Strid, Ingvar

Working Paper

Metropolis-Hastings prefetching algorithms

SSE/EFI Working Paper Series in Economics and Finance, No. 706

Provided in Cooperation with:
EFI - The Economic Research Institute, Stockholm School of Economics

Suggested Citation: Strid, Ingvar (2008) : Metropolis-Hastings prefetching algorithms, SSE/EFI Working Paper Series in Economics and Finance, No. 706, Stockholm School of Economics, The Economic Research Institute (EFI), Stockholm

This Version is available at:
https://hdl.handle.net/10419/56369

Efficient parallelisation of Metropolis-Hastings algorithms using a prefetching approach

Ingvar Strid *
Dept. of Economic Statistics and Decision Support,
Stockholm School of Economics
SSE/EFI Working Paper Series in Economics and Finance No. 706

2009-12-02 (revised version)

Abstract

Prefetching is a simple and general method for single-chain parallelisation of the Metropolis-Hastings algorithm based on the idea of evaluating the posterior in parallel and ahead of time. Improved Metropolis-Hastings prefetching algorithms are presented and evaluated. It is shown how to use available information to make better predictions of the future states of the chain and increase the efficiency of prefetching considerably. The optimal acceptance rate for the prefetching random walk Metropolis-Hastings algorithm is obtained for a special case and it is shown to decrease in the number of processors employed. The performance of the algorithms is illustrated using a well-known macroeconomic model. Bayesian estimation of DSGE models, linearly or nonlinearly approximated, is identified as a potential area of application for prefetching methods. The generality of the proposed method, however, suggests that it could be applied in other contexts as well.
Keywords: Prefetching, Metropolis-Hastings, Parallel Computing, DSGE model, Optimal acceptance rate, Markov Chain Monte Carlo (MCMC)

*Email adress: ingvar.strid@hhs.se. Tel: +4687369232. Fax: +468348161. Permanent adress: Stockholm School of Economics, P.O. Box 6501, SE-113 83 Stockholm, Sweden. I thank Sune Karlsson, John Geweke, Ingelin Steinsland, Karl Walentin, Darren Wilkinson and Mattias Villani for comments and discussions that helped improve this paper. I also thank seminar participants at Sveriges Riksbank, Örebro University and the 14^{th} International Conference on Computing in Economics and Finance in Paris. The Center for Parallel Computers at the Royal Instititute of Technology in Stockholm provided computing time for experiments conducted in the paper.

1 Introduction

Efficient single-chain parallelisation of Markov Chain Monte Carlo (MCMC) algorithms is difficult due to the inherently sequential nature of these methods. Exploitation of the conditional independence structure of the underlying model in constructing the parallel algorithm and parallelisation of a computationally demanding likelihood evaluation are examples of problem-specific parallel MCMC approaches (Wilkinson (2006); Strid (2007*b*)). The prefetching approach and 'automatic' parallelisation using parallel matrix routines are more general approaches (Brockwell (2006); Yan et al. (2007)).

In this paper we propose simple improvements to the prefetching Metropolis-Hastings algorithm suggested by Brockwell (2006), which is a parallel processing version of the method originally proposed by Metropolis et al. (1953) and later generalised by Hastings (1970). As the name suggests the idea of prefetching is to obtain several draws from the posterior distribution in parallel via multiple evaluations of the posterior ahead of time. It is assumed that the proposal density depends on the current state of the chain, such that the future states must be predicted. We show how the random walk Metropolis-Hastings (RWMH) prefetching algorithm can be improved by utilising information on the acceptance rate, the posterior and the sequence of realised uniform random numbers in making these predictions. It is also explained how the optimal acceptance rate of the RWMH algorithm depends on the number of processors in a parallel computing setting.

When the proposal density does not depend on the current state the prediction problem vanishes. Parallelisation of the Metropolis-Hastings algorithm is simplified considerably and reminiscent of parallelisation by running multiple chains. The attractiveness of the independence chain Metropolis-Hastings (ICMH) algorithm is therefore obvious from a parallel computing perspective.

The main focus here is on developing efficient prefetching versions of the one-block RWMH algorithm. The one-block case is the most attractive setting from a pure parallel efficiency perspective. Further it allows for a simple exposition of concepts and algorithms. The prefetching method generalises to the multiple blocks case, as we describe briefly. In general, however, we expect it to be less effective in that context, at least in situations where some of the full conditional posteriors can be sampled directly using Gibbs updates.

Prefetching has obvious limitations in terms of parallel efficiency but there are at least three reasons why the approach is still interesting. First is the generality of the one-block prefetching method; it is largely problem independent. Second, prefetching is easy to combine with other parallel approaches and can therefore be used to multiply the effect of a competing parallel algorithm, via construction of a two-layer parallel algorithm. As a result, even if prefetching, by itself, is reasonably efficient only for a small number of processors its contribution to overall performance is potentially large. Finally, the method is easy to implement and provides a cheap way of speeding up already existing serial programs.

The prefetching algorithms are applied exclusively in the context of Bayesian estima-

tion of Dynamic Stochastic General Equilibrium (DSGE) models. Estimation of large-scale linearised DSGE models or nonlinearly approximated DSGE models of any size are computationally demanding exercises. This class of models is also chosen because the one-block RWMH algorithm has been the predominant choice of sampling method, for reasons explained below.

Despite the focus on macroeconomic models the generality of the prefetching approach suggests that the methods should be useful in other contexts. Estimation of long memory time series models and α-stable distributions using the RWMH algorithm are direct examples (Brockwell (2006); Lombardi (2007)). For latent variable models we argue that prefetching is suitable in conjunction with marginalisation techniques whereas other parallel approaches are required with sampling schemes based on data augmentation and Gibbs updates. To illustrate this further we discuss how to apply prefetching methods to the hierarchical Bayesian models with Gaussian Markov Random Field (GMRF) components encountered, for example, in spatial statistics (Knorr-Held and Rue (2005)).

In section 2 a brief overview of parallel MCMC is given. In section 3 terminology is introduced and the main ideas are conveyed via some simple examples before the algorithms are presented. In section 4 the prefetching algorithms are illustrated using two example problems. First the algorithms are compared using a well-known medium-scale macroeconomic model due to Smets and Wouters (2003) and the potential gains from using prefetching methods in the context of Bayesian estimation of large-scale linearised DSGE models are discussed. Second the algorithms are applied to the nonlinear estimation of a small DSGE model in an easy-to-use personal high performance computing (PHPC) environment, using Matlab on a multi-core computer. Finally, in section 5 it is illustrated how prefetching can be combined with lower level parallelism to increase the overall parallel efficiency of an estimation algorithm.

2 A brief overview of parallel MCMC

Parallel algorithms require the existence of independent tasks that can be performed concurrently. The granularity, or size, of the tasks determines to what extent an algorithm can be successfully parallelised. Bayesian inferential techniques fit unequally well with the parallel paradigm. Nonsequential methods, e.g. importance sampling, are better suited for parallelisation than sequential methods, such as Gibbs sampling. The relative merits of different sampling algorithms thus change in a parallel setting. This becomes apparent below when the random walk prefetching and parallel independence chain Metropolis-Hastings algorithms are compared. Furthermore, as we demonstrate in this paper the optimal scaling of the proposal density of the RWMH algorithm changes in a parallel computing environment, since statistical efficiency is no longer the sole concern.

The most obvious approach to parallel Metropolis-Hastings is simply to run multiple chains in parallel. This parallel chains approach does not require any parallel programming

although a parallel program could be used to automise the procedure of running the same program with different inputs on several machines (Rosenthal (2000); Azzini et al. (2007)).

In some situations it may also be of interest to parallelise a single chain. Poor mixing and long burn-in times are factors which increase the attractiveness of single-chain parallelisation (Wilkinson (2006)). More generally, MCMC methods are computationally intensive for a wide range of models and can require days or even weeks of execution time.

Parallelisation of a single chain can be divided into *within* and *between draw* parallelisation. The former includes, necessarily problem-specific, parallelisation of the likelihood evaluation, since typically this is the computationally challenging part of the posterior evaluation (Strid (2007*b*)). The blocking strategy based on exploitation of the underlying model's conditional independence structure also falls into this category (Wilkinson (2006); Whiley and Wilson (2004)). In both these approaches several processors collaborate to obtain a draw from the posterior. Between draw parallelism has been given the name prefetching (Brockwell (2006)). In the one-block prefetching approach each processor works independently on a posterior evaluation.

Parallel independence chain Metropolis-Hastings (ICMH) and parallel approaches based on regeneration are important special cases (Brockwell and Kadane (2005)). From a parallel computing perspective both these methods are largely equivalent to the parallel chains approach. Any other single-chain parallel algorithm requires frequent communication between processors. The precise nature of the parallel computer then determines whether an algorithm can be successfully used. Within-draw parallel algorithms are expected to be (typically substantially) more communication intensive than the prefetching algorithm. Hence if a particular parallel computer is deemed unsuitable for the prefetching algorithm, e.g. because the interconnection network is too slow in relation to the capacity of processors, then it should be even more inappropriate for any other, competing, within-draw single-chain parallel algorithm.

3 Prefetching

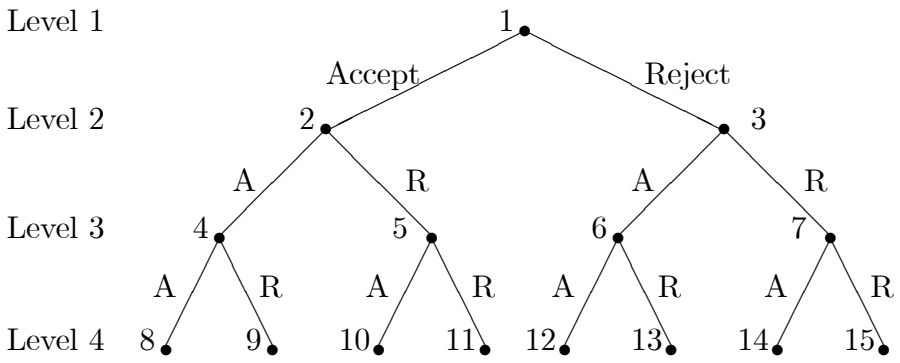
3.1 Terminology

The objective is to generate draws, $\{\theta^i\}_{i=1}^R$ where R is the length of the chain, from a posterior distribution, using the one-block Metropolis-Hastings prefetching algorithm. Throughout the paper we use the convention that parameters with superindices refer to draws whereas parameters with subindices refer to proposals. The assumption that all parameters in the vector θ are updated jointly, i.e. in one block, allows for a simple exposition of concepts.

The prefetching algorithm can be pictured using a tree; here called a Metropolis tree (figure 1). The nodes in the tree represent the *possible* future states of the chain. The number of *levels* of the tree, K, is related to the number of nodes, M, through $M = 2^K - 1$.

The branches represent the decisions to accept or reject a proposal. Accepts are pictured as down-left movements and rejects as down-right movements in the tree.

Figure 1 Four-level Metropolis tree.



A node in the tree is associated with a state of the chain and a proposal based on this state. An evaluation of a proposed parameter will always occur at the start node $i_1 = 1$. This corresponds to the evaluation required for a serial Metropolis-Hastings algorithm. The state of the chain at node i_1 is θ^i, assuming that i draws have been obtained previously, and the proposed parameter is $\theta_1 \backsim f(.|\theta^i)$. If θ_1 is accepted the chain moves from node 1 to node 2, with state $\theta^{i+1} = \theta_1$, and otherwise the chain moves to node 3 and $\theta^{i+1} = \theta^i$. More generally the states of the chain at nodes i_p and $2i_p + 1$ are the same whereas the proposed parameter is unique to each node.

The number of processes/processors (the terms are used interchangeably in this paper) is given by P. A description of the P nodes and the associated *proposed* parameters, at which the posterior is evaluated in parallel, is called a *tour* of size P

$$T(P) = \{i_1, i_2, ..., i_P;\ \theta_{i_1}, \theta_{i_2}, ..., \theta_{i_P}\}.$$

Often we refer to a tour merely by the node indices, assuming that it is understood how to determine the parameter points. The number of draws, $\tilde{D}(P,T)$, produced by a tour is a stochastic variable with support $1, ..., P$. A chain consists of the draws obtained from a sequence of tours, $\{T_n\}_{n=1}^N$, and the last parameter reached by a tour becomes the starting point for the next tour.

For an even node i_p the *parent node* is $\frac{i_p}{2}$ and for an odd node it is $\frac{i_p-1}{2}$. If a node belongs to a tour, then its parent must also belong to the tour. The expected number of draws obtained from a tour of size P is denoted $D(P,T)$ and it is given by

$$D(P,T) = \sum_{p=1}^{P} \Pr(i_p), \tag{1}$$

where $\Pr(i_p)$ is the probability of reaching node i_p and trivially $\Pr(1) = 1$. The expected number of draws of the *optimal tour*, the tour that maximises $D(P, T)$ conditional on the branch probabilities in the Metropolis tree, is denoted $D(P)$.

For some of the algorithms below the expected draws per tour can be obtained exactly whereas for others it is estimated via the average

$$\bar{d} = \frac{1}{N}\sum_{n=1}^{N} d_n, \tag{2}$$

using the output of the Metropolis-Hastings prefetching algorithm, where $\{d_n\}_{n=1}^{N}$ are the realised number of draws from the tours $\{T_n\}_{n=1}^{N}$ and

$$\bar{d} \xrightarrow{p} D,$$

by a law of large numbers. To obtain a chain of length R the required number of tours is $N = ceil\left[R/\bar{d}\right]$ and $\tilde{R} = NP > R$ posterior evaluations occur in total. Thus $\tilde{R} - R$ of the posterior evaluations are useless.

Expected draws per tour, $D(P)$, is equivalent to the theoretical speedup of the algorithm, i.e. it is the speedup in the absence of communication costs and other factors which affect parallel efficiency in practice. Draws per tour therefore provides an upper bound on the observed relative speedup

$$S(P) = \frac{T(1)}{T(P)}, \tag{3}$$

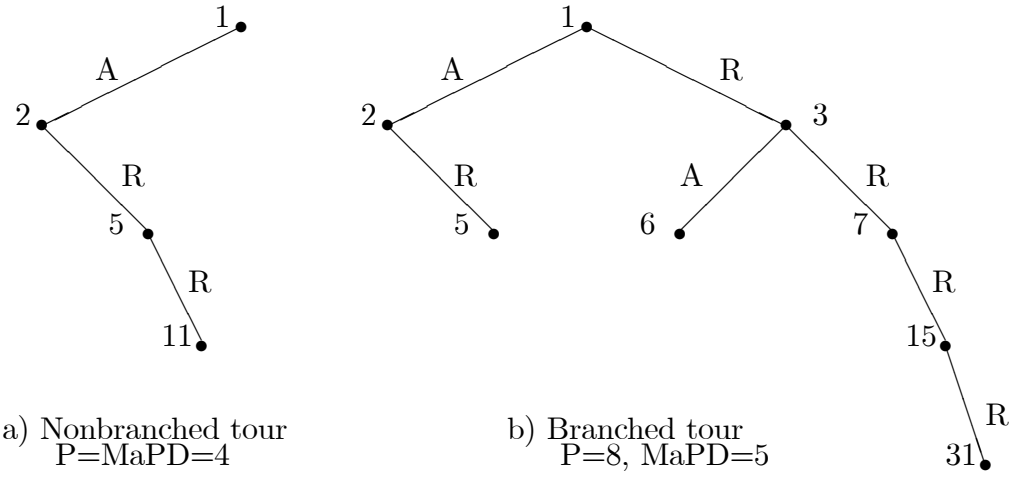
where $T(p)$ is the time of executing the parallel program using p processors in a particular hardware and software environment. The difference $D(P) - S(P) > 0$ thus depends on the precise nature of the parallel computing environment.

The *maximum possible depth* (MaPD) is the maximum number of draws that can be obtained from a tour, such that $MaPD \leq P$. Also, let $L = L(i_p)$ be the function that maps Metropolis tree indices to levels, e.g. $L(7) = 3$. A *branched tour* is a tour for which $MaPD < P$. In other words it is a tour with the property that at least two nodes at the same level of the Metropolis tree are evaluated, i.e. $L(i_p) = L(i_{\tilde{p}})$ for some pair $i_p, i_{\tilde{p}} \in T$. A *nonbranched tour* satisfies $MaPD = P$, or alternatively $L(i_p) = P$ for exactly one $i_p \in T$. The concepts are illustrated in figure 2.

A *static prefetching algorithm* is an algorithm where all tours consist of the same nodes, i.e. the indices $i_1, ..., i_P$ are constant across tours. For a *dynamic prefetching algorithm*, on the other hand, the nodes of the tour vary as we move through the chain.

Parallel random number generation is not an issue in our context since naturally all random numbers are generated by one process. The random numbers should be generated such that the chain is independent of the number of processors used. If this is the case we say that the random numbers are *draw-specific* and it is assured that prefetching does

Figure 2 Tours.



a) Nonbranched tour
P=MaPD=4

b) Branched tour
P=8, MaPD=5

not affect the statistical properties of the chain. This is an implementational issue. The attachment of random numbers to draws implies that if tour $n-1$ produces $d_{n-1} < P$ draws then the random numbers attached to levels $d_{n-1}+1, ..., P$ of the Metropolis tree, i.e. the levels that are not reached by the tour, should instead be used in tour n.

3.2 Examples

EXAMPLE 1 Assume that two processors, $P1$ and $P2$, are available for implementation of the random walk Metropolis-Hastings (RWMH) algorithm. The first processor is necessarily employed for evaluation of the posterior kernel p at

$$\theta_1 = \theta^i + \epsilon^{i+1},$$

where θ^i is the current state of the chain.

The second processor, $P2$, can be used either to prepare for an accept in the first stage, i.e. for evaluation of the posterior at

$$\theta_2 = \theta_1 + \epsilon^{i+2} = \theta^i + \epsilon^{i+1} + \epsilon^{i+2},$$

or to prepare for a reject and evaluate the posterior at

$$\theta_3 = \theta^i + \epsilon^{i+2}.$$

Assume that the acceptance probability can be chosen approximately as α by proper selection of the increment density $g(\epsilon)$, e.g. by scaling the covariance matrix of a Normal

proposal density appropriately. Now, if $\alpha < 0.5$ it is obviously optimal to use $P2$ to prepare for a reject at the first stage (R1) and the optimal tour is $T(2) = \{1, 3\}$.

If the first proposal is rejected two draws are obtained in one time unit using the two processors. If, on the other hand, θ_1 is accepted the posterior evaluation at θ_3 is useless. In this case one draw is obtained in one time unit using two processors. The expected number of draws per time unit, or the theoretical speedup, is

$$D(2, \{1, 3\}) = \alpha + 2(1 - \alpha) = 2 - \alpha.$$

This is an example of static prefetching since the tours that constitute the chain will contain the same node indices. It should also be noted that if instead $\alpha > 0.5$ the analysis is symmetric. In the remainder of the paper we restrict attention to the case $\alpha < 0.5$. The reason for this is our focus on the random walk variant of the Metropolis-Hastings method and it is further motivated by the discussion in section 3.3.5 below.

EXAMPLE 2 In the case of three processors, $P1$,$P2$ and $P3$, assume $\alpha < 0.5$ as before. The optimal allocation of $P1$ and $P2$ is clearly the same as in the example above. For the third processor there are three options. First it can be employed for evaluation of

$$\theta_7 = \theta^i + \epsilon^{i+3},$$

i.e. anticipating rejects in the first two stages (R1,R2). A second possibility is to evaluate θ_2 in anticipation of an accept at the first stage. This yields the branched tour $\{1, 2, 3\}$ and $D(3, \{1, 2, 3\}) = 2$ draws are obtained with certainty, since $P2$ was used to prepare for a reject at the first stage.

The final possibility is to prepare for a reject in the first stage followed by an accept in the second stage (R1,A2) and evaluate

$$\theta_6 = \theta^i + \epsilon^{i+2} + \epsilon^{i+3}.$$

In the first case (R1,R2) the expected number of draws is

$$D(3, \{1, 3, 7\}) = \alpha + 2(1 - \alpha)\alpha + 3(1 - \alpha)^2. \tag{4}$$

For example, with $\alpha = 0.25$ this yields 2.31. It is optimal to choose the nonbranched tour $\{1, 3, 7\}$ if

$$D(3, \{1, 3, 7\}) > D(3, \{1, 2, 3\}) = 2,$$

which is if

$$\alpha \leq \frac{3 - \sqrt{5}}{2} \approx 0.38.$$

Note that

$$D(3, \{1, 3, 7\}) - D(3, \{1, 3, 6\}) = (1 - \alpha)(1 - 2\alpha) > 0,$$

for $\alpha < 0.5$ so it is never optimal to prepare for (R1,A2).

Again, if instead $\alpha > 0.5$ is assumed the optimal tour for $\alpha \geq \frac{\sqrt{5}-1}{2} \approx 0.62$ would be given by $\{1, 2, 4\}$ due to the symmetry.

EXAMPLE 3 In the random walk Metropolis-Hastings algorithm a proposal at stage $i+1$ is accepted with probability $\alpha^{i+1} = \min\{X^{i+1}, 1\}$, i.e. if $u^{i+1} < X^{i+1}$ for some $u^{i+1} \sim U[0,1]$ where

$$X^{i+1} = \frac{p(\theta^i + \epsilon^{i+1})}{p(\theta^i)}.$$

Consider example 1 again and assume that $u^{i+1} = 10^{-5}$ where u^{i+1} is the realised uniform random number associated with draw $i+1$. Clearly, in this hypothetical situation it is wise to use $P2$ to prepare for an accept, assuming no information about the posterior ratio X^{i+1} is available. The sequence of uniform random numbers, $\{u^i\}_{i=1}^R$ which are fixed from the outset, thus carries some information on how to best structure prefetching.

EXAMPLE 4 The acceptance rate, α, and the sequence of uniform random numbers, $\{u^i\}_{i=1}^M$, can be used to improve the efficiency of prefetching. Finally, knowledge about the posterior, p, can be used to make better predictions on where the chain is moving. Assume that an approximation to the posterior, p^*, is available and that the evaluation of p^* takes a fraction of the time to evaluate p. Then this approximate distribution can be used to suggest the states which are likely to be visited subsequently.

In the limiting case when $p^* = p$ we say that prefetching is perfect; it is analogous to importance sampling using the posterior as the importance function. In other words it is a situation when neither prefetching or importance sampling is necessary.

3.3 Algorithms

3.3.1 One-block random-walk Metropolis-Hastings prefetching

In this section the one-block Metropolis-Hastings prefetching algorithm is presented. This algorithm applies when all the parameters in θ are updated jointly and the proposal density f depends on the current state of the chain, e.g. for the one-block random walk Metropolis-Hastings sampler which we focus on here. It would be possible to use other proposal densities, e.g. a more general autoregressive proposal, but this route is not pursued here. Multiple block prefetching is discussed briefly at the end of this section and in appendix B.

The algorithm assumes that the posterior evaluation time is not parameter dependent and it is intended for application on a homogeneous cluster. The key assumption for application of the algorithm in practice is, loosely speaking, that the time of a posterior evaluation is significant in relation to the other steps of the algorithm.

Algorithm 1 *Metropolis-Hastings prefetching algorithm*

1. Choose a starting value θ^0. Set the draw counter $S_0 = 0$.

2. (Prefetching step) Assume that the chain is in state $\theta^{S_{n-1}}$ when the n^{th} tour begins, where S_{n-1} is the state of the draw counter after $n-1$ completed tours. Construct a tour, $T_n = \{i_1, ..., i_P; \theta_{i_1}, \theta_{i_2}, ..., \theta_{i_P}\}$ (serial).

3. Distribute $\theta_{i_p}, p = 1, .., P$, to the processes (*scatter*).

4. Evaluate $p\left(\theta_{i_p}\right)$ in parallel.

5. Return $p\left(\theta_{i_p}\right)$ (*gather*).

6. (Metropolis-Hastings step) Run the Metropolis-Hastings algorithm for the tour T to generate D_n draws, where $1 \leq D_n \leq P$. Update the draw counter, $S_n = S_{n-1} + D_n$, and assign the starting state for the next tour, θ^{S_n} (serial).

7. Go to 2. Stop when $S_n \geq R$ draws from the posterior have been obtained. ⊡

The posterior evaluations, step 4 above, are performed in parallel. Communication between processes takes place in steps 3 and 5 and message passing collective communication tokens are used to describe the required operations. Steps 2 and 6 are performed by a master process or are replicated by all processors.

The Metropolis-Hastings step, step 6 in the above algorithm, is implemented in the following way for the random walk algorithm:

Algorithm 2 *Random walk Metropolis-Hastings step*

1. Set $j = 1$. The current state of the chain is $\theta^{S_{n-1}}$, where S_{n-1} is the state of the draw counter after $n-1$ completed tours. Let $\theta_{i_1} = \theta_1$ be the proposal conditional on the current state, i.e. $\theta_1 = f\left(.|\theta^{S_{n-1}}\right) = \theta^{S_{n-1}} + \epsilon^{S_{n-1}+1}$.

2. The current node is i_p, with associated state $\theta^{S_{n-1}+j-1}$, and its level is $L\left(i_p\right) = j$. If

$$u^{S_{n-1}+j} < \alpha^{S_{n-1}+j} = \min\left\{1, \frac{p\left(\theta^{S_{n-1}+j-1} + \epsilon^{S_{n-1}+j}\right)}{p\left(\theta^{S_{n-1}+j-1}\right)}\right\}, \tag{5}$$

   where $u = U\left[0, 1\right]$, accept the draw and set $\theta^{S_{n-1}+j} = \theta^{S_{n-1}+j-1} + \epsilon^{S_{n-1}+j}$. Otherwise set $\theta^{S_{n-1}+j} = \theta^{S_{n-1}+j-1}$.

3. If (i) the proposal $\theta_{i_p} = \theta^{S_{n-1}+j-1} + \epsilon^{S_{n-1}+j}$ was accepted and $i_{\tilde{p}} = 2i_p$ or (ii) if θ_{i_p} was rejected and $i_{\tilde{p}} = 2i_p + 1$ for some node $i_{\tilde{p}} \in T$ then move to $i_{\tilde{p}}$. In this case set $j = j + 1$ and the current node to $i_{\tilde{p}}$ and return to 2. Otherwise exit. ⊡

The output from algorithm 2 are the D_n draws from the tour n, $\theta^{S_{n-1}+1}, ..., \theta^{S_n}$, and the output from the prefetching algorithm consists of the draws

$$\left\{\theta^{S_{n-1}+1}, ..., \theta^{S_n}\right\}_{n=1}^{N} = \left\{\theta^1, \theta^2, ..., \theta^{S_N}\right\},$$

where N is the total number of completed tours and

$$\sum_{n=1}^{N} D_n = S_N \geq R.$$

The remaining discussion in this section focuses on the prefetching step (step 2 of algorithm 1). The prefetching, or tour construction, problem consists of two separate parts:

1. Determine a rule for calculating the probabilities of reaching the nodes in the Metropolis tree based on some set of information.

2. Find the tour that maximises the expected number of draws per tour, D, based on these probabilities. We call this the *optimal tour*, with the implicit understanding that optimality is conditioned on the specific rule used.

The second task is easy and an algorithm which constructs the optimal tour given a rule for assigning the probabilities of the Metropolis tree is presented in appendix A. Below a parallel independence chain Metropolis-Hastings algorithm is first presented and then five variants of prefetching are discussed, named as follows: (1) Basic prefetching, (2) Static prefetching, (3) Dynamic prefetching using the sequence of uniform random numbers, (4) Dynamic prefetching using a posterior approximation and (5) Most likely path. Finally we discuss prefetching with multiple blocks and other proposal densities.

3.3.2 Independence chain Metropolis-Hastings

In the special case when the proposal density is fixed

$$f\left(\theta|\theta^i\right) = f\left(\theta\right),$$

parallelisation is simplified further since there are no dependencies between posterior evaluations. Let

$$R = \sum_{p=1}^{P} R_p,$$

where R_p is the number of posterior evaluations performed by process p and R is the length of the chain.

Algorithm 3 *Parallel Independence Chain Metropolis-Hastings (ICMH) algorithm*

1. Each process p generates $\theta_p = \left\{\theta_{1p}, \theta_{2p}, ..., \theta_{R_p p}\right\}$ where

$$\theta_{ip} \sim f,\ i = 1, ..., R_p,$$

   and collects the values of the posterior evaluated at these parameters in the vector $p_p = \left\{p\left(\theta_{1p}\right), p\left(\theta_{2p}\right), ..., p\left(\theta_{R_p p}\right)\right\}$ (parallel).

2. The master process gathers θ_p and p_p, $p = 1, ..., P$ (*gather*).

3. Run the Metropolis-Hastings algorithm with the posterior already evaluated at R parameter values (serial). ⊡

In this algorithm the master process only collects the results from the other processes once. If memory limitation is a concern it can be solved by collecting the local results more often. In the case of inhomogeneous processors load balance is easily restored by allowing R_p to vary across processors.

The parallel ICMH algorithm is *embarrassingly parallel* both in the sense that it is simple to implement and because there is no essential dependency between the parallel tasks. As a consequence this algorithm will yield close to linear speedup, i.e. $S(P) \approx P$, on *any* homogeneous parallel computer. The prefetching RWMH algorithm is also simple to implement but good parallel performance for this algorithm requires a balance between the problem (or problem size) and hardware/network since communication is frequent.

3.3.3 Basic prefetching

The basic prefetching algorithm suggested by Brockwell (2006) has the property that all future states at the same level in the Metropolis tree are treated as being equally likely to be reached. No information is used to structure prefetching. In our framework this algorithm is obtained if every branch in the Metropolis tree is assigned the probability 0.5.

The basic prefetching tour evaluates the proposed parameters of all nodes up to a given level of the Metropolis tree, i.e. $T(P) = \{1, 2, ..., P\}$, and thus produces a certain number of draws

$$D(P) = MaPD = \log_2(P + 1). \tag{6}$$

This approach provides a lower bound on the scalability that can be achieved using prefetching. Note that if the number of processors P does not correspond to the values $2^p - 1, p = 1, 2, ...$ there is no clear guide on how to select the 'surplus' nodes for evaluation.

3.3.4 Static prefetching

The researcher can typically target an acceptance rate, α, with good precision by selection of the proposal density, f, in the Metropolis-Hastings algorithm. Knowing α it is easy to improve on the basic algorithm. Let $T(P|\alpha, \tilde{\alpha})$ denote the tour when the acceptance rate is α and the 'perceived' acceptance rate used to construct the tour is $\tilde{\alpha}$, thus allowing for imperfect targeting. The static prefetching tour is obtained by attaching the probability $\tilde{\alpha}$ to all accept (down-left) branches of the Metropolis tree and $1 - \tilde{\alpha}$ to reject (down-right) branches. This was explained in some detail in examples 1 and 2 above for the cases $P = 2$ and $P = 3$. The nonbranched tour shown in figure 2 is obtained if we choose $P = 8$ and target, for example, $\tilde{\alpha} = 0.25$.

The node indices of the tours only needs to be obtained once per chain and the approach is general, i.e. model independent, since it only depends on the acceptance rate. For large enough α and/or P the tour will be branched.

In figure 3 the expected number of draws per tour, $D(\alpha, P)$, and the maximum possible depth (MaPD) of the optimal static prefetching tour is plotted against the acceptance rate $\tilde{\alpha} = \alpha \in (0, 0.5)$ for $P = 3, 7, 15$. The expected number of draws decreases smoothly in α and above some threshold $\alpha_h(P)$ it becomes optimal to choose the basic prefetching tour. If α is below some threshold $\alpha_l(P)$ the optimal tour is nonbranched and an 'always prepare for a reject' strategy is chosen. The MaPD curve jumps at points where the optimal tour changes.

What happens if the tour is constructed based on a perceived acceptance rate $\tilde{\alpha}$ when the true acceptance rate is in fact α? In figure 4 the expected number of draws is plotted for $\tilde{\alpha} \in (0, 0.5)$ and $P = 3, 7, 15, 31$ when the true acceptance rate is $\alpha = 0.25$. It is seen that it is optimal to use $\tilde{\alpha} = \alpha$ in constructing the tour but the optimal choice of $\tilde{\alpha}$ is not unique. For example, as was seen in example 2 above, in the case of three processors any $\tilde{\alpha} \in (0, 0.38)$ will suggest the nonbranched tour $T(3|0.25, \tilde{\alpha}) = \{1, 3, 7\}$ which is the optimal tour if $\alpha = 0.25$.

3.3.5 Optimal static prefetching

A parallel efficiency perspective obviously suggests targeting a low acceptance rate, as seen in figure 3. Intuitively it becomes very easy to predict where the chain is moving when α is small.

The optimal acceptance rate, $\alpha_{opt,1}$, for the random walk Metropolis-Hastings algorithm has been derived under various assumptions on the target density p (Roberts et al. (1997); Roberts and Rosenthal (2001)). Here we show how the optimal acceptance rate depends on the number of processors in a parallel computing framework, thus considering jointly Markov chain, or statistical, efficiency and parallel efficiency of the static prefetching RWMH algorithm.

Figure 3 Static prefetching performance, $\tilde{\alpha} = \alpha$.

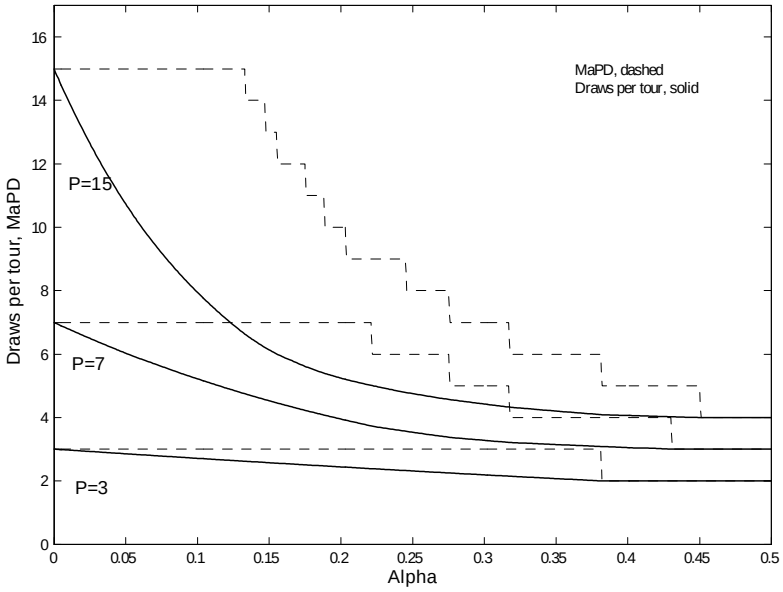
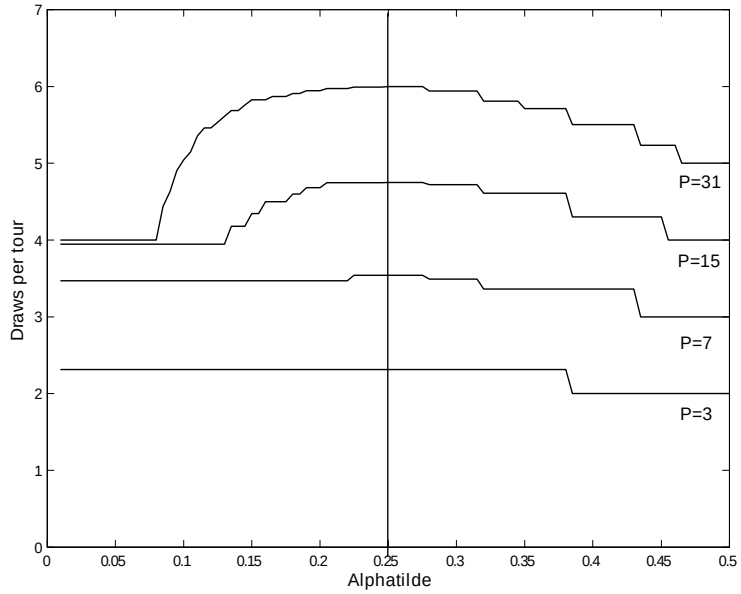


Figure 4 Static prefetching performance. The true acceptance rate is $\alpha = 0.25$ and tours are constructed based on $\tilde{\alpha}$.



We revisit the special case where the posterior density has the product form

$$p(\theta) = \prod_{i=1}^{k} \tilde{p}(\theta_k),$$

and the increment density, g, is of the form $g(\epsilon) = N(0, I_k \sigma_k^2)$ where $\sigma_k^2 = l^2/k$ and l is the scaling parameter. In the high-dimensional limit, i.e. as $k \longrightarrow \infty$, and under certain regularity conditions on $\tilde{p}$ it can be shown that the Markov chain converges to a diffusion process and the optimal acceptance rate is obtained by maximising the 'efficiency function'

$$E^1(\alpha) \propto \alpha \times \left[\Phi^{-1}\left(\frac{\alpha}{2}\right)\right]^2, \tag{7}$$

where Φ is the standard normal cumulative distribution function. This yields the famous result of an optimal acceptance rate $\alpha_{opt,1} = \arg\max_\alpha E^1(\alpha) \approx 0.234$ (see Theorem 1 in Roberts and Rosenthal (2001) and the subsequent discussion).

In figure 5 the joint output measure

$$E(\alpha, P) = E^1(\alpha) \times D(\alpha, P) \times c, \tag{8}$$

is plotted as a function of the acceptance rate α and the number of processors P. The constant c is chosen such that $E(\alpha_{opt,1}, 1) = 1$. The figure is interpreted as follows: the *time* needed to achieve a given accuracy in estimating any function $h(\theta)$ of the posterior is inversely related to the output E.

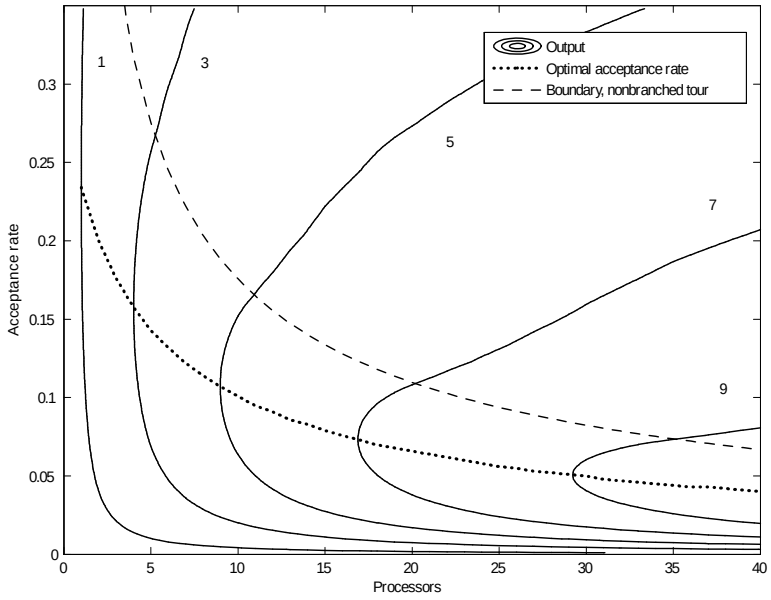
The *optimal acceptance rate in a parallel setting* is the acceptance rate which minimises the time of obtaining a sample of a fixed size and quality, e.g. a fixed number of iid draws, from the posterior, conditional on the number of processors used. It is seen in figure 5 that the optimal acceptance rate implies an 'always prepare for a reject' prefetching strategy. The optimal rate is below the nonbranched tour boundary which is defined implicitly by $\alpha = (1-\alpha)^{P-1}$. For example, using $P = 7$ processors the optimal acceptance rate is $\alpha_{opt,7} = 0.120$ and a sample of a given size and quality from the posterior can be obtained $E(\alpha_{opt,7}, 7) = 4.3$ times faster than when applying one processor and the optimal acceptance rate $\alpha_{opt,1} = 0.234$.

Our choice of c implies that $E(\alpha_{opt,P}, P)$ is interpreted as the *optimal speedup* of the static prefetching RWMH algorithm and it satisfies

$$D(\alpha_{opt,1}, P) \leq E(\alpha_{opt,P}, P) \leq D(\alpha_{opt,P}, P),$$

where equality holds for $P = 1$. For the case of $P = 7$ processors we have $E(\alpha_{opt,1}, 7) = D(\alpha_{opt,1}, 7) = 3.65$. The benefit of lowering the acceptance rate from $\alpha_{opt,1} = 0.234$ to $\alpha_{opt,7} = 0.120$, the gain in parallel efficiency, is larger than the cost, the loss of statistical efficiency.

Figure 5 Optimal static prefetching acceptance rates.



More generally the optimal acceptance rate $\alpha_{opt,P}$ will be determined by the curvature of the efficiency function $E^1(\alpha)$. In a serial computing setting a flat efficiency curve implies that it is 'of little value to finely tune algorithms to the exact optimal values'(Roberts and Rosenthal (2001)). In a parallel computing setting the implication is that parallel efficiency can be improved, by lowering the acceptance rate, without incurring a large cost in terms of statistical efficiency.

In our applications below the efficiency function $E^1(\alpha)$ is not available analytically and hence the optimal acceptance rate $\alpha_{opt,P}$ cannot be solved for. Instead we target an acceptance rate $\alpha = 0.25$ for all P, a 'conservative' choice, and compare the parallel efficiency of different prefetching approaches while keeping the statistical properties fixed. In the main illustration the empirical static prefetching optimal speedup and the associated empirical optimal acceptance rates are also obtained.

Finally, in the processor limit

$$\lim_{P \longrightarrow \infty} \left\{ \arg\max_{\alpha} E(\alpha, P) \right\} = \lim_{P \longrightarrow \infty} \alpha_{opt,P} = 0.$$

In other words we can afford to make the algorithm arbitrarily poor, in the statistical sense, only if parallel computing technology is arbitrarily cheap.

3.3.6 Dynamic prefetching based on the uniform random numbers

The probability of accepting a proposal θ_z drawn from $f(\theta|\theta^i)$ depends on the realised value of a uniform random number, u^{i+1}. Let

$$X^{i+1} = \frac{p(\theta_z)}{p(\theta^i)},$$

and consider the conditional acceptance probability

$$\alpha^{i+1}\left(u^{i+1}\right) = \Pr\left(u^{i+1} < X^{i+1}|u^{i+1}\right),$$

where u is treated as fixed and X as stochastic. Since it is possible to characterise the relationship between α and u the static prefetching algorithm can be improved upon.

In the applications below we incorporate information from the sequence of realised uniform random numbers in the following way. First R draws from the Metropolis-Hastings sampler are obtained. Next, the unit interval is divided into K subintervals I_k and an empirical acceptance rate

$$\alpha_k = \frac{\#\left\{\{u < X\} \cap \{u \in I_k\}\right\}}{\#\left\{u \in I_k\right\}}, \tag{9}$$

is calculated for each subinterval $k = 1, ..., K$ based on approximately R/K draws. The constant acceptance rate used in the static prefetching algorithm above is then replaced by an acceptance probability which depends on a uniform random number

$$\alpha^{i+1}\left(u^{i+1}\right) = \sum_{k=1}^{K} \alpha_k I\left(u^{i+1} \in I_k\right), \tag{10}$$

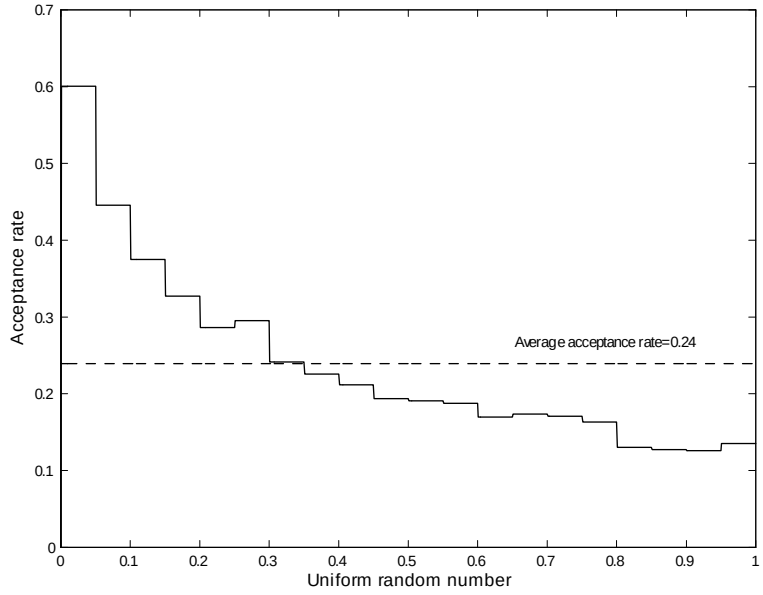
where $I_k = [\frac{(k-1)}{K}, \frac{k}{K}]$ and I is the indicator function. This algorithm has the property that the branch probabilities in the Metropolis tree are level-specific, i.e. all accept branches at the same level in the tree will have the same probability attached to them.

The procedure is illustrated in figure 6 where the acceptance rate is plotted against the uniform random number for the linear estimation example presented later in the paper. Here $R = 100,000$ draws are used to obtain estimates of α_k for $K = 20$ equally sized subintervals. For example it is seen that $\alpha_1 = 0.60$ is the estimate of $\Pr(u < X|u \in [0, 0.05])$.

3.3.7 Dynamic prefetching based on a posterior approximation

If a posterior approximation, p^*, is available and if evaluation of p^* is fast in comparison with the posterior kernel p a possibility is to use p^* to determine at which parameters to evaluate p. The objective is thus the same as in importance sampling or independence chain Metropolis-Hastings (ICMH) sampling, i.e. to find a good approximation

Figure 6 Acceptance rate as a function of the uniform random number in the estimation of a linear DSGE model.



to the posterior. However, a difference is that the quality of the approximating density used for prefetching has no implications for the statistical analysis; it will 'merely' affect computational efficiency.

For the RWMH algorithm one possible strategy is to simply replace the constant α in the construction of the static prefetching tour with the probability

$$\min\left\{\beta, \frac{p^*\left(\theta_{i_p}\right)}{p^*\left(\theta^i\right)}\right\},$$

where $\theta_{i_p} = \theta^i + \epsilon$ and thus incorporate information about the posterior in the construction of the tour. We call this *dynamic prefetching using a posterior approximation*. If $\beta < 1$ then $\Pr\left(i_p\right) > 0$ for all p, i.e. all nodes of the Metropolis tree have positive probability of being visited. In the application below we let $\beta = 1$, thus eliminating parts of the tree. As in the static prefetching case the tour will in general be branched.

Another possibility is to run the chain P steps ahead using the approximate density, thereby also incorporating the information contained in the sequence of realised uniform random numbers. The states visited using the approximate posterior yield the parameter points at which the posterior p is subsequently evaluated in parallel. The tour is nonbranched and we call this type of algorithm a *most likely path algorithm*.

3.4 Marginalisation and blocking

The Dynamic Stochastic General Equilibrium (DSGE) models used for the illustrations in this paper are represented as state-space models. The coefficients of the DSGE-state-space model are nonlinear functions of the structural parameters, θ. The latent variables, collected in x, are integrated out of the joint posterior $p(\theta, x|y)$ using filtering techniques and attention is restricted to the marginal posterior density, $p(\theta|y)$, where y denotes the data. Sampling from the conditional $p(\theta|x, y)$ is precluded for these models. In the main example, in section 4.1, the Kalman filter is applied for integration in a degenerate linear and Gaussian state-space (LGSS) model. If there is interest in the posterior distribution for the state variables, $p(x|y)$, this distribution is obtained via smoothing *after* $p(\theta|y)$ has been sampled, see e.g. Durbin and Koopman (2001). The marginalisation technique has also been used in other contexts, e.g. Garside and Wilkinson (2004).

For comparison, consider the more typical LGSS model where interest centers on $p(\theta, x|y)$ and where, in contrast to the linearised DSGE model, the classic two-block data augmentation scheme is available. The conditional posterior $p(\theta|x, y)$ is sampled using Gibbs or Metropolis updates and $p(x|\theta, y)$ is sampled using a simulation smoother, e.g. Strickland et al. (2009). A Gibbs step, viewed as a Metropolis-Hastings update with acceptance probability 1, implies perfect prefetching and hence prefetching does not apply. In situations where a Metropolis-Hastings block sampler with reasonably good mixing properties is available we expect prefetching to be of limited use and, if thought necessary, other parallelisation approaches should be considered. Wilkinson (2006) shows how to exploit the conditional independence structure to parallelise the multimove simulation smoother used for estimation of the baseline stochastic volatility (SV) model. This approach can also be applied to extended SV models and the stochastic conditional duration (SCD) model, for which efficient block sampling schemes have been developed (Omori and Watanabe (2008); Takahashi et al. (2009); Strickland et al. (2006)). A similar 'parallel blocks' approach is also applied by Whiley and Wilson (2004).

There are two features, in addition to the inability to sample from $p(\theta|x, y)$, which increase the attractiveness of the one-block prefetching approach for DSGE models. First, it is typically a nontrivial task, at least *a priori*, to split the parameter vector θ into blocks such that there is weak posterior dependency between blocks. Second, the resulting Metropolis sampler with B blocks is 'penalised' by a B-factor increase in computational time relative to the one-block sampler.

More generally, although prefetching generalises to the case of Metropolis multiple block sampling parallel efficiency is expected to suffer in that context, at least when a subset of the full conditionals can be sampled directly. If high dimensionality of the parameter vector, rather than exploitable structure, is the motive for splitting the parameters into blocks then Metropolis block prefetching *could* be interesting but this is not investigated further here. A brief description of block prefetching is given in appendix B.

Another potentially interesting application for prefetching methods, not fully explored

in this paper, is the class of hierarchical Gaussian Markov random field (GMRF) models with a wide range of applications, e.g. in spatial statistics (Knorr-Held and Rue (2005)). Knorr-Held and Rue (2002) develop a 'hyperblock' sampler for this type of model, using the methods for fast sampling from GMRFs presented by Rue (2001). This is another, more elaborate, example of a marginal updating scheme where the hyperparameters θ and the latent field x are updated jointly using a Metropolis-Hastings step.

Knorr-Held and Rue (2002) demonstrate that the hyperblock sampler mixes more rapidly than the traditional single-site updating scheme and various intermediate blocking schemes in three typical disease mapping applications. The results are driven by the strong posterior dependency between θ and x in these applications. A one-block sampler for linear mixed regression models, along similar lines, is suggested by Chib and Carlin (1999). The marginalisation approach is also used by Gamerman et al. (2003) for spatial regression models and in Moreira and Gamerman (2002) for time series models.

The possibility of prefetching parallelisation may be interpreted as a further argument in favour of the sophisticated one-block sampling scheme for GMRF models, in addition to the documented improvement in chain mixing. In fact, the 'always prepare for a reject' prefetching strategy applies to a more general one-block sampler, suggested by Wilkinson and Yeung (2004) for the related class of linear Gaussian Directed Acyclic Graph (DAG) models. In appendix B we outline how to apply prefetching in the GMRF context and also mention the possibility of combining prefetching with the parallel GMRF sampling approach suggested by Steinsland (2007).

4 Illustrations

4.1 Linear estimation of a DSGE model

Long burn-in times and poor mixing are factors that can motivate interest in single-chain parallel methods. In the area of Bayesian estimation of Dynamic Stochastic General Equilibrium (DSGE) models the one-block random walk Metropolis-Hastings algorithm is the most commonly used estimation method and our impression is that poor mixing of chains is a typical experience of researchers in this field. For an exception, see Belaygorod and Dueker (2006).

The performance of the five variants of the prefetching algorithm presented above and the parallel ICMH algorithm is compared using one of the core macroeconomic models at the European Central Bank, the Smets and Wouters (SW) model (Smets and Wouters (2003)). The model is chosen because it is well known and since it is the backbone of larger DSGE models currently developed at many central banks. A major determinant of the computational cost of estimating a DSGE model is the number of state variables of the model. Our version of the SW model has 15 state variables, including 8 shocks. Recently developed large-scale microfounded models in use at central banks have as many as 65

state variables and there is at least one very recent example of a model which contains more than 100 state variables (Adolfson, Laséen, Lindé and Villani (2007); Christiano et al. (2007)). In a relative sense, and using the jargon of parallel computing, this implies that our estimation problem can be viewed as fine-grained.

The empirical analysis of large-scale DSGE models is restricted by the computational cost of estimating them. Re-estimation is necessary as different model specifications are evaluated or when new data arrives. Furthermore, a central bank is facing real-time constraints on these activities. In our view the development of larger and more complex models and the unavailability of adequately efficient sampling methods increase the attractiveness of applying the parallel methods presented here.

4.1.1 Model, prior, solution and likelihood

The economic content of the model is largely unimportant for our evaluation of the parallel algorithms and therefore it is not presented here. Similar models have been analysed and/or estimated in many articles (Smets and Wouters (2003); del Negro et al. (2005)). The model used here is presented in detail in Strid (2007*a*).

The model consists of a set of nonlinear expectational equations describing the equilibrium and a linear approximation to the policy function of the model is obtained (the solution). The linear approximate policy function is represented as a linear state-space model:

$$X_t = T(\theta) X_{t-1} + R(\theta) \epsilon_t \tag{11}$$

and

$$Y_t = d(\theta) + Z X_t + v_t, \tag{12}$$

where [11] is the state equation and [12] the measurement equation. Here X_t (dimension n_x) is a vector containing the state variables and Y_t (n_y) is a vector containing the observed variables. The structural parameters of the model are collected in the vector θ (n_θ) and the coefficient matrices, T (which is dense), R and Z, and the vector d are nonlinear functions of θ. The innovations, ϵ_t (n_ϵ), and the measurement errors, v_t (n_v), are assumed to be independent and normally distributed, $\epsilon_t \backsim N(0, \Sigma_\epsilon)$ and $v_t \backsim N(0, \Sigma_v)$.

The likelihood evaluation consists of two parts. First the model is solved using one of the many available methods (Klein (2000)). Second the likelihood function of the model is evaluated using the Kalman filter, see e.g. Durbin and Koopman (2001). The prior distribution for θ is similar to the distributions typically used in the literature on estimation of New Keynesian models.

For the model estimated here we have the dimensions $n_x = 15, n_y = 5, n_\epsilon = 8$ and $n_\theta = 23$. The model is estimated on Euro Area data for the period 1982:1-2003:4 (88 observations). The five data series used for estimation are the short-term interest rate, inflation, output growth, the consumption-to-output ratio and the investment-to-output ratio.

4.1.2 Parallel approaches for linearised DSGE models

In the context of estimation of large-scale linearised DSGE models, what other strategies for single-chain parallelisation are available apart from prefetching? An obvious within-draw approach is to apply parallel matrix algorithms (e.g. using ScaLAPACK or PLAPACK) to the computations involved in Kalman filtering, primarily the matrix multiplications of the Riccati equation which account for a significant part of Kalman filtering time. However, if this approach is attempted it is crucial to apply parallel routines *whenever fruitful*, including in the solution algorithm. The extent to which a few standard matrix operations account for overall Kalman filtering and model solution time determines the parallel efficiency properties of the approach. Furthermore, it is hard to improve on optimal serial DSGE-specific Kalman filter implementations using parallel methods, see Strid and Walentin (2008).

Note again that it would be straightforward, at least in principle, to combine such local computations strategies, which are somewhat more demanding to implement, with prefetching.

4.1.3 Parallel efficiency: draws per tour

The posterior approximation used for prefetching (variants 4 and 5) is a normal approximation $p^* = N(\theta_m, \Sigma_m)$ where θ_m is the posterior mode, H_m the Hessian matrix at the mode and $\Sigma_m = H_m^{-1}$. The scaled inverse Hessian is also used as the covariance matrix for the proposal density in the random walk Metropolis-Hastings algorithm, i.e. proposals are generated with

$$\theta^z = \theta^i + \epsilon^{i+1},$$

where $\epsilon^{i+1} \sim N(0, cH_m^{-1})$ and c is a scaling parameter.

Prior to optimisation of the posterior p and estimation using the RWMH algorithm some parameters in θ are reparameterised to make the parameter space unbounded. The reparameterisation serves two purposes: first, it makes optimisation easier and, second, the efficiency of sampling using the RWMH algorithm is improved because the approximation to normality is better. In the context of prefetching the reparameterisation will thus also improve the parallel efficiency of the algorithms which rely on the posterior approximation.

For each combination of prefetching algorithm and number of processors P the RWMH prefetching algorithm is used to sample $R = 500,000$ draws from the posterior distribution. In total the posterior is evaluated $\tilde{R} = NP = RP/\bar{d}$ times where N is the number of tours and $\bar{d}$ is the average number of draws per tour. In all estimations the chain is started at the posterior mode and $50,000$ draws are discarded as burn-in. A target acceptance rate $\tilde{\alpha} = 0.25$ is used to construct the tours in static prefetching (2) and dynamic prefetching based on the uniform random numbers (3) and the average acceptance rate α turns out to be 0.24. Later on we allow $\tilde{\alpha}$ to vary with P for the static prefetching

Table 1 Performance of prefetching algorithms for the linearised DSGE model. Average number of draws per tour, $\bar{d}$.

	Processors					
Algorithm variant	1	3	7	15	31	∞
1. Basic prefetching	1	2	3	4	5	∞
2. Static prefetching	1	2.31	3.54	4.75	6.00	∞
3. Dynamic prefetching, uniform	1	2.38	3.81	5.04	6.24	∞
4. Dynamic prefetching, post. approx.	1	2.57	4.58	6.45	8.04	-
5. Most likely path	1	2.66	4.99	7.31	8.42	8.70
6. Optimal static prefetching speedup	1	2.44	4.30	6.55	6.55	-
Optimal empirical acceptance rate	0.22	0.16	0.13	0.09	0.05	

algorithm. Detailed results on prefetching performance are presented for $P = 2^p - 1$, $p = 1, ..., 5$. These values are chosen because they allow for a neat comparison with the benchmark basic prefetching algorithm, which has integer draws per tour for these values of P (see expression [6]).

In table 1 the average number of draws per tour $\bar{d}$ are presented. It is seen, first, that static prefetching allows us to make easily reaped, although quite small, gains in comparison with the basic algorithm. Second, incorporating knowledge about the sequence of uniform random numbers lead to additional small gains. These efficiency improvements are model independent and can always be obtained.

Third, in our example the inclusion of information about the posterior via the normal approximation yields the largest gains. For example, using the most likely path algorithm (5) with $P = 7$ processors the extra draws, i.e. the draws which are added to the sure draw, is doubled in comparison with the basic algorithm (1). If a reasonable approximation to the posterior is available it thus appears possible to improve the parallel efficiency of prefetching quite substantially.

Finally, we report empirical measures of the static prefetching optimal speedup and optimal acceptance rates for this problem (6). Note that these quantities are adjusted for differences in sampling efficiency when targeting different acceptance rates. Thus direct comparison with the other approaches is possible. The experiments conducted to obtain these numbers are described in appendix A. Also note that we have not been able to increase the quality-adjusted speedup for the most likely path algorithm (5) by targeting lower acceptance rates in this way.

The results reported above must be interpreted with caution. If the parallel performance of the most likely path algorithm is becoming 'too good' it suggests that other approaches, such as independence chain Metropolis-Hastings (ICMH), should be considered. For our estimation problem we have verified that the sampling efficiency of ICMH using standard choices of proposal densities, e.g. the multivariate t, is inferior to the efficiency of RWMH. The acceptance rate of the best ICMH sampler is 13%. If the thinning

factor for ICMH is roughly eight times the factor used with the RWMH sampler it is found that the relative numerical efficiencies, or the inefficiency factors, of the two approaches are roughly similar.

4.1.4 Parallel efficiency on an HPC cluster

Draws per tour, $D(P)$, is an abstract measure of scalability since the cost of constructing tours and communication cost is not taken into account. In order to assess the magnitude of the difference between theoretical and actual speedup, $D(P) - S(P)$, the algorithms are taken to a parallel computer. The prefetching algorithm is implemented using Fortran and the Message Passing Interface (MPI) and tested on the Lenngren cluster at the Center for Parallel Computers (PDC) at the Royal Institute of Technology (KTH) in Stockholm, a high performance computing (HPC) environment. The cluster uses Intel Xeon 3.4GHz processors connected by an Infiniband 1GB network. The MPI implementation on the cluster is Scali MPI. Further information on the performance characteristics of the cluster is available at *www.pdc.kth.se*.

For each combination of P and algorithm an acceptance rate $\tilde{\alpha} = 0.25$ is targeted and $R = 500,000$ draws from the posterior are obtained. In table 2 the draws per tour D and relative speedup S, as defined in [3], are reported for $P = 2^p, p = 0, .., 5$ for the static prefetching algorithm (2) and the best performing algorithm according to table 1, the most likely path algorithm (5). Using one processor $R = 500,000$ draws are obtained in 80 minutes and using $P = 8$ processors the static prefetching algorithm (2) executes in 23 minutes and the most likely path algorithm (5) in 15 minutes.

It is seen that for the particular model, programming language, implementation and hardware the speedup is acceptably close to the upper bound, at least for $P = 1 - 8$. We conclude that on a representative HPC cluster the RWMH prefetching algorithm can be implemented successfully for a fine-grained problem. For the large-scale models mentioned above we expect $S \approx D$ in this environment.

4.1.5 Discussion

In an article written by researchers at the Riksbank, the central bank of Sweden, some econometric issues related to the estimation of a large-scale linearised open economy DSGE model (the RAMSES model) are addressed (Adolfson, Lindé and Villani (2007)). We present some of their observations because we believe they are representative to this field of research:

1. Computing time is a major concern when Bayesian methods are employed to analyse large-scale DSGE models.

2. The one-block RWMH algorithm has been the common choice of sampler in the estimation of linearised DSGE models. A substantial problem of the one-block

Table 2 Performance of prefetching algorithms for the linearised DSGE model on the Lenngren cluster.

Static prefetching						
Processors, P	1	2	4	8	16	32
Draws per tour, D	1	1.76	2.77	3.77	4.92	6.16
Relative speedup, S	1.00	1.72	2.70	3.46	4.61	5.18
Most likely path						
Processors, P	1	2	4	8	16	32
Draws per tour, D	1	1.88	3.35	5.39	7.36	8.54
Relative speedup, S	1.00	1.82	3.23	5.15	6.45	7.34

ICHM algorithm is that it easily gets stuck for long spells when the parameter space is high-dimensional.

3. Blocking approaches are difficult to implement for these models because full conditional posteriors are not easy to simulate from. Furthermore, this approach requires multiple evaluations of the likelihood per posterior draw.

4. It is found for the RWMH algorithm that decreasing the targeted acceptance rate from 0.25 to 0.10 leaves the inefficiency factors largely unaltered.

5. Reparameterisation increases the efficiency of sampling substantially.

These observations are largely confirmed by our exercise above and taken together we believe that they increase the attractiveness of prefetching methods in the context of large-scale linearised DSGE models.

Computing time The issue of when computing time becomes a real concern is largely context-specific. Clearly in our environment, characterised by a relatively small DSGE model, Fortran code and decent computing power, the rationale for parallel methods is perhaps limited, as indicated by the absolute execution times reported. Even for a model of this size it is however clear that the scope of experiments that can be performed in a given time span is increased.

In a more typical desktop computing environment, and using state-of-the-art DSGE modelling tools like DYNARE or YADA, single runs may take several days for the most recent generation of policy-relevant DSGE models. Total project computing time, including a substantial amount of experimentation, is measured in months.

RWMH vs. ICMH The parallel efficiency of ICMH is far superior to that of RWMH. Using ICMH with $P = 64$ processors to generate one million draws from the posterior distribution of the SW model the total execution time is around 2.5 minutes on the Lenngren cluster and the relative speedup is $S(64) = 63$. Although the ICMH sampler has much to recommend to it, especially in a parallel setting, in our example problem it does not seem straightforward to find an efficient proposal density. In a serial programming context the choice of estimation strategy would certainly be to use the RWMH algorithm. In a parallel framework the trade-offs are slightly more complicated and several factors must be taken into account.

The following example clarifies the trade-offs for our example under the assumption that ICMH requires a thinning factor which is 8 times that of RWMH to achieve roughly the same sampling efficiency. If $P = 128$ processors are used to estimate the model above with the two algorithms, parallel ICMH and RWMH prefetching, then a posterior sample of a given quality, as judged by the inefficiency factor, can be obtained roughly twice as fast for the ICMH algorithm in comparison with prefetching. If $P = 8$ processors are used the execution time of prefetching is more than 5 times faster than for parallel ICMH.

Acceptance rate The fourth observation above is especially interesting in relation to prefetching. As demonstrated above the empirical optimal static prefetching efficiency is not far behind the efficiency of the most likely path approach for the relevant range of processors in our problem. These two approaches are also the simplest to implement, since they imply nonbranched tours.

To conclude a simple heuristic strategy is suggested: choose the scaling of the incremental density to obtain the optimal static prefetching acceptance rate conditional on the number of processors (see figure 5) and use the 'always prepare for a reject ' tour. It can be expected that often this will be reasonably close to the optimal overall prefetching strategy.

4.2 Nonlinear estimation of a DSGE model

In this section the Metropolis-Hastings prefetching algorithm is applied to the problem of estimating a nonlinearly approximated small-scale DSGE model using Bayesian methods. In the previous section we established that in a high performance computing (HPC) environment, i.e. on the Lenngren cluster, the prefetching algorithm works successfully for a fine-grained estimation problem. Our main objective in this section is to assess prefetching performance in a particular personal high performance computing (PHPC) environment: using the parallelism of Matlab (the *Distributed Computing Toolbox, DCT*) on a quad-core desktop computer. Alternatively, and perhaps more correctly, we may interpret this exercise as an evaluation of the parallel functionality of Matlab, using the prefetching algorithm as the test ground. The multi-core/Matlab environment is presumably one of the most accessible and easy-to-use PHPC environments.

The nonlinear estimation example is chosen first because it is an estimation problem that should be sufficiently coarse-grained to deliver reasonable parallel efficiency also in the PHPC environment. Note that for a given model the particle filter based evaluation of the likelihood for the quadratically estimated model is roughly 1000 times slower than the Kalman filter likelihood evaluation for the corresponding linearly approximated model. Second, the discontinuous likelihood approximation in the nonlinear case implies that prefetching methods which rely on a posterior approximation (variants 4 and 5 above) cannot be implemented without modification. In other words, the most successful variants of prefetching in the linear estimation example of the previous section are not readily available to us here.

4.2.1 Model, prior, solution and likelihood

The prototypical small-scale New Keynesian model is borrowed from An (2005). Again, the economic content of the model is largely irrelevant for our purposes here and therefore no discussion of the model is provided.

The policy function of the model is approximated to the second order using the approach of Schmitt-Grohe and Uribe (2004). The approximative solution can be cast in the following state-space form. The state equation is separated into an equation for the exogenous state variables (the shocks)

$$X_{1t} = AX_{1t-1} + \varepsilon_t, \tag{13}$$

and an equation for the endogenous predetermined variables and a subset of the nonpredetermined variables

$$X_{2t} = B\tilde{X}_{t-1} + Cvech(\tilde{X}_{t-1}\tilde{X}_{t-1}^T) + e, \tag{14}$$

where $\tilde{X}_{t-1} = \left[\begin{array}{cc} X_{1t}^T & X_{2t-1}^T \end{array} \right]^T$ and $X_t = \left[\begin{array}{cc} X_{1t}^T & X_{2t}^T \end{array} \right]^T$.

In this way a linear observation equation,

$$Y_t = d + ZX_t + v_t, \tag{15}$$

is obtained. The innovations and measurement errors are assumed to be independent and normally distributed, $\varepsilon_t \sim N(0, Q)$ and $v_t \sim N(0, H)$. The second order policy function approximation of a large class of DSGE models can be cast in this form. The state-space representation of the corresponding linearly approximated model, which was used in the previous example, is obtained by letting $C = 0$ and $e = 0$. The matrices $A(\theta_2)$, $B(\theta_1)$, and $C(\theta_1)$ and the vectors $e(\theta)$ and $d(\theta_1)$ are functions of the parameters of the economic model, which are collected in $\theta = \left(\begin{array}{cc} \theta_1^T & \theta_2^T \end{array} \right)^T$ where θ_1 consists of the structural parameters of the model and θ_2 contains the auxiliary parameters, i.e. the parameters describing the shock processes. The dimensions of the vectors are $n_{x_1} = 3$, $n_{x_2} = 4$, $n_x = 7$, $n_y = 3$, $n_\theta = 13$ and $n_\epsilon = 3$.

The model is estimated on simulated data and we use the same data-generating process, θ^{dgp}, as An (2005) and a prior distribution for θ similar to the one in An's paper. The likelihood function of the model is evaluated using an Adaptive Linear Particle Filter (ALPF), designed for the particular state-space model described by equations [13]-[15] [Strid (2007*a*)]. The particle filter and its application to the nonlinear estimation of DSGE models is not discussed in detail here and the reader is referred to the referenced articles (Arulampalam et al. (2002); Doucet et al. (2000); Fernández-Villaverde and Rubio-Ramírez (2007); Amisano and Tristani (2007)).

4.2.2 Implementation

The estimation routine is implemented in Matlab, using a Fortran mex function only for the systematic resampling (SR) step of the Adaptive Linear Particle Filter. The SR algorithm cannot be implemented as vectorised code, implying that a Fortran implementation of this part of the particle filter is considerably faster than its Matlab counterpart. The likelihood evaluation accuracy of the ALPF filter applied here is *at least* as good as for a standard particle filter with $40,000$ particles and the time of a posterior evaluation is roughly $2.5s$ for this Matlab implementation.

The parallel section of the prefetching algorithm, step 4 of algorithm 1, is implemented using the parallel for-loop (*parfor*) construct contained in the Parallel Computing Toolbox. The number of calls to *parfor* is thus equal to the total number of tours, N.

4.2.3 Experiment

The Metropolis-Hastings prefetching algorithm is executed on an Opteron 275, 2.2 Ghz, using three of the four available cores. Three variants of prefetching are tested: (i) static prefetching, (ii) dynamic prefetching based on the uniform random numbers and (iii) a modified most likely path (MLP) algorithm. The modification of the MLP algorithm in (iii) consists of updating the mode of the normal approximation to the posterior in order to obtain a successively better approximation, even though the exact mode cannot be obtained. This type of adaptation only affects the quality of prefetching and not the statistical properties of the chain. The starting mode of the approximation is the linear posterior mode.

An acceptance rate $\tilde{\alpha} = 0.25$ is targeted and the actual rate is 0.24. The potential suboptimality of this choice for $P = 3$ is disregarded. In each case $50,000$ draws from the posterior are obtained. Draws per tour and speedup are presented in Table 3. First, the problem is sufficiently coarse-grained for implementation in the particular PHPC environment, in the sense that absolute speedup is acceptably close to draws per tour. Second, the results suggest that it is hard to improve on static prefetching when a good posterior approximation is not immediately available. All three variants roughly halves estimation time when three cores are used, from 35 to 17 hours of computing time.

Table 3 Performance of prefetching algorithms for the nonlinear DSGE model in a multi-core/Matlab environment.

	Static prefetching		Dynamic prefetching		Modified MLP	
Processors	1	3	1	3	1	3
Draws per tour	1	2.32	1	2.37	1	2.32
Speedup	1.0	2.09	1.0	2.11	1.0	2.06

5 Two-layer parallelism

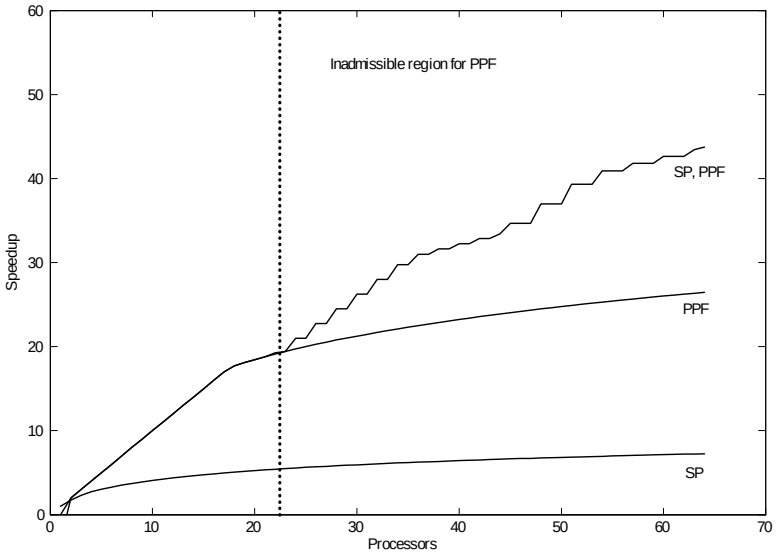
In this section we briefly discuss the possibility of either (i) combining prefetching with lower level parallelism or (ii) using prefetching performance to evaluate competing parallel algorithms. In the context of the Metropolis-Hastings algorithm lower level, and hence competing, parallelisation means any type of within-draw parallelisation of the posterior evaluation. Since prefetching is a general, i.e. largely problem independent, single-chain parallel algorithm it can be used to suggest admissible regions for the number of processors used by the lower level parallel algorithm. Competing parallel algorithms are necessarily problem-specific, presumably more complicated both to develop and implement and they certainly require more frequent communication between processors. This suggests a natural benchmark role for the prefetching algorithm.

The potential performance gain of combining prefetching with a lower level parallel algorithm is illustrated using the nonlinear estimation example considered above. A Parallel Standard Particle Filter (PPF) is used for the likelihood evaluation. The speedup numbers for the PPF refer to Fortran/MPI implementations run on the Lenngren cluster (Strid (2007*b*)). The number of particles employed is $N = 40,000$. For the static prefetching (SP) algorithm an acceptance rate of $\alpha = 0.25$ is targeted.

In figure 7 it is seen that the PPF is much more efficient than the prefetching approach. (The kink in speedup for the PPF is explained by the fact that we estimate a speedup function based on a set of observations $S(P)$, $P = 1, 2, 4, ..., 64$, while ruling out the possibility of superlinear speedup.) However, when the number of processors exceed $P^*_{opt} = 22$ it becomes optimal to switch to the two-layer parallel algorithm which combines static prefetching with the PPF. As the number of processors increase the speedup difference between the two-layer algorithm and the PPF becomes quite pronounced. It is also possible that a minor improvement in speedup, adjusted for statistical efficiency, can be achieved by targeting a lower acceptance rate for prefetching.

The simple calculations underlying figure 7 show how consideration of the prefetching alternative can guide decisions on the tolerable number of processors to use for the lower level parallel algorithm for a given estimation problem and problem size (here the number of particles employed in the particle filter). Importantly this type of scalability analysis can be performed without ever implementing the prefetching algorithm.

Figure 7 Two-layer parallel performance. Static prefetching and a parallel particle filter applied for the estimation of a nonlinear DSGE model.



6 Conclusions

Prefetching algorithms have obvious limitations in terms of parallel efficiency. This is due to the inherently sequential nature of the Metropolis-Hastings algorithm. The advantages of the independence chain Metropolis-Hastings algorithm in a parallel context are obvious.

In this paper we have shown how to substantially increase the efficiency of prefetching using some simple techniques. Even using these techniques on reasonably well-behaved, unimodal, problems it is hard to imagine anyone applying more than, say, $10-15$ processors to a prefetching algorithm. Despite this the simplicity of implementing the method and the possibility of a multiplication effect if combined with lower level parallelisation were claimed to motivate interest in the method.

The paper has highlighted some complicated trade-offs. First, for the prefetching algorithm a parallel efficiency perspective suggests targeting of a low acceptance rate. This must be weighted against Markov chain efficiency considerations. For the random walk Metropolis-Hastings algorithm it was demonstrated that the optimal acceptance rate decreases with the number of processors applied to the algorithm. Second, 'good' scalability of prefetching when a posterior approximation is used to construct tours may indicate that other sampling approaches should also be considered.

Bayesian estimation of DSGE models was identified as one potential arena for prefetching methods. In this context the experiences reported by researchers at the Riksbank,

which largely correspond to the results reported here for a smaller model, suggest that prefetching methods is a viable alternative in reducing estimation time. The generality of the proposed method, however, suggests that it could be applied in other contexts as well. Brockwell (2006) applies prefetching to long memory time series models. The application of prefetching with the hyperblock sampling approach for GMRF models and in related contexts should be more thoroughly explored in future research.

Finally, it would be straightforward to implement a prefetching version of the adaptive RWMH algorithm proposed by Haario et al. (2001). In many cases we expect the proposal distribution of the RWMH and the posterior approximation used to make prefetching predictions to coincide, possibly with a different scaling as in our linear estimation example. The adaptive RWMH can then potentially increase statistical and parallel efficiency simultaneously.

Appendix A

The prefetching algorithm

Let $L = L(i_p)$ be the function that maps Metropolis tree node indices to levels, e.g. $L(10) = 4$. The expected number of draws $D(P)$ for the tour $T(P) = \{i_1, ..., i_P\}$ is given by

$$D(P) = \sum_{p=1}^{P} p \Pr(\tilde{D} = p) = \sum_{p=1}^{P} p \left[\Pr(\tilde{D} \geq p) - \Pr(\tilde{D} \geq p+1) \right] =$$

$$= \sum_{p=1}^{P} \Pr(\tilde{D} \geq p) = \sum_{p=1}^{P} \left(\sum_{i_p : L(i_p) = p} \Pr(i_p) \right) = \sum_{p=1}^{P} \Pr(i_p), \tag{16}$$

where $\Pr\left(\tilde{D} = p\right)$ is the probability of obtaining p draws and $\Pr(i_p)$ is the probability of reaching node i_p of the Metropolis tree, where $\Pr(i_p) = 1$.

A general prefetching algorithm which constructs the optimal tour, i.e. the tour that maximises [16], conditional on the probabilities assigned to the branches of the Metropolis tree, is presented below. The algorithm satisfies the obvious need to avoid calculation of all branch probabilities up to level P in order to obtain a tour of size P. Instead the number of probabilities that must be calculated grows linearly in P. Observe that if i_p belongs to the tour then its parent must belong to the tour. This 'connectedness property' is used to restrict the number of probabilities calculated.

The expected number of draws per tour may be written recursively as

$$D(P) = D(P-1) + \Pr(i_P),$$

and the nodes of the optimal tour satisfy

$$1 = \Pr(1) = \Pr(i_1) \geq ... \geq \Pr(i_P) > 0,$$

such that the marginal value of an additional processor is decreasing.

Algorithm 4 *Prefetching (tour construction)*

1. Set $p = 2$, $D(1) = 1$, $T(1) = \{i_1; \theta_{i_1}\}$ where $i_1 = 1$ and define the sets $\Pi(1) = I(1) = \varnothing$.

2. When the p^{th} step begins the tour is $T(p-1) = \left\{i_1, ..., i_{p-1}; \theta_{i_1}, ..., \theta_{i_{p-1}}\right\}$ and i_{p-1} was added to the tour in the previous iteration. Construct a set of candidate nodes/indices

$$I(p) = I(p-1) \cup \{2i_{p-1}, 2i_{p-1}+1\}.$$

3. Construct a set containing the probabilities of reaching the candidate nodes

$$\Pi(p) = \Pi(p-1) \cup \{\Pr(2i_{p-1}), \Pr(2i_{p-1}+1)\}.$$

The required probabilities are

$$\Pr(2i_{p-1}) = \pi_p(L) \Pr(i_{p-1}) \tag{17}$$

and

$$\Pr(2i_{p-1}+1) = (1 - \pi_p(L)) \Pr(i_{p-1}), \tag{18}$$

where $\Pr(i_{p-1})$ is available from the previous iteration and where $L = L(i_{p-1})$. The probabilities attached to the branches of the Metropolis tree, π_p, define the different variants of prefetching and they are given below.

4. Find the index $i_{\max}$, the node with the largest probability of being reached among the candidate nodes in $I(p)$, i.e. $\Pr(i_{\max}) = \max \Pi(p)$ and let $i_p = i_{\max}$.

5. Add i_p to the tour, $T(p) = T(p-1) \cup \{i_p\}$, and calculate the maximum expected draws per tour, $D(p) = D(p-1) + \Pr(i_p)$. Then remove i_p and $\Pr(i_p)$ from $I(p)$ and $\Pi(p)$ respectively but store $\Pr(i_p)$ temporarily for calculation of [17] and [18] in the next iteration.

6. If $p < P$ go to step 2, otherwise stop. ⊡

Variants of prefetching

At iteration p of the algorithm above a node i_p is allocated to the p^{th} process. The node i_{p-1} was added in the previous iteration. Note that for branched algorithms i_p is not necessarily the child of i_{p-1}. For a nonbranched algorithm the p^{th} processor will necessarily evaluate a parameter at level p in the tree, i.e. $i_p = 2i_{p-1}$ or $i_p = 2i_{p-1}+1$ and hence $L(i_p) = p$. For branched algorithms at least one node i_p will satisfy $L(i_p) < p$.

The static prefetching algorithm (2) is defined by

$$\pi_p = \alpha,$$

and the basic prefetching algorithm (1) is obtained as the special case

$$\pi_p = \alpha = 0.5.$$

The prefetching algorithm which utilises information from the sequence of uniform random numbers (3) is obtained when

$$\pi_p = \alpha\left(u^L\right),$$

where L is the level of the parent

$$L = L\left(i_{p-1}\right).$$

The conditional acceptance rate, [10], is repeated here

$$\alpha\left(u^L\right) = \sum_{k=1}^{K} \alpha^k I\left(u^L \in I_k\right),$$

where $I_k = [\frac{(k-1)}{K}, \frac{k}{K}]$ and where α^k was defined in [9]. Note that the uniform random numbers $\{u^p\}_{p=1}^P$ are connected to levels 1 to P of the Metropolis tree. Since the random numbers are draw-specific the attachment of random numbers to levels of the Metropolis tree requires that we keep track of the number of draws obtained previously.

The algorithm which is based on a posterior approximation (4) is obtained when

$$\pi_p = \min\left\{\beta, \frac{p^*\left(\theta_{i_{p-1}}\right)}{p^*\left(\theta_{i_{p-1}} - \epsilon^L\right)}\right\},$$

where $0 < \beta \leq 1$ and ϵ^L is the random vector associated with level $L\left(i_{p-1}\right)$. Note that $\theta_{i_{p-1}} - \epsilon^L$ is the *state* associated with node i_{p-1}. The random vectors $\{\epsilon^p\}_{p=1}^P$ are specific to levels 1 to P of the Metropolis tree, in the same way as the uniform random numbers.

The most likely path algorithm (5) is defined by

$$\pi_p = I\left\{u^{p-1} < \min\left\{1, \frac{p^*\left(\theta_{i_{p-1}}\right)}{p^*\left(\theta_{i_{p-1}} - \epsilon^{p-1}\right)}\right\}\right\},$$

since the tour is nonbranched and thus we know that $L = L\left(i_{p-1}\right) = p - 1$.

For the latter two algorithms, with our notation it is more convenient to express the posterior ratios in terms of proposed parameters. In implementations it is of course easy to keep track of the associated states as well.

Optimal static prefetching for the linear DSGE model

The DSGE model is estimated ten times, targeting acceptance rates $\alpha_1, ..., \alpha_{10}$ evenly spread in the interval $(0.015, 0.40)$. In each estimation $500,000$ draws from the posterior are obtained and the mean $\bar{\tau}(\alpha)$ of the inefficiency factors

$$\hat{\tau}_j(\alpha) = 1 + 2\sum_{i=1}^{I} Corr\left(\theta_j^t, \theta_j^{t+i}\right),\ j = 1, ..., n_\theta,$$

is calculated for each α. The estimated output is $\hat{E}(\alpha, P) \propto D(\alpha, P)\,\tau_m/\bar{\tau}(\alpha)$ where $\bar{\tau}(\alpha)$ is approximated using simple linear interpolation on the grid of acceptance rates and $\tau_m = \min_\alpha \bar{\tau}(\alpha)$. The empirical optimal acceptance rates across processors obtained in this way are quite similar to those in figure 5 and therefore we are content with reporting only the values that appear in table 1.

Appendix B

The one-block sampler for Gaussian Markov random field models

The reader is referred to the articles referenced below and Knorr-Held and Rue (2005) for a treatment of Gaussian Markov random field (GMRF) models and their use in various areas of statistics. Let $x = (x_1, ..., x_n)^T$ be a GMRF which depends on a vector of hyperparameters, θ, with prior density $p(\theta)$. Let y denote the data and assume that the posterior density of interest is of the form

$$p(x, \theta|y) \propto p(x|\theta)\, p(\theta) \prod_i p(y_i|x_i),$$

where $p(y_i|x_i)$ is the likelihood for one observation.

The current state of the chain is denoted $\eta^i = (x^i, \theta^i)$ and $\tilde{\eta} = (\tilde{x}, \tilde{\theta})$ is the state at the start node of a tour. Knorr-Held and Rue (2002) suggest the updating scheme

$$\theta_z \sim f_1\left(.|\theta^i\right), \qquad x_z \sim f_2\left(.|\theta_z, y, x^i\right), \tag{19}$$

where the proposal $\eta_z = (\theta_z, x_z)$ is accepted/rejected jointly using a Metropolis-Hastings step. The vector of hyperparameters θ may be sampled using a random walk step. If the likelihood function is Gaussian the full conditional posterior of the field is a GMRF and it is used to sample the proposal, i.e. $f_2(.|\theta_z, y, x^i) = p(.|\theta_z, y)$. Otherwise f_2 is a GMRF approximation to this conditional. GMRFs can be sampled using the fast sampling methods presented by Rue (2001). The proposed joint update is accepted with probability $\min\{1, A\}$ where

$$A = \frac{p\left(\theta_z\right) p\left(x_z|\theta_z\right) p\left(y|\theta_z, x_z\right) f_2\left(x^i|\theta^i, y, x_z\right)}{p\left(\theta^i\right) p\left(x^i|\theta^i\right) p\left(y|\theta^i, x^i\right) f_2\left(x_z|\theta_z, y, x^i\right)}, \tag{20}$$

if we assume a symmetric proposal for θ.

In the one-block prefetching algorithm presented in the paper generation of proposals is performed serially in the prefetching step. In typical applications in spatial statistics the dimension of the latent field x is large and sampling from the GMRF (approximation) f_2 accounts for a major part of computational time. Therefore it is required that prefetching is applied only to the, typically low-dimensional, set of hyperparameters, θ. The GMRF proposals for the latent field, $x_{i_1}, ..., x_{i_P}$, are then sampled in parallel and conditional on the hyperparameters of the tour $T = \{\theta_{i_1}, ..., \theta_{i_P}\}$.

If the observation model is Gaussian the ratio [20] simplifies to

$$A = \frac{p\left(\theta_z\right) p\left(y|\theta_z\right)}{p\left(\theta^i\right) p\left(y|\theta^i\right)}, \tag{21}$$

and all the strategies for prefetching presented in the paper apply to the vector of hyperparameters, θ. In Knorr-Held and Rue (2002) $p\left(x|\theta, y\right)$ is approximated using a second order Taylor approximation around the current state of the latent variable, x^i, such that $f_2\left(.|\theta_z, y, x^i\right) = \tilde{p}\left(.|\theta_z, y, x^i\right)$. In this case there is only one possibility for prefetching: to target a low acceptance rate in order to obtain the 'always prepare for a reject' static prefetching tour. All proposals, $x_{i_1}, ..., x_{i_P}$, are generated conditional on the latent field at the start node of the tour, $\tilde{x}$, and the process-specific proposal $\theta_{i_p} \sim f_1\left(.|\theta^i\right)$.

It is realised that this strategy also applies in the more general case where there is x dependence in f_1 since all θ_{i_p} are generated conditional on the state at the start node, $\tilde{\eta}$. This proposal is suggested by Wilkinson and Yeung (2004) in the context of linear Gaussian directed acyclic graph (LGDAG) models.

Further, if the proposal in [19] is used with the LGDAG model parallel sampling could proceed in two stages. First the marginal chain $\{\theta^i\}_{i=1}^R$ is obtained using prefetching. The acceptance probability is given by [21]. Next the latent variables $\{x^i\}_{i=1}^R$ are sampled in embarassingly parallel fashion from $f_2 = p\left(x|\theta^i, y\right) = N_C(h^i, K^i)$. Here it is assumed that $h^i = K^i E(x|\theta^i, y)$ and the Cholesky factor G^i of the sparse precision matrix K^i, which are used to evaluate $p\left(\theta|y\right)$ in the first stage, have been stored. The two-stage approach is possible since $p\left(y|\theta\right)$, can be evaluated in a simplified way for the LGDAG model. However, sampling from f_2 is cheap given that the Cholesky factor G is available. Hence we expect the, seemingly wasteful, approach where x is instead sampled *inside* the prefetching algorithm to be more efficient in practice. In this case there is no need to store h^i and G^i after x^i has been sampled.

The crudest approximation considered by Rue et al. (2004) is a GMRF approximation in the mode x^m of $p\left(x|\theta, y\right)$ such that $f_2\left(.|\theta_z, y, x^i\right) = \tilde{p}\left(.|\theta_z, y, x^m(\theta_z)\right)$. In this case all the strategies for θ-prefetching apply. This follows since the current value of the latent field,

x^i, is not explicitly needed to construct the GMRF approximation. However, a 'suitable value' of x is required as a starting value for finding the mode, x^m. A natural candidate in a prefetching context is $\tilde{x}$ or, if it is important to reduce communication requirement, the locally stored mode, $x^m_{i_p}$, from the previous tour. In the latter case the latent field need not be transferred between processors.

A potential weakness of the resulting prefetching algorithm is the inhomogeneous computing time in the optimisation step but we expect this to be of minor importance. Some efficiency loss could be acceptable and, if thought necessary, the number of iterations of the optimiser, e.g. Newton-Raphson, could be fixed at a low number to restore load balance. Note that a cruder approximation yields a lower acceptance rate which, in turn, increases prefetching efficiency. Finally it would be possible to construct a two-layer parallel algorithm using the parallel sampling approach for GMRFs suggested by Steinsland (2007).

Multiple block prefetching

Consider the multiple block Metropolis-Hastings algorithm with B blocks where the parameter vector θ is split into blocks $\theta^0, ..., \theta^{B-1}$. A chain of length R requires $\tilde{B} = RB$ block updates. For notational convenience, assume that the blocks are updated in a fixed order from block 0 to $B-1$ using the proposal densities

$$\theta^b \sim f_b\left(.|\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right),\ b = 0, ..., B-1,$$

where $\theta^{i,>b} = (\theta^{i,b+1}, \theta^{i,b+2}, ..., \theta^{i,B-1})$ and $\theta^{i-1,b}$ is the state of block b when the i^{th} sweep begins. The choice of scanning strategy does not affect prefetching since the scan order for the complete chain is realised at the outset of sampling.

The probability of a move for block b is given by

$$\alpha^b = \min\left\{1, \frac{p\left(\theta^b, \theta^{i,<b}, \theta^{i-1,>b}\right)}{p\left(\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right)} \frac{f_b\left(\theta^{i-1,b}|\theta^b, \theta^{i,<b}, \theta^{i-1,>b}\right)}{f_b\left(\theta^b|\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right)}\right\}, \tag{22}$$

and we must, as before, assume that evaluation of p is 'expensive'. For simplicity we consider only nonbranched tours. In this context a tour is a collection of *proposed block updates* of the form

$$T = \left\{\theta^{\mathrm{mod}(b+1,B)}_{i_1}, \theta^{\mathrm{mod}(b+2,B)}_{i_2}, ..., \theta^{\mathrm{mod}(b+P,B)}_{i_P}\right\},$$

where it is assumed that block b has just been updated when the tour is entered. Tours and sweeps thus overlap and blocks are associated with levels of the Metropolis tree. Let $\tilde{\theta} = \left(\theta^{i,\leq b}, \theta^{i-1,>b}\right)$ be the parameter value at the start node of the tour. For random walk updates it will be desirable to tune each f_b such that bold moves are proposed, in order to decrease block acceptance rates and increase parallel efficiency.

For example, assume $B = 3$, $P = 6$ and that block 1 has just been updated such that the current paraneter is $\tilde{\theta} = (\theta^{i,0}, \theta^{i,1}, \theta^{i-1,2})$. The 'always prepare for a reject' tour is $T = \{\theta_1^2, \theta_3^0, \theta_7^1, \theta_{15}^2, \theta_{31}^0, \theta_{63}^1\}$ and each processor evaluates $p(\theta_{i_p}^b, \tilde{\theta}^{-b})$ where $\theta_{i_p}^b \in T$. The tour produces at least one block update and at most six block updates, i.e. two draws.

A Gibbs step yields perfect prefetching since

$$f_b\left(.|\theta^{i-1,b}, \theta^{i,<b}, \theta^{i-1,>b}\right) = p\left(.|\theta^{i,<b}, \theta^{i-1,>b}\right),$$

implies $\alpha^b = 1$ and hence prefetching does not apply for a pure Gibbs sampler. A Gibbs component within a Metropolis block sampler would be handled by grouping the Gibbs update with a Metropolis update on a single processor, under the assumption that the Gibbs update is 'cheap'. Clearly, whenever a Gibbs step accounts for a dominant part of computational time prefetching is not feasible. An example of this is when the simulation smoother is used to sample the latent variables in linear Gaussian state-space models.

References

Adolfson, M., Laséen, S., Lindé, J. and Villani, M. (2007), 'Bayesian Estimation of an Open Economy DSGE Model with Incomplete Pass-Through', *Journal of International Economics* **72(2)**, 481–511.

Adolfson, M., Lindé, J. and Villani, M. (2007), 'Bayesian Inference in DSGE Models - Some Comments', *Econometric Reviews* **26(2-4)**, 173–185.

Amisano, G. and Tristani, O. (2007), 'Euro Area Inflation Persistence in an Estimated nonlinear DSGE model'. Working paper 754, European Central Bank.

An, S. (2005), 'Bayesian Estimation of DSGE Models: Lessons from Second-order Approximations'. Working Paper, University of Pennsylvania.

Arulampalam, S., Maskell, S., Gordon, N. and Clapp, T. (2002), 'A Tutorial on Particle Filters for On-Line Non-Linear/Non-Gaussian Bayesian Tracking', *IEEE Transactions on Signal Processing* **50(2)**, 174–188.

Azzini, I., Girardi, R. and Ratto, M. (2007), 'Paralellization of Matlab codes under Windows platform for Bayesian estimation: a Dynare application'. Working Paper 1, Euro-area Economy Modelling Centre.

Belaygorod, A. and Dueker, M. (2006), 'The Price Puzzle and Indeterminacy in an Estimated DSGE model'. Working Paper 2006-025, Federal Reserve Bank of St. Louis.

Brockwell, A. (2006), 'Parallel Markov Chain Monte Carlo Simulation by Pre-fetching', *Journal of Computational and Graphical Statistics* **15(1)**, 246–261.

Brockwell, A. E. and Kadane, J. B. (2005), 'Identification of Regeneration Times in MCMC Simulation, with Application to Adaptive Schemes', *Journal of Computational and Graphical Statistics* **14(2)**, 436–458.

Chib, S. and Carlin, B. P. (1999), 'On MCMC Sampling in Hierarchical Longitudinal Models', *Statistics and Computing* **9(1)**, 17–26.

Christiano, L. J., Trabandt, M. and Walentin, K. (2007), 'Introducing Financial Frictions and Unemployment into a Small Open Economy Model'. Working paper 214, Sveriges Riksbank.

del Negro, M., Schorfheide, F., Smets, F. and Wouters, R. (2005), 'On the Fit and Forecasting Performance of New-Keynesian Models'. Working Paper Series, No. 491, European Central Bank.

Doucet, A., Godsill, S. and Andrieu, C. (2000), 'On Sequential Monte Carlo Sampling Methods for Bayesian Filtering', *Statistics and Computing* **10**, 197–208.

Durbin, J. and Koopman, S. J. (2001), *Time Series Analysis by State Space Methods*, Oxford University Press.

Fernández-Villaverde, J. and Rubio-Ramírez, J. F. (2007), 'Estimating Macroeconomic Models: A Likelihood Approach', *Review of Economic Studies* **74(4)**, 1059–1087.

Gamerman, D., Moreira, A. R. and Rue, H. (2003), 'Space-varying regression models: Specifications and Simulation', *Computational Statistics and Data Analysis* **42(3)**, 513–533.

Garside, L. and Wilkinson, D. (2004), 'Dynamic Lattice-Markov Spatio-Temporal Models for Environmental Data', *Bayesian Statistics* **7**, 535–542.

Haario, H., Saksman, E. and Tamminen, J. (2001), 'An adaptive Metropolis algorithm', *Bernoulli* **7(2)**, 223–242.

Hastings, W. (1970), 'Monte Carlo Sampling Methods Using Markov Chains and Their Applications', *Biometrika* **57(1)**, 97–109.

Klein, P. (2000), 'Using the Generalized Schur Form to Solve a Multivariate Linear Rational Expectations Model', *Journal of Economic Dynamics and Control* **24(10)**, 1405–1423.

Knorr-Held, L. and Rue, H. (2002), 'On Block Updating in Markov Random Field Models for Disease Mapping', *Scandinavian Journal of Statistics* **29(4)**, 597–614.

Knorr-Held, L. and Rue, H. (2005), *Gaussian Markov Random Fields: Theory and Applications*, Chapman and Hall.

Lombardi, M. J. (2007), 'Bayesian Inference for Alpha-stable Distributions: A Random Walk MCMC Approach', *Computational Statistics and Data Analysis* **51(5)**, 2688–2700.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. (1953), 'Equation of State Calculations by Fast Computing machines', *Journal of Chemical Physics* **21(6)**, 1087–1092.

Moreira, A. R. and Gamerman, D. (2002), 'Bayesian Analysis of Econometric Time Series Models using Hybrid Integration Rules', *Communications in Statistics. Theory and Methods* **31(1)**, 49–72.

Omori, Y. and Watanabe, T. (2008), 'Block Sampler and Posterior Mode estimation for Asymmetric Stochastic Volatility Models', *Computational Statistics and Data Analysis* **52(6)**, 2892–2910.

Roberts, G., Gelman, A. and Gilks, W. (1997), 'Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms', *The Annals of Applied Probability* **7(1)**, 110–120.

Roberts, G. O. and Rosenthal, J. S. (2001), 'Optimal Scaling for Various Random Walk Metropolis Algorithms', *Statistical Science* **16(4)**, 351–367.

Rosenthal, J. S. (2000), 'Parallel Computing and Monte Carlo algorithms', *Far Eastern Journal of Theoretical Statistics* **4**, 207–236.

Rue, H. (2001), 'Fast Sampling of Gaussian Markov Random Field Models', *Journal of the Royal Statistical Society Ser B.* **63(2)**, 325–338.

Rue, H., Steinsland, I. and Erland, S. (2004), 'Approximating Hidden Gaussian Markov Random Models', *Journal of the Royal Statistical Society* **66(4)**, 877–892.

Schmitt-Grohe, S. and Uribe, M. (2004), 'Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function', *Journal of Economic Dynamics and Control* **28**.

Smets, F. and Wouters, R. (2003), 'An Estimated Stochastic Dynamic General Equilibrium Model of the Euro Area', *Journal of the European Economic Association* **1(5)**, 1123–1175.

Steinsland, I. (2007), 'Parallel Exact Sampling and Evaluation of Gaussian Markov Random Fields', *Computational Statistics and Data Analysis* **51(6)**, 2969–2981.

Strickland, C. M., Forbes, C. S. and Martin, G. (2006), 'Bayesian Analysis of the Stochastic Conditional Duration Model', *Computational Statistics and Data Analysis* **50(9)**, 2247–2267.

Strickland, C. M., Forbes, C. S. and Martin, G. M. (2009), 'Efficient Bayesian Estimation of Multivariate State Space Models', *Computational Statistics and Data Analysis* **53(12)**, 4116–4125.

Strid, I. (2007*a*), 'A Simple Adaptive Particle Filter for Second-Order Approximated DSGE Models'. Mimeo, Stockholm School of Economics.

Strid, I. (2007*b*), 'Parallel particle filters for likelihood evaluation in DSGE models: An assessment'. Mimeo, Stockholm School of Economics.

Strid, I. and Walentin, K. (2008), 'Block Kalman Filters for Large-scale DSGE models', *Computational Economics* **33(3)**, 277–304.

Takahashi, M., Omori, Y. and Watanabe, T. (2009), 'Estimating Stochastic Volatility Models using Daily Returns and Realized Volatility Simultaneously', *Computational Statistics and Data Analysis* **53(6)**, 2404–2426.

Whiley, M. and Wilson, S. P. (2004), 'Parallel Algorithms for Markov Chain Monte Carlo in Latent Spatial Gaussian Models', *Statistics and Computing* **14(3)**, 171–179.

Wilkinson, D. (2006), Parallel Bayesian Computation, *in* E. J. Kontoghiorghes, ed., 'Handbook of Parallel Computing and Statistics', Chapman and Hall, chapter 16, pp. 477–508.

Wilkinson, D. and Yeung, S. K. (2004), 'A Sparse Matrix Approach to Bayesian Computation in Large Linear Models', *Computational Statistics and Data Analysis* **44(3)**, 493–516.

Yan, J., Cowles, M. K., Wang, S. and Armstrong, M. P. (2007), 'Parallelizing MCMC for Bayesian Spatiotemporal Geostatistical Models', *Statistics and Computing* **17(4)**, 323–335.