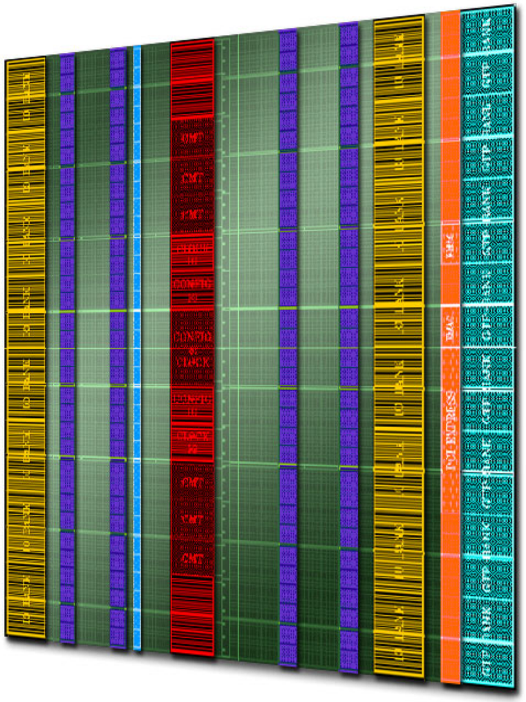
Imperial College London

Reconfigurable Computing for Bayesian inference

Christos Bouganis
Imperial College London



Joint work with: G. Mingas, S. Liu

Imperial College London

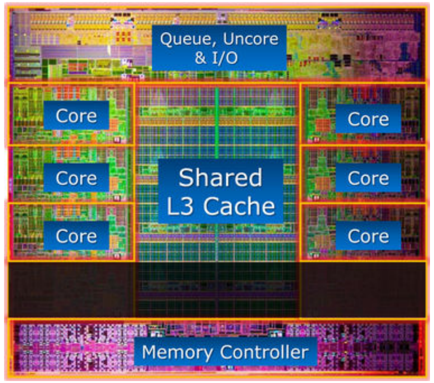
Group

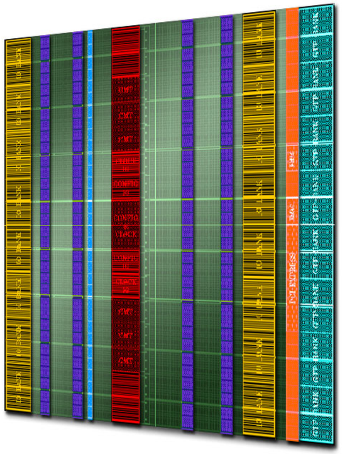
Machine Learning

Computer Vision

Realibility

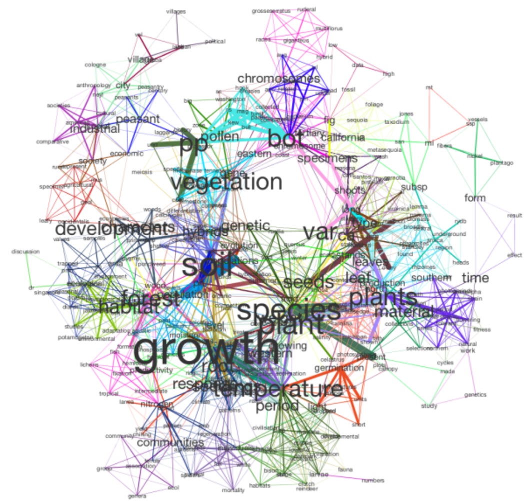
Bayesian Inference

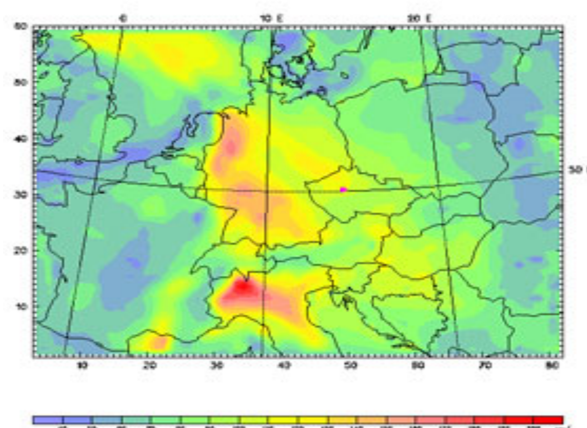




Motivation 1/2

- Bayesian models are becoming increasingly complex:
  - **Large-scale data, high dimensionalities**





Motivation 2/2

- **Stochastic inference methods:**
  - MCMC, SMC, Quasi-MC and many variants…

- **Runtimes can reach weeks, months or more**
  - Reasons:
    - Complex/"intractable" likelihoods
    - More data to process (Big Data)
    - Computationally demanding samplers

300K predictors, 4K responses

↓

MCMC needs 20 days to sample posterior

Imperial College London

Direction 1: Design "better" algorithms

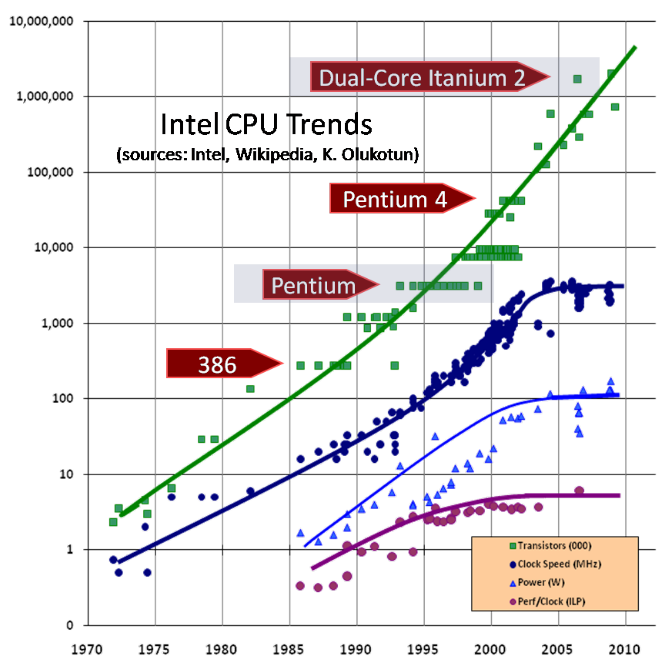
- Design more efficient samplers (Hamiltonian MC, Population-based MCMC, Adaptive MCMC, etc)
- Approximate methods (ABC, INLA, Variational Bayes)
- Data sub-sampling/blocking (Firefly MC, Consensus MC, Composite likelihoods)
- See previous talks for more details

Imperial College London

Direction 2: Buy better hardware

Do nothing. Just wait for the next generation processor

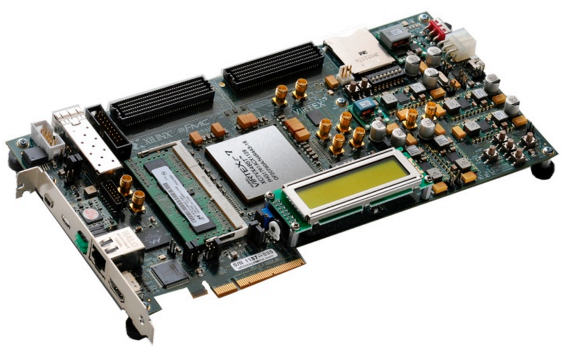
But no more...Power wall



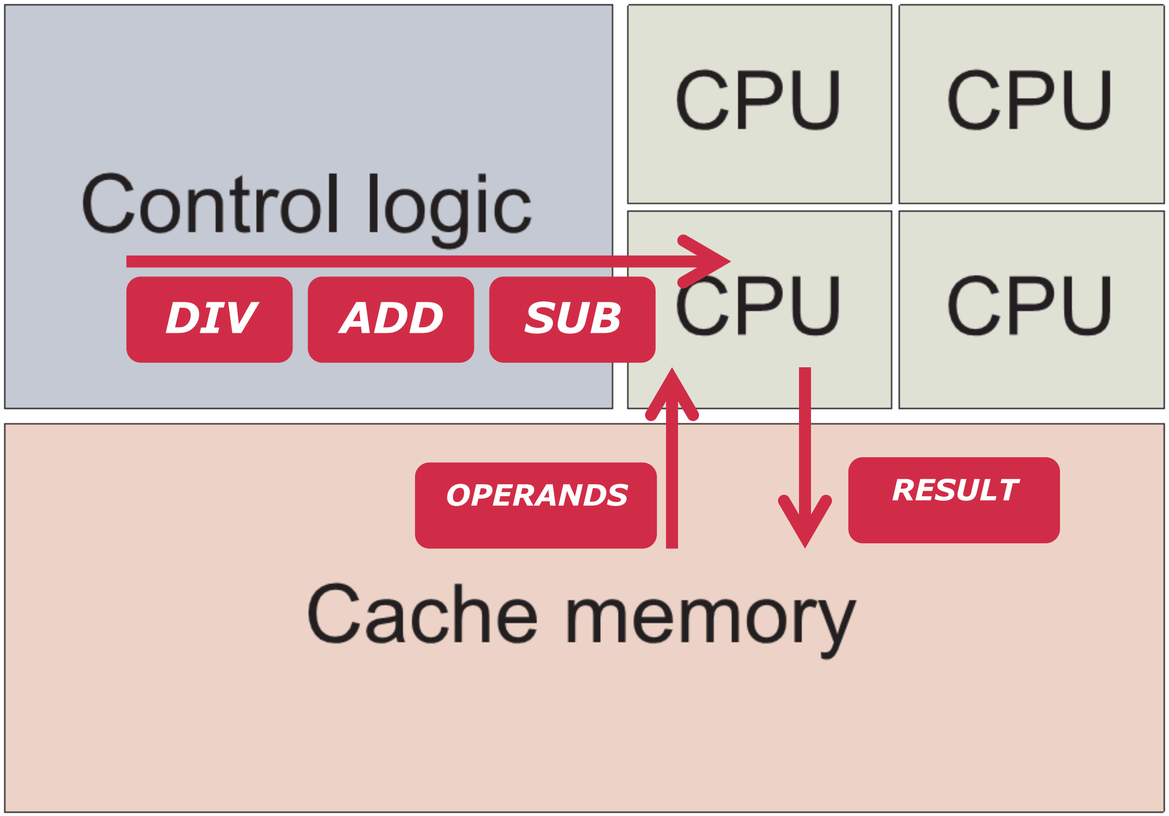
No more "free-lunch"

Direction 2: Use Parallel Architectures (or what I call : Unconventional Computing)

- ***Parallel architectures*** *are the present and the future (?):*
  - *Multi-core CPUs*
  - *Graphical Processing Units (GPUs)*
  - *Field Programmable Gate Arrays (FPGAs)*



Parallel hardware platforms – Multi-core CPU

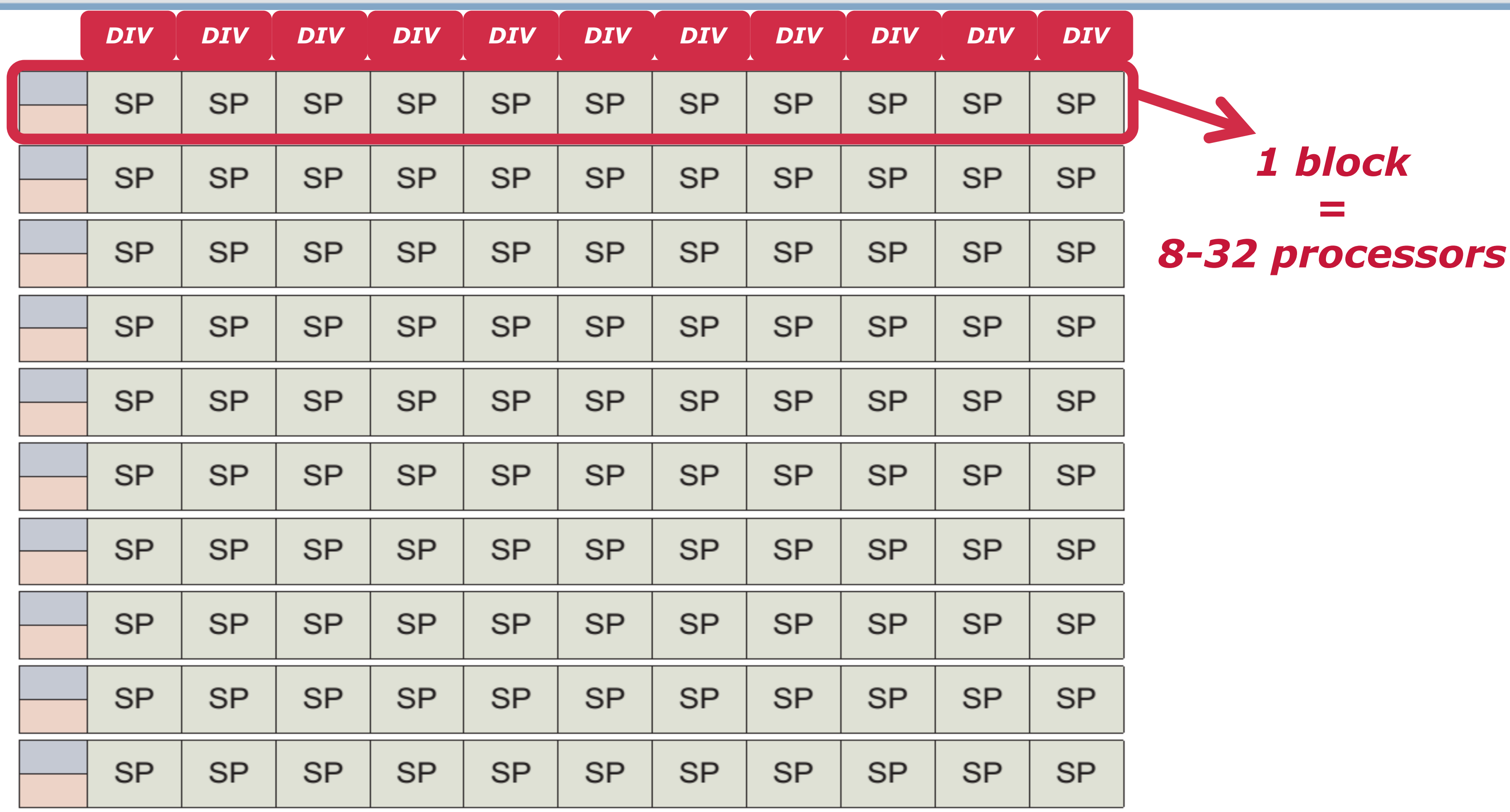


- Few powerful processing cores
- Lots of control logic and cache memory – designed for sequential code
- Fixed instruction set
- Pros: Flexible, easy to use, Cons: Limited speedup, high power

Parallel hardware platforms – GPU

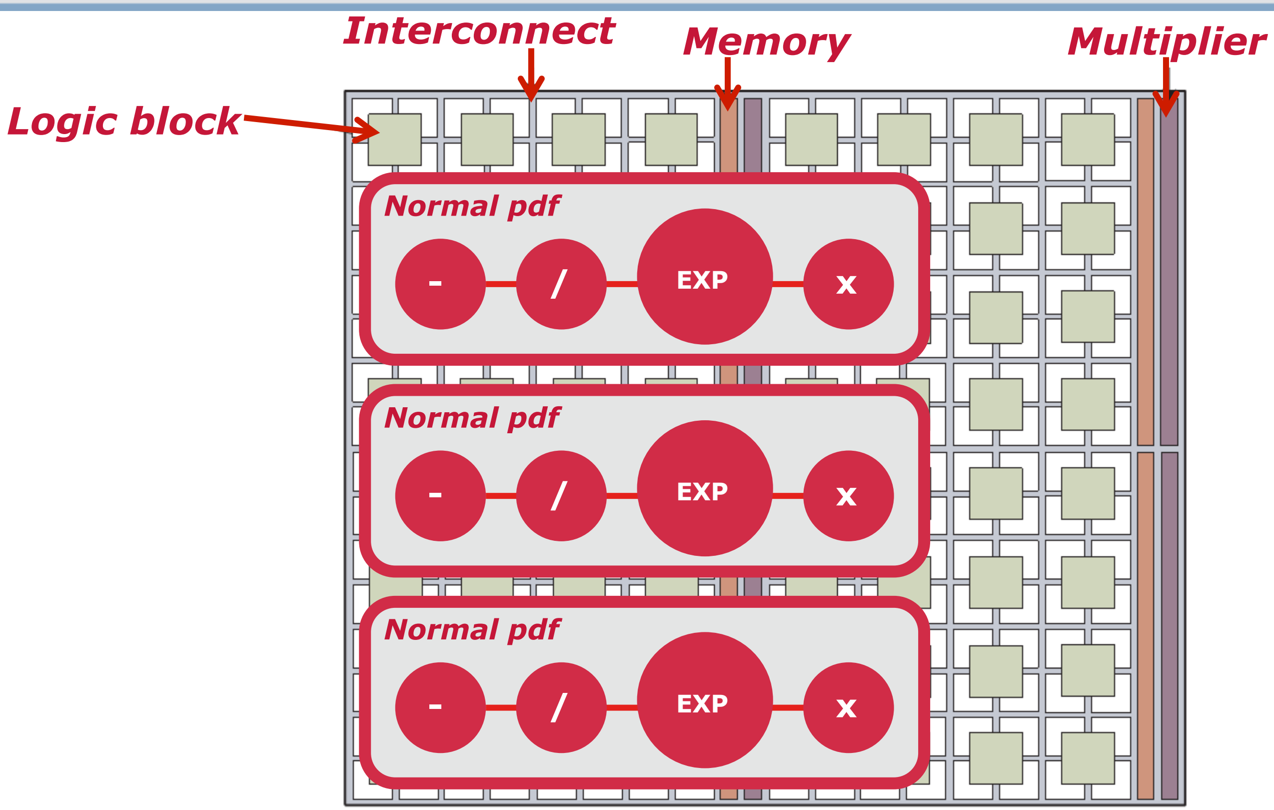
- Many light-weight processors, minimal control and cache
- Pros: Massive peak performance, good for SIMD applications, easy to program
- Cons: Underperforms on non-SIMD code, medium power efficiency

Parallel hardware platforms – GPU

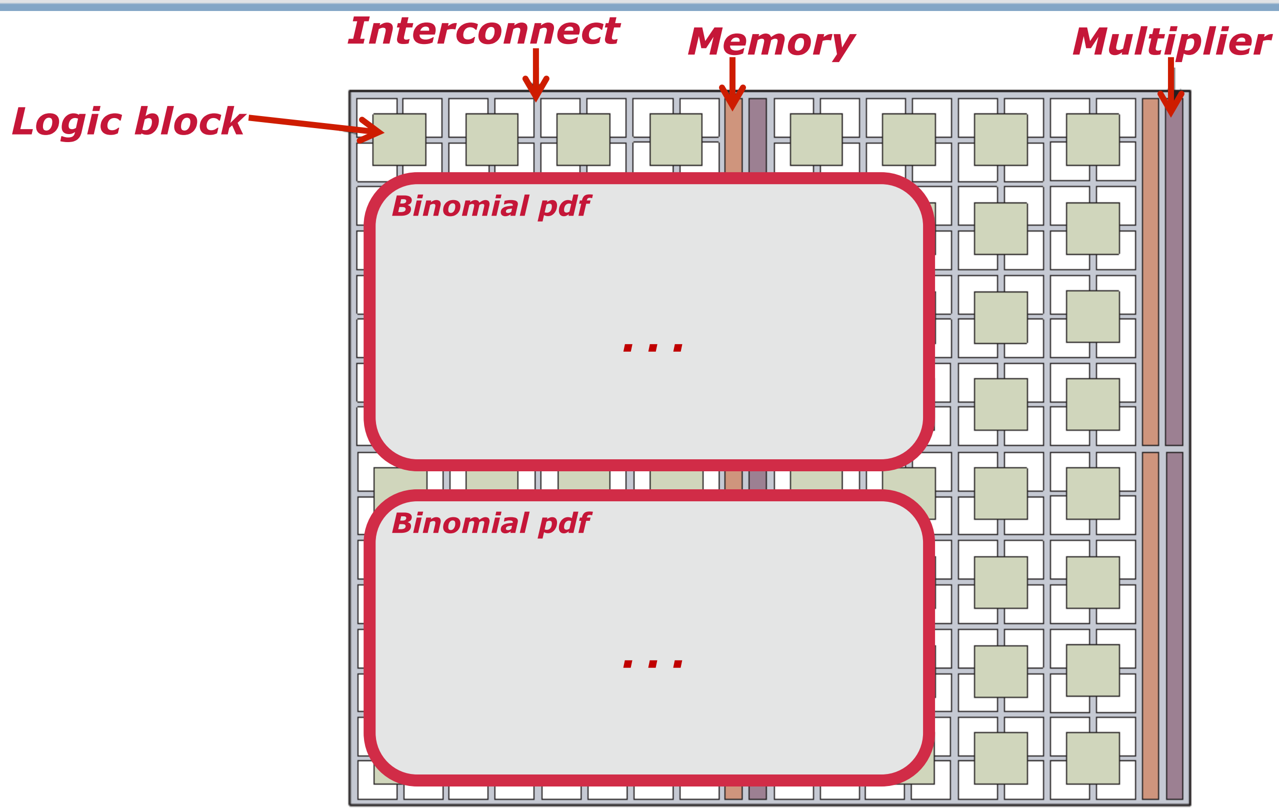


- Many light-weight processors, minimal control and cache
- Pros: Massive peak performance, good for SIMD applications, easy to program
- Cons: Underperforms on non-SIMD code, medium power efficiency

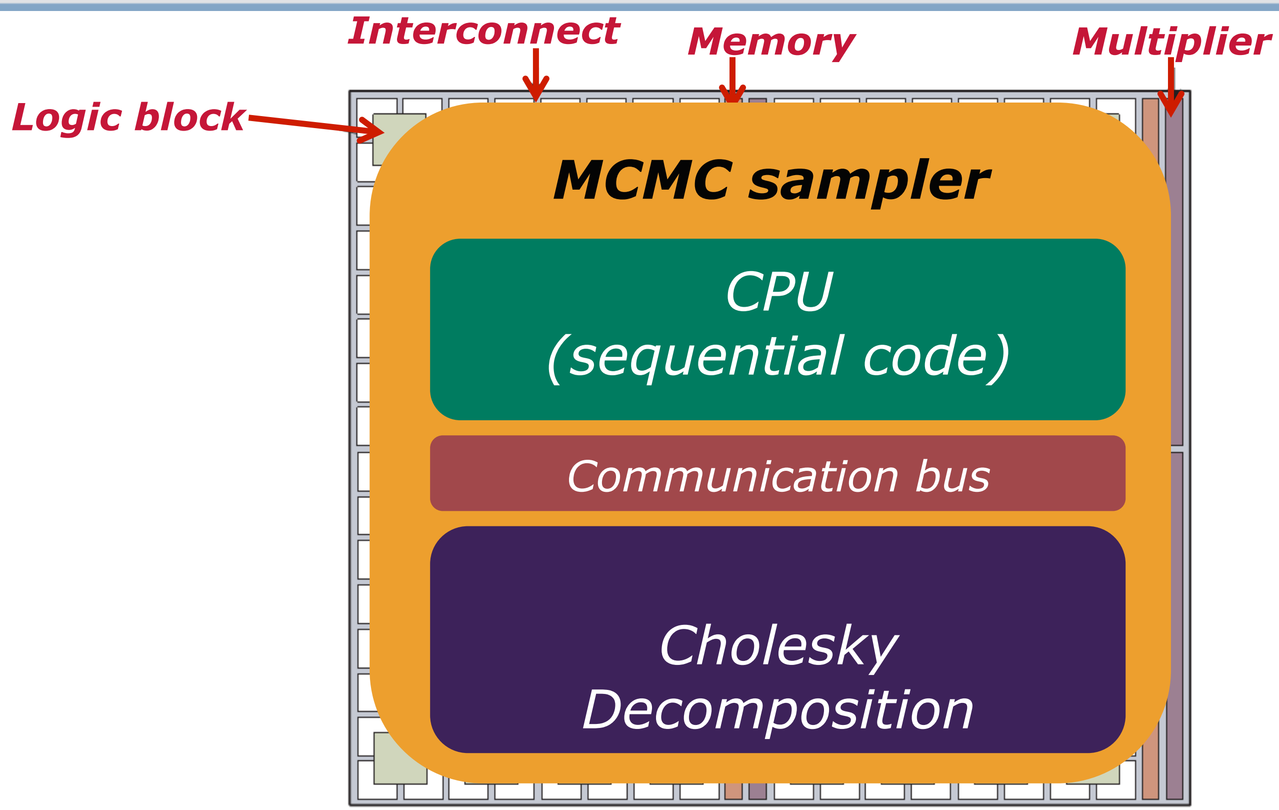
Parallel hardware platforms – FPGA



Parallel hardware platforms – FPGA



Parallel hardware platforms – FPGA



- No fixed processing architecture – no pre-defined instruction set
- Pros: Can be tailored to application, low power, runtime re-configuration
- Cons: Difficult to program, slow compilation ← We started to address that

Programming models

Multi-core CPU

- C/C++ (or any other language)
- Vector Instructions (compiler)
- Implicit threading (Hyper-Threading, Multiple Cores)
- OpenMP API (explicit threading opportunities in the code – shared memory)

GPU

- C/C++ and extra keywords
- CUDA from Nvidia, OpenCL from Khronos group, libraries
- SIMD architecture

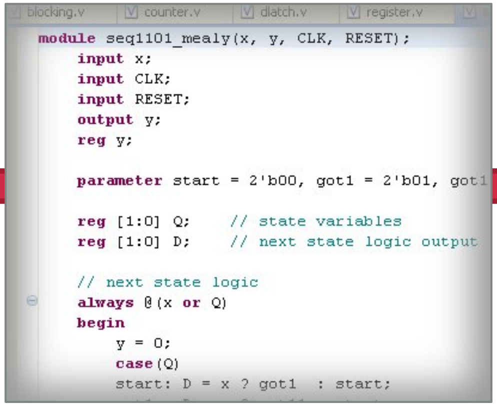
FPGAs

- *(They need their own slide…)*

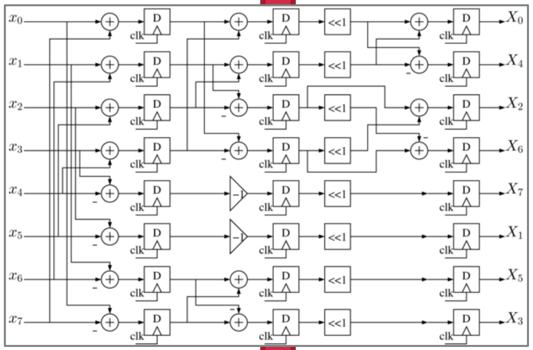
Imperial College London

FPGA design flow

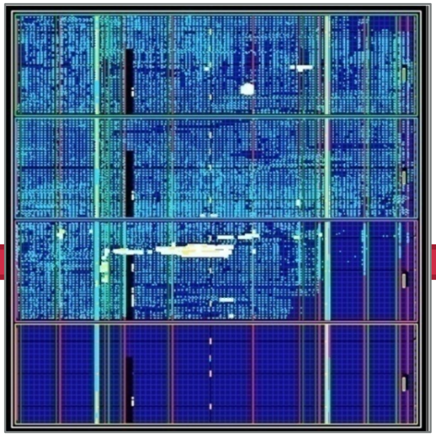
Code writing


```
blocking.v | counter.v | dlatch.v | register.v
module seq1101_mealy(x, y, CLK, RESET);
    input x;
    input CLK;
    input RESET;
    output y;
    reg y;

    parameter start = 2'b00, got1 = 2'b01, got1

    reg [1:0] Q;    // state variables
    reg [1:0] D;    // next state logic output

    // next state logic
    always @(x or Q)
    begin
        y = 0;
        case(Q)
        start: D = x ? got1  : start;
```

Synthesis (minutes)



Place and route (minutes-hours)



Configuration file generation (seconds)

A high-level comparison of devices (1/2)

	FPGA: Xilinx XCVU440	GPU: Nvidia K40	CPU: Intel Xeon E7-4800
Peak performance	4,904 GFLOPs	5,364 GFLOPs	1.246 GFLOPs
Power consumption	20-40 W	235 W	140 W
Memory bandwidth (global)	50-200 GB/sec	336 GB/sec	85 GB/sec
Memory bandwidth (cache)	~300 TB/sec	~50 TB/sec	~10 TB/sec

A high-level comparison of devices (2/2)

Spectrum of computational devices

ASIC FPGA GPU General-purpose CPU

- High Performance
- Low power

- Low performance
- High power

- Long development time

- Short development time

GPU or FPGA?

No device is best for all applications

- **Use GPU when:**
  - Algorithm is SIMD/ embarrassingly parallel
  - No conditional statements
  - Memory access is deterministic and structured and data set is massive
  - Ease of use is the primary concern
- **Use FPGA when:**
  - Algorithm has medium-high parallelism degree – not necessarily SIMD
  - Algorithm is streaming-based – allows extensive pipelining
  - Custom arithmetic precision can be employed
  - Power efficiency is important

Imperial College London

Outline

- Motivation
- Unconventional Computing
  - What are multi-core CPUs, GPUs and FPGAs?
  - How are they different?
- Employing reconfigurable computing for Bayesian Inference
  - aka: What have we done so far?
- What do we plan to do?
- What to keep from this talk

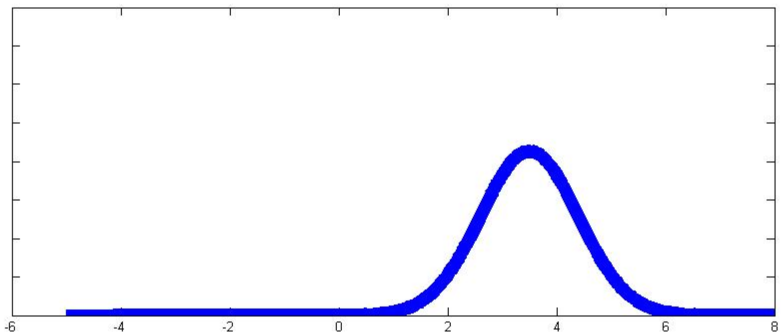
Our research on Bayesian Inference

Techniques to perform fast Bayesian inference using parallel devices (multi-core CPUs, GPUs, **FPGAs**):

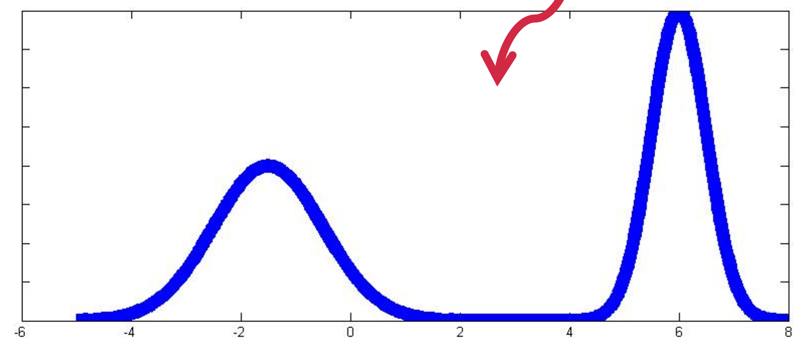
- **Target inference methods (FPGAs)**
  - Sampler-specific hardware design (Population-based MCMC, Particle MCMC)
  - Algorithmic modifications and new algorithms
- **Target Bayesian models (FPGAs and GPUs)**
  - Variable selection for linear regression with many predictors/responses
  - Work on QR factorization
- **Target optimizations for MCMC algorithms (FPGAs)**
  - Arithmetic precision optimization for MCMC algorithms

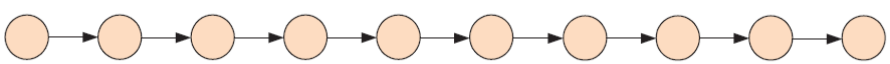
How do we map efficiently (not necessary faithfully) an MCMC algorithm into an FPGA?

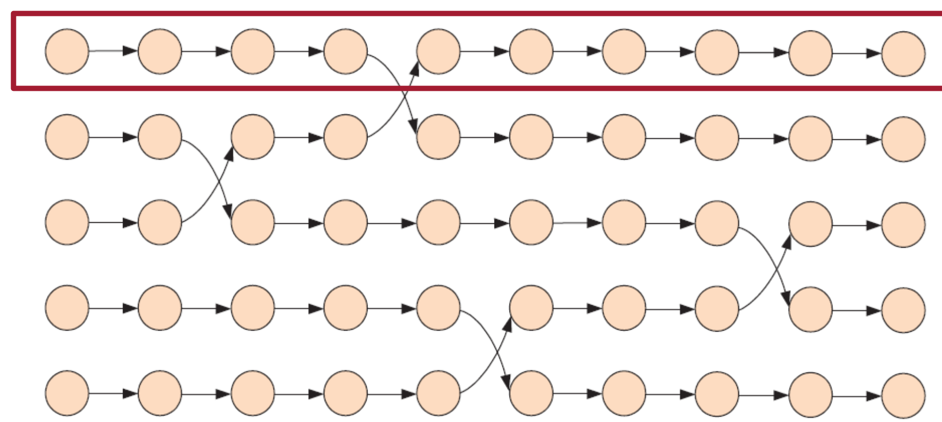
Inference Methods: Population-based MCMC



slow mixing







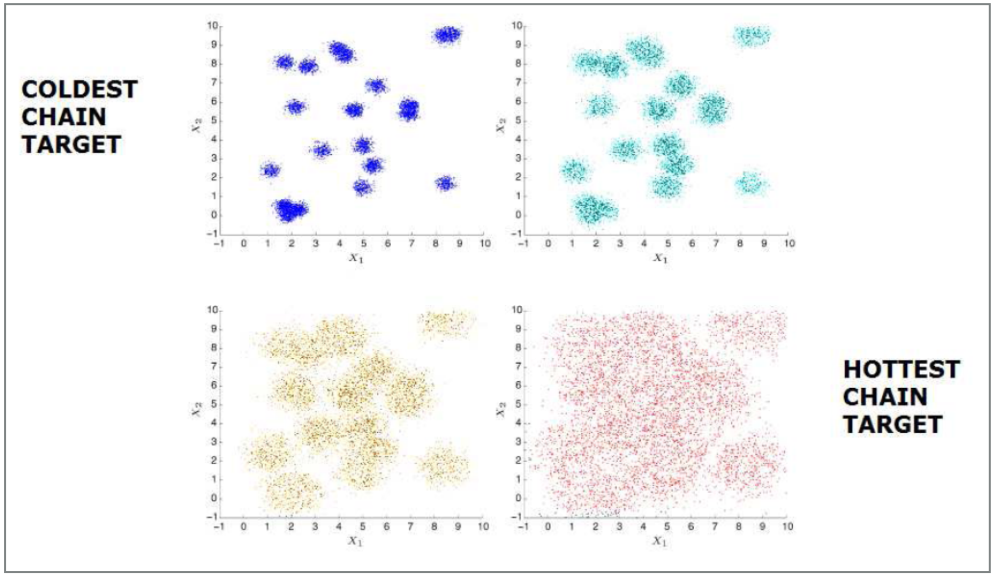
Imperial College London

Parallel Tempering

- Each chain has a different target distribution:

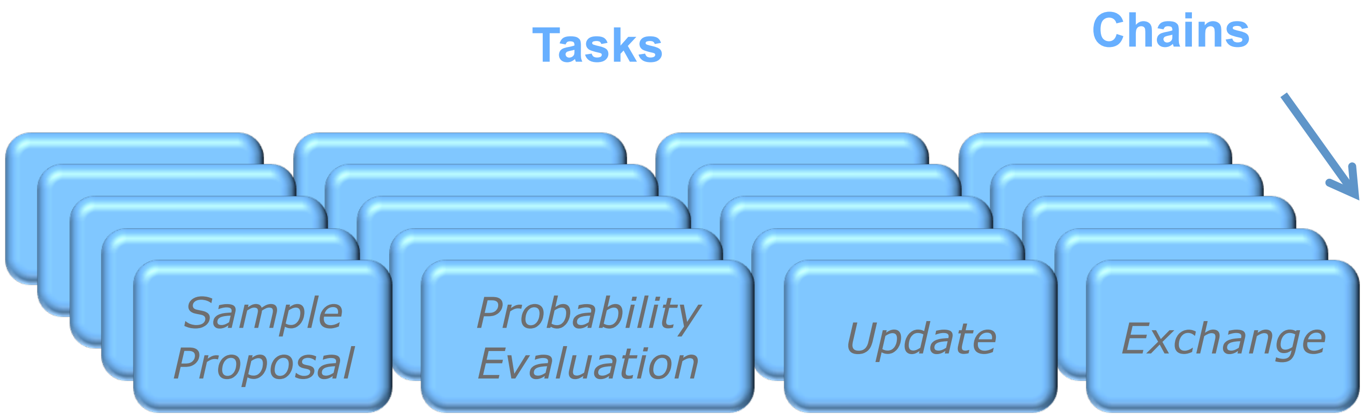
$$p_j(\mathbf{x}) = p(\mathbf{x})^{\frac{1}{T_j}}$$

Chain temperature



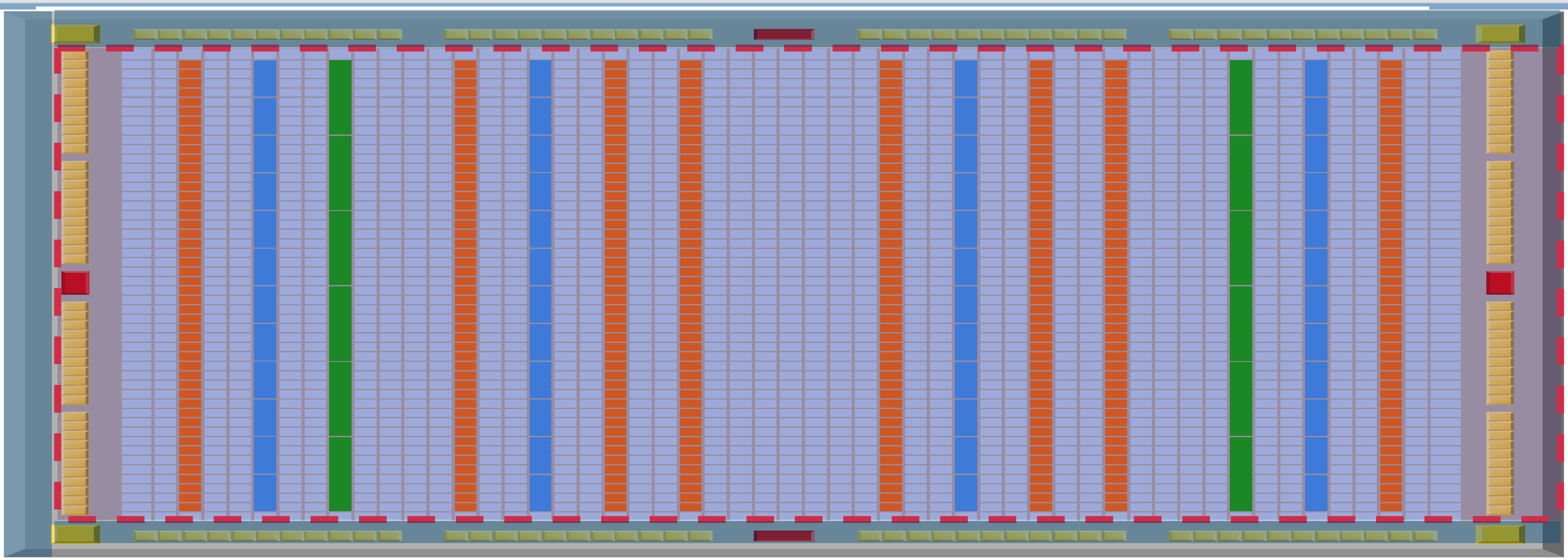
- By exchanging samples, mixing is improved

Traditional architecture - CPU



Processing blocks that run sequentially

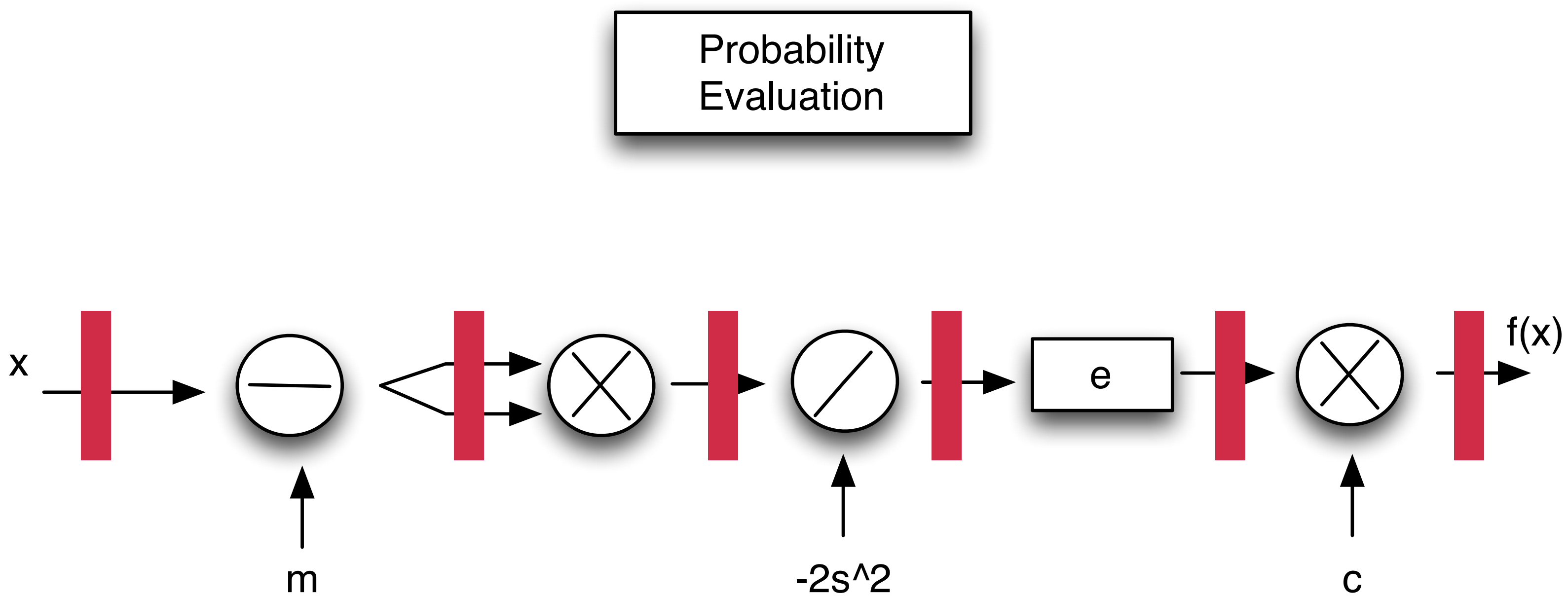
FPGA architecture



- Spatial computations (i.e. RNGs generated on-chip concurrently to processing)
- Pipelined data-path to exploit parallel chains' independency…
- Probability evaluation heavily parallelized
- Access to all memories in one clock cycle (on-chip memories)

Processing blocks that run concurrently in a streaming fashion => high performance

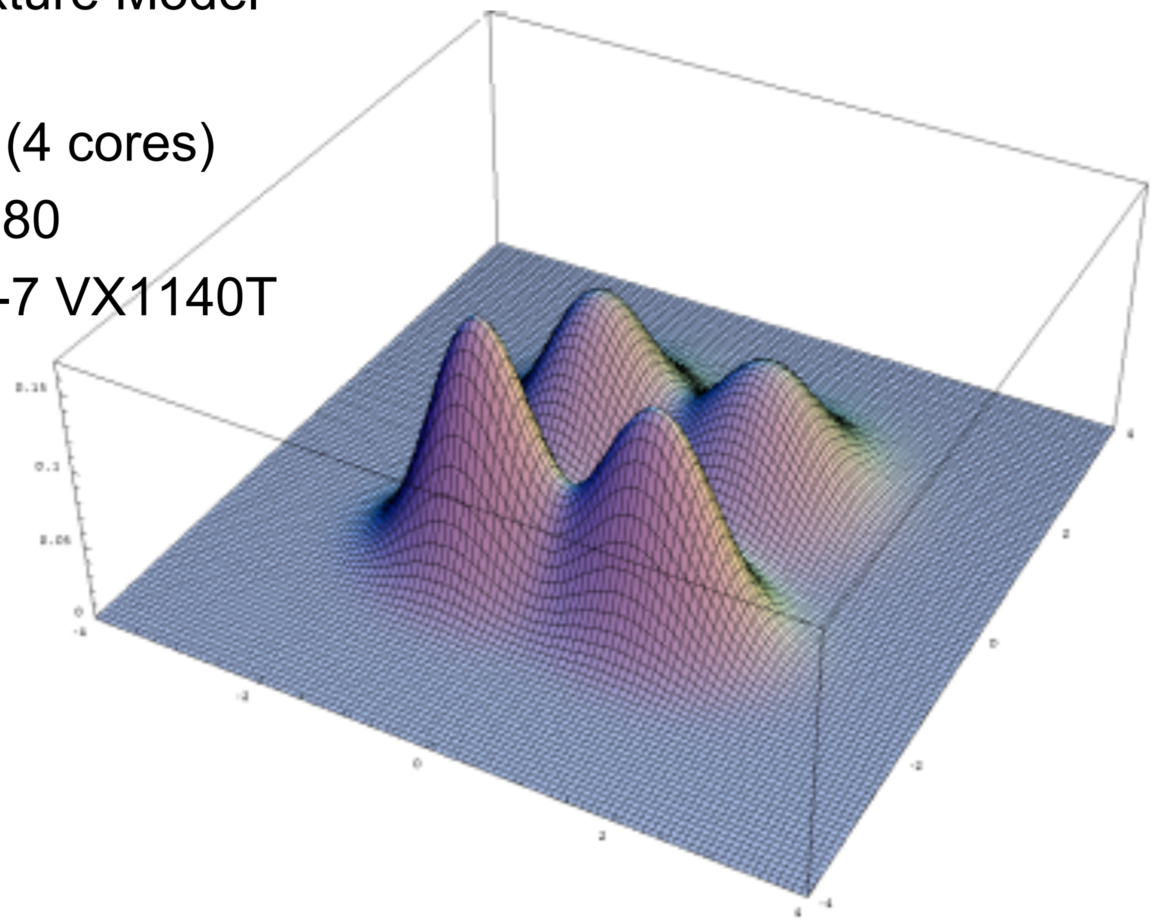
Pipelining



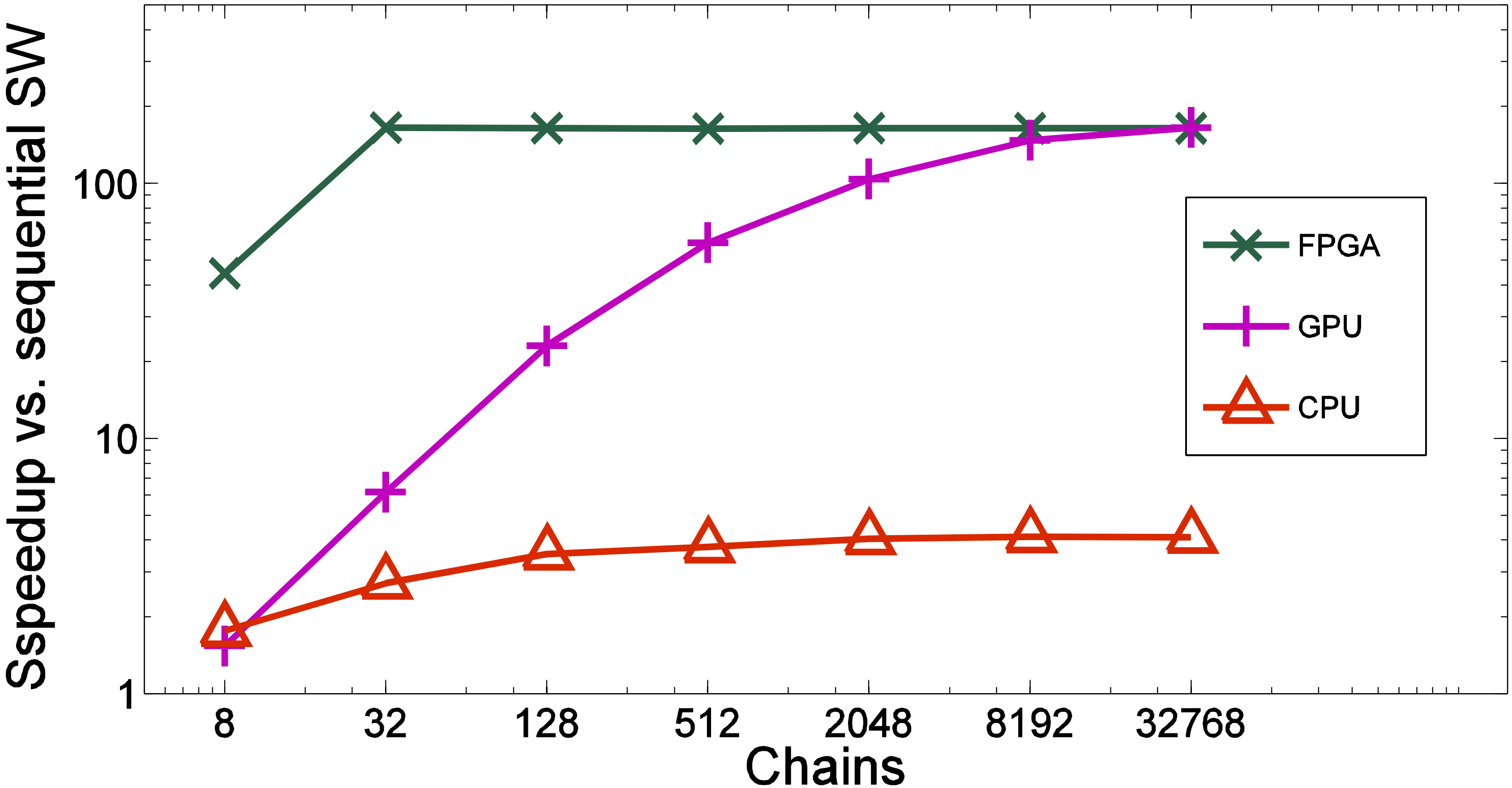
Another form of parallelism

Evaluation

- Finite Gaussian Mixture Model
- Synthetic data
- CPU : Intel i7-2600 (4 cores)
- GPU : Nvidia GTX480
- FPGA: Xilinx Virtex-7 VX1140T



Speedup (vs. Number of chains)



- GPU can reach FPGA performance only for massive chain population
  - → in reality, no gain in mixing after 100-200 chains
- CPU much slower

Performance increase

Can we boost the performance further?

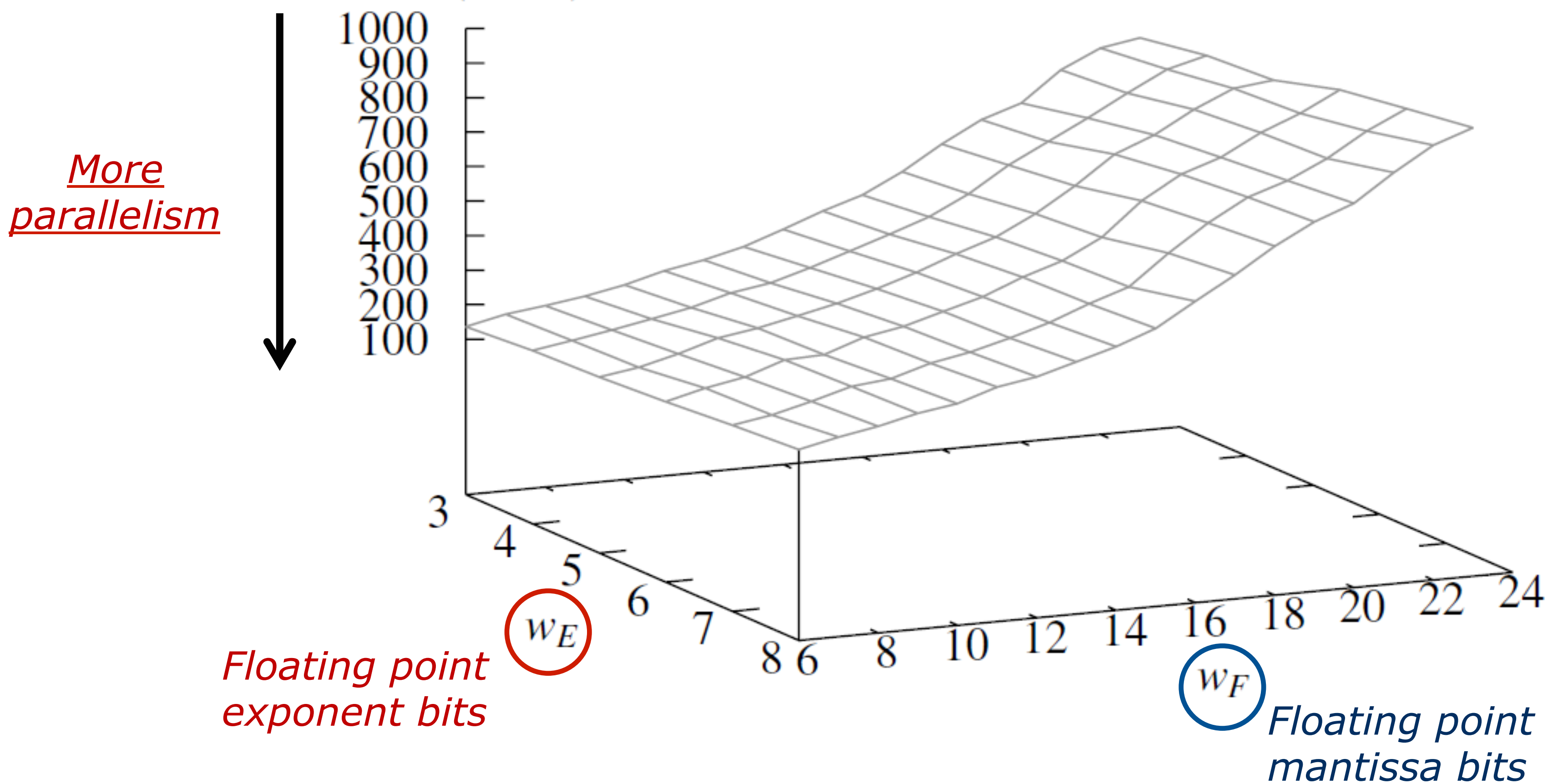
Yes, we can !!!

Tuning arithmetic precision

- Resources = Performance in FPGAs
- Double/single precision floating point is the de facto precision in MCMC
  - Is it really necessary?
- FPGAs: Can use custom precision in **parts** of MCMC sampler to speedup performance.
  - In which parts? What is the impact?
- Trade-off: Speed vs. Accuracy

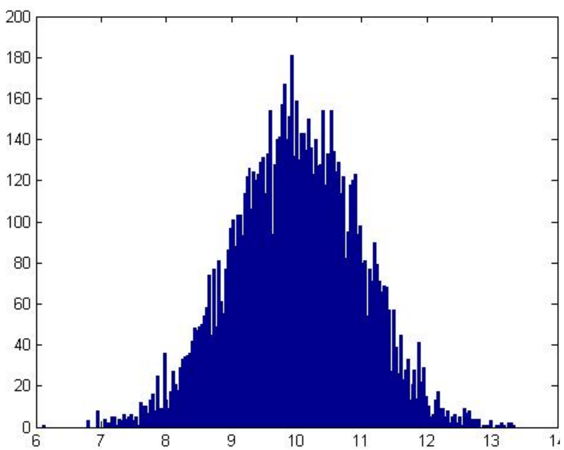
Precision: Effect on FPGA resources

Cost of 1 logarithm operator

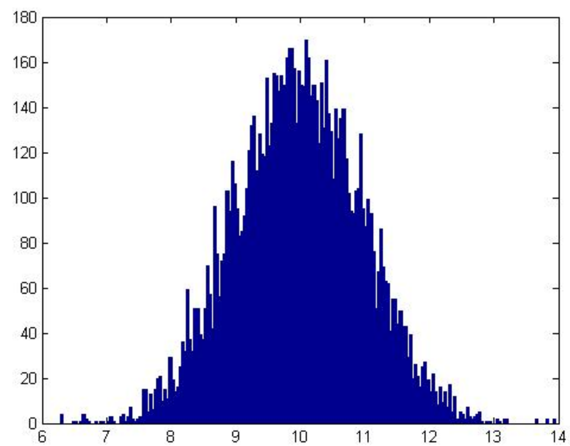


Imperial College
London

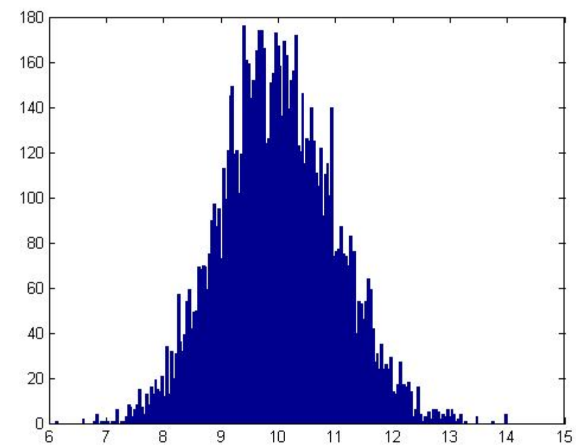
Precision: Effect on sampling accuracy



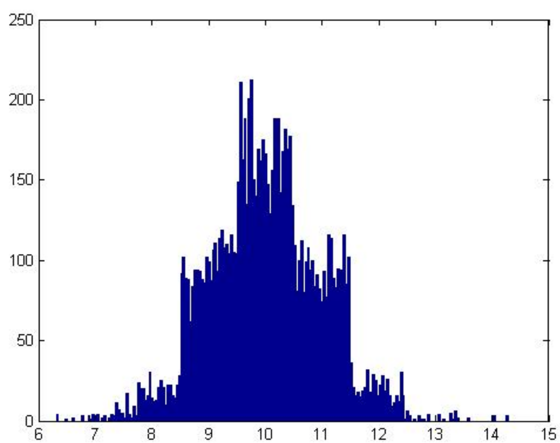
23 bits



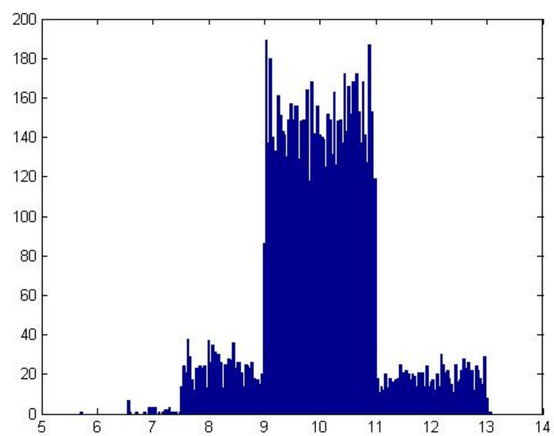
11 bits



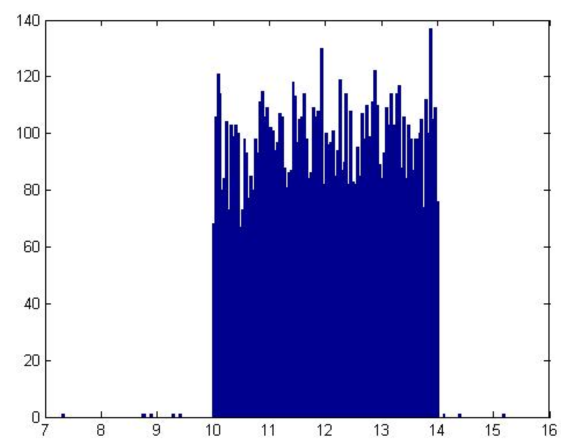
6 bits



4 bits



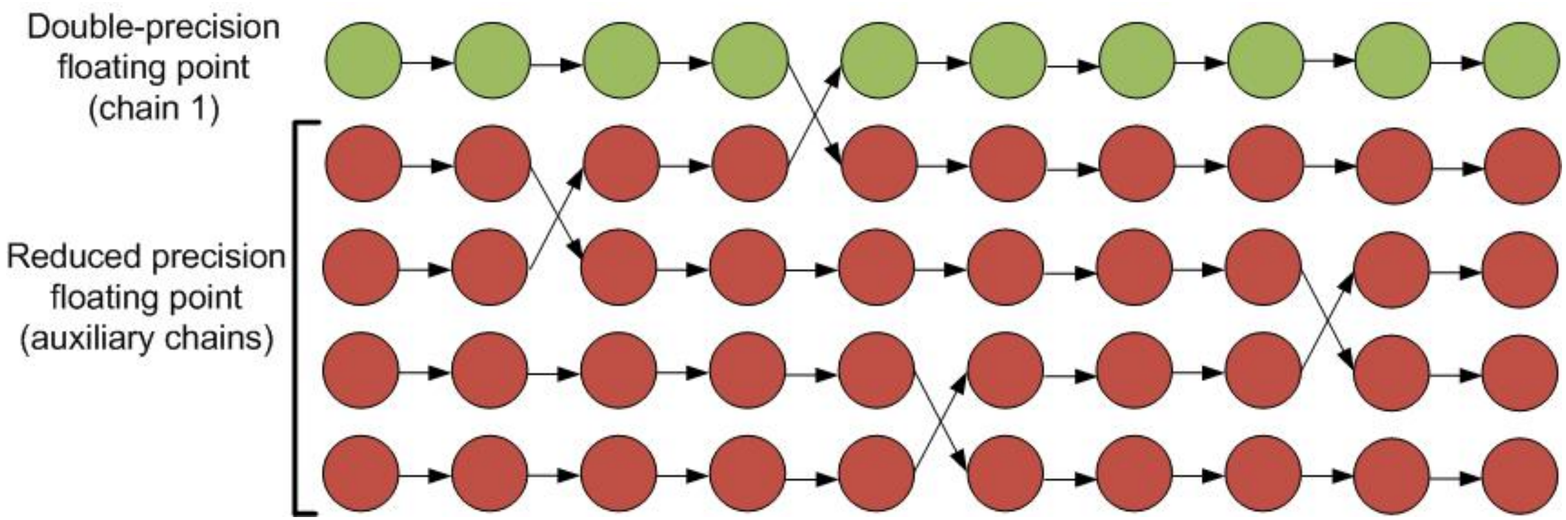
3 bits



2 bits

Custom precision in Population-based MCMC

Scheme 1: Mixed precision

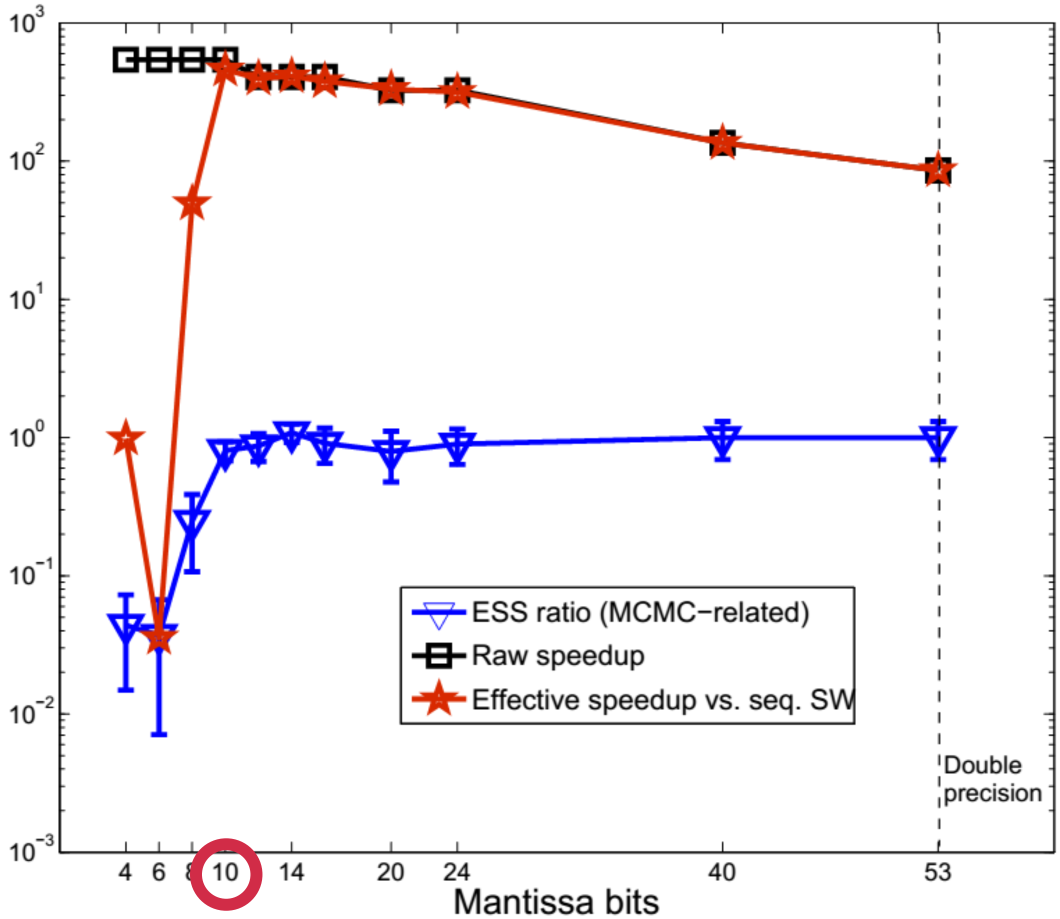


- **Idea: Use double precision only in the first chain**
- Correct sampling is guaranteed
- Penalty paid: Mixing might drop due to low precision in auxiliary chains

Imperial College London

Custom precision in Population-based MCMC

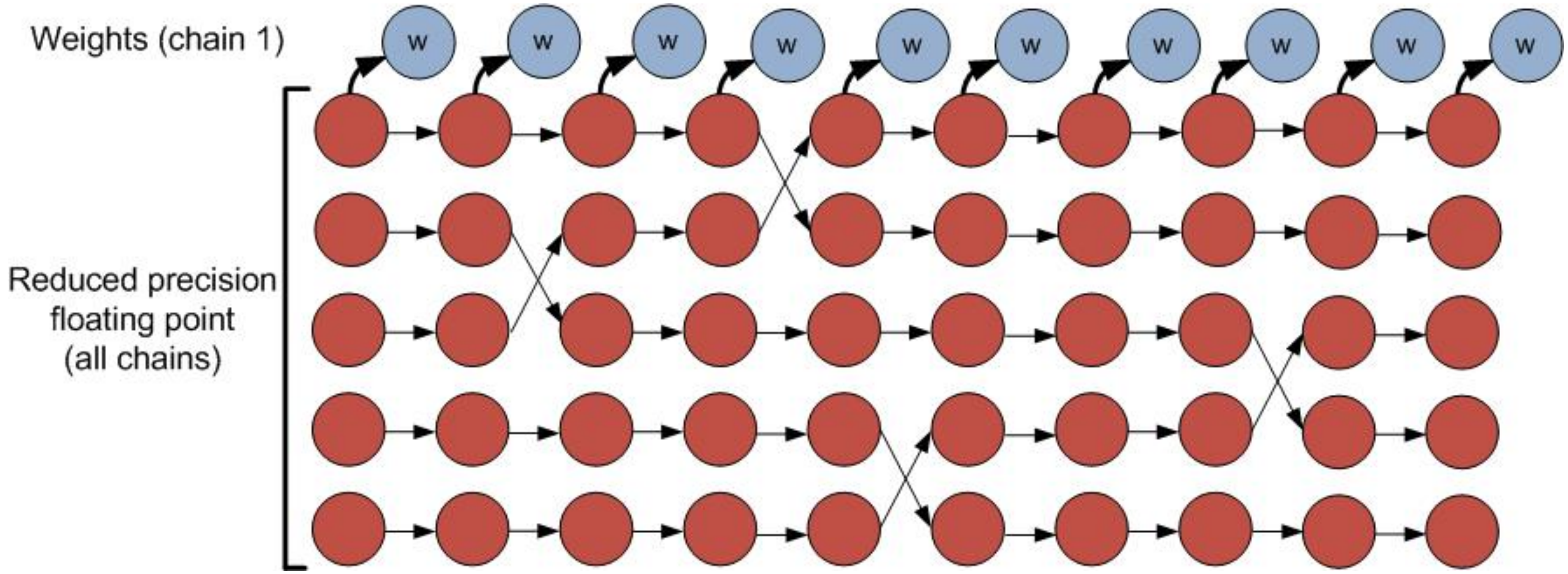
Scheme 1: Mixed precision



- Baseline: Single core CPU
- Mixing – speedup tradeoff
- Optimal precision maximizes effective samples per second

Custom precision in Population-based MCMC

Scheme 2: Weights-based

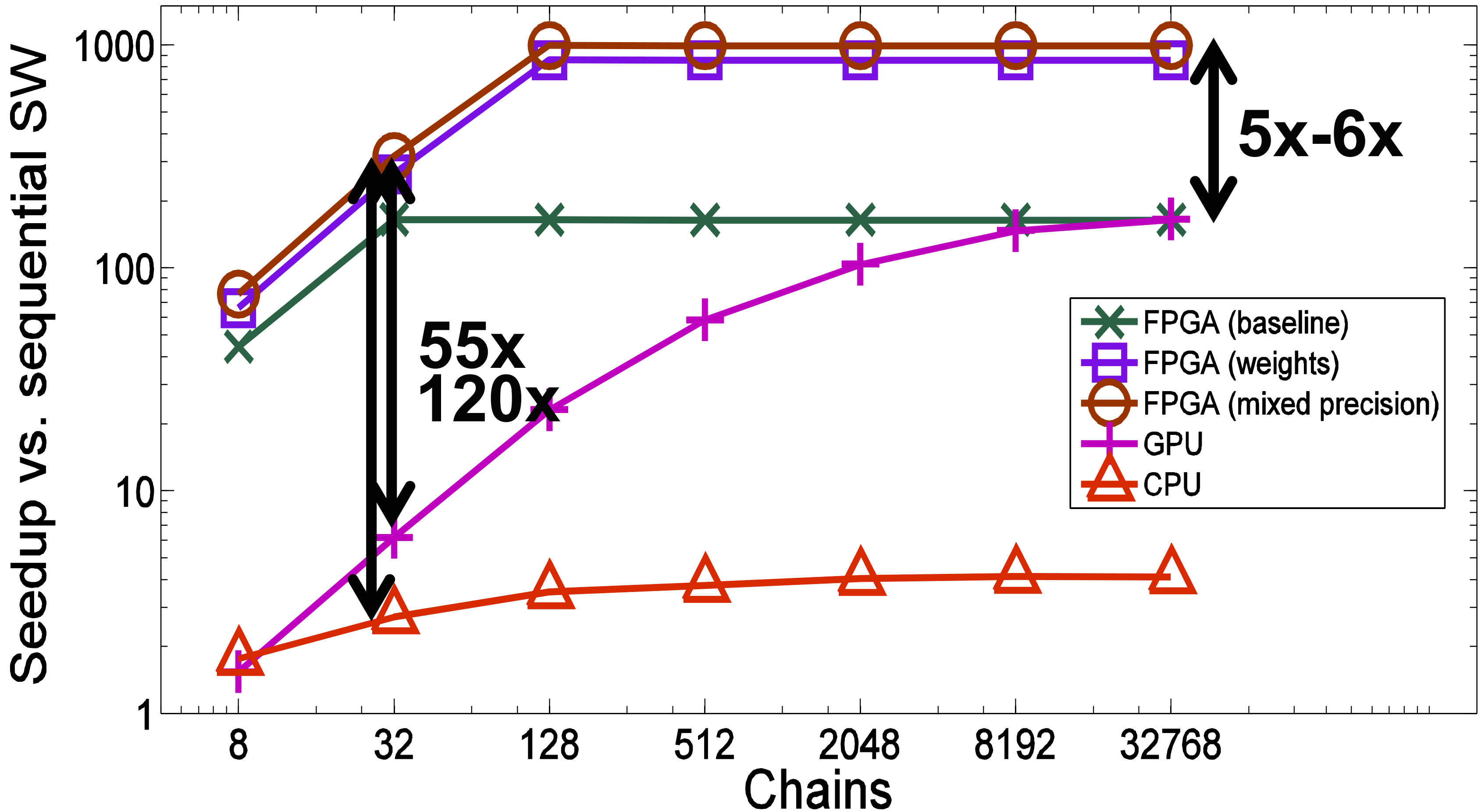


- **Idea: Sample in reduced precision, use weights to correct estimates**
- Similar to Importance Sampling
- Penalty paid: Mixing + effect of IS

Imperial College London

Population-based MCMC: Performance comparison (Mixture model)

FPGA (Mingas et al. 2012) vs CPU vs GPU (Lee et al. 2010)



Generic MCMC precision optimization

Previous custom precision methods:

- No bias
- Specific for Population-based MCMC

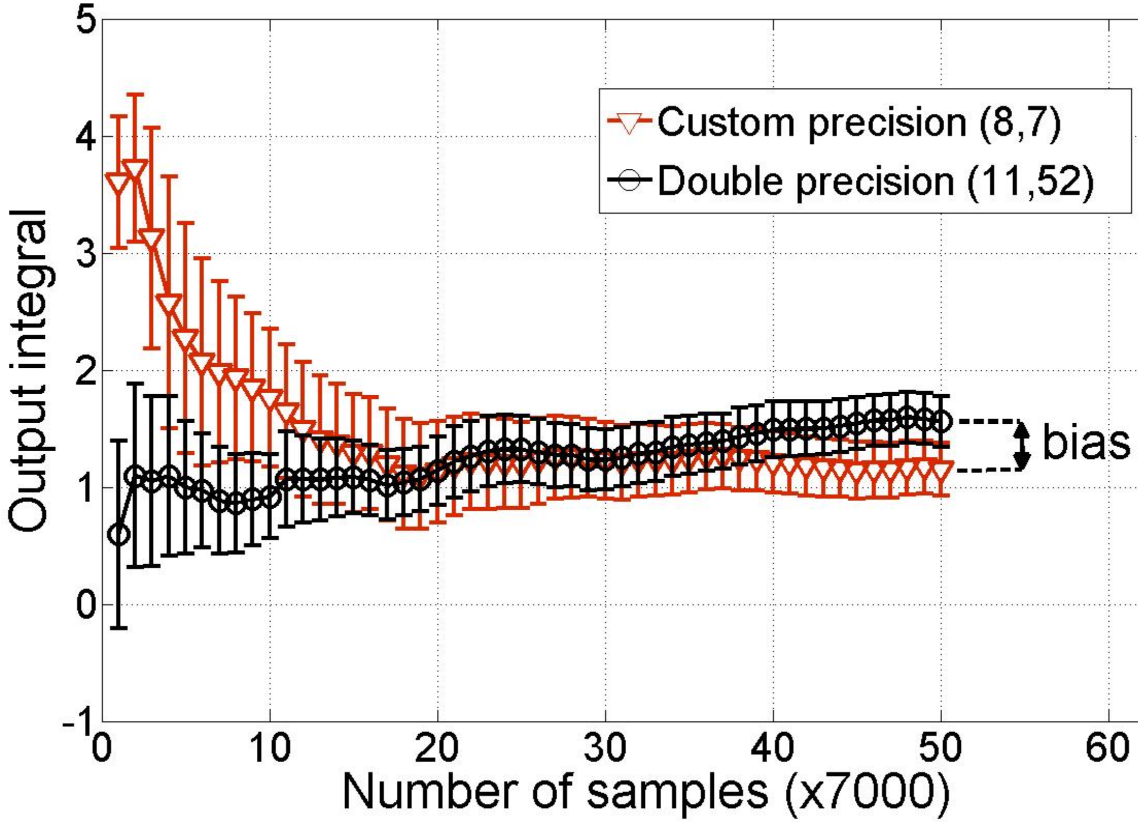
This method:

- Bias allowed but controlled
- Optimize precision given a user-specified bias tolerance
- Applies to any MCMC method

Imperial College London

Precision: Effect on MC estimate

$$E_p[f(\theta)] = \int f(\theta)\,p(\theta)\,dx$$



We need to estimate this

Proposed bias estimator

Bias for precision a

Posterior in double precision

Posterior in custom precision a

$$b_\alpha = \int f(\theta)p(\theta)d\theta - \int f(\theta)p_\alpha(\theta)d\theta = \ldots$$

$$\ldots = \int f(\theta)\left(\frac{p(\theta)}{p_\alpha(\theta)} - 1\right)p_\alpha(\theta)d\theta$$

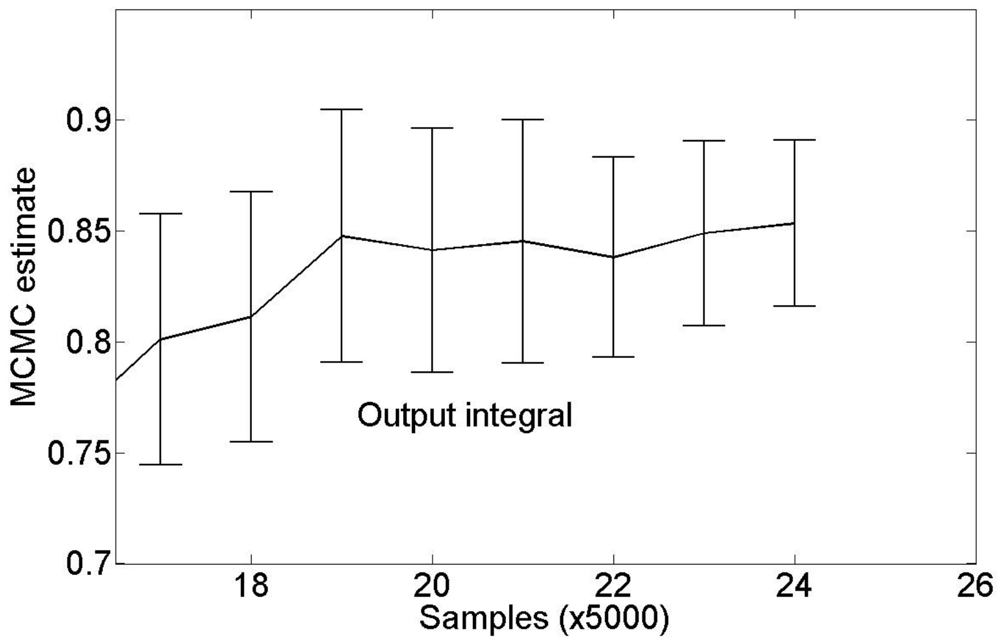
Short MCMC pre-runs to estimate bias integrals

Imperial College London

Optimization process on FPGA

User input	MCMC pre-runs	Estimate biases	Choose precision	MCMC final run
SW	HW	SW	SW	HW

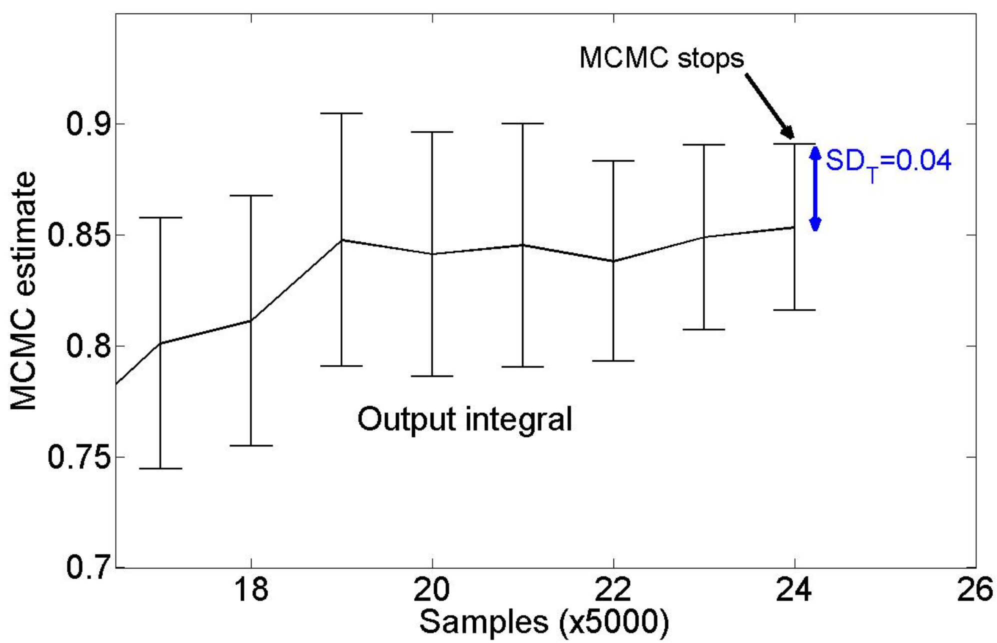
1) *Target standard deviation*
2) *Tolerable bias*



Optimization process

User input	*MCMC pre-runs*	*Estimate biases*	*Choose precision*	*MCMC final run*
SW	HW	SW	SW	HW

1) Target standard deviation
2) Tolerable bias

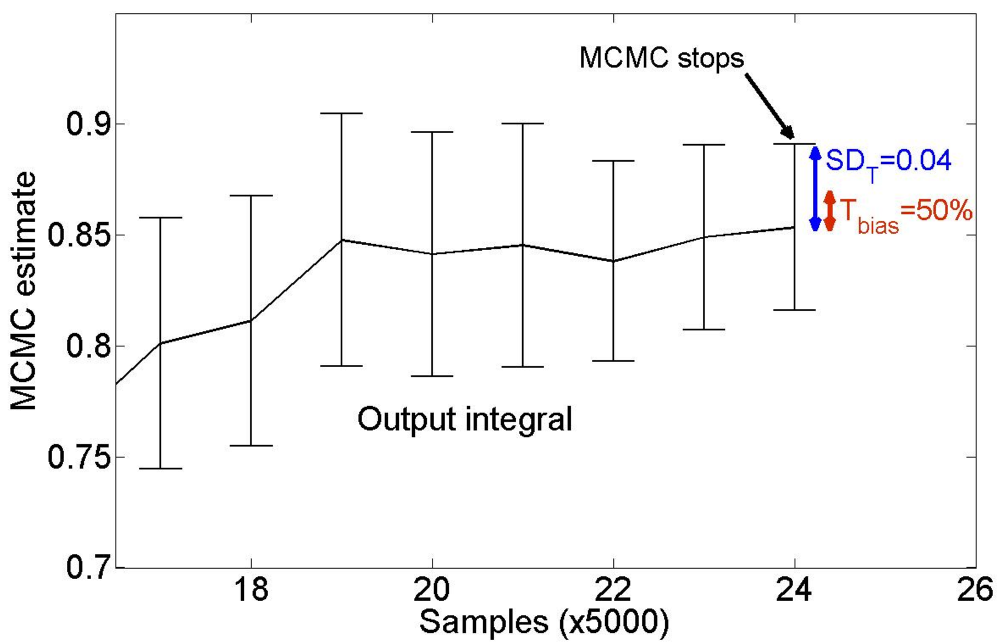


Optimization process

User input | **MCMC pre-runns** | **Estimate biases** | **Choose precision** | **MCMC final run**

SW | **HW** | **SW** | **SW** | **HW**

1) Target standard deviation
2) Tolerable bias

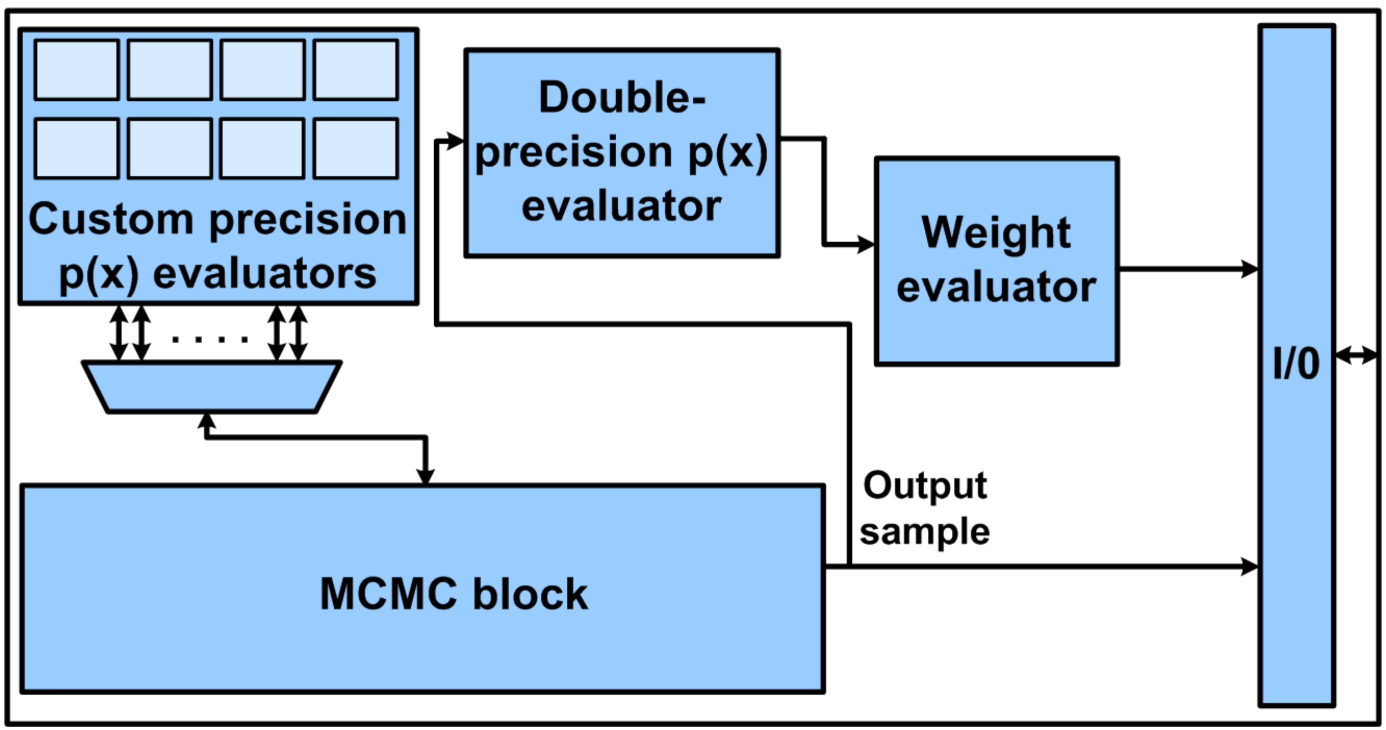


Optimization process

User input	MCMC pre-runs	Estimate biases	Choose precision	MCMC final run
SW	HW	SW	SW	HW

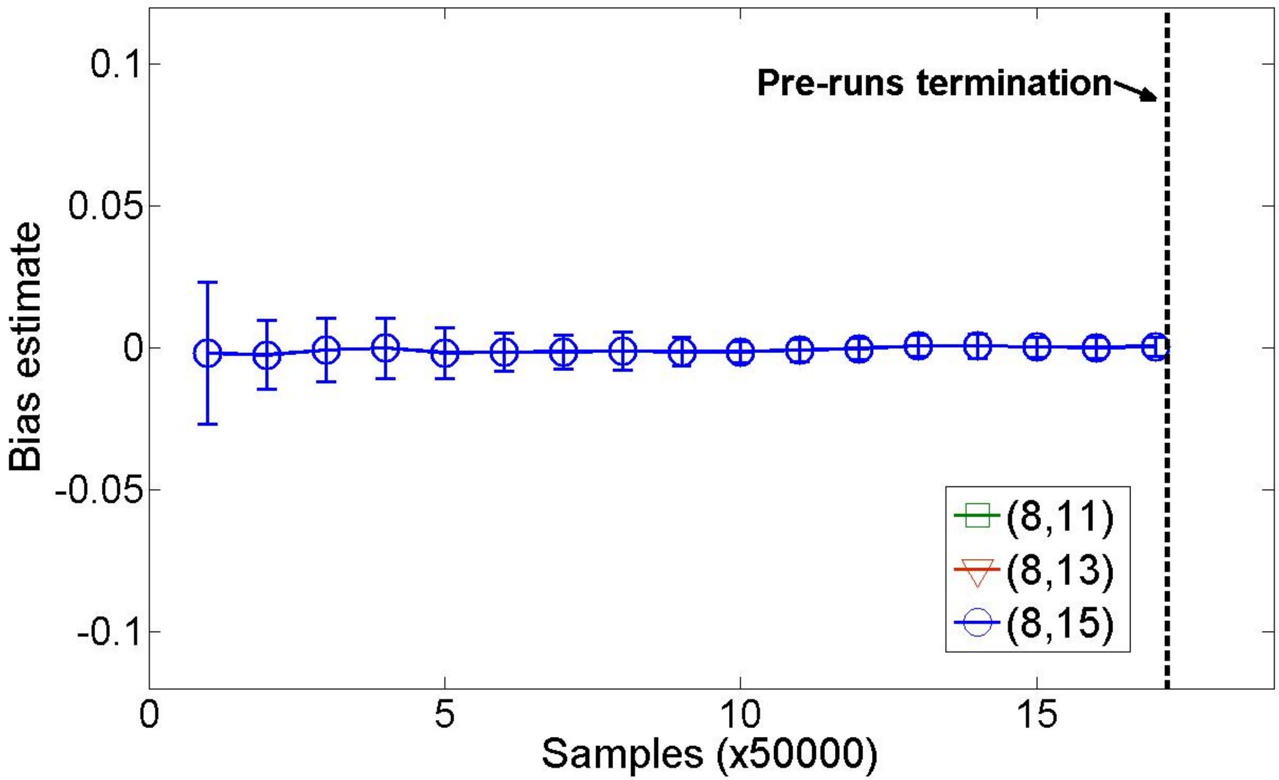
FPGA configuration 1

Short MCMC pre-runs in all candidate precisions



Optimization process

User input	MCMC pre-runs	**Estimate biases**	Choose precision	MCMC final run
SW	HW	**SW**	SW	HW



Imperial College London

Optimization process

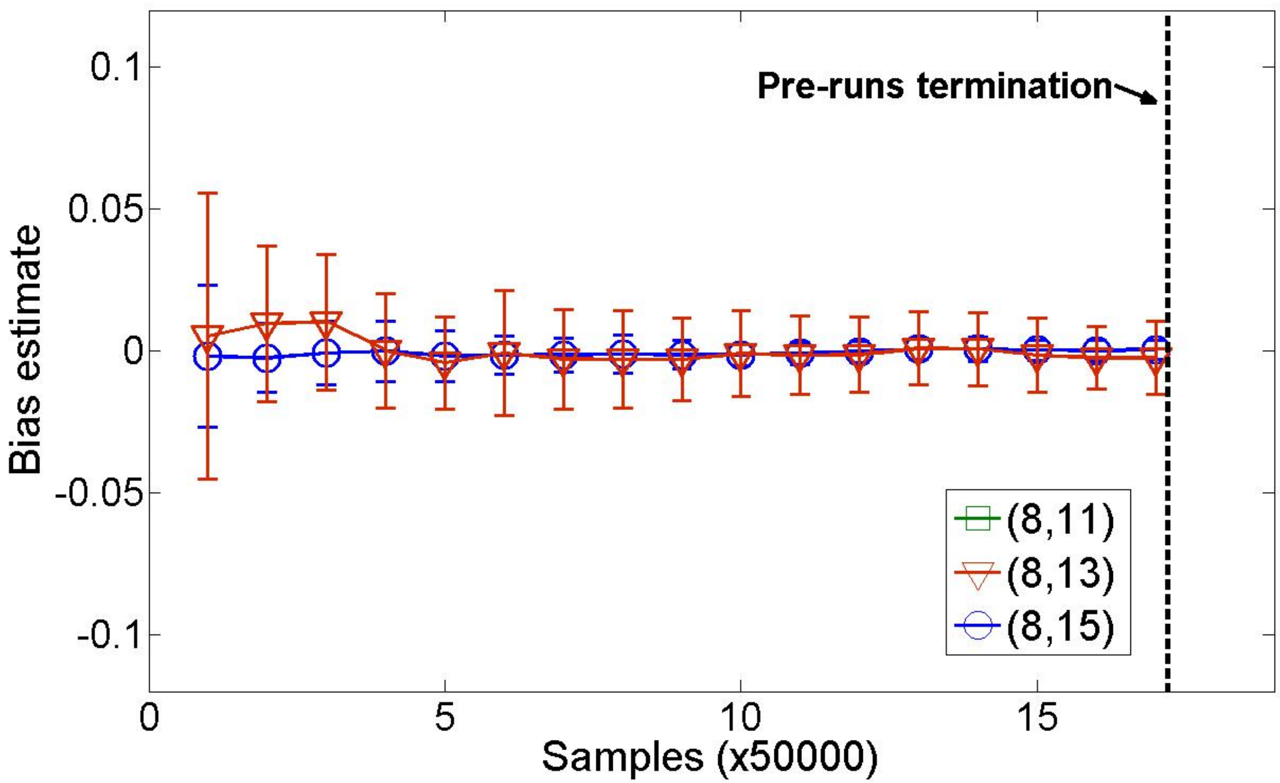
User input — SW

MCMC pre-runs — HW

Estimate biases — SW

Choose precision — SW

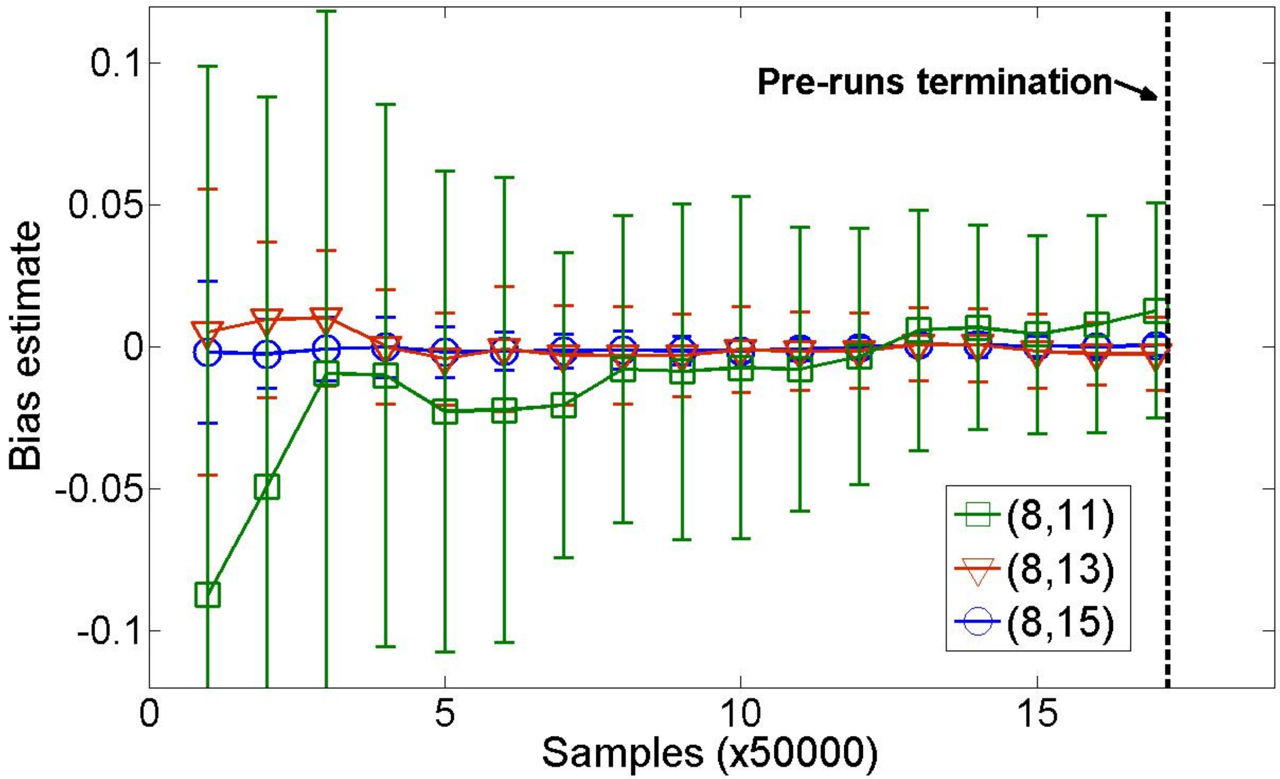
MCMC final run — HW



Imperial College London

Optimization process

User input	MCMC pre-runs	**Estimate biases**	Choose precision	MCMC final run
SW	HW	**SW**	SW	HW



Imperial College London

Optimization process

User input — SW

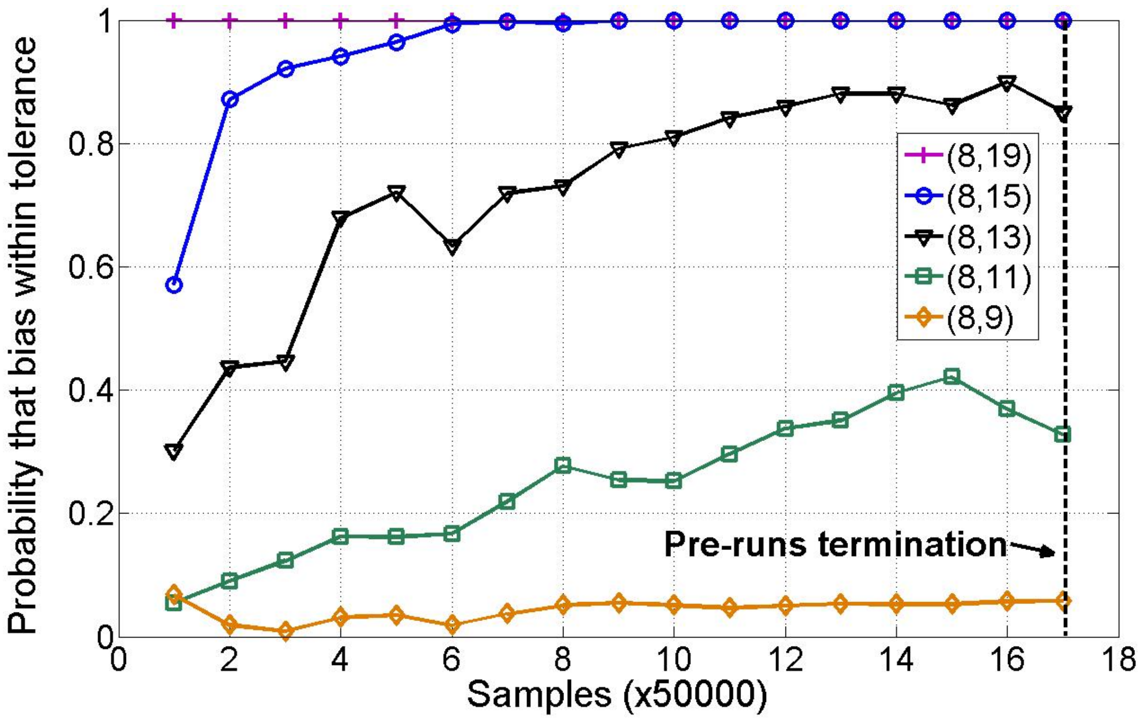
MCMC pre-runs — HW

Estimate biases — SW

Choose precision — SW

MCMC final run — HW

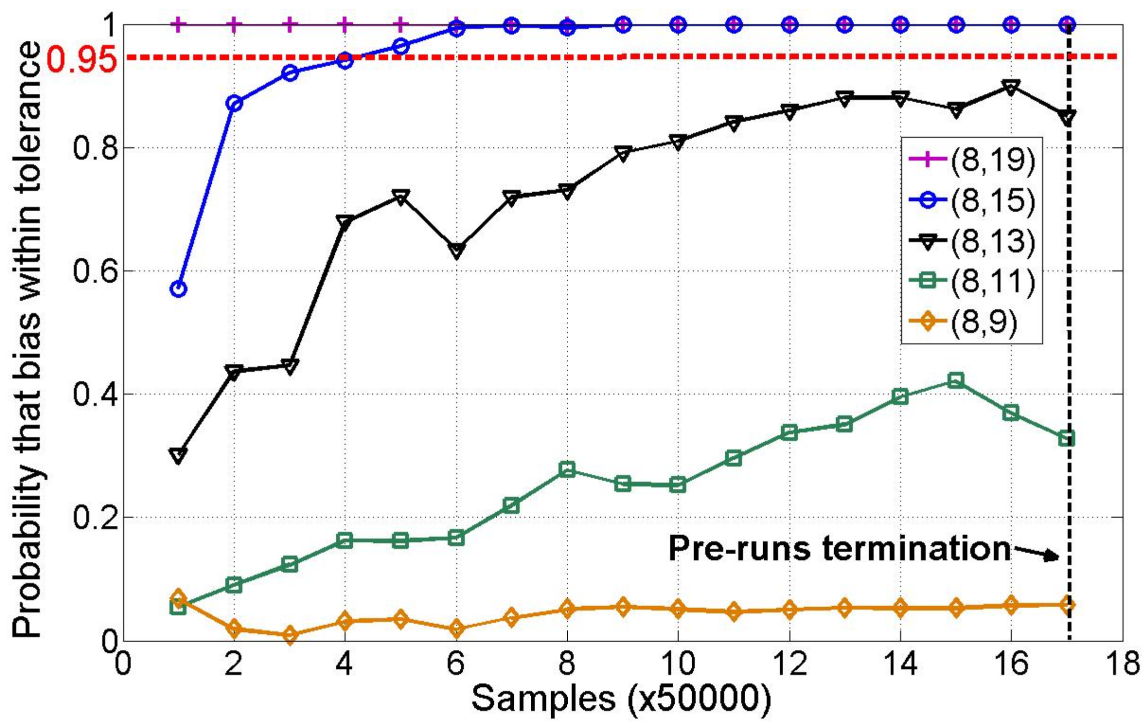
Choose the lowest precision for which: $P(|bias| < SD_T \cdot T_{bias}) > 0.95$



Optimization process

User input	MCMC pre-runs	Estimate biases	**Choose precision**	MCMC final run
SW	HW	SW	**SW**	HW

Choose the lowest precision for which: $P(|bias| < SD_T \cdot T_{bias}) > 0.95$



Optimization process

User input — SW

MCMC pre-runs — HW

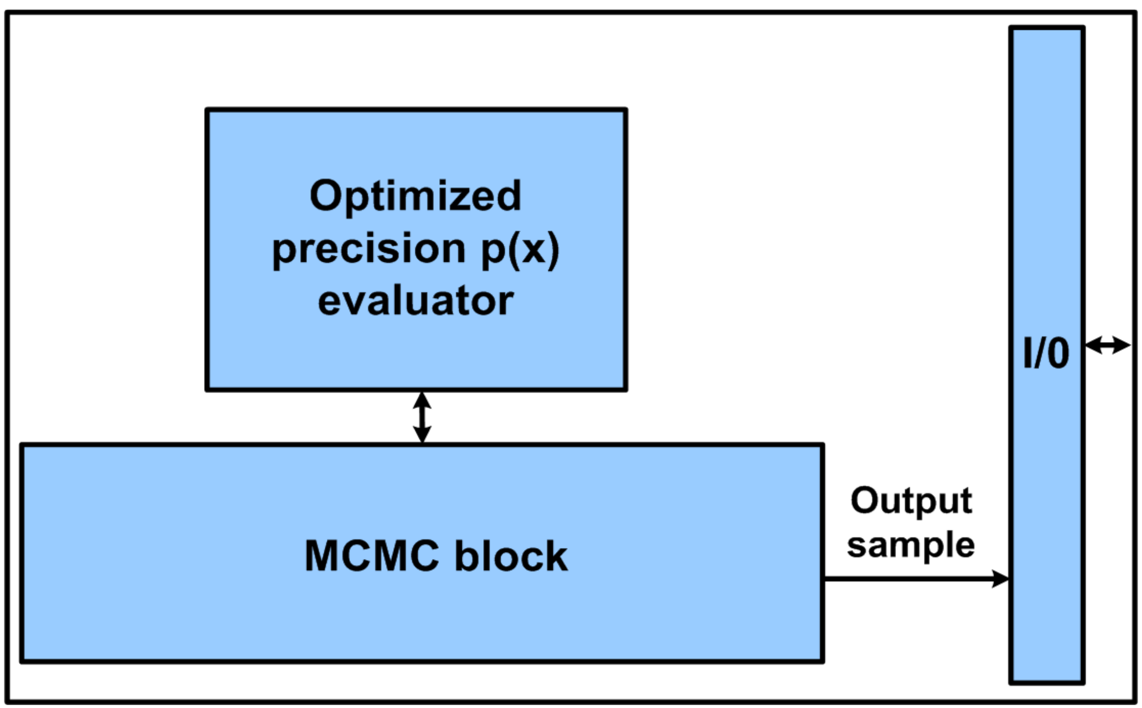
Estimate biases — SW

Choose precision — SW

MCMC final run — HW

FPGA configuration 2

Final, long MCMC run in optimized precision

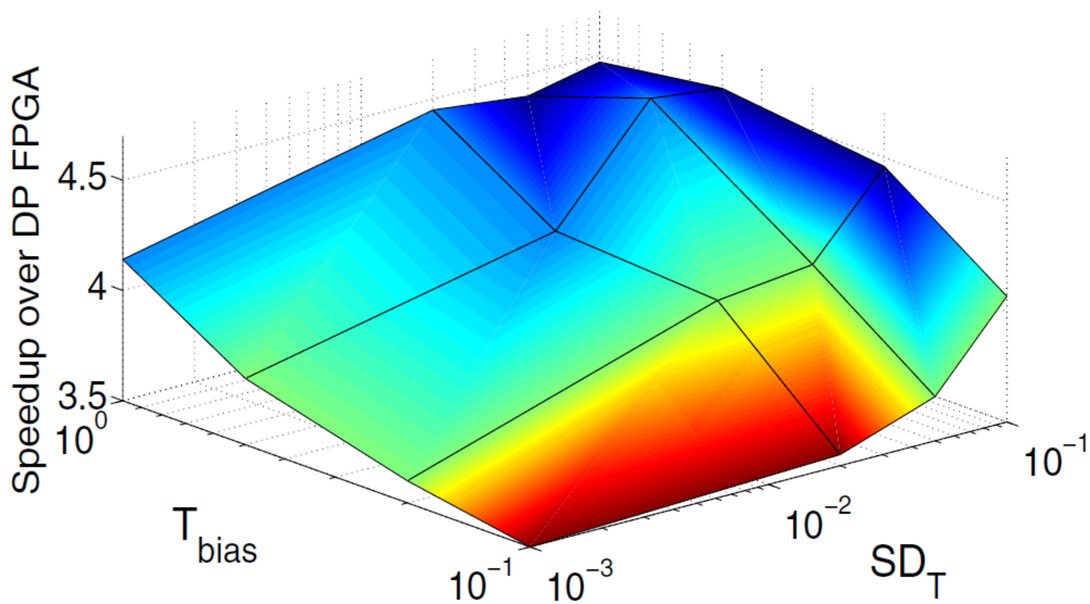


Imperial College London

Evaluation

MCMC target: Mixture model (Jasra et al. 2007)

SD_T	T_{bias}	Optimized precision	Speedup vs. DP FPGA
0.05	100%	**(8,13)**	**4.48x**
0.05	50%	**(8,13)**	**4.59x**
0.02	50%	**(8,15)**	**4.10x**



Exact MCMC acceleration using mixed-precision

Method-Specific Accelerator (PT-MCMC)

Generic Accelerator (but biased)

↓

Unbiased Generic Accelerator

Imperial College London

What to keep from the talk

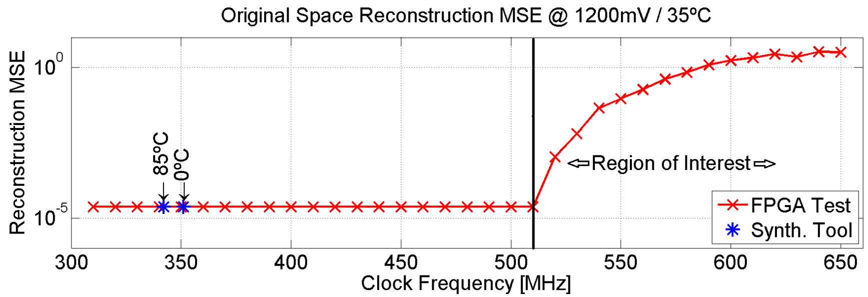
- **Summary:**
  - Benefits from spatial computation
  - Benefits from custom precision
    - Population-based MCMC (+ custom precision)
    - Generic precision optimization but bias
    - Generic precision optimization without bias

- **Device choice depends on many factors but FPGAs are cool…**
  - More freedom to exploit the characteristics of the application, interaction between algorithmic design and algorithmic implementation
  - Frequently (but not always) faster than GPUs
  - More power efficient

Imperial College London

What we plan to do

- **Closer collaboration with statistics community:**
  - Which are the most promising methods? SMC^2, Firefly MC, Sequential Quasi-MC,…
  - Which are the really demanding applications? Why?
- **Working with large datasets (Big Data):**
  - Keeping data inside the chip for as long as possible is critical for performance…
  - Do not move the data around ➔ consumes power
- **Computing with unreliable components:**
  - Devices will become less reliable
  - Clock them beyond the safe frequency



Back to the Future...

Is FPGAs the future of statistical computing?

Probably not

Heterogeneous Systems
(multi-core with a GPU and FPGA-like in the same chip)

*We need tools, libraries and MCMC algorithms **aware** of hardware*
We need to work on it
(We := Hardware Designers, Software Programmers, Statisticians)

The future will be interesting

Imperial College London

Questions?

Team working on this topic:

- Grigoris Mingas
- Shuanglong Liu
- Stelios Venieris

http://cas.ee.ic.ac.uk/people/ccb98/

ccb98@ic.ac.uk