Parallel birth and death process for cell nuclei extraction in histopathology images

Christophe Avenel, Pierre Fortin, Dominique Béréziat

Parallel birth and death process for cell nuclei extraction in histopathology images

Christophe Avenel, Pierre Fortin, Dominique Béréziat
UPMC Univ Paris 06 and CNRS UMR 7606, LIP6,
4 place Jussieu, F-75252, Paris cedex 05, France
Email: christophe.avenel@lip6.fr

July 12, 2013

Abstract

Cell nuclei extraction from histopathology images is necessary for breast cancer grading, and has become one of the major problem in the domain of automatic image analysis. Stochastic marked point processes combined with birth and death processes are promising tools for such extraction, but they are extremely compute intensive, especially on large images such as scanned microscope slides. We here show that the original birth and death process applied to marked point processes is inherently sequential. We thus rewrite this algorithm in order to obtain a highly parallel birth and death process. This algorithm is finally efficiently deployed on multi-core and many-core architectures, and the corresponding performance results are presented and analyzed.

1 Introduction

The Nottingham Grading System [1] used for breast cancer grading in histopathology – the study of diseased tissues at microscopic level – is strongly based on the size and aspect of nuclei. The detection and extraction of nuclei are thus important issues in the domain of automatic image analysis of histopathology images. Moreover, the reduction of the computation time is becoming critical, because of the huge microscope slide sizes (up to $100,000$ by $100,000$ pixels at full resolution).

Extraction algorithms can be divided into two main categories: classification and segmentation. The classification ones, see [2] for instance, will not be able to separate and count nuclei. The segmentation can be performed using active contour model (Snakes) [3], but as one snake would be needed for every nucleus, this solution is not relevant here. The segmentation can also be performed using the level set approach using free shapes [4, 5], or parametric ones [6]. A parametric shape based model decreases the amount of computation, and is sufficient for a good cell nuclei detection here.

In breast cancer images, the nuclei often appear joint or even overlapped. We thus consider in this paper a marked point process, which enables us to extract nuclei as individual joint or overlapping objects without necessarily discarding overlapping parts and therefore without major loss in delineation precision [7]. Various authors applied point processes to image analysis [8, 9]. We use here a simulated annealing algorithm combined with a "birth and death" process as described in [10].

This method could also easily be extended to other sort of cancers, such as the renal cancer, whose gradation is mainly based on the cytonuclear atypia, *i.e.* the variation in size and shape of cell nuclei.

This process is extremely compute intensive, espe-

cially on large images: its parallelization is therefore crucial, as well as its good scaling on the number of nuclei, hence on the image size. Moreover, with the increasing number of cores on all parallel architectures (multi-core CPUs, GPUs, Intel Xeon Phi, *etc.*), it is necessary to rely on parallel algorithms that scale with the number of cores. Several images can of course be processed in parallel on multiple compute nodes: we focus here on the parallel processing of a given image targeting in particular fine-grained parallelism for many-core architectures. To our knowledge, such birth and death process has however not been deployed on parallel architectures yet.

In Section 2, we thus present the original birth and death process applied to histology images and show that such algorithm is inherently sequential. We then detail in Section 3 how we have revised this algorithm in order to obtain a parallel birth and death algorithm that scales on the number of cores and on the number of nuclei. Performance results are presented in Section 4 on both multi-core CPUs and GPU architectures. Finally, in Section 5 we present concluding remarks and discuss future work.

2 Birth and death process for cell nuclei extraction in histopathology images

In this paper, we consider the framework of histopathology images, more precisely using hematoxylin and eosin (H&E) stained images (see Fig. 1). But this work could easily be generalized for the detection of any elliptically shaped object. The goal here is to perform the cell nuclei detection for the purpose of breast cancer gradation. According to the Nottingham grading system, the cell nuclei sizes are one of the major criteria for breast cancer grading. Small cell nuclei of almost same sizes will denote a small grade, whereas a marked size variation will conduct to a higher grade. Fig. 1 presents H&E images which have been classified as grades 1 to 3 by experts.

2.1 Marked Point Process framework: mathematical background

A Marked Point Process (MPP) is used in order to detect an arbitrary number of objects. MPP is a stochastic process where a realization $\mathbf{w}$ is a set of marked points $w_1, ..., w_n$. We denote $\mathbf{W}$ the set of all possible realizations of $\mathbf{w}$. A marked point w_i is an object described by its position x_i, and its mark m_i. A mark either represents a complex shape, or a simple parametric one like a circle or an ellipse. As cell nuclei may be correctly approximated by ellipses (see [11] for a justification), an object w_i will be described by the center of an ellipse x_i, with small and big axes $a_i, b_i \in [r_{min}, r_{max}]$, and with an orientation $\theta_i \in [0, 2\pi]$. The mark of an object is then defined by $m_i = (a_i, b_i, \theta_i)$.

We suppose that the distribution of w is ruled by a Gibbs field, *i.e.* a realization $\mathbf{w}$ has the probability:

$$P(\mathbf{w}) = \frac{1}{Z_\beta} \exp\left(-\beta U(\mathbf{w})\right) \quad , \tag{1}$$

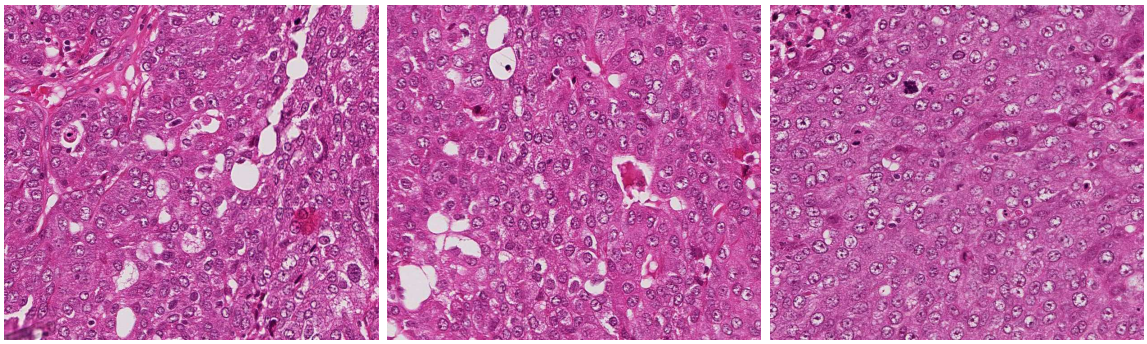
where U is the energy of the Gibbs field, β is a real and positive parameter and $Z_\beta = \sum_{\mathbf{w}\in\mathbf{W}} \exp\left(-\beta U(\mathbf{w})\right)$ is a normalization constant. The problem is then to determine the realization $\mathbf{w}$ having the highest probability.

The function $U(\mathbf{w})$ is defined as:

$$U(\mathbf{w}) = \gamma_d \sum_i U_d(w_i) + \gamma_p \sum_{i\neq j} U_p(w_i, w_j) \quad , \tag{2}$$

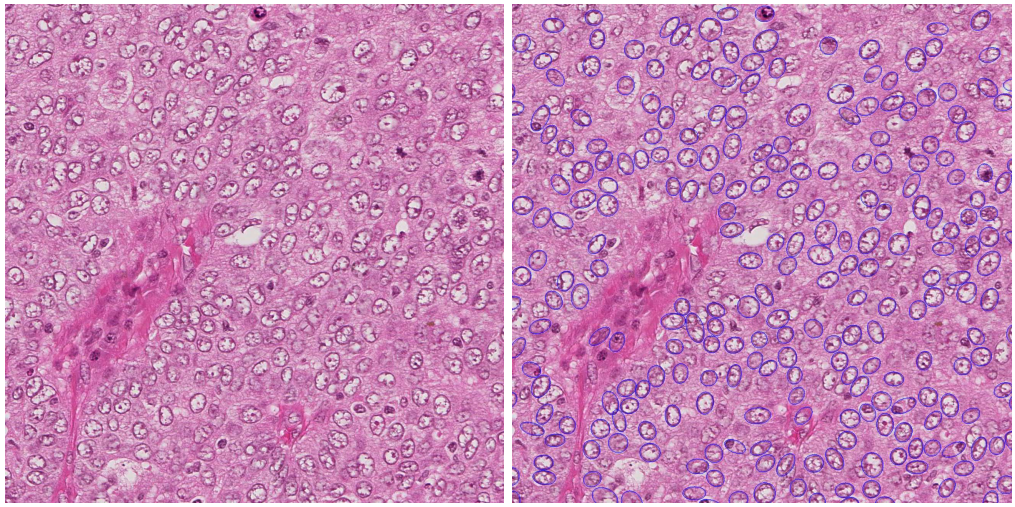
where U_d is the data-fidelity term, which measures the relevance between $\mathbf{w}$ and the image, and U_p is the interaction term measuring the coherence of $\mathbf{w}$. The parameters γ_d and γ_p are weighting coefficients. The goal of the data-fidelity term is to evaluate the relevance of the objects in the image: for instance an object w_i correctly placed should give a low value for U_d. The interaction term must be defined in order to avoid the superposition of the objects w_i. These terms are discussed in the next subsection.

To compute the optimal realization $\mathbf{w}$, we use a simulated annealing algorithm combined to a "birth and death" process: the birth step consists in generating a large number of objects w_i, each one be-



(a) Grade 1 (b) Grade 2 (c) Grade 3

Figure 1: The three grades of the Nottingham system.



(a) (b)

Figure 2: Example of a good quality detection on a H&E image

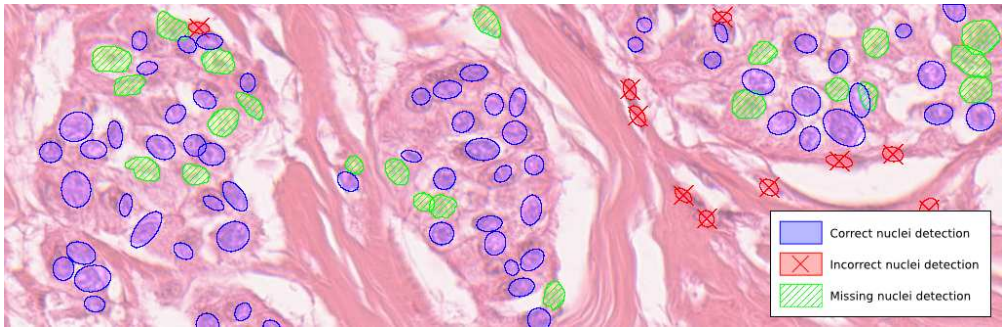


Figure 3: Example of detection and comparison to a ground truth

ing one realization of a Poisson distribution of intensity z; the death step examines each object w_i: the object is isolated from $\mathbf{w}$ and we compute the new energy $U(\mathbf{w} \setminus w_i)$ with $\mathbf{w} \setminus w_i \equiv \{w_j, \forall j \neq i\}$. If the energy increases, the object is effectively removed from $\mathbf{w}$ otherwise the object is kept with probability $\frac{\delta \triangle P_i}{1+\delta \triangle P_i}$ with $\triangle P_i = P(\mathbf{w} \setminus w_i)/P(\mathbf{w}) = \exp\left(-\beta(U(\mathbf{w}) - U(\mathbf{w} \setminus w_i))\right)$ and δ a positive parameter. The birth and death process is iterated until convergence. To expect a fast convergence, the parameter δ should decrease whereas the parameter β should increase.

The convergence of such algorithm has been proved in [10].

2.2 Elliptically shaped objects

In order to detect and extract correctly the cell nuclei, we need to use a robust data-fidelity term. This term depends on the position and mark of the ellipses. Following [11], we compute the Bhattacharyya distance between $F_{\text{in}}(w_i)$ and $F_{\text{out}}(w_i)$ which are respectively the inside and outside borders of the ellipse w_i:

$$d_B(F_{\text{in}}, F_{\text{out}}) = \frac{(\mu_{\text{in}} - \mu_{\text{out}})^2}{4\sqrt{\sigma_{\text{in}}^2 + \sigma_{\text{out}}^2}} - \frac{1}{2}\log\frac{2\sigma_{\text{in}}\sigma_{\text{out}}}{\sigma_{\text{in}}^2 + \sigma_{\text{out}}^2} \quad (3)$$

where μ_{in}, σ_{in}, μ_{out}, σ_{out} are respectively the mean and standard deviation of borders $F_{\text{in}}(w_i)$, $F_{\text{out}}(w_i)$. It is shown in [11] that the insider border of a cell nucleus is darker than the outside one. The data-fidelity term is then chosen as $U_d(w_i) = Q_d\left(d_B(F_{\text{in}}(w_i), F_{\text{out}}(w_i))\right)$, with $Q_d(d_B) \in [-1, 1]$ a quality function which gives a negative value for a high Bhattacharyya distance (*i.e.* for well placed objects), and a positive value otherwise.

The prior energy U_p controls the overlapping of objects by measuring the intersection between all ellipses:

$$U_p(\mathbf{w}) = \sum_{w_i \in \mathbf{w}} \left(\bigcup_{w_j \in \mathbf{w} \setminus w_i} w_j \right) \cap w_i. \quad (4)$$

This term increases when the percentage of overlapped surface grows.

More details on the mathematical background and the elliptically shaped objects can be read in [11].

2.3 Overview of the original sequential birth and death algorithm

Algorithm

1. *Initialization:* give suitable values for parameters β, z, γ_d, γ_p, and δ. Set $\mathbf{w}$ to an empty set.
2. *Birth:* sample a realization of objects following a Poisson distribution with intensity $\delta \times z$ and add them to the current realization $\mathbf{w}$.
3. *Data-fidelity term computation:* for each object of the realization, U_d is computed.
4. *Overlap map computation:* in order to compute U_p in the next step, we first build a map giving the number of ellipses overlapping each pixel. For each ellipse, we have to visit the pixels inside this ellipse and update the map.
5. *Death:* sort the objects w_i in decreasing order of their data-fidelity term. This sorting step ensures the best ellipses (i.e. the ones having the lowest U_d values) are processed as last, with thus lowest U_p values. Then for each sorted object w_i: compute the overlapping energy U_p. This term is directly given by the sum of elements of the overlap map over the ellipse. The object is then removed with probability $1 - \frac{\delta \triangle P_i}{1+\delta \triangle P_i}$, and in this case, its contribution is removed from the overlap map.
6. *Stop* if all the objects added in the birth step and only them are removed in the death step; otherwise, update δ and β such as: $\beta \leftarrow \beta \times c_\beta$, $\delta \leftarrow \frac{\delta}{c_\delta}$, with $c_\beta > 1$ and $c_\delta > 1$, and go to 2).

2.4 Quality measurement

The validity of our method can be verified by comparing the results with a ground truth. The ground truth has been obtained from a manual segmentation of the cell nuclei, under the supervision of histopathologist experts.

Given the ground truth, we can deduce the number of nuclei correctly or incorrectly detected, and the number of nuclei missed by the algorithm. In

order to match the manually segmented nuclei with our automatically segmented ones, we have used the Munkres' Assignment Algorithm [12] based on the number of common pixels between two nuclei. For simplicity, we will call a correctly detected nucleus a "true positive", an incorrectly detected one a "false positive", and a missing one a "false negative". Fig. 3 presents an example of detection, showing true positives, false positives and false negatives.

The quality of the cell nuclei detection is evaluated using the *F-measure* widely used in the pattern recognition community:

$$F\text{-}measure = \frac{2\mathrm{TP}}{(2\mathrm{TP} + \mathrm{FN} + \mathrm{FP})} \ ,$$

where TP, FP and FN are respectively the number of true positives, false positives and false negatives.

An example of a good quality result is shown in Fig. 2.

2.5 Inherent sequentiality of the death step

As seen in Section 2.3, the birth and death algorithm is principally composed of four steps: a birth step, a data-fidelity term computation, an overlap map computation and a death step. The three first steps can be easily computed in parallel (with however some synchronizations required among the threads: see Section 3.2). The death step is however inherently sequential: according to the original birth and death algorithm, the ellipses have to be treated in the decreasing order of their data-fidelity term.

One could first consider not to parallelize the death step. However, this would have a strong negative impact on the overall performance of the birth and death process. Indeed, given t_b, t_f, t_o and t_d respectively the birth step computation time, the data-fidelity term computation time, the overlap map computation time and the death step computation time, and assuming we keep a sequential death step, the theoretical maximum speedup, given by Amdahl's law, is:

$$R = \frac{1}{(1-s) + \frac{s}{N}},$$

with $s = \frac{t_b+t_f+t_o}{t_b+t_f+t_o+t_d}$ the proportion of the execution time that can be made parallel, and N the number of processors used. We have measured the performance of a serial CPU implementation of the original birth and death algorithm. For one iteration, we have the following mean values: $t_b = 10.0$ms, $t_f = 126.5$ms, $t_o = 42.9$ms, $t_d = 45.0$ms, which gives $s = \frac{t_b+t_f+t_o}{t_b+t_f+t_o+t_d} = 0.8$. The theoretical maximum speedup (with $N = \infty$) is then $R = 5.0$. The death step thus has clearly to be parallelized in order to reach high enough speedups on many-core architectures.

2.6 Attempt to compute the original death step in parallel

Considering the original birth and death algorithm, one could then try to parallelize the death step by browsing the sorted list of ellipses with a fixed number of threads N_T. Starting with the ellipses with the worst data-fidelity term, each thread proceeds with the death computation for one ellipse at a time until all ellipses have been treated. Doing so, an ellipse can be treated concurrently to (or even before) another ellipse with a worse data-fidelity. The ordering of the ellipse is therefore only partially respected, in contrary to the sequential execution for which the ordering was strictly respected. This can decrease the quality of the results since, in parallel, an ellipse can thus be deleted due to overlapping ellipses with worse data-fidelity whose death step is not yet completed. However, as the number of ellipses is quite large (for example, around $20,000$ on a 1024×1024 image or $320,000$ on a 4096×4096 image), one could aim to have this way high enough parallel speedups with reasonably good result quality. Therefore we will now simulate such parallel executions on multi-core and many-core architectures (with up to thousands of active threads) and study the impact of the parallelism degree (the number of threads used) on the quality of the results. We use here the OpenMP multi-threading on 12 CPU cores with 2-way SMT (Simultaneous multi-threading, hence up to 24 hardware threads), but similar results were obtained on a dual-core CPU. The browsing of the ellipse sorted list

is here parallelized in OpenMP with either a static or a dynamic scheduling. Figure 4 presents the corresponding F-measure value depending on the number of threads for various OpenMP parallelization strategies: the quality of the results quickly drops to unacceptable values as the degree of parallelism increases. We now deeply analyze this issue thanks to the different OpenMP parallelization strategies presented.

We first consider a static OpenMP scheduling (with chunk of size 1, *i.e.* a 1D cyclic distribution of the ellipses among the threads) and a synchronization (OpenMP barrier) between all threads after each ellipse computation: this enables to simulate a true parallel execution with one core per thread. With such synchronizations, the worst quality measurements are obtained by treating each block of N_T ellipses sequentially in reverse ordering. In each block of N_T ellipses, we indeed start by the ellipses with the best data-fidelity instead of the ones with the worst data-fidelity as in the original sequential algorithm. This corresponds thus to the worst case parallel execution with barriers. Performing the parallel execution with a static OpenMP scheduling (*Static, with barrier*) gives in practice results similar to this worst case execution strategy (*Reverse ordering by block*), and does not scale beyond 128 threads.

If we remove the synchronizations, concurrent executions can now proceed in parallel between ellipses belonging to different blocks of N_T ellipses. The gap between the data-fidelity terms of the ellipses concurrently treated thus increases which strongly degrades the quality of results (*Static, without barrier*). A standard GPU parallelization of the death step with thousands of active threads would treat the sorted list of ellipses according to a cyclic distribution of the ellipses among the threads. These static schedulings show that such GPU deployment is doomed to give results with an unacceptable quality.

As far as multi-core CPU parallelization is concerned, we would rather rely on a dynamic scheduling because of the varying computation load per ellipse. The data-fidelity gap is here reduced compared to the *Static, without barrier* strategy, since the dynamic load balancing leads to a distribution of the ellipses among the threads that follows the list ordering. The quality of the results for this strategy (*Dynamic, without barrier*) drops for more than 16 threads. But a realistic multi-core CPU parallelization would require a coarse enough computation grain, with for example a chunk of size 32 (each thread treating 32 consecutive ellipses at a time). This again increases the data-fidelity gap among ellipses concurrently computed and the corresponding quality drops with more than 4 threads. This shows that a multi-core CPU parallelization of the original algorithm is unlikely to scale beyond 4 threads.

Furthermore, we can see in Fig. 5 that the reduction of quality comes along an increase of the number of iterations, and thus of the overall computation time. Indeed, the quality drop implies that we keep fewer good ellipses at each iteration, which increases the total number of iterations required to meet the convergence criterion. The iteration number drops again when the quality is too deteriorated, and the number of kept ellipses then tends to zero.

The PBD curve in Fig. 4 will be discussed in the next section, where a new death step algorithm will be presented in order to solve these problems.

3 Scalable parallel birth and death process

We show in this section how we have revised the original birth and death algorithm in order to obtain a parallel birth and death (PBD) process.

3.1 A new birth and death algorithm

3.1.1 A parallel death step

In the algorithm presented in Section 2.3, the death step is computed for each object in the order of the decreasing data-fidelity term. As shown in Section 2.6, this order is important for good quality results because the objects with better data-fidelity terms are hence computed last, ensuring they are unlikely to be deleted due to overlapping with objects with worse data-fidelity terms.

In order to obtain an algorithm which scales with the number of cores, a new way to compute the overlapping energy independently of any ordering is re-

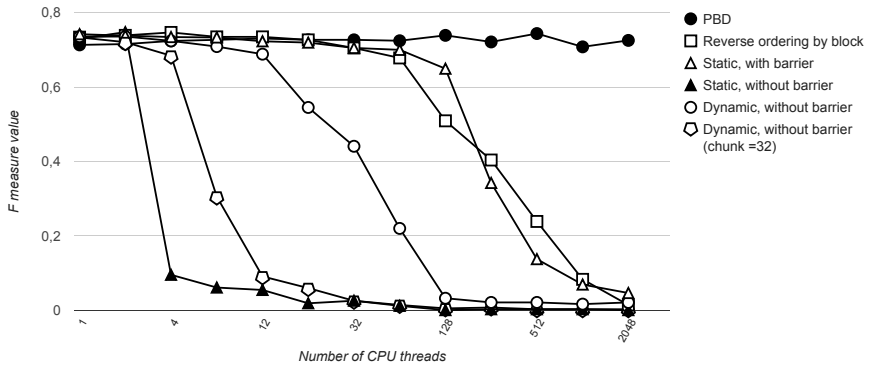


Figure 4: Quality measurements (F-measure) depending on the degree of parallelism (number of threads) for various parallelization strategies. Results obtained on a 1024×1024 image.

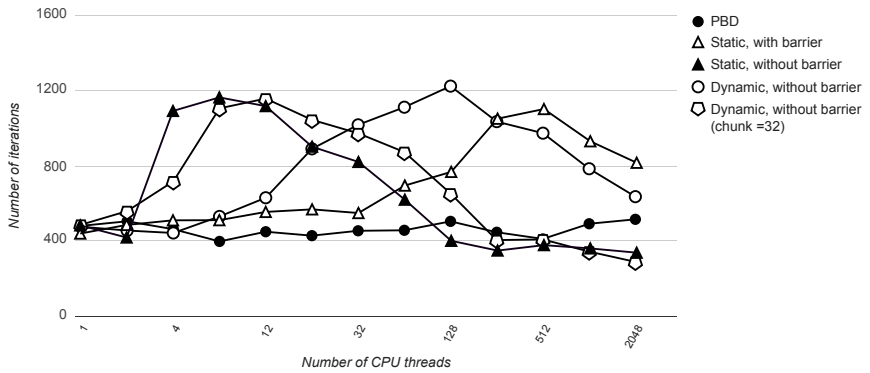


Figure 5: Number of iterations depending on the degree of parallelism (number of threads) for different parallelization strategies. Results obtained on a 1024×1024 image.

quired. In that aim, we propose a new overlapping energy $U_p^*(w_i)$:

$$U_p^*(w_i) = \sum_{p_x \in A_i} \min(p_x), \tag{5}$$

with A_i the set of all pixels inside the ellipse w_i, and $\min(p_x)$ the minimal data-fidelity term of all ellipses overlapping the pixel p_x. The energy $U(\mathbf{w})$ described in equation (2) is now computed as:

$$U(\mathbf{w}) = \gamma_d \sum_i U_d(w_i) + \gamma_p \sum_i U_p^*(w_i). \tag{6}$$

We can now compute this overlapping energy based only on the objects with the best (*i.e.* minimal) data-fidelity terms. Each ellipse thus needs to know the exact number of its pixels which are overlapped by an ellipse with a better data-fidelity term. Therefore, we have to change the overlap map computation step along with the death step. The 2D overlap map now stores for each pixel the minimal data-fidelity term among all ellipses overlapping this pixel. During the new overlap map computation step, for each ellipse we thus have to store its data-fidelity term d in the overlap map, for each of its pixels p, if d is lower than the current value of p.

During the death step, each ellipse can then count the number of pixels having a better data-fidelity term than its own one. It is thus possible to determine which ellipse to keep or to delete in any order. This allows the death step to be computed in parallel. Moreover, when an ellipse is deleted, the overlap map is not modified any more, which reduces the computation load. The overlapping energy is indeed now based on all possible ellipses, including both the already removed ones and the currently kept. This new algorithm ensures that no ellipse will be removed due to an overlapping with a worst ellipse. Thus, the results will be as good as or better than those of the original algorithm.

We can see on Fig. 4 that the quality of the new algorithm matches indeed the one of the original algorithm in a serial execution, and that this quality is not degraded when increasing the number of threads. Moreover, the number of iterations should not change depending on the number of threads: the variations shown in Fig. 5 are due to the stochastic feature of the algorithm, and could be equalized by averaging on an increased number of runs.

3.1.2 Stop criterion

The stop criterion used in [11] – waiting for no new ellipse created and no previous ellipse deleted – is too strict: the number of iterations grows proportionally to the image size. As we are looking for a scalable algorithm, which can handle the biggest images, this criterion is not relevant.

In order to deal with this problem, we introduce a new stop criterion guaranteeing a number of iterations independent of the image size. Our new birth and death algorithm now stops whenever $\frac{R_{\text{total}}}{R_{\text{new}}+R_{\text{old}}} > \mu$, with R_{total} the total number of ellipses kept in the realization after the death step, R_{new} the number of ellipses added during the last birth step and not deleted, and R_{old} the number of ellipses from the past realization deleted during the last death step.

This new criterion checks that there is less than one change (a new ellipse added or an old one deleted) every μ ellipses. As the number of ellipses depends on the image size, this leads to a stop criterion that scales with the image size. Results presented here were computed using $\mu = 500$.

3.2 Efficient deployment on parallel architectures

In order to show the scalability of our new PBD algorithm, we have deployed it on both multi-core CPUs and on GPU. We show here how these deployments have been efficiently performed, and present performance results in Section 4.

We focus here on the processing of one image on one single node with multi-core CPUs or with one GPU. One could easily add an MPI layer to treat multiples images concurrently on multiple nodes. But this application is primarily intended for histopatologists in hospitals where only one single workstation will be available.

3.2.1 Deployment on multi-core CPUs

We have used C++ programming and multi-threading with OpenMP in order to implement in parallel the PBD algorithm on one compute node with multi-core CPUs and shared memory. We now show how each step of the algorithm has been parallelized.

- *Birth step:* as far as the birth step is concerned, the parallelization of the for loop over each pixel of the image is straightforward and each thread treats pixels according to a dynamic scheduling. Each pixel presents indeed a different computation load depending on whether an ellipse will be created in its center or not, which cannot be known a priori. In order to have a coarse enough computation grain per thread, a chunk of 512 is used in the OpenMP dynamic scheduling. The only limitation to the parallelism is the need of atomic operations for the creation of ellipses, as all threads store the created ellipses in the same array. These atomic operations are not expected to be problematic in practice since the number of created ellipses is very small regarding the number of pixels: on a 1024×1024 image, around $20,000$ ellipses are created whereas the number of pixels is 50 times greater.

- *Data-fidelity term:* in this step, the parallelization of the for loop over the number of ellipses is also straightforward. As each ellipse takes a different amount of time to compute U_d due to different ellipse sizes, a dynamic scheduling is used to balance the computation load between each thread.

- *Overlap map computation:* we choose to parallelize this step on the for loop of the number of ellipses, with a dynamic scheduling. As each ellipse needs to write the minimum data energy for each of its pixels, we need to ensure that two parallel threads will not write different minima on the same pixel. This has been solved using a lock table with $10 \times N_T$ locks, N_T being the number of threads. Each line of the image is protected by a given lock, according to a 1D cyclic distribution of the $10 \times N_T$ locks. Thanks to a 1D block distribution of the ellipse array among the threads, and thanks to the approximate geographical locality of the ellipses within the array, this enables us to reduce the lock contention. It can be noticed that a more efficient deployment may possibly be obtained with the future implementations of the forthcoming OpenMP 4.0 standard[1]. This new OpenMP version will indeed enable atomic "exchange" operations that could replace our multiple OpenMP locks (as detailed on GPU in Section 3.2.2).

- *Death step:* the parallelization of the death step is similar to the one of the birth step, and only requires atomic operations to store the kept ellipses in the same array. These atomic operations are not expected to be problematic as the number of kept ellipses is very small regarding the number of created ellipses: on a 1024×1024 image, around 200 ellipses are kept whereas the total number of ellipses is 100 times greater.

Finally, in order to accelerate the generation of random numbers, we used a specific random generator for each thread. The initialization (with different seeds) of this generator is negligible compared to the other steps, and is only done once for multiple images.

3.2.2 Deployment on GPU

We here detail how we have efficiently deployed the PBD algorithm on GPU using CUDA. A CUDA program consists of device codes (kernels) running on the GPU, and a host code running on the CPU that can invoke these device kernels. A CUDA kernel is executed by a grid of thread blocks. When a multi-core implementation can only use a few tens of threads, a GPU requires thousands of threads to achieve efficient computations. We refer the reader to the NVIDIA CUDA documentations[2] for more details on CUDA, especially for the notions of warp, coalesced memory accesses and for the tuning of the number

[1] See: `http://openmp.org`

[2] See: `http://docs.nvidia.com/cuda/index.html`

of threads per block. We now detail how each PBD step has been deployed on GPU.

- *Birth step:* similarly to the OpenMP one, the GPU deployment of the birth step is straightforward. Each GPU thread handles one single pixel, within a 2D grid of the size of the image, and the blocks of threads are organized so as to ensure coalesced memory loads. CUDA atomic operations are also required: like in OpenMP their overhead is expected to be low. The ellipse data are stored in three buffers, containing axe sizes and angles. These buffers are stored on the device. The only communications between the host and the device are the number of ellipses kept at each iteration (one integer).

- *Data-fidelity term:* in a first naive version, directly based on the CPU version, each GPU thread handles one single ellipse to compute its data-fidelity term. In order to better match the fine-grained parallelism of GPUs, we modified this using multiple threads per ellipse. Each thread computes the value of a point from its polar coordinates inside or outside the ellipse. Reductions are then applied to retrieve the variances and the means from the inside and outside borders, needed to compute the Bhattacharyya distance. In that aim, we use an efficient GPU reduction provided in the NVIDIA CUDA GPU computing SDK. [3]

- *Overlap map computation:* in a first naive GPU version, directly based on the CPU version, each thread handles one single ellipse, in order to compute the minimal data-fidelity term for each of its pixels. We scan the ellipse area line by line, from top left to bottom right, and for each pixel we store in the overlap map the data-fidelity term of the current ellipse if this one is lower than the current value. This however leads to uncoalesced memory accesses, to irregularities in the control flow within each warp (each ellipse having a different size), and to a high data throughput per thread.

[3] See: `https://developer.nvidia.com/gpu-computing-sdk`

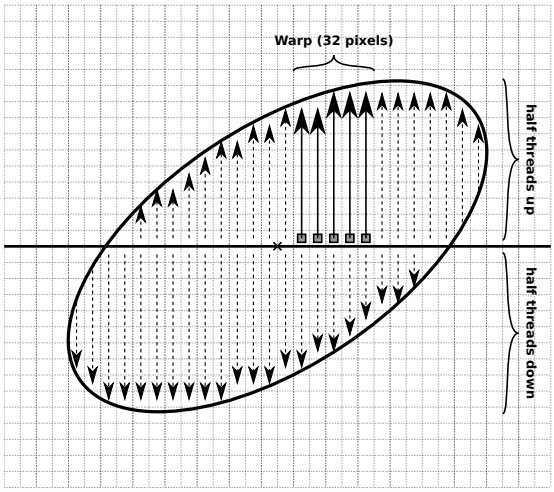


Figure 6: A new scan of the ellipse area on GPU.

Considering instead a surrounding rectangle with one pixel per thread implies a much greater computation load for each pixel to test its belonging to the ellipse. This has been discarded after serial performance tests on CPU.

We thus propose here a new area scan of the ellipse more efficient on GPU. As presented in Fig. 6, we start the scan from the horizontal line passing by the center of the ellipse. One half of the threads then scans up, while the other half scans down. This ensures much more coalesced memory accesses and threads within a warp now share their cache lines.

In order to compute the minimal value during the scan, we could use an atomic "min" function. Unfortunately, the corresponding CUDA function only deals with integers, whereas our data-fidelity term is a single precision floating-point number. Our solution is to use the atomic "exchange" function: for each pixel p of each ellipse, in case the data-fidelity term d of the current ellipse is smaller than the actual value of p in the overlap map, we exchange this value with d and we check that no other thread has performed an exchange with an even lower data-fidelity term in between. In this case, the exchange process

is repeated until the correct minimal value has been written.

- *Death step:* for this step, the overlapping energy computation relies on a scan of each ellipse, which is performed in the same way as in the previous step. However, we need here to sum the number of pixels which are overlapped by an ellipse with a better data-fidelity term. Therefore, since an ellipse area scan is performed by multiple threads, a reduction is needed to sum up the values of all threads. In that aim, we use the GPU reduction provided in the NVIDIA CUDA GPU computing SDK.

  Besides, we store the kept ellipses by performing an atomic "increment" on the number of ellipses. In the same way as for the OpenMP deployment, the number of atomic instructions is negligible compared to the number of ellipses.

Finally, the random numbers are generated on GPU by the MWC64X Random Number Generator developed by David B. Thomas[4], based on the Multiply-With-Carry (MWC) generator [13]. In order to enable each thread to compute quickly a random number, the generator was initialized once for every pixel. This step takes a few seconds on GPU and can be done once for multiple images, so it has not be taken in account in our performance results.

4 Performance results

We now present performance results for the new PBD algorithm on both multi-core CPUs and on a GPU, in order to show our parallel speedups and their scalability when the number of cores increases. All tests are performed on one compute server composed of 48 GB of DDR3 memory, one NVIDIA Fermi C2070 GPU and two Intel X5650 hex-core CPUs running at 2.67 GHz with 2-way SMT. We use `gcc` (version 4.7.2) with OpenMP 3.1 for CPU multi-threading, and CUDA (version 5.0) for GPU programming. Since the computer server is a NUMA architecture, we use `LikwidPin`[5] to bind each OpenMP thread on one CPU core.

Figures 7 and 8 present the detailed and overall performance results obtained on a 4096×4096 image. All computation times are averaged over 20 runs of the complete application. In our new PBD algorithm, the total number of iterations required for a complete cell nuclei extraction does not depend on the underlying architecture, neither on the execution mode (parallel or sequential). More precisely, on such image we require between 500 and 580 iterations, with a mean value of 540. Besides, in the first (respectively last) iterations of the birth and death process roughly 320,000 (resp. 20,000) ellipses are created and roughly 150 (resp. 2,000) kept. In all these performance results, we include all the required CPU-GPU communication times, but we do not consider the following times that can be amortized over the processing of multiple histopathology images: GPU initialization, CPU and GPU memory allocations, initialization of the random seed array. As far as multi-core CPU performance is concerned, we use 1, 6, 12 and 24 threads which correspond respectively to a sequential CPU run (without OpenMP), to a one-processor run without SMT (6 threads on 6 physical CPU cores), to a two-processor run without SMT (12 threads on 12 physical cores) and to a two-processor run with 2-way SMT (24 threads on 12 physical cores).

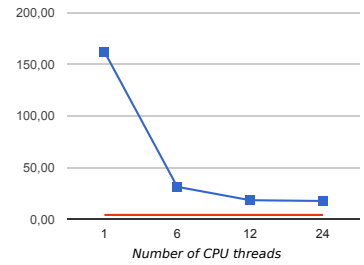
As presented in Section 3.2.1, the parallelization of the birth step of the PBD process is straightforward (except for the atomic operations required to store the resulting ellipses). We therefore obtain very good parallel performance on both CPU and GPU on Figs. 7a and 7b, with speedups up to 9.21 on the two multi-core CPUs and up to 39.46 on the GPU.

The parallelization of the data-fidelity term computation step is also straightforward and offer very good speedups on CPU (up to 14.47) as presented on Figs. 7c and 7d. A good speedup of 21.28 is also obtained on GPU when using one GPU thread per ellipse (CUDA NAIVE). Using multiple threads per ellipse enables us to reach an acceleration of 29.58.

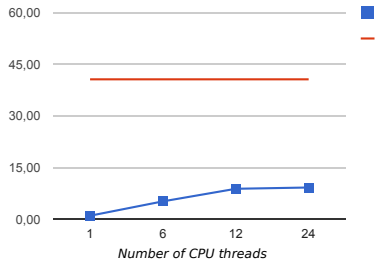
The performance of the overlap map computation

[4] See: `http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-mwc64x.html`

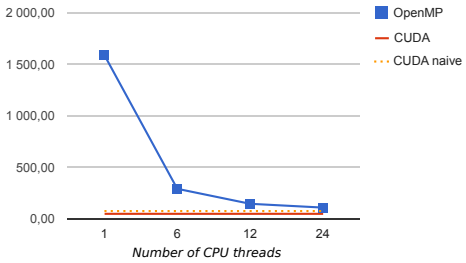
[5] See: `http://code.google.com/p/likwid/wiki/LikwidPin`



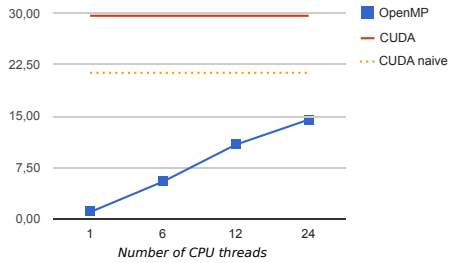
(a) Birth computation times per iteration (ms)



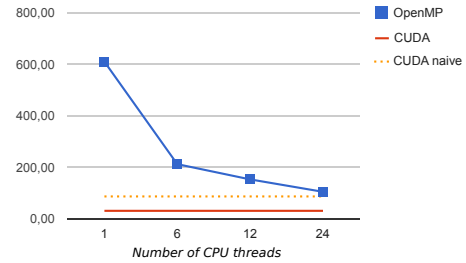
(b) Speedups for the birth step



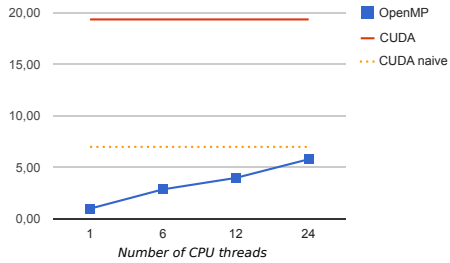
(c) Data-fidelity term computation times per iteration (ms)



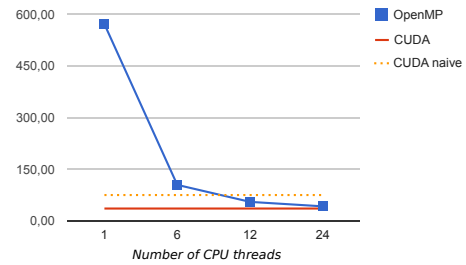
(d) Speedups for the data-fidelity term computation



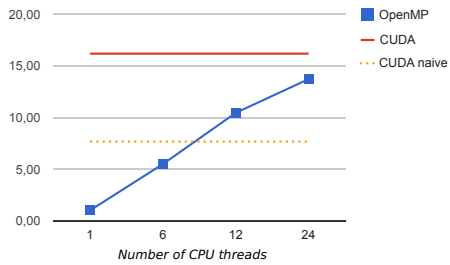
(e) Overlap map computation times per iteration (ms)



(f) Speedups for the overlap map computation

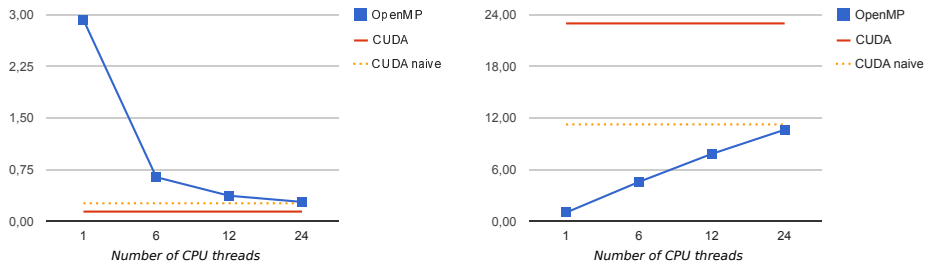


(g) Death computation times per iteration (ms)



(h) Speedups for the death step

Figure 7: Average computation times and corresponding speedups (with respect to the sequential CPU run without OpenMP) for a GPU and 12 CPU cores of the birth and death steps on a 4096×4096 image.



(a) Total computation times per iteration (s)

(b) Total speedups

Figure 8: Total computation times and speedups (with respect to the sequential CPU run without OpenMP) per iteration (averaged over all iterations) for a GPU and 12 CPU cores of the birth and death process on a 4096×4096 image.

step (see Figs. 7e and 7f) is however constrained by the numerous atomic operations required to update, in each pixel, the minimum data-fidelity value of all the ellipses that overlap on this pixel. In OpenMP, we rely on multiple locks (see Section 3.2.1) and we manage to obtain a speedup of 5.79 on the two CPUs. As presented in Section 3.2.1, better performance results may possibly be obtained with the forthcoming OpenMP 4.0. The CUDA code can already benefit from atomic "exchange" operations for this overlap map computation step, but the naive CUDA implementation (with one CUDA thread per ellipse) offers only a speedup of 6.99. This is due to the non-coalesced memory accesses and to the GPU cache trashing. Our new scan of the ellipse area presented in Section 3.2.2 enables here to have much more coalesced memory accesses and to reduce the cache trashing within each warp. Moreover, the reduction over all pixels within each ellipse is also efficiently performed. Thanks to this new ellipse area scan we thus obtain a good speedup of 19.35.

With the new PBD algorithm, the death step is also efficiently performed in parallel on both CPU and GPU as shown in Figs 7g and 7h. On the multi-core CPUs, the few atomic operations required to store the ellipses that live through the death step do not prevent us from reaching a very good speedup of 13.73. On the GPU, the death step also benefits strongly from our new ellipse area scan which offers a speedup of 19.86 (against 11.26 for the naive GPU implementation).

Besides, as far as 2-way SMT on CPU is concerned, one can see that depending on the considered step we obtain very good additional gains between 4.20% and 33.57% for this birth and death process. Moreover, it has to be noticed that for serial executions the new PBD process is already up to 1.3 times faster than the original birth and death process. As presented in Section 3.1.1, the computation load of the death step has indeed been reduced with PBD.

If considering a complete run with all iterations of the four steps on Figs 8a and 8b, we obtain very good overall speedups (up to 11.00) with OpenMP on multi-core CPUs. Whereas the naive GPU implementation offers a limited speedup of 11.64, whereas a better match with the fine-grained parallelism of GPUs enables us to obtain a good GPU speedup of 22.94 over a sequential CPU run. This is almost 2 times faster than the two multi-core CPUs for this application which justifies the GPU code development and optimization. We recall that according to Amdahl's law, we were limited to a theoretical maximum speedup of 5.0 for the complete original birth and death process with a non-parallelized death step. Therefore our new parallel birth and death process enables us to obtain much higher effective speedups

on both multi-core and many-core architectures.

5 Conclusion

In this paper, we have presented a new scalable parallel birth and death algorithm for cell nuclei extraction in histopathology images. Contrary to the original birth and death algorithm, it scales on the number of cores and on the number of ellipses. Thanks to efficient deployements in OpenMP and in CUDA, we manage to obtain very good speedups on multi-core CPUs and good speedups on GPU. We emphasize that such birth and death process can accelerate breast cancer grading applications, as well as numerous other applications based on extraction of elliptically-shaped objects.

We are currently developing an OpenCL implementation of our parallel birth and death algorithm in order to have one single source code for both CPU and GPU with similar (or better) performance than in OpenMP and in CUDA. In particular, we plan to rely on the OpenCL atomic "exchange" operations in order to obtain an efficient overlap map computation on CPU, to study which step of the birth and death process can benefit from the implicit vectorization on multi-core CPU (SSE, AVX) and to test the scalability of our algorithm on the 60 cores of the Intel Xeon Phi coprocessor.

Acknowledgements

This work was performed with the support of the project MICO COgnitive MIcroscope: a cognition-driven visual explorer for histopathology for application to breast cancer grading, a project supported by ANR the French National Research Agency, program TecSan 2010 ANR-10-TECS-015.

References

[1] Christophe Avenel and Maria S. Kulikova. Marked point processes with simple and complex shape objects for cell nuclei extraction from breast cancer h&e images. In *SPIE Medical Image*, pages 1–4, 2012.

[2] A. J. Baddeley and M. N. M. van Lieshout. Object recognition using markov spatial processes. In *International Conference on Pattern Recognition (ICPR)*, pages 136–139, 1992.

[3] X. Bresson, P. Vandergheynst, and J.P. Thiran. A variational model for object segmentation using boundary information and shape prior driven by the Mumford-Shah functional. *International Journal of Computer Vision*, 68(2):145–162, 2006.

[4] X. Descombes, F. Kruggel, C. Lacoste, M. Ortner, G. Perrin, and J. Zerubia. Marked point process in image analysis: from context to geometry. In *International Conference on Spatial Point Process Modelling and its Application (SPPA)*, 2004.

[5] Xavier Descombes, Robert Minlos, and Elena Zhizhina. Object extraction using a stochastic birth-and-death dynamics in continuum. Research Report RR-6135, INRIA, 2007.

[6] C. W. Elston and I. O. Ellis. Pathological prognostic factors in breast cancer. I. The value of histological grade in breast cancer: experience from a large study with long-term follow-up. C. *Histopathology*, 19(3A):403–410, 1991.

[7] A. Hafiane, F. Bunyak, and K. Palaniappan. Evaluation of level set-based histology image segmentation using geometric region criteria. In *International Conference on Symposium on Biomedical Imaging (ISBI)*, pages 1–4, 2009.

[8] Adam Karlsson, Kent Stråhlén, and Anders Heyden. Segmentation of histopathological section using snakes. In *Proceedings of the 13th Scandinavian conference on Image analysis (SCIA)*, pages 595–602, 2003.

[9] Maria S. Kulikova, Antoine Veillard, Ludovic Roux, and Daniel Racoceanu. Nuclei extraction from histopathological images using a marked

point process approach. In *Proc. SPIE Medical Imaging*, San Diego, California, USA, 2012.

[10] George Marsaglia and Arif Zaman. A new class of random number generators. *The Annals of Applied Probability*, 1(3):462–480, 1991.

[11] James R. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, March 1957.

[12] L. Vese and T. Chan. A multiphase level set framework for image segmentation using the Mumford and Shah model. *International Journal of Computer Vision*, 50(3):271–293, 2002.

[13] Y. Xu, J.-Y. Zhu, E. Chao, and Z. Tu. Multiple clustered instance learning for histopathology cancer image classification, segmentation and clustering. In *Computeur Vision and Pattern Recognition (CVPR)*, pages 964–971, 2012.