Parallel Cosegmentation via Submodular Optimization on Anisotropic Diffusion

Dinesh Majeti*, Aditya Prakash*, S. Balasubramanian, PK Baruah
Sri Sathya Sai Institute of Higher Learning, Prasanthi Nilayam,India.
{dinesh.majeti,words.adi.worth}@gmail.com, {sbalasubramanian,pkbaruah}@sssihl.edu.in

***Abstract*—With large number of related images being used for applications such as MR spectroscopy imaging, Object of interest 3D modelling and photo collages, the need of the hour is to accelerate image cosegmentation algorithms. Cosegmentation refers to the process of segmenting common regions from multiple related images. A novel distributed algorithm, CoSand [1], for cosegmentation of large-scale image collections was proposed. CoSand involves preprocessing, solving multiple systems of linear equations and clustering for every image, thereby making it computationally intensive. In this work various approaches to parallelize the MATLAB code for this problem were considered. The most time consuming part of code which is solving multiple systems of linear equations was offloaded to the GPU. Other operations have been performed on multicores. Further, unlike other parallelized image processing algorithms that require a single image data transfer to GPU, CoSand requires a set of image data (500 in our case) accounting for a substantial data transfer overhead. Even in this scenario, the parallel implementation on GPU has achieved a significant performance gain which is comparable to the multicore implementation of the entire application.**

***Index Terms*—Cosegmentation, submodular function, anisotropic diffusion, lazy execution.**

I. Introduction

General purpose GPU computing languages like CUDA and OpenCL provide interface to tap the power of the massive multi-threading of GPUs and gain performance. Even without any experience with CUDA, programmers can benefit from GPUs by using accelerated linear algebra libraries such as CULA [2] and MAGMA [3]. Products like MATLAB's Parallel Computing Toolbox and AccelerEyes Jacket [4] provide performance boost to MATLAB based applications. Depending on the scope of parallelization of a given application these softwares provide a gamut of tools for high performance computing. Image cosegmentation is one such application that lends itself to parallelization.

*Student Author

Cosegmentation of images is the identification of similar objects in multiple images. Cosegmentation has applications in interactive graphics, video segmentation, measuring image similarity, medical imaging and building 3D models from photo collections. Co-Segmentation via anisotropic diffusion (CoSand) [1], is a novel and highly scalable cosegmentation algorithm. CoSand is readily applicable to large scale image collection with high variability. For details readers are referred to [1].

II. Related Work

There are several recent articles that target the general task of cosegmentation of multiple images [5]–[7]. Particularly, [5] and [7] cast the problem of cosegmentation as Markov Random Field based segmentation of the image pair with a regularized difference of the two histograms. Vincente et al. [8] proposed the idea of object cosegmentation based on a similarity measure while Joulin et al. [9] combined tools like normalized cuts with kernel methods commonly used in object recognition for cosegmentation.

Despite the availability of lot of work on cosegmentation, literature on implementation of cosegmentation on GPU is scarce. Collins et al. [10] recast the cosegmentation problem using Random Walker segmentation. They also proposed GPU based optimizations by expressing the operations leading to cosegmentation in terms of linear algebra operations.

In this work parallelization of CoSand using MATLAB PCT, AccelerEyes Jacket, and CULA is investigated. Section III explains the serial CoSand, section IV discusses the various approaches taken to parallelize CoSand and section V presents the various results obtained.

III. CoSand - The Algorithm

The schematic diagram of CoSand is given in Figure 1. The input to the algorithm is an image set $\mathcal{I}$ and the number of segments K. The main objective of the

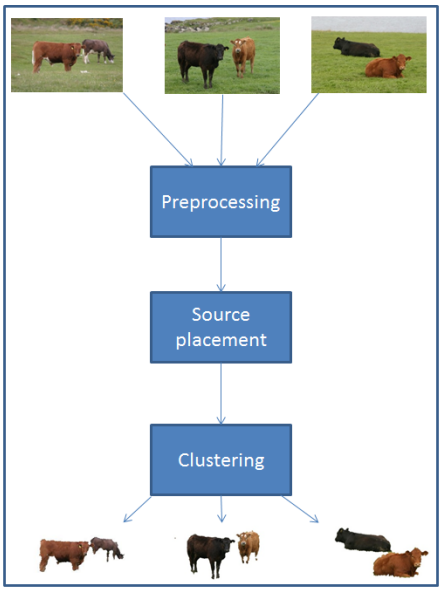

Fig. 1: CoSand algorithm

algorithm is to place the K segment centres in order to maximize the segmentation confidence at each pixel in an image while enforcing inter-image similarity between the chosen segments across images in the image set. CoSand consists of three stages:

I *Preprocessing Stage*:
  (a) The intra-image graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i, \mathcal{D}_i)$, where the vertex set $\mathcal{V}_i$ is the set of extracted superpixels and the edge set $\mathcal{E}_i$ is the set of all pairs of adjacent superpixels, is constructed. Gaussian similarity is used to compute the diffusivity $\mathcal{D}_i$ on the features of superpixels. The 3-D CIE Lab color and 4-D texture features are extracted in each superpixel.
  (b) Agglomerative clustering is run on $\mathcal{G}_i$ to find the evaluation points $\mathcal{L}_i$.

II *Source placement Stage*:
  (a) The gain at each of the evaluation points for every image is obtained by solving a system of linear equations that requires matrix inversion, a computationally intensive operation.
  (b) Subsequently, belief propagation is performed to obtain the constant factor approximation to the algorithm.

III *Clustering Stage*: This stage uses the source points from the previous stage to obtain the cosegmentation by clustering the superpixels that share the same source point as the most probable destination in each image.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
script_cosegment	1	599.986 s	0.473 s	
run_coseg_naive_greedy	1	414.561 s	0.507 s	
get_potential_mult_lineq_inc_inv	400	374.392 s	374.358 s	
run_superpixel	100	83.638 s	12.919 s	
get_sp_turbo	100	67.684 s	0.004 s	
superpixels	100	67.658 s	0.347 s	
evolve_height_function_N	300	56.427 s	2.562 s	
...coseg_naive_greedy>run_bp_max_sources	4	39.642 s	19.007 s	

Fig. 2: MATLAB profiler result

As it is evident from the algorithm, stages I, II(a) and III can be done independently for every image. Only the belief propagation stage (II(b)) requires data from all the images. Thus all the stages other than II(b) lend themselves to parallelization.

IV. Approaches taken to parallelize the application

Using MATLAB profiler, it was observed that step II(a) consumed 62% of the time of sequential CoSand as seen in Figure 2 (*get_potential_mult_lineq_inc_inv* is the procedure performing II(a)). This step involves solving systems of linear equations at a set of evaluation points for each image independent of other images. We discuss various approaches to parallelization of II(a) below:

A. Thrust through nvmex

Thrust is a C++ template library for CUDA. It mimics Standard template library. It provides programmers with various containers and algorithms.

Step II(a) was ported on to GPU using a vector of structures, one structure per image, containing the image data including Laplacian matrix, set of evaluation points and the corresponding weights using Thrust library. However, Thrust compilation was significantly slow. For example, it consumed 30 minutes for 10 images under nvcc -O3 option. Even worse than this was the no response scenario when O3 option was not used. This could probably be because of a dominant proportion of compilation being spent in *cicc*, a compiler component, with 2GB of system memory being used[1].

In addition to a long compilation time, Thrust, unlike MATLAB, does not provide efficient data structures and interfaces viz., vector of vectors to encapsulate image

[1]communication with Cliff Woolley, NVIDIA, also asserts this observation

data to perform operations like removal and updation of specific rows of Laplacian matrix as required by CoSand. Hence parallelization of CoSand is infeasible using Thrust.

B. CULA

Solving a system of linear equations, matrix inversion and multiplication can be ported to GPU using CULA library. CULA functions can be called either explicitly using MEX interface or implicitly from MATLAB. For matrices of size about 1024x1024, the overhead of using CULA from MATLAB undermines the performance enhancement (see Table I). For larger matrices (>2048x2048), performance gain is visible as shown in Table I. But CoSand uses matrices of dimensions 1024x1024.

Matrix Size	MATLAB	CULA	JACKET
1024x1024	0.0809	0.0815	0.0371
2048x2048	0.489	0.277	0.109
4096x4096	3.013	1.25	0.436

TABLE I: Time taken for inverse operation in seconds

C. MATLAB'S PCT

MATLAB's Parallel Computing Toolbox (PCT) enables users to harness the power of a multicore computer, GPU, cluster, grid or cloud to solve computational and data intensive problems. It's gpuArray functionality allows for a certain set of supported MATLAB functions to be ported on to the GPU.

Also the Parallel for-loops (parfor) helps to run loop iterations in parallel. The overhead for starting and shutting down the worker pool is about 10-15 seconds and 5 seconds [11] respectively. There are also communication overheads between the MATLAB session and the local worker processes. So, this approach is worth considering only if the loop in the serial version takes more than 30 seconds. Thus using *parfor* to port independent computations for each image on multicores would not give good performance if number of images is small. As CoSand involves a large number of images, *parfor* is suitable for parallelization.

D. AccelerEyes Jacket

Jacket accelerates MATLAB code on GPUs. It provides support for a large number of MATLAB functions. It performs automatic translation of MATLAB code to high performance primitives suited for GPUs.

Jacket employs a compile on-the-fly approach referred to as lazy execution. Under lazy execution Jacket does not dispatch kernels for every Jacket call. Instead it compiles and invokes kernel only when a result from a Jacket call is required by a non-Jacket computation.

As can be seen in Table I Jacket outperforms MATLAB and CULA in case of matrix inversion. Hence, Jacket is an appropriate tool for CoSand parallelization.

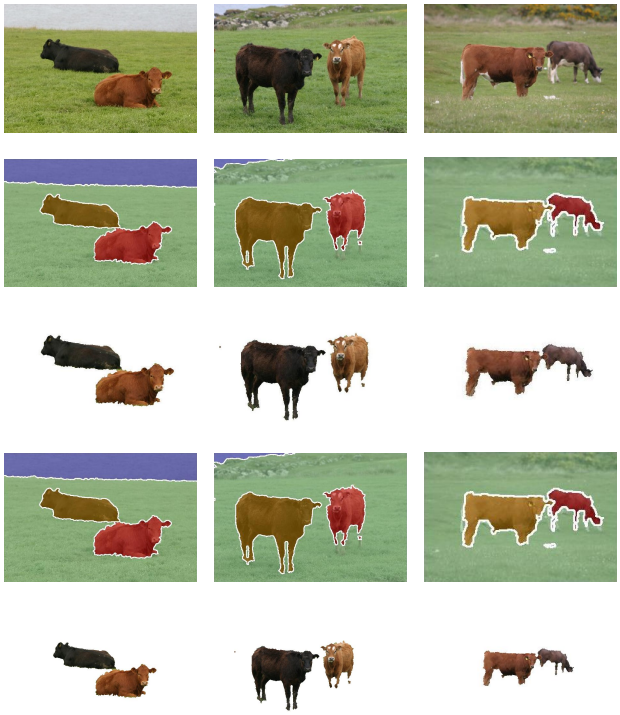

Fig. 3: Cosegmentation result for MSRC *Cow* dataset - *row 1*: input images, *row 2*: serial color-coded cosegmentation, *row 3*: segments from the image, *row 4*: parallel color-coded segmentation, *row 5*: segments - parallel result

V. Results

Lion, Gorilla, Ferret and Butterfly synsets of ImageNet dataset [12] and *Cow* images from the MSRC dataset [13] were used in the experiment. The number of superpixels was limited to 1000 and the images were resized to a maximum of 300 x 300. The value of K i.e., the number of sources was fixed to 4. All the images are color images.

Correctness of parallel CoSand was verified by visual inspection as shown in Figure 4. The performance of parallel and serial implementation was tested in following environment:

(a) Intel(R) Xeon(R) CPU X5650 2.67GHz processor with 12 cores and 24GB RAM.

Fig. 4: Cosegmentation result for Imagenet *Butterfly* synset - *row 1*: input images, *row 2*: segments from the image, *row 3*: segments - parallel result

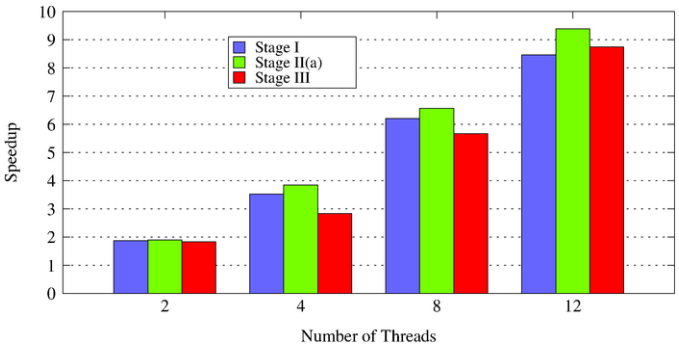

Fig. 5: Performance of Stages I, II(a) and III

(b) Tesla S2050 device with CUDA Driver and Runtime versions 4.2.

The total time spent in each of the stages viz., I, II and III were recorded. Figure 5 shows the speedup achieved for each stage on multicores using MATLAB PCT. The speedup increases with the number of threads upto 12 threads. Using MATLAB on SMP type systems such as PSC Blacklight [14], exploiting more cores may further increase the speedup. With respect to GPU, as envisaged, the performance diminishes as the number of images increase (see Fig. 6) because of increased data transfer overhead. The experimental results also showed that using GPU and multicore together for stage II(a) reduces the performance as shown in Figure 7 (see second bar). This clearly illustrates the fact that the code becomes serialized when multiple threads try to run code on the same GPU device. This problem can be resolved in a multi-GPU node if, each of the multicore thread can be bound to a unique GPU.

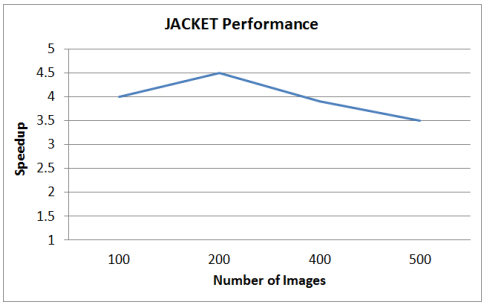

Fig. 6: Performance of Jacket with increase in number of images

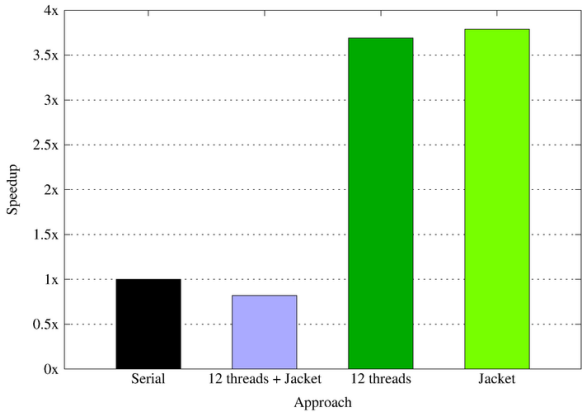

Fig. 7: Speedup using various approaches for stage II(a)

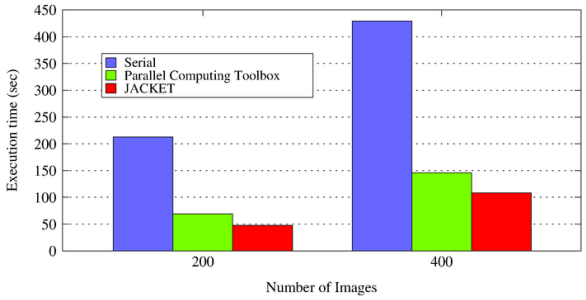

Fig. 8: Comparison of Serial, PCT and JACKET

We also compare the result for stage II(a) using MATLAB PCT gpuArray and AccelerEyes Jacket with serial code for 500 images in Figure 8. This emphasizes that Jacket outperforms MATLAB's PCT. This can be attributed to lazy execution method adopted in Jacket.

It is to be noted that CoSand uses two different approaches for stage II(a) namely, *incremental inverse* and *LU decomposition*. In serial execution of CoSand, *incremental inverse* is faster than *LU decomposition*. But on multicores, the speedup for *incremental inverse* is lesser than *LU decomposition* with increase in number of threads (see Fig.9). This behaviour is observed because

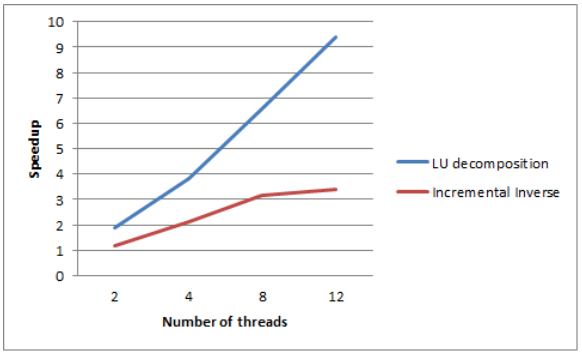

Fig. 9: Comparison between incremental inverse and LU decomposition algorithms

the *incremental inverse* method, unlike *LU decomposition* involves matrix inversion and multiplication operations. These operations are implicitly multi-threaded in MATLAB. But parallelization using multicores causes individual threads to perform matrix inversions and multiplication. This leads to decline in performance.

The overall performance gain achieved using the different approaches of Jacket on GPU and MATLAB PCT on multicores leveraging the *incremental inverse* method are comparable and are about 4x. Employing multicores for parallelization using MATLAB PCT shows a speedup of about 9x for each stage namely, stage I, II(a) using LU method and III.

VI. Conclusion and Future Work

Various approaches to parallelize MATLAB code for Distributed Cosegmentation via Submodular Optimization on Anisotropic Diffusion were considered. The approaches include use of Thrust, Acceleryes Jacket, MATLAB's PCT and GPU accelerated linear algebra library CULA. While Thrust is infeasible for performance gain in CoSand, CULA does not give significant speedup.

MATLAB PCT for multicore and AccelerEyes Jacket for GPU are possible solutions to parallelism. Jacket outperforms PCT on GPU. The overall performance gain achieved using the different approaches of Jacket on GPU and MATLAB PCT on multicores leveraging the *incremental inverse* method are comparable. Employing multicores for parallelization using MATLAB PCT shows a speedup of about 9x for each stage namely, stage I, II(a) using LU method and III.

For future work we propose a complete implementation of the given cosegmentation algorithm in CUDA using the OpenCV library. Also the change in performance with varying number of sources and superpixels can be examined.

Acknowledgment

This work is dedicated to our Divine Founder Chancellor Bhagawan Sri Sathya Sai Baba who is the source of our inspiration and motivation.

We would like to thank Gunhee Kim for generously making a MATLAB implementation of Distributed Cosegmentation via Submodular Optimization on Anisotropic Diffusion available online. [2]

This work was partially supported by a NVIDIA grant under Professor Partnership program, a Defence Research and Development Organization (DRDO) grant under Extramural Research and Intellectual Property rights and the Extreme Science and Engineering Discovery Environment (XSEDE) of National Science Foundation under grant number OCI-1053575.

References

[1] Gunhee Kim, Eric P. Xing, Li Fei-Fei, and Takeo Kanade. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *13th International Conference on Computer Vision (ICCV 2011)*, 2011.

[2] *CULA Reference Manual*.

[3] developer.nvidia.com/cuda/magma.

[4] www.accelereyes.com.

[5] Mukherjee et. al. Half-integrality based algorithms for cosegmentation of images. In *CVPR'09*, pages 2028–2035, 2009.

[6] Vicente et. al. Cosegmentation revisited: models and optimization. In *Proceedings of the 11th European conference on Computer vision: Part II*, ECCV'10, pages 465–479, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] Rother et. al. Cosegmentation of image pairs by histogram matching. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 993–1000, Washington, DC, USA, 2006. IEEE Computer Society.

[8] Vicente et. al. Object cosegmentation. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 2217–2224, Washington, DC, USA, 2011. IEEE Computer Society.

[9] A. Joulin, F. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[10] Collins et. al. Random walks based multi-image segmentation: Quasiconvexity results and gpu-based solutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence, Rhode Island, June 2012.

[11] *Parallel Computing Toolbox User's Guide R2012b*.

[12] Deng et. al. Imagenet: A large-scale hierarchical image database. In *CVPR'09*, pages 248–255, 2009.

[13] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2*, ICCV '05, pages 1800–1807, Washington, DC, USA, 2005. IEEE Computer Society.

[14] www.psc.edu/index.php/computing-resources/blacklight.

[2] http://www.cs.cmu.edu/∼gunhee/r_seg_submod.html