Highly Efficient Reconfigurable Parallel Graph Cuts for Embedded Vision

Antonis Nikitakis
Technical University of Crete
School of Electronic and Computer Engineering
Kounoupidiana, Chania, Crete, GR73100, Greece
anikita@mhl.tuc.gr

Ioannis Papaefstathiou
Synelixis Solutions Ltd,
Farmakidou 10,Chalkida, GR34100, Greece
ygp@ synelixis.com

Abstract*— Graph cuts are very popular methods for combinatorial optimization mainly utilized, while also being the most computational intensive part, in several vision schemes such as image segmentation and stereo correspondence; their advantage is that they are very efficient as they provide guarantees about the optimality of the reported solution. Moreover, when those vision schemes are executed in mobile devices there is a strong need, not only for real-time processing, but also for low power/energy consumption. In this paper, we present a novel architecture for the implementation, in reconfigurable hardware, of one of the most widely used graph cuts algorithms, which is also the fastest sequential one, called BK. Our novelty comes from the fact that we use a 2-level hierarchical decomposition method to parallelize it in a very modular way allowing it to be efficiently implemented in FPGAs with different number of logic cells and/or memory resources. We fast-prototyped the architecture, using a High level synthesis workflow, in a state-of-the-art FPGA device; our implementation outperforms an optimized reference software solution by more than 6x, while consuming 35 times less energy;. To the best of our knowledge this is the first parallel implementation of this very widely used algorithm in reconfigurable hardware.

Keywords—Graph Cuts; Markov Random Field; Dual Decomposition; Low Power; FPGA; Embedded System.

I. INTRODUCTION

Minimum S-T cut (MinCut) is a classical combinatorial problem [1] employed in several low-level computer vision problems, such as image smoothing, denoising, image segmentation, stereo correspondence, and image registration of different modalities (i.e. fusion) [2]. The common characteristic of all those applications is that they all use graphs with regular structure, with vertices arranged into an N-D grid. At the same time there are numerous applications in the consumer industry sector requiring efficient graph solvers. For example, there was the ground breaking, at their time, Lytro startup company that introduced its first generation pocket-sized camera, capable of refocusing images after being taken [3]; Lytro uses a plenoptic camera that uses a micro lens array to capture 4D light field information [4] and it involves a graph cut scheme. Similar schemes are employed by large phone vendors, such as HTC, LG and Nokia which are trying to simulate Lytro's light-field technology. Morever Google has introduced a mobile application for android devices [5] which features lens blur during the post processing of the photo, as well as the Tango Project [6] which uses an Android phone with a highly customized hardware to track full 3-dimensional motion; it features depth sensing cameras providing a huge point cloud that is then fused with the modalities of normal cameras. In both systems, a certain graph-cut scheme is involved in order to produce the desired imaging effect.

This paper presents an architecture, tailored to reconfigurable logic, for the implementation of probably the most widely used graph-cut algorithm; the proposed system, when prototyped in a state-of-the-art FPGA consumes 35 times less energy than the existing optimized software approaches while it is more than 6x faster.

II. RELATED WORK

Our work introduces a variant of the path augmentation algorithm that was originally presented from Boykov and Kolmogorov in [7]. The original scheme, since it is not natively parallel, is hard to be partitioned while its large memory footprint does not allow it to fit within the limited memory of certain embedded devices including the FPGAs.

There are several proposals for the parallelization of the Boykov and Kolmogorov's algorithm in the literature. Liu and Sun [8], partition the graph in parallel disjoint graphs; in a second phase they merge the parallel disjoint graphs into larger graphs in order to obtain the global solution. The limitation of this method is that the whole graph should be placed in the limited on-chip memory. In [9], Shekhovtsov and Hlaváč perform path augmentations inside the parallel disjoint graphs and update them by using the "push-relabel" approach between different regions. The main problem with this approach is that it is a control intensive design that cannot be efficiently parallelized.

There is also an FPGA-based implementation of a max-flow algorithm which is presented in [10]. This approach utilizes a push-relabel method tailored to the FPGA characteristics. The authors claim a 3-5x speedup when compared to a slower (in comparison with today's state of the art) Intel CPU. However, they don't give any references for the compile and optimization levels of the software implementation that they are compared neither for the I/O latency and bandwidth demands of their solution. They also do not address scalability issues as their design can support only certain image resolutions

In our work, we start from the approach of Strandmark and Kahl [11], where the max flow/min cut problem is parallelized by splitting the graph into disjoint parts. More importantly, we introduce, for the first time, an extension of the above framework in a 2-level hierarchical implementation with different granularities at each level in order to allow it to take full advantage of the reconfigurable logic features as well as the external large memories of today's reconfigurable embedded systems.

The result is the first fast, while very power efficient, FPGA-based system implementing a popular graph-cut scheme. The contribution of this work though, goes beyond this particular implementation as it proposes certain novel approaches on how inherently serial graph-cut algorithms can be efficiently parallelized while addressing the on-chip memory limitations of reconfigurable devices.

III. GRAPH SPLITTING

Our hardware architecture is based on splitting the initial graph into smaller disjoint sub-graphs in a similar way to the one presented in [11]. This graph splitting serves a two-fold purpose: firstly, it enables us to increase the level of parallelization and take advantage of the parallelism of today's reconfigurable devices and secondly it permits us to scale to larger problems that cannot fit in the current on-chip memory.

Fig. 1. Original Graph

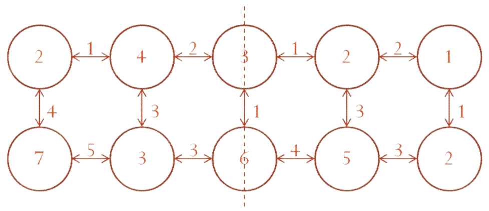
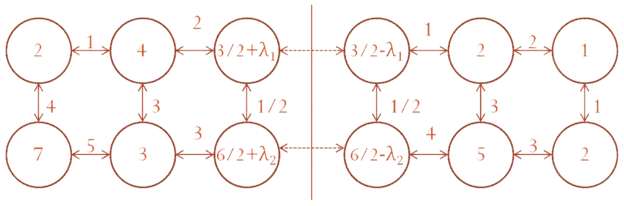


Fig. 2. 1-D graph split



Figures 1 and 2 show in an illustrative example how a single graph is split in to two disjoint sub-graphs. The graph split consists of two steps. In the first step, the node data are duplicated across the split region while weighted by a ½ factor. Thus if we merge again the disjoint parts we get the initial graph. In this respect, the disjoint sub-graphs partially overlap while there is an overlap margin which defines how many nodes in the two sub-graphs overlap. In our simplified example the overlap is 2 but typically it is set equally to the linear dimension of the *n* x *m* graph (i.e overlap is equal to *m* or *n*) in the case of single dimensional splits. The second step forces the sub-graphs to "agree" (i.e. introduce the same values) for the overlap region. It introduces the dual variables λ_i as s/t connections, a technique introduced by Everett [10] and applied in different contexts. Our work is based on the approach of [11]; through an iteration process, it performs simultaneous updates in λ_i in both sub-graphs and assures convergence to the same solution.

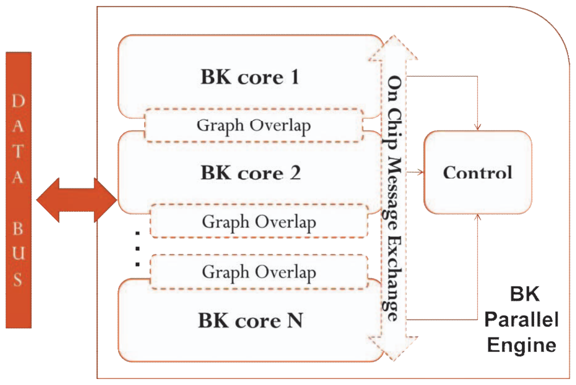
IV. HARDWARE ARCHITECTURE

In this section we describe in detail the hardware architecture of the proposed BK parallel system. In subsection A we give the organization of the basic hardware module where in sub-section B we demonstrate how this hardware module can be replicated, in order to take advantage of the vast reconfigurable resources of today's systems, and form a larger parallel system.

A. System's Core Architecture

As a starting case study our basic hardware module, which fully implements the scheme presented in [11], consists of 16 parallel cores each processing a disjoint part of the graph as shown in Figure 3. The BK Parallel engine in this way processes 16 sub-graphs consisting of 1K[1] nodes each, thus simultaneously process 16K nodes. Larger graphs can be processed on-chip, depending on the devices resources as demonstrated in Section V. Moreover, in sub-section C we discuss how the proposed system can scale to larger graphs that cannot fit on-chip.

Fig. 3. BK parallel engine organization overview



Each of the cores can communicate through the common bus in order to load new data. The data loading is performed in batches of 16K nodes as described in detail in subsection B.. By performing certain in-order iterations the algorithm converges for the 16K nodes batch while it makes only local updates in the residual graph. This operation is carried out through a message exchange scheme we developed and is coordinated by the control unit. During the write back process the BK Parallel Engine returns only the 1-bit labels of the graph (in a predefined order), so as to minimize the output time.

B. Parallelism during Data Loading

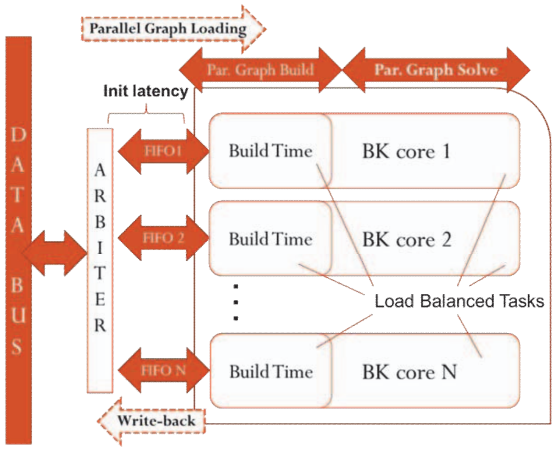
In most hardware accelerators a certain amount of data is initially loaded on-chip and then those data are processed in parallel. Graph processing is not any different, but it further demands time-consuming operations during the initial graph

[1] The 1K nodes per core was derived from the desired level of parallelism and the capacity of the target FPGA

building task. In this respect, our novel scheme exploits the parallelization, not only in the graph solving process, but also in the graph building task, which in terms requires a load-balanced loading process. Thus we sort and organize the data before loading them in order to fully exploit, during the graph building task, all the available HW parallel cores of the FPGA.

The classical approach of most SW threaded or distributed implementations is to sequentially load the node data (without reordering them) to the different computation units. In our approach we use a common shared-bus communication in a round-robin fashion to evenly distribute, during loading, the graph-data to all the available cores. By sending small parts of the graph to each core we minimize the initial latency while we evenly distribute the graph building task among the parallel cores. In this respect our novel architecture hides the bus transfer overhead in the graph building time. As the graphs are equally sized the graph solving task is also load balanced; as a result we achieve a fully pipelined operation through all the stages of our scheme. The aforementioned configuration is depicted in Figure 4.

Fig. 4. Our Load-balanced loading scheme completely hiding bus latency while also exploiting parallelism during graph building.



On the other hand as our system achieves significant acceleration of the graph solving task, the bus may become the bottleneck of the system; our accelerator triggers a very high throughput in terms of nodes/sec (see Section VI), so even a high-bandwidth bus can get into its limits. As a result we introduce a scheme which increases the effective bus bandwidth; we apply a low latency payload compression scheme so as to reduce the size of the data transmitted over our bus. When a graph is built the neighboring nodes, which are added-in sequentially, usually share the same weights or just slightly different ones. For this reason we employ a delta encoding compression scheme [12] so as to only transmit the differences of the weights between the added nodes and not the weights themselves; our delta values are 1-byte long Thus, as shown in Table I, we can save up to 4 bytes in each edge transmission and up to 3 bytes in each node transmission. In the cases that we cannot fit the weight-difference within the 1-byte delta value we transmit all the information in an Uncompressed manner.

TABLE I. BUS OPTIMIZATION USING DELTA COMPRESSION

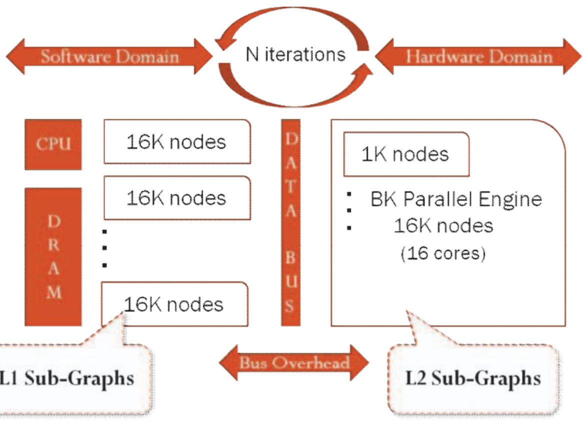
Payload config.	*Uncompressed (Bytes)*	*delta Compressed (Bytes)*	*Hit Rate*	*BW reduction*
nodes	6	3	60%	
arcs	8	4	98%	
avg. BW	680 MB/sec	520 MB/sec		-23%

Based on our simulation results, as also shown in Table I, we demonstrate that during the nodes data transmission we reduce the data sent in 60% of the cases, where we can compress the arc data in 98% of the cases. This saves us about 23% of the total bandwidth needed. The bandwidth figures refer to average values in a batch of 16K nodes while they are all based on real-world graphs.

C. *Hierarchical Graph partitioning and Scalability*

As previously mentioned in Section III, graph split allows us to scale to larger problems exceeding the on-chip memory resources. We follow the same principle and split our bigger problem (L0) in sub-graphs, each one of size 16K nodes, for our case study. We call those graphs for the rest of this text L1 sub-graphs. In L1 sub-graphs an overlap area is also introduced to ensure the smooth transition of the result while, and more importantly, the global convergence.

Fig. 5. A Hierarchical design Architecture. 16K Nodes Graphs sequentially loaded in the HW parallel Engine and parallelized in L2 Sub-graphs.



In this respect we propose a two-step hierarchical sub-graph splitting approach. The first step (L1) refers to a graph size equal to the on-chip capacity of their HW parallel engine (e.g 16K nodes) while the second step (L2) refers to the capacity of each individual HW core (e.g. 1K nodes). In this regard each of the L1 sub-graphs is loaded to the FPGA and as demonstrated in the previous subsection, they are all processed in parallel on-chip. Finally there is an iteration procedure so as to converge to a global solution; Figure 5 depicts this approach.

V. IMPLEMENTATION AND HARDWARE COST

The architecture presented in the previous section is implemented using a high-level synthesis (HLS) design flow.

The basic building block (i.e core) implements the scheme of [7] which is based on Breadth First Search (BFS) [13]. The code has been hand-crafted so as to allow for a fully parallel implementation; the tool itself could not automatically optimize/parallelize such a scheme as it is inherently serial. It was crucial that in the HLS design process all the pointer operations involved in the graph traversal were abstracted [2] and this pushed the design effort at a higher architectural level. In this respect, our novel system scales to many cores while efficiently supporting the control intensive task of the graph split and the message exchange between the cores. Each of the cores handled equally sized disjoints parts of the graph; thus all the distributed tasks were optimally pipelined across them. The control unit depicted in Figure 3 is mainly implementing the scheme of [11] and handles the communication between cores that guarantees the global convergence of all the sub-graphs.

We also made sure that the new hardware interface is 100% compatible with the software interface of the reference code, since our scheme can be part of a Hardware Library which can be utilized by the embedded applications that use the BK algorithm. For the code synthesis we used Xilinx's Vivado HLS while in order to parallelize our basic block we manually instantiated 16 cores; this was due to the fact that the HLS tool could not handle automatically this parallelization task even after we have utilized all the requested design directives.

The whole design is fully parameterizable in terms of number of cores as well as of graph sizes since all the scalable components of the design have been described in a parametric way using synthesizable-C++. This enables our scheme to be implemented in any available reconfigurable device, no matter what its silicon capacity is .

TABLE II. TYPICAL HARDWARE COST ON XILINX VIRTEX 7 DEVICE (XC7VX330TFFG1761-3)-16 CORES. WE UTILIZE 16K NODES AND 64K EDGES ON-CHIP.

Logic Utilization	*Used*	*Available*	*Utilization*
Number of Flip Flops	26813	408000	6%
Number of Slice LUTs	33208	204000	16%
Number of DSP48E	16	1120	~1%
Number of Block RAM_18K	1320	1500	88%

For our reference design we targeted both an average Xilinx Virtex 7 and a small Zynq. In Table II we show the hardware utilization of the Virtex 7 device; moreover for validation and proof of concept [3] our system was also implemented in a very low end Zedboard evaluation platform [14] powered by a small Xilinx Zynq device and interconnected with a standard PC.

Table II shows that the critical resource is the on-chip memory (Block RAM), as the graph data dominate the utilization of the device. The ratio between the supported on-chip nodes and edges could be easily changed according to the target application. As described in Section IV.C our architecture can handle arbitrary size graphs by splitting them in smaller ones that can fit in the available on-chip resources and then re-merged in a host computer.

[2] This is done manually by "bookkeeping" indexes in predefined arrays.

[3] By supporting the same parallel cores while handling smaller graphs.

VI. EVALUATION AND PERFORMANCE

A. Performance and Power Efficiency

In this sub-section we evaluate the performance of the proposed scheme when handling graph sizes that can fit in the on-chip device memory. In the next section we demonstrate the scalability of our system so as to handle larger graphs.

The clock speed we achieved on the Virtex 7 device (speed grade -3) is 260MHz. The table below shows the speedup triggered when our system is compared with a mobile Intel i5 CPU, clocked at 3GHz (2.5 GHz with Turbo Boost). We have selected this CPU as a reference since it is power efficient while it delivers high performance, in single threaded applications such as the BK one. The software compiler platform used was Microsoft Visual Studio 2012 and our reference software code [1] was optimized for maximum speed (maximum optimization level, i.e. -O2. It should be stressed that we use the code of [7] and not the code of [11] as the reference for our evaluation, as the former is considered a baseline implementation and an important benchmark for numerous similar systems such as the ones in [11], [10], [15]. In all the software experiments we excluded the disk I/O time and we measured the latency times after the data were loaded to the DRAM of the CPU. This is in favor of the CPU as the cache (3MB L3) is hot in most of the measured cases. In any case our measurements show that our hardware is at least six times faster than a state-of-the-art Intel CPU. Table III shows an 8-core and a 16-core configuration of the proposed system. This speedup is not affected by the input figures in any manner

TABLE III. BK ENGINE PERFORMANCE FIGURES FOR GRAPH SIZES UP TO 16K AT 260MHZ (XC7VX330TFFG1761-3)-16 CORES

Config.	*Total Nodes*	*HW latency ms*	*SW Latency ms*	*I/O Bandwidth* **MB/sec**	*HW speedup*
8cores	16K	1.79	6.1	229	3.3x
16 cores	16K	0.95	6.1	521	6.2x

As analytically described in Section IV the I/O latency of our system is minimal as it needs to cache only a very few nodes in order to start the computations. Thus the initial latency is almost zero for each graph processed. The write-back time is also very small as we send only the graph labels (i.e. single bit encoded label output). As a result the bandwidth figures are quite low and fully tolerable by current FPGAs.

In terms of power consumption according to Xilinx Power estimator [16] our system consumes less than 3.6 watts for the whole processing cycle. On the other hand Intel i5's power consumption as measured by Intel's Power Gadget (which monitors precisely and estimates real-time power information in watts using the energy counters of the processor [17]) was 22.3 Watts in average throughout the software execution. Thus our system is 6x more power efficient. Given the 6x speedup and the 6x reduction of the power consumption we conclude that our FPGA-based device consumes more than 35 times less energy than a low-power state-of-the-art CPU.

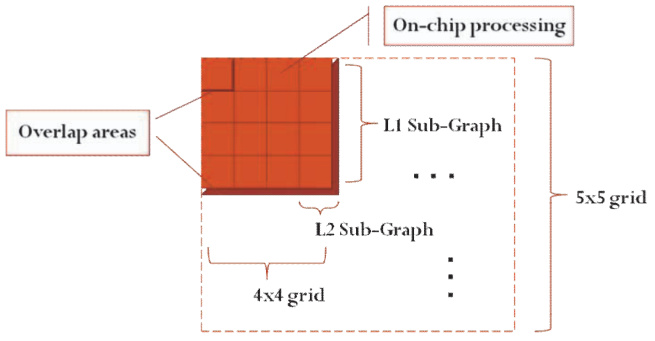
Moving to a comparison with the high end GPUs, in [18] the authors report a 10 to 12 times speedup of this same reference algorithm on a Nvidia GTX-280 GPU; the GPU executes only the image segmentation tasks. In the work of [15] the authors report a 5.2x speedup in a GTX-580 for the stereo matching tasks. In both works though, no details are given for the reference CPU against which they compare their performance nor about the optimization level of the reference software. Even if GPUs achieve similar or slightly better speedup figures, they consume hundreds of Watts (200W-300W). As a result they trigger a much lower performance per watt ratio that our FPGA-based system; even if we assume the maximum 12x speedup at 200Watts, our FPGA-based system is 27 times more power efficient than the best GPU-based one.

B. *Scaling vs. Accuracy*

In this sub-section we describe how our system scales to larger problems which cannot fit on-chip. Apart from the overlapping nodes that slightly decrease system's performance, a second issue can arise: the converging procedure that ensures a global solution for all L1 sub-graphs[4] may force them to be re-processed on-chip for a second iteration. In such cases the I/O would increase the total execution time as, during the write back, we should read the whole graph (and not only the labels) in order to be able to reprocess all of them. In order to alleviate this issue we extend the idea of [8] in a 2-level hierarchical configuration in order to efficiently exploit the geometrical characteristics of the graph.

We extend the already described 1-dimentional splits into 2-dimentional splits forming a 4x4 grid of 16 on-chip parallel engines dealing with the 16 disjoint L2 sub-graphs. In this respect the larger L1 sub-graphs are forming a square grid (e.g 4x4 L1 sub-graphs) as shown in Figure 6. This approach drops the error introduced by our scaling scheme more than 4 times and as a result the communication, for the vast majority of the cases, between the larger graphs is not necessary; by using just the overlapping area between them, the system converges to a minimum error global solution in a single iteration.

Fig. 6. A larger problem is in a 2-dimentional grid

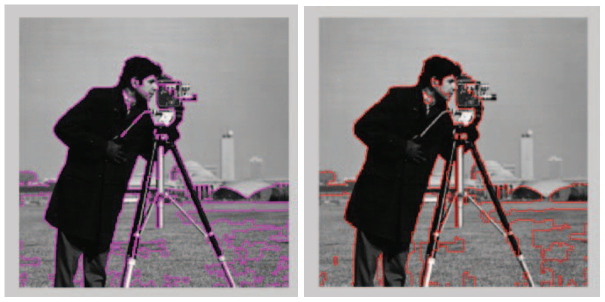


[4] The convergence to minimum error in L2 sub-graphs is always assured through on-chip communication between them.

Furthermore after experimenting with various images up to High Definition (HD) resolutions, we concluded that the final error, when each sub-graph is processed only once, is minimized to less than 0.5% while the global convergence is not affected. The only cases where we may have a slightly larger error, compared with the sequential version of the BK algorithm, is when we handle a very large regularization parameter[5] **λ** for a given large image resolution.

We should note that the need for convergence in larger graphs increases as the regularization parameter **λ** increases as well. This is the case as the smoothing role that λ plays in the graph building task increases the need for communication between the different L1 sub-graphs. In general, all the distributed versions of the global optimization algorithms suffer from the same problem no matter if they are implemented in software, hardware or even in distributed clusters: the larger the smoothing the more I/O intensive the processing. We claim that if a meaningful segmentation depends on a large λ, maybe it is better and more efficient, in performance demanding applications, to down-sample the image and then apply to that the large regularization parameter. For example the authors in [19] suggest that applying a large λ to a whole image segmentation is not a good strategy. This is due to the fact that while large λ improves the segmentation in the noisy parts of the image, it also worsens it in the content-rich parts.

Fig. 7. On the left the serial BK algorithm. On the right our scheme handles sensitivity to noise by scaling down and regularizing with smaller λ.



In Figure 7, we show a case where a noisy image is processed with a large λ parameter. Though the error between the sequential algorithm seems to be relatively high (about 1%) it produces equally interpretable (or better) results when we scale down the image before we apply the same regularization level.

Obviously, we don't claim that we are able to scale infinitely our scheme without being sensitive to local spatial characteristics. We can, though, handle high resolutions (i.e. High Definition or HD) at a very high speed and with low energy consumption, while also controlling the sensitivity to local noise. Based on this analysis we claim that our system can handle up to HD image resolutions without performing any processing iterations through the larger L1 graphs while still having very good performance as well as accuracy .

[5] the regularization parameter **λ** (lambda) it is a smoothing term in the graph

C. Comparison with Related Work

Moving to the comparison of our scheme with the work of [20] we see that though they claim to achieve a similar speedup, our scheme is compared with a 50% faster CPU while, in our case, we run a fully optimized software code (the difference in the CPU performance when no compiler optimizations were utilized was more than 40%). Furthermore in our work the performance is always guaranteed as the parallel stages of our scheme (i.e parallel cores processing disjoints parts of the graph) are always fully utilized. In the work of [10] as the pipeline is more fine-grained it is not guaranteed that it is always full and thus the performance is not constant when compared with the reference BK serial code of [7]. Finally, in our work we give a detailed description of the communication demands between the host and the accelerator where in the case of [10] such description is not available and more importantly we demonstrate how our system can be scaled so as to handle larger graph sizes, whereas their system is limited to small graphs that can fit on-chip.

VII. Conclusions

In this paper we propose a novel 2-level hierarchical graph split scheme tailored to reconfigurable devices. Moreover, we present a novel modular architecture, which among others, incorporates a simple, yet efficient data compression scheme so as to minimize the I/O overhead. Our design is fully scalable to very large graphs while also offering certain performance and accuracy guarantees.

Our results show that our approach outperforms a state of the art CPU by at least 6 times in the very challenging graph cut problem. In terms of energy consumption the gain is even better as our prototype scheme achieves 35 times lower consumption for this same task and at the same accuracy than a power efficient CPU while it is at least 27 times more power efficient than the best GPU-based implementation.

Acknowledgment

The work presented in this paper has been supported by the Greek General Secretariat for Research and Technology (GSRT) through the funding of research project “AFORMI – Reconfigurable Systems for scientific research” with proposal code 2427 within the context of action “ΑΡΙΣΤΕΙΑ” of the Lifelong Learning Program.

The authors would like to thank Peter Strandmark for his constructive comments that greatly contributed to improving the final version of the paper.

References

[1] E. Lawler, “4.5. Combinatorial Implications of Max-Flow Min-Cut Theorem, 4.6. Linear Programming Interpretation of Max-Flow Min-Cut Theorem,” *Comb. Optim. Netw. Matroids*, pp. 117–120, 2001.

[2] M. B. A. Haghighat, A. Aghagolzadeh, and H. Seyedarabi, “Multi-focus image fusion for visual sensor networks in DCT domain,” *Comput. Electr. Eng.*, vol. 37, no. 5, pp. 789–797, Sep. 2011.

[3] “http://www.digitaltrends.com/photography/lytro-the-camera-that-could-change-photography-forever/.” .

[4] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan, “Light Field Photography with a Hand-Held Plenoptic Camera,” Stanford University Computer Science Tech Report CSTR 2005-02, Apr. 2005.

[5] “Research Blog: Lens Blur in the new Google Camera app.” [Online]. Available: http://googleresearch.blogspot.gr/2014/04/lens-blur-in-new-google-camera-app.html. [Accessed: 07-Sep-2015].

[6] “Project Tango – Google.” [Online]. Available: https://www.google.com/atap/project-tango/. [Accessed: 07-Sep-2015].

[7] Y. Boykov and V. Kolmogorov, “An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 9, pp. 1124–1137, Sep. 2004.

[8] J. Liu and J. Sun, “Parallel graph-cuts by adaptive bottom-up merging,” in *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 2181–2188.

[9] A. Shekhovtsov and V. Hlaváč, “A Distributed Mincut/Maxflow Algorithm Combining Path Augmentation and Push-Relabel,” *Int. J. Comput. Vis.*, vol. 104, no. 3, pp. 315–342, Sep. 2012.

[10] D. Kobori and T. Maruyama, “An acceleration of a graph cut segmentation with FPGA,” in *2012 22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 407–413.

[11] P. Strandmark and F. Kahl, “Parallel and distributed graph cuts by dual decomposition,” in *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 2085–2092.

[12] C. Xiao and B. Bing, “Delta compression with fixed-length substring coding for fast content download,” *IEEE Commun. Lett.*, vol. 9, no. 3, pp. 243–245, Mar. 2005.

[13] S. S. Skiena, *The Algorithm Design Manual*. London: Springer London, 2008.

[14] “Zedboard Evaluation Board.” [Online]. Available: http://zedboard.org/.

[15] Y. Choi and I. K. Park, “Efficient GPU-Based Graph Cuts for Stereo Matching,” in *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2013, pp. 642–648.

[16] “Xilinx Power Estimator (XPE).” [Online]. Available: http://www.xilinx.com/products/technology/power/xpe.html. [Accessed: 07-Sep-2015].

[17] “Intel® Power Gadget | Intel® Developer Zone.” [Online]. Available: https://software.intel.com/en-us/articles/intel-power-gadget-20. [Accessed: 07-Sep-2015].

[18] V. Vineet and P. J. Narayanan, “CUDA cuts: Fast graph cuts on the GPU,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. CVPRW ’08*, 2008, pp. 1–8.

[19] J. Rao, R. Abugharbieh, and G. Hamarneh, “Adaptive Regularization for Image Segmentation Using Local Image Curvature Cues,” in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Springer Berlin Heidelberg, 2010, pp. 651–665.

[20] A. V. Goldberg, S. Hed, H. Kaplan, R. E. Tarjan, and R. F. Werneck, “Maximum Flows by Incremental Breadth-First Search,” in *Algorithms – ESA 2011*, C. Demetrescu and M. M. Halldórsson, Eds. Springer Berlin Heidelberg, 2011, pp. 457–468.