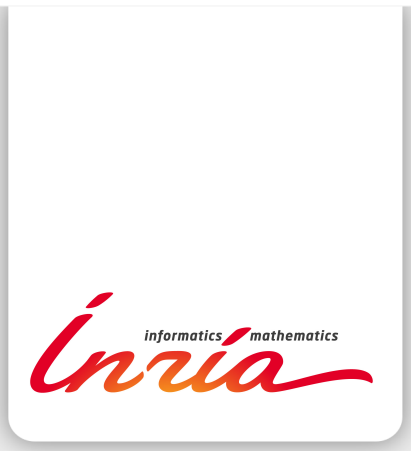
Resource-Centered Distributed Processing of Large Histopathology Images

Daniel Salas, Jens Gustedt, Daniel Racoceanu, Isabelle Perseil

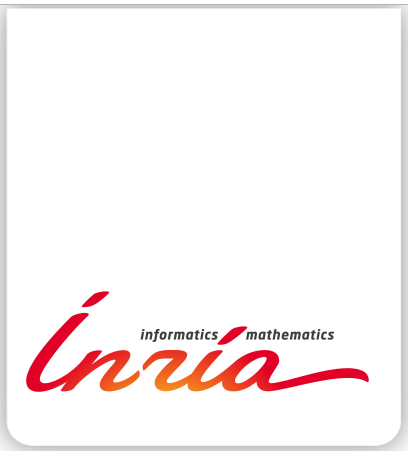


Resource-Centered Distributed Processing of Large Histopathology Images

Daniel Salas
Daniel Racoceanu
Jens Gustedt
Isabelle Perseil

RESEARCH REPORT
N° 8921
June 2016
Project-Team Camus

ISSN 0249-6399 ISRN INRIA/RR--8921--FR+ENG



Resource-Centered Distributed Processing of Large Histopathology Images

Daniel Salas[*†‡] Jens Gustedt[†‡]
Daniel Racoceanu[§] Isabelle Perseil[*]

Project-Team Camus

Research Report n° 8921 — June 2016 — 11 pages

Abstract: Automatic cell nuclei detection is a real challenge in medical imagery. The Marked Point Process (MPP) is one of the most promising methods. To handle large histopathology images, the algorithm has to be distributed. A new parallelization paradigm called Ordered Read-Write Locks (ORWL) is presented as a possible solution for solving some of the unwanted side effects of the distribution, namely an imprecision of the results on the internal boundaries of partitioned images. This solution extends a parallel version of MPP that has reached good speedups on GPU cards, but was not scaling to complete images as they appear in practical data.

Key-words: parallelization; parallel computing; distributed computing; marked point process; ordered read-write locks; cell nuclei recognition; histopathology

* Inserm CISI, Paris, France
† INRIA, Nancy – Grand Est, France
‡ ICube – CNRS, Université de Strasbourg, France
§ Université Pierre et Marie Curie, Paris, France

RESEARCH CENTRE NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Calcul distribué centré resources pour de larges images histopathologiques

Résumé : La détection automatique de noyaux cellulaires est un vrai challenge pour l'imagerie médicale et la lutte contre le cancer. L'un des axes de recherche les plus prometteurs est l'utilisation de Processus Ponctuels Marqués (PPM). L'algorithme tiré de cette méthode a été parallélisé et atteint de bonnes performances d'accélération sur carte GPU. Cependant, cette parallélisation ne permet pas de traiter une lame complète issue d'un prélèvement de biopsie. Il est ainsi nécessaire de distribuer les calculs. Cette distribution entraîne toutefois des pertes de précision au niveau des axes de coupe de l'image. Un nouveau paradigme de parallélisation appelé Ordered Read-Write Locks (ORWL) est une solution possible à ce problème.

Mots-clés : parallélisation; calcul parallèle; calcul distribué; processus ponctuel marqué; ordered read-write locks; détection de noyaux cellulaires; histopathologie

1 Introduction

Breast cancer is the third common cancer all over the world with 1.677 million cases per year [1]. Both its detection and treatment are important concern for public health. *E.g*, pathologists of the Paris hospital *La Pitié Salpétrière* have to analyze more than 2000 stained biopsies per day. They evaluate cancer gradation on a scale from 1 to 3 according to 6 criteria. One of the most important criteria is the nuclei size atypia. Along with the cancer progression, nuclei sizes are growing until becoming abnormally big. In 2012, the International Conference on Pattern Recognition organized a contest [2]. Pathologists had to analyze and grade images. A free database of commented slides results from this contest. This is a great opportunity for designing and testing tools that could offer pathologists a second opinion.

Within this context, a team of the *Laboratoire d'Imagerie Biomédicale*[1] (LIB) has begun to test techniques to automate cell nuclei detection. An original approach in observing cell nuclei atypia is the *Marked Point Process*, MPP. In collaboration with the *Laboratoire d'Informatique de Paris 6*[2] (LIP6), an algorithm has been implemented and then parallelized on CPU with OpenMP and on GPU with CUDA. An acceleration of 22 has been reached with the GPU version. But unfortunately a single GPU is not able support the analysis of a complete slide and so we have to investigate different ways of distributing the algorithm. Section 2 discusses MPP, its existing parallelization and its limits.

In Section 3, we then discuss a conventional distribution method with MPI and a new one using Ordered Read-Write Locks (ORWL) and the benefits of the latter. Section 4 concludes and gives an outlook to future work.

2 The Marked Point Process for Cell Nuclei Recognition in Breast Cancer Images

2.1 Marked Point Process Algorithm

A team from LIP6 has implemented a discretization of the Marked Point Process [3]. The global algorithm is based on a simulated annealing [4] process. The solution tested on each run is the result of a Birth and Death process. The steps of this process are as follows:

Initialization: Image load and resource allocation

Birth Step: A randomly distributed configuration of points is generated. A probability coefficient of birth is decreased in each iteration, to favor convergence.

Mark Step: The best approaching form of a cell nuclei is an ellipse. Points previously created are therefore marked with a wide axis, a short axis

[1] See: https://www.lib.upmc.fr/
[2] See: http://www.lip6.fr/

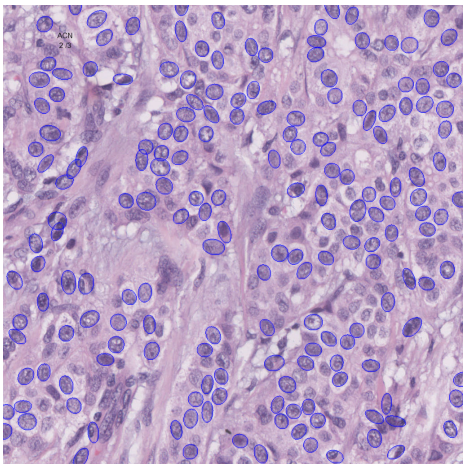


Figure 1: Parallel Birth and Death algorithm result

and an inclination angle. From these parameters, fidelity to the data (also called attachment) is computed, namely the ellipse's accuracy with respect to the underlying image. The Bhattacharyya distance [5] is used to compare the intensity difference between the dark border and the inner cell nuclei.

Neighbors step: A neighbor map is built with the data fidelity values of each pixel. For every pixel of every ellipse, if the map's pixel value is greater than the ellipse's data fidelity value, then the map is updated.

Death Step: A first selection is made between overlapping ellipses. Only the best attached survive. Then a death rate filter is applied on ellipses created from the beginning. This filter is based on the data fidelity value and a death probability. This coefficient is increased in each iteration to favor convergence.

Convergence Test: Convergence is reached when all objects created in the birth phase are killed in the death phase.

2.2 A Parallel Algorithm for MPP

LIP6 has also parallelized this algorithm on CPU with OpenMP (see a processed image in Figure 1) and on GPU using CUDA. In each step, the parallelization is straightforward. The loops used to iterate over pixels are distributed to available threads. An initial parallelization problem concerning the death step has been solved using the principle of the neighbors map to test and set the minimal data fidelity values.

Reported parallelization speedups [6] are shown in Table 1.

	Sequential	4 OpenMP threads	GPU
Time	200 seconds	60 seconds	10 seconds
Speedup	-	3.32x	22 x

Table 1: MPP results on a 1024 by 1024 pixels image

2.3 Parallel Birth and Death Limitations

The PBD (Parallel Birth and Death) algorithm is reaching a good speedup on GPU devices. But it is limited to the memory size of the GPU. A complete slide size may be up to 100,000 by 100,000 pixels at full resolution. Modern GPU cards offer 12GiB memory which is almost the size of our slide. In addition to the image, the application allocates 12 times the number of pixels in arrays of floating points. The total memory needed is about 450 GiB. This is far more than the available memory of our GPU.

Therefore, the PBD algorithm has to use subdivisions of the complete slide. On our 12GiB memory GPU, we could maximally take in charge an image side of about 16,000 px. If we consider analyzing images with a side length of 10,000 pixels, a complete slide would be composed of a hundred images. This leads us to another kind of problem. Subdividing the slide will have the side effect of truncating cell nuclei. The truncated ellipses would not be taken into account for the diagnostic. For an image of x40 magnification, if we consider that our cell nuclei is 80 pixels long, the band of image data that would not be considered correctly would represent a surface of about 3,000,000 px. For the total image this surface represents about 3.2%. If we look at the criteria of nuclei size for cancer gradation [2] (see Table 2),

grade	% of atypic nuclei
1	0 to 30% nuclei are bigger
2	30 to 60% nuclei are bigger
3	more than 60% nuclei are bigger

Table 2: Breast cancer gradation for atypic nuclei criteria

the unconsidered 3% of nuclei are sufficient for establishing a diagnostic of grade 1. A cancer may be therefore detected in its early stage. In the following we present a strategy for taking into account carefully the boundary pixels by using a new distributed and parallel computation model.

3 Distributed PBD Process

In distributed computing, each processor has its own private memory. This adaptable memory size enables us to manage bigger issues and in particular to analyze bigger images.

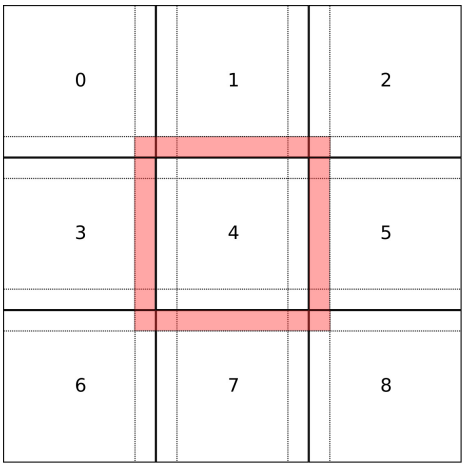


Figure 2: Image distribution strategy

3.1 Implementation strategy

The main problem for distributing the algorithm is the neighbor step. For each ellipse data fidelity has to be taken into account to classify for the best attached values. A global map could be distributed among all nodes, according to Figure 2. Every processor could then compute its part of the image locally. At the end of the neighbor step, when the local neighbor map has been drawn, the processors could send each other messages to detect crossing ellipses, according to Figure 3. In the example, Processor 4 sends a message to Processor 1 and informs him of the position and data fidelity value of its ellipses. Processor 1 should be listening asynchronously for this possible message. To avoid deadlock situations, the sends and receives should be non blocking. Then, synchronization barriers must be use to guarantee the coherence of the execution. Such barriers would force the first processor to wait until the last has finished. So, the whole execution would be as slow as the slowest processor. If the work load is not completely balanced, this would add a lot of waiting time, which can be largely sub-optimal. Furthermore, this method could easily encounter a deadlock situation.

Another distribution strategy consists in writing directly into the map of our neighbors. In Figure 2, Processor 4 would access the neighbor map of its surrounding neighbors (the red band). In the example of Figure 3, Processor 4 adds the data fidelity value of its ellipse into Processor 1's neighbor map during its neighbor step. The step in which Processor 1 is at this moment does not matter. The value is taken into account, when Processor 1 proceeds to kill his worst attached ellipses the next time. Then, the map of the whole slide can be updated along with the computing process, no synchronization is necessary. This strategy is possible thanks to a new parallelization paradigm called Ordered

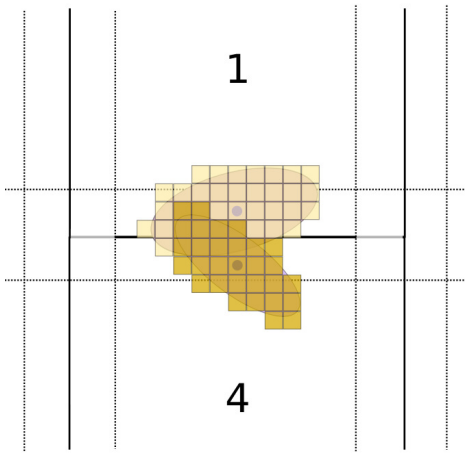


Figure 3: Data fidelity competition

Read-Write Locks[3] (ORWL).

3.2 A solution with ORWL

Before explaining how ORWL will handle the remote write of the neighbors map, we will present its global concepts.

3.2.1 A new paradigm

ORWL [7] models parallel and distributed computing by means of tasks. Its particularity is the way in which resources can be shared by the different tasks. Accesses are regulated by a FIFO that guarantees the liveness of the application and the equity of access for the tasks.

3.2.2 Workflow

Computation is distributed according to the number of available nodes in a configuration file. The TakTuk library [8] is in charge of sending and collecting files and starting the remote tasks. During a network recognition phase, an address book collects all nodes properties that are necessary to communicate (IP address, port number). Then this book is distributed to all nodes, allowing a point to point communication between all tasks. In a post-computation phase, TakTuk collects the results.

[3]See: http://orwl.gforge.inria.fr/orwl-html/

Listing 1: ORWL task launch loop

```
for(size_t i=0; i<orwl_lt; i++) {
  task_obj* task = P99_NEW(task_obj, param1);
  task_obj_create_task(task, orwl_tids[i]);
}
```

3.2.3 Tasks

The number of ORWL tasks is defined at runtime. For each task, a new thread is created and tasks are identified by an unique number `orwl_mytid`, see Listing 1 for launching tasks.

In this example, the number of tasks is given by `orwl_lt`. **P99_NEW** is a macro defined in `P99`[4]. It allocates an element of type `task_obj` and initializes it with the argument `param1`. `task_obj_create_task` executes the block in Listing 2

Listing 2: ORWL handlers definition

```
ORWL_DEFINE_TASK(task_obj) {
 // code executed by each task
}
```

3.2.4 Handling the data

ORWL data is held in abstract objects called locations. These are uniquely identifiable entities represented by a global location ID (`orwl_myloc`). Access to the locations is handled through a queue. To gain control in read or write mode to a location, a handle must insert a request in the queue. Locations have to be declared and an amount of memory has to be allocated as follows:

Listing 3: Parametrization of an ORWL program

```
/* global                                  */
ORWL_LOCATIONS_PER_TASK(north_loc);
/* inside a function                       */
orwl_scale(100,
 ORWL_LOCATION(orwl_mytid, north_loc));
```

`orwl_handle2` for iterative access are initialized and defined as in Listing 4. The macro **ORWL_HANDLE2_INITIALIZER** initializes the handle object. A memory size is allocated by a call to `orwl_scale`. Functions `orwl_write_insert` and `orwl_read_insert` are used, respectively, to connect an handle with a location in write or read mode. The last parameter of the function provides the initial position of the request in the queue. This strict ordering will schedule data access such that it is deadlock free [9].

[4] http://p99.gforge.inria.fr/p99-html/

Listing 4: ORWL handlers definition

```
orwl_handle2 here_hdl = ORWL_HANDLE2_INITIALIZER;
orwl_handle2 there_hdl = ORWL_HANDLE2_INITIALIZER;
orwl_scale(DATA_SIZE,
  ORWL_LOCATION(orwl_mytid, n_loc));
orwl_write_insert(&here_hdl,
  ORWL_LOCATION(orwl_mytid, here_loc),
  0);
orwl_read_insert(&there_hdl,
 ORWL_LOCATION(orwl_mytid - n, there_loc),
 1);
```

Data is accessible through critical sections as in Listing 5.

Here, the location associated with `north_hdl` handle is mapped with a pointer to `unsigned char`. Modifying data through the pointer directly modifies the data of the location itself.

Once a task has gained access to a critical section in write mode, the access is exclusive.

Listing 5: handler data access in write mode

```
ORWL_SECTION(north_hdl) {
 unsigned char *north_ptr = orwl_write_map(north_hdl);
 memcpy(north_ptr, my_map.north, band_size);
}
ORWL_SECTION(xnorth_hdl) {
 unsigned char *xnorth_ptr = orwl_read_map(xnorth_hdl);
 interfaceStitchBand(str, size, there_hdl_ptr, frontier);
}
```

3.2.5 An ORWL implementation of the MPP

ORWL is implemented in C. On the other hand, the PBD algorithm has been written in C++. A first step of the implementation consisted in transposing the code from C++ to C. However, we preserved some useful functionalities of the C++ library, for keeping the use of the CImg[5] library.

In CImg, an image is represented as a matrix of pixels. The type can be determined during image creation. In our case, we have used images with pixel type `unsigned char`. Pixels values are accessible through an `image.data` object. The three color channels R, G, and B are stored linearly in a single memory buffer as follows: $R_0R_1R_2\cdots G_0G_1G_2\cdots B_0B_1B_2\cdots$.

Other CImg functions are `get_rows` and `get_columns`. They allow to get horizontal and vertical parts of an image. This is used in the distributed PBD algorithm to splice the borders of the neighbor map. The border width is half the wide axis of the ellipses.

3.2.6 The Distributed PBD algorithm

Figure 4 shows a workflow of the algorithm.

[5] http://cimg.eu/

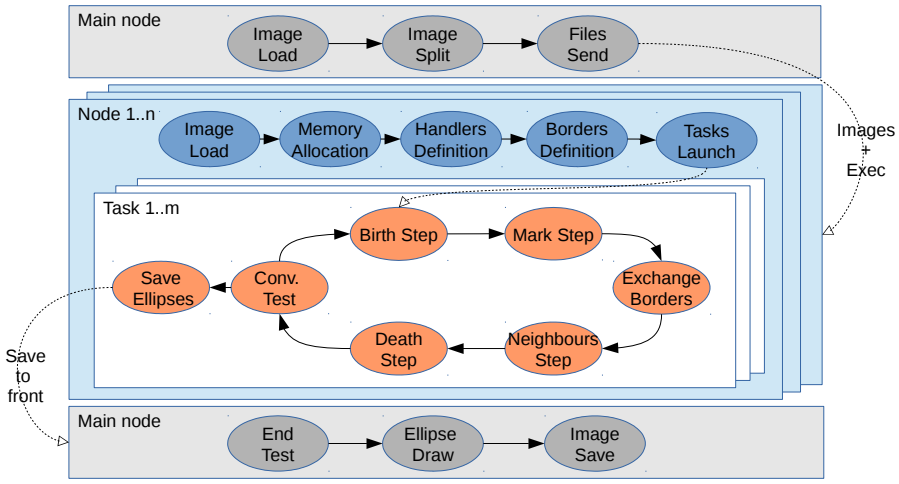


Figure 4: Workflow of PBD ORWL implementation

In an initial phase, a main node loads the entire image and splits it. A Ruby script using TAKTUK sends files to the computing node. Then each node loads its own image. Locations and handlers are defined. Several instances or `orwl_tasks` can be launched on a single node to parallelize the computation (see Listing 1). After the *Mark Step*, a new *Exchange Border* step is added. A first critical section enables a processor to write its border into a location. Then, a second critical section allows to retrieve the borders of the neighbors (Listing 5). Own borders are obtained by slicing the neighbors map. The `interfaceStitchBand` function adds neighbor borders to the processors neighbor map. Thereby, in the *Neighbors Step* the computation takes into account neighbors ellipses.

Once a Birth and Death process has converged, ellipse properties are sent to the main node. The final ellipse drawing can be done in a single loop.

4 Conclusion and future work

Thanks to the ORWL model, we presented a distributed version of the PBD algorithm that is ready to scale. It allows to properly consider all cell nuclei on the boundary between two partial images. An implementation of this algorithm is in progress and tests will be executed on a cluster of GPU and Xeon Phi accelerators. The final application is designed to map an entire slide in a few minutes. The goal is to let pathologists navigate through the slides, such that they may zoom in and out, just as they used to do with a microscope. Colored ellipses could be shown on a separate layer, helping to detect nuclei size atypia.

References

[1] W. H. Organization, "GLOBALCAN 2012: Estimated Cancer Incidence, Mortality and Prevalence Worldwide in 2012," 2012. [Online]. Available: http://globocan.iarc.fr/

[2] I. C. on Pattern Recognition, "MITOS-ATYPIA-14," 2014. [Online]. Available: http://mitos-atypia-14.grand-challenge.org/

[3] C. Avenel, P. Fortin, and D. Béréziat, "Parallel birth and death process for cell nuclei extraction in histopathology images," Jul. 2013, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00844001

[4] K. E. Geltman, "The Simulated Annealing Algorithm," 2014. [Online]. Available: http://katrinaeg.com/simulated-annealing.html

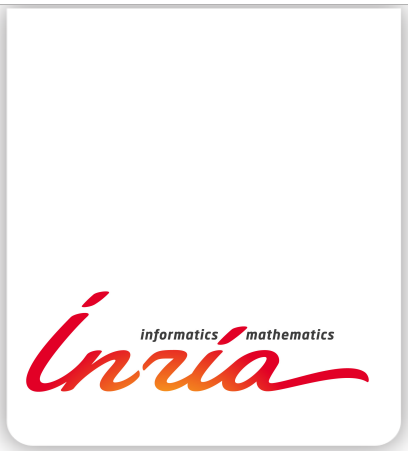
[5] F. J. Aherne, N. A. Thacker, and P. I. Rockett, "The bhattacharyya metric as an absolute similarity measure for frequency coded data," *Kybernetika*, vol. 34, no. 4, pp. 363–368, 1998.

[6] C. Avenel, P. Fortin, and D. Béréziat, "Parallel birth and death process for cell nuclei extraction in histopathology images," 2013.

[7] J. Gustedt and E. Jeanvoine, "Relaxed Synchronization with Ordered Read-Write Locks," in *Euro-Par 2011: Parallel Processing Workshops*, ser. LNCS, M. Alexander *et al.*, Eds., vol. 7155. Springer, May 2012, pp. 387–397. [Online]. Available: http://hal.inria.fr/hal-00639289

[8] B. Claudel, G. Huard, and O. Richard, "TakTuk, Adaptive Deployment of Remote Executions," in *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*. Munich, Germany: ACM, 2009, pp. 91–100. [Online]. Available: https://hal.inria.fr/hal-00788923

[9] P.-N. Clauss and J. Gustedt, "Iterative Computations with Ordered Read-Write Locks," *Journal of Parallel and Distributed Computing*, vol. 70, no. 5, pp. 496–504, 2010. [Online]. Available: https://hal.inria.fr/inria-00330024



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399