AcMC\textsuperscript{2}: Accelerated Markov Chain Monte Carlo for Probabilistic Models

ASPLOS 2019
Probabilistic Models: Core of Many AI Apps

- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
Probabilistic Models: Core of Many AI Apps

• Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
• Probabilistic programs: Encode probability models
Probabilistic Models: Core of Many AI Apps

- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
- Probabilistic programs: Encode probability models

Sensor Fusion in Self Driving Vehicles
Skill Matching in Online Gaming (TrueSkill 1&2 from Microsoft)
4G or 5G Communication Devices (Turbo/LDPC Codes)
Probabilistic Models: Core of Many AI Apps

- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
- Probabilistic programs: Encode probability models

Sensor Fusion in Self Driving Vehicles

Skill Matching in Online Gaming (TrueSkill 1&2 from Microsoft)

4G or 5G Communication Devices (Turbo/LDPC Codes)

- Inference: General solutions based on Markov Chain Monte Carlo
Probabilistic Models: Core of Many AI Apps

• Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
• Probabilistic programs: Encode probability models

- Sensor Fusion in Self Driving Vehicles
- Skill Matching in Online Gaming (TrueSkill 1&2 from Microsoft)
- 4G or 5G Communication Devices (Turbo/LDPC Codes)

- Inference: General solutions based on Markov Chain Monte Carlo
- Extremely compute intensive & real time constraints
Our Approach

Automatically generate efficient accelerator from high level description
  1. Abstraction: Domain specific languages
  2. Mapping abstractions to an architecture
Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages
2. Mapping abstractions to an architecture

Probabilistic Programming Languages
Our Approach

Automatically generate efficient accelerator from high level description
1. Abstraction: Domain specific languages
2. Mapping abstractions to an architecture

Probabilistic Programming Languages

Probabilistic Graphical Model based IR

BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)
Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages
2. Mapping abstractions to an architecture

Probabilistic Programming Languages

Probabilistic Graphical Model based IR

BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)
Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages
2. Mapping abstractions to an architecture

Probabilistic Programming Languages

Probabilistic Graphical Model based IR

AcMC$^2$

Inference Procedure
Markov Chain Monte Carlo

BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)
Our Approach

Automatically generate efficient accelerator from high level description
1. Abstraction: Domain specific languages
2. Mapping abstractions to an architecture

Probabilistic Programming Languages

Probabilistic Graphical Model based IR

AcMC²

Inference Procedure
Markov Chain Monte Carlo

Traditional synthesis flow

FPGAs

ASICs

Probabilistic Programming Languages

BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)
Our Approach

Automatically generate efficient accelerator from high-level description

1. Abstraction: Domain specific languages
2. Mapping abstractions to an architecture

Contributions:
1. Identify accelerable kernels
2. Opportunities for parallelism
3. Knobs for trading off accuracy and performance

Probabilistic Programming Languages

Probabilistic Graphical Model based IR

Inference Procedure
Markov Chain Monte Carlo

AcMC^2

BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)
AcMC$^2$ Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method
AcMC$^2$ Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method
AcMC² Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method

![Diagram of Probabilistic Program and Compiler Frontend]

- Bayesian model described in a DSL
- Template Spec
  - Uniform RNG
  - Distribution RNG
  - Arith. Operators
- Expert-generated hardware templates

Analyze and optimize PPL program

Compiler Frontend

AcMC² Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method

```
Probabilistic Program

Bayesian model described in a DSL

Compiler Frontend
- IR Generator
- Var. Elimination
- Opt. Pass 1
- ... Opt. Pass n

Analyze and optimize PPL program

Controller Builder
- Identify parallelism using statistical independences

Sampling Element Builder

Template Spec
- Uniform RNG
- Distribution RNG
- Arith. Operators

Expert-generated hardware templates
```
AcMC\(^2\) Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method

---

**Compiler Frontend**

- IR Generator
- Var. Elimination
- Opt. Pass 1
- Opt. Pass n

**Analyzer and optimize PPL program**

**Controller Builder**

- Identify parallelism using statistical independences

**Sampling Element Builder**

**Hardware Generation**

- Scheduler
- Interface Generator
- Chisel

**High-Level Constraints**

- FPGA Constraints
- Host Interface
- Memory Interface

**Template Spec**

- Uniform RNG
- Distribution RNG
- Arith. Operators

**Expert-generated hardware templates**

**Hardware Accelerator**

- Interface and on-chip resource constraints
**AcMC² Deep Dive**

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method

---

**Diagram Description**

1. **Template Spec**
   - Uniform RNG
   - Distribution RNG
   - Arith. Operators

2. **Compiler Frontend**
   - IR Generator
   - Var. Elimination
   - Opt. Pass 1
   - …
   - Opt. Pass n

3. **Analyzer and optimizer PPL program**

4. **Controller Builder**
   - Identify parallelism using statistical independences

5. **Sampling Element Builder**

6. **High-Level Constraints**
   - FPGA Constraints
   - Host Interface
   - Memory Interface

7. **Hardware Generation**
   - Scheduler
   - Interface Generator
   - Chisel
   - Synthesis Flow

---

**Legend**

- **AcMC²**: Acronym for Advanced Monte Carlo Compute²
- **MCMC**: Markov Chain Monte Carlo
- **PPL**: Probabilistic Programming Language
Example: GMM Clustering Data
Example: GMM Clustering Data

Data

Inference
Example: GMM Clustering Data

Data

Generative Bayesian Model

Inference

\[ \pi \sim \text{Dirichlet}(\alpha) \]
\[ \Lambda_k \sim \text{Wishart}(\Lambda_0, \nu) \]
\[ \mu_k \mid \Lambda_k \sim \text{Normal}(0, (\beta \Lambda_k)^{-1}) \]
\[ z_n \mid \pi \sim \text{Categorical}(\pi) \]
\[ x_n \mid z_n = k, \mu_k, \Lambda_k \sim \text{Normal}(\mu_k, \Lambda_k^{-1}) \]
Accelerable Kernels: Random Number Generators

Fundamental operation used in MCMC: sampling uniform random numbers
Accelerable Kernels: Random Number Generators

Fundamental operation used in MCMC: sampling uniform random numbers

- Expert optimized FPGA URNG
  - 4bit LFSR
  - 1 cycle latency
  - 1 op/cycle
Accelerable Kernels: Random Number Generators

Fundamental operation used in MCMC: sampling uniform random numbers

- Expert optimized FPGA URNG
  - 4bit LFSR
  - 1 cycle latency
  - 1 op/cycle

- Traditional RNGs
  - Cryptographically Secure
  - Rejection sampling: Stalls
Accelerable Kernels: Random Number Generators

Fundamental operation used in MCMC: sampling uniform random numbers

- Expert optimized FPGA URNG
  - 4bit LFSR
  - 1 cycle latency
  - 1 op/cycle

- Traditional RNGs
  - Cryptographically Secure
  - Rejection sampling: Stalls

Auto-generated: Discrete Distributions

Auto-generated: Continuous Distributions
Identifying Parallelism: Enter Markov Blankets

• How do we compose the samplers?
  • Program dataflow ordering is too conservative
Identifying Parallelism: Enter Markov Blankets

• How do we compose the samplers?
  • Program dataflow ordering is too conservative

• Use conditional dependencies to identify parallelism

• Set of nodes $B_{x_i}$ for a node $x_i$ such that:
  \[
  \Pr(x_i|B_{x_i}, A) = \Pr(x_i|B_{x_i})
  \]

• Markov blanket is the only knowledge needed to predict behavior node.
Identifying Parallelism: k-Colorings
Identifying Parallelism: k-Colorings

• Compute a k-coloring of the graphical model
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
Identifying Parallelism: k-Colorings

• Compute a k-coloring of the graphical model
• Sample all variables with same color in parallel (Conditionally Independent)
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
Identifying Parallelism: k-Colorings

• Compute a k-coloring of the graphical model
• Sample all variables with same color in parallel (Conditionally Independent)
• Synthesize state machines corresponding to coloring
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
- Synthesize state machines corresponding to coloring
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
- Synthesize state machines corresponding to coloring
- Equivalent to sequential
Parallelism in the GMM Clustering Example
Parallelism in the GMM Clustering Example

\[ Pr(\alpha|\pi) \]

\[ Pr(\pi|\alpha, z_n) \]

\[ Pr(z_n|\pi, x_n, \Lambda_k, \mu_k) \]

\[ Pr(x_n|z_n, \Lambda_k, \mu_k) \]

\[ Pr(\mu_k|\beta, \Lambda_k, x_n) \]

\[ Pr(\Lambda_k|\Lambda_0, \mu_k, x_n, z_n) \]
Parallelism in the GMM Clustering Example

- \( \operatorname{Pr}(\alpha|\pi) \)
- \( \operatorname{Pr}(\pi|\alpha, z_n) \)
- \( \operatorname{Pr}(z_n|\pi, x_n, \Lambda_k, \mu_k) \)
- \( \operatorname{Pr}(x_n|z_n, \Lambda_k, \mu_k) \)
- \( \operatorname{Pr}(\mu_k|\beta, \Lambda_k, x_n) \)
- \( \operatorname{Pr}(\Lambda_k|\Lambda_0, \mu_k, x_n, z_n) \)
Revisiting the GMM Clustering Example

\[
\begin{align*}
\text{Pr}(\alpha | \pi) \\
\text{Pr}(z_n | \pi, x_n, \Lambda_k, \mu_k) \\
\text{Pr}(\pi | \alpha, z_n) \\
\text{Pr}(\Lambda_k | \Lambda_0, \mu_k, x_n, z_n) \\
\text{Pr}(x_n | z_n, \Lambda_k, \mu_k) \\
\text{Pr}(\mu_k | \beta, \Lambda_k, x_n)
\end{align*}
\]
Revisiting the GMM Clustering Example

Pr(α|π)

Pr(z_n | π, x_n, Λ_k, μ_k)

Pr(π|α, z_n)

Pr(Λ_k | Λ_0, μ_k, x_n, z_n)

Pr(x_n | z_n, Λ_k, μ_k)

Pr(μ_k | β, Λ_k, x_n)

Batch Size = 1
Revisiting the GMM Clustering Example

Pr(\(\alpha|\pi\))

Pr(\(z_n|\pi, x_n, \Lambda_k, \mu_k\))

Pr(\(\pi|\alpha, z_n\))

Pr(\(\Lambda_k|\Lambda_0, \mu_k, x_n, z_n\))

Pr(\(x_n|z_n, \Lambda_k, \mu_k\))

Pr(\(\mu_k|\beta, \Lambda_k, x_n\))

Batch Size = 1

Acceptance Test

Store Query in Memory
Revisiting the GMM Clustering Example

More optimizations:
- Reuse across units (RNGs, Arithmetic units)
- Pipelining

Batch Size = 1
Other Details in the Paper

- Compositional MCMC
  - Gibbs, Metropolis Hastings, Hamiltonian

- Speculative Execution
  - Speculate past rejected samples

- Accuracy – Performance Tradeoffs
  - Bloom Filters; Precision

- Generating IBM-CAPI based DMA Engine
  - Little’s Law
Implementation

- **IBM POWER8**
- **FPGA**
  - CAPI attached Virtex 7 FPGA

**Sampling Element (SE) = RNGs + k-coloring controller**

- $N \times M$ sampling elements
- $N = 4$ (4 DRAM channels on FPGA board)
- $M = \text{max that can be fit on FPGA}$
Evaluation: AcMC² in Real World Models

**Epilepsy/Neuroscience**: Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

**Security**: Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]
Evaluation: AcMC² in Real World Models

**Epilepsy/Neurosicnece**: Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

**Security**: Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]
Evaluation: AcMC² in Real World Models

**Epilepsy/Neuroscience**: Identifying epilepsy affected brain regions \[\text{Varatharajah, NeurIPS17}\]

**Security**: Preempting advanced persistent threats using host/network IDSs \[\text{Cao, HOTSOS15}\]

---

**Diagram**

- **Embedded Medical Devices**
  - iEEG electrode
  - FFT
  - K-Means
  - AcMC²
  - Healthy electrodes?

- **Datacenter network monitoring tools**
Evaluation: AcMC$^2$ in Real World Models

**Epilepsy/Neuroscience:** Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

**Security:** Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]

---

**Embedded Medical Devices**

- iEEG electrode
- FFT
- K-Means
- Healthy electrodes?

**Datacenter network monitoring tools**

- Protocol Parser + Flow state
- 10GbE Data Stream
- AcMC$^2$
- Alerts
Evaluation: \( \text{AcMC}^2 \) in Real World Models

**Epilepsy/Neuroscience:** Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

**Security:** Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]

---

**Embedded Medical Devices**

- iEEG electrode
- FFT
- K-Means
- \( \text{AcMC}^2 \)
- Healthy electrodes?

**Datacenter network monitoring tools**

- Protocol Parser + Flow state
- 10GbE Data Stream
- \( \text{AcMC}^2 \)
- Alerts

Inference accounts for >90% of the latency
Results: Real World Case Studies

![Normalized Speedup Chart]

- **Epilepsy**
  - CPU: 1x
  - OpenCL-GPU: 6.2x
  - OpenCL-FPGA: 1.6x
  - AcMC²: 102.1x

- **Security**
  - CPU: 1x
  - OpenCL-GPU: 4.3x
  - OpenCL-FPGA: 4.7x
  - AcMC²: 48.4x

**Tool**
- CPU
- OpenCL-GPU
- OpenCL-FPGA
- AcMC²

**TDP**
- 190W
- 300W
Results: Real World Case Studies

[Graph showing normalized speedup and normalized power consumption for different tools and benchmarks.]
Results: Real World Case Studies

Significantly better results at much simpler code complexity

LoC AcMC$^2$ – 183 for C1 & 146 for C2

LoC OpenCL – 961 for C1 & 4861 for C2
Conclusion

• AcMC$^2$: A High Level Synthesis Compiler for Probabilistic Programs

• Code is open-source and available at
  https://gitlab.engr.Illinois.edu/DEPEND/AcMC2
Conclusion

• AcMC$^2$: A High Level Synthesis Compiler for Probabilistic Programs

• Code is open-source and available at https://gitlab.engr.Illinois.edu/DEPEND/AcMC2

• Looking forward
  • How does these models fit in the context of Deep Learning? – Bayesian Deep Learning