Speculative Moves: Multithreading Markov Chain Monte Carlo Programs

Jonathan M. R. Byrd, Stephen A. Jarvis and Abhir H. Bhalerao

Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK
{J.M.R.Byrd, Stephen.Jarvis, Abhir.Bhalerao}@dcs.warwick.ac.uk

Abstract. The increasing availability of multi-core and multi-processor architectures provides new opportunities for improving the performance of many computer simulations. Markov Chain Monte Carlo (MCMC) simulations are widely used for approximate counting problems, Bayesian inference and as a means for estimating very high-dimensional integrals. As such MCMC has found a wide variety of applications in biology and biomedical analysis.
This paper presents a new method for reducing the runtime of Markov Chain Monte Carlo simulations by using SMP machines to speculatively perform iterations in parallel, reducing the runtime of MCMC programs whilst producing statistically identical results to conventional sequential implementations. We calculate the theoretical reduction in runtime that may be achieved using our technique under perfect conditions, and test and compare the method on a selection of multi-core and multi-processor architectures. Experiments are presented that show reductions in runtime of 35% using two cores and 55% using four cores.

1 Introduction

Markov Chain Monte Carlo (MCMC) is a computational intensive technique that may be used to conduct Bayesian inference, allowing prior knowledge to guide the analysis of image data. For instance, the expected size, distribution and density and organisation of cells in an image. By incorporating expected properties of the solution, the stability of the Markov Chain simulation is improved and the chances of consistent false-positives is reduced. MCMC is also good at identifying similar but distinct solutions (i.e. is an artifact in a blood sample one blood cell or two overlapping cells) and giving relative probabilities or confidence values to the solutions it provides.

MCMC, as a means to implement Bayesian inference, has been successfully applied to many areas of computational biology [1], notably in the field of phylogenetic analysis where several well established implementations exist (MrBayes [2] and others). Although the technique has yet to be applied to biomedical imaging applications to the same extent, there are several examples of its use. In addition to the cell nuclei identification method presented as the case study in this paper, MCMC has been used to map muscle cells using Voronoi polygons [3] and tracing retinal vascular trees [4, 5]. The work in [6] demonstrates the use

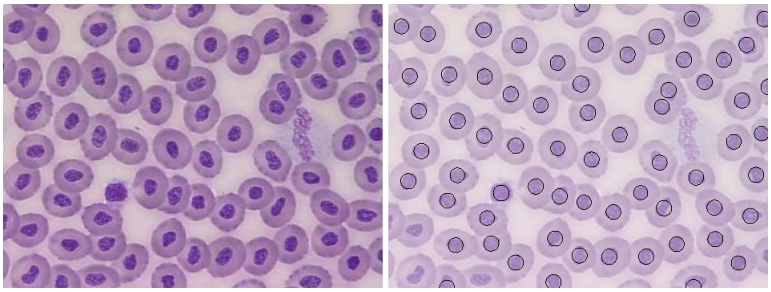

Fig. 1. Demonstration of the feature recognition program used for testing in this paper. Left: The initial image. Right: The stained cell nuclei found after 10,000 MCMC iterations (approx 4 seconds processing time).

of MCMC in solving segmentation problems for prostate and thalamus magnetic resonance images.

Monte Carlo applications are generally considered embarrassingly parallel [7], since samples can be obtained twice as fast by running the problem on two independent machines. This also applies for Markov Chain Monte Carlo, provided a sufficient burn-in time has elapsed and a sufficient number of distributed samples are taken. Unfortunately, for the high-dimensional problems found in biomedical imaging, the burn-in time required for obtaining good samples can be considerable. Although simple applications examining small images have runtimes of seconds, mapping complicated structures or identifying many features in large images can take minutes or hours. As an example, the mapping of retinal vascular trees in as detailed in [4, 5] took upwards of 4 hours to converge when run on a 2.8GHz Pentium 4, and takes much longer to explore alternative modes (additional potential interpretations for the input data). The practicality of such solutions (in real-time clinical diagnostics for example) is therefore limited. High throughput microscopy applications face a similar problem, although individual images may be processed quickly, the large number of images to analyse make reductions in runtime desirable.

If modes are not expected to be radically different, duplicating the simulation will not substantially reduce the runtime, as the time required for convergence will dominate over the time taken collecting samples. Statistical techniques already exist for improving the rate of convergence, indeed most current optimisation and/or parallelisation strategies take this approach. In this paper we focus on complimenting such methods by reducing the runtimes of MCMC applications through implementational changes rather than by modifying the statistical algorithm. The contributions of this paper are threefold:

- We propose a new method (termed 'speculative moves') of implementing Markov Chain Monte Carlo algorithms to take advantage of multi-core and multi-processor machines;
- We fully implement and test this method on a number of different machine architectures and demonstrate the suitability of these architectures for this

new approach. We also demonstrate an example of this technique in processing biomedical images: a program that finds stained cell nuclei, as seen in figure 1;
– Finally, we provide a method for predicting the runtime of MCMC programs using our speculative moves approach, therefore providing: (i) increased certainty in real-world MCMC applications, (ii) a means of comparing alternative supporting architectures in terms of value for money and/or performance.

'Speculative moves' may be used alongside most existing parallelisation and optimisation techniques whilst leaving the MCMC algorithm untouched, and so may safely be used without fear of altering the results. Our method is designed to operate on the increasingly available multiprocessor and multicore architectures. As the technology improves (e.g. by increasing the number of processing cores that are placed onto a single chip) the speculative moves method will yield greater runtime reductions over a wider range of applications.

The remainder of this paper is organised as follows. In section 2 we explain the MCMC method and the difficulties in parallelising it. Section 3 reviews the current forms of parallel MCMC. Our method of speculative moves is outlined in section 4, with the theoretical improvements possible calculated in section 5. We introduce a real-world case study to which we applied speculative moves in section 6 and review the results in section 7. Section 8 concludes the paper.

2 Markov Chain Monte Carlo

Markov Chain Monte Carlo is a computationally expensive nondeterministic iterative technique for sampling from a probability distribution that cannot easily be sampled from directly. Instead, a Markov Chain is constructed that has a stationary distribution equal to the desired distribution. We then sample from the Markov Chain, and treat the results as samples from our desired distribution. For a detailed examination of the MCMC method the reader is referred to [8]. Here we provide a summary of what the algorithm does in practise, excluding much of the theoretical background. At each iteration a transition is proposed to move the Markov Chain from state x to some state y, normally by making small alterations to x. The probability of applying this proposed move is calculated by a transition kernel constructed in such a way that the stationary distribution of the Markov Chain is the desired distribution. The construction of a suitable kernel is often surprisingly easy, and is frequently done by applying Bayesian inference [9]. Such kernels produce the probability for advancing the chain to state y from x based on how well y fits with the *prior* knowledge (what properties the target configuration is expected to have) and the *likelihood* of y considering the actual data available. Transitions that appear to be favourable compared to the current state of the chain have acceptance probabilities > 1 and so are accepted unconditionally, whilst moves to apparently worse states will be accepted with some reduced probability. Once the move/transition has been either accepted (causing a state change) or rejected the next iteration begins. This

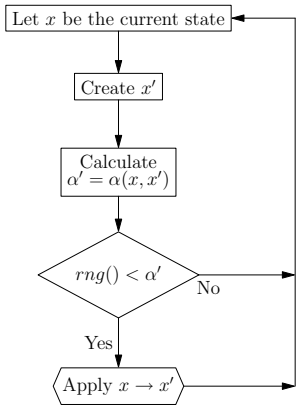

Fig. 2. Conventional Markov Chain Monte Carlo Program Cycle - one MCMC iteration is performed at each step of the cycle. $rng()$ generates a uniform random number between 0 and 1.

program cycle is shown in figure 2. MCMC can be run for as many iterations as are required: the conventional use is to keep taking samples of the chain's state at regular intervals after an initial burn-in period to allow the chain to reach equilibrium. Depending on the needs of the application these samples will either be the subject of further processing or compared to identify the 'best' (most frequently occurring characteristics amongst the samples). In some applications (typically those dealing with high-dimensional states, such as for image processing problems) a single sample of a chain that has reached equilibrium (converged) may be enough. Determining when a chain has converged (and therefore may be sampled) is an unsolved problem beyond the scope of this paper.

This paper concerns parallelising MCMC applications where the initial burn-in time is the most time-consuming period. Obtaining many samples is embarrassingly parallel as multiple chains can be run on multiple computers, each using a different initial model but keeping all other factors the same. Samples from all the chains can be simply grouped [7], not only reducing the time to obtain a fixed number of samples but also reducing the chances that all the samples will occur in local rather than global optima, since the chains will be starting from different positions in the state-space. However, running multiple chains does not change the initial burn-in time (waiting for the chains to move from their initial models to achieving equilibrium around optimal states), which for biomedical imaging applications (being complex and high-dimensional) may be considerable.

3 Related Work

The conventional approach to reducing the runtime of MCMC applications is to improve the rate of convergence so that fewer iterations are required. The main parallel technique is called Metropolis-Coupled MCMC (termed $(MC)^3$)

[10, 11], where multiple MCMC chains are performed simultaneously. One chain is considered 'cold', and its parameters are set as normal. The other chains are considered 'hot', and will be more likely to accept proposed moves. These hot chains will explore the state-space faster than the cold chain as they are more likely to make apparently unfavourable transitions, however for the same reason they are less likely to remain at near-optimal solutions. Whilst samples are only ever taken from the cold chain, the states of the chains are periodically swapped, subject to a modified Metropolis-Hastings test. This allows the cold chain to make the occasional large jump across the state-space whilst still converging on good solutions.

$(MC)^3$ differs from our work in that communication between chains is infrequent, thus the chains can be executed across networked computers. The aims are also very different - $(MC)^3$ increases the mixing of the chain, improving the chances of discovering alternative solutions and helping avoid the simulation becoming stuck in local optima. Essentially it reduces the number of iterations required for the simulation to converge, whereas our method reduces the time required to perform a number of iterations. The two approaches will complement each other, particularly since $(MC)^3$ permits its chains to be spread over multiple computers connected by comparatively low speed interconnects.

4 Speculative Moves: A New Parallelising Approach

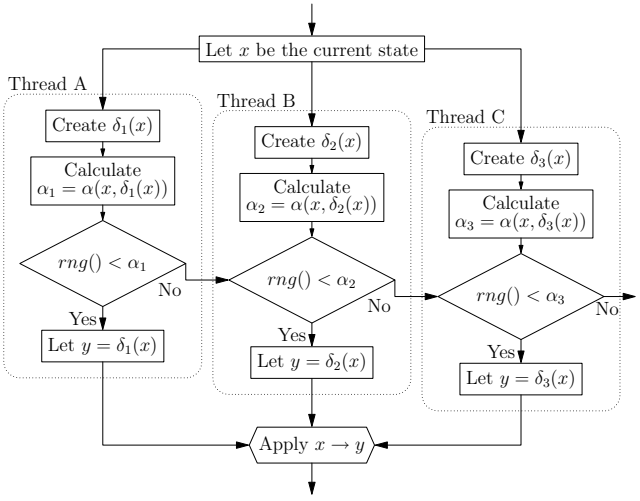

Fig. 3. Speculative Move Enabled Program Cycle. In this case three potential moves are considered at each step of the program cycle. This translates to one, two or three MCMC iterations being performed, depending on whether the first and second potential moves are accepted or rejected.

Although by definition a Markov chain consists of a strictly sequential series of state changes, each MCMC iteration will not necessary result in a state change. In each iteration (see figure 2) a state transition (move) is proposed but applied subject to the Metropolis-Hastings test. Moves that fail this test do not modify the chain's state so (with hindsight) need not have been evaluated. Consider a move 'A'. It is not possible to determine whether 'A' will be accepted without evaluating its effect on the current state's posterior probability, but we can assume it will be rejected and consider a backup move 'B' in a separate thread of execution whilst waiting for 'A' to be evaluated (see figure 3). If 'A' is accepted, the backup move 'B' - whether accepted or rejected - must be discarded as it was based upon a now supplanted chain state. If 'A' is rejected, control will pass to 'B', saving much of the real-time spent considering 'A' had 'A' and 'B' been evaluated sequentially. Of course, we may have as many concurrent threads as desired, so control may pass to move 'C' if 'B' is rejected, then 'D', 'E', and so on. Clearly for there to be any reduction in runtime, each thread must be executed on a separate processor or processor core.

To be useful the speculative move must not compete with the initial move for processor cycles. In addition, the communication overhead for synchronising on the chain's current state, starting the speculative moves and obtaining the result must be small compared to the processing time of each move. An SMP architecture is most likely to meet these criteria, although a small cluster might be used if the average time to consider a move is long enough. As many speculative moves may be considered as there are processors/processing cores available, although there will be diminishing returns as the probability of accepting the mth speculative move is $(p_r)^{m-1}(1-p_r)$ where p_r is the probability of rejecting any one move proposal.

Since speculative moves compress the time it takes to perform a number of iterations, the method will complement existing parallelisation that involves multiple chains to improve mixing or the rate of convergence (such as $(MC)^3$ or simply starting multiple chains with different initial models), provided sufficient processors are available. As the other parallelisation methods tend to be less tightly coupled, it is feasible for physically distinct computers to work on different chains, whilst each chain makes use of multiple cores/processors on its host computer for speculative moves.

5 Theoretical Gains

When using the speculative move mechanism with n moves considered simultaneously, the program cycle consists of repeated 'steps' each performing the equivalent of between 1 and n iterations. The moves are considered in sequence, once one move has been accepted all subsequent moves considered in that step must be ignored.

Given that the average probability of a single arbitrary move being rejected is p_r, the probability of the i^{th} move in a step being accepted whilst all preceding moves are rejected is $p_r^{i-1}(1-p_r)$. Such a step counts for i iterations. Including

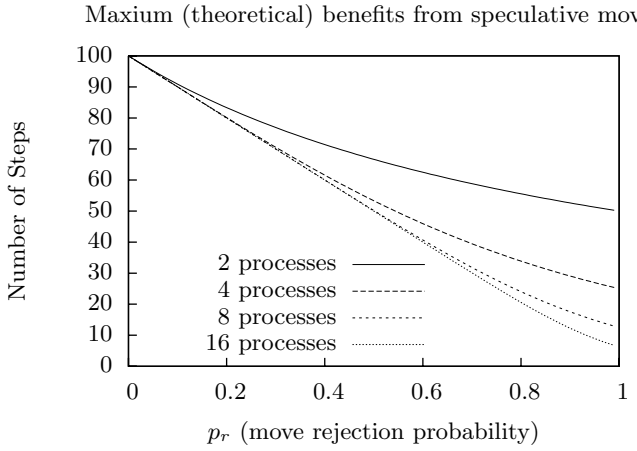

Fig. 4. The number of steps required to perform 100 iterations using multiple processors. The serial implementation performs exactly one iteration in each step, the number of steps will always be 100 irrespective of $\boldsymbol{p_r}$.

the case where all moves in a step are rejected (occurring with probability p_r^n). The number of iterations (I) performed by S_n steps (where n is the number of moves considered in each step) can be expressed as

$$I = S_n \left[\sum_{i=1}^{n} i p_r^{i-1}(1 - p_r) + n p_r^n \right]$$
$$I = S_n \left[\sum_{i=1}^{n-1} p_r^i + 1 \right]$$
$$I = S_n \frac{1 - p_r^n}{1 - p_r}$$

Rearranging for S_n

$$S_n = I \frac{1 - p_r}{1 - p_r^n} \tag{1}$$

which is plotted in figure 4 for varying p_r. Assuming the time taken to apply an accepted move and the overhead imposed by multithreading are both negligible compared to the time required for move calculations, the time per step $\approx$ time per iteration. Therefore figure 4 also shows the limits of how the runtime could potentially be reduced. For example, if 25% of moves in an MCMC simulation are accepted ($p_r = 0.75$), 100 sequential iterations are equivalent to ≈ 57 steps for a two-threaded speculative move implementation or ≈ 37 steps on a four-threaded implementation. Four thread speculative moves could therefore at best reduce the runtime of a MCMC application accepting 25% of its moves by about 63%, while the two threaded version could achieve up to a 43% reduction.

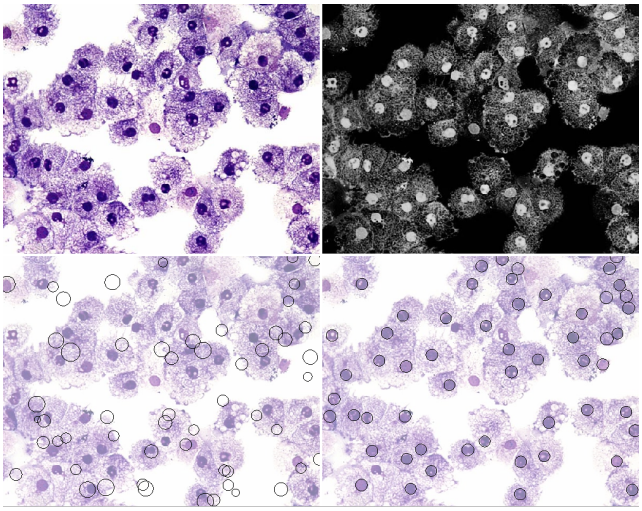

Fig. 5. Demonstration of the feature recognition program. Top left: The initial image. Top right: The filtered image used by the MCMC simulation. Bottom left: The initial randomly generated configuration overlaid on the image. Bottom right: The configuration after 10,000 iterations (approx 4 seconds processing time).

In practise speedups of this order will not be achieved. Threads will not receive constant utilisation (as they are synchronised twice for each iteration) so may not be consistently scheduled on separate processors by the OS. For rapidly executing iterations the overhead in locking/unlocking mutexes and waiting for other threads may even cause a net increase in runtimes. In addition, proposing and considering the moves may cause conflicts over shared resources, particularly if the image data cannot fit entirely into cache. Figure 4 can only be used to estimate the maximum possible speedup, actual improvements will fall short of this by an amount determined by the hardware and characteristics of the MCMC simulation to which speculative moves are applied.

6 Case Study: Finding White Blood Cell Nuclei

An application of MCMC to medical imaging is the identification of artifacts in an image. For instance, the finding of stained cell nuclei. For simplicity we abstract this into the finding of circles of high colour intensity. In this case study the input image is filtered to emphasise the colour of interest. This filtered image can then be used to produce a model for the original image - a list of artifacts i.e. circles defined by their coordinates and radii. A random configuration is generated and used as the initial state of the Markov Chain. At each iteration a type of alteration is chosen at random. The possible alterations are the addition of an artifact, the deletion of an artifact, merging two artifacts together, splitting an artifact into two, and altering the position or properties (i.e. radius) of an existing artifact. A *move proposal* (possible state transition) is then generated

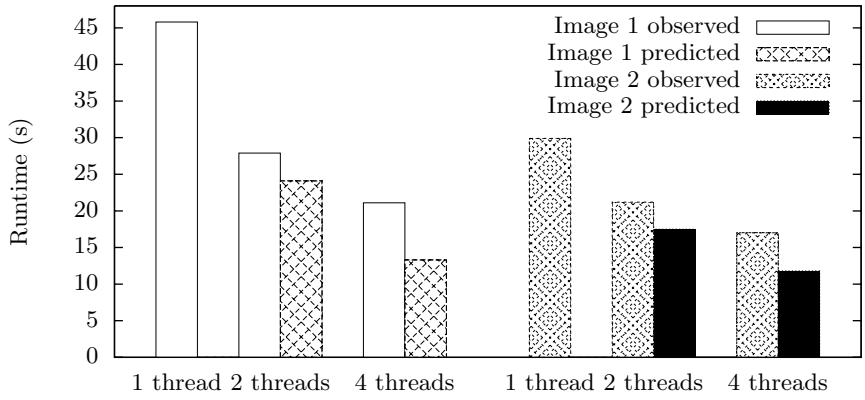

Fig. 6. Runtime for cell-nuclei segmentation program on Q6600 (quad core machine). Image A (left) is an image from which figure 1 was taken, processed with a $\boldsymbol{p_r}$ of 0.9. Figure 5 is a subset of Image B (right) and was processed with a $\boldsymbol{p_r}$ of 0.7.

that implements an alteration of that type, randomly choosing the artifact(s) to alter and the magnitude of that alteration. The probability of accepting this move is generated by a Metropolis-Hastings transition kernel constructed using Bayesian inference.

Two terms, the *prior* and *likelihood*, are calculated to evaluate the configuration that would be created by this move proposal. The *prior* term evaluates how well the proposed configuration matches the expected artifact properties, in this case the distribution and size of the nuclei and the degree to which overlap is tolerated. The *likelihood* of the proposed configuration is obtained by comparing the proposed artifacts against the filtered image. Together the prior and likelihood terms make up the *posterior* probability of the proposed configuration, which is compared with the posterior probability of the existing state using a reversible-jump Metropolis-Hastings transition kernel [12]. Put simply, whilst the prior and likelihood probabilities cannot be expressed exactly, the ratio between the posterior probabilities of the current and proposed configurations can be calculated and used to give a probability for accepting the state change.

7 Results

The images shown in figures 1 and 5 can be processed in a few seconds. To allow for more accurate results, figure 6 shows processing times for images four times as large (∼1200x1000) as those shown in this paper. Note that the difference in the single threaded processing times between images A and B is due to the larger number of cells in image A. Compared to image B, image A has a proportionality larger reduction in runtime for the multithreaded versions due to the higher move rejection rate used to process that image. For a more general examination of runtime improvements using the speculative move system we used a second,

similar application that allowed more direct control over the processing time per iteration and iteration rejection probabilities. Results using this application on two computer architectures (a dual core Pentium-D and a quad core Q6600) are shown in figure 7. The results predicted by equation 1 have been overlaid. Recall that these predictions do not account for multithreading overhead so cannot be completely achieved. Nonetheless there is a good correlation between the predicted and observed results. The discrepancy on the Q6600 using only two threads is a consequence of the its architecture and thread scheduler, its four processing cores are arranged across two chips between which communication is slower than when communicating between cores on the same chip.

When considering whether to implement speculative moves in a MCMC application one must contemplate whether the benefits of speculative moves 'breaks even' with the cost involved in implementing the multithreaded scheme. The breakeven points, when the benefits of speculative moves equals the time spent in multithreading overheads, for a number of different computer architectures, are presented in tables 1 and 2. As a point of reference, the cell nuclei pro-

Table 1. Breakeven point when $p_r = 0.75$

	Iteration Time (μs)	Iteration Rate (s^{-1})
Xeon Dual-Processor	70	14 285
Pentium-D (dual core)	55	18 181
Q6600 (using 2 cores)	75	13 333
Q6600 (using 4 cores)	25	40 000

Table 2. Breakeven point when $p_r = 0.60$

	Iteration Time (μs)	Iteration Rate (s^{-1})
Xeon Dual-Processor	80	12 500
Pentium-D (dual core)	70	14 285
Q6600 (using 2 cores)	130	7 692
Q6600 (using 4 cores)	30	33 333

gram searching for ∼200 circles in a 1200x1000 image (four times the size of the image in figure 5) performed around 3000 iterations per second ($300\mu s$ per iteration), whilst the vascular tree finding program from [4, 5] was generally performing $20 - 200$ iterations per second ($5 - 50ms$ an iteration). Based on this we can expect a non-trivial image processing MCMC application to be well above the iterations per second value needed to break even, and can therefore expect significant real-time savings by using speculative moves for real applications.

8 Conclusion

Using our new 'speculative moves' approach takes advantage of multithreaded machines without altering any properties (other than runtime) of the Markov

Chain. Our method can safely be used in conjunction with other parallelisation strategies, most notably Metropolis-Coupled Markov Chain Monte Carlo where each parallel chain can be processed on a separate machine, each being sped up using speculative moves. Speculative moves can be applied to any MCMC application performing fewer than some tens of thousands of iterations per second (dependant on machine architecture), which will cover most (if not all) non-trivial medical imaging MCMC applications. When the average move rejection rate is high (i.e. $> 50\%$) substantial savings in runtime can be expected. On typical MCMC applications, developers aim for rejection rates of around 75%, in which case runtimes could be reduced by up to 40% using a dual core machine, or up to 60% on a quad core machine.

Multiprocessor machines will provide speed ups, but CPUs with multiple processing cores on the same die will obtain results closer to the maximum theoretical limit. As multicore technology continues to improve the speculative move method will become more potent, applicable to an even wider range of applications and achieving greater performance improvements. By taking advantage of developments in modern and future processor designs speculative moves will help make the use of MCMC-based solutions more productive and increasingly applicable to a wide range of biomedical applications.

References

1. Wilkinson, D.J.: Bayesian methods in bioinformatics and computational systems biology. Brief Bioinform (2007)
2. Huelsenbeck, J.P., Ronquist, F.: MrBayes: A program for the Bayesian inference of phylogeny. Technical report, Department of Biology, University of Rochester (2003)
3. Dryden, I., Farnoosh, R., Taylor, C.: Image segmentation using Voronoi polygons and MCMC, with application to muscle fibre images. Journal of Applied Statistics **33**(6) (2006)
4. Fan, D.C.K.: Bayesian Inference of Vascular Structure from Retinal Images. PhD thesis, University of Warwick (May 2006)
5. Thonnes, E., Bhalerao, A., Kendall, W., Wilson, R.: A Bayesian approach to inferring vascular tree structure from 2D imagery. In: International Conference on Image Processing. Volume 2. (2002) 937–940
6. Fan, A.C., Fisher, J.W., Wells, W.M., Levitt, J.J., Willsky, A.S.: MCMC curve sampling for image segmentation. In: MICCAI 2007. (2007)
7. Rosenthal, J.S.: Parallel computing and Monte Carlo algorithms (1999)
8. Green, P.J. In: Practical Markov Chain Monte Carlo. Chapman and Hall (1994)
9. Green, P.J.: MCMC in action: a tutorial. given at ISI, Helsinki (August 1999)
10. Altekar, G., Dwarkadas, S., Huelsenbeck, J.P., Ronquist, F.: Parallel metropolis-coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. Technical Report 784, Department of Computer Science, University of Rochester (July 2002)
11. Harkness, M., Green, P.: Parallel chains, delayed rejection and reversible jump MCMC for object recognition. In: British Machine Vision Conference. (2000)
12. Green, P.: Reversible jump Markov chain Monte Carlo computation and Bayesian model determination (1995)

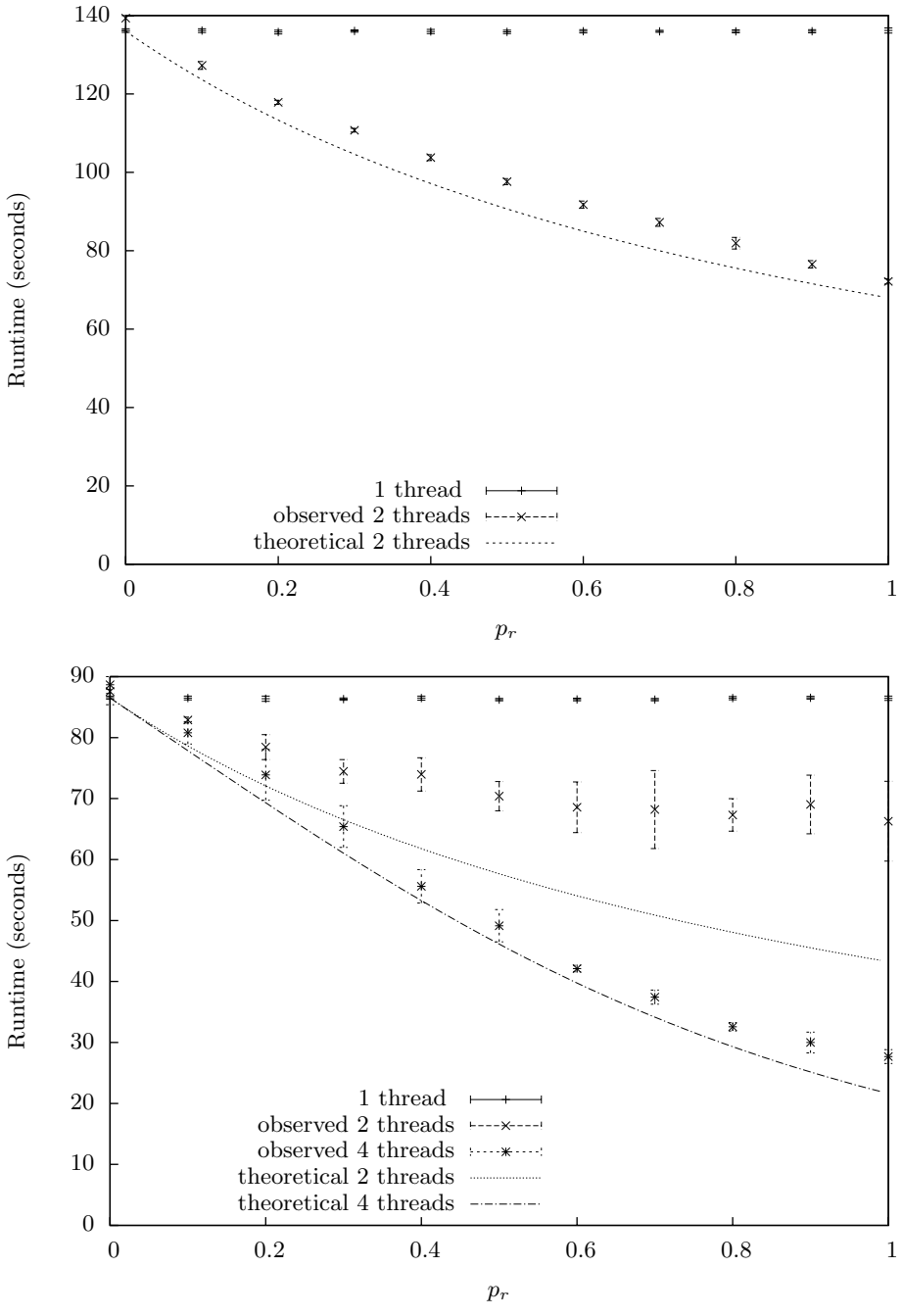

Fig. 7. Runtime plotted against move rejection probability ($\boldsymbol{p_r}$) on the Pentium-D (top) and the Q6600 (bottom).