FPGA IMPLEMENTATION OF A GRAPH CUT ALGORITHM FOR STEREO VISION

Ryo Kamasaka, Yuichiro Shibata, Kiyoshi Oguri
Graduate School of Engineering
Nagasaki University, Japan
kamasaka@pca.cis.nagasaki-u.ac.jp, {shibata,oguri}@cis.nagasaki-u.ac.jp

ABSTRACT

This paper presents an FPGA implementation of a graph cut algorithm for stereo vision, in which object surfaces are estimated from a 3D grid graph composed using differences of two images. Difficulty of real-time graph cut of 3D grid graphs is due to a large amount of computing time. To solve this problem, we implemented a graph cut system in an FPGA, making the best use of a rich on-chip memory bandwidth. In this work, we propose a hardware architecture for the push-relabel algorithm with gap relabeling heuristics. We achieved approximately 5 frames per second for a graph with $129 \times 129 \times 16$ nodes at the clock frequency of 95 MHz. Our approach proves 2.7 times acceleration compared to an existing graph cut software library.

1. INTRODUCTION

Stereo vision, which reconstructs 3D information from images obtained by two cameras, is one of the topics most actively addressed in the filed of computer vision. In a typical approach of the stereo vision, the first step is to extract feature points from the two images. Then, correspondence of the feature points between the left and right images, so that a disparity map is eventually obtained. One of the difficulties of this approach comes from an occlusion issue, in which an extracted point appears only in one of the two images. Therefore, it is not straightforward to find correspondence of the feature points and sophisticated techniques are needed [1] [2].

To cope with this problem, we are investigating another approach of stereo vision [3]. Instead of extracting feature points from 2D images, we consider points in a 3D space directly. As shown in Fig. 1, let us consider a point A in a 3D space emits light rays to the centers of the lenses of the two cameras. If an object surface exists at A and it reflects light, the almost same colors and intensities will be obtained at the pixel X in the left camera and at the pixel Y in the right camera. In other words, if the pixel colors and intensities at X and Y are similar, there is likely to be an object surface at A in the 3D space. To the contrary, if the two pixels at X and Y differ largely, there would be nothing at the point A. Furthermore, when we compare the pixel X in the left camera and the right adjacent pixel to Y in the right camera, we can examine the surface existence at the point B as shown in Fig. 2. The point A and B are on the same circle called a horopter. When we compare the left adjacent pixel to X and the right adjacent pixel to Y, the point C on a smaller horopter can be analyzed. In this way, we can think of many points on horopters, which we call *hypothetical surface points*, and calculate *likelihoods* of surface existence on those points by comparing pixels between the images obtained by the two cameras.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

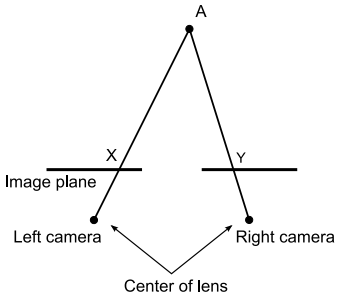


Figure 1: Camera layout in which light rays from A to centers of two lenses are observed at pixels X and Y

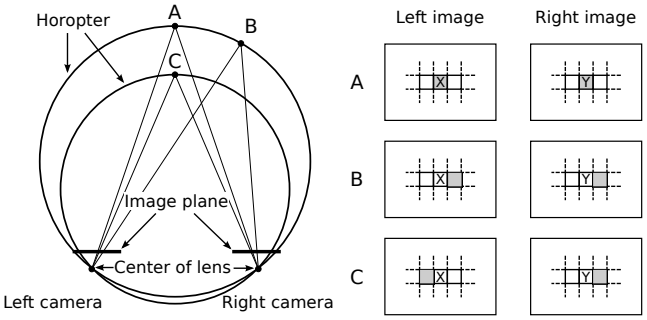
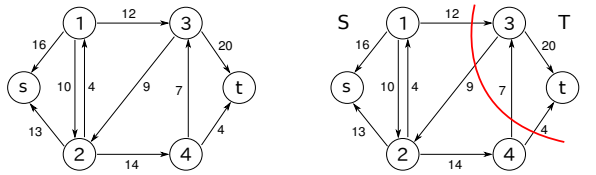


Figure 2: Horopters and hypothetical surface points

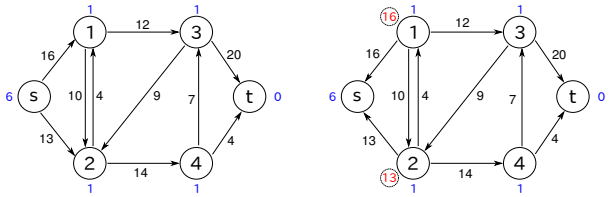
In this way, we can construct a 3D grid graph, whose nodes correspond to hypothetical surface points and edges represent likelihoods of existence of surfaces. Since surfaces of typical objects are geometrically smooth, extraction of object surfaces in a 3D space can be considered as division of hypothetical surface points into two groups: front and back of the surface. Thus, by finding a min cut of the 3D grid graph with inverse likelihoods of surface existence, 3D information of the surface of a target object can be obtained. Note that 3D points that may cause occlusion issues are naturally eliminated from hypothetical surface points in this method.

A computational bottleneck of this approach is graph cuts of a 3D grid graph. In order to realize this approach in a realtime embedded computational environment such as robotics, energy efficient acceleration of a graph cut algorithm is essential. So far, many attempts to accelerate graph cut algorithms have been reported. For example, Cudacut, a graph cut library for GPUs provided by NVIDIA, can process 1024×1024-pixel images at 60 frames per



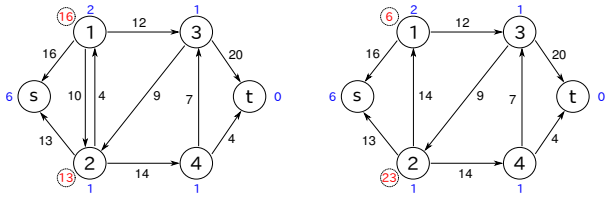
(a) (b)

Figure 3: Example of min-cut: (a) given graph and (b) min-cut result



(a) (b)

Figure 4: Initialization steps in push-relabel method: (a) initial heights and (b) initial excess flows



(a) (b)

Figure 5: Example of push-relabel operations: (a) relabel and (b) push

second (fps) [4]. An FPGA accelerator of graph cuts can perform image segmentation for 640×450-pixel images at 20 to 30 fps with limited power consumption [5]. However, these attempts are dedicated or optimized for 2D grid graphs, as acceleration of graph cut algorithms has been mainly investigated in a context of 2D image segmentation. Therefore, we cannot apply directly these existing approaches to this stereo vision method.

In this paper, we propose and implement an FPGA-based accelerator of graph cuts for 3D grid graphs in stereo vision. The rest of the paper is organized as follows. In Section 2, algorithms for graph cuts are introduced. In Section 3, graph structure to be processed in this stereo vision approach is briefly explained. Section 4, design and implementation of the proposed acceleration system are presented. The evaluation results with actual 3D stereo vision graphs are demonstrated and discussed in Section 5. Finally, the paper is concluded in Section 6.

2. GRAPH CUT ALGORITHMS

2.1 Graph cuts and max-flow problem

Graph cuts are one of methods for solving energy minimization problems. Let $G = (V, E)$ be a graph consisting of a set of nodes V and a set of edges E. The source node is denoted by $s \in V$ and the sink node is denoted by $t \in V$. A graph cut is an operation to divide V into two sub sets S and T. Let us consider each edge from node u to node v has a positive capacity denoted by $c(u, v)$ as indicated in Fig. 3 (a). For example, $c(1, 3)$, the capacity of the edge from node 1 to node 3, is 12 in this graph. The cut capacity $c(S, T)$ of a graph divided into two subgraphs S and T is defined as the summation of capacities of edges from the nodes in S to nodes in T. A cut which minimizes this capacity is called min-cut. An example of the min-cut is shown in Fig. 3 (b), where $c(S, T) = 12 + 7 + 4 = 23$. Note that capacities for the edges from the nodes in T to the nodes in S are not included in the cut capacity [6].

The max-flow problem is defined as a problem of finding the maximum flow from s to t in a flow network, which is a directed graph with capacities. A flow from node u to node v is denoted as $f(u, v)$ and its value does not exceed the flow capacity $c(u, v)$, that is,

$$0 \le f(u, v) \le c(u, v). \tag{1}$$

The amount of a flow from s to t is defined as

$$|f| = \sum_{u \in V} f(u, t) = \sum_{v \in V} f(s, v), \tag{2}$$

that is, $|f|$ is the amount of the total flows leaving from s or the total flows received by t. The goal of the max-flow problem is to find the maximum flow from s to t for a given flow network.

By the max-flow min-cut theorem, the amount of the maximum flow and the min-cut capacity are the same, therefore the min-cut problem can be replaced with the max-flow problem. Major algorithms for finding the maximum flow are the augmenting-path method [7][8] and push-relabel method [9]. In these algorithms, a concept called a residual network is used. A residual network is a network with remaining capacities and is created from a flow network. The residual capacity c_f is defined as

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & (u, v) \in E, \\ f(v, u) & (v, u) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The augmenting-path method is not suitable for hardware implementation as it needs to scan the entire graph to search for paths from s to t. In the push-relabel method, the number of data to be handled is relatively small, since the max-flow can be obtained by locally manipulating flows. Thus, we adopted the push-relabel method for our implementation.

2.2 Push-relabel method

In the push-relabel method, a height label h is given to each node, and a preflow g that flows from a higher node to a lower node is utilized. For the preflows, an inflow amount to a node is allowed to be larger than an outflow amount of the node. Thus, there is a possibility that flows that cannot be flowed out from a node will be accumulated. This accumulated flow is called an excess flow and is defined by

$$e(v) = \sum_{u \in V} g(u, v) \ge 0 \qquad \forall v \in V - \{s\}. \tag{4}$$

A node v with $e(v) > 0$ is called an active node. In the push-relabel method, the maximum flow is obtained by iteratively applying the following Push operation and Relabel operation to active nodes.

- push(u, v) : If node u has a larger height than node v, which is one of the neighbor nodes of u and $c_f(u, v) > 0$, an excess flow is flowed from u to v as much as possible.
- relabel(u) : If the height of node u is less than or equal to the height of all neighboring nodes that can accept a flow, the

height $h(u)$ is increased by 1 from the lowest height among the neighboring nodes.

By repeating these two operations as far as there exist active nodes, the maximum flow is obtained [6][9].

A procedure when the push-relabel method is applied to the example network shown in Fig. 3 (a) is as follows.

1. Initialization of heights: The heights of s and t are initialized to the number of nodes and 0, respectively. For the other nodes, the initial height of 1 is given. The numbers labeled near the nodes in Fig. 4 (a) represent the initial heights.

2. Initialization of excess flows: By pushing flows as much as possible from the source node to its neighbor nodes, initial excess flows are given as shown in Fig. 4 (b). The numbers surrounded by dotted lines show excess flows accumulated in the corresponding nodes.

3. Relabel operation: Relabel operations are performed on active nodes that satisfy the condition of the Relabel operation. Fig. 5 (a) shows the residual network after performing the Relabel operations.

4. Push operation: Push operations are performed on active nodes that satisfy the condition of the Push operation. Fig. 5 (b) is the residual network after performing the Push operations.

5. Repeat of two operations: The Relabel operations and Push operations are repeated as long as there exist active nodes.

Finally, the maximum flow is found to be 23. The set of nodes $\{1, 2, 4\}$ that can be traced from s belong to the subgraph S and the rest belongs to the subgraph T, as shown in Fig. 3 (b).

2.3 Heuristics

For the push-relabel method, the following two heuristics have been proposed for reducing the calculation amount [9][10].

- Global-relabeling: Periodically searches for a route from a certain node u to t and changes the heights of the nodes along that route according to the distance to the t.

- Gap-relabeling: If a node with a certain height k does not exist while a higher node exists, it is invalidated.

Note that these heuristics do not affect the accuracy in graph cut.

The global-relabeling is not suitable for hardware implementation, since it makes indirect reference to the entire graph in the search process of a route to t. On the other hand, histograms of node heights for gap-relabeling can be easily implemented on FPGAs with parallel accessible registers. Therefore, we adopted the gap-relabeling heuristics for our system.

3. STRUCTURE OF TARGET GRAPHS

Fig. 6 (a) shows a structure of a 3D grid graph to be processed in our stereo vision system, while Fig. 6 (b) illustrates one of 2D planes of the graph. The nodes correspond to hypothetical surface points and edges represents inverse likelihoods of surface existence. By finding the min-cut of this graph, 3D information of the target surface can be obtained. Unlike graphs for image segmentation, s and t are not connected to all nodes in stereo vision graphs. As shown in Fig. 6 (a), each node has connections for 14 directions (14 neighbors) or 15 directions (14 neighbors and s or t).

Each edge has an attribute of *Cost*, *Inhibit*, or *Penalty*. The number of edges per node with each attribute is 2 for *Cost*, 8 for *Inhibit*, and 4 for *Penalty*. The *Inhibit* and *Penalty* attributes were introduced to prohibit unnatural surfaces to be extracted. Therefore, to the labels for *Inhibit* and *Penalty* edges, fixed values are given in advance regardless of image information obtained by cameras. In this implementation, 1023 and 7 are assigned to the values for *Inhibit* and *Penalty*, respectively. On the other hand, values for *Cost* are inverse likelihoods of surface existence and are calculated from differences between two images captured by cameras. In this implementation, values within the range not exceeding 1023 are assigned for *Cost*.

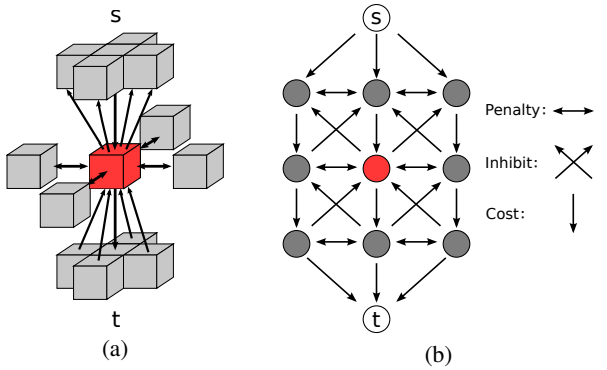


Figure 6: Example of 3D grid graph for stereo vision: (a) connection to neighboring nodes and (b) a 2D plane in the graph

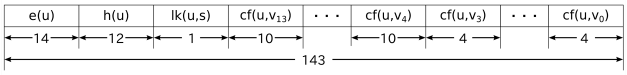


Figure 7: Data structure and bit widths for one node

4. PROPOSED ARCHITECTURE

First of all, we designed the data structure for each node as depicted in Fig. 7, where required data widths were determined from the result of preliminary software evaluation. Each node has residual capacities $c_f(u, n_i)$ $(0 \leq i \leq 13)$ to 14 neighbors. In addition, there is also the possibility of having residual capacity to s or t. However, edges to s and t can be eliminated by changing their initial capacities to excess flows as shown in Fig. 8. As a result, there is no need to keep residual capacities to s and t in each node. Instead, only a connection flag to s denoted by $lk(u, s)$ is needed for the process of finding the maximum flow. The total data width for each node becomes 143 bits.

Fig. 9 shows a block diagram of the proposed system. Unlike [4] and [5], the system was tuned for processing 3D grid graphs. In this implementation, entire graph information is stored in internal memory of an FPGA and any external memories are not utilized.

A given graph to be processed is stored in the memory module, and the address of the first active node is stored in the start address register. The process of graph cuts is started by issuing an address to the address queue from the start address register. The address queue holds the issued addresses and sends them to the address manager. An address sent to the address manager is judged whether it is for one of the neighboring nodes of the node being processed in the pipeline stage, and if not, it is sent to the pipeline stage. In the memory module, data for the input address and data for its neighboring nodes are read out to the push-relabel unit. The push-relabel unit performs the Push operations and the Relabel operations on the fetched node data. Thereafter, the processed node

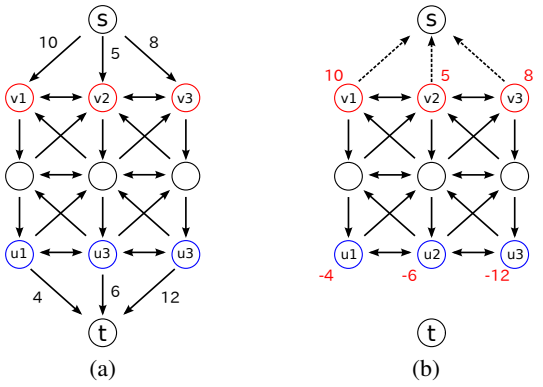


Figure 8: Graph pre-processing for data size reduction: (a) given graph and (b) optimized graph where capacities associated with s and t are eliminated

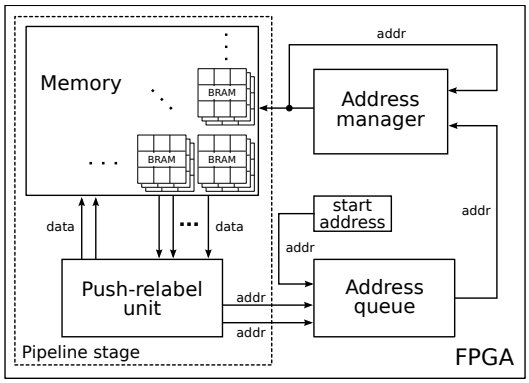


Figure 9: Overview of the implemented graph cut system

data and the node data to which the Push and Relabel operations were applied are written back to the memory module. If the processed node or the nodes to which the Push and Relabel operations were applied are active, addresses of those nodes are sent to the address queue. This process is performed until active nodes disappear, and finally the obtained maximum flow is output. The outline of each sub module is described below.

4.1 Push-relabel unit

As shown in Fig. 10, the push-relabel unit has 5-stage pipelined structure. In this unit, the push-relabel method with the gap-relabeling heuristics is executed. As pre-processing, v_m, which has the lowest height among the neighboring nodes v_i ($0 \leq i \leq 13$) that can accept a flow from node u, is chosen. For the gap-relabeling, a histogram of heights is provided as a register array. The processed data results are written back to the memory module. At the same time, if u and v_m are active nodes even after two nodes have been processed, their addresses are sent to the address queue.

The operation flow for the push-relabel unit is as follows.

Input: The active node u and its neighboring nodes v_i ($0 \leq i \leq 13$).

① Nodes that can accept a flow from u are detected. The neighboring nodes such that $c_f(u, v_i) \leq 0$ are treated as don't care.

② The node v_m which has the lowest height among v_i found in ① is determined.

③ The height $h(u)$ is changed to $h(v_m) + 1$.

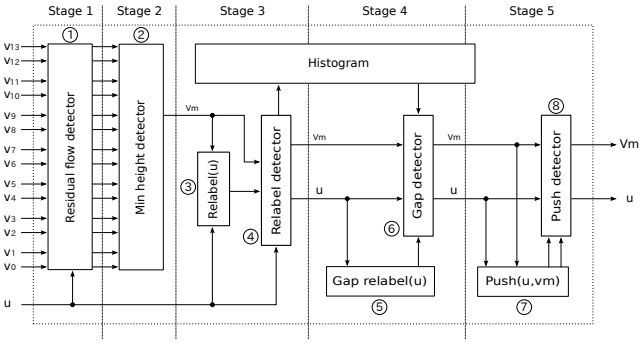


Figure 10: Push-relabel unit

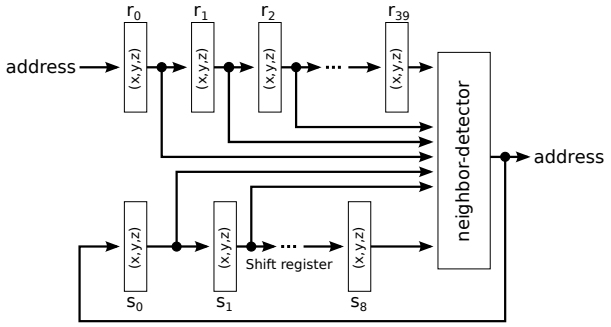


Figure 11: Address manager

④ It is determined whether the Relabel operation is applicable. If applicable, the value of relabel(u) is output. If not, value before the Relabel operation is output. When the Relabel operation is performed, the height of u in the histogram is also updated.

⑤ The height $h(u)$ is changed to $\max(h(u), |V| + 1)$.

⑥ If *gap* exists, the value of gap-relabel(u) is output. If not, the value before the Gap-relabel operation is output.

⑦ From u to v_m, flows are moved as much as possible.

⑧ It is determined whether the Push operation is applicable. If applicable, the value of push(u, v_m) is output. Otherwise, the value before the Push operation is output.

Output: u and v_m.

As Fig. 10 illustrates, the first, second, third, forth, and fifth pipeline stages of the push-relabel unit correspond to the operation ①, the operation ②, the operations ③ and ④, the operations ⑤ and ⑥, and the operations ⑦ and ⑧, respectively.

The gap-relabeling can be easily implemented by managing the histogram with registers that can be accessed in parallel. If there are 10 nodes of height k for example, a value of 10 is stored in the register for height k in the histogram. According to the preliminary software evaluation, it was confirmed that the maximum height does not exceed 1000. Thus, we provided 1000 registers for the height histogram in this implementation.

4.2 Address manager

It is not possible to process the neighboring nodes of the node being processed in the pipeline stage in parallel, since there is a possibility that $c_f(v, u)$ for those nodes will be changed due to the

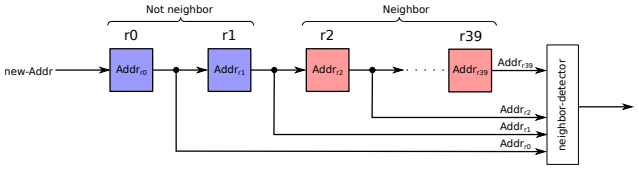


Figure 12: Selection of output address in the address manager

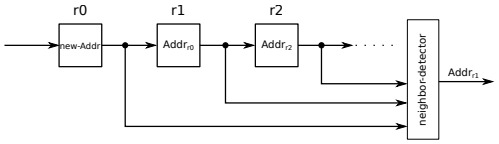


Figure 13: Address shift result after output selection in Fig. 12

Push and Relabel operations. The address manager prevents such neighboring nodes from entering the pipeline stage. Fig. 11 shows the structure of the address manager. The addresses of the node being processed in the pipeline stage is stored in the shift register s_i $(0 \le i \le 8)$. By feeding back the address to be output into s_0, the address currently being processed in the pipeline stage can be held. The addresses sent from the address queue are stored in the registers r_i $(0 \le i \le 39)$.

Let Addr_{r1}, x_{r1}, y_{r1}, and z_{r1} be the address, x-coordinate, y-coordinate, and z-coordinate of the node in the register $r1$, respectively. If Addr_{r_1} is an address of a neighboring node of the nodes under processed, that is,

$$|x_{r_1} - x_{s_i}| \le 2 \text{ and } |y_{r_1} - y_{s_i}| \le 2 \text{ and } |z_{r_1} - z_{s_i}| \le 2 \ (0 \le i \le 8), \quad (5)$$

Addr_{r_1} keeps staying in the module. Otherwise, Addr_{r_1} is chosen as output.

Fig. 12 shows how the address to be output is selected. When there are multiple registers that hold addresses for non-neighboring nodes of the nodes under processed, the address held in the register with the largest number is selected (the priority of r_{39} is the highest). Therefore, Addr_{r_1} is selected as the output address in Fig. 12. Fig. 13 shows how addresses are shifted after Addr_{r1} is selected as an output address. Addr_{r_0} moves to register r_1, and input address is stored in the register r_0.

4.3 Memory management

Given graph data is stored in 27 memory banks composed of BRAM. Fig. 14 shows how graph data is divided, where each small squares in the left side represent one node, which has data of 143 bits. In order to speed up the memory access, it is desired to be able to simultaneously read out data for one node to be processed as well as its 14 neighboring nodes. In addition, two data items processed by the push-relabel unit should be written back at the same time. In our implementation, graph data was divided into $3 \times 3 \times 3$ regions as shown in Fig. 14. A total of 27 nodes form one group, and each node is assigned to different memory banks ('a' to 'A'). The node coordinate (x, y, z) is indicated by the bank position $(x\%3, y\%3, z\%3)$. The node at the bank position $(0, 0, 0)$ is stored in the bank 'a'. Similarly, the node at the bank position $(0, 1, 0)$ is stored in the bank 'd', while the node at the bank position $(0, 0, 1)$ goes to the bank 'j'. Since each memory bank is implemented with dual-port BRAM, the 27 nodes can be accessed in parallel.

For the nodes in the same group, the address of each memory bank becomes the same. Therefore, as shown in Fig. 15, when Addr_0 of bank 'n' is selected as a node to be processed by the push-relabel unit, its 14 neighboring nodes are read out from Addr_0 of

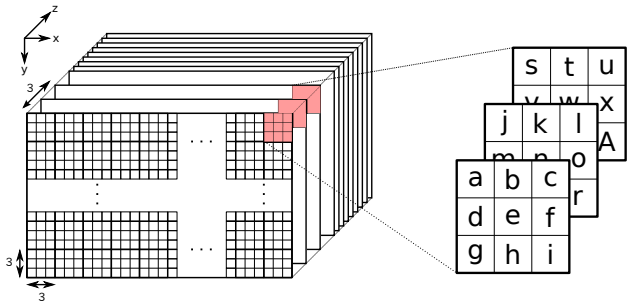


Figure 14: Memory bank structure

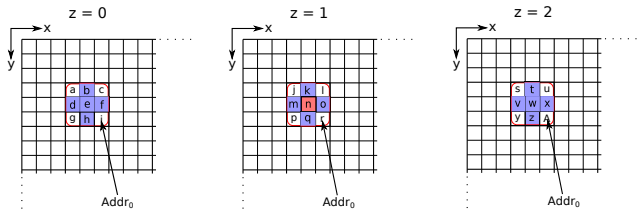


Figure 15: Memory access pattern with the same address

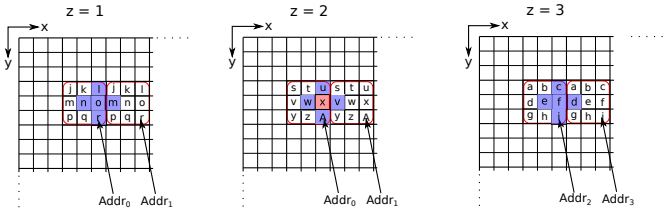


Figure 16: Memory access pattern with different addresses

the banks 'b', 'd', 'e', 'f', 'h', 'k', 'm', 'o', 'q', 't', 'v', 'w', 'x', and'z'. If Addr_0 of the bank 'x' is selected as the next node to be processed as shown in Fig. 16, the neighboring nodes are accessed with Addr_0 of the banks 'l', 'n', 'o', 'r', 'u', 'w', and 'A', Addr_1 of the banks 'm' and 'v', Addr_2 of the bank 'c','e','f',and 'i', and Addr_3 of the bank 'd'.

5. EVALUATION AND DISCUSSIONS

FPGA implementation of the proposed system was carried out in RTL description using Verilog-HDL. Vivado 2016.4 was used as a synthesis tool, and Virtex UltraScale xcvu095-ffva2104-2-e-es2 was used as the target FPGA. We implemented two patterns of address queue architectures, to evaluate how a selection method of active nodes influences the performance. In the first pattern, the selection is performed in an FIFO manner. Three addresses are stored in the queue simultaneously, and addresses are output one by one. In the second pattern, the address queue illustrated in Fig.17 is implemented. The addresses sent from the push relabel unit are stored in Queue 1 and Queue 2. The address sent from the start address is selected and output with the highest priority. When no address is sent from start address register, the address in the Queue 1 or the Queue 2 is selected. Once the Queue 1 or Queue 2 is selected, it is continued to be selected until the queue becomes empty.

Table 1 shows the implementation results of the system for graphs with up to $129 \times 129 \times 16$ nodes. While the Pattern 2 required slightly more resources than Pattern 1, only a small difference was

Table 2: Performance comparison

Graph		Execution time (msec)			Frames per second			Speedup to CPU	
No.	Max-flow	CPU	Pattern 1	Pattern 2	CPU	Pattern 1	Pattern 2	Pattern 1	Pattern 2
#1	554,763	550	461	148	1.82	2.17	6.76	1.19	3.72
#2	520,542	530	542	328	1.89	1.85	3.05	0.98	1.62
#3	680,495	540	238	123	1.85	4.20	8.13	2.27	4.39
Average		540.0	413.7	199.7	1.85	2.42	5.01	1.31	2.70

Table 1: FPGA mapping results

Resource	Pattern 1		Pattern 2	
	Used	Used (%)	Used	Used (%)
LUT	83,752	15.58	86,637	16.12
FF	31,101	2.89	31,393	2.92
BRAM	1,317	76.22	1,321	76.45

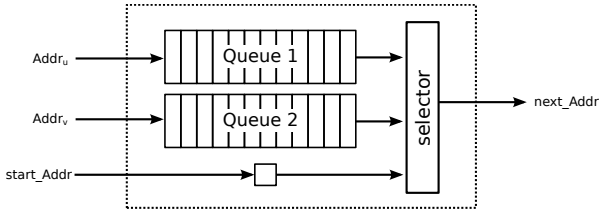


Figure 17: Address queue in Pattern 2 architecture

observed between the two patterns of implementation. For both implementation, the operational frequency of 95 MHz was achieved. The critical path was a path from the register in the address manager which stores the address of the node being processed to the judging logic of neighboring nodes.

Table 2 summarizes performance evaluation results of graph cuts for three kinds of 3D grid graphs. Each graph was constructed using actual images obtained by the stereo camera. For comparison, a software program implemented with the 'maxflow-v3.00' graph cut library [11] was executed on a Linux machine equipped with a 3-GHz Intel Core i7-5960X CPU.

As can be seen in Table 2, large differences in the execution time between Pattern 1 and Pattern 2 were observed, suggesting the method of selecting the active nodes is quite important. Compared to the CPU, Pattern 2 achieved performance improvement by up to 3.7 times, and its maximum frame rate was approximately 8 fps. Fig. 18 shows some different views of the reconstructed 3D information obtained by the graph cuts of Graph #1.

As Table 1 shows, approximately 70 % of the usable amount of BRAM was used for the memory unit, and this restricts the size of operational graphs to $129 \times 129 \times 16$ nodes. In terms of scalability for lager graphs, the configuration of the memory should be modified. For example, use of an external DRAM with a cache mechanism with on-chip memory is a promising approach. By using large size on-chip memory like URAM, relatively large access latencies of external DRAM would be mitigated. For further performance improvement, providing multiple pipelines in the push-relabel unit will be worthy to try.

6. CONCLUSION

In this paper, we presented an FPGA accelerator of a graph cut algorithm for 3D grid graphs for stereo vision. The push-relabel method with gap-relabeling heuristic was pipelined on an FPGA.

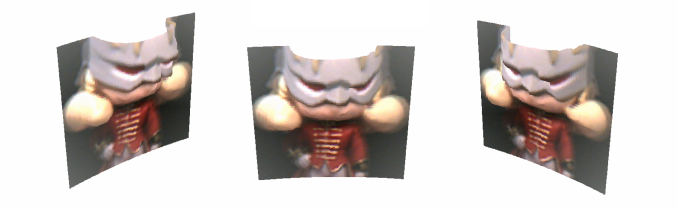


Figure 18: Example views of 3D reconstruction results

The system can perform graph cuts for 3D grid graphs with up to $129 \times 129 \times 16$ nodes and this size was restricted by the BRAM capacity. The average execute time for actual stereo vision graphs was approximately 200 ms, which corresponds to 5 fps. This was approximately 2.7 times better than software implementation. Further performance and scalability improvement using multiple pipelines and external memory with an on-chip cache mechanism is our future work. In addition, we will analyze the selection mechanism of active nodes and optimize for stereo vision graphs.

7. REFERENCES

[1] B. M. Smith, L. Zhang, and H. Jin, "Stereo matching with nonparametric smoothness priors in feature space," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 485–492, 2009.

[2] O. Woodford, P. Torr, I. Reid, and A. Fitzgibbon, "Global stereo reconstruction under second-order smoothness priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2115–2128, 2009.

[3] Paper information is eliminated for double-blind reviewing.

[4] V. Vineet and P. Narayanan, "CUDA cuts: Fast graph cuts on the GPU," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CUPRW)*, pp. 1–8, 2008.

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

[6] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[7] E. Dinits, "Algorithm of solution to problem of maximum flow in network with power estimates," *Doklady Akademii Nauk SSSR*, vol. 194, no. 4, p. 754, 1970.

[8] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.

[9] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM (JACM)*, vol. 35, no. 4, pp. 921–940, 1988.

[10] U. Derigs and W. Meier, "Implementing Goldberg's max-flow-algorithm—A computational investigation," *Zeitschrift für Operations Research*, vol. 33, no. 6, pp. 383–403, 1989.

[11] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 147–159, 2004.