TOWARDS LARGE SCALE AUTOMATED INTERPRETATION OF CYTOGENETIC BIODOSIMETRY DATA

Yanxin Li[1], Asanka Wickramasinghe[1], Akila Subasinghe A.[1], Jagath Samarabandu[1], Joan Knoll[1,2], Ruth Wilkins[3], Farrah Flegal[4], and Peter Rogan[1,2]

[1]*Western University*, [2]*Cytognomix Inc.*, [3]*Health Canada*, [4]*Atomic Energy of Canada Ltd., Ontario, Canada*

Abstract—**Cytogenetic biodosimetry is the definitive test for assessing exposure to ionizing radiation. It involves manual assessment of the frequency of dicentric chromosomes (DCs) on a microscope slide, which potentially contains hundreds of metaphase cells. We developed an algorithm that can automatically and accurately locate centromeres in DAPI-stained metaphase chromosomes and that will detect DCs. In this algorithm, a set of 200-250 metaphase cell images are ranked and sorted. The 50 top-ranked images are used in the triage DC assay (DCA). To meet the requirement of DCA in a mass casualty event, we are accelerating our algorithm through parallelization. In this paper, we present our finding in accelerating our ranking and segmentation algorithms. Using data parallelization on a desktop system, the ranking module was up to 4-fold faster than the serial version and the Gradient Vector Flow module (GVF) used in our segmentation algorithm was up to 8-fold faster. Large scale data parallelization of the ranking module processed 18,694 samples in 11.40 hr. Task parallelization of Image ranking with parallelized labeling on a desktop computer reduced processing time by 20% of a serial process, and GVF module recoded with parallelized matrix inversion reduced time by 70%. Overall, we estimate that the automated DCA will require around 1 min per sample on a 64-core computing system. Our long-term goal is to implement these algorithms on a high performance computer cluster to assess radiation exposures for thousands of individuals in a few hours.**

I. INTRODUCTION

CYTOGENETIC biodosimetry, which is recommended by the WHO for assessing radiation exposures [1], determines the frequency of dicentric chromosomes (DCs) on a microscope slide of fixed metaphase cells and compares this result to a dose-response calibration curve. In a radiation-associated mass casualty event, hundreds of thousands of patient samples (each consisting of 200-300 images) must be processed rapidly to identify and treat those with clinically significant exposures [2]. Microscope images of metaphase chromosomes are variable in morphology, requiring dedicated image segmentation methods. We have previously developed an algorithm that can automatically and accurately locate centromeres in chromosomes and detect DCs in samples prepared by either clinical cytogenetic or biodosimetry reference labs [3]. Fig. 1 indicates the flow chart of our algorithm. We stain chromosomes with 4'-6-diamidino-2-phenylindole (DAPI) before image capture. The system consists of 8 modules that are functionally classified as Image Ranking, Image Segmentation via Gradient Vector Flow (GVF) and Centerline-based Centromere Detection. Image Ranking classifies and sorts images to eliminate cell images with overlapping chromosomes or incomplete chromosome set [4]. Individual chromosomes in a selected image are separated by image segmentation via GVF [5], [6]. Centerlines are extracted through a skeleton pruning method based on Discrete Curve Evolution (DCE) [7], [8]. This improves on previous methods using Medial Axis Transform and morphological thinning, which both suffer from spurious branching and lead to incorrect centromere placement. The resulting centerline is more accurate than previous approaches, especially for bent chromosomes. Traditionally, the centromere location has been determined by the most prominent chromosome constriction along the centerline. For greater accuracy, both chromosome width and fluorescence intensity of the DAPI-stain are used to locate the centromere for every cross section of a chromosome. The first centromere is located at the cross section with the minimum width and intensity. The reliability of detected centromere location is measured by 'centromere confidence value' (CCF) [9]. Preparative methods used by biodosimetry reference labs can result in short chromosomes with separated sister chromatids, affecting accurate centromere detection. The prototype software implements algorithms to detect separated chromatids and a fuzzy learning, rule-based method to refine mis-located centromeres. By masking the first centromere, the second centromere is located in the same way. Results are independent of the length, shape and structure of chromosomes in different cells which laboratory protocol is followed, or the patient source.

In a mass casualty event, thousands of DCA tests will need to be accurately interpreted within a few days to assure that those individuals exposed to significant levels of ionizing radiation are prioritized for treatment. This level of throughput is not possible by current manual scoring of DCA with Giemsa-stained chromosomes. Our current

978-1-4673-1975-1/12/$31.00 ©2012 IEEE

efforts to accelerate these procedures are focused on recoding MATLAB-based scripts used to develop DCA image processing algorithms as C++/OpenCV based software and accelerating this software via parallelization with the Message Passing Interface (MPI) and Intel Threading Building Blocks (TBB).

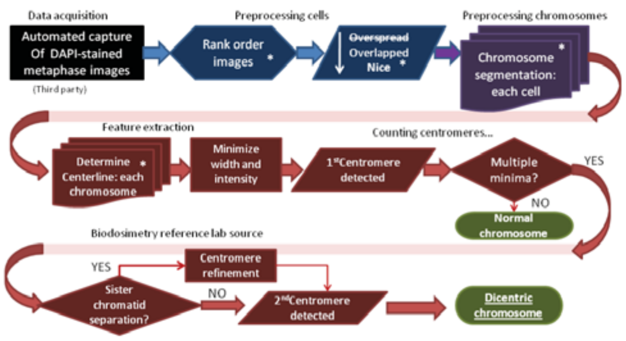

Fig. 1. Flow chart of DCA algorithm

II. Method

We implemented and evaluated the performance of data and task parallelization of the completed ranking and GVF C++ modules. Data parallelization distributes each image / chromosomes to different compute nodes. It provides high parallelism and low overhead and is suitable for our project because of the large quantity of distributable data that must be processed. By contrast, task parallelization modifies key low-level steps in the serial data analysis with parallel approaches. The performance depends on whether and how these steps can be parallelized, CPU availability and overhead costs. It is effective when there is excess computing capacity remaining after data parallelization.

A. Data Parallelization

Based on the large number of discrete images in the dataset, we anticipated that parallel processing would be likely to significantly positively impact performance. Data parallelization is implemented based on Message Passing Interface (MPI) or Intel Threading Building Blocks (TBB). These platforms works well with computing cluster systems and shared memory systems, and fewer modifications are required to convert serial codes to a parallel version compared with other platforms. Each computing node will be assigned with a subset of images and chromosomes to process, respectively in image ranking and GVF modules.

B. Task Parallelization

Since task parallelization usually produces higher overhead, only computationally expensive steps are parallelized. Binary image labeling, which requires 47% of processing time in the ranking procedure, and circulant tri-diagonal matrix inversion which is iterated several times in GVF to derive the contour of a chromosome, have been parallelized here.

Binary image labeling is based on the union-find method (which represents components with trees; union trees are used to merge components; and labeling finds the corresponding tree) [10]. Two passes are used to label each image with a set of trees, representing all components in the image (i.e. Chromosomes). During the first pass, coarse labeling, building and union of component trees occurs. In the second pass, labeling is refined by searching and locating the component trees. Parallel labeling divides an image into sub-images, labels each sub-image, and combines results from all sub-images. Considering an image with n pixels, the time required for a serial process to label components is:

$$t = n + c_2 fn + c_1 . \qquad (1)$$

c_1 is the number of trees building and union operations which depend on the number and shapes of components in the image. c_2 is the average number of tree levels. In metaphase images, both c_1 and c_2 are constants whose values can be empirically defined in integral intervals. f represents the proportion of foreground pixels in all in the images. If an image is divided to k sub-images and processed in a p processors computer with parallel labeling, the theoretically processing time is

$$t = \lceil k / p \rceil (n / k + c_2 ' fn / k + r + c_1 ') + \log_2 k . \qquad (2)$$

$c_1 '$ is the number of tree building and union operations, which themselves depend on the number and shapes of components in the sub-images. $c_2 '$ is the average number of tree levels in all sub-images. r is the number of columns of the image. Division of an image into k sub-images requires a time complexity of $\log_2 k$. The time to labeling each sub-image is $n / k + c_2 ' fn / k + c_1 '$, In merging labeled adjacent sub-images, the first row and last row of each sub-images are scanned, which costs r.

To solve an energy function in GVF requires inversion of circulant tri-diagonal-like matrices. Matrix inversion is a NC complex class problem, implying that it should be feasible to effective improve algorithm performance by parallelization. The most general method to invert a matrix is Gaussian Elimination [11], which has a time complexity of $O(n^3)$ to invert a n-by-n matrix. For circulant matrices a widely-used method is using Fourier Transform. If Fast Fourier Transform (FFT) like the library FFTW used [12], the work (time required by parallel program to run using a single CPU) of inversion is $O(n\log n)$ and the span (time required by parallel program run with infinite CPUs) of inversion is $O(\log n)$. For a circulant matrix, inversion is equivalent to solving a system of linear equations represented by this matrix. We parallelized the Thomas Algorithm [13], which has an equivalent or better performance than FFTW. Assuming p processors available, the time is

$$t = \sum_{i=1}^{\lceil \log_2 n \rceil} \lceil n/(2^i p) \rceil. \tag{3}$$

The work of parallelizing the Thomas Algorithm is $O(n)$ and the span is $O(\log n)$.

III. Experiments

Both data parallelization and task parallelization of ranking module and GVF module on small datasets were tested on an 8-core i7 desktop. In ranking module, 200 metaphase cell images, the least number of images required for a valid sample, were ranked and in GVF module, 46 chromosomes, number of chromosomes in a human cell, were processed to extract their contours. In these tests on a desktop, data parallelization was implemented with MPI and task parallelization was implemented with TBB. Large scale data parallelized ranking was also tested. With a 64-core shared memory computer (Symmetric Computing), 18,694 samples, each of which consists of 250 to 300 images, were ranked by data parallelized ranking module implemented with TBB. Besides module testing, the parameters and performance of these two parallelized functions were subsequently tested by task parallelization (described below).

IV. Results and Discussion

A. *Data Parallelization*

In data parallelization on a multiprocessor desktop PC, processing time of ranking and GVF module was recorded. Parallel ranking was 4-fold faster than serial version and parallel GVF was almost 8-fold faster than serial GVF, as shown in Table I. In the large scale test of parallel ranking module implemented with TBB, 18,694 samples were processed in 11.4 hours which was around 6 percent of time needed by serial ranking on the same computer (see Fig. 2). The fraction of non-parallelizable code by data parallelization is usually very small. Hence according to Amdahl's law [14], the theoretical speedup from data parallelization could increase approximately linearly with the increasing of number of processors used. This was found to be the case for the test of parallel GVF module, but not for the test of parallel ranking module. This may due to overhead of task management from MPI. We also find that in the large scale data parallelization that involved > 30 processors, there was no obvious gain in speed regardless of the amount of memory used. We attribute this to overhead cost, however analysis of performance may be consistent with suboptimal process partitioning by TBB.

B. *Task Parallelization*

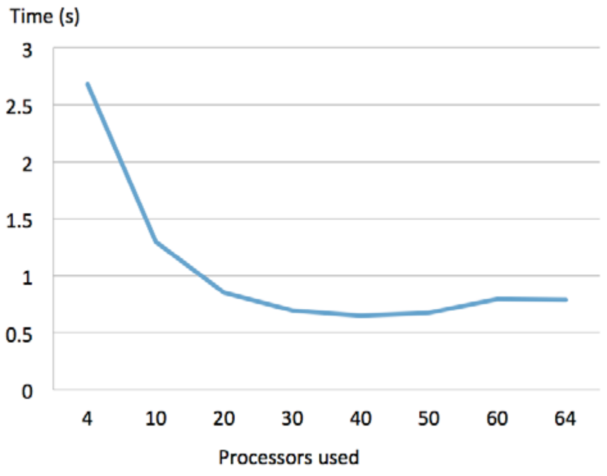

Fig. 2. Large scale test of Data Parallelization on Ranking module implemented with TBB. System tested consisted of 64-AMD processors, with 1.5 Tb shared memory. Average time for one sample (250-300 images) was recorded in 100 samples.

TABLE I
Performance of Data Parallelization on Ranking and GVF modules using a 8-core desktop system

Module tested with MPI	Ranking (seconds)	GVF (seconds)
Serial code (2,4,6,8 processors)	10.761	13.444
Parallel code with 2 processors	5.008	6.432
Parallel code with 4 processors	2.526	3.155
Parallel code with 8 processors	2.216	1.755

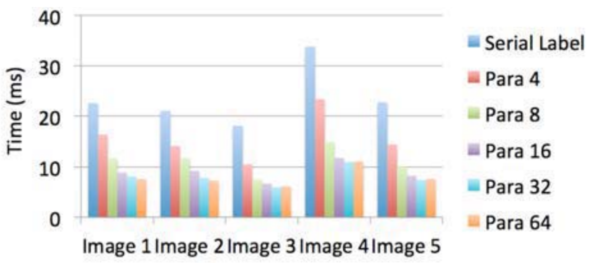

Fig. 3. Comparison of serial and parallel labeling for different numbers of sub-images. Para 4 signifies division into 4 sub-images. 32 sub-images provide the best performance, which was used in following test.

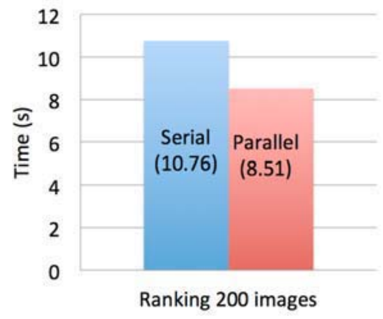

Fig. 4. Comparison of ranking speed using serial vs. parallel labeling.

TABLE II
Matrix inversion times using TBB on a desktop computer with 8 logic cores

Matrix size: N	Average Time (millisec) for serial inversion provided by OpenCV	Average time (millisec) for parallelized inversion
100-200	5.8399	0.2134
201-300	27.0717	0.44
301-400	75.311	0.7223
401-500	271.7779	1.1422
501-600	818.2553	1.7553

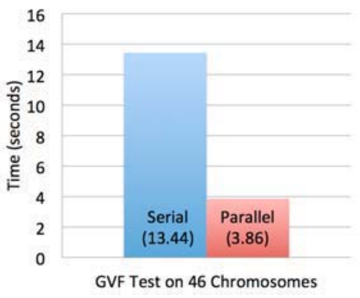

Fig. 5. GVF performance comparison of serial and parallelized matrix inversion on a single image

In task parallelization, different numbers of sub-images in parallel binary image labeling were tested. Dividing an image to 32 sub-images brought the best performance, 2 to 3 times faster than the serial labeling (Fig. 3). Ranking module with this parallelized labeling saved 20% time compared with ranking module with serial labeling (Fig. 4).

In GVF module, matrices to be inverted usually are of size 150 to 300. We tested the performance of general Gaussian Elimination provided by OpenCV and parallelized Thomas algorithm on matrices whose sizes ranging from 100 to 600, as in Table II. The computing time of Gaussian Elimination grows quickly with the increasing of matrix size, while the time growing of parallelized Thomas algorithm is well controlled. GVF module with parallelized matrix inversion saved around 70% time compared with the GVF module with general Gaussian Elimination matrix inversion (Fig. 5).

C. *Overall Performance with Parallelization*

We attempt to estimate the time to required analyze a single metaphase cell from the sum of each of the steps required to detect DCs, excluding overhead from communication between these processes. As recoding and parallelization of all steps have not been completed, the expected performance improvements due to implementation

TABLE III
Observed performance and comparison of different versions of Ranking and GVF modules

Stages of chromosome image processing	Image ranking	GVF segmentation	Maximum speedup
Time to process one sample (sec; Matlab)	266	2290	
Time to process one sample (sec; C++/OpenCV, Serial)	10.7	670	20-fold compared with Matlab
Time to process one sample (sec; C++/OpenCV; 8-core Parallel)	2.22	88	5-fold compared with serial C++
Time to process one sample (sec; C++/OpenCV; 64-core Parallel)	0.65		15-fold compared with serial C++

TABLE IV
Expected performance and comparison of different DCA system versions

Stages of chromosome image processing	Time to process one sample (sec; Matlab)	Time to process one sample (sec; C++/ OpenCV) Above sub-cells: serial; Below sub-cells: 64-core parallel	Performance (accuracy compared to expert cytogeneticist) Reference [3], [9]
Image ranking	266	10.7	~98%
		0.65	
Chromosome separation [a]	722	36 *	>90%
		2.4 **	
GVF segmentation	2290	670	
		43.6 **	
Centerline Extraction	1195	48 *	
		3.12 **	
Centromere detection	55	3 *	96.6% normal; 85% DC
		0.195 **	
Sister chromatid separation	322	16.1 *	93.1%
		1.05 **	
Centromere Refinement	3240	162 *	100%
		10.5 **	
Expected Time for entire system of one sample	166 min	18 min *	98% DCA; 88% overall
		1.02 min **	

* Estimated time, based the observation that serial C++ ranking module is 20 fold faster than Matlab ranking module. ** Estimated time based on the observation that 64-core parallel C++ ranking module is 15 fold faster than serial C++ ranking module.

[a] Assumes ≤ 10 touching or overlapping chromosomes present in an image.

of C++ serial and parallel versions of completed MATLAB modules were estimated from the performance of completed versions of the metaphase ranking software module. Table III indicates measured and estimated elapsed times for each module, based on the data collected from parallelization testing and estimation of performance for modules under development. GVF segmentation is the most compute intensive process, however inversion of circulant tri-diagonal-like matrices has not yet been incorporated into this code (which, according to Table II, should result in a 25-50 fold improvement in speed). We assume 200 cells will be ranked for each sample and the complete DCA will involve triage assessment of the top ranked 50 cells. With a desktop PC running the serial version, it should be feasible to analyze one sample in 18 min, which is faster than the time required for manual scoring. With parallelization, this performance should be significantly improved. The Ranking module is ~20 fold faster with serial C++ code than the Matlab routines. Assuming a similar performance improvement is achieved for the serialized C++ modules under development and given that parallelization of the Ranking C++ code with 64 cores accelerates processing by an additional 15 fold, we estimate that C++ parallelization of these modules will give similar levels of speedup. We attempted to benchmark the entire process based on the gains obtained by implementation of the parallelized ranking procedure. Table III shows details of the speedup from Matlab to serial C++ and from serial C++ to parallelized C++. In Table IV, we apply these scaling factors to the other Matlab codes. With this caveat, approximately 1 min should be required to analyze each sample using data parallelization on 64 cores. In our testing, we determined however that disk I/O can bottleneck processing of large numbers of samples, and strategies will need to be developed to minimize these effects. Nevertheless, numerous samples can be analyzed simultaneously by exploiting multi-scale cluster computing architectures and data parallelization. Theoretically if 128 cores are used for data parallelization, the processing time should be half that of a similar 64-core system, from which we infer that a 128 core system can process a sample in 31 seconds. It should be feasible to analyze 1000 samples in 8.5 hours with a 128 CPU cluster computer.

V. Conclusion

In endeavoring to accelerate this system, we developed and tested two software parallelization schemas. Data and task parallelization in a desktop computing environment reduce CPU requirements up to 86% and 70% respectively, relative to the serial versions of this software. Large scale data parallelization required only 6% of the time used by the corresponding serial process. These results demonstrate that parallelization can efficiently accelerate interpretation of the dicentric chromosomes analysis and suggest that parallelization of the remaining modules of our system should be worthwhile. Once completed, it should be feasible to combine data and task parallelization approaches to evaluate larger scale hardware configurations. It is encouraging that such a strategy may be able to achieve the image processing throughout demanded by the short diagnostic and treatment windows to analyze a large number of individuals exposed to a varying levels of ionizing radiation.

Acknowledgment

Supported by the Western Innovation Fund (University of Western Ontario), Natural Sciences and Engineering Research Council of Canada and the DART-DOSE CMCR (5U01AI091173-02 from the US Public Health Service). We thank Symmetric Computing (Boston MA) for access to their computing facility. PKR also acknowledges support from Canadian Foundation for Innovation and Canada Research Chairs Secretariat.

References

[1] W. F. Blakely, *Health Physics*, 89 (5), 2005, pp. 494-504.

[2] R. C. Wilkins, H. Romm, T. C. Kao, A. A. Awa, M. A. Yoshida, G. K. Livingston et al., *Radiat* Res. 2008 May;169(5): pp. 551-60.

[3] A. S. Arachchige, J. Samarabandu, J. Knoll, W. Khan, P. K. Rogan, "An image processing algorithm for accurate extraction of the centerline from human metaphase chromosomes," *Proc. IEEE ICIP* 2010, pp. 3613-3616.

[4] P. Yanala, Lu T, El-Ghussein F, Zhao C, Medhi D, Wang Y-P, Knopp J, Knoll JH, Rogan PK. "Automated Detection of Metaphase Chromosomes for FISH and Routine Cytogenetics," *54th Annual ASHG Meeting, Toronto ON, 2004, p. 195*; T. Kobayashi, Shyu C-R, He L., Rogan PK, and Knoll J., "Content and classification based ranking algorithm for metaphase chromosome images," *IEEE Conference on Multimedia Imaging*, Taipei, 2004.

[5] C. Xu and J. L. Prince, "Gradient vector flow: A new external force for snakes," *Proc. IEEE Comp Soc Conf on Computer Vision and Pattern Recognition*, 1997.

[6] C. Li, J. Liu, M. D. Fox, "Segmentation of edge preserving gradient vector flow: An approach toward automatically initializing and splitting of snakes," Proc. IEEE Comp Soc Conf on Computer Vision and Pattern Recognition, 2005 pp. 162-7.

[7] X. Bai, L. J. Latecki, W. Y. Liu, "Skeleton pruning by contour partitioning with discrete curve evolution," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 29, no. 03, 2007.

[8] L. J. Latecki and R. Lakaamper, "Polygon evolution by vertex deletion," *Proc. of Second International Conference on Scale-Space Theories in Computer Vision*. Springer-Verlag London, UK, 1999, pp. 398-409.

[9] A. S. Arachchige, J. Samarabandu, J. Knoll, W. Khan, P. K. Rogan, "An Accurate Image Processing Algorithm for Detecting FISH Probe Locations Relative to Chromosome Landmarks on DAPI Stained Metaphase Chromosome Images," *Canadian Conf. on Computer & Robot Vision*, 2010, pp. 223 – 230, DOI: 10.1109/CRV.2010.36.

[10] L. Shapiro, G. Stockman, *Computer Vision,* 2002, pp. 69-73.

[11] R. L. Burden, J. D. Faires, *Numerical Analysis 7th edition,* pp. 370-378.

[12] M. Frigo, S. G. Johnson, "The design and implementation of FFTW3," *Proc. of the IEEE* 93 (2): pp. 216-231.

[13] L. H. Thomas, *Elliptic Problems in Linear Differential Equations over a Network,* 1949.

[14] G. Amdahl, “Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities, ” *AFIPS Conference Proceedings*, vol. 30, 1967, pp. 483-485.