Parallel Implementation of Particle MCMC Methods on a GPU [⋆]

Soren Henriksen [*] **Adrian Wills** [*] **Thomas B. Schön** [**] **Brett Ninness** [*]

[*] *School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan, NSW, 2308, Australia (Tel: +61 2 49216028; e-mail: {soren, Adrian.Wills, Brett.Ninness}@newcastle.edu.au).*
[**] *Division of Automatic Control, Linköping University, SE-581 83 Linköping, Sweden (Tel: +46 13 281373; e-mail: schon@isy.liu.se).*

Abstract: This paper examines the problem of estimating the parameters describing system models of quite general nonlinear and multi-variable form. The approach is a computational one in which quantities that are intractable to evaluate exactly are approximated by sample averages from randomized algorithms. The main contribution is to illustrate the viability and utility of this approach by examining how high computational loads can be simply managed using commodity hardware. The proposed algorithms and solution architectures are profiled on concrete examples.

Keywords: Nonlinear dynamical systems, nonlinear estimation, Particle filter, Markov Chain Monte Carlo, parallel computation, Graphics Processing Unit.

1. INTRODUCTION

This work considers the estimation of general but nevertheless parametrized nonlinear state-space models of the form

$$x_{t+1} \sim f_\theta(x_{t+1} \mid x_t), \tag{1a}$$
$$y_t \sim h_\theta(y_t \mid x_t). \tag{1b}$$

Here $\theta \in \mathbf{R}^d$ denotes the unknown parameterization that is to be estimated based on measurements $y_{1:N} \triangleq \{y_1, \ldots, y_N\}$, and $x_t \in \mathbf{R}^n$ denotes the underlying state vector.

This sort of estimation problem is a central one and has received significant research attention over the past several decades, mainly with regard to specific instances of the general structure (1) such as for example linear, or Hammerstein–Wiener cases.

Recently, there has been a surge of interest in addressing the general structure (1) within the statistics, econometrics and wider scientific communities [Andrieu et al., 2010, Flury et al., 2011, Jones et al., 2010]. Central aspects of this work are that a Bayesian approach is employed, and that the potentially complicated issue of computing functions of posterior distributions is attacked via computing approximations based on randomized algorithms.

To be more specific, consider the following nonlinear state-space model, which has become a benchmark since its introduction in Nett et al. [1978],

$$x_{t+1} = \theta_0 x_t + \theta_1 \frac{x_t}{1+x_t^2} + \theta_2 \cos(1.2t) + \theta_3 w_t, \tag{2a}$$
$$y_t = \theta_4 x_t^2 + \theta_5 e_t, \tag{2b}$$
$$\begin{bmatrix} w_t \\ e_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \tag{2c}$$

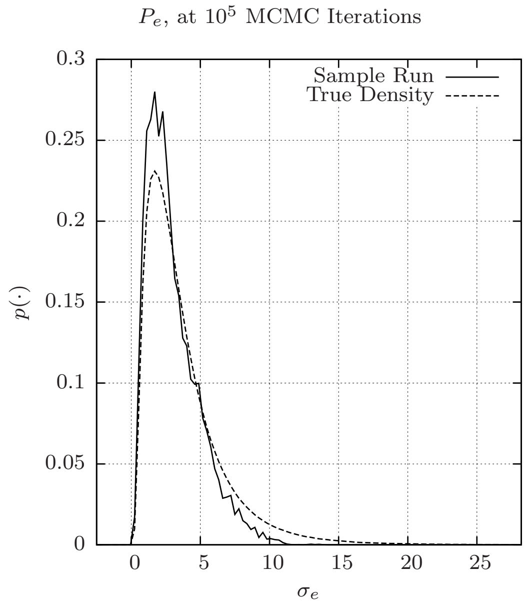

Fig. 1. *Plot of* $p(\theta_5 \mid y_{1:N})$ *for the nonlinear example* (2).

where the true parameters are

$$\theta^\star = \left(0.5, 25, 8, 1.0, 0.05, \sqrt{10}\right)^T. \tag{3}$$

This paper will consider forming an estimate of θ via computation of the posterior density $p(\theta \mid y_{1:N})$ or functions of this density, such as the expectation $E\{\theta \mid y_{1:N}\}$. Precise expressions for these quantities are not available, so instead approximations will be used, such as illustrated in Figure 1.

This illustrates an approximation (solid line) based on a randomized algorithm of the posterior marginal density $p(\theta_5 \mid y_{1:N})$ of the last element of (3) which is the standard deviation σ_e of the measurement noise term e_t.

⋆ This work was supported by: the Australian Research Council through their Discovery Project Program; and CADICS, a Linneaus Center funded by the Swedish Research Council and the project Calibrating Nonlinear Dynamical Models (Contract number: 621-2010-5876) also funded by the Swedish Research Council.

In Figure 1 this approximation is compared with the posterior marginal shown as a dashed line. An essential point is that this "truth" is computed using the same randomized algorithm as for the solid line, but having run for (perhaps many) more iterations until convergence has occurred.

That is, while information measures such as posteriors in Figure 1 for general nonlinear structures such as (1) (or even linear ones) have traditionally been avoided based on the impossibility of analytical calculation, they can now be computed due to the modern availability of cheap and powerful computing resources.

This paper addresses this issue, with a focus on how Markov Chain Monte Carlo (MCMC) methods [Robert and Casella, 2004, 2011] and Sequential Monte Carlo (SMC) methods (A.K.A. particle filtering) [Gordon et al., 1993, Doucet and Johansen, 2011] may be implemented and combined on cheap, but high performance "Graphics Processing Unit" (GPU) cards. Related work on this topic includes the papers by Hendeby et al. [2010], Lee et al. [2010].

These GPU's are inexpensive because they service a high volume consumer demand for interactive gaming, which requires high speed numerical computation for 3D-projected graphics. As such these GPU's have evolved to provide hundreds of parallel processing cores, each clocked in the gigahertz range.

As illustrated in this paper, these GPU units can also be employed to implement computationally demanding system identification methods. To achieve this it is necessary to recognize that GPU architectures are based on a SIMD (Single Instruction Multiple Data) machine, and so at any one time many cores must execute the identical instruction, but on different data.

To effectively make use of this computational power, the entire algorithm needs to be structured in an appropriate parallel fashion. Due also to memory bandwidth limitations, it is important to also to have many more process threads ready to execute than what there are processor cores. Consequently, for best performance the algorithm must be structured for the order of thousands of parallel computation streams, and this motivates considering new approaches for the implementation of randomized algorithms.

This paper studies these issues, profiles a particular solution strategy, and by doing so illustrates the potential of what is possible by moving to the employment of GPU-based parallel computing architectures.

2. RANDOMIZED ALGORITHMS

The essential approach of this paper is to consider how to develop a random number generator to deliver realisations for the posterior of the parameters θ given the data $y_{1:N}$; viz.

$$\theta_k \sim p(\theta \mid y_{1:N}), \tag{4}$$

and then use M realisations to compute approximations for arbitrary functions $f : \theta \mapsto \mathbf{C}^m$ with dimension m also arbitrary via

$$E\{f(\theta) \mid y_{1:N}\} \approx \frac{1}{M}\sum_{k=1}^{M} f(\theta_k). \tag{5}$$

A very simple and general method for achieving this is the Metropolis–Hastings algorithm defined below, which dates back to the 1950's [Robert and Casella, 2004] and provides realisations such that via the strong law of large numbers, equality in the approximation (5) occurs in the infinite limit:

$$\lim_{M\to\infty} \frac{1}{M}\sum_{k=1}^{M} f(\theta_k) = E\{f(\theta) \mid y_{1:N}\} \quad \text{with probability 1.} \tag{6}$$

Algorithm 1 *Metropolis–Hastings Sampler*

```
1: Initialise $\theta_0$ at some value such that $p(\theta_0 \mid y_{1:N}) > 0$ and set $k = 1$;
2: At iteration $k$, consider a candidate value $\xi_k$ for $\theta_k$ which is drawn from a ***proposal*** density $\gamma(\xi_k \mid \theta_{k-1})$. That is, find a possible realisation for $\theta_k$ as
       $\xi_k \sim \gamma(\cdot \mid \theta_{k-1});$   (7)
3: Compute the acceptance probability
       $\alpha(\xi_k \mid \theta_{k-1}) = \frac{p(\xi_k \mid y_{1:N})}{p(\theta_{k-1} \mid y_{1:N})} \cdot \frac{\gamma(\theta_{k-1} \mid \xi_k)}{\gamma(\xi_k \mid \theta_{k-1})};$   (8)
4: Accept the proposed $\xi_k$ and set $\theta_k = \xi_k$ with probability $\alpha(\xi_k \mid \theta_{k-1})$, otherwise leave $\theta_k$ unchanged by setting $\theta_k = \theta_{k-1}$;
4: Increment $k$ and return to step 2.
```

The key computational difficulty is the evaluation of

$$p(\theta \mid y_{1:N}) = c \cdot p(\theta) \prod_{t=1}^{N} p(y_t \mid y_{1:t-1}, \theta), \tag{9}$$

where c is a θ-independent constant. In the linear system case, the prediction density $p(y_t \mid y_{1:t-1}, \theta)$ can be simply computed using a Kalman filter. In the general nonlinear case (1) the equivalent time and measurement update equations are impossible to compute exactly.

To address this difficulty, this paper employs an SMC (particle filter) approximation of $p(y_t \mid y_{1:t-1}, \theta)$. At first reading, this appears to be an ad-hoc convenience.

However, a major new result by Andrieu et al. [2010], is that the employment of such a randomized SMC-based approximation of $p(\theta|y_{1:N})$ in (8) preserves the convergence property (6). The resulting algorithm is dubbed in Andrieu et al. [2010] the "Particle Marginal Metropolis-Hastings" (PMMH) method.

It is the focus of what follows in this paper, and hence formally defined as follows in Algorithm 2. The two key steps in the implementation of this algorithm are:

(1) The generation of samples from $p_\theta(x_{1:N} \mid y_{1:N})$;
(2) The computation of an estimate of the likelihood $p_\theta(y_{1:N})$.

To address the first implementation issue - item 1, this paper will employ an SMC algorithm targeting the density $p(x_{1:t} \mid y_{1:t})$, using $q_t(x_t \mid x_{1:t}) = f(x_t \mid x_{t-1})$ as proposal density and resampling at every time step. This particular SMC algorithm is commonly referred to as the bootstrap particle filter in the literature, first introduced by Gordon et al. [1993]. This is defined in Algorithm 3 below where it should be noted that whenever the index i is used, it is shorthand for "for all $i = 1, \ldots, L$".

To address the second implementation issue in item 2 above, note that by Bayes' rule, the likelihood $p_\theta(y_{1:N})$ can be written as

Algorithm 2 *Particle Marginal Metropolis-Hastings (PMMH) Sampler*

```
1: Initialize, $r = 0$:
2: Set $\theta(0)$ arbitrary.
3: Use an SMC algorithm targeting $p(x_{1:N} \mid y_{1:N})$ to sample $X_{1:N}(0) \sim \widehat{p}_{\theta(0)}(x_{1:N} \mid y_{1:N})$ and to compute an estimate $\widehat{p}_{\theta(0)}(y_{1:N})$ of the likelihood.
4: while $r \geq 1$ do
5:     Sample $\theta' \sim q\{\theta \mid \theta(r-1)\}$.
6:     Use an SMC algorithm targeting $p_{\theta'}(x_{1:N} \mid y_{1:N})$ to sample $X'_{1:N} \sim \widehat{p}_{\theta'}(x_{1:N} \mid y_{1:N})$ and to compute an estimate $\widehat{p}_{\theta'}(y_{1:N})$ of the likelihood.
7:     Compute the acceptance probability
           $$\alpha = \frac{\widehat{p}_{\theta'}(y_{1:N})p(\theta')}{\widehat{p}_{\theta(r-1)}(y_{1:N})p(\theta(r-1))}\frac{q\{\theta(r-1) \mid \theta'\}}{q\{\theta' \mid \theta(r-1)\}} \quad (10)$$
8:     With probability $a$ set $\theta(r) = \theta', X_{1:N}(r) = X'_{1:N}, \widehat{p}_{\theta(r)}(y_{1:N}) = \widehat{p}_{\theta'}(y_{1:N})$, otherwise, set $\theta(r) = \theta(r-1), X_{1:N}(r) = X_{1:N}(r-1), \widehat{p}_{\theta(r)}(y_{1:N}) = \widehat{p}_{\theta(r-1)}(y_{1:N})$.
9: end while
```

Algorithm 3 *Bootstrap Particle Filter*

```
1: Initialize, $t = 1$:
2: Sample $X_0^i \sim q_\theta(x_0)$.
3: for $t = 1 : N$ do
4:     Sample $X_t^i \sim f(x_t \mid \widetilde{X}_{t-1})$ and set $X_{1:t}^i = \{\widetilde{X}_{1:t-1}^k, X_t^i\}$.
5:     Compute the importance weights
           $$W_t^i = \frac{h_\theta(y_t \mid X_t^i)}{\sum_{j=1}^{L} h_\theta(y_t \mid X_t^j)}. \quad (11)$$
6:     Resample $\{W_t^i, X_{1:t}^i\}$ to obtain $L$ equally weighted particles $\{1/L, \widetilde{X}_{1:t}^i\}$.
7: end for
```

$$p_\theta(y_{1:N}) = \prod_{t=1}^{N} p_\theta(y_t \mid y_{1:t-1}), \tag{12}$$

and via the law of total probability

$$p(y_t \mid y_{1:t-1}) = \int p(y_t \mid x_t)p(x_t \mid y_{1:t-1})\,\mathrm{d}x_t. \tag{13}$$

The SMC algorithm produces realisations

$$X_t^i \sim p(x_t \mid y_{1:t-1}) \tag{14}$$

and hence via the model (1) and the strong law of large numbers, we can approximate

$$\widehat{p}(y_t \mid y_{1:t-1}) \approx \frac{1}{L}\sum_{i=1}^{L} g(y_t \mid X_t^i) \tag{15}$$

with equality, with probability one, as the number of particles $L \to \infty$.

As already mentioned, but repeated now for emphasis and clarity, the very surprising and important result in Andrieu et al. [2010] is that even though (15) is an approximation for finite number of particles L, the convergence result (6) still holds as the Metropolis–Hastings method is run for more iterations M, but with L fixed.

While this is of theoretical interest, it also has important practical consequences. The algorithm just proposed, involves that for each of the M Metropolis–Hastings iterations, a particle filter with L particles needs to be evaluated for each of N data samples.

This implies an $M \times L \times N$ computational load, and the result of Andrieu et al. [2010] is important in that it establishes that even with L fixed, the Metropolis–Hastings method will converge with increasing M to the true posterior density with respect to the available length N finite data $y_{1:N}$.

Nevertheless, the associated computational load is still high, and a contribution of this paper is to examine how this may be efficiently implemented using GPU-based computation.

3. PARALLEL GPU IMPLEMENTATION

MCMC and particle filters are both inherently quite parallelisable algorithms. In the case of MCMC, instead of one long Markov Chain, a number of shorter parallel chains may be implemented. Apart from some loss of efficiency in reaching chain "burn-in" for all chains, the algorithm performance per kernel evaluation remains largely static. This may be achieved as the individual chains may operate largely in independence of one another.

Similarly, a particle filter appears quite suitable for implementation in parallel hardware. There are many particles that conceptually exist in parallel, which each need to be propagated along a number of time steps.

There are three main components of the computation for each time step of a particle filter and they are propagation, weighting, and resampling. The first two of these, the propagation and weighting are quite straightforward to implement in a parallel manner. With one thread per particle, the values for each particle may be computed at once with a SIMD machine.

For propagation, each state moves independently according to the model state update equation. In the example nonlinear system posed in the introduction, the state update described in (2) implies that for the i-th particle,

$$x_{t+1}^i = \theta_1 x_t^i + \theta_2 \frac{x_t^i}{1 + (x_t^i)^2} + \theta_3 u_t + \theta_4 w_t^i. \tag{16}$$

The parameter vector $\{\theta_1, \theta_2, \theta_3, \theta_4\}$, and the input signal value u_t, are common to all particles in the filter. Where this filter is used as part of an MCMC algorithm with multiple parallel chains. There will be a group of particles for each chain, where each group will operate with a different parameter vector, according to the state of the associated Markov chain.

The noise term w_t^i, is simply a realisation drawn from the state noise density, with each particle receiving a different realisation.

The weighting step of the particle filter algorithm involves computing the likelihood of the measured output for the current state. For the example system, this is according to (2). The likelihood computation is achieved by first evaluating the residual,

$$\hat{e}_t^i = y_t - \theta_5 (x_t^i)^2. \tag{17}$$

Then $p_e(\hat{e}_t)$, the probability density function of the measurement noise, may be evaluated. For zero-mean normally distributed noise, this is simply,

$$p_e(e_t) = \frac{1}{\sqrt{2\pi\sigma_e^2}} \exp\left(-\frac{e_t^2}{2\sigma_e^2}\right). \tag{18}$$

As in the case of the propagation step, this computation may be efficiently performed entirely in parallel.

The remaining step is that of resampling. This is where a new set of unit-weight particles are drawn according to the weighted distribution of the present particles. This is where a number of problems are presented for a parallel implementation. There is now significant interaction between each of the particles, and even the mapping of one thread per particle is in question, as some particles disappear, and others are multiply sampled to produce several new particles.

The conventional sequential approach is to start by forming a cumulative sum of the weights from the existing particles. This performs the dual role of a means for resampling out of the cumulative distribution function (CDF), and also computing a sum of the weights to allow for normalization. A cumulative sum is very efficient on a scalar processor, simply requiring L sum and store operations for L particles. However, in a parallel architecture, this cannot simply be achieved in L parallel operations. The nearest alternative is using a tree-structured adder, requiring an order of $\log_2 L$ iterations over L parallel threads. One such

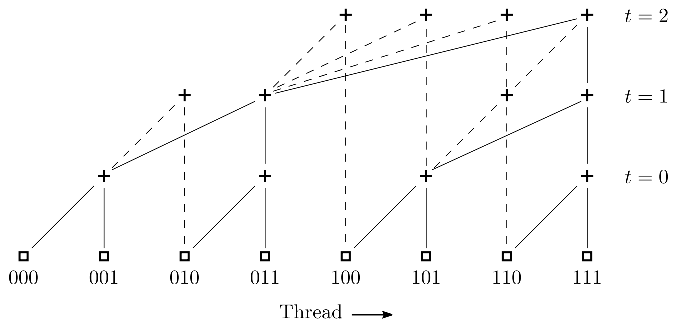

Fig. 2. *Tree structured cumulative sum adder.*

approach is illustrated in Figure 2. Each of the particle weights are laid along the horizontal axis, and the stages of addition are shown vertically. At the first time step, adjacent weights are accumulated. Over further iterations, different combinations of results are summed to produce the final cumulative sum. The solid lines represent the operations needed to simply find the sum of the values, while the dashed lines show the additional operations required for the intermediate cumulative sum results.

This addition structure uses only 50% of the available resources, and alternate arrangements allow for this computation with half the number of threads. However, as L threads are executing anyhow, this becomes the efficient way of doing it, without the overheads of altering the structure for a relatively small computational gain. Given that there are only $\log_2 L$ addition operations, any additional overheads in setting up loops and pointers can be very costly.

Once the cumulative sum of the particle weights has been calculated, there still remains the task of sampling from the weighted distribution. With a cumulative distribution function now available, inverse transform sampling may be used to convert a uniform distribution into the desired distribution.

The approach used here is systematic resampling, which uses a variant of inverse transform sampling, requiring only a single random number draw per set of particles.

After drawing one initial uniform number per particle filter,

$$\varepsilon_s = \mathcal{U}\left(0, \frac{1}{L}\right), \tag{19}$$

the uniformly distributed samples for each particle are chosen as

$$s_i = \varepsilon_s + \frac{i}{L}, \tag{20}$$

for the i-th newly-drawn particle. These uniformly distributed values will now need to be transformed according to the target distribution by mapping to the inverse of

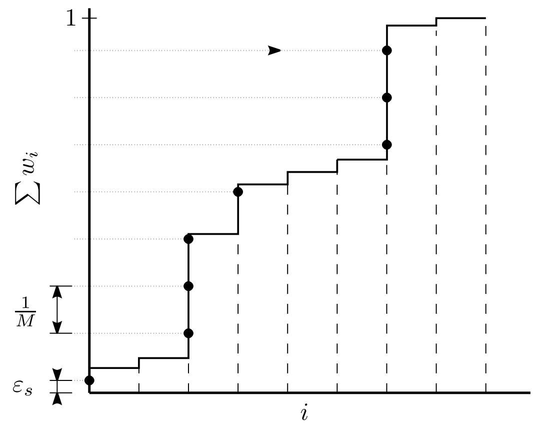

Fig. 3. *Systematic resampling out of the cumulative distribution function.*

the CDF. Using inverse transform scaling, this involves mapping the set $\{s_i\}$ through the inverse of the CDF, as shown in Figure 3.

Scalar processor implementations will typically walk through the list of $\{s_i\}$ and the sum of weights, to perform the inverse lookup into the CDF. This will take an order of L steps to move through the L particles and weights. Again this algorithm presents a challenge for parallel implementation, due to the dependence between the successive calculated values.

A solution to this can be reached by now allocating one thread of the parallel processor to each of the new particles to be drawn. Then each of the threads can run a binary search across the CDF to find the appropriate particle to draw as the new sample. Just as the case of the cumulative sum algorithm, the binary search requires $\log_2 L$ iterations.

In addition to these three basic stages of the particle filter, there is also some additional serial code. In order to avoid dynamic range problems with likelihood products, the log of the likelihood is accumulated over all of the time steps. This means computing the logarithm of the final element of the cumulative sum at each step, and then summing all of these over the number of steps of the particle filter.

An approach taken to make use of the parallel resources for this example was to instead of directly computing the logarithm, rather store the sum in memory into an array, and then compute all the logarithms in parallel at the end of the run, before summing with a parallel summer. A complication in this is that there are a different number of threads to what there are time steps, and it isn't even clear which of the two would be the greater.

The solution adopted was to maintain L memory locations, each initialized to unity. With each time step, one element of that array is multiplied by the likelihood, and after reaching the end, will wrap back to the beginning again. As there will only be at most a few likelihoods assigned to each array location, there won't be dynamic range issues in the multiplications. The logarithms then can all execute in parallel, and the overall process is quite efficient in utilizing the parallel resources.

3.1 Example 1 - linear SISO first order state-space system

In this first example the PMMH Algorithm 2 is profiled on the well-known linear state-space model in order to demonstrate its utility and build confidence in the proposed method. Importantly, for this case it is possible to compute the required likelihood exactly using a Kalman filter. As such, the PMMH approach outlined above will be profiled against a standard Metropolis-Hastings MCMC

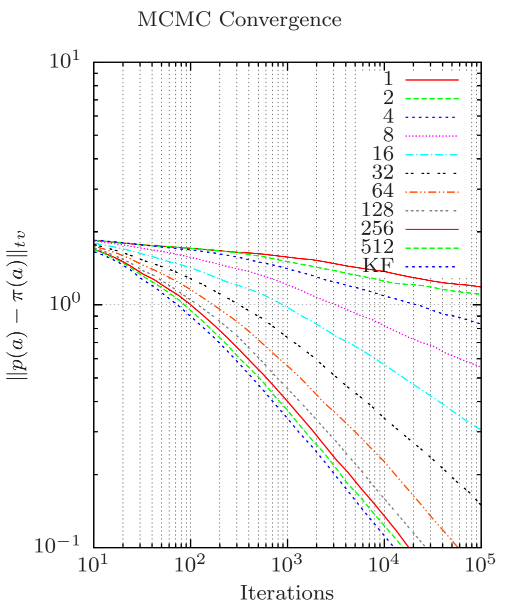

Fig. 4. *Total Variation distance between PMMH and $p(\theta \mid Y)$ for different numbers of particles. The KFMH convergence is also shown for comparison.*

approach (henceforth called the KFMH algorithm) that employs the exact likelihood calculation.

The model is a first order linear single-input single-output state-space system of the form

$$x_{t+1} = \theta_0 x_t + \theta_1 u_t + \theta_2 w_t, \tag{21a}$$

$$y_t = x_t + \theta_3 u_t + \theta_4 e_t, \tag{21b}$$

where the state $x_t \in \mathbf{R}$, the input $u_t \in \mathbf{R}$, the output $y_t \in \mathbf{R}$ and the noise terms w_t and e_t are normally distribution via

$$\begin{bmatrix} w_t \\ x_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right). \tag{21c}$$

The unit scaling of the state that appears in the measurement equation (21b) was chosen to avoid identifiability issues.

A set of $N = 100$ outputs was generated using (21), with parameter values chosen as

$$\theta = [0.9,\ 1.0,\ 0.1,\ 0.0,\ 0.1]^T \tag{22}$$

and with the initial state and input signal chosen as

$$x_0 \sim \mathcal{N}(0,\ 0), \qquad u_t \sim \mathcal{N}(0,\ 1). \tag{23}$$

Figure 4 shows the convergence of various PMMH runs using a range of particles. Here, the total variation distance is employed as a means to determine distance between densities. This requires an accurate estimate of $p(\theta \mid Y)$ with which to compare the PMMH generated densities. For this purpose, the KFMH algorithm was allowed to run for 10^{10} MCMC iterations and the resulting density for θ_1 is shown as the dashed line in Figure 5 (note that page limits restrict the presentation of all parameter densities).

As expected, Figure 4 shows that the PMMH algorithm employing 512 particles most closely matches the KFMH algorithm. It is also worth noting that the PMMH algorithm employing just 1 particle appears to be converging, albeit slowly.

Figure 4 does not indicate the computational cost as a function of the number of particles. To show this relationship, a target value of total variation distance of 0.1 was selected. An example density with $TV = 0.1$ is shown in Figure 5 to illustrate the obtained accuracy.

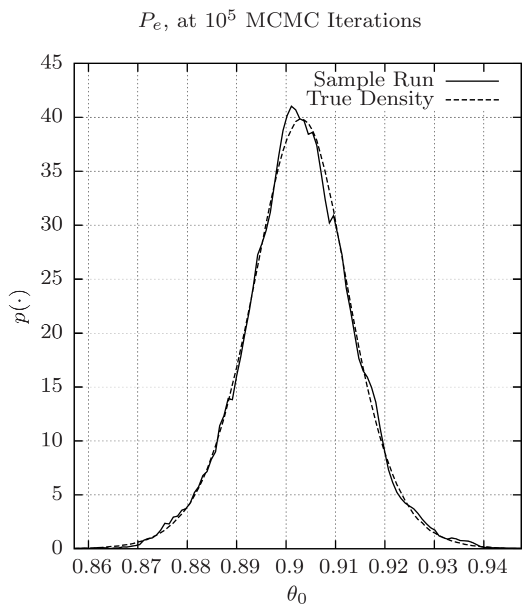

Fig. 5. *PDF $p(\theta_0 \mid Y)$ for the linear example* (21) *at a total variation of* 0.1.

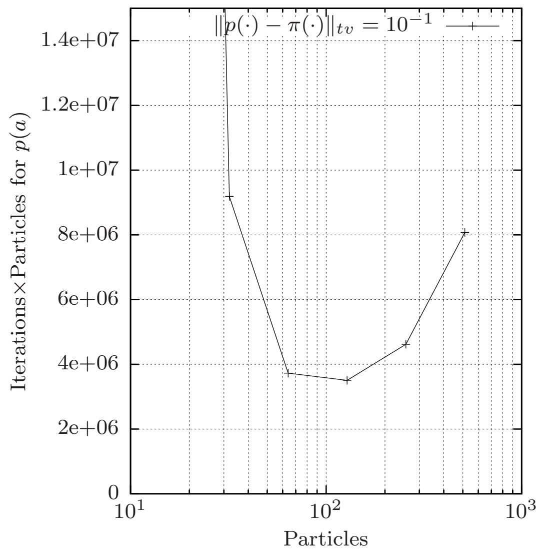

Fig. 6. *Computational cost vs number of particles for the linear example* (21).

The PMMH algorithm was run until the total variation distance achieved the required level of 0.1 for each choice of particle number. Figure 6 shows the computational cost as a function of the number of particles. From Figures 4 and 6 it can be seen that although 512 particles performs better in terms of convergence in MCMC iterations, it does not perform better in terms of computational load. Put another way, if the total number of GPU cycles is limited, then using 128 particles will achieve the best result on this particular problem. While it would be dangerous to conclude anything general from this, it does show that even a modest number of particles can achieve very good performance.

3.2 IFAC Example

A more challenging situation is now considered that involves the nonlinear and time-varying system given in (2). This example has been chosen since it is acknowledged as a challenging estimation problem in many previous

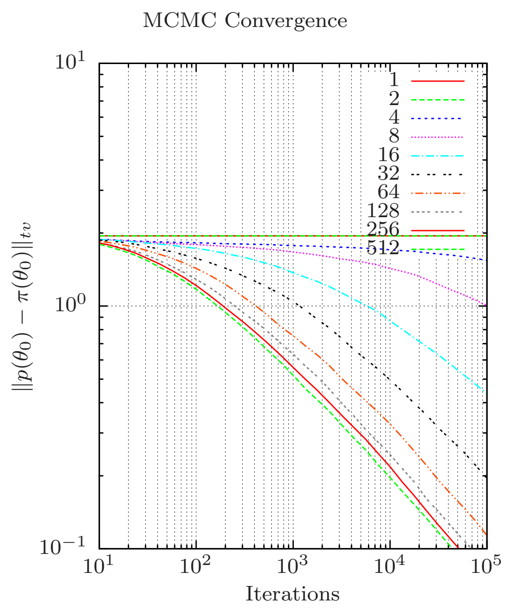

Fig. 7. *Total Variation distance between PMMH and $p(\theta_0 \mid Y)$ for different numbers of particles for the nonlinear example* (2).

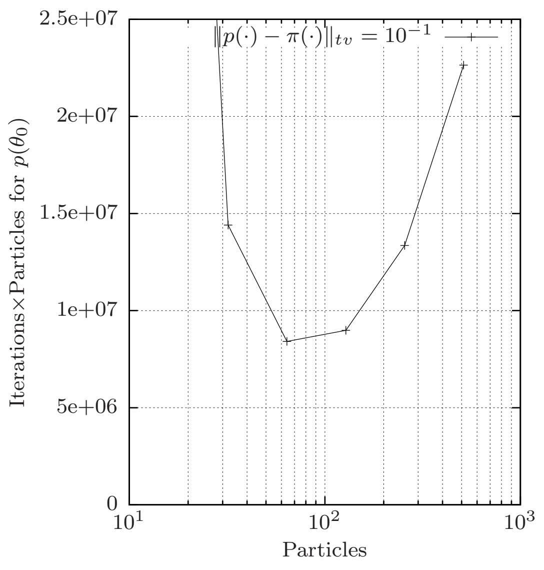

Fig. 8. *Computational cost vs number of particles for the nonlinear example* (2).

studies in the area, see e.g. Netto et al. [1978], Gordon et al. [1993], Doucet et al. [2000], Godsill et al. [2004], Andrieu et al. [2010]. In contrast to the first example, here it is not possible to compute the likelihood exactly. Therefore, as a means of gauging the performance of the PMMH algorithm, a baseline density was generated using 10^9 MCMC iterations and employing $L = 512$ particles.

Figure 7 shows the convergence rate in terms of total variation distance as a function of iterations. Again, a range of particles are profiled and similarly to the first example, employing more particles improves the convergence rate. Note that employing only 1 particle results in a convergent chain, as the theory predicts Andrieu et al. [2010].

Motivated by the reasoning for the first example, Figure 8 shows the computational cost versus number of particles. This illustrates that 64 particles is the best choice for this example since it achieves a total variation distance of 0.1 using the least number of GPU cycles. For comparison, the baseline sensitivity is profiled against a density with total variation distance of 0.1 in Figure 1.

4. CONCLUSION

The recent work by Andrieu et al. [2010] has attracted significant research attention due to the potential that this PMCMC approach offers for solving difficult identification problems. At the same time, the approach fundamentally depends on SMC methods, which are known to be computationally demanding.

This paper has demonstrated one possible means of ameliorating the computational burden of these methods by employing a parallel computing platform, namely, the graphics processing unit. The MCMC algorithm itself is directly amenable to a parallel platform, while the SMC methods require a little more attention to make them suitable to parallel computing.

The dividend for this extra attention is that the PMCMC method implemented on a GPU runs significantly faster when compared with a single core CPU platform implementation. For the nonlinear example in this paper, the GPU code ran approximately 40 times faster than the single core CPU version. Understandably, these types of claims are typically met with fierce criticism. The critical point to make is that by employing a GPU architecture, the PMCMC methods are even closer to becoming a standard tool for the system identification community.

REFERENCES

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society:Series B*, 72(2):1–33, 2010.

A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In D. Crisan and B. Rozovsky, editors, *Nonlinear Filtering Handbook*. Oxford University Press, 2011.

A. Doucet, S. J. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

Thomas Flury, Neil Shephard, and Roman Holenstein. Bayesian inference based only on simulated likelihood:Particle filter analysis of dynamic economic models. *Econometric Theory*, 59:1–24, 2011.

S. J. Godsill, A. Doucet, and M. West. Monte Carlo smoothing for nonlinear time series. *Journal of the American Statistical Association*, 99(465):156–168, March 2004.

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. A novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings on Radar and Signal Processing*, volume 140, pages 107–113, 1993.

G. Hendeby, R. Karlsson, and F. Gustafsson. Particle filtering: The need for speed. *EURASIP Journal on Advances in Signal Processing*, 2010. Article ID 181403, doi:10.1155/2010/181403.

Emlyn Jones, John Parslow, and Lawrence Murray. A bayesian approach to state and parameter estimation in a phytoplankton-zooplankton model. *Australian Meteorological and Oceanographic Journal*, 59:7–16, 2010.

A. Lee, C. Yau, M.B. Giles, A. Doucet, and C.C. Holmes. On the utility of graphics cards to perform massively parallel simulation of advanced monte carlo methods. *J Comput Graph Stat.*, 19(4): 769–789, dec 2010.

M.L.Andrade Nett, L.Gimenao, and M.J.Mendes. A new spline algorithm for non-linear filtering of discrete time systems. In *Proceedings of the 7th IFAC World Congress, Helsinki*, 1978.

M. L. A. Netto, L. Gimeno, and M. J. Mendes. A new spline algorithm for non-linear filtering of discrete time systems. In *Proceedings of the 7th World Congress of the International Federation of Automatic Control (IFAC)*, pages 2123–2130, Helsinki, Finland, June 1978.

C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer texts in statistics. Springer, New York, USA, second edition, 2004.

C. P. Robert and G. Casella. A history of Markov chain Monte Carlo-subjective recollections from incomplete data. *Statistical Science*, 26(1):102–115, 2011.