Parallel Image Segmentation Using Reduction-Sweeps On Multicore Processors and GPUs

Renato Farias, Ricardo Farias, Ricardo Marroquim
Programa de Engenharia de Sistemas de Computacao
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
Email: {rmfarias,rfarias,marroquim}@cos.ufrj.br

Esteban Clua
Instituto de Computacao
Universidade Federal Fluminense
Niteroi, Brazil
Email: esteban@ic.uff.br

***Abstract*—In this paper we introduce the Reduction Sweep algorithm, a novel graph-based image segmentation algorithm that is designed for easy parallelization. It is based on a clustering approach focusing on local image characteristics. Each pixel is compared with its neighbors in an implicitly independent manner, and those deemed sufficiently similar according to a color criterion are joined. We achieve fast execution times while still maintaining the visual quality of the results. The algorithm is presented in four different implementations: sequential CPU, parallel CPU, GPU, and hybrid CPU-GPU. We compare the execution times of the four versions with each other and with other closely related image segmentation algorithms.**

***Keywords*-Image segmentation; computer vision; GPU programming, parallel programming;**

I. Introduction

Image segmentation is an image processing technique that divides an image into contiguous regions, or segments. All pixels in one of these segments are similar according to some criterion, such as color or intensity. By forming these clusters of similar elements, the process eliminates noise and facilitates posterior analysis.

Segmentation is a crucial step for many applications where it is important to discern objects or boundaries within the image. Some examples are: pattern recognition applied to medical imaging [1] [2]; object recognition for AI and robotics [3] [4]; and the labeling and categorization of content in images [5] [6].

There has been much research done on the subject, but many of the approaches are inherently sequential in nature, making them difficult to parallelize. There are some methods in which parallelization is achieved by dividing the image, executing the algorithm over each of the parts, then stitching the borders between results. This solution may be better than a purely sequential one in terms of performance, but it is still not ideal for many applications.

We present a graph-based image segmentation algorithm designed with easy and efficient parallelization in mind. Because it only uses local image characteristics, the Reduction Sweep algorithm can compare and join many different regions of the image independently. This leads to faster execution without sacrificing visual quality. No prior knowledge of the image is required, although tweaking the parameters may lead to better results for individual images.

This paper is organized as follows: Section II provides a brief overview of related research on image segmentation. Section III presents the Reduction Sweep algorithm and explains its steps in detail. Section IV describes the four different implementations of the algorithm. Section V presents the results while section VI draws some conclusions and provides some future directions.

II. Related work

In this section we will go over other efforts in the field of image segmentation. We focus on, but are not limited to, works that make use of graphs and that present GPU implementations, since they are more closely related to our approach.

Baatz and Schape [7] present an image segmentation algorithm that combines local and global optimization techniques with homogeneity definitions based on spectral and spatial information. Initially, each pixel represents its own image segment, and at each iteration a pair of segments are merged if they satisfy a chosen criterion. They proposed several strategies for the two important decisions of choosing what pair of segments to compare at each iteration and what criterion to use to compare them.

Felzenszwalb and Huttenlocher [8] propose a graph-based approach to image segmentation. Each pixel represents a node in the graph and adjacent pixels have non-directed edges connecting them. Theses edges possess weights representing the degree of dissimilarity between pixels according to some criterion. The list of all edges is sorted in order of increasing dissimilarity and are compared in this order. Note that, though the algorithm is greedy in its decisions, the segmentation produces good results because the threshold which must be surpassed for two segments to remain distinct is adapted based on the internal difference of each segment's own pixels.

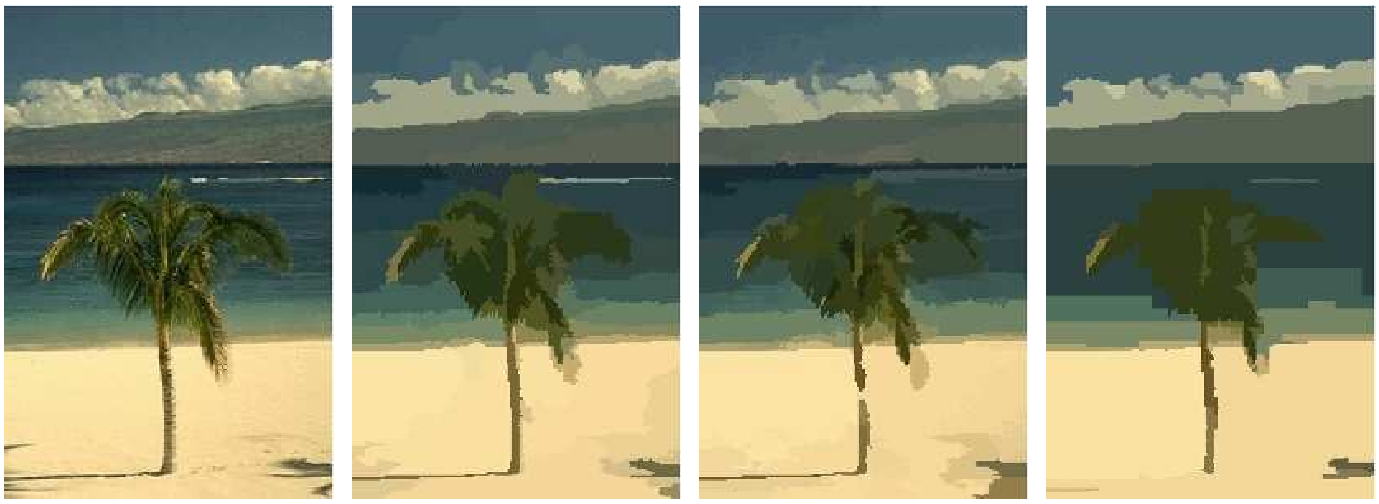

(a) Original (b) Baatz [7] (c) Felzenszwalb [8] (d) Reduction Sweep

Fig. 1. *Beach* comparisons.

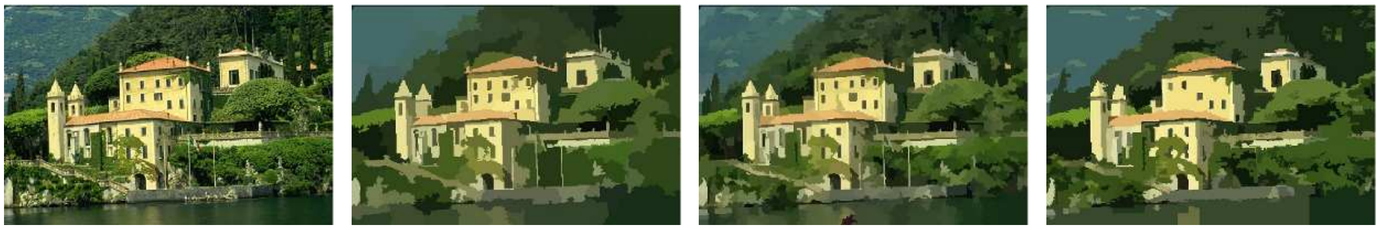

(a) Original (b) Baatz [7] (c) Felzenszwalb [8] (d) Reduction Sweep

Fig. 2. *Lakeside* comparisons.

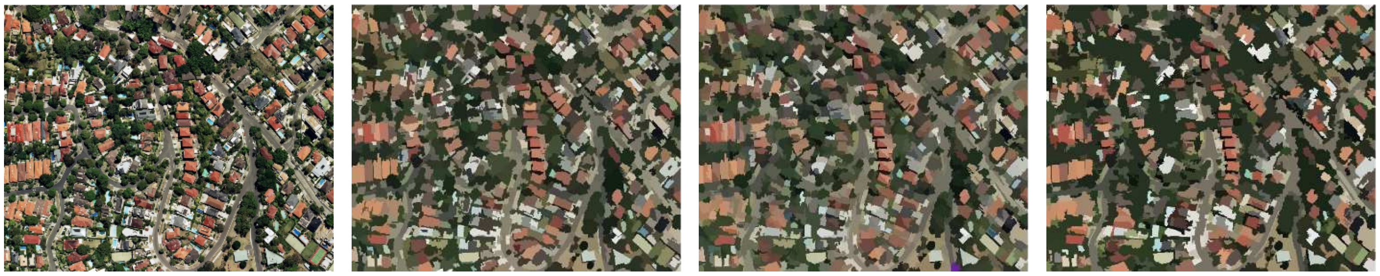

(a) Original (b) Baatz [7] (c) Felzenszwalb [8] (d) Reduction Sweep

Fig. 3. *Sidney* comparisons.

The active contour algorithm is the basis of the work of Zhiyu He and Falko Kuester [9]. Their approach consists in identifying segments borders using a reference curve which is repeatedly adjusted with a combination of external forces (calculated from the image) and internal forces (calculated from the contours themselves). They focused on a GPU implementation for calculating the external force using Gradient Vector Flow (GVF). GVF is able to capture concavities and is less sensitive to the initialization compared to the original active contour algorithm.

In the work of Alexey Abramov et al. [10], a real-time GPU image segmentation is presented that makes use of the Potts model from statistical mechanics. Their algorithm equates image segments to the equal-spin regions of a system of ferromagnets and finds them by calculating the equilibrium states of the system. It is a non-parametrized parallel algorithm aimed at segmentations where there is no prior knowledge of the images.

Brian Fulkerson and Stefano Soatto [11] present a GPU-based image segmentation implementation of the quick shift algorithm. They use a density probability function to connect each point to the next point of higher density probability, resulting in a tree. The edges of the tree are distances based on color and space (x,y) differences between the pixels. Lastly, the edges with distances greater than a specified threshold are broken; the resulting disconnected parts form the image segments.

Ying Zhuge et al. [12] use a fuzzy logic approach to segment volumetric data. The voxels of the image are given a fuzzy adjacency value which increases the closer they are spatially, as well as a value based on their affinity considering the whole image. These two attributes are used to lump together the voxels that best represent the global characteristics of the image.

Another graph-based approach is proposed by Lattari et al. [13] to identify and separate human skin. Their premise is that the colors of human skin inhabit a small part of the RGB spectrum. They build a graph with each pixel as a node and use a color-based heuristic to give weights to the edges. A minimum cut algorithm is applied to divide the graph in two disjoint parts: pixels belonging to human skin and pixels not belonging to human skin.

Jan Wassenberg et al. [14] introduce a parallel algorithm based on graphs with a cut heuristic and a minimum-spanning tree algorithm. The authors note that using a graph-based method increases the amount of data handled since each pixel is part of several edges. However, the advantage of the minimum-spanning tree is that it offers the opportunity for parallelization since it can be constructed from sub-trees.

In this paper, we compare our results with those of Baatz and Schape [7] and Felzenszwalb and Huttenlocher [8], since these approaches most closely resemble ours. Both Wassenberg et al [14] and Lattari et al [13] also employ graph-based approaches, however, the former limits the size of regions while the latter is directed at a specific context. Both diverge from our goal of having a general segmentation algorithm for delineating an arbitrary number of unbounded regions.

III. Algorithm

The criterion used to compare the segments and the order in which they are compared are two decisions that heavily impact the quality of the results [7]. In this section, we will explore our approach from a high-level perspective.

A. The disjoint set

Our algorithm utilizes directed graphs to represent image segments. The segments will be referred to as 'regions', while the terms 'node' and 'pixel' will be used interchangeably. Regions start with a single node. Adjacent regions are compared and merged if they are deemed to be sufficiently similar. The regions are combined into progressively larger regions, and the resulting segmentation is a disjoint set of regions.

Every node has the following attributes:

Size	How many pixels are part of the region to which this pixel belongs
Next	The index of the next node in the region
Color	The pixel's (r,g,b) color

Initially, all nodes constitute regions of *Size* 1 and *Next* equal to their own index. Only one node per region has its *Next* index pointing to itself; we call this node the **representative node** and it holds information about the region. By keeping only this node up-to-date with the region's size and color information, we are spared the cost of having to update every node in the region. All other nodes in the region can eventually reach the representative node by iterating over *Next*.

Algorithm 1 The Reduction Sweep algorithm.

```
1:  $iteration \Leftarrow 0$
2:  $start \Leftarrow 0$
3:  $offset \Leftarrow 2$
4:  while $start < width - 1$ or $start < height - 1$ do
5:      for $y \Leftarrow 0$ to $height - 1$ do
6:          for $x \Leftarrow start,\ start + offset,\ start + offset \times 2,\ \ldots,\ width - 2$ do
7:              Verify edge $(x, y)$, $(x + 1, y)$
8:      for $y \Leftarrow 0,\ offset \div 2,\ (offset \div 2) \times 2,\ \ldots,\ height - 2$ do
9:          for $x \Leftarrow start,\ start + offset,\ start + offset \times 2,\ \ldots,\ width - 2$ do
10:             $limit \Leftarrow (offset \div 2) - 1$
11:             if $y + limit > height$ then
12:                 $limit \Leftarrow height - y - 1$
13:             for $n \Leftarrow 0$ to $limit - 1$ do
14:                 Verify edge $(x, y + n)$, $(x + 1, y + n + 1)$
15:                 Verify edge $(x + 1, y + n)$, $(x, y + n + 1)$
16:     for $y \Leftarrow start,\ start + offset,\ start + offset \times 2,\ \ldots,\ height - 2$ do
17:         for $x \Leftarrow 0$ to $width - 1$ do
18:             Verify edge $(x, y)$, $(x, y + 1)$
19:     for $y \Leftarrow start,\ start + offset,\ start + offset \times 2,\ \ldots,\ height - 2$ do
20:         for $x \Leftarrow 0,\ offset,\ offset \times 2,\ \ldots,\ width - 2$ do
21:             $limit \Leftarrow offset - 1$
22:             if $x + limit > width$ then
23:                 $limit \Leftarrow width - x - 1$
24:             for $n \Leftarrow 0$ to $limit - 1$ do
25:                 Verify edge $(x + n, y)$, $(x + n + 1, y + 1)$
26:                 Verify edge $(x + n + 1, y)$, $(x + n, y + 1)$
27:     $iteration \Leftarrow iteration + 1$
28:     $start \Leftarrow 2^{iteration} - 1$
29:     $offset \Leftarrow offset \times 2$
```

The two most important operations of the disjoint set are:

Find	Receives the index *i* of a node belonging to a certain region **A** and returns the index *j* of the representative node of **A** by iterating over *Next* until it finds a node whose *Next* is equal to its own index. Also sets the *Next* of node *i* as *j* to speed up future searches.
Join	Receives the indexes of two nodes representing regions **A** and **B** and compares them. If the given criterion is satisfied a merge happens; in that case, assuming that **A**'s *Size* is smaller than **B**'s *Size*, this operation will make **A**'s representative node's *Next* point to **B**'s representative node, and update the *Color* and *Size* attributes in **B**'s representative node.

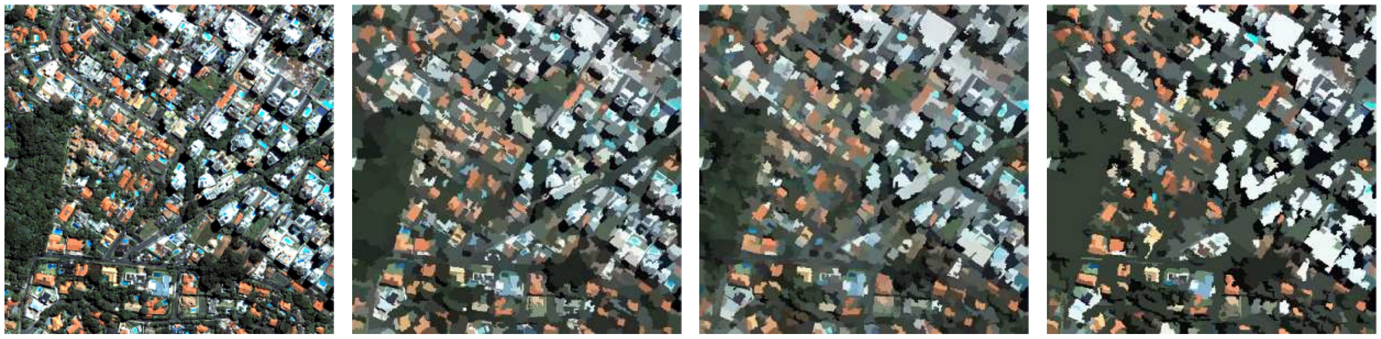

(a) Original (b) Baatz [7] (c) Felzenszwalb [8] (d) Reduction Sweep

Fig. 4. *Morumbi* comparisons.

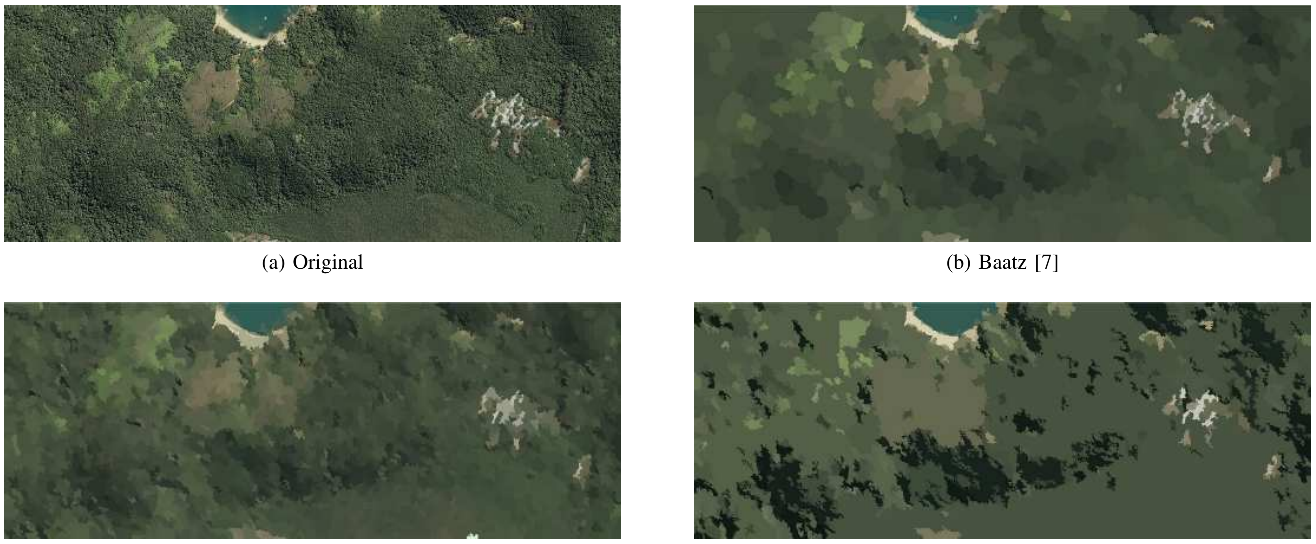

(a) Original (b) Baatz [7] (c) Felzenszwalb [8] (d) Reduction Sweep

Fig. 5. *Ilha Grande* comparisons.

The new *Color* is calculated as follows:

$$Color_{new} = \frac{Color_A \times Size_A + Color_B \times Size_B}{Size_A + Size_B}$$

Before segmentation occurs, we smooth the pixels of the set by applying a Gaussian blur. For a fair comparison, we used the same Gaussian blur as Felzenszwalb and Huttenlocher [8].

B. Merge criterion

We chose a color-based merge criterion for our tests, where the color of a region is the average of its pixels. We check if the Euclidean distance in RGB space between the color of two regions is below or equal to a certain threshold. So, given two regions **A** and **B**, each with a color vector **C** = (r,g,b), the comparison is:

$$C_B - C_A \leq t$$

$$\sqrt{(r_B - r_A)^2 + (g_B - g_A)^2 + (b_B - b_A)^2} \leq t$$

In practice, to avoid having to calculate a square root, we square both sides of the equation, giving us:

$$(r_B - r_A)^2 + (g_B - g_A)^2 + (b_B - b_A)^2 \leq t^2$$

C. Sweep Order

Our algorithm uses the concept of edges but it does so implicitly, without building a list as in Felzenszwalb and Huttenlocher [8]. An edge is simply a pair of adjacent pixels. This means that pixels on the corner of an image have three incident edges, pixels on the borders have five, and all other pixels have valence eight.

It is important to verify all edges to achieve the best possible segmentation. To verify an edge means to take both pixels that comprise it, use the *Find* operation to retrieve their representative nodes, and then use the *Join* operation to compare them using the criterion explored in the previous sub-section. Since one node stores a region's up-to-date information and any verification will read from and possibly (in the case of a merge) write to this node, only one of a region's incident edges can be verified per iteration. The worst case for parallelism occurs when each verification results in a merge, reducing the number of regions that can be verified in the next iteration. Because we have no prior knowledge of the image, we assume the worst case always happens to guarantee safe, independent

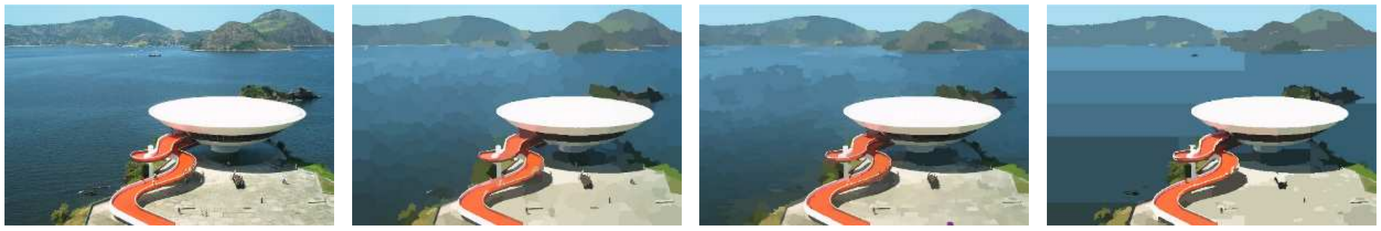

(a) Original (b) Baatz [7] (c) Felzenszwalb [8] (d) Reduction Sweep

Fig. 6. *Museu* comparisons.

verifications.

Changes to the order in which the edges are verified will produce different segmentations. We developed three sweep orders: Horizontal First, which verifies all horizontal edges before the vertical and diagonal edges; Vertical First, which verifies all vertical edges before the horizontal and diagonal edges; and a Mixed order, which alternatively verifies horizontal, vertical, and diagonal edges. We limit our descriptions and results to the Mixed sweep order, since it is the one that gives the best visual results.

Figure Fig. 7 illustrates the Mixed sweep order on a small image where regions are represented by red rectangles. As can be seen, the number of edges incident to different regions decreases exponentially with each iteration, similar to what happens in parallel programming's reduction technique. Note that it is possible for both nodes of an edge to belong to the same region due to a previous merge, and that the edges incident to the same region or pair of regions must be executed by the same thread to prevent concurrency issues.

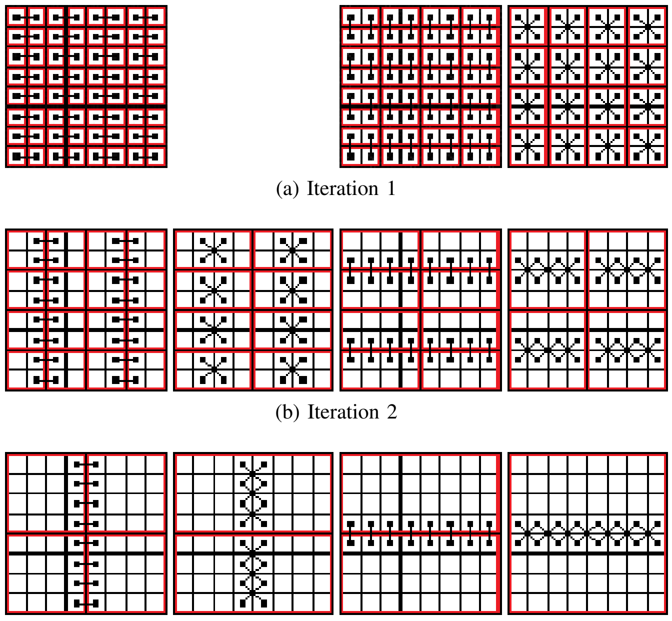
D. Post-process

Our algorithm also possesses a last step meant to eliminate regions that are too small. The sweep order used is the same, but the edge verifications do not use the criterion described in section III-B; instead, two regions are merged if at least one of their *Size* attributes are under the minimum allowed size. The post-processing step is inherently sequential because we can make no assumptions about which edges we can verify independently.

IV. Implementation

We developed four implementations of the Reduction Sweep algorithm. The first is a sequential CPU implementation which acted as our proof-of-concept. The second is a parallel CPU implementation which uses the pthreads library. The third implementation uses CUDA 4 to execute the algorithm on the GPU.

By comparing individual iteration execution times from the second and third implementations, we observed that while initial iterations tend to be faster on the GPU, later ones are faster on the CPU. The fourth implementation is a hybrid of the second and third implementations that attempts to take advantage of this; it performs only a specified number of sweep iterations on the GPU, then returns to the CPU and finishes the process.


(a) Iteration 1

(b) Iteration 2

(c) Iteration 3

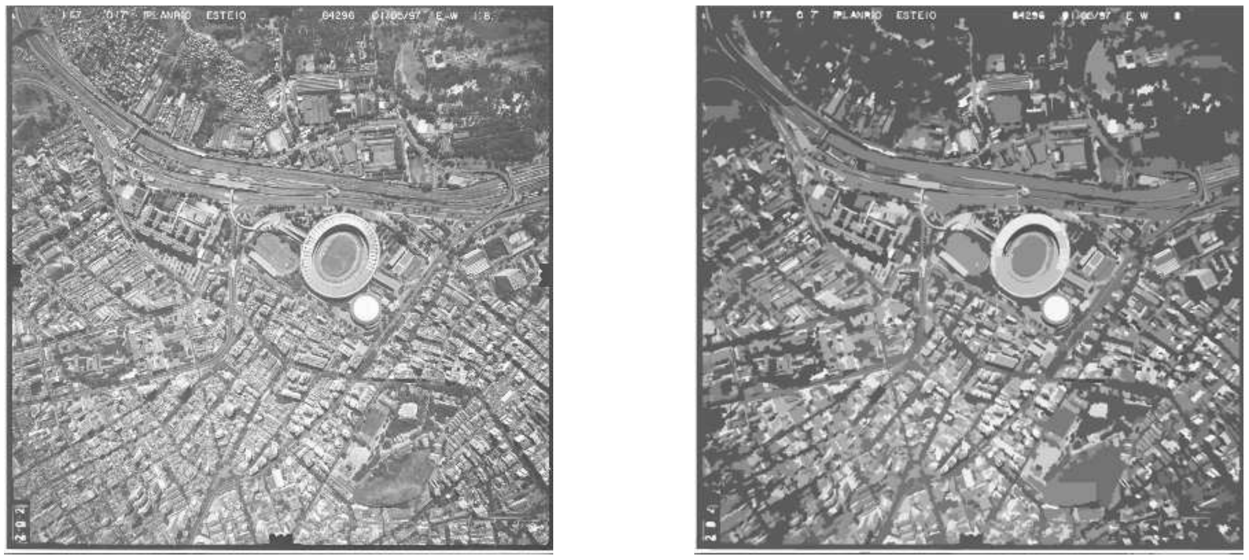
Fig. 7. Illustrated example of the Reduction Sweeps performed on an 8x8 image. Each iteration performs horizontal, diagonal, vertical, and (again) diagonal verifications. The exception is Iteration 1, which skips the first diagonal sweep due to the algorithm's formulation.

V. Results

Table I shows the average total execution times for all algorithms tested in this paper. Table II shows the average execution times for the Reduction Sweeps of each implementation, ignoring various overheads. Each algorithm was run 100 times on each image. We used the 64-bit distribution of Ubuntu 12.04, on a computer with 16GB of RAM, an Intel Core i7 CPU, and an Nvidia GeForce GTX 470. We manually selected the parameters for each algorithm and image to give the best visual results; they are shown in Table VI.

The sequential implementation of our Reduction Sweep algorithm is already 8 to 14 times faster than the algorithm it is based on [8], and 24 to 62 times faster than Baatz and Schape [7], depending on the image. We also observed that our total execution time tends to increase almost linearly with the image area, and the execution time of the actual Reduction Sweeps increases practically sub-linearly with image area (Table V).

Table IV shows the speed-ups that the parallel implemen-



(a) Original (b) Reduction Sweep

Fig. 8. *Maracana* comparisons.

	Beach 200x300	**Lakeside** 481x321	**Sidney** 1024x828	**Morumbi** 1000x1000	**Museum** 1280x857	**Ilha Grande** 2177x833	**Maracana** 2800x2800	**Maracana** 5000x5000
Baatz [7]	344.35	926.41	5603.78	6949.96	11736.57	27193.53	90846.92	328283.91
Felzenszwalb [8]	113.79	294.30	1898.82	2276.31	2473.24	4596.48	22117.43	75058.91
RS-Sequential	13.83	32.81	181.59	223.76	206.39	436.63	1714.78	5369.67
RS-Threads	14.49	30.35	121.15	145.25	146.13	273.71	1110.71	3722.75
RS-GPU	79.41	98.30	206.51	234.06	242.60	382.56	1400.60	4457.67
RS-Hybrid	82.98	101.19	211.52	257.00	254.04	378.01	1402.69	4565.99

TABLE I
TOTAL EXECUTION TIMES IN MILLISECONDS. ALL STEPS ARE TAKEN INTO ACCOUNT, INCLUDING OVERHEADS SUCH AS THE INITIAL SMOOTHING, MEMORY ALLOCATION, MEMORY TRANSFER (GPU), AND POST-PROCESSING.

	Beach 200x300	**Lakeside** 481x321	**Sidney** 1024x828	**Morumbi** 1000x1000	**Museum** 1280x857	**Ilha Grande** 2177x833	**Maracana** 2800x2800	**Maracana** 5000x5000
Sequential	5.56	15.72	95.48	118.89	98.68	247.08	862.46	2425.36
Threads	4.76	9.12	32.61	40.67	35.67	79.78	264.41	765.46
GPU	4.86	12.79	42.41	48.65	50.72	90.05	309.93	870.68
Hybrid	6.41	10.82	43.41	68.97	59.23	84.30	287.70	989.36

TABLE II
EXECUTION TIMES (IN MILLISECONDS) OF ONLY THE REDUCTION SWEEPS STEP, NOT INCLUDING OVERHEADS SUCH AS THE INITIAL SMOOTHING, MEMORY ALLOCATION, MEMORY TRANSFER (GPU), AND POST-PROCESSING.

tations achieved relative to the sequential implementation. We built our algorithm with parallelism in mind, so we were surprised to find that the parallel implementations achieved only slight speed-ups compared to the sequential implementation. In fact, in many cases, the parallel implementations lost to the sequential implementation. This can be attributed, in part, to the steps of the process that weren't optimized (the initial smoothing) or that are inherently sequential (the post-process). The two implementations that make use of the GPU were also hindered by the random nature of memory accesses during the Sweeps, which leads to divergences among the threads. When we consider only the Reduction Sweeps (Table II), the speed-ups achieved are 2 to 4 times higher.

The Hybrid implementation did not improve on the GPU implementation in any significant way, doing worse on half of the images; it inherits the GPU overheads, and its CPU iterations are slightly slower, possibly due to reduced cache coherence after copying the data back to RAM.

According to the Berkeley Computer Vision Group's image segmentation benchmark [15], the Reduction Sweep algorithm achieves an F-measure of 0.558. This is slightly lower than the F-measure of 0.58 of Felzenszwalb and Huttenlocher's graph-based algorithm [8], but with significant speed-ups.

	Smoothing[1]	Allocation[2]	cudaMemcpy (CPU $\rightarrow$ GPU)	Sweeps[3] (GPU)	cudaMemcpy (GPU $\rightarrow$ CPU)	Sweeps[3] (CPU)	Post-process[4]	Extraction[5]
Sequential	49.76	2.69	–	–	–	98.68(11)	48.02	5.32
Threads	49.99	2.68	–	–	–	35.67(11)	50.60	5.31
GPU	47.96	60.10	5.35	50.72(11)	5.30	–	48.91	22.62
Hybrid	49.36	59.57	5.32	20.71(3)	5.26	38.52(8)	52.27	21.45
Sequential	1586.49	56.35	–	–	–	2425.36	1150.39	127.47
Threads	1592.74	55.87	–	–	–	765.46	1157.40	127.64
GPU	1598.68	114.49	92.19	870.68	92.72	–	1171.20	486.68
Hybrid	1592.87	114.12	91.92	713.59	92.33	275.77	1169.48	485.33

TABLE III
EXECUTION TIMES (IN MILLISECONDS) OF ALL STEPS ON THE *Museum* (TOP) AND *Maracana5000* (BOTTOM) IMAGES. [1]APPLICATION OF A GAUSSIAN BLUR. [2]INCLUDES BOTH MALLOC AND CUDAMALLOC, WHEN APPLICABLE. [3]THE NUMBERS IN PARENTHESIS REFER TO HOW MANY ITERATIONS WERE PERFORMED ON THE CPU OR GPU. [4]DESCRIBED IN SECTION III-D. [5]REFERS TO EXTRACTING THE FINAL RGB CHANNELS FROM THE DISJOINT SET.

	Beach	Lakeside	Sidney	Morumbi	Museum	Ilha Grande	Maracana	Maracana
	200x300	481x321	1024x828	1000x1000	1280x857	2177x833	2800x2800	5000x5000
Threads	0.95 (1.17)	1.08 (1.72)	1.50 (2.93)	1.54 (2.92)	1.41 (2.77)	1.60 (3.10)	1.54 (3.26)	1.44 (3.17)
GPU	0.17 (1.14)	0.33 (1.23)	0.88 (2.25)	0.96 (2.44)	0.85 (1.95)	1.14 (2.74)	1.22 (2.78)	1.21 (2.79)
Hybrid	0.17 (0.87)	0.32 (1.45)	0.86 (2.20)	0.87 (1.72)	0.81 (1.67)	1.16 (2.93)	1.22 (3.00)	1.18 (2.45)

TABLE IV
SPEED-UPS OF THE DIFFERENT IMPLEMENTATIONS COMPARED TO THE SEQUENTIAL CPU IMPLEMENTATION. THE VALUES IN PARENTHESIS ARE THE SPEED-UPS CONSIDERING ONLY THE REDUCTION SWEEPS, AS IN TABLE II.

	Maracana350	Maracana700	Maracana1400	Maracana2800	Maracana5600	Maracana11200
Area	122500	490000 (4)	1960000 (4)	7840000 (4)	31360000 (4)	125440000 (4)
Total	26.97	70.13 (2.60)	284.85 (4.06)	1117.81 (3.92)	4644.24 (4.15)	19896.39 (4.28)
Reduction Sweeps	8.43	18.48 (2.19)	76.44 (4.14)	263.29 (3.45)	933.14 (3.54)	3353.43 (3.59)

TABLE V
EXECUTION TIMES (IN MILLISECONDS) ON THE SAME IMAGE AT DIFFERENT SIZES, USING THE PARALLEL CPU IMPLEMENTATION. INCREASES COMPARED TO THE PREVIOUS IMAGE ARE SHOWN IN PARENTHESIS.

	Beach	Lakeside	Sidney	Morumbi	Museum	Ilha Grande	Maracana
Baatz [7]	0.3/30/20	0.3/15/40	0.3/10/50	0.3/10/50	0.3/10/50	0.3/10/50	0.3/10/50
Felzenszwalb [8]	0.5/50/50	0.5/20/50	0.5/50/200	0.5/50/200	0.5/50/200	0.5/50/200	0.5/50/200
Reduction Sweep	0.5/30/50	0.5/30/50	0.5/40/200	0.5/40/200	0.5/25/100	0.5/35/200	0.5/40/200

TABLE VI
EXECUTION PARAMETERS: SMOOTHING FACTOR (σ), THRESHOLD, AND MINIMUM SEGMENT SIZE.

VI. CONCLUSIONS

In this work, we presented the Reduction Sweep algorithm, a novel graph-based parallel segmentation method. We described four implementations of the algorithm, and how each compares to closely related classic algorithms [7] [8] in terms of execution time and visual quality. We also explained the significance of the speed-ups (and in some cases, slow-downs) achieved.

We discovered that the actual gain from parallelism was lower than expected. This shows that, despite the independence among edge verifications, the un-optimized or sequential steps of the segmentation process, the necessity of synchronizing threads at each iteration, and the random nature of memory accesses when finding representative pixels, had a significant impact, especially on the GPU.

There are several possibilities for future work. One would be to modify the Hybrid implementation to keep the data in the CPU's memory (zero-copy), eliminating the memory allocation and copy overheads. Another possibility would be to devise new sweep orders that mesh better with the GPU architecture, or new merge criteria that aren't based on color. Lastly, expanding this segmentation from 2D images to 3D volumes would be another possibility.

ACKNOWLEDGMENT

Renato Farias is thankful for the scholarship granted by CAPES. Esteban Clua is thankful to Nvidia and FAPERJ.

REFERENCES

[1] F. Ma, M. Bajger, J. P. Slavotinek, and M. J. Bottema, "Two graph theory based methods for identifying the pectoral muscle in mammograms," *Pattern Recogn.*, vol. 40, no. 9, pp. 2592–2602, Sep. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2006.12.011

[2] L. Xu, B. Stojkovic, H. Ding, Q. Song, X. Wu, M. Sonka, and J. Xu, “Faster segmentation algorithm for optical coherence tomography images with guaranteed smoothness,” in *Proceedings of the Second international conference on Machine learning in medical imaging*, ser. MLMI’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 308–316. [Online]. Available: http://dl.acm.org/citation.cfm?id=2046063.2046101

[3] Z. Kira, “Inter-robot transfer learning for perceptual classification,” in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, ser. AAMAS ’10. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 13–20. [Online]. Available: http://dl.acm.org/citation.cfm?id=1838206.1838209

[4] L. Markham, S. C. Melton, and Z. Dodds, “Robot control via region-based 3d reconstruction,” in *Proceedings of the 10th IASTED International Conference on Intelligent Systems and Control*, ser. ISC ’07. Anaheim, CA, USA: ACTA Press, 2007, pp. 29–34. [Online]. Available: http://dl.acm.org/citation.cfm?id=1647449.1647455

[5] N. D. Anh, P. T. Bao, B. N. Nam, and N. H. Hoang, “A new cbir system using sift combined with neural network and graph-based segmentation,” in *Proceedings of the Second international conference on Intelligent information and database systems: Part I*, ser. ACIIDS’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 294–301. [Online]. Available: http://dl.acm.org/citation.cfm?id=1894753.1894789

[6] K. Ren and J. Calic, “Freeeye: interactive intuitive interface for large-scale image browsing,” in *Proceedings of the 17th ACM international conference on Multimedia*, ser. MM ’09. New York, NY, USA: ACM, 2009, pp. 757–760. [Online]. Available: http://doi.acm.org/10.1145/1631272.1631406

[7] M. Baatz and A. Schape, “Multiresolution segmentation: an optimization approach for high quality multi-scale image segmentation,” *Journal of Photogrammetry and Remote Sensing*, vol. 58, no. 3–4, pp. 12–23, 2000.

[8] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient graph-based image segmentation,” *International Journal of Computer Vision*, vol. 59, no. 2, 2004.

[9] Z. He and F. Kuester, “Gpu-based active contour segmentation using gradient vector flow,” *International Symposium on Visual Computing*, 2006.

[10] A. Abramov, T. Kulvicius, F. Worgotter, and B. Dellen, “Real-time image segmentation on a gpu,” *Facing the Multicore-Challange*, 2010.

[11] B. Fulkerson and S. Soatto, “Really quick shift: Image segmentation on a gpu,” *Workshop on Computer Vision using GPUs*, 2010.

[12] Y. Zhuge, Y. Cao, J. K. Udupa, and R. W. Miller, “Gpu accelerated fuzzy connected image segmentation by using cuda,” *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2009.

[13] L. Lattari, A. Montenegro, A. Conci, E. Clua, V. Mota, M. B. Vieira, and G. Lizarraga, “Using graph cuts in gpus for color based human skin segmentation,” *Integrated Computer-Aided Enginnering*, vol. 18, no. 41–59, 2011.

[14] J. Wessenberg, W. Middelmann, and P. Sanders, “An efficient parallel algorithm for graph-based image segmentation,” *Computer Analysis of Images and Patterns*, 2009.

[15] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2010.161