2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)

Demystifying Bayesian Inference Workloads

Yu Emma Wang* Yuhao Zhu† Glenn G. Ko* Brandon Reagen‡ Gu-Yeon Wei* and David Brooks*
ywang03@g.harvard.edu yzhu@rochester.edu gko@g.harvard.edu reagen@fb.com {guyeon,dbrooks}@eecs.harvard.edu

*John A. Paulson School of Engineering and Applied Sciences Harvard University
†Computer Science Department University of Rochester
‡Facebook

***Abstract*—The recent surge of machine learning has motivated computer architects to focus intently on accelerating related workloads, especially in deep learning. Deep learning has been the pillar algorithm that has led the advancement of learning patterns from a vast amount of labeled data, or supervised learning. However, for unsupervised learning, Bayesian methods often work better than deep learning. Bayesian modeling and inference works well with unlabeled or limited data, can leverage informative priors, and has interpretable models. Despite being an important branch of machine learning, Bayesian inference generally has been overlooked by the architecture and systems communities.**

In this paper, we facilitate the study of Bayesian inference with the development of BayesSuite, a collection of seminal Bayesian inference workloads. We characterize the power and performance profiles of BayesSuite across a variety of current-generation processors and find significant diversity. Manually tuning and deploying Bayesian inference workloads requires deep understanding of the workload characteristics and hardware specifications. To address these challenges and provide high-performance, energy-efficient support for Bayesian inference, we introduce a scheduling and optimization mechanism that can be plugged into a system scheduler. We also propose a computation elision technique that further improves the performance and energy efficiency of the workloads by skipping computations that do not improve the quality of the inference. Our proposed techniques are able to increase Bayesian inference performance by 5.8× on average over the naive assignment and execution of the workloads.

***Keywords*-machine learning; bayesian inference; workload characterization**

I. Introduction

Recent advances in deep learning have captivated the scientific community. Systems based on neural network models have defeated world champion Go players [1], surpassed humans at image classification tasks [2], and advanced the state of the art for speech recognition [3]. However, neural networks are not the end-all solution and in many cases are not applicable. Deep learning requires massive datasets for training, is prone to overfitting, and is not conducive to reasoning about causality.

Bayesian inference is another branch of machine learning technique that complements deep learning in many ways. Bayesian inference thrives when data is limited, and its models are more interpretable, making it possible to understand how and why decisions are made. These benefits stem from the ability to combine prior knowledge with new observations.

Bayesian inference is a popular topic among machine learning researchers. Among top machine learning conferences (NIPS, ICML, and KDD), over 200 Bayesian inference papers have been published each year since 2014 and the number is steadily increasing. Notable milestones for Bayesian inference include industrial applications [4], [5], [6], Bayesian program learning for generalization of visual concepts from as few as one example [7], and the development of an intuitive physics engine that aids physical scene understanding [8], [9].

As with deep learning, Bayesian inference models are computationally demanding, requiring attention from the hardware and systems community to improve performance and facilitate innovation. To enable systems research in Bayesian inference and to understand the architectural implications of these models, we present *BayesSuite*: a collection of seminal, representative Bayesian inference workloads. BayesSuite draws from rich application domains (ranging from economics to biology) in which Bayesian inference has been demonstrated to excel. We rigorously characterize BayesSuite on general-purpose processors found in contemporary datacenter servers. In doing so, we provide an academic understanding of the computational characteristics of a wide range of Bayesian inference workloads, including performance bottlenecks that are amenable to optimization.

Our analysis leads to two major conclusions. First, while Bayesian inference workloads show no obvious architectural bottlenecks on single-core machines, we find that variations in the Bayesian models reveal higher sensitivity to server architecture on multicore systems. Specifically, the performance of the workloads with complex probability distribution between the observed data and the underlying features causes contention in the last-level cache (LLC). The workloads with less complicated models result in smaller working set sizes and thus tend to be more compute-bound. Leveraging these observations, we developed a scheduling and optimization mechanism that analyzes Bayesian inference jobs and automatically identifies the server configuration most likely to maximize its performance.

Second, we find that the workloads entail substantial redundant computation in the form of sampling iterations.

978-1-7281-0746-2/19/$31.00 ©2019 IEEE
DOI 10.1109/ISPASS.2019.00031

Thus, eliding unnecessary computation through convergence detection can improve performance without reducing accuracy. We developed an intelligent mechanism that dynamically determines when to terminate a job to reduce latency and save energy without jeopardizing model accuracy.

As Bayesian inference continues to transition from academic to commercial use, bloated, proof-of-concept models need to be refined and tuned into industrial-grade code capable of performing at scale. We envision that our characterizations and proposed techniques can facilitate the deployment of Bayesian inference as a generic web service, similar to the "deep learning as a service" paradigm provided by Google Cloud Machine Learning Engine, Microsoft Azure Machine Learning, and Apache MXNet on Amazon Web Services.

This paper makes the following contributions:

- BayesSuite: a benchmark collection of state-of-the-art Bayesian inference models for research on performance optimization by computer architects and system designers.
- A detailed characterization of the BayesSuite workloads on datacenter server architectures. We identify key bottlenecks to performance scaling and present insights on performance and energy trade-offs.
- Mechanisms that automatically provision hardware resources for specific Bayesian inference jobs in order to optimize performance and power efficiency. Across BayesSuite, we achieve an average speedup of 5.8×.

II. Bayesian Inference

In this section, we briefly go over the concept of Bayesian modeling and inference to familiarize readers with the algorithms. This section serves only as a primer; a thorough treatment is beyond the scope of this paper. For more details on Bayesian inference, readers can refer to [10].

A. Bayesian Inference

Probabilistic models describe the data that could be observed from a system and use probability theory to describe the uncertainty or noise associated with the model. In supervised learning, such as deep learning, models are trained using labeled data and the uncertainty or noise is not explicitly modeled. In a situation where we do not have enough labeled data or when we are trying to create an uncertain relationship between observed data of different types, we can construct a Bayesian model and perform inference to learn what we want given some data. This process can be done by using Bayes' theorem, shown as

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}, \tag{1}$$

where D is the evidence, or the observed data and θ is the hypothesis whose probability is updated on the new data D. We refer to $P(\theta|D)$ as the posterior probability, the probability of a hypothesis given the observed evidence. $P(D|\theta)$ is the probability of observing D given θ and is called the likelihood. $P(\theta)$ and $P(D)$ are referred to as the prior probability and marginal likelihood, respectively. Because $P(D)$ does not depend on θ, the posterior probability of the hypothesis given evidence is proportional to its likelihood and prior probability.

Bayesian inference uses Equation (1) to find the posterior distribution. Analytically computing the conditional probability distribution over variables of interest becomes intractable as the number of variables increases and the complexity of the model grows. Our work focuses on large and complex models for which exact inference is impractical. Instead we use approximate inference that is tractable and still produces satisfactory results.

B. Inference Algorithm

In this section, we illustrate the Bayesian inference algorithm for computing the posterior distribution. The workloads in this paper have different models and data, but apply the same inference algorithm. Common approximate Bayesian inference algorithms include sampling and optimization techniques. This paper focuses on one of the sampling methods; a variant of the Hamiltonian Monte Carlo algorithm (HMC) [11], nicknamed the No-U-Turn Sampler (NUTS) [12]. NUTS auto-tunes the Hamilton parameters including the step size and number of steps. It is implemented in the Stan [10] probabilistic programming framework, the framework used in this paper. We describe the more intuitive Metropolis-Hastings algorithm to illustrate the important computational characteristics, which are shared with NUTS.

Algorithm Assume we have a model θ and observed data D. The θ can be replaced with an arbitrary user-defined model. The likelihood of the data given model θ, $P(D|\theta)$, and the prior probability of model θ, $P(\theta)$, are known. The goal is to estimate the posterior probability $P(\theta|D)$. Algorithm 1 shows a naive Metropolis-Hastings algorithm with multiple Markov chains doing sampling. A Markov chain consists of a sequence of samples, and the current sample depends on the previous one. q determines the probability of new sample θ' given previous sample $\theta(t-1)$. u is a random number drawn from a uniform distribution. In line 4 of Algorithm 1, Metropolis-Hastings draws samples from arbitrary probability distribution, often called proposal density, which results in a random-walk behavior. NUTS explores high-dimensional space by building a set of likely candidate points recursively, which eliminates random-walk behavior exhibited by the Metropolis-Hastings algorithm. In the NUTS implementation of Stan, the acceptance rate in line 5 is found by averaging acceptance probability across the entire candidate set. While each iteration of NUTS tends to be more computationally expensive, it explores the target distribution much more efficiently, resulting in faster convergence.

Algorithm 1 Metropolis-Hastings Algorithm. An example of sampling posterior $P(\theta|D)$, given observed data D, prior $P(\theta)$, and likelihood $P(D|\theta)$.

```
1: for chain from 1 to nchain do
2:     θ(0) ~ init
3:     for t from 1 to n do
4:         θ' ~ q(θ|θ(t−1))
5:         r = P(θ')P(D|θ') / P(θ(t−1))P(D|θ(t−1))
6:         u ~ uniform(0,1)
7:         if u < min{r, 1} then
8:             θ(t) = θ'
9:             P(D|θ(t)) = P(D|θ')
10:        else
11:            θ(t) = θ(t−1)
12:        end if
13:    end for
14:    Collect Samples
15: end for
```

Computation Algorithm 1 has an outer loop over the Markov chains and an inner loop that does sampling. Each new sample is kept or discarded based on the Metropolis-Hastings rule in lines 7–12. Because the current sample depends on the previous sample, the inner loop is sequential.

There are two key computation characteristics. The first is the *sampling* in line 4, which is defined by specific models, and the computation of *acceptance rate* in line 5, which involves computations such as likelihood computation iterating over all observed data. The second characteristic is the *loop* structure. The outer loop drives the Markov chains. Its iterations are independent, so the chains can run on different cores in parallel.

Other Algorithms Other popular practical algorithms include variational inference, which approximates probability densities though optimization. However, these techniques do not output posterior distributions as sampling algorithms do, and do not have guarantees to be asymptotically exact. They are not as robust as sampling algorithms and need carefully crafted models and data types to avoid numerical issues, which are sometimes unavoidable. We selected NUTS as it has been widely used in the Stan community, which gives us access to a rich collection of workloads to study. We briefly discuss the performance of HMC together with NUTS in Section IV-A.

III. BayesSuite: Bayesian Inference Workloads

In this section, we present BayesSuite[1]: a Bayesian inference benchmark suite with models and datasets representing real-world use cases. We study Bayesian inference workloads developed in Stan [10]. Each BayesSuite workload consists of a model and data, both of which are fed to the NUTS inference algorithm implemented in the framework.

Workloads were selected to cover key application domains in which Bayesian inference has excelled, leveraging important models and real datasets, and to have diverse execution behaviors. The workloads are selected from StanCon 2017 [29], StanCon 2018 [30], Knitr [25], and BPA [27]. Below we briefly introduce the workloads. Table I summarizes the models, applications, sources, and data for each workload. We also list the corresponding publications of the workloads. If the work has not been published, we list the corresponding source.

12cities: Shows that lowering speed limits saves pedestrian lives. Uses Poisson regression on data for 12 cities obtained from FARS [14], the Fatality Analysis Reporting System maintained by the National Highway Traffic Safety Administration.

ad: Quantifies the effectiveness of various advertising channels for the movie industry. Survey data combining demographics with chosen advertising channels are fitted into a logistic regression model.

ode: Builds ordinary differential equations (ODE) to quantitatively study how drug compounds circulate in and affect the patient's body. Margossian et al. applied the Friberg-Karlsson semi-mechanistic model to this nonlinear system [16].

memory: Models the human mechanism for memory retrieval in sentence comprehension [18]. Data was collected via experiments measuring recall accuracy and latency after participants were asked to memorize words or numbers of letters. This workload implements a direct access model based on a content-addressable memory system [31].

votes: Forecasts presidential elections in all states of the US from 2020 to 2028 using historical election data from 1976 to 2016. A Gaussian process model is applied to the observed votes. Gaussian processes are very good at modeling observations over a continuous domain such as space or time.

tickets: Investigates whether the New York Police Department manages officers with productivity targets, which contravenes New York state law. A generative model is proposed to describe how officers write traffic tickets. The trained model indicates that the officers alter their ticket writing substantially to match departmental targets.

disease: Models the progression of Alzheimer's disease, which is described by biomarkers and eventual loss of memory and decision-making functions. It is useful for clinical and biological purposes to understand the order of biomarkers' deterioration and their distributions for various stages of the disease. This model uses I-splines to model the monotonically increasing progression and is fitted with real patient data.

racial: Tests for racial bias in vehicle searches by police. Simoiu et al. developed a new statistical test of discrimination, the threshold test [23]. The test uses a hierarchical latent

[1]We will publish the source code of BayesSuite.

Table I: A summary of BayesSuite workloads.

Name	*Model*	*Application*	*Reference*	*Data*
12cities	Poisson Regression	Does lowering speed limits save pedestrian lives?	[13]	FARS [14]
ad	Logistic Regression	Advertising attribution in the movie industry	StanCon 2017	[15]
ode	Friberg-Karlsson Semi-Mechanistic	Solving ordinary differential equations of non-linear systems	[16]	[17]
memory	Hierarchical Bayesian	Modeling memory retrieval in sentence comprehension	[18]	[18]
votes	Hierarchical Gaussian Processes	Forecasting presidential votes	StanCon 2017	historical (1976-2016) presidential votes
tickets	Logistic Regression	Do police officers alter the ticket writing to match departmental targets?	[19]	[20]
disease	Logistic Regression	Measuring the continually worsening progression of Alzheimer's disease	[21]	[22]
racial	Hierarchical Bayesian	Testing for racial bias in vehicle searches by police	[23]	[23], [24]
butterfly	Hierarchical Bayesian	Estimating butterfly species richness and accumulation	Knitr [25]	[26]
survival	Cormack-Jolly-Seber	Estimating animal survival probabilities	BPA [27]	[28]

Bayesian model, and is applied to a dataset of 4.5 million police stops in North Carolina. It is found that when searching minorities, officers apply lower standards of evidence than when searching whites.

butterfly: Uses a hierarchical Bayesian model developed by Dorazio et al. to estimate butterfly species richness and accumulation [26]. Statistical estimation is necessary due to the difficulty of collecting data in grasslands with small habitat fragments in south-central Sweden, where the study was conducted. Predictions show that sample locations could be reduced by half without affecting the estimation.

survival: Cormack-Jolly-Seber (CJS) models estimate animal survival from capture-recapture data collected by capturing, tagging, and releasing animals in the population being studied. Feeding data on recapture rates into the CJS model allows survival rate to be estimated.

IV. Performance Analysis

In this section, to understand the execution behavior of Bayesian inference on general-purpose microprocessors, we conduct system- and architecture-level analysis of BayesSuite. Through single core performance analysis, we show that most BayesSuite workloads have higher computational efficiency than conventional sequential CPU benchmarks like SPEC CPU2006, except for some outliers suggesting possible computational bottlenecks (Section IV-A). By studying multicore performance, we further find that the bottleneck for BayesSuite multicore scaling is last-level cache (LLC) size (Section IV-B).

A. Single Core Performance

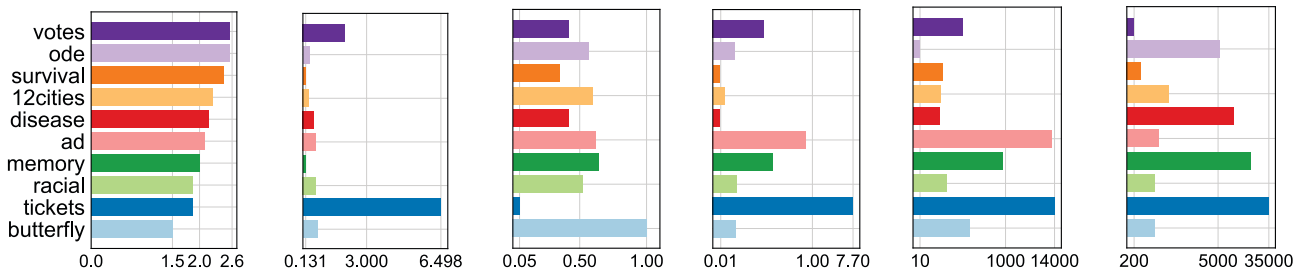
In this subsection, we study the single core performance of BayesSuite. The experiments are conducted on a modern CPU, the Intel® Core™ i7-6700K, which has four physical cores, 4.2 GHz single-thread frequency, an 8 MB last-level cache, and 34 GB/s max memory bandwidth. The performance characteristics are collected with performance counters. The sampling does not affect the results because the workloads behave consistently throughout the execution. We use RStan, the R interface of Stan, with version 2.17.3 [32] and the R version is 3.4.4.

Workload Diversity The varying complexities of inference models and datasets among the workloads lead to variations in runtime behavior. Figure 1 shows several key dynamic characteristics of the workloads, including instructions per cycle (IPC), instruction cache misses per thousand instructions (MPKI), branch misprediction MPKI, last-level cache (LLC) MPKI, average memory bandwidth, and total execution time.

Total execution time and average bandwidth vary significantly across benchmarks, and the workloads also differ at the architecture level. For instance, the IPC of *votes* is more than 1.7× that of *butterfly*. Similar variation in the other microarchitecture characteristics demonstrate BayesSuite's diversity.

Benign Architectural Behavior Although Bayesian inference workloads differ in absolute behavior, they tend to employ CPU microarchitecture efficiently, as suggested by the IPC values in Figure 1a. Instruction-level parallelism is greater in Bayesian inference workloads than in traditional sequential applications like event-driven web servers [33] and most SPEC CPU2006 benchmarks. Higher IPC results from efficient instruction supply and data feeding. Specifically, for most workloads the instruction cache MPKI (Figure 1b) and branch misprediction MPKI (Figure 1c) are several times lower than for SPEC CPU2006 [34] and datacenter workloads [35]. Similarly, the LLC miss rate of Bayesian inference workloads is also insignificant, corroborated by the low bandwidth requirement (hundreds of MB/s for most of the workloads shown in Figure 1e, compared to tens of GB/s reached in server systems). The low data bandwidth indicates that the working set of Bayesian inference workloads can largely fit in on-chip memory.

Computation Bottlenecks Although the average performance of BayesSuite is benign, there are some special cases. The i-cache and LLC MPKI of *tickets* is higher than that of other BayesSuite workloads. By studying multicore behaviors



(a) IPC (b) I-Cache MPKI (c) Branch MPKI (d) LLC MPKI (e) Bandwidth (MB/s) (f) Total Execution Time (s)

Figure 1: Runtime statistics of BayesSuite.

in Section IV-B, we will show more workloads suffering from architectural bottlenecks in details. The execution times of *tickets*, *memory*, *disease* and *ode* are much higher, which is not intrinsic but an algorithmic artifact. We will examine it in Section VI.

Performance of HMC The single-core performance characteristics of HMC are very similar to NUTS. As a result we do not include the HMC data and focus on NUTS in the rest of this paper. The IPC of HMC ranges from 1.5 to 2.7. The LLC MPKI of *tickets* is 8.3, and that of the other workloads is below 1 MPKI. The memory bandwidths of *ad* and *tickets* are over 12 GB/s and that of *memory* is close to 10 GB/s. The memory bandwidths of other workloads are all below 100 MB/s.

Architectural Implication The benign architectural behavior of BayesSuite workloads on a modern CPU, together with the CPU's general-purpose programmability, suggests that the CPU is a good execution target for Bayesian inference. The observed cache bottleneck is the exception, which we will analyze in the next section. We will discuss GPUs and specialized hardware accelerators in Section VII.

B. Architectural Bottleneck

In this subsection, we analyze the performance bottlenecks of BayesSuite. We find that the multicore scalability of BayesSuite is strongly correlated with last-level cache (LLC) size.

Parallelism Opportunity Sampling algorithms are inherently parallel in that the computations of different chains are independent. The `for` loop at line 1 of Algorithm 1 can be completely parallelized, providing opportunities for performance improvement using multiple cores. We sweep the number of CPU cores used while keeping 4 Markov chains as suggested in [36], and iterations as defined in the original applications.

Performance Analysis As shown in the previous section, branch misses are minimal, the i-cache is private to each core, and memory bandwidth correlates to LLC miss rates. Therefore we focus on the LLC miss rate in this section and show the IPC, LLC MPKI, and speedup in Figure 2. The

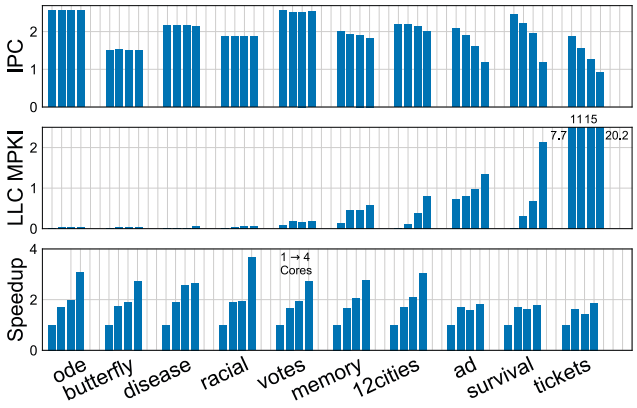


Figure 2: The IPC, LLC miss rates, and speedups of running the workloads on from 1 core to 4 cores of a Skylake processor. The plot shows that workloads such as *ad*, *survival* and *tickets* have increasingly frequent LLC misses and lower IPC. Therefore their speedup saturates at two cores.

workloads are sorted by the LLC MPKI of 4 cores. We use speedup and IPC as performance metrics.

Speedup is typically the most important performance metric for users. Across BayesSuite, the speedup using four cores is less than 4 because the execution times of the 4 chains are not equal and the 4-core execution time depends on the slowest chain, which will be explained in Section VI. We observe that the performance scaling is constrained by LLC misses. This is because when using one core, the four chains are executed sequentially and only one chain needs to fit into the LLC at a time. On the other hand, when using four cores with each core executing one chain, the global working set becomes 4× larger and sometimes does not fit in the LLC. This causes more frequent accesses to off-chip memory, thereby limiting performance scalability.

In Figure 2, when 4-core LLC MPKI is larger than 1, the speedup does not scale linearly with the number of cores, as with *ad*, *survival* and *tickets*, whose maximum speedups are less than 2. *tickets* has especially frequent LLC misses, up to 7.7 MPKI for 1 core and 20 MPKI for 4 cores. The high LLC miss rate leads to up to 25 GB/s memory bandwidth

for the three workloads, which is not shown here.

The speedup is affected not only by LLC miss rates but also by the fact that multicore latency is constrained by the slowest chain. Thus, we also compare IPC values, which helps to see how the LLC affects the efficiency of the architecture. As the number of cores in use increases, workloads such as *memory*, *12cities*, *ad*, *survival*, and *tickets* have lower IPC and more LLC misses. Increased working set size leads to more frequent LLC misses because every chain fetches data independently. The resulting performance degradation reduces IPC.

Architectural Implication LLC size is the major architectural bottleneck for BayesSuite workloads. Therefore, distinguishing workloads with large LLC needs before execution is valuable for computing resource management.

V. Bottleneck Resolution

In the previous section, we showed that BayesSuite workloads are constrained by LLC size, making it important to identify workloads with high LLC needs. To avoid the bottleneck, we first demonstrate our LLC miss prediction, using static indicators extracted from the model and data (Section V-A). We then show that prediction can facilitate a speedup of 1.16× through scheduling of BayesSuite workloads on appropriate platforms (Section V-B).

A. Last Level Cache Miss Prediction

We find **4-core LLC miss rates** can be predicted using a static feature, the *modeled data* size. Modeled data are the observed data required for finding a likelihood distribution. More specifically, modeled data are used to compute the acceptance rate in line 5 of Algorithm 1. A larger modeled data size translates to more computation and possibly a larger working set size. Note that the exact sizes of modeled data (on the order of KBs) are not working set sizes (on the order of MBs), but are only proportional to them, because there are more computations and intermediate variables in the inference algorithm, such as the likelihood and the Hamiltonian computation, and the automatic tuning in NUTS.

Figure 3 plots 4-core LLC miss rates against corresponding modeled data sizes. Points with labels suffixed -h and -q are for runs using half and a quarter of the original modeled data, respectively. We find that modeled data sizes are positively correlated with the 4-core LLC miss rates. Particularly for workloads with LLC MPKI larger than 1, modeled data size accurately predicts LLC miss rate.

For the workloads with LLC MPKI less than 1, the correlation is weaker, so the points with y-axis less than 1 do not form a straight line. That is because when the LLC miss rate is low, it is more affected by factors such as the memory prefetcher, the design of the LLC, including its size and associativity, the structure of the cache hierarchy, and the replacement policy.

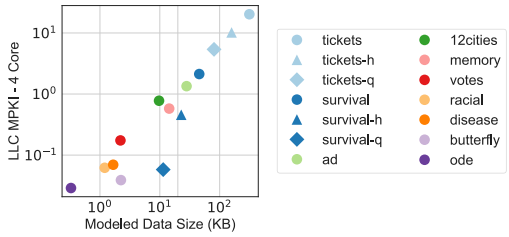


Figure 3: LLC miss rate prediction. For workloads with LLC miss rates larger than 1 MPKI, modeled data size predicts miss rate accurately. Points with labels suffixed -h and -q are for runs using half and a quarter of the original modeled data, respectively.

Architectural Implications Based on Figure 3, workloads with larger than 1 LLC MPKI including *tickets*, *survival*, and *ad* can be identified and predicted by setting a proper threshold for modeled data size. Resource management mechanisms can use the prediction to make better use of available computing resources. The threshold can be adjusted accordingly when applied to other machines.

B. Evaluation

In this section, we show that choosing proper platforms based on LLC miss prediction achieves 1.16× speedup in BayesSuite compared to using one platform alone.

1) Experimental Setup: We use two contemporary Intel CPU platforms in our evaluation: Broadwell (E5-2697A v4), and Skylake (i7-6700K), each with distinct specifications. We summarize the specifications in Table II, including microarchitecture, technology, peak frequency, physical core count, LLC size, memory bandwidth, and thermal design power (TDP).

We use the Broadwell server as our baseline because it was launched in 2016, later than the Skylake machine. The Broadwell processor has a large LLC size (40 MB) with only modest peak CPU frequency (3.6 GHz). In contrast, the Skylake processor has a high CPU frequency but small LLC size. We show that they naturally complement each other for BayesSuite workloads.

2) Performance Comparison: According to the models presented in Section V-A, the LLC-bound workloads are *ad*, *survival*, and *tickets*. In order to optimize performance, we choose to run them on Broadwell for its large LLC and other workloads on Skylake. We use Broadwell as the baseline and we achieve 1.16× speedup by adding a Skylake machine.

In Figure 4, we compare the speedup over Broadwell, IPC, and LLC MPKI of the platforms running with 4 cores. Speedup shows the end-to-end performance. IPC shows the throughput and performance regardless of frequency. LLC miss rate is used to explain speedup and IPC differences.

Skylake outperforms Broadwell on all workloads other than *ad*, *survival*, and *tickets*. Broadwell outperforms Skylake in

Table II: A summary of experiment platforms.

Codename	Processor #	Microarch	Tech (nm)	Turbo Freq (GHz)	Cores	LLC (MB)	Bandwidth (GB/s)	TDP (W)
Skylake	i7-6700K	Skylake	14	4.2	4	8	34.1	91
Broadwell	E5-2697A v4	Haswell	14	3.6	16	40	78.8	145

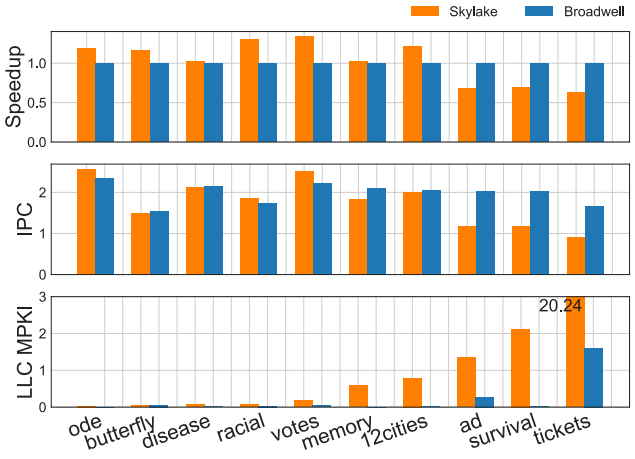


Figure 4: Performance comparison of the platforms.

speedup and IPC on those three workloads because its larger (40 MB) LLC leads to lower LLC miss rates. The IPC of *memory* and *12cities* is higher on Broadwell, also due to the much lower LLC miss rates, but Skylake's high frequency gives it a slightly better overall speedup than Broadwell.

Architecture Implication Following predictions by models in Section V, we run *ad*, *survival*, and *tickets* on Broadwell and other workloads on Skylake. This yields an average speedup of 1.16×. LLC size and frequency are the key factors determining the performance of BayesSuite workloads.

VI. Algorithm Convergence

Previous sections analyze the performance and architectural bottlenecks of BayesSuite. In this section, we study algorithmic aspects, the convergence and result quality of BayesSuite benchmarks. The number of sampling iterations (line 3 in Algorithm 1) is selected by users, and we find that all BayesSuite workloads have redundant iterations. We propose runtime convergence detection for users who want quick inference results with minimal overhead (Section VI-A). We then show that with convergence detection, BayesSuite workloads reach better design points and save 70% energy, on average. Overall, we speed up BayesSuite by 5.8× with convergence detection and LLC miss prediction (Section VI-B).

A. Convergence Study

Convergence detection is closely related to the number of chains used. Multiple chains prevent converging to local optima, and complex models prefer more chains. Convergence detection is based on the Gelman-Rubin diagnostic ($\hat{R}$) [37], which quantifies sample variations within and between chains. A smaller $\hat{R}$ indicates more consistent samples, and when $\hat{R}$ reaches 1, chains have converged completely. As suggested by Brooks et al. [36], we typically use 4 chains. Because several iterations are needed to warm up the chains, we only use the second half of the samples for inferring the posterior distribution [36]. A value of $\hat{R}$ less than 1.1 is taken as indicating convergence [36].

We study the convergence process and confirm that when using multiple chains, once $\hat{R}$ is less than 1.1, the results (posterior distributions) have good quality. To judge quality, we estimate the ground truth by running each benchmark with twice as many iterations as were initially specified by the model developer. To evaluate the intermediate results, we compute the KL divergence (a measure of how much one distribution diverges from another [38]) between intermediate results and the ground truth. A smaller KL divergence indicates that the result is closer to the ground truth.

We conduct a convergence study for BayesSuite and find that on average, the workloads have over 70% excess iterations. As an example, we show the convergence of *12cities* in Figure 5. The blue line is $\hat{R}$ and the green line shows KL divergence. With more iterations, the KL divergence decreases monotonically, showing that the results are getting closer to the ground truth. The trace of $\hat{R}$ fluctuates because the four independent chains are exploring different regions of the space. When they are sampling from the same region, $\hat{R}$ gets small; when one chain happens to jump out of that region, $\hat{R}$ increases. At the 600th iteration (orange dots), $\hat{R}$ is less than 1.1 for the first time and the KL divergence is minimal, indicating that the results are close to the ground truth. The original number of iterations of *12cities* is 2000. We find it converges after 600 iterations, eliminating 70% of the sampling iterations as unnecessary.

Reducing excess iterations can increase performance, but the iteration reduction percentage does not directly translate to latency saving. That is because the time required to auto-tune Hamiltonian parameters in NUTS may vary depending on the position of the Markov chain. Within a single chain, the latency per iteration is smaller after the chain converges, and different chains have different latencies. When multiple chains run in parallel, the overall latency is constrained by the slowest chain. For example, for *12cities* with 4 chains of 2000 iterations, the latency ratio of the slowest to the fastest chain is 1.7. The latency of 2000 iterations is 865 seconds and that of 600 iterations is 401 seconds, thus the latency is reduced by 53%. For the same reason, we should

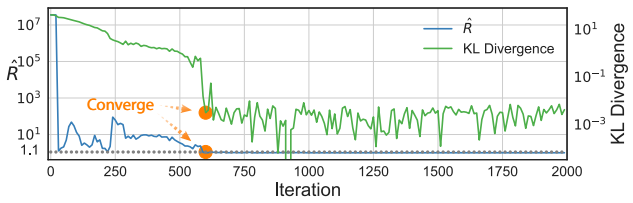


Figure 5: The convergence process of *12cities* in log scale. The blue line is $\hat{R}$, for detecting convergence. The green line is the KL divergence between the current result and ground truth. The orange dots mark the convergence point.

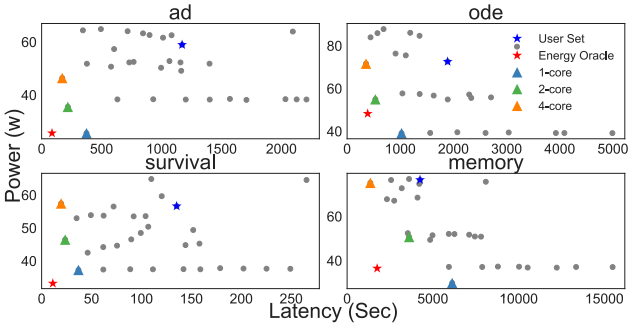


Figure 6: Design space exploration examples. *ad* and *survival* are LLC-bound. *ode* and *memory* are compute-bound. The design points in triangles, which are achievable with convergence detection, are much closer to the energy oracle (red star) than the original user setting (blue star).

expect the average latency savings to be less than the iteration reductions.

Runtime Convergence Detection Detecting convergence at runtime can be implemented by dynamically computing $\hat{R}$ in the framework, such as Stan in this case. Instead of executing a preset number of iterations, as in line 3 of Algorithm 1, the workload exits each iteration when it is determined to have converged. This detection is optional, for statisticians whose interests are in developing new models and would like to choose the number of iterations and test the model. But convergence detection can give quick results for those interested in using existing models with their own data.

Overhead Analysis We implement the computation of $\hat{R}$ in C++, based on the algorithm in [37]. We consider the worst case by taking the maximum number of iterations in BayesSuite (2000) and half of the samples for inference (i.e., 1000 data points of 4 chains). That takes 0.06 seconds on a single core of Skylake, which is minimal. In reality, optimizations can be applied to reduce the overhead.

Architectural Implication By our analysis, the overhead of convergence detection is minimal, and the savings are huge.

B. Design Space Exploration

We evaluate the convergence detection mechanism using design-space exploration (DSE) techniques and compare the optimization decision with other (suboptimal) design points in terms of latency and power savings.

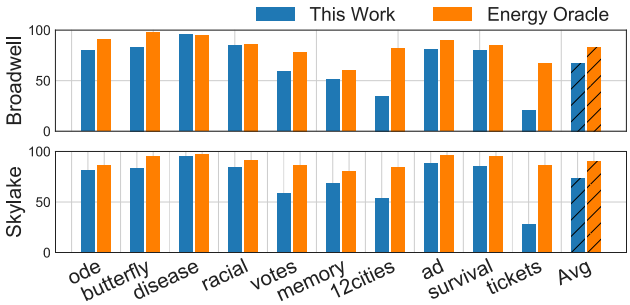


Figure 7: A summary of energy savings of our design points and the energy oracle.

Our DSE setup considers three major parameters: the number of CPU cores, the number of chains, and the number of iterations. We show the design space of 4 representative workloads on Skylake as a case study in Figure 6. The blue stars are original user settings. The triangle markers denote the design points that are achievable with convergence detection under 1, 2, and 4 cores. We refer to the design point that has the lowest energy consumption as the *energy oracle*, as denoted by red stars in the figure.

We find that the energy oracle design points always use only 1 or 2 chains and a small number of iterations while user-specific settings always use 4 chains with a much higher number of iterations. Although they have small KL divergence, without knowing the ground truth a priori, it is infeasible to use only 1 or 2 chains; hence, the oracle.

Energy Savings The triangles that we choose are much closer to the red stars than the original user settings. When applying this technique, a scheduler can be programmed to choose one of the triangles to optimize power or latency. In this section, we choose to use energy as the cost function, considering both latency and power.

We summarize our energy savings for 10 workloads on two platforms in Figure 7. The savings are in percentage relative to the original user settings. The average energy saving is 70% across 2 platforms and 10 workloads.

Architectural Implication BayesSuite workloads reach better design points with convergence detection.

C. Overall Speedup

We present the overall speedup resulting from the techniques in this paper: convergence detection in Section VI-A and selecting the best platform in Section V. In Figure 8 we show the overall speedups of BayesSuite over the baseline. *ad*, *survival*, and *tickets* are on Broadwell and the rest of the workloads are on Skylake. *ode* and *memory* achieve higher speedups than the energy oracle, which can be explained using Figure 6. *ode* and *memory* use 4 cores (orange triangles) on Skylake, and have lower latency than the red stars, the oracle points. The same explanation applies to *disease*.

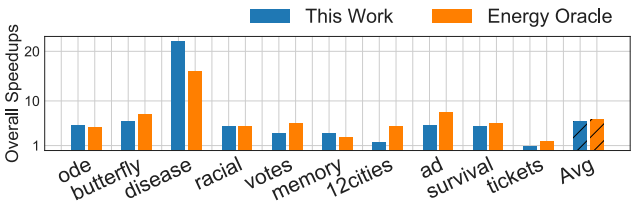


Figure 8: Overall speedup of the techniques proposed in this paper. Note that the oracle points are with respect to energy, not performance.

With the techniques proposed, the average speedup of BayesSuite is 5.8×, and the oracle average speedup is 6.2×, over the baseline with no convergence detection running on the Broadwell server.

VII. Implications for Future Acceleration

Intelligent scheduling on today's server processors can readily provide performance improvement for Bayesian inference jobs. Pushing the efficiency a step further requires us to apply hardware specialization. Previous work on hardware specialization only focuses on a specific type of model. In this section, based on the insights that we gained from analyzing and improving workload efficiency on CPUs, we examine opportunities to accelerate Bayesian inference using specialized hardware such as GPUs, FPGAs, and ASICs, in preparation for an accelerator-centric system to further speed up Bayesian inference workloads.

We first discuss the choice of different acceleration approaches. We argue that a programmable SIMD architecture augmented with special functional units is a good accelerator style that matches well with a wide range of Bayesian inference workloads (Section VII-A). We then discuss the memory system requirements on LLC and i-cache (Section VII-B).

A. Hardware Choice

The first and foremost question is which accelerator style, e.g., SIMD or CGRA, is a good fit for Bayesian inference workloads. We find that these workloads exhibit both coarse-grained and fine-grained parallelism.

Chain-Level Parallelism Both coarse-grained and fine-grained parallelism exist in sampling algorithms. Coarse-grained parallelism typically manifests at the chain level (line 1 in Algorithm 1), which can be captured by a multicore CPU as we discussed in Section IV-B. To better exploit the chain-level parallelism, the key is to address the LLC bottleneck inhibiting CPU core scaling shown in Figure 2.

Computation Parallelism Within a chain, there are opportunities for parallelism in the computation. For example, in the acceptance rate computation iterating through a series of observed data (line 5 in Algorithm 1), the computation of each observed data point can be conducted in parallel. At a finer grained level, BayesSuite workloads contain a diverse collection of vector and matrix operations beyond matrix multiplication, indicating the importance of architectural support for such operations. Therefore the workloads can benefit from the parallelism of SIMD-style hardware like GPUs.

Variable Sampling Parallelism The sampling of variables (line 4 in Algorithm 1) provides parallelism opportunities as well, which can benefit from SIMD hardware or specialized accelerators. When presenting the models as graphs, in which the model variables are nodes and variable dependencies are edges, the variables at the same layer can be sampled in parallel. Previous work on FPGAs and ASICs exploits the parallelism [39], [40]. With multiple sampling units on chip, the sampling latency can be shortened.

It is beneficial to implement the sampling units as accelerators. The current implementation of sample drawing needs cumulative distribution functions (CDFs) of corresponding distributions. Thus the implementation depends on individual distributions. We study the distributions in BayesSuite and find the most popular distributions are Gaussian and Cauchy. It is worth building accelerators for the most popular distributions. The CDFs use functions with values stored in lookup tables, such as the error function erf (Gaussian) and arctangent function $atan$ (Cauchy), which introduces overhead to the system and trades off precision for efficiency. Sampling accelerators can help to reduce system overheads by having their own scratchpad memory or private cache.

Parallelism in other Algorithms Finally, we note that different inference algorithms exhibit different opportunities for parallelism. For instance, sampling algorithms such as the one studied in this paper are general but sequential. Exact inference has more parallelism but the complexity is exponential. Some recent work combines sampling and exact inference to get the parallelism of exact inference and the linear complexity of sampling [41], [42], [43]. The general idea is to do exact inference with a subset of the data within the MCMC sampling iterations. Such algorithms, exposing ample parallelism, are promising to explore in future work.

Need for Programmability As we discussed, the workloads have very diverse models, requiring different combinations of matrix, vector, and scalar operations, as well as different preferences for distributions. Thus, to accelerate Bayesian inference workloads, we need to program the models.

B. Memory System

To reduce the overhead of transferring data between LLC and main memory, the LLC should be large enough to contain the whole working set. According to results in Figure 4, an LLC of 2 MB per core (8 MB / 4 cores on Skylake) is large enough for workloads other than *ad*, *survival*, and *tickets*. An LLC smaller than 10 MB per core (40 MB / 4 cores on Broadwell) is enough to hold *ad* and *survival*. Workloads like *tickets* need larger LLC. However, with larger datasets applied to Bayesian models, simply scaling up the LLC is not

the solution. Instead, the inference algorithm should be tuned to subsample the data such that the working set fits the LLC. Figure 3 can be used to estimate the proper sub-sampled data size.

In addition to LLC size, a 32 KB i-cache constrains the performance of *tickets*, as shown in Figure 1, and leads to high LLC miss rates in Figure 4. Thus the hardware needs i-caches larger than 32 KB to better serve workloads like *tickets*.

VIII. Related Work

In this section, we compare and summarize the previous work related to Bayesian models, inference algorithms, hardware advancement of Bayesian inference, probabilistic programming, and workload characterization.

Bayesian models In addition to the BayesSuite workloads, Bayesian inference has been applied to image classification [7], [44], semantics analysis [45], language learning [46], program synthesis [7], [47], [48], intuitive physics [9], [8], [49], [50], and structure learning [51], [52], [53]. A Bayesian approach that is sometimes called Bayesian neural networks (BNN) [54] is being applied to deep learning to learn weight distributions. These models are known to achieve better results by using optimization techniques, rather than more general and easy-to-use approaches like NUTS. It is important to note that models can have varying results, depending on the inference algorithm used.

Inference Algorithms Exact inference is often intractable as it has exponential complexity, while sampling is linear with regard to the number of samples [55]. This paper focuses on NUTS, a variant of Hamiltonian Monte Carlo that requires no hand-tuning of step size and number of steps [12]. It is a turnkey sampling method that can be used in a black-box fashion and has been adapted by Stan as the default inference engine. Other sampling algorithms include Gibbs sampler, Hamiltonian Monte Carlo, slice sampling, sequential Monte Carlo, and particle Markov chain Monte Carlo [11]. Variational inference is an optimization algorithm that tends to be fast but has no guarantee on convergence to global optima [56].

We discussed the combination of sampling and exact inference in Section VII-A [41], [42], [43]. Those hybrid algorithms have the potential to benefit from parallel hardware such as GPUs and we plan to explore them in future work.

Hardware Advancements Efforts have also been made to speed up Bayesian inference with specialized hardware. The BAMBI project[2] proposed hardware implementations of probabilistic computations [40], [57], [58]. Mansinghka et al. have built stochastic circuits and evaluated them with Markov Random Fields and the Dirichlet Process Mixture Model [55], [39]. Some of the acceleration work has been done as parallel implementations on GPUs [59], [60], [61] and scalable CPUs [62], [63]. Furthermore, there are FPGA and ASIC implementations accelerating BNNs [64] and Markov Random Fields on perceptual applications [65], [66]. [67] uses a novel device to implement the sampling procedure.

These projects primarily focus on speeding up a specific type of model or application and often require substantial effort for programming GPUs or designing the hardware. Our work is the first in the literature to introduce Bayesian inference to the architecture community as a key machine learning technique and to reveal computational bottlenecks across a suite of benchmarks on commodity datacenter CPUs.

Probabilistic Programming Some probabilistic programming frameworks focus mostly on language design and expressiveness [55], and some provide efficient sampling for a certain subset of models [68], [69], [70]. Infer.NET focuses on variational approximation [71]. TensorFlow Probability [72] and Edward [73] support distributed and parallel training on top of TensorFlow, and Pyro [74] is based on PyTorch. We choose Stan mainly because of its unmatched popularity in the science community.

Workload Characterization People develop benchmark suites to facilitate the advancement of architecture [75], [76], [77] or to introduce important workloads [78], [79], [80], as BayesSuite does. We use static workload features to estimate dynamic characteristics, similar to [81], and we are different by focusing on a key bottleneck of Bayesian inference. Therefore our static analysis has minimal overhead.

IX. Conclusion

The advances enabled by deep learning have overshadowed other aspects of the vast field of machine learning. Bayesian inference is a particularly important machine learning technique that complements deep learning in many domains. This paper introduces BayesSuite, a suite of Bayesian inference workloads to help bridge the gap between Bayesian inference researchers and computer architects. Through detailed characterization, we show that these workloads exhibit diverse behaviors that call for a variety of processor architectures. They also entail inherent redundancy that leads to execution inefficiency. We propose a scheduling mechanism and a computation elision technique to automatically speed up Bayesian inference workloads. Experiments and evaluations conducted on real systems show that we speed up BayesSuite workloads by 5.8× on average.

Acknowledgment

The authors would like to thank the anonymous reviewers for their thoughtful comments and constructive suggestions. The authors would like to thank Bob Adolf, Glenn Holloway, Svilen Kanev, Lifeng Nai, Margo Seltzer, and Cliff Young for their feedback. This work was supported in part by the Center for Applications Driving Architectures (ADA),

[2]Bottom-up Approaches to Machines dedicated to Bayesian Inference: www.bambi-fet.eu

one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA. The work was also partially supported by NSF grant # CCF-1438983 and Intel.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[3] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *International Conference on Machine Learning*, pp. 173–182, 2016.

[4] R. Analytics, "Google uses R to calculate ROI on advertising campaigns," 2014. http://blog.revolutionanalytics.com/2014/09/google-uses-r-to-calculate-roi-on-advertising-campaigns.html.

[5] B. Jitwasinkul, B. H. W. Hadikusumo, and A. Q. Memon, "A Bayesian belief network model of organizational factors for improving safe work behaviors in thai construction industry," *Safety science*, vol. 82, pp. 264–273, 2016.

[6] D. Ohlssen, "An industry perspective of the value of Bayesian methods," 2016. https://pharmacy.ucsf.edu/sites/pharmacy.ucsf.edu/files/ohlssen.pdf.

[7] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[8] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18327–18332, 2013.

[9] J. Hamrick, P. Battaglia, and J. B. Tenenbaum, "Internal physics models guide probabilistic judgments about object dynamics," in *Proceedings of the 33rd annual conference of the cognitive science society*, pp. 1545–1550, Cognitive Science Society Austin, TX, 2011.

[10] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian data analysis*, vol. 2. Chapman & Hall/CRC Boca Raton, FL, USA, 2014.

[11] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning," *Machine learning*, vol. 50, no. 1-2, pp. 5–43, 2003.

[12] M. D. Hoffman and A. Gelman, "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

[13] J. Auerbach, C. Eshleman, and R. Trangucci, "A hierarchical model to evaluate policies for reducing vehicle speed in major American cities," *arXiv preprint arXiv:1705.10876*, 2017.

[14] "Fatality analysis reporting system," https://www.nhtsa.gov/research-data/fatality-analysis-reporting-system-fars.

[15] N. Lei, Victor Sanders and A. Dawson, "Advertising attribution modeling in the movie industry," 2016. https://github.com/stan-dev/stancon_talks/blob/master/2017/Contributed-Talks/03_lei/ad_attribution.Rmd.

[16] C. Margossian and W. R. Gillespie, "Stan functions for Bayesian pharmacometric modeling," in *Journal of Pharmacokinetics and Pharmacodynamics*, vol. 43, pp. S52–S52, SPRINGER/PLENUM PUBLISHERS 233 SPRING ST, NEW YORK, NY 10013 USA, 2016.

[17] K. T. Baron, M. R. Gastonguay, A. Bioavailability, I. SS, and M. ADDL, "Simulation from ODE-based population PK/PD and systems pharmacology models in R with mrgsolve," *Omega*, vol. 2, p. 1x1, 2015.

[18] B. Nicenboim and S. Vasishth, "Models of retrieval in sentence comprehension: A computational evaluation using Bayesian hierarchical modeling," *arXiv preprint arXiv:1612.04174*, 2016.

[19] J. Auerbach, "Are New York City drivers more likely to get a ticket at the end of the month?," *Significance*, vol. 14, no. 4, pp. 20–25, 2017.

[20] "Parking or moving violation tickets in New York City between january 2014 and december 2015. https://raw.githubusercontent.com/jauerbach/Seventy-Seven-Precincts/master/data/tickets.csv.zip,"

[21] A. A. Pourzanjani, B. B. Bales, L. R. Petzold, and M. Harrington, "Flexible modeling of Alzheimer's disease progression with I-splines," 2018.

[22] C. R. Jack Jr, M. A. Bernstein, N. C. Fox, P. Thompson, G. Alexander, D. Harvey, B. Borowski, P. J. Britson, J. L. Whitwell, C. Ward, *et al.*, "The Alzheimer's disease neuroimaging initiative (ADNI): MRI methods," *Journal of Magnetic Resonance Imaging: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 27, no. 4, pp. 685–691, 2008.

[23] C. Simoiu, S. Corbett-Davies, S. Goel, *et al.*, "The problem of infra-marginality in outcome tests for discrimination," *The Annals of Applied Statistics*, vol. 11, no. 3, pp. 1193–1216, 2017.

[24] "A dataset of 4.5 million stops conducted by the 100 largest local police departments in North Carolina. https://github.com/stan-dev/stancon_talks/blob/master/2018/Contributed-Talks/11_simoiu/north_carolina.RData,"

[25] Y. Xie, *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2016. R package version 1.15.1.

[26] R. M. Dorazio, J. A. Royle, B. Söderström, and A. Glimskär, "Estimating species richness and accumulation by modeling species occurrence and detectability," *Ecology*, vol. 87, no. 4, pp. 842–854, 2006.

[27] M. Kéry and M. Schaub, *Bayesian population analysis using WinBUGS: a hierarchical perspective*. Academic Press, 2011.

[28] *Complete code and data files of book "Bayesian population analysis using WinBUGS"*. http://www.vogelwarte.ch/de/projekte/publikationen/bpa/complete-code-and-data-files-of-the-book.html.

[29] "Stancon 2017," 2017. http://mc-stan.org/events/stancon.

[30] "Stancon 2018," 2018. http://mc-stan.org/events/stancon2018/.

[31] B. McElree, "Sentence comprehension is mediated by content-addressable memory structures," *Journal of psycholinguistic research*, vol. 29, no. 2, pp. 111–123, 2000.

[32] *Modeling Language User's Guide and Reference Manual, Version 2.17.0*. http://mc-stan.org/users/documentation/.

[33] Y. Zhu, D. Richins, M. Halpern, and V. J. Reddi, "Microarchitectural implications of event-driven server-side web applications," in *Proceedings of International Symposium on Microarchitecture*, 2015.

[34] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," *Web Copy: http://www.glue.umd.edu/ajaleel/workload*, 2010.

[35] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 158–169, IEEE, 2015.

[36] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, *Handbook of Markov chain Monte Carlo*. CRC press, 2011.

[37] A. Gelman and D. B. Rubin, "Inference from iterative simulation using multiple sequences," *Statistical science*, pp. 457–472, 1992.

[38] J. R. Hershey and P. A. Olsen, "Approximating the Kullback Leibler divergence between Gaussian mixture models," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV–317, IEEE, 2007.

[39] E. M. Jonas, *Stochastic architectures for probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, 2014.

[40] M. Faix, R. Laurent, P. Bessière, E. Mazer, and J. Droulez, "Design of stochastic machines dedicated to approximate Bayesian inferences," *IEEE Transactions on Emerging Topics in Computing*, 2016.

[41] D. Maclaurin and R. P. Adams, "Firefly Monte Carlo: Exact MCMC with subsets of data," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[42] N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven, "Bayesian sampling using stochastic gradient thermostats," in *Advances in neural information processing systems*, pp. 3203–3211, 2014.

[43] M. Quiroz, R. Kohn, M. Villani, and M.-N. Tran, "Speeding up MCMC by efficient data subsampling," *Journal of the American Statistical Association*, no. just-accepted, pp. 1–35, 2018.

[44] R. Salakhutdinov, J. Tenenbaum, and A. Torralba, "One-shot learning with a hierarchical nonparametric Bayesian model," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 195–206, 2012.

[45] T. L. Griffiths, M. Steyvers, and J. B. Tenenbaum, "Topics in semantic representation," *Psychological review*, vol. 114, no. 2, p. 211, 2007.

[46] F. Xu and J. B. Tenenbaum, "Word learning as Bayesian inference," *Psychological review*, vol. 114, no. 2, p. 245, 2007.

[47] K. Ellis, A. Solar-Lezama, and J. Tenenbaum, "Unsupervised learning by program synthesis," in *Advances in Neural Information Processing Systems*, pp. 973–981, 2015.

[48] Y. Pu, Z. Miranda, A. Solar-Lezama, and L. Kaelbling, "Selecting representative examples for program synthesis," in *International Conference on Machine Learning*, pp. 4158–4167, 2018.

[49] C. Bates, P. Battaglia, I. Yildirim, and J. B. Tenenbaum, "Humans predict liquid dynamics using probabilistic simulation," in *CogSci*, 2015.

[50] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti, "Visual interaction networks: Learning a physics simulator from video," in *Advances in Neural Information Processing Systems*, pp. 4539–4547, 2017.

[51] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine learning*, vol. 20, no. 3, pp. 197–243, 1995.

[52] C. Kemp and J. B. Tenenbaum, "The discovery of structural form," *Proceedings of the National Academy of Sciences*, vol. 105, no. 31, pp. 10687–10692, 2008.

[53] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman, "How to grow a mind: Statistics, structure, and abstraction," *science*, vol. 331, no. 6022, pp. 1279–1285, 2011.

[54] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[55] V. K. Mansinghka, *Natively probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, 2009.

[56] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[57] R. Canillas, R. Laurent, M. Faix, D. Vaufreydaz, and E. Mazer, "Autonomous robot controller using bitwise Gibbs sampling," in *The 15th IEEE International Conference on Cognitive Informatics and Cognitive Computing. IEEE*, 2016.

[58] A. Coninx, P. Bessière, and J. Droulez, "Quick and energy-efficient Bayesian computing of binocular disparity using stochastic digital signals," *International Journal of Approximate Reasoning*, vol. 83, pp. 400–412, 2017.

[59] L. Zheng, O. Mengshoel, and J. Chong, "Belief propagation by message passing in junction trees: Computing each message faster using GPU parallelization," *arXiv preprint arXiv:1202.3777*, 2012.

[60] J. Ferreira, P. Lanillos, and J. Dias, "Fast exact Bayesian inference for high-dimensional models," in *Workshop on Unconventional computing for Bayesian inference (UCBI), IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[61] Y. Wang, W. Qian, S. Zhang, X. Liang, and B. Yuan, "A learning algorithm for Bayesian networks and its efficient implementation on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 17–30, 2016.

[62] V. K. Namasivayam and V. K. Prasanna, "Scalable parallel implementation of exact inference in Bayesian networks," in *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, vol. 1, pp. 8–pp, IEEE, 2006.

[63] Y. Xia, *Exploration of parallelism for probabilistic graphical models*. University of Southern California, 2010.

[64] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "VIBNN: Hardware acceleration of Bayesian neural networks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 476–488, ACM, 2018.

[65] G. G. Ko and R. A. Rutenbar, "A case study of machine learning hardware: Real-time source separation using Markov Random fields via sampling-based inference," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2477–2481, March 2017.

[66] G. G. Ko and R. A. Rutenbar, "Real-time and low-power streaming source separation using Markov Random field," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 14, pp. 17:1–17:22, May 2018.

[67] S. Wang, X. Zhang, Y. Li, R. Bashizade, S. Yang, C. Dwyer, and A. R. Lebeck, "Accelerating Markov random field inference using molecular optical Gibbs sampling units," in

Proceedings of the 43rd International Symposium on Computer Architecture, pp. 558–569, IEEE Press, 2016.

[68] A. Pfeffer, *Practical Probabilistic Programming*. Manning Publications Co., 2016.

[69] S. Hershey, J. Bernstein, B. Bradley, A. Schweitzer, N. Stein, T. Weber, and B. Vigoda, "Accelerating inference: towards a full language, compiler and hardware stack," *arXiv preprint arXiv:1212.2991*, 2012.

[70] L. Li and S. J. Russell, "The blog language reference," tech. rep., Technical Report UCB/EECS-2013-51, EECS Department, University of California, Berkeley, 2013.

[71] S. S. J. Wang and M. P. Wand, "Using Infer.NET for statistical analyses," *The American Statistician*, vol. 65, no. 2, pp. 115–126, 2011.

[72] "TensorFlow probability," 2018. https://www.tensorflow.org/probability/.

[73] D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei, "Edward: A library for probabilistic modeling, inference, and criticism," *arXiv preprint arXiv:1610.09787*, 2016.

[74] "Uber AI labs open sources Pyro, a deep probabilistic programming language," 2017. http://eng.uber.com/pyro/.

[75] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[76] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 110–119, IEEE, 2014.

[77] C. Bienia and K. Li, "PARSEC 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[78] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, "Fathom: reference workloads for modern deep learning methods," in *Workload Characterization (IISWC), 2016 IEEE International Symposium on*, pp. 1–10, IEEE, 2016.

[79] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "SD-VBS: The San Diego vision benchmark suite," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 55–64, IEEE, 2009.

[80] S. Thomas, C. Gohkale, E. Tanuwidjaja, T. Chong, D. Lau, S. Garcia, and M. B. Taylor, "CortexSuite: A synthetic brain benchmark suite," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 76–79, IEEE, 2014.

[81] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and QoS-aware cluster management," in *ACM SIGPLAN Notices*, vol. 49, pp. 127–144, ACM, 2014.