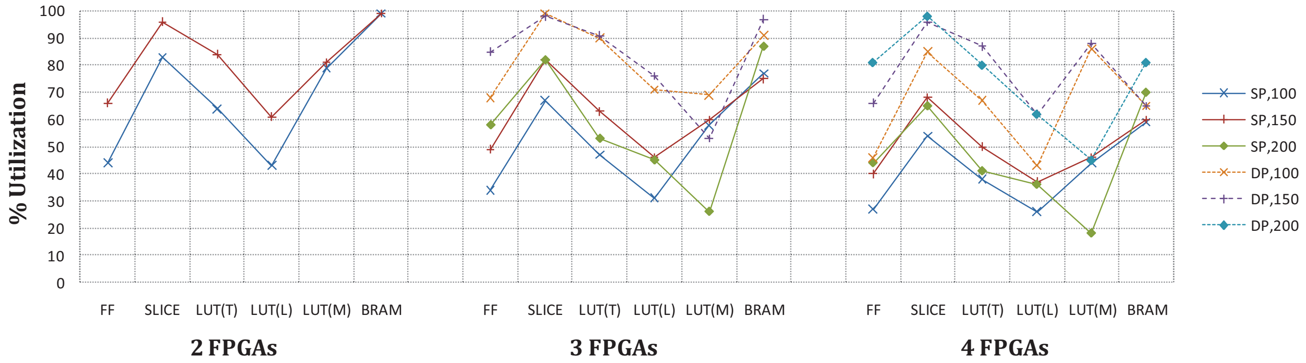


Figure 6: Resource utilization for the 22 parameter CyTof data. "LUT ({T, L, M})" refers to total LUTs, LUTs used as logic, and LUTs used as memory (LRAM), respectively. Each dataset is labeled "{S,D}P,X" which refers to single or dual-ported SC schemes and the clock frequency ('X') in Mhz.

for different problems. Table 2 shows different configurations for problems spanning over four FPGAs (a single BEE3 board). All configurations maximize PPN and place performance as a second-order constraint in order to tolerate denser networks. Thus, this study synthesizes only single-ported BRAMs and shows results for the highest attainable clock frequency (150 MHz for every experiment).

All configurations are first shown assuming that all available parent set memory is used. The 22 parameter CyTof data's performance is shown in rows whose PPN column is marked with a $\star$. The closest configuration to the CyTof data (^{A}) achieves .8× performance relative to the most aggressive, attainable bitfile (marked $\oplus$ in Table 1) while the next closest (^{B}) achieves .6× the performance, and the last (^{C}) maintains .4× performance. The drop-off is due to there being less SCs per FPGA, which is due to larger networks requiring wider parent set logic.

Nodes	Nodes / FPGA	SCs / Node	PPN			OPS
			Max	Actual	Limited	
8	2	58	59,392	128	99	–
16	4	28	28,672	28,672	1,941	113,208
24	6	15	15,360	15,360	10,903	110,457
				$\star$7,547	""	180,288^A
32	8	11	11,264	11,264	4,992	103,306
				$\star$7,547	""	134,168^B
40	10	8	8,192	8,192	9,920	91,296
				$\star$7,547	""	94,399^C

Table 2: MCMC Kernel flexibility study. All experiments are carried out with single-ported SCs and a 150 Mhz clock. "Max" refers to the highest PPN that our current hardware can support, "Actual" corresponds to the PPN used to benchmark the system (performance is shown in the "OPS" column) and "Limited" is the PPN when the indegree is fixed (Section 5.2.5).

Table 2 also shows the point at which the MCMC Kernel becomes BRAM limited in handling different networks. For a network of N nodes, there are a possible 2^{N-1} different parent sets. Theoretically, the 8 node network across four FPGAs can tolerate 59,392 parent sets but a network of that size will only ever have $2^{8-1} = 128$ parents (for this reason, an 8 node network would never be spread across four FPGAs in practice). The 16 node network, on the other hand could have 32,768 different parent sets, yet the MCMC Kernel can only tolerate 28,672 parents.

ParaLearn tolerates BRAM constrained networks through limiting the number of parent sets. This can be done in two ways:

1. Using smarter parent set filtering algorithms based on supervised learning techniques that can learn the subset of variables that are correlated with a node value and then include those as possible parent sets.

2. Limiting the maximum indegree of the graphs. With this approach, ParaLearn includes parent sets up to size K, where usually $2 < K < 6$. This decreases the parent set count from 2^{N-1} to $\sum_{i=0}^{K} \binom{N-1}{i}$.

Method 1 is more flexible as candidate parent sets for each node will be optimized and learned separately. For the 22 parameter CyTof data we used method 2 ($K = 4$) as it does not impose additional runtime and is a reasonable assumption for 22 node networks.

5.2.5 *Projected Flexibility*

Using Method 2, Figure 9 shows peak performance curves for each network, again as a function of SCs per node. The 8, 16, 22, and 24 node networks use indegree = 4 while the 32 and 40 node networks use indegree = 3. **X** notches in the graph represent where 22 parameter CyTof configurations sit on each curve that supports ≥ 22 nodes and ≥ 7547 PPN. As in Table 2, bitfiles designed for less similar networks, relative to the 22 parameter data, show less performance. In general, networks will lower PPN requirements reach peak performance with less SCs per node because the ratio between SC kernel time and result accumulation decreases as problem size decreases.

5.3 End-to-End Computation

In this section we compare 22 parameter CyTof end-to-end ("time to graphs") performance between an optimized GPP implementation [17] and an MCMC Kernel using different

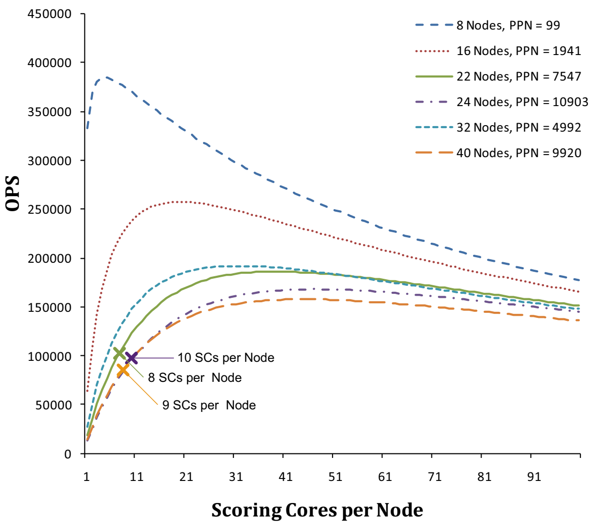


Figure 9: The effect of scaling the number of SCs per node, fixed over 4 FPGAs, for various networks. All experiments are carried out with single-ported SCs, a 150 Mhz clock, and assuming that an FPGA chip can support any number of SCs.

numbers of FPGAs. This study runs 100,000 MCMC iterations over 50 random restarts, where each random restart resets the system with a different initial order (starting point). After the MCMC Kernel completes, the final graph results are sorted by their scores and probabilities are normalized against a weighted average to produce a final graph.

Architecture	End-to-end time (s)
GPP (optimized)	0 +44.6+18.8=**63.4**
FPGA (2x)	4.5+40.6+ 0 =**45.1**
FPGA (3x)	4.5+25.5+ 0 =**30.0**
FPGA (4x)	4.5+22.1+ 0 =**26.6**

Table 3: MCMC Kernel end-to-end performance comparison against an optimized GPP solution. Times are broken into: *Load Time* + *Order Sampler* + *Graph Sampler*.

Table 3 shows how the Order Sampler's speedup scales with the number of available FPGA chips. The pre-processing time starts to dominate the problem as the core is accelerated. Preprocessing currently takes 185 seconds for multinomial CPDs and 50 seconds for Linear Gaussian CPDs.

6. SOURCES OF SPEEDUP

In this section we compare different MCMC algorithm implementations, using different optimizations, to show where the speedup in our approaches comes from. Table 4 shows the "best effort" end-to-end run times for a *baseline* GPP implementation, optimized GPP implementation (derived from [17]), and attainable FPGA configurations. For this particular problem and network's size, [17] has shown that the GPU does not perform well relative to the GPP. Thus, only GPP numbers are shown here. As can be seen, FPGA load time weighs down the FPGA's performance while the graph sampler weighs down the GPP.

The *baseline* GPP implementation was written in C and applies no software optimizations. The optimized GPP implementation performs bitwise operations where possible, uses software multi-threading, and caches results to improve performance. The FPGA implementation exploits the fine-grained parallelism afforded by the BRAMs and uses bitwise manipulation, however, does not cache. In the remainder of the section, we will distill the different optimizations used in each approach to try to explain performance differences.

First, we compare the benefits of threading the GPP and FPGA implementations. We implicitly implemented multi-threading on the FPGA through exploiting BRAM parallelism. Through adding `pthreads` to the baseline C implementation, we were able to achieve a 2× speedup with 10 threads running on a quad core/8 thread machine. The cross-thread communication in the result accumulation step seems to weigh down the `pthreads` speedup. The FPGA can best take advantage of parallelism because of fine-grained BRAM access and because it can optimize the result accumulation step at the gate level.

Second, we implemented a fixed point implementation of the baseline GPP model so that it can more closely be compared to the FPGA, which is also fixed point. Fixed point on the GPP does not improve performance. This may be due to the fact that the floating point units are nearly as optimized and fast as the integer units on modern processors.

Third, we compare the effect of caching on the GPP to the FPGA. Caching works by pairing orders with their scores, which is valid because a given order will always have the same score. We did not implement caching on the FPGA because it requires dedicated logic and makes execution time problem dependant (unlike this work's "orders per second" metric). Work done by [17] (which provided data for the optimized GPP in Tables 3 and 4) shows how different problem configurations have different cache performance. The benefit achieved through caching is based on how long it takes the problem to converge.

Caching is naturally suited for the GPP/GPU because these platforms have built-in caches that don't come at extra design cost. After paying the logic penalty, however, caching can be implemented on FPGAs. In order to construct an analytic model of cache performance, we consider the following types of caches:

Order: A cache that sits next to the MCU and caches whole orders.

Node: A cache at each node that stores only local orders and their partial scores. A GPP benefits from every hit in every node cache. The FPGA only benefits if *every*[5] node's cache hits in the same iteration.

Both caches' performance will depend on their capacity and hash function. On an FPGA, we have two options for cache storage: off-chip memory (we assume DRAM) and on-chip BRAM. DRAM requires a DRAM controller and has gigabytes of capacity, however carries an access latency that varies with address and locality. BRAM requires only the minimum support logic as is necessary and is single-cycle access, however takes up BRAMs that would otherwise be used for SCs. In order to maximize cache capacity, we will

[5] If a single node misses in the cache, the rest of the system has to wait for the partial score to be computed at that node.

Architecture	Time (s)		
	100,000 iterations, 0 restarts	100,000 iterations, 50 restarts	100,000 iterations, 100 restarts
GPP (*baseline*)	0 +970+ 20 = **990**	0 +48500+ 60 = **48560**	0 +97000+ 60 = **97060**
GPP (optimized)	0 + 5.2 +0.85= **6.05**	0 + 44.6 +18.8= **63.4**	0 + 78.8 +31.3= **110.1**
FPGA (2x)	4.5+0.81+ 0 = **5.31**	4.5+ 40.6 + 0 = **45.1**	4.5+ 81.3 + 0 = **85.8**
FPGA (3x)	4.5+0.51+ 0 = **5.01**	4.5+ 25.5 + 0 = **30.0**	4.5+ 51.1 + 0 = **55.6**
FPGA (4x)	4.5+0.44+ 0 = **4.94**	4.5+ 22.1 + 0 = **26.6**	4.5+44.15+ 0 = **48.65**

Table 4: MCMC Kernel end-to-end performance comparison against an optimized GPP solution. Times are broken into: *Load Time* **+** *Order Sampler* **+** *Graph Sampler***.**

# FPGAs	Time (s)				Speedup
	Baseline	Order	Node	Both	
3	25.50	25.47	10.17	10.07	2.53×
4	22.1	21.98	8.63	8.53	2.59×

Table 5: FPGA speedup when using both order and node caches. This test compares 22 node CyTof data over 100,000 iterations and 50 restarts.

assume DRAM-based caches. Since DRAM is centralized, logic dedicated to hash functions and cache control are a one time cost.

Work done by [17] has shown us that when running the 22 node CyTof data with 100,000 iterations and 50 restarts, there are 22,978 hits in the order cache, 103,757,230 hits in the node caches (∼ 95% GPP node cache hit rate), and 3,274,755 iterations that hit in every node's cache (∼ 65% FPGA node cache hit rate). Given this, we construct an analytical model of cache benefit on the FPGA:

$$\mathrm{CPO} = \frac{\mathrm{Raw} \times (\mathrm{Total} - \mathrm{O_h} - \mathrm{N_h}) + \mathrm{O_a} \times \mathrm{O_h} + \mathrm{N_a} \times \mathrm{N_h}}{\mathrm{Total}}$$

where CPO is "cycles per order," $\mathrm{O}_{\{\mathrm{h,a}\}}$ are the order cache hit count and access time (in cycles), $\mathrm{N}_{\{\mathrm{h,a}\}}$ are the same figures for the node cache, *Raw* is cycles per order without caching and $Total$ is the number of orders proposed to the system. Table 5 shows a theoretical study of how the FPGA implementation would fare with the caching optimization. We assume a pipelined hash function initiates back-to-back DRAM reads for node cache lookups and a single read for an order lookup. Thus, cache lookup latency is given by:

$$\mathrm{O_a} = \mathrm{T_{MCU}} + \mathrm{T_h} + \mathrm{DRAM_a} = 5 + 10 + 25 = 40$$

$$\mathrm{N_a} = \mathrm{T_{MCU}} + \mathrm{T_h} + \mathrm{DRAM_a} + \mathrm{NS} = 5 + 10 + 25 + 22 = 62$$

where $\mathrm{T_{MCU}}$ is the mandatory overhead in the MCU, $\mathrm{T_h}$ is a conservative cycle latency for the pipelined hash function, $\mathrm{DRAM_a}$ is a conservative DRAM access time (in cycles), and NS is the network size (in nodes). This study assumes that all FPGA caches are non-blocking and proceed with order scoring during the cache look-up. Furthermore, we assume that partial scores retrieved from the node cache accumulate as each new score arrives, hiding the result accumulation step's latency. For this data set, we observe ≈ 2.59× improvement against the baseline FPGA implementation when using both caches. Furthermore, the caching FPGA attains a ≈ 4.86× improvement over the optimized GPP.

7. DISCUSSION AND CONCLUSIONS

Graphical models and in particular Bayesian Networks are of great importance in our data intensive era. They can help scientists learn causal interactions and are useful tools in personalized medicine where there are substantial high-throughput experiments to measure gene expressions, DNA sequence and protein abundance. In this paper we demonstrated ParaLearn's usability in learning network structure and used the STN in human T-cells as a motivating example. ParaLearn's accelerated and integrated solution enables scientists and physicians to use computationally intensive BN inference algorithms in real time settings.

As discussed in Section 5, the learned graphs can be quite different for different CPD formulations. While conventional BN structure learning techniques limit their search to one or few CPD models because of computation complexities, ParaLearn can execute many of these possible CPDs in parallel. We would like to further explore this *MetaSearch* approach in our future work. We will study how the different learned graphs can be validated and compared or integrated to produce a final model. We believe that the novel *MetaSearch* approach increases the robustness of the algorithms and makes BN modeling a more useful and reliable tool for scientists.

In Section 5.2, we provided power results for currently attainable FPGA configurations. Comparing our power results to the GPP shown in Section 5.2, we show a 2.74× power improvement (considering only the FPGA and GPP die) using our most aggressive configuration over 4 FPGAs. Furthermore, our lowest performance 2 FPGA configuration from Table 1 shows a 6.15× power advantage and maintains 1.02× the performance of the optimized GPP in Table 3. We used GPP TDP to perform power analysis meaning that our power advantage is an upper bound. In a complete system, the infrastructure around both the FPGA and GPP costs additional power. Unlike a GPP, however, ParaLearn does not require any external devices except for an Ethernet PHY, reducing off-FPGA power overhead.

In Section 5.2, we also projected how performance will increase with larger and more FPGAs. To help meet projected speedups, the new generation of Xilinx FPGAs, the Virtex-6 family, which retails in 2010, provides up to 5× the BRAMs (our limiting factor) and 3× the logic of the FPGAs used in this work. Architecturally, these new FPGAs are the same as our current FPGAs. Thus, ParaLearn can be ported to this new platform to realize the projected speedup. Taking into account larger FPGAs, without caching, we project 4.88× speedup relative to the current optimized GPP (a single Virtex-6 can accommodate the 22 parameter CyTof problem).

As we presented in Section 5.3, the pre-processing step that currently runs on a single GPP has become the bottleneck relative to the FPGA kernel. We are currently exploring a parallel implementation of the pre-processing step for different CPD methods. The pre-processing step for discrete data maps best to an FPGA as it involves bit operations and *popcount* functions. The continuous data pre-processing would probably benefit most from a GPU implementation because of the heavy usage of floating point matrix operations. On the FPGA, however, pre-processing would have the added benefit of eliminating the *Load Time* bottleneck revealed in Section 5.3 and 6.

In Section 6, we observed how caching, and other optimizations, have benefitted GPP implementations by several orders of magnitude over a baseline. We plan to explore different caching implementations on the FPGA in order to learn about the performance/area tradeoff in that space. Factoring caching into our projected speedup with larger FPGAs, we have a 8.15× advantage over the current optimized GPP.

ParaLearn is a framework and design suite for conducting interaction networks research. We have shown different approaches and performed a design space exploration at the pre-processing, order, and graph sampling steps. Through this work, our goal is to make interaction network studies tractable as technology, such as CyTof, produces data sets at orders of magnitude greater size and complexity than ever seen before.

8. ACKNOWLEDGEMENTS

This project was supported by NIH grant 130826-02. The authors would like to thank Sean Bendall for carrying out the Cytof experiments for this paper and sharing the data. We would also like to express our gratitude to Michael Linderman et. al for sharing their data and results on the optimized GPP implementation [17].

9. ADDITIONAL AUTHORS

Eric N. Glass (eglass@stanford.edu)
Electrical Engineering Department, Stanford University,
Stanford, CA, USA 94305,
Karen Sachs (karensachs@stanford.edu)
Microbiology and Immunology Department, Stanford
University, Stanford, CA, USA 94305
Daniel Burke (drburke@berkeley.edu)
Electrical Engineering and Computer Sciences Department,
University of California, Berkeley, CA, USA 94720
Zoey Zhou (cuizy@stanford.edu)
Electrical Engineering Department, Stanford University,
Stanford, CA , USA 94305

10. REFERENCES

[1] Bayesian network repository: http://compbio.cs.huji.ac.il/repository/.

[2] Beecube: http://beecube.com/.

[3] Biolearn software: http://www.c2b2.columbia.edu/danapeerlab

[4] D. R. Bandura, V. I. Baranov, O. I. Ornatsky, A. Antonov, R. Kinach, X. Lou, S. Pavlov, S. Vorobiev, J. E. Dick, and S. D. Tanner. Mass cytometry: Technique for real time single cell multitarget immunoassay based on inductively coupled plasma time-of-flight mass spectrometry. *Analytical Chemistry*, 81(16):6813–6822, 2009.

[5] N. Bani Asadi, T. H. Meng, and W. H. Wong. Reconfigurable computing for learning bayesian networks. 2008.

[6] D. M. Chickering. Learning bayesian networks is np-complete. In *Learning from Data: Artificial Intelligence and Statistics*. V. Springer Verlag, 1996.

[7] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Mach. Learn.*, 9(4):309–347, 1992.

[8] B. Ellis and W. H. Wong. Learning causal bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103(482), 2008.

[9] J. H. Friedman, T. Hastie, and R. Tibshirani. Sparse covariance estimation with the graphical lasso. *Biostatistics*, 2007.

[10] N. Friedman and D. Koller. Being Bayesian about Bayesian network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1–2):95–125, 2003. Full version of UAI 2000 paper.

[11] D. Geiger and D. Heckerman. Learning gaussian networks. Technical Report MSR-TR-94-10, Redmond, WA, 1994.

[12] G. Gibeling and et al. Gatelib: A library for hardware and software research. Technical report, 2010.

[13] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[14] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Mach. Learn.*, 20(3):197–243, September 1995.

[15] J. M. Irish, N. Kotecha, and G. P. Nolan. Mapping normal and cancer cell signaling networks: towards single cell proteomics. *Nature Reviews Cancer*, 6(2):146–155, 2006.

[16] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[17] M. D. Linderman, R. Bruggner, V. Athalye, T. H. Meng, N. B. Asadi, and G. P. Nolan. High-throughput bayesian network inference using heterogeneous multicore computers. *Proceedings of the 24th International Conference on Supercomputing*, 2010.

[18] I. L. M. Lin and J. Wawrzynek. Platform interconnect network for reconfigurable computing clusters. Technical report, 2010.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, September 1988.

[20] I. Pournara, C. S. Bouganis, and G. A. Constantinides. Fpga-accelerated bayesian learning for reconstruction of gene regulatory networks. *International Conference on Field Programmable Logic and Applications*, 2005.

[21] K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, and G. P. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523– 529, 2005.

[22] M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning bayesian networks. In *Proceedings of the Twenty-first Conference on Uncertainty in AI (UAI)*, pages 584–590, Edinburgh, Scotland, UK, July 2005.