Massively Parallel Population-Based Monte Carlo Methods with Many-Core Processors

Anthony Lee

University of Oxford, Department of Statistics & Oxford-Man Institute of Quantitative Finance

MIR@W Monte Carlo Methods
7th March, 2011

Joint work with Chris Yau (Oxford), Mike Giles (Oxford), Arnaud Doucet (UBC) & Chris Holmes (Oxford)

1. Computational Statistics
2. Graphics Cards as Many-Core Processors
3. Example 1: Mixture Model
   - Population-Based MCMC
   - Sequential Monte Carlo Samplers
4. Example 2: Factor Stochastic Volatility
   - Particle Filtering
5. Remarks

1 Computational Statistics

2 Graphics Cards as Many-Core Processors

3 Example 1: Mixture Model
- Population-Based MCMC
- Sequential Monte Carlo Samplers

4 Example 2: Factor Stochastic Volatility
- Particle Filtering

5 Remarks

Trends

- Computation is central to much of statistics.
- Increased computational power has allowed us to analyze larger datasets and consider more complex models for our data.
- Advances in Bayesian statistics, in particular, have gone hand-in-hand with advances in computation.
- The 20th Century saw single-threaded computational power increase exponentially.
- At the beginning of the 21st Century, it looks like we will see processors with many cores instead of processors with faster cores.
- We need parallel algorithms to take advantage of emerging technology!

Monte Carlo Methods

- Many statistical problems are computational in nature.
- Here, we are primarily interested in estimating expected values:
- Given a test function $\phi : \mathcal{X} \to \mathbb{R}$ and a density $\pi : \mathcal{X} \to \mathbb{R}_+$, we want to estimate

$$I \stackrel{\text{def}}{=} E_\pi[\phi(\mathbf{X})] = \int_{\mathcal{X}} \phi(\mathbf{x})\pi(\mathbf{x})d\mathbf{x}$$

- In Bayesian statistics, most quantities of interest have this form.
  - for example, $\pi(\cdot)$ may represent the posterior distribution on a parameter x
- With samples $\{\mathbf{x}^{(i)}\}_{i=1}^N \stackrel{\text{iid}}{\sim} \pi$, the Monte Carlo estimate is

$$\hat{I}_{MC} \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} \phi(\mathbf{x}^{(i)})$$

- Unfortunately, we often have no way of sampling according to π

Two General Approaches to Sampling

1. Construct an ergodic, π-stationary Markov chain sequentially (MCMC).
   - Once the chain has converged, use the dependent samples to estimate I.
   - Easy to formulate, but the rate of convergence is often an issue.
2. Sample according to some other density γ and weight the samples accordingly (Importance Sampling).
   - Use the weighted samples to estimate I:

$$\hat{I}_{IS} \overset{\text{def}}{=} \sum_{i=1}^{N} W^{(i)} \phi(\mathbf{x}^{(i)}), \; W^{(i)} = \frac{w(\mathbf{x}^{(i)})}{\sum_{j=1}^{N} w(\mathbf{x}^{(j)})} \text{ and } w(\mathbf{x}^{(i)}) = \frac{\pi^*(\mathbf{x}^{(i)})}{\gamma^*(\mathbf{x}^{(i)})}$$

   - Simple conceptually, but it is difficult to come up with γ such that the asymptotic variance $C(\phi, \pi, \gamma)/N$ is reasonable.

Parallel Computation in Bayesian Statistics

- Parallel computation is not new to statistics.
- Many algorithms are trivially parallelizable on clusters or distributed systems.
- The focus in this talk is on a different architecture that provides a number of distinct advantages.
- In particular, processors share both memory and instructions.

1 Computational Statistics

2 Graphics Cards as Many-Core Processors

3 Example 1: Mixture Model
- Population-Based MCMC
- Sequential Monte Carlo Samplers

4 Example 2: Factor Stochastic Volatility
- Particle Filtering

5 Remarks

Graphics Cards Characteristics: eg. NVIDIA GTX 280

- 30 multiprocessors, each can support 1024 active threads.
  - each multiprocessor has 8 ALU's $\approx$ 240 processors per card
  - you can think of a thread as being executed by a virtual processor
- 1GB of very fast on-card memory (low latency and high bandwidth).
- Relatively inexpensive at around £200.
- Easy to install.
- Dedicated and local.
- Can be more energy-efficient than a cluster.
- Becoming easier to program.

View of the link between host and graphics card

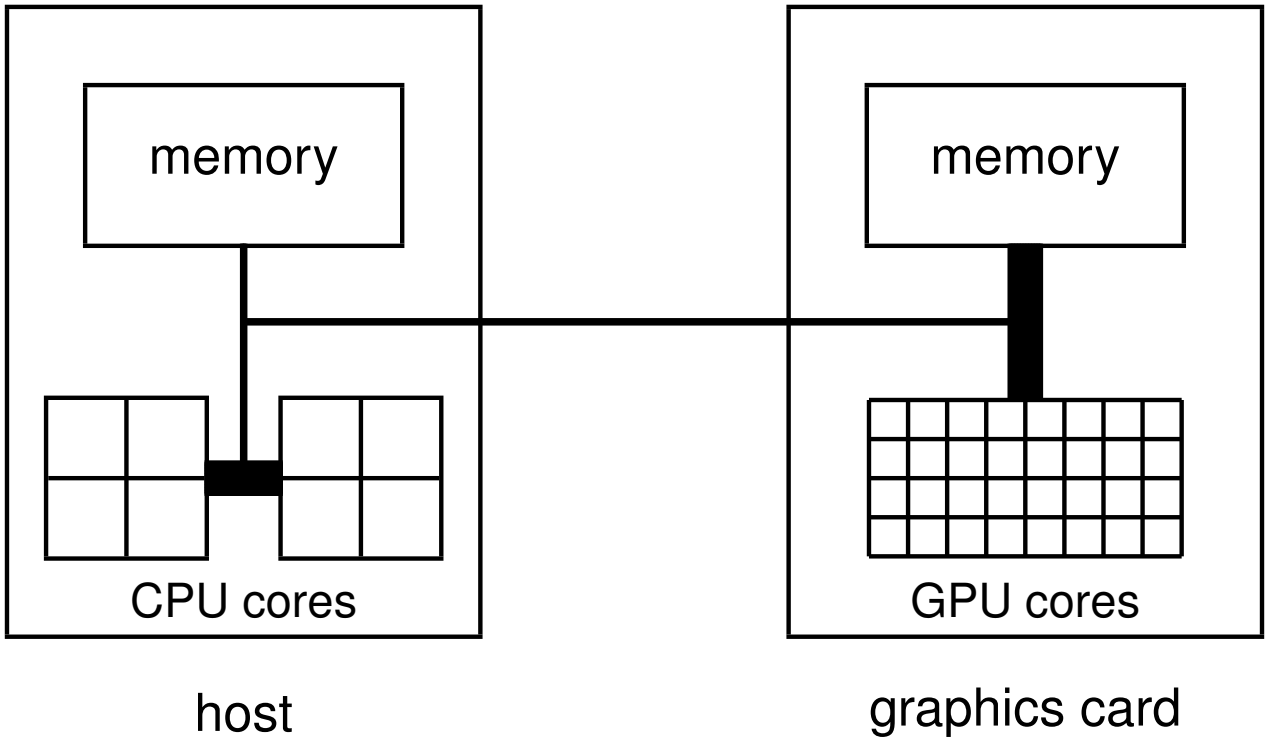


Figure: Link between host and graphics card. The thicker lines represent higher data bandwidth while the squares represent processor cores.

What's The Catch?

- GPUs have devote more transistors to arithmetic logic units and less to caches and flow control.
  - Less general purpose.
  - Effective with data-parallel computation with high arithmetic intensity.
  - Architecture is known as Single Instruction, Multiple Data (SIMD).
- We use NVIDIA cards with CUDA, their interface to compliant graphics cards.
  - An extension of the C programming language.
    - $\approx$ 4 weeks take up time
  - Users code 'kernels', which are executed in parallel on the GPU based on a unique thread identifier.
  - Further conditional branching is possible, but hurts performance.

What is SIMD?

- Single Instruction, Multiple Data.
- Several processors share flow control, ie. one controller tells all processors to execute the same instruction.
- This means that algorithms must exhibit instruction-level parallelism!
- Typically this architecture is exploited by algorithms in which we compute the same function for many pieces of data.

Example of C-like Syntax

Kernel:

```
__global__ void importance_sample(int N, float* d_array,
    float* d_array_out) {
    // thread id = threads per block * block id + thread id within block
    const int tid = blockDim.x * blockIdx.x + threadIdx.x;
    // total number of threads = threads per block * number of blocks
    const int tt = blockDim.x * gridDim.x;
    int i;
    float w, x;
    for (i = tid; i < N; i += tt) {
        x = d_array[i];
        w = target_pdf(x) / proposal_pdf(x);
        d_array_out[i] = phi(x) * w;
    }
}
```

Calling the kernel in host code:

```
...
importance_sample<<<64,128>>>(N, d_array, d_array_out);
...
```

Taking Advantage

- We sought to investigate methods amenable to computation on this architecture.
- Two such methods are from the class of population-based Monte Carlo methods
  - Population-based MCMC
  - Sequential Monte Carlo

Population-Based Monte Carlo Methods

- A class of Monte Carlo methods that is particularly well-suited to parallelization in shared memory systems.
- Population-based methods include SMC samplers, particle filters and population-based MCMC.
- The idea in all of these is to use a collection of samples generated at each iteration of an algorithm to approximate an artificially constructed distribution.
- We introduce auxiliary distributions which aid in the sampling of some complex target density $\pi(\cdot)$
- We will focus on these today but many other applications of GPUs in stochastic simulation are possible.

1 Computational Statistics

2 Graphics Cards as Many-Core Processors

3 Example 1: Mixture Model
- Population-Based MCMC
- Sequential Monte Carlo Samplers

4 Example 2: Factor Stochastic Volatility
- Particle Filtering

5 Remarks

Example 1: Mixture Model

- Let $\mathbf{y} = y_{1:m}$ be a vector of i.i.d. observations where $y_j \in \mathbb{R}$ for $j \in \{1, \ldots, m\}$
- Univariate Gaussian mixture model with k components (f is Gaussian pdf):

$$p(\mathbf{y}|\mu_{1:k}, \sigma_{1:k}, w_{1:k-1}) = \prod_{j=1}^{m} \sum_{i=1}^{k} w_i f(y_j|\mu_i, \sigma_i)$$

- For simplicity, assume $k = 4$, $w_i = w = 0.25$, $\sigma_i = \sigma = 0.55$, $i \in \{1, \ldots, 4\}$ are known and let $p(\mu_{1:k})$ denote the uniform density on $[-10, 10]^k$.
- Invariance of the posterior to permutations of the labels of the parameters gives it $k! = 24$ symmetric modes:

$$p(\boldsymbol{\mu}|\mathbf{y}) \propto p(\mathbf{y}|\boldsymbol{\mu})\mathbb{I}(\boldsymbol{\mu} \in [-10, 10]^4)$$

Mixture Model Density

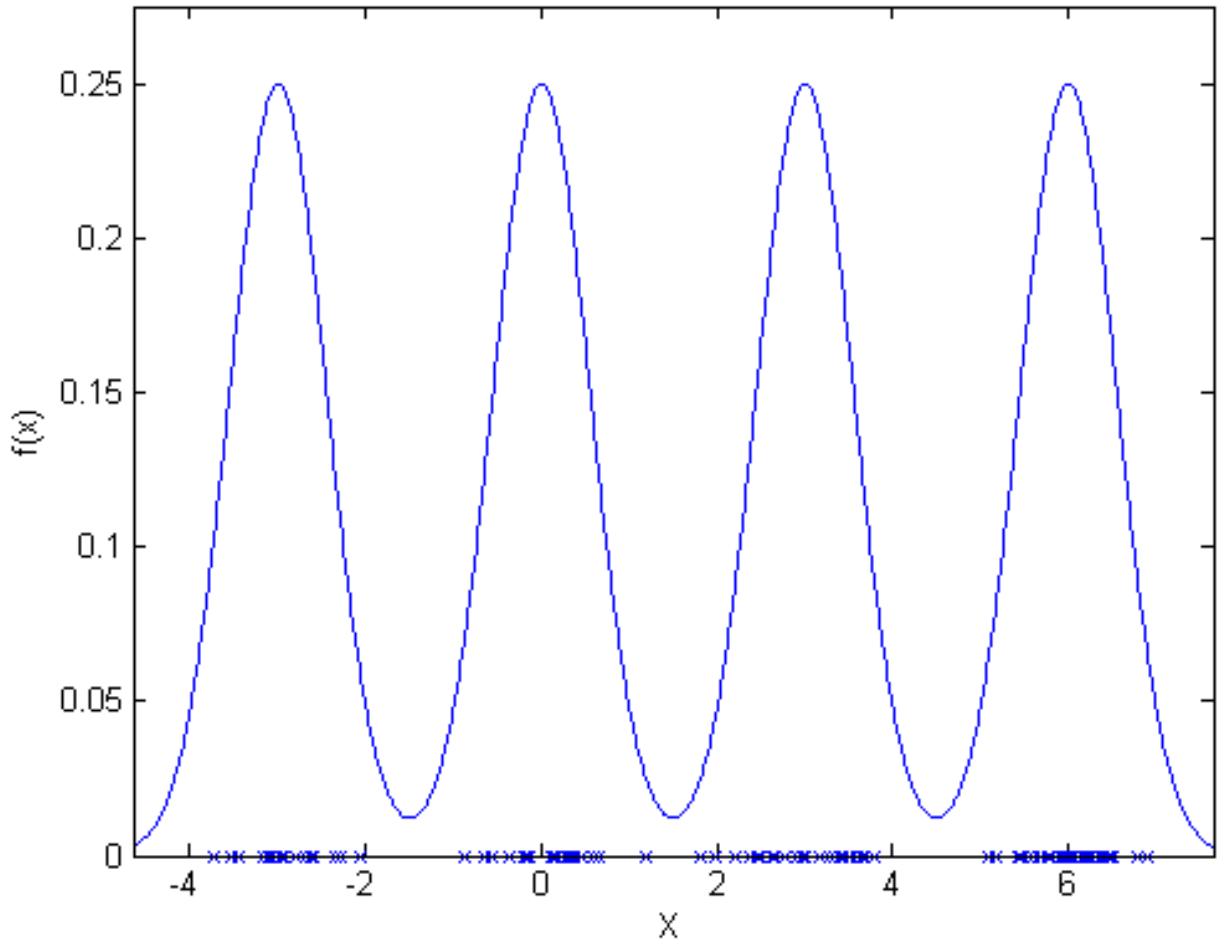


Figure: $p(y|\mu_{1:4}, \sigma)$

What the posterior looks like (marginally)

- We simulate $m = 100$ data points with $\mu = (-3, 0, 3, 6)^T$.

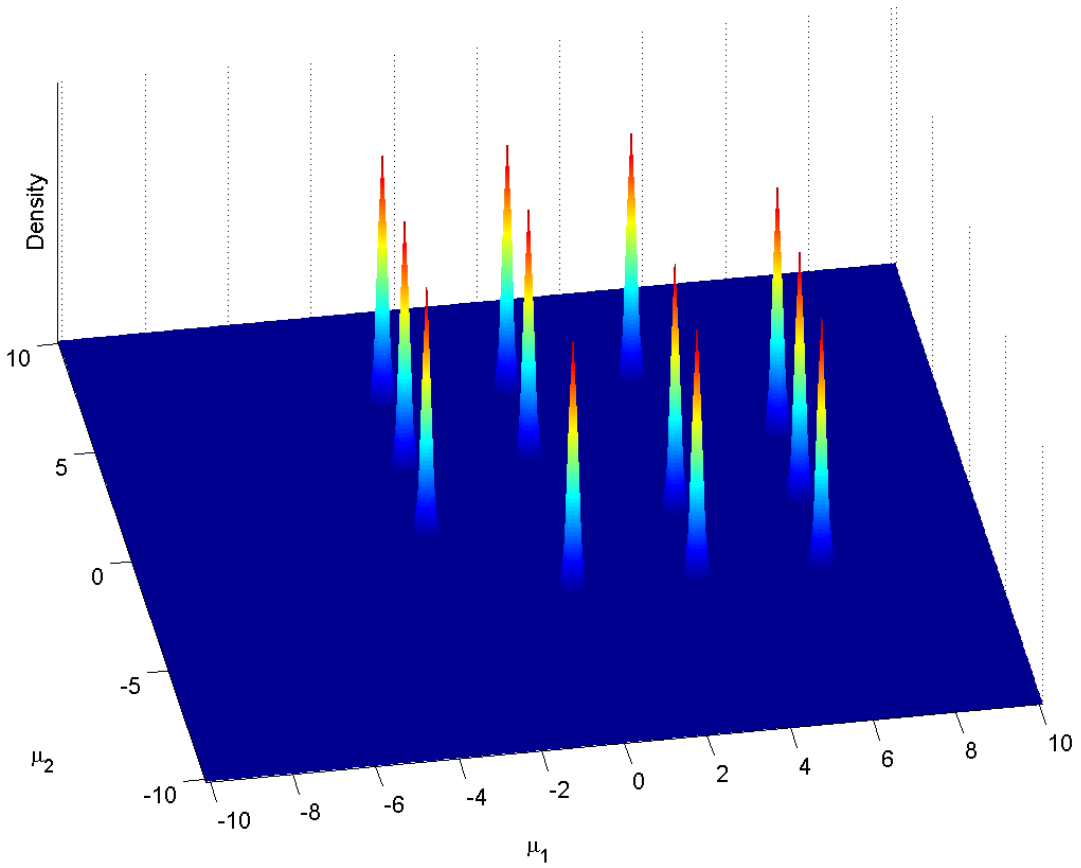


Figure: Marginal posterior density $p(\mu_{1:2}|\mathbf{y})$ on $[-10, 10]^2$

Population-Based MCMC

- First note that conventional Metropolis-Hastings random walk MCMC on $\pi(\boldsymbol{\mu}) = p(\boldsymbol{\mu}|\mathbf{y})$ doesn't converge.
- In population-based MCMC, we do MCMC on an *extended* target distribution $\bar{\pi}$

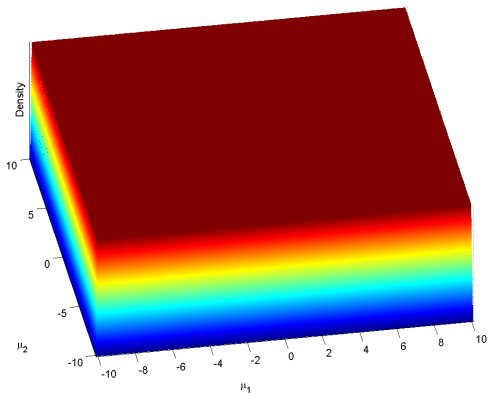
$$\bar{\pi}(\boldsymbol{\mu}_{1:M}) \stackrel{\text{def}}{=} \prod_{i=1}^{M} \pi_i(\boldsymbol{\mu}_i)$$

- For this example, we choose $\pi_i(\boldsymbol{\mu}_i) = \pi(\boldsymbol{\mu}_i)^{\beta_i}$ and $\beta_i = (i/M)^2$.
- $\boldsymbol{\mu}_{1:M-1}$ are auxiliary variables.
- All moves must leave $\bar{\pi}$ invariant.
- We parallelize across chains.
- Both single chain moves and interactions between chains can be done in parallel.

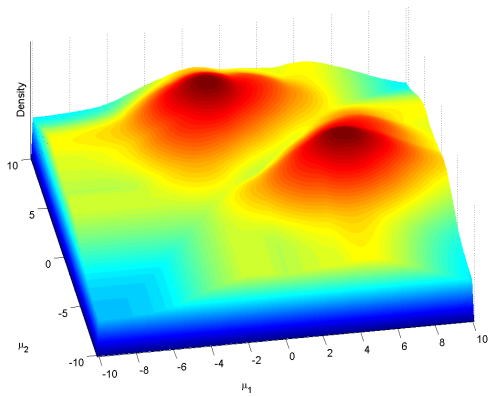
Population-Based MCMC Details

- There are two types of moves:
  1. In parallel, each chain i performs an MCMC move targetting π_i.
  2. In parallel, adjacent chains i and $i+1$ perform an MCMC 'exchange' move targetting $\pi_i \pi_{i+1}$.
- A simple exchange move at time n proposes to swap the values of the two chains and has acceptance probability $\min\{1, \frac{\pi_i(x_{i+1}^{(n)})\pi_{i+1}(x_i^{(n)})}{\pi_i(x_i^{(n)})\pi_{i+1}(x_{i+1}^{(n)})}\}$.
- In order to ensure (indirect) communication between all the chains, we pick the exchange partners at each time with equal probability from $\{\{1,2\},\ldots,\{M-1,M\}\}$ and $\{\{2,3\},\ldots,\{M-2,M-1\}\}$.

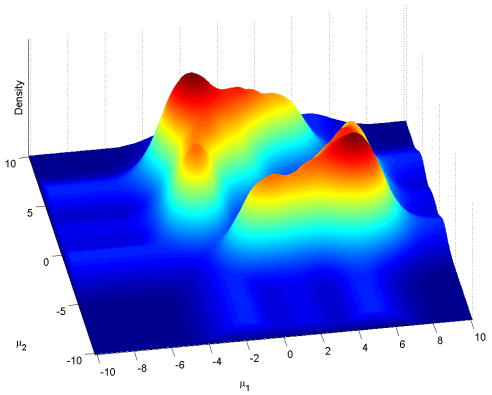
Visualizing the Auxiliary Distributions I



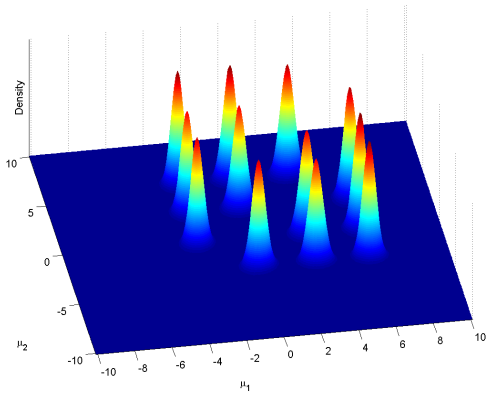
(a) $\beta = 0$



(b) $\beta = 0.001$



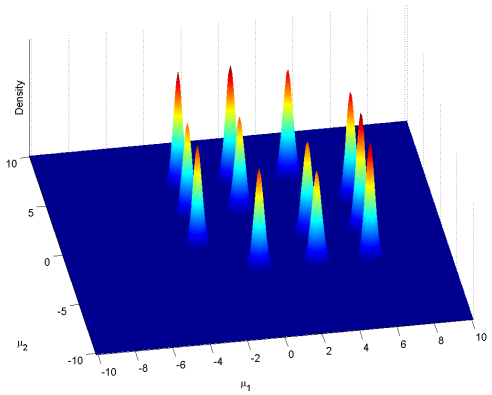
(c) $\beta = 0.01$



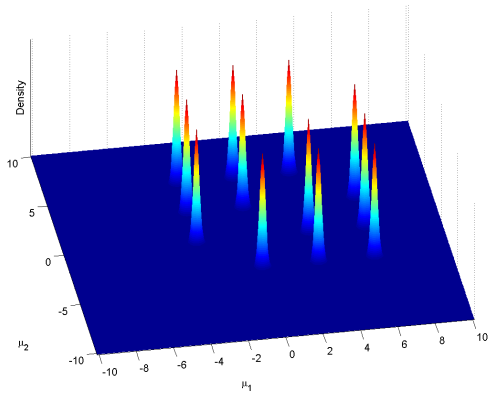
(d) $\beta = 0.1$

Figure: Marginal posterior density $p(\mu_{1:2}|\mathbf{y})^{\beta}$ on $[-10, 10]^2$

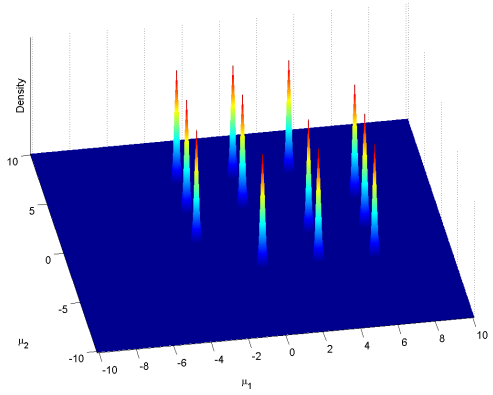
Visualizing the Auxiliary Distributions II



(a) $\beta = 0.2$



(b) $\beta = 0.5$



(c) $\beta = 1$

Figure: Marginal posterior density $p(\mu_{1:2}|\mathbf{y})^{\beta}$ on $[-10, 10]^2$

Population-Based MCMC Results

Table: Running times for the Population-Based MCMC Sampler for various numbers of chains M. We sample $N = 8192$ points from chain M.

M	CPU (mins)	GTX280 (secs)	Speedup
8	0.0166	1.083	0.9
32	0.0656	1.098	4
128	0.262	1.100	14
512	1.04	1.235	51
2048	4.16	1.427	175
8192	16.64	2.323	430
32768	66.7	7.729	527
131072	270.3	28.349	572

Sequential Monte Carlo Samplers

- Similarly, standard importance sampling won't work with a standard choce of γ.
- SMC is a population-based extension of importance sampling.
- In SMC, we introduce the same type of auxiliary distributions $\pi_i(\boldsymbol{\mu}_i) = \pi(\boldsymbol{\mu}_i)^{\beta_i}$ with $\beta_i = (i/M)^2$.
- Whereas in population-based MCMC we sample from the joint distribution of auxiliary variables, in SMC we construct a weighted particle approximation of each auxiliary distribution in turn using N particles.
- Particle evolution moves are parallelizable & particle interaction via resampling is somewhat parallelizable (but infrequent!).
- Unlike population-based MCMC, we fix M and vary N since we are parallelizing across particles.

Sequential Monte Carlo Sampler Details

- Very similar methodology to particle filters for state-space models but applicable to static inference problems.
- One can think of particle filters as a special case of an SMC sampler with auxiliary distributions selected via data tempering.
- In state-space models the importance weights are defined naturally on a joint density but in general one must specify a backwards transition to compute importance weights on an artificial joint density.

$$\psi_t(x_{1:t}) = \pi_t(x_t) \prod_{i=1}^{t-1} L_i(x_{i+1}, x_i)$$

- A simple but effective option in many cases is to use an MCMC step that leaves π_{i+1} invariant when moving a particle approximating π_i and specify the backwards transition to be the 'reverse' MCMC step that leaves π_i invariant.

Algorithmic Details

1. At time $t = 0$:
   - For $i = 1, \ldots, N$, sample $\mathbf{x}_0^{(i)} \sim \eta(\mathbf{x}_0)$
   - For $i = 1, \ldots, N$, evaluate the importance weights:

$$w_0(\mathbf{x}_0^{(i)}) \propto \frac{\pi_0(\mathbf{x}_0^{(i)})}{\eta(\mathbf{x}_0^{(i)})}$$

2. For times $t = 1, \ldots, T$:
   - For $i = 1, \ldots, N$, sample $\mathbf{x}_t^{(i)} \sim K_t(\mathbf{x}_{t-1}^{(i)}, \cdot)$
   - For $i = 1, \ldots, N$, evaluate the importance weights:

$$w_t(\mathbf{x}_t^{(i)}) \propto w_{t-1}(\mathbf{x}_{t-1}^{(i)}) \frac{\pi_t(\mathbf{x}_t^{(i)}) L_{t-1}(\mathbf{x}_t^{(i)}, \mathbf{x}_{t-1}^{(i)})}{\pi_{t-1}(\mathbf{x}_{t-1}^{(i)}) K_t(\mathbf{x}_{t-1}^{(i)}, \mathbf{x}_t^{(i)})}$$

   - Normalize the importance weights.
   - Depending on some criteria, resample the particles. Set $w_t^{(i)} = \frac{1}{N}$ for $i = 1, \ldots, N$.

Algorithmic Details

- For the special case where L_{t-1} is the associated backwards kernel for K_t, ie.

$$\pi_t(x_t)L_{t-1}(x_t, x_{t-1}) = \pi_t(x_{t-1})K_t(x_{t-1}, x_t)$$

the incremental importance weights simplify to

$$w_t(\mathbf{x}_t^{(i)}) \propto w_{t-1}(\mathbf{x}_{t-1}^{(i)}) \frac{\pi_t(\mathbf{x}_{t-1}^{(i)})}{\pi_{t-1}(\mathbf{x}_{t-1}^{(i)})}$$

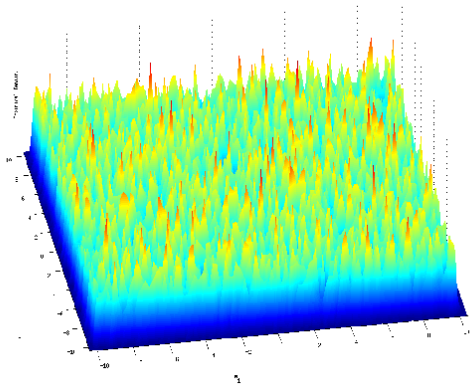
- The green steps are trivially parallelizable.
- The normalization step is a reduction operation and a divide operation.
- The resampling step involves a parallel scan.

Sequential Monte Carlo Sampler Results

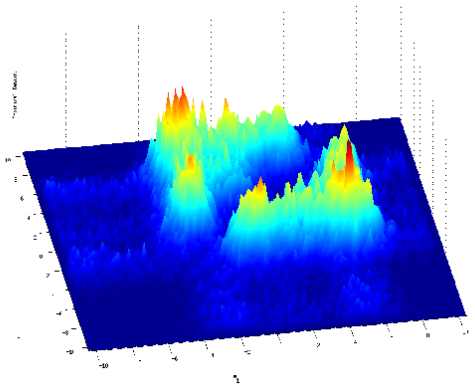
Table: Running times for the Sequential Monte Carlo Sampler for various values of N.

N	CPU (mins)	GTX280 (secs)	Speedup
8192	4.44	0.597	446
16384	8.82	1.114	475
32768	17.7	2.114	502
65536	35.3	4.270	496
131072	70.6	8.075	525
262144	141	16.219	522

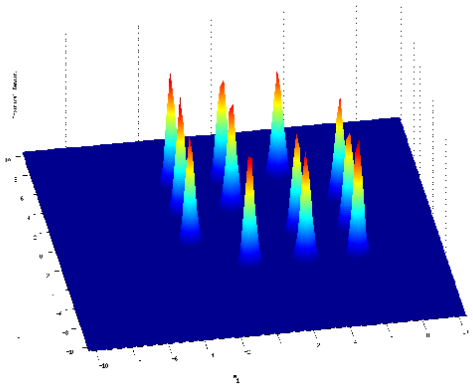
Empirical Approximations to the Auxiliary Distributions



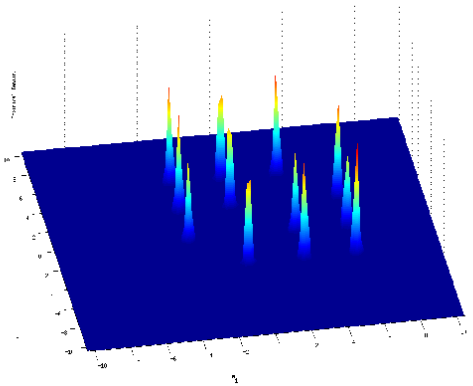
(a) $\beta = 0.000025$



(b) $\beta = 0.02$



(c) $\beta = 0.3$



(d) $\beta = 1$

Figure: Empirical approximations of the marginal posterior density $p(\mu_{1:2}|\mathbf{y})^{\beta}$ on

1 Computational Statistics

2 Graphics Cards as Many-Core Processors

3 Example 1: Mixture Model
- Population-Based MCMC
- Sequential Monte Carlo Samplers

4 Example 2: Factor Stochastic Volatility
- Particle Filtering

5 Remarks

Sequential Monte Carlo a. k. a. Particle Filtering

- We use auxiliary distributions of the form $p_t(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ in SMC.
- Again, a weighted particle approximation of each auxiliary distribution is constructed using a combination of importance sampling and resampling.
- In this case, evaluation of the importance weights does not require choosing an artificial joint density.
- Essentially the same parallelization issues as before.

Sequential Importance Sampling / Resampling (generic)

1. At time $t = 0$:
   - For $i = 1, \ldots, N$, sample $\tilde{\mathbf{x}}_0^{(i)} \sim q(\mathbf{x}_0|\mathbf{y}_0)$
   - For $i = 1, \ldots, N$, evaluate the importance weights:

$$w_0(\tilde{\mathbf{x}}_0^{(i)}) = \frac{p(\tilde{\mathbf{x}}_0^{(i)}, \mathbf{y}_0)}{q(\tilde{\mathbf{x}}_0^{(i)}|\mathbf{y}_0)} = \frac{p(\mathbf{y}_0|\tilde{\mathbf{x}}_0^{(i)})p(\tilde{\mathbf{x}}_0^{(i)})}{q(\tilde{\mathbf{x}}_0^{(i)}|\mathbf{y}_0)}$$

2. For times $t = 1, \ldots, T$:
   - For $i = 1, \ldots, N$, sample $\tilde{\mathbf{x}}_t^{(i)} \sim q(\mathbf{x}_t|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})$ and set $\tilde{\mathbf{x}}_{0:t}^{(i)} \stackrel{\text{def}}{=} (\mathbf{x}_{0:t-1}^{(i)}, \tilde{\mathbf{x}}_t^{(i)})$
   - For $i = 1, \ldots, N$, evaluate the importance weights:

$$w_t(\tilde{\mathbf{x}}_{0:t}^{(i)}) = w_{t-1}^{(i)} \frac{p(\mathbf{y}_t|\tilde{\mathbf{x}}_t^{(i)})p(\tilde{\mathbf{x}}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{q(\tilde{\mathbf{x}}_t^{(i)}|\mathbf{y}_t, \mathbf{x}_{t-1}^{(i)})}$$

   - Normalize the importance weights.
   - Depending on some criteria, resample the particles. Set $w_t^{(i)} = \frac{1}{N}$ for $i = 1, \ldots, N$.

Example 2: Factor Stochastic Volatility

- $\mathbf{x}_t \in \mathbb{R}^K$ is a latent factor, $\mathbf{y}_t \in \mathbb{R}^M$ is a vector of asset values.
- $\mathbf{B}$ is the $M \times K$ factor loading matrix.
- Model:

$$\begin{aligned} \mathbf{y}_t &\sim N(\mathbf{B}\mathbf{f}_t, \Psi), & \Psi &\stackrel{\text{def}}{=} \text{diag}(\psi_1, \ldots, \psi_M) \\ \mathbf{f}_t &\sim N(\mathbf{0}, \mathbf{H}_t), & \mathbf{H}_t &\stackrel{\text{def}}{=} \text{diag}(\exp(\mathbf{x}_t)) \\ \mathbf{x}_t &\sim N(\Phi\mathbf{x}_{t-1}, \mathbf{U}), & \Phi &\stackrel{\text{def}}{=} \text{diag}(\phi_1, \ldots, \phi_K) \end{aligned}$$

- We want samples from $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T})$ given an initial distribution on $\mathbf{x}_0$.
- This is a $T \times K$-dimensional problem!

FSV Results

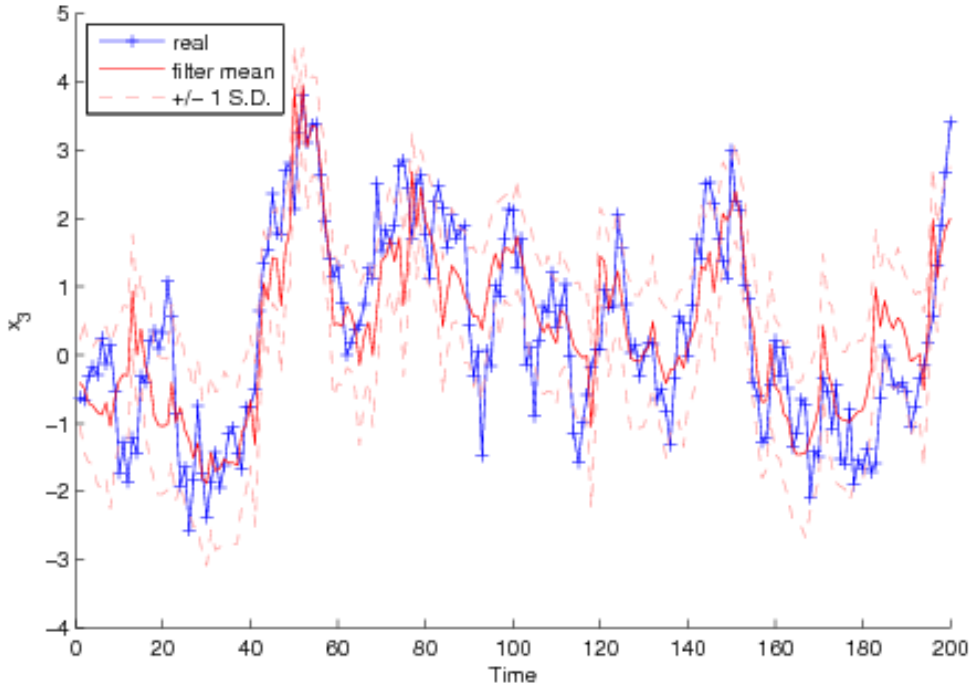


Figure: Estimated and real values of x_3 from time 0 to 200 given $\mathbf{y}_{1:200}$

Particle Filtering Results

Table: Running time (in seconds) for the particle filter for various values of N.

N	CPU	GTX280	Speedup
8192	2.167	0.082	26
16384	4.325	0.144	30
32768	8.543	0.249	34
65536	17.425	0.465	37
131072	34.8	0.929	37

The main reasons for the decrease in speedup are

- Increased space complexity of each thread.
- Increased number of times we have to resample.

1 Computational Statistics

2 Graphics Cards as Many-Core Processors

3 Example 1: Mixture Model
- Population-Based MCMC
- Sequential Monte Carlo Samplers

4 Example 2: Factor Stochastic Volatility
- Particle Filtering

5 Remarks

Remarks

- The speedups have practical significance.
- Arithmetic intensity is important.
- There is a roughly linear penalty for the space complexity of each thread.
- Emerging many-core technology is likely to have the same kinds of restrictions.
- There is a need for methodological attention to this model of computation.

  For example, SMC sampler methodology can be more suitable to parallelization when the number of auxiliary distributions one wants to introduce is not very large.
- There are many other algorithms that will benefit from this technology.
- `http://www.oxford-man.ox.ac.uk/gpuss`