On the Parallelisation of MCMC-based Image Processing

Jonathan M. R. Byrd
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, UK
Email: jbyrd@dcs.warwick.ac.uk

Stephen A. Jarvis
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, UK
Email: saj@dcs.warwick.ac.uk

Abhir H. Bhalerao
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, UK
Email: abhir@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 fields including computational biology and physics, financial econometrics, machine learning and image processing.**

Whilst divide and conquer is an obvious means to simplify image processing tasks, "naively" dividing an image into smaller images to be processed separately results in anomalies and breaks the statistical validity of the MCMC algorithm. We present a method of grouping the spatially local moves and temporarily partitioning the image to allow those moves to be processed in parallel, reducing the runtime whilst conserving the properties of the MCMC method. We calculate the theoretical reduction in runtime achievable by this method, and test its effectiveness on a number of different architectures. Experiments are presented that show reductions in runtime of 38% using a dual-core dual-processor machine.

In circumstances where absolute statistical validity are not required, an algorithm based upon, but not strictly adhering to, MCMC may suffice. For such situation two additional algorithms are presented for partitioning of images to which MCMC will be applied. Assuming preconditions are met, these methods may be applied with minimal risk of anomalous results. Although the extent of the runtime reduction will be data dependent, the experiments performed showed the runtime reduced to 27% of its original value.

I. Introduction

Markov Chain Monte Carlo (MCMC) is a computationally intensive iterative technique that may be used to conduct Bayesian inference, allowing prior knowledge to guide the analysis of input data. As such it 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 of MCMC in solving segmentation problems for prostate and thalamus magnetic resonance images.

MCMC using Bayesian inference is particularly suited to problems where there is prior knowledge of certain aspects of the solution. For instance when analysing a tissue sample, knowing the expected size, distribution, density and organisation of cells in an image not only allows a MCMC algorithm to map the cells to be created with surprising ease, but improves the stability of the simulation and reduces the chances of consistent false-positives compared to algorithms that do not utilise such information. 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 the relative probabilities of these different interpretations of the original data.

The main limitation of the MCMC method is its runtime, particularly when processing large images. The time per iteration can increase exponentially with the number artifacts to be found in the image, and the total number of iterations required increases with both the number of artifacts and the size of the image. As an example, the mapping of vascular trees in retinal images 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

The obvious means of reducing this runtime via parallel processing is to split the image data into partitions then analyse each partition independently (and, if possible, in parallel). Unfortunately the inherently linear nature of the MCMC algorithm does not (in general) allow such parallel processing to be "naively" applied. The purpose of this paper is to consider the methods and circumstances in which parallel processing by partitioning may be performed without impairing the quality of the end results, allowing faster responses and greater throughput of MCMC-based biological image analysis.

The contributions of this paper are fourfold:

- We propose a new method (termed 'periodic partitioning') of implementing Markov Chain Monte Carlo algorithms that takes advantage of localised spatial properties of images to permit limited parallel processing within the bounds of the MCMC method. We fully implement and test this method on three different machine architectures (a Intel Q6600, a Intel Pentium-D and a AMD Xeon) and demonstrate the suitability of these architectures for this new approach.

- We provide a method for predicting the fastest possible runtime of MCMC programs using our periodic parallelisation approach, either on its own or in conjunction with a complimentary parallelisation method (speculative moves).
- We present two alternative partitioning-based parallelisation methods ('intelligent partitioning' and 'blind partitioning') for use when the statistical backing of the MCMC does *not* need to be preserved. Although the results are application specific there is the potential for much greater reductions in runtime than periodic parallelisation, with the cost of a possible (but for aminiable applications, minimal) loss of accuracy.
- The previous three contributions are demonstrated using an MCMC medical imaging application - the identification of stained cell nuclei in an image.

The remainder of this paper is organised as follows. In section II we explain the MCMC method and the difficulties in parallelising it. To clarify and provide context we introduce the case study which we later use for testing in section III. Section IV reviews the current forms of parallel MCMC. Our method of periodic parallelisation is outlined in section V, the theoretical improvements possible calculated in section VI and the results of applying it to the case study in section VII. The aggressive parallelisation methods that do not strictly preserve the MCMC algorithm are then detailed in section VIII and tested in section IX. Section X concludes the paper.

II. 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 [7]. Here we provide a summary of what the algorithm does in practise, excluding much of the theoretical detail.

At each iterations a transition is proposed to move the Markov Chain from state i to some state j, normally by making small alterations to i. 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. Such kernels produce the probability for advancing the chain to state j from i based on how well j fits with the *prior* knowledge (what properties the target configuration is expected to have) and the *likelihood* of j (considering the actual data available). The basic Metropolis-Hastings transition kernel can be expressed as

$$\alpha = \frac{\text{prior}(j)}{\text{prior}(i)} \times \frac{\text{likelihood}(j|f)}{\text{likelihood}(i|f)} \times \frac{p(j,i)}{p(i,j)} \qquad (1)$$

where $p(j,i)$ is the probability of proposing the move from state j to i. Transitions that appear to be favourable compared to the current state of the chain have $\alpha > 1$ and are accepted unconditionally, whilst moves to apparently worse states will be accepted with probability α. Once the move/transition has been either accepted (causing a state change) or rejected the next iteration begins. MCMC can be run for as many iterations as are required. The conventional use is to allow the chain to reach equilibrium then to take samples of the chains state at regular intervals, analysis of these samples will reveal the stationary distribution. 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 may be enough. Determining when a chain has converged (and therefore may be sampled) is an unsolved problem beyond the scope of this paper.

For the non-trivial MCMC applications found in computational biology, the initial burn-in time is the most time-consuming period. Unfortunately the nature of the MCMC algorithm prohibits simple image partitioning (divide and conquer) in the general case. As each iteration depends on its predecessor, simple parallel execution of iterations is not possible. Working on different areas of the image at the same time causes competing changes in the prior term of the transition kernel, as the prior contain properties that are global to the entire image. "Naively" bisecting an image and considering the two equal halves separately will therefore not yield the same results as processing the entire image at once. Even in the absence of global properties, artifacts that intersect with a partition boundary may be found twice (once in each half of the image), be poorly identified (incorrectly positioned or sized due to lack of interaction with the other half of the image), or not be found at all. This paper aims to reduce the burn-in time by finding ways in which the image may be (temporarily) partitioned into smaller sub-images that may then be considered in parallel, without causing significant causing anomalies that would undermine the usefulness of the results.

III. Case Study: Finding White Blood Cell Nuclei

We demonstrate the use of MCMC to a medical imaging problem - the identification of artifacts in an image, in this case the finding of stained cell nuclei. For simplicity we abstract this into the finding of circles of high colour intensity. First 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 the circular nuclei 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 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 [8]. 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.

IV. 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 $(MC)^3$) [9], [10], where multiple MCMC chains are performed simultaneously. All but one of these chains are 'heated' so they are more likely to accept proposed moves and explore the statespace. Only the 'cold' chain is sampled, but periodically two chains are selected at random and their states swapped, subject to a modified Metropolis-Hastings test. This allows the cold chain to make the occasional large jump across the state-space to avoid or escape local optima, saving the time the chain would have been 'stuck' there. The intent of $(MC)^3$ is to improve the rate of convergence, the aim of the other methods in this paper is to distribute the original workload across multiple processors.

Another general purpose and complimentary parallelisation method is termed 'speculative moves', and detailed in [11]. Its aim is to reduce the realtime lost to considering moves that are not accepted by the transition kernel by speculatively considering multiple independent iterations simultaneously. To preserve the rules of the Markov Chain, at most one of these simultaneously considered iterations may cause a state change, but with the rejection rate of individual iterations/moves typically being around 75% this can result in significant reductions in program runtime. This is a fine-grain task parallelism, the purpose of this paper is to explore options for data-parallelisation.

V. A new parallelising approach based on periodic partitioning

Since runtime increases significantly with the complexity and size of the image, the obvious parallelisation method is to break a large image up into partitions and consider each separately. Unfortunately this will cause anomalous results along the partition boundaries as image artifacts are not detected, imperfectly detected, or duplicated (detected in both partitions). Furthermore, it is not always the case that the *prior* assumptions concerning the full image hold when applied to subset of that image. For example, taken across a set of tissue samples, cells may be distributed at random; yet if we examine only a subset of one particular image it may well be the case that not only does the distribution no longer seem random, but the density of cells (the number per unit area) may be substantially different to that of the entire image (either by chance, or by the presence of some unexpected or unpredictable entity dislodging the cells).

Despite these problems, in many cases it is possible to make use of parallelisation-by-partitioning without impairing the statistical properties of MCMC. First we separate the moves that may be applied to the MCMC chain into two groups, global ($\mathbf{M_g}$) and local ($\mathbf{M_l}$), with the probability of an arbitrary move being in $\mathbf{M_g}$ as q_g. $\mathbf{M_g}$ contains all moves that alter the configuration in a manner that impacts prior/likelihood calculations across the entire image/configuration. As such, a $\mathbf{M_g}$ move cannot be performed in parallel with any other move. $\mathbf{M_l}$ moves make limited changes (akin to fine-tuning) whose impact is restricted to a small area and makes no changes to 'global' properties (such as the number of features in the configuration). Since the decision to accept or reject such a move is based solely on the image data in close proximity to the changed feature, multiple $\mathbf{M_l}$ moves may be performed simultaneously without violating the MCMC criteria so long as the features modified by these moves are sufficiently distant.

We then modify the MCMC implementation to clump groups of $\mathbf{M_g}$ and $\mathbf{M_l}$ together. Instead of selecting a new proposed move at random from $\mathbf{M_g} \cup \mathbf{M_l}$ we alternate between performing z consecutive moves from $\mathbf{M_g}$ then $z\frac{q_g}{1-q_g}$ consecutive moves from $\mathbf{M_l}$. This leaves the long-term move proposal probabilities unaffected. Provided the number of moves performed in each $\mathbf{M_g}$ and $\mathbf{M_l}$ phase is small compared to the total number of moves performed (and the interval between post-convergence sampling) the alternating phases will not substantially alter the long-term deveoplment of the Markov Chain's state.

By definition, consecutive $\mathbf{M_l}$ moves may be performed in parallel provided that artifacts altered are sufficiently distant such that they do not affect the moves' prior and likelhood terms. To ensure this is the case, we partition the image with a uniform grid of spacing x_m along the x-axis and y_m along the y-axis. A $\mathbf{M_l}$ move may now be performed in each area partitioned by the grid simultaneously. To avoid any potential conflicts between partitions, features whose prior/likelihood calculations would draw on data from another partition may not be selected for modification. Additionally, no feature may be created or moved such that any part of it (or its prior/likelihood considered area) intersects with its partition's boundary. To avoid the partition grid imposing a long-term bias on the results, for each phase of $\mathbf{M_l}$ moves performed, a new x and y offset for the grid is chosen at random from the ranges $0..x_m$ and $0..y_m$ respectively. Assuming that the switch between phases of $\mathbf{M_g}$ and $\mathbf{M_l}$ moves occurs sufficiently frequently there will be no persistent partition boundary anomalies.

The number of iterations performed in the global and local phases must be set such that the overall move proposal probabilities are unaffected. If i MCMC iterations are to be performed in total in each local move phase, and $\mathbf{M_g}$ moves are 'supposed' to be occurring with probability q_g, then $i\frac{q_g}{1-q_g}$ iterations must be performed in the global move phase. Additionally we need to split the number of iterations to perform during the $\mathbf{M_l}$ phase between each of the partitions. If all dimensionality-modifying moves are in the set $\mathbf{M_g}$, each partition can be allocated the number of local iterations to perform in the same proportion as the number of model features contained within the partition's boundaries and that may be legitimately modified (not too close or intersecting with the partition boundary) compared to the number of such (modifiable) features taken across all partitions. If any dimensionality changing moves are in $\mathbf{M_l}$ it may be worth moving them to $\mathbf{M_g}$ anyway. Otherwise we find that certain partitions may perform more than their 'fair share' of iterations if features are not added/removed from all partitions at an equal rate.

To summarise, a number of the MCMC moves that cannot be run in parallel are performed. The image is then randomly partitioned and a number of MCMC moves than can be run in parallel are executed in each of these partitions simultaneously, with safeguards preventing changes that could potentially impact the consideration of moves in another partition. The changes to each partition are then combined back into a single model and the cycle repeats, with a number of the non-parallelisable moves being performed on the whole image. This cycle is repeated with sufficient frequency that the grouping of $\mathbf{M_g}$ and $\mathbf{M_l}$ moves and the partitioning in the $\mathbf{M_l}$ phase is not significant long-term. Whilst the stationary distribution of a specific $\mathbf{M_l}$ or $\mathbf{M_g}$ phase will be different from the target distribution, by frequent cycling it will average out such that long-term the stationary distribution will be the same as that of conventional MCMC. Depending on the application, this periodic parallelisation may have a positive or negative effect on the number of iterations until the chain converges on its stationary distribution but again, frequent cycling reduces this.

VI. Theoretical Gains

Let the mean time to perform $\mathbf{M_g}$ and $\mathbf{M_l}$ moves be τ_g and τ_l respectively. Assuming that the parallelisation overhead is negligible, the time to perform N iterations with s partitions in the $\mathbf{M_l}$ phase (each partition running on a separate thread) should be

$$N q_g \tau_g + \frac{N(1-q_g)\tau_l}{s} \tag{2}$$

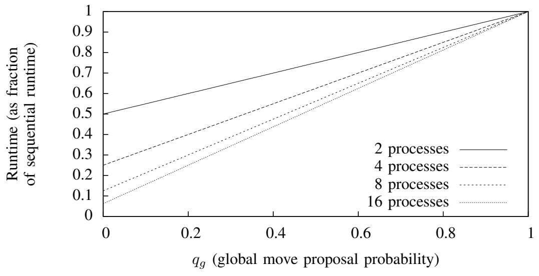

Fig. 1. Predicted results for periodic parallelisation. $\tau_g = \tau_l$

as plotted in fig. 1. The $\mathbf{M_g}$ moves, not being partition parallelisable, are the limiting factor. We can obtain further performance improvements by implementing speculative moves during the $\mathbf{M_g}$ phases. As explained in [11], if each MCMC iteration has the probability p_r of being rejected by the Metropolis-Hastings transition kernel and we assume that the overhead imposed by speculative moves is negligible compared to the processing time per iteration (a reasonable assumption on SMP computers), the use of speculative moves using n processors or processor cores will reduce the runtime to $\frac{1-p_r}{1-p_r^n}$ of its sequential runtime (under ideal conditions). Combined with eq. (2) this gives the runtime for periodic parallels plus speculative execution of the global phases as

$$N q_g \tau_g \frac{1-p_{gr}}{1-p_{gr}^s} + \frac{N(1-q_g)\tau_l}{s} \tag{3}$$

Where p_{gr} is the probability that a $\mathbf{M_g}$ move will be rejected (similarly p_{lr} is the probability a $\mathbf{M_l}$ move will be rejected).

We can also use speculative moves to further increase the number of $\mathbf{M_l}$ moves that may be performed per unit time. Though it appears preferable to utilise any spare threads/processors to allow a greater number of partitions to be made and processed simultaneously, there is a limit as to how small a partition can be. Since we prohibit any changes to features intersecting or very near to a partition boundary (to avoid potential conflicts with the changing features in neighbouring partitions) the area in which features may be changed is somewhat smaller than the area of the partition. For instance, for a square partition of length x, populated with features whose impact on the likelihood/prior terms when subject to $\mathbf{M_l}$ moves is restricted to an area y across, the actual area available for features to be moved/manipulated is only $(x-y)^2$. We may therefore choose to use speculative moves during the $\mathbf{M_l}$ phase if we have spare processors and do not wish to shrink the partition sizes any further. The use of speculative moves during the $\mathbf{M_l}$ phase may also be encouraged by system architecture. We note that if $\mathbf{M_l}$ phases are set to be long enough it becomes feasible to deploy each partition to a separate physical machine in a cluster. If this is the case, and each machine in the cluster also has true multithreading capabilities, it is natural to use separate

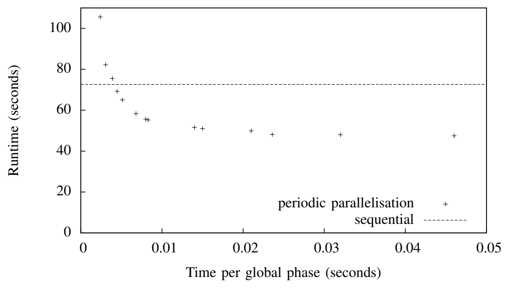

Fig. 2. Example of periodic parallelisation on 1024x1024 images with only four partitions, run on a Q6600. The horizontal line represents the runtime of the sequential implementation.

physical machines for each partition, and multithreading in each machine for speculative moves. For a cluster with s machines each with t threads, the best possible runtime (assuming negligible overhead) is

$$Nq_g\tau_g\frac{1-p_{gr}}{1-p_{gr}^t}+\frac{N(1-q_g)\tau_l(1-p_{lr})}{s(1-p_{lr}^t)} \tag{4}$$

In practise the frequency with which we alternate between $\mathbf{M_g}$ and $\mathbf{M_l}$ phases will also have an impact on the total runtime (recall the above predictions assume negligible overhead). Statistically we want these phases to be as short as possible to minimise any potential impact the partitioning may have to the short-term results. Practically we want each phase to be long enough to overshadow the overhead required in partitioning, distributing the workload to the parallel threads/machines, and the subsequent recombining the models. A similar balance must be made between the number of partitions (more=faster) and the corresponding size of the partitions. More partitions mean each partition is smaller, which means the number of features that may be modified (and how those features may be modified) is more limited, thus is more likely to delay the convergence of the MCMC algorithm.

Since different partitions will be allocated different numbers of iterations to perform (depending on the number of model features fully enclosed within each partition), the time taken to complete the assigned iterations will vary considerably (even if features are uniformly distributed, partitions along the edge of the image will inevitably be less than their full size, contain fewer than normal features and thus be allocated fewer iterations to perform per local move phase). The processor dead-time that results can be reclaimed through the use of a task scheduler, allowing more partitions than there are available processors to be employed.

VII. Periodic partitioning results

Let us consider processing a 1024x1024 image containing 150 cells of mean radius 10. Since the expected number of cells is part of the prior term, any move that changes the number of cells in the model must be a global move. Therefore $\mathbf{M_g} = \{$add, delete, merge, split, replace$\}$ and $\mathbf{M_l} = \{$alter position, alter radius$\}$. The proposal probabilities are such that 60% of moves are from $\mathbf{M_l}$. The image will be split into four rectangular partitions using a single coordinate where all partitions meet (each square in the partition grid is larger than the image). Whilst suboptimal in terms of processor utilisation when 4 processors are available (partitions will rarely be of equal size) this does minimise overhead from splitting and merging configurations by keeping such operations simple.

Using these parameters, fig. 2 shows the time taken to perform a fixed number (500 000) of MCMC iterations for different frequencies of repartitioning on a quad-core Q6600, the horizontal line representing the runtime of the sequential implementation. In this case each global move phase must last at least 4ms ($\sim$ 23 iterations) for the periodic parallelisation method to be faster than the sequential implementation. There is no substantial runtime improvement from spending more than around 20ms in each global phase. This equates to around 130 iterations, and thus each local phase will perform 194 iterations spread amongst all the partitions, taking around 14ms). From this data, spending 20ms per global phase is the ‘sweet spot’. More frequent cycling between phases substantially impairs runtime, whilst less frequent cycling brings minimal runtime benefits and increases the risk of the alternative global/local phases distorting the development of the Markov Chain.

With the 20ms global phase, the apparent runtime on the Q6600 has been reduced by $\sim 29\%$ of the sequential implementation. Repeating this program on a dual-processor Intel Xeon achieved a runtime reduction of 23%, whilst a dual-core Pentium-D managed a 38% reduction. The differences in performance between these machines is due to the difference between the time per iteration and the overhead required to duplicate, arrange for parallel execution, and merge the partitions. The Pentium-D can be expected to have the best inter-thread communication times as it is a dual-core machine and can execute two threads on the same physical die, whilst the dual-processor Xeon machine has greater communication times between threads as they must execute on different physical processors. The Q6600 machine, with two dual-core processors, falls in-between the Xeon and Pentium-D.

The Q6600 falls short of the 45% reduction as predicted by eq. (2) when $q_g = 0.4$, $\tau_g = \tau_l$ (as is the case when processing is strictly sequential) and $s = 4$. However, recall that by restricting the number of partitions to 4 but permitting (requiring) those partitions to be of varying sizes, the size of the largest partition will always be greater than a quarter of the image and potentially range to the size of the image itself. Consequentially the four processors will never be fully utilised, indeed comparable results were obtained using only the two processors on the Xeon and Pentium-D. In these cases, one processor will take the largest partition, the remaining three small partitions being performed on the second processor.

More substantial reductions in runtime more in line with predictions could be obtained by using a finer partitioning grid and load balancing if (as in this case) the number of partitions is greater than the number of available processors. The runtime of the local phase would then tend towards 1/(number of partitions) of the sequential runtime, if the overhead from communication and configuration split/merge operations remained negligible compared to the time spent in each phase.

VIII. Image Partitioning

There are some applications where MCMC is a convenient means of structuring a solution to some problem, but the statistical robustness offered by MCMC is not required - obtaining an 'reasonable' answer promptly is often more important than waiting for a statistically pure result (for instance when the program is used to flag samples for human review, such as in high throughput screening). Experience and testing determines whether the tradeoff is acceptable. Periodic parallelisation takes the divide and conquer as far as it can inside the Markov Chain framework, but more dramatic runtime reductions can be obtained by partitioning the image, applying MCMC to the partitions individually, then intelligently combining the results. The MCMC processing of each of the partitions can proceed much faster than the processing of the combined image as a) there are fewer artifacts to match and b) those artifacts present have a much smaller range of states (positions) they may occupy. To be clear, unlike the periodic parallelisation and speculative moves this method is *not* statistically equivalent to conventional MCMC and so is not guaranteed to eventually converge on the target solution. Though it depends on the application, this may produce 'reasonable' solutions albeit with the potential for anomalies and biases for certain type of configurations. The advantage is that the results that are produced can be obtained in a much shorter space of time as all processing in each partition is performed independantly, unlike periodic parallelisation with its $\mathbf{M_g}$ phases. Of course, the use of partitioning and the recombination heuristic make it impossible to obtain the differing model alternatives along with their relative probabilities.

There are two issues with partitioning an image outside the MCMC mechanism, whilst keeping legitimate MCMC inside each partition. The first is the allocation of prior knowledge to the separate positions, properties held to be true over the single image may not apply to the partitions (expected number of artifacts being the obvious example). The second is coping with artifacts that span the partition boundaries. Whilst the later is a common problem to all segmentation algorithms employing image partitioning for parallelisation e.g. [12], the former is unique to Bayesian inference-based algorithms, primarily MCMC.

The allocation of prior knowledge concerning the partitions is application specific, and plays a large role in determining the viability of image partitioning as a tactic. For example, consider the property of the expected artifact density of an image (how many artifacts are expected to be contained in a given area). Prior knowledge might provide the total number of artifacts that may be in an image, but say nothing of their distribution inside that image. If the artifact density is incorrectly assumed to be constant throughout an image then the partitions will all inherit this value thus assigning potentially inaccurate prior information to some partitions. The degree to which this matters depends on how the likelihood

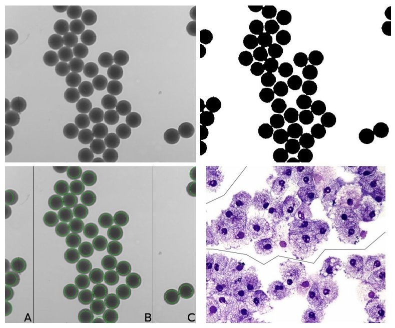

Fig. 3. Intelligent Partitioning in action.

and prior terms are balanced and whether enough iterations will be performed on the most artifact-dense partition to allow full convergence.

Ideally the estimate for the properties like the artifact density should be mechanically generated based on the actual image data, in which case the same mechanism used to obtain the estimate for the complete image should be applied to the partitions. A good estimate for artifact density in cell sample images (such as in fig. 3) can be obtained by applying a threshold filter and counting how many pixels are of high intensity. Assuming all pixels passing the threshold criteria belong to a cell nucleus we can estimate the number of cells that are present to be

$$\frac{|\{\forall(x,y) \in M : I(x,y) > \rho\}|}{\pi r_\mu^2} \quad (5)$$

where M is the set of all pixels in the image or subimage, $I(x,y)$ is the intensity of pixel (x,y) and ρ is some suitable threshold value. Note that we may use r_μ in this expression as unlike the estimated circle density the expected circle radii can be assumed constant throughout all partitions (for these images at least).

The second issue to address is that of the artifacts that span partition boundaries. The first tactic we consider for addressing these we term *intelligent partitioning*. If the target artifacts are sufficiently disperse and identifiable, a comparatively fast pre-processor may be applied to crop and segment the image such that artifacts do not intersect the subimage boundaries, thus sidestepping the issue. Note that artifacts must be far enough away from the subimage boundaries to avoid double-dipping - their presence influencing the results of more than one subimage. This method devolves some of the operations of the larger MCMC algorithm to the pre-processor, using the (presumably) faster algorithm to reduce the statespace of the Markov Chain needing to be explored. Complete confidence in the reliability of the pre-processor is required, thus restricting its use to situations where the presence, or more importantly the *absence* of artifacts can be ascertained.

The degree of parallelisation (and hence performance improvement) of this method is not under the user's control, as partition boundaries are set according to the arranagement of the data. Inconvenient datasets may not offer any substantial parallelisation, consequently the program's runtime for any given image is also unpredictable.

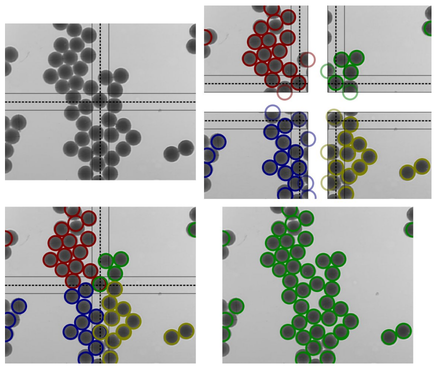
Fig. 4. Blind Partitioning in action.

The second method for coping with partition-boundary-spanning artifacts is *blind partitioning*. Here we do away with the partitioning pre-processor and simply partition the image in some arbitrary manner, such as a simple grid. When combining the results for each partition we employ some heuristics to procedurally 'patch up' any anomalies resulting from the partitioning. To do this in a systematic fashion whilst making the most of the per-partition MCMC processing, we propose there be overlap between each partition such that the largest expected artifact will fit inside (i.e. each partition will extend r_{MAX} further than normal in each direction), as in fig. 4 (top left). Every artifact will then have the oppertunity to be fully examined by MCMC processing. When merging the resultant configurations, artifacts with their centerpoint in the non-overlapping regions are automatically accepted, whereas artifacts with centres in a overlapping region will need comparing with nearby features from the other partition(s) - fig. 4 (top right). If the MCMC algorithm applied to the partitions yielded good results, such artifacts should appear in both partitions with minimal differences and so can be merged with little difficulty - fig. 4 (bottom left). Features without a counterpart from the other partitions are disputable, you may wish to accept or discard them depending on whether it is more important to avoid false-positives or not missing potential artifacts. Either way, blind partitioning does not rely on a separate potentially complex partitioning algorithm, thus places fewer preconditions on what datasets it can process. The tradeoff is the reduction in confidence in the final result, unlike intelligent partitioning and periodic partitioning the statistical assurances accompanying MCMC cannot be applied to the final model produced for the image. Anomalies may persist along the partition boundaries if the MCMC processing in the two partitions failed to construct a common interpretation for any shared artifacts.

IX. Intelligent and Blind Image Partitioning Results

Figure 3 (top-left) shows a number of latex beads[1] in a petri dish. Due to the clumping of the beads and the ease by which the beads may be distinguished from their surroundings this makes a good candidate for the application of intelligent partitioning. We apply a threshold filter as in eq. (5) where $\rho = 0.5$ and pixel intensity values in the range $0..1$ to identify the likely artifacts (fig. 3, top-right), and then partition the image by scanning for rows or columns that are completely empty. The partitions are made on columns/rows equidistant between the closest columns/rows containing pixels(s) that passed the threshold criteria (fig. 3 bottom-left). There are 48 artifacts in the image. Were this number only provided as part of the *prior* knowledge, we might be forced to assume the distribution of artifacts was uniform, thus allocating an expected artifact count of 7 to partition A, 30 to partition B and 11 to the partition C. Fortunately this image contains no artifacts that can be confused with non-target image entities when using the threshold filter, so we may calculate an estimate for the number of artifacts (λ) based on eq. (5). The results using this information are in table I: If at least 3 processors are available the intelligent-partitioning program runtime is the longest time taken to process any of the partitions (in this case 0.97 seconds, a reduction in runtime of 10%), as combining the results for the three separate partitions is trivial. With only two processors load balancing should be used, which for this example gives the same runtime of 0.97 (as $0.07 + 0.02 < 0.97$).

[1]latex beads are used for clarity in this illustration, the MCMC program at work is the same used for finding stained cell nuclei.

TABLE I
Results of intelligent partitioning on fig. 3. Timing values are taken from runs on a Q6600 processor. Number of iterations and runtime required averaged over 20 runs.

		A	B	C
Area (pixels^2)	2.13×10^5	3.14×10^4	1.33×10^5	4.82×10^4
Relative area	1	0.147	0.624	0.226
# obj. (visual)	48	6	38	4
# obj. (density)	–	7.08	29.97	10.86
# obj. (thresh.)	46	4.9	38	3.1
$\approx$ time/iteration	4×10^{-5}	1.9×10^{-5}	4.3×10^{-5}	2.0×10^{-5}
# itr to converge	27 000	4 000	22 500	900
Runtime (secs)	1.08	0.08	0.97	0.02
Relative runtime	1	0.07	0.90	0.02

An irregular partitioning as in fig. 3 (bottom right) imposes little additional overhead on the MCMC algorithm once the partition lines have been drawn. The likelihood and prior calculations will be oblivious to the partitioning as the pixel data for neighbouring partitions will be blanked out (this is safe to do as the validity of intelligently chosen partitions depends upon the presumption that the contents of neighbouring partitions are irrelevant to the consideration of the current partition). Since there will be no pixel data for beyond the partition boundary, features will not be placed there by the MCMC algorithm. Should additional checks be necessary to keep all artifacts within the partition bounds, they will take place when changes to the model are proposed, before the prior and likelihood calculation (that dominate runtime) take place. The only difficulty will be the creation of such irregular partition boundaries and the time this takes, though since detecting where features definitely do not exist is easier than identifying with certainty the position and properties of features, a range of comparatively fast segmentation algorithms (some of which may themselves be based upon MCMC) will generally be available.

To apply blind partitioning to the same example, the image is first split into four equal sized areas as shown by the dotted lines in fig. 4, top left. These areas are then expanded to the solid lines to fully enclose any artifact whose centre was in the original area. In this case we have extended each partition boundary edge by 1.1 times the expected artifact radius, easily encompassing such features as there is very little variation in the radii of the latex beads. After determining the expected number of beads in each partition using eq. (5) the MCMC algorithm was applied to give the results in the top right of fig. 4. Beads not completely enclosed in the partition ($\approx$ beads whose centre is not inside the dotted line marking the simple quartering of the image) are deleted from each partition's model. The union of the partition's models is then taken and any beads centred in the overlap area that are in close proximity (centerpoints within say 5 pixels of each other) are merged (replaced with a bead with centerpoint and radii that are the average of the original bead).

After performing a comparison similar to that of table I, the relative runtimes of the top-left, top-right, bottom-left, and bottom-right corners were 0.12, 0.08, 0.27, and 0.11 respectively. The runtime of the whole procedure (if four processors are available) is $\approx$ to the longest time taken to process a partition as the merging of the partition models takes negligible time compared to thousands of MCMC iterations. In the example the runtime was reduced to 27% of the original, with no apparent anomalies present as a result of the partitioning. For this test image this is clearly superior to the intelligent partitioning result of reducing the runtime to only 90% of the original.

X. Conclusion

We have presented two approaches for applying the divide and conquer technique of dividing image data to Markov Chain Monte Carlo simulations. Periodic parallelisation conserves the statistical model of MCMC, whilst image partitioning (be it blind or intelligent) potentially allows for 'reasonable' (though not statistically pure) results to be obtained much faster, although the magnitude of the speedup is highly data-dependant.

Periodic parallelisation runtimes can be approximately predicted using eq. (4) (and has been demonstrated to reduce runtime on the test data by 38% using a dual core machine). The image partitioning runtime cannot be reasonably predicted as it is highly data-dependant, however under ideal conditions (artifacts evenly distributed amongst partitions, partitions of equal size, and prior values correctly assigned) image partitioning can be expected to provide speedups exceeding $(1 - \frac{1}{n})\%$ as both the expected number of artifacts and the image area over which they can range will have been reduced by this ratio compared to the original image. As demonstrated by the results (reduction of 5% with intelliegent- and 73% for blind- partitioning) a suitable dataset and the correct choice of method are essential for obtaining desired benefits. Although no anomalies were present in the tests, image partitioning does not have the full statistical reliability and usefullness of MCMC, and places preconditions on what datasets and application it can be applied to.

Whilst periodic parallelisation does scale with the image size (by allowing more partitions in $\mathbf{M_l}$ phases), this is limited by the inability to parallelise $\mathbf{M_g}$ phases other than by the speculative moves method covered in [11]. In contrast the image partitioning methods are more suited for scaling to large images as their partitions are considered independantly of each other. Intelligent partitioning requires having the initial cost of the preprocesor (scalability determined by the algorithm used for this), whereas blind partitioning requires only a post-processor that scales linearly with the number of partitions to resolve artifacts in the overlapping areas of partitions. Both intelligent and blind partitioning require a preprocessor to assign suitable values for each partition's 'prior knowledge'.

Assuming multicore and/or multiprocessor hardware is available, the methods presented in this paper can reduce the impact that image size/complexity has on the time required for processing to a set standard. When combined with more traditional means of reducing MCMC runtime, this makes viable the application of Markov Chain Monte Carlo methods (thus Bayesian inference) to the processing of large/high-resolution biomedical images, and can increase the throughput of existing applications.

References

[1] D. J. Wilkinson, "Bayesian methods in bioinformatics and computational systems biology," *Brief Bioinform*, vol. 8, no. 2, pp. 109–116, 2007.

[2] J. P. Huelsenbeck and F. Ronquist, "MrBayes: A program for the Bayesian inference of phylogeny," Department of Biology, University of Rochester, Tech. Rep., 2003.

[3] I. Dryden, R. Farnoosh, and C. Taylor, "Image segmentation using Voronoi polygons and MCMC, with application to muscle fibre images," *Journal of Applied Statistics*, vol. 33, no. 6, 2006.

[4] D. C. K. Fan, "Bayesian inference of vascular structure from retinal images," Ph.D. dissertation, University of Warwick, May 2006.

[5] E. Thonnes, A. H. Bhalerao, W. Kendall, and R. Wilson, "A Bayesian approach to inferring vascular tree structure from 2D imagery," in *International Conference on Image Processing*, vol. 2, 2002, pp. 937–940.

[6] A. C. Fan, J. W. Fisher, W. M. Wells, J. J. Levitt, and A. S. Willsky, "MCMC curve sampling for image segmentation," in *MICCAI 2007*, 2007.

[7] P. J. Green, *Practical Markov Chain Monte Carlo*. Chapman and Hall, 1994.

[8] "Reversible jump markov chain monte carlo computation and bayesian model determination," *Biometrika*, vol. 82, pp. 711–732, 1995.

[9] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist, "Parallel Metropolis-Coupled Markov chain Monte Carlo for Bayesian Phylogenetic Inference," Department of Computer Science, University of Rochester, Tech. Rep. 784, July 2002.

[10] M. Harkness and P. Green, "Parallel chains, delayed rejection and reversible jump MCMC for object recognition," in *British Machine Vision Conference*, 2000.

[11] J. M. R. Byrd, S. A. Jarvis, and A. H. Bhalerao, "Reducing the run-time of MCMC programs by multithreading on SMP architectures," in *IEEE International Symposium on Parallel and Distributed Systems (IPDPS)*, 2008.

[12] J. Wassenberg, W. Middelmann, and P. Sanders, "An efficient parallel algorithm for graph-based image segmentation," in *CAIP '09: Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1003–1010.