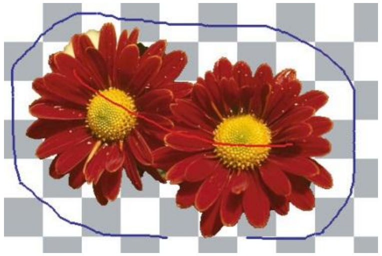
An Acceleration of a Graph Cut Segmentation With FPGA

Daichi Kobori and Tsutomu Maruyama
University of Tsukuba

What Is Graph Cut Segmentation?

- Graph cut is one of the segmentation methods based on energy minimization, and graph cut based segmentation is widely used.
- The following images are examples of segmentation [*].
- The seed pixels (target objects or background) are given by the user, and then only the target objects are extracted.

Input image with seeds



Output image

[*] Tomoyuki Nagahashi, Hironobu Fujiyoshi, and Takeo Kanade, “Image Segmentation Using Iterated Graph Cuts Based on Multi-scale Smoothing,” in *ACCV 2007, Part II, LNCS 4844*, pp. 806--816, 2007.

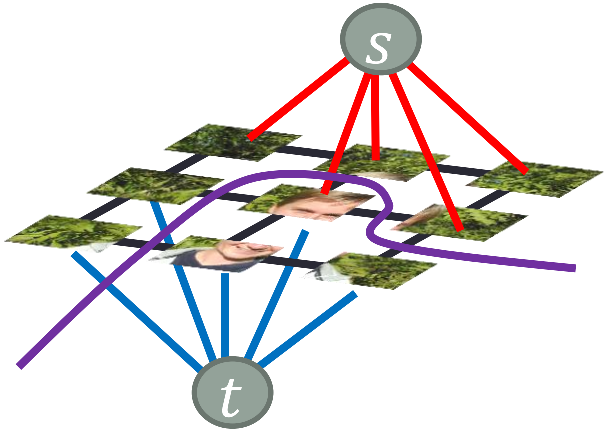
Background

- For calculating the graph cut, max-flow algorithm is widely used, but it requires long computation time.
- We need an acceleration by FPGA or GPU for real-time processing of the max-flow algorithm.
- The performance of a GPU (GeForce GTX280) system [7] is 25 graph cuts per second on 640 x 480 pixel images, which is about 5 times faster than CPU.

[7] V. Vineet and P. J. Narayanan, “Cudacuts: Fast graph cuts on the gpu,” in *CVPR Workshop on Visual Computer Vision on GPUs*, 2008.

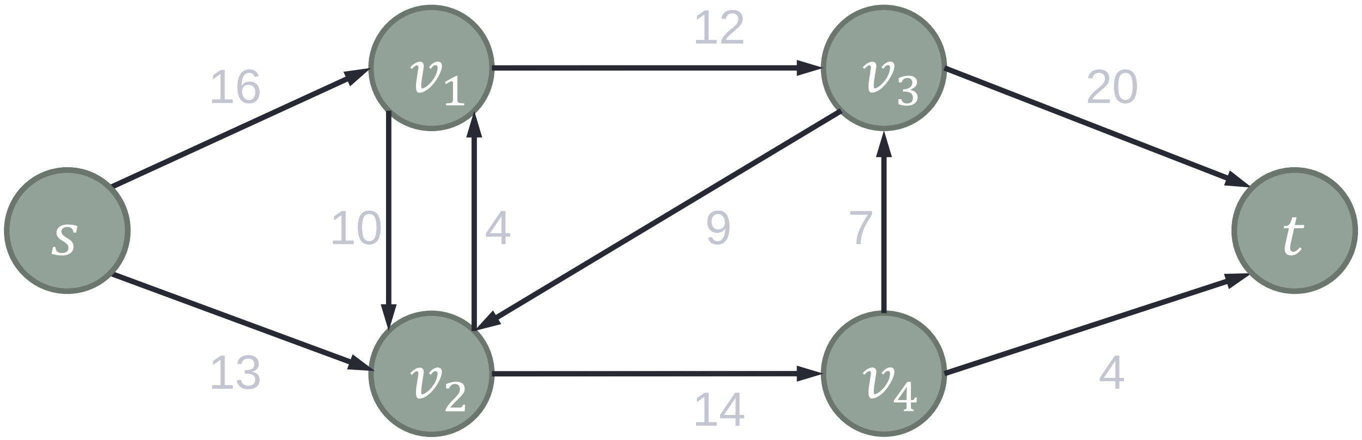
Segmentation Procedure

- Seed pixels (on objects or background) are specified by the user.
- A weighted directed graph among the pixels in the image is generated based on the seed pixels.
- A min-cut of the weighted directed graph is calculated using max-flow algorithm.



A Graph

- $G = (V, E)$: a weighted directed graph
  - V is a set of vertices (pixels), and it includes two special nodes, s and t.
  - E is a set of edges between two vertices, and each edge has a non-negative capacity $c(u, v)$.

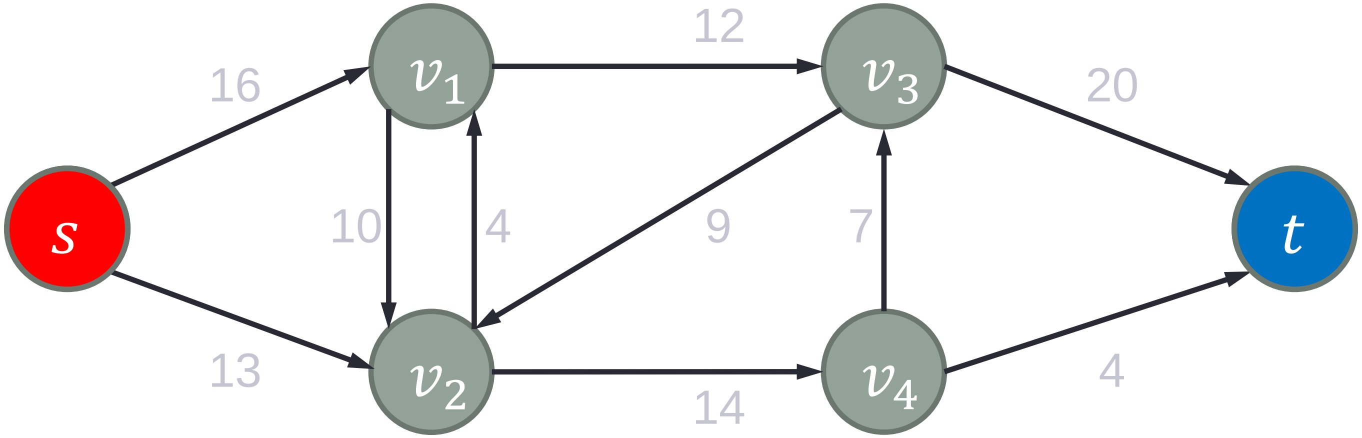


A Graph

- $G = (V, E)$: a weighted directed graph

  V is a set of vertices (pixels), and it includes two special nodes, s and t.

  E is a set of edges between two vertices, and each edge has a non-negative capacity $c(u, v)$.

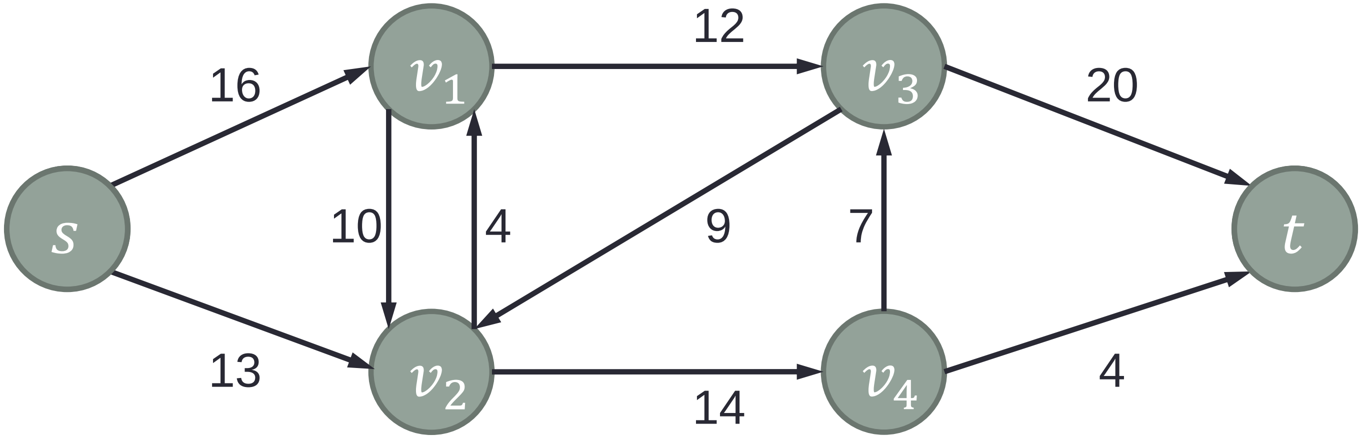


A Graph

- $G = (V, E)$: a weighted directed graph

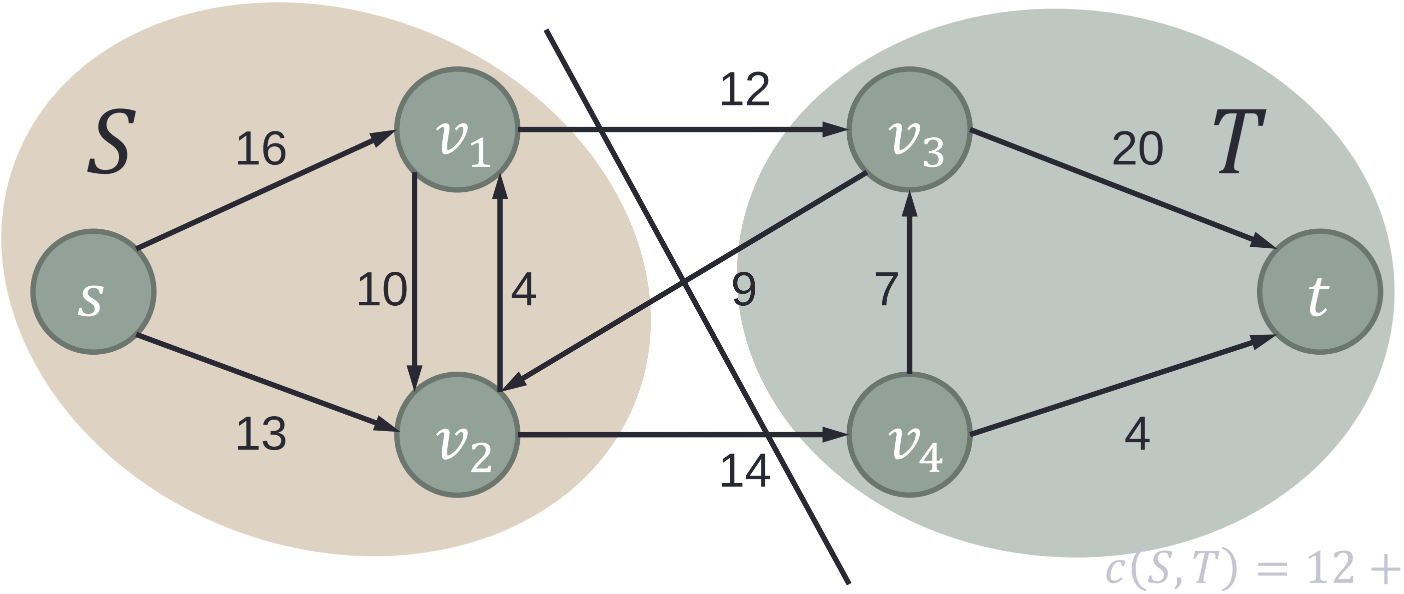
  V is a set of vertices (pixels), and it includes two special nodes, s and t.

  E is a set of edges between two vertices, and each edge has a non-negative capacity $c(u, v)$.



Cut of a Graph

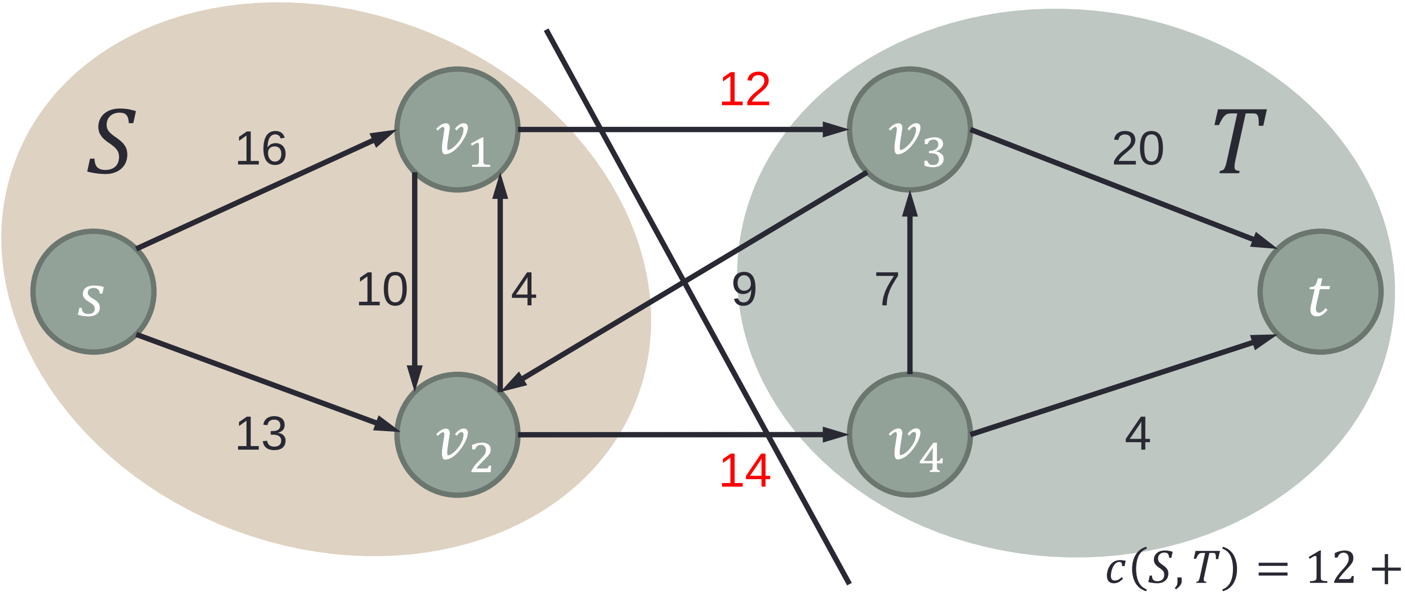
- A cut of the graph shows the division of V into two groups; S and T.
- The capacity of the cut $c(S,T)$ is defined as the capacity of the edges from S to T.
- The cut which minimizes $c(S,T)$ is called min-cut.



$c(S,T) = 12 + 14 = 26$

Cut of a Graph

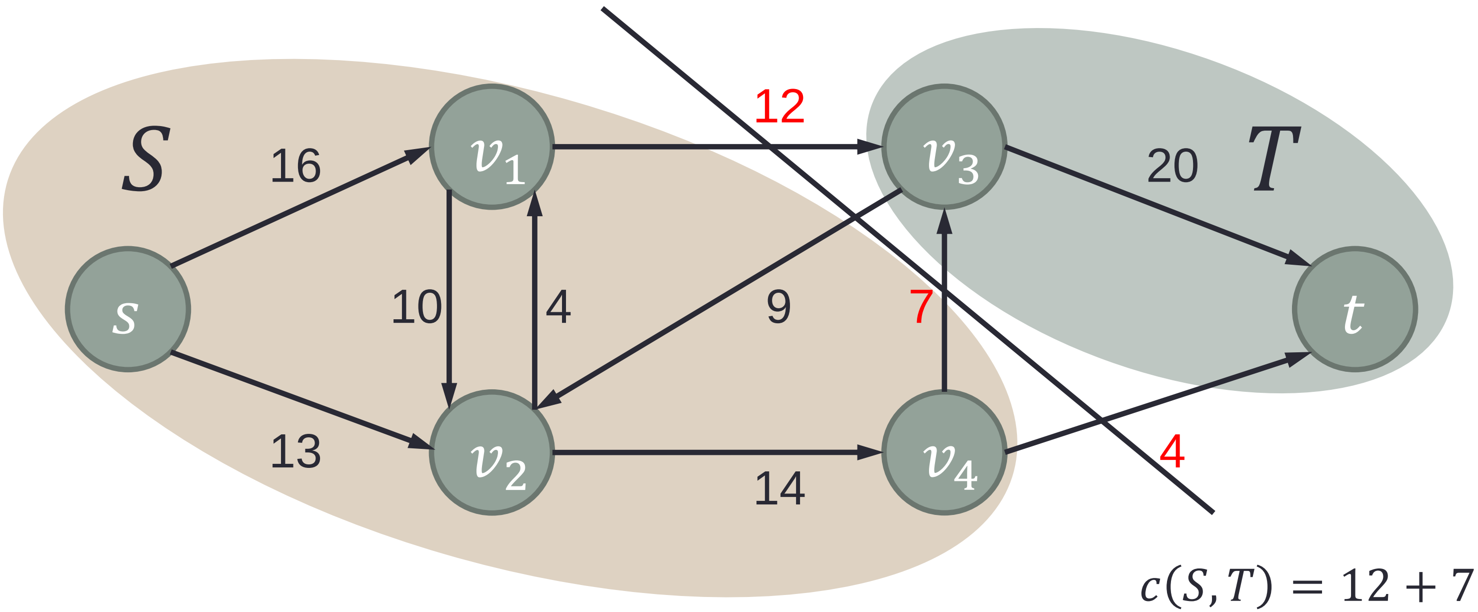
- A cut of the graph shows the division of V into two groups; S and T.
- The capacity of the cut $c(S,T)$ is defined as the capacity of the edges from S to T.
- The cut which minimizes $c(S,T)$ is called min-cut.



$c(S,T) = 12 + 14 = 26$

Cut of a Graph

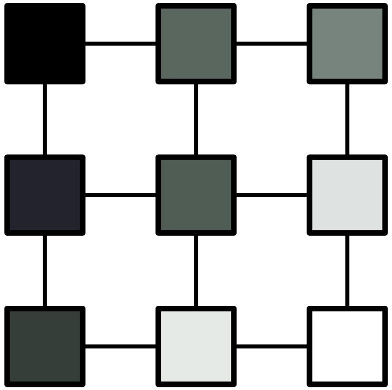
- A cut of the graph shows the division of V into two groups; S and T.
- The capacity of the cut $c(S,T)$ is defined as the capacity of the edges from S to T.
- The cut which minimizes $c(S,T)$ is called min-cut.



$c(S,T) = 12 + 7 + 4 = 23$

Making a Graph (Color)

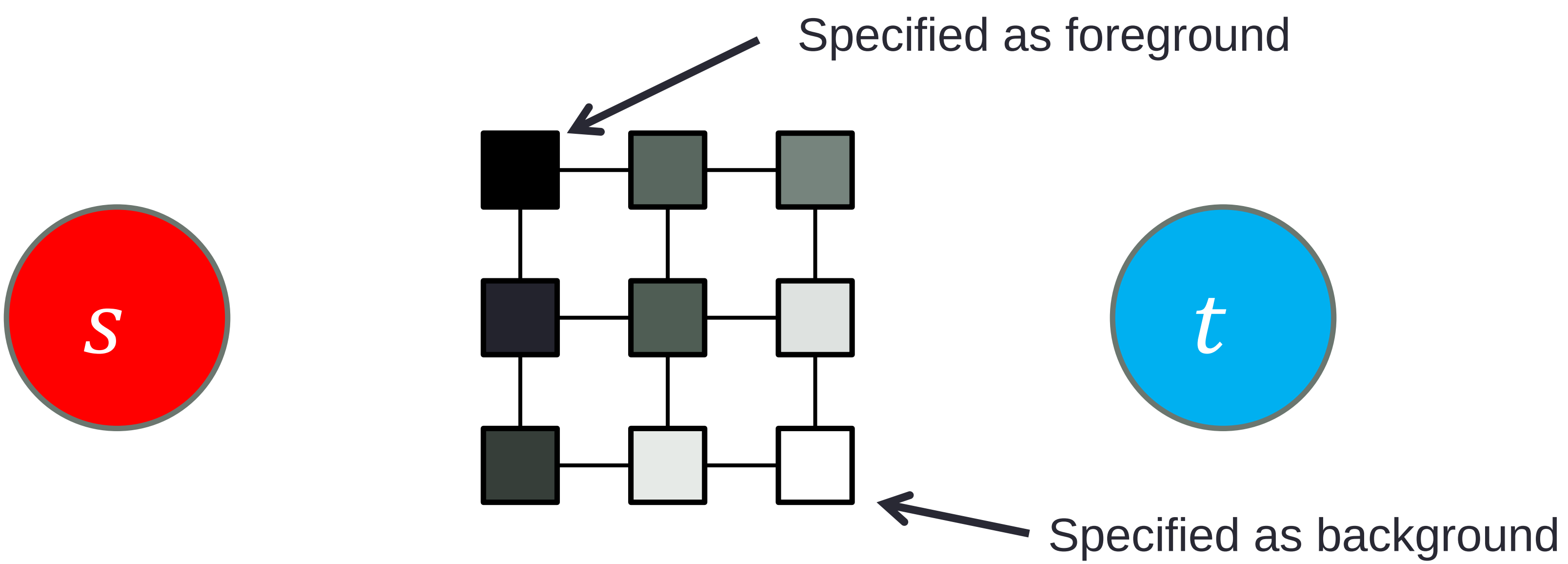
- The weighted directed graph is generated from the pixels in the image.



image

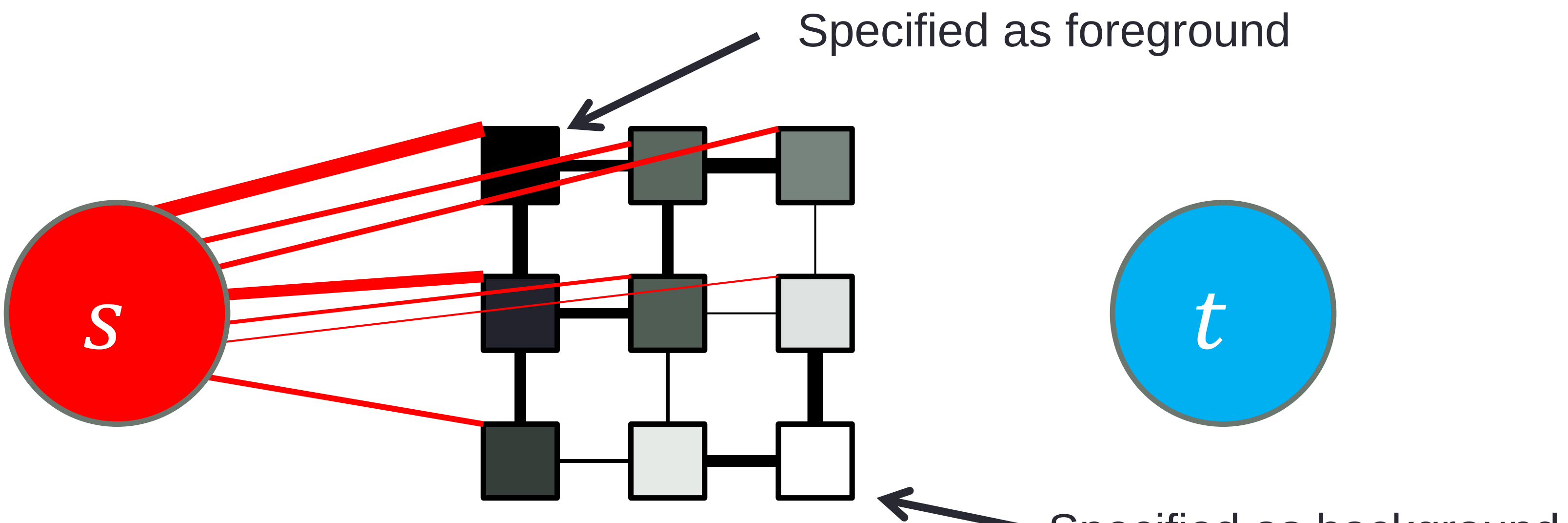
Making a Graph (Seed)

- Suppose that a black pixel is specified as foreground, and a white pixel is specified as background.
- Then, pixels that have similar color to black have strong connection to s.
- On the other hand, pixels that have similar color to white have strong connection to t.



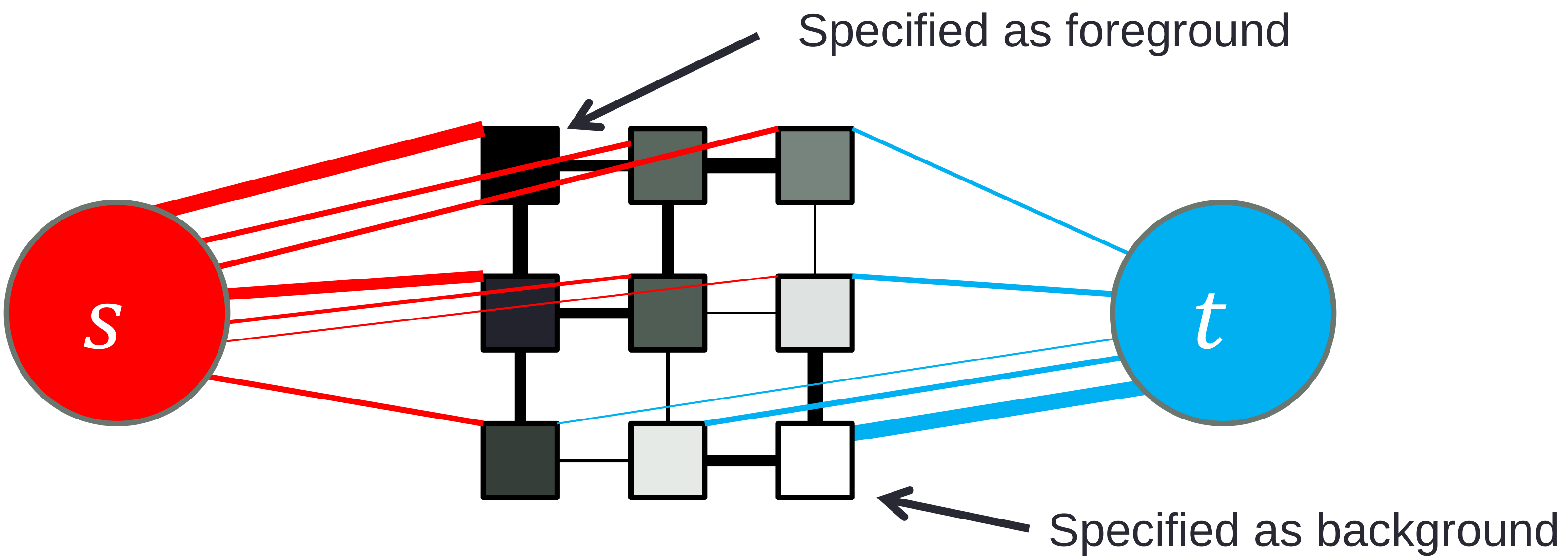
Making a Graph (Seed)

- Suppose that a black pixel is specified as foreground, and a white pixel is specified as background.
- Then, pixels that have similar color to black have strong connection to s.
- On the other hand, pixels that have similar color to white have strong connection to t.



Making a Graph (Seed)

- Suppose that a black pixel is specified as foreground, and a white pixel is specified as background.
- Then, pixels that have similar color to black have strong connection to s.
- On the other hand, pixels that have similar color to white have strong connection to t.



Making a Graph (Energy)

- Min-cut corresponds to the minimum energy of the following equation.

$$E(\boldsymbol{L}) = \lambda \sum_{p \in V} R_p(L_p) + \sum_{\{p,q\} \in E} B_{\{p,q\}} \cdot \delta(L_p, L_q)$$

- λ is a parameter which controls the affect by the seeds (the larger the value, the more affect by the seeds).

How to Compute Min-cut

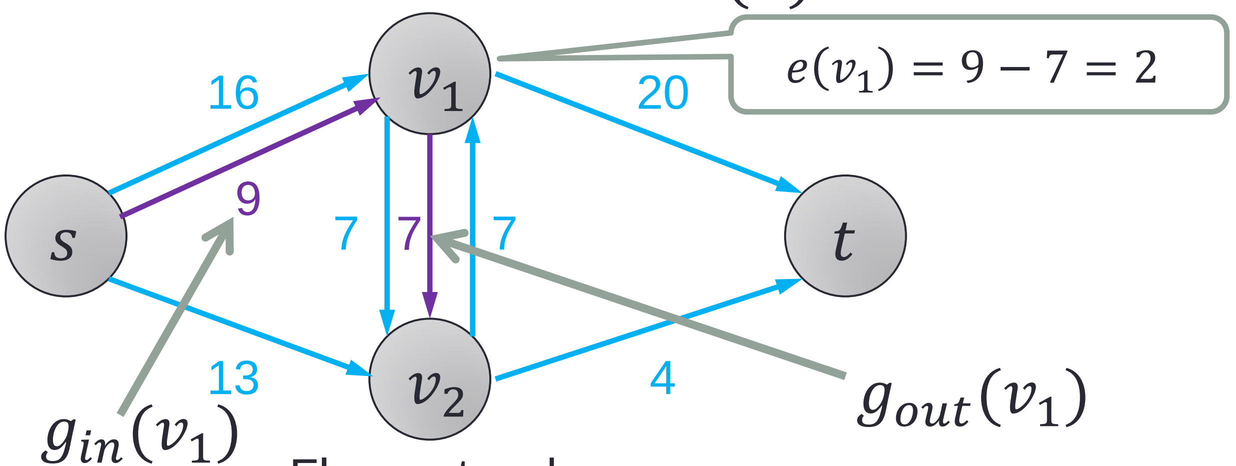
- According to “max-flow min-cut theorem”, min-cut is obtained from the result of max-flow.
- In order to calculate max-flow, two methods are commonly used.
  1. “augmenting path method” scans the graph to find a path from source (s) to sink (t).
     This method is NOT suitable for hardware implementation.
  2. “push-relabel method” uses only the connection from one vertex to its neighbors.
     This method is suitable for hardware implementation.

Push-relabel Method

- In the push-relabel method, a weighted directed graph is considered as a flow network.
- We can flow preflow g in each edge if g is smaller than flow capacity $c(u,v)$.
- All vertices have excess flow

$$e(u) = g_{in}(u) - g_{out}(u) \geq 0.$$

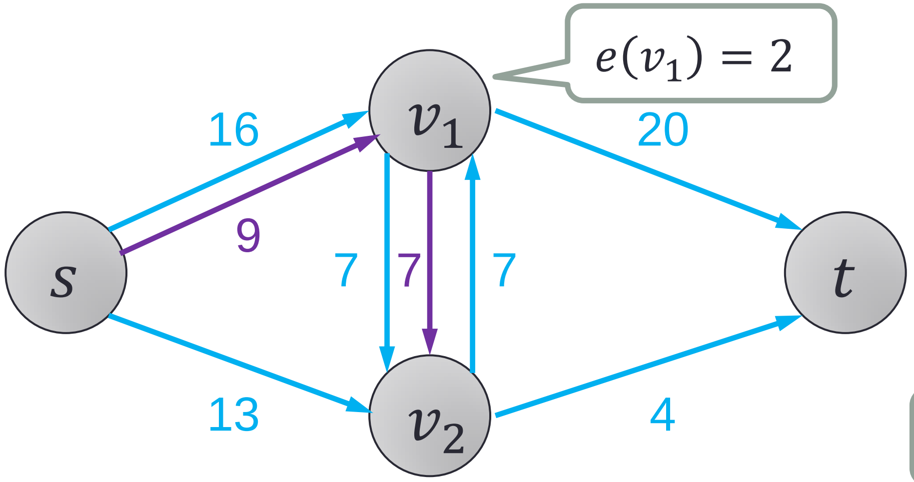
- Vertex u is active if $e(u) > 0$.



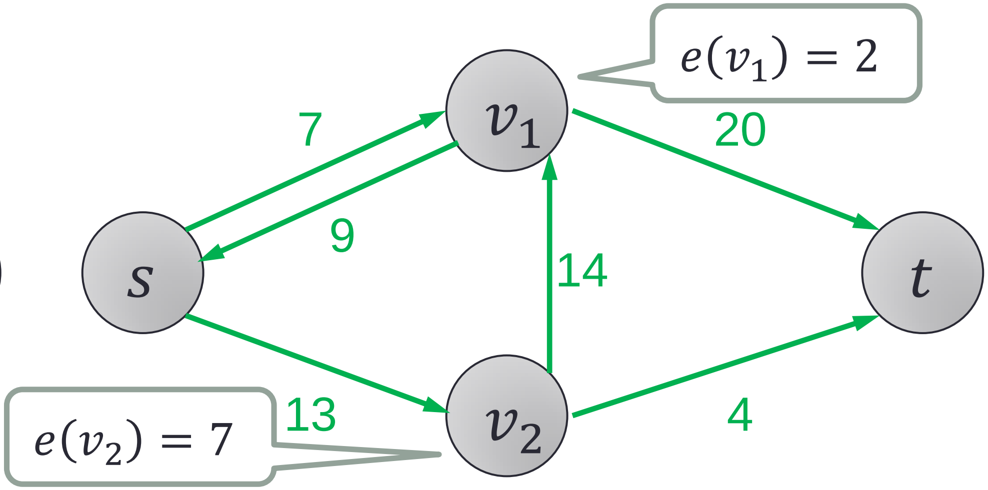
Flow network

Residual Network

- The residual capacity of an edge is given by
$$c_f(u, v) = c(u, v) - g(u, v)$$
which is the rest of the capacity that we can flow from u to v.
- By flowing 7 from v_1 to v_2,
$$c_f(v_1, v_2) = 7 - 7 = 0$$
$$c_f(v_2, v_1) = 7 - (-7) = 14$$



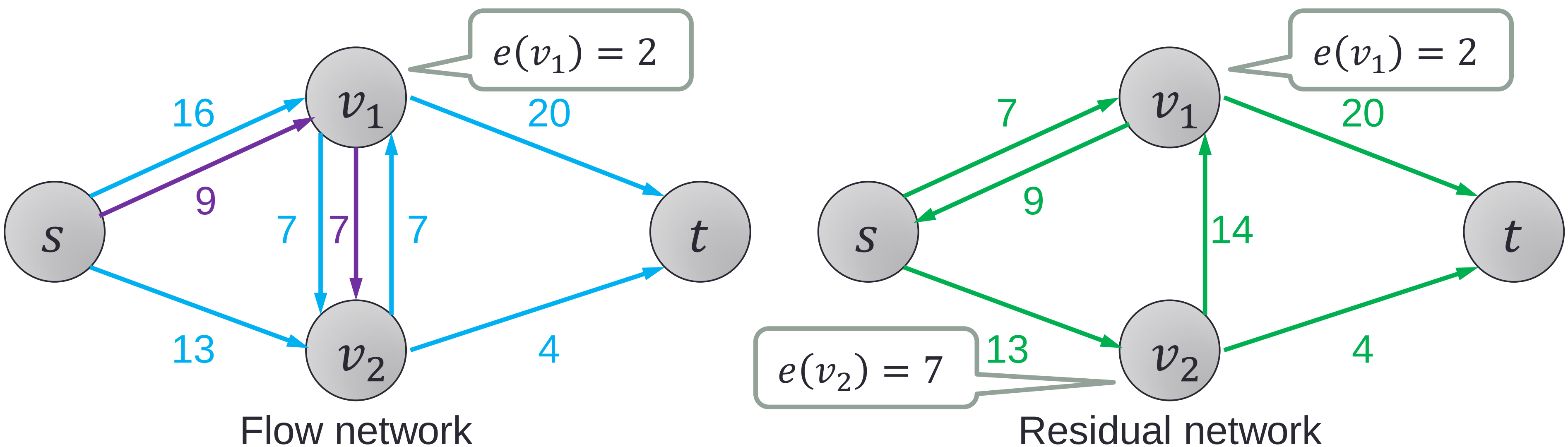
Flow network



Residual network

Residual Network

- Using the residual network, we can easily understand how much more we can flow on the network.
- However, we must store excess flow of each vertex.

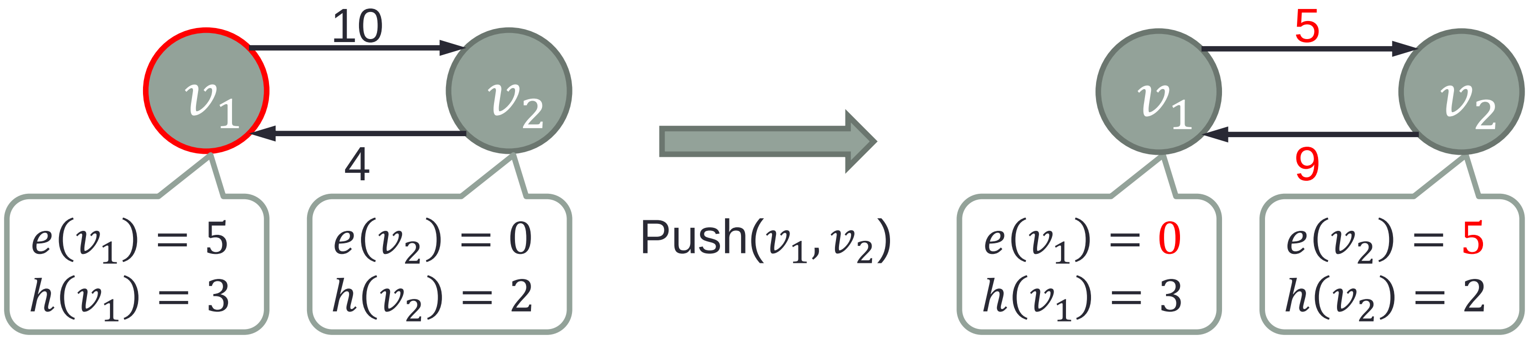


Operations of Push-relabel Method

- There are two main operations, and they are applied to the active vertices.
  1. Push(u, v)
  2. Relabel(u)
- If u is active, either operation can be applied to u.

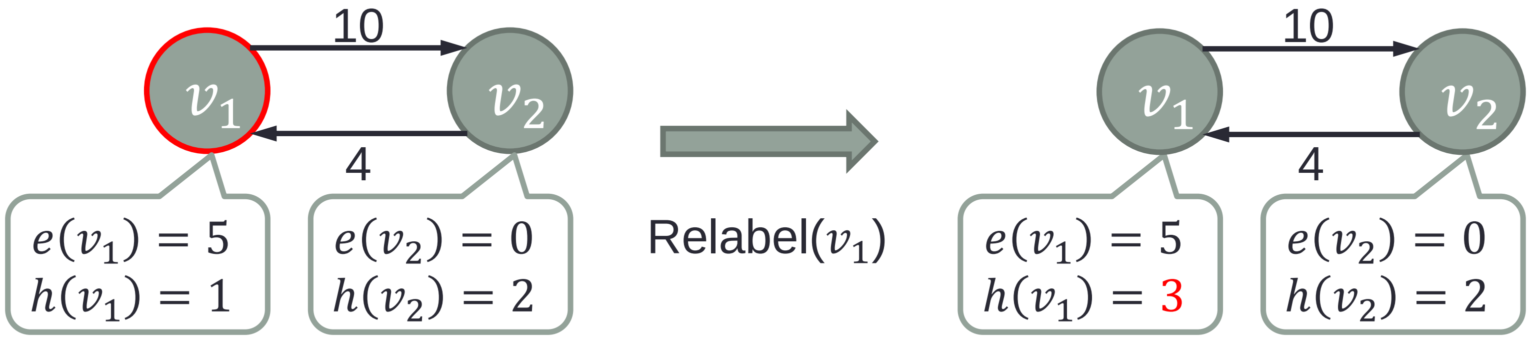
Push(u, v)

- Applicable condition
  - Vertex u is active.
  - $c_f(u, v) > 0$
  - $h(u) = h(v) + 1$
- Operation
  - $\min(e(u), c_f(u, v))$ is flowed from u to v.
- Example
  - Preflow 5 is flowed from v_1 to v_2.
  - Residual capacity $c_f(v_1, v_2)$ is reduced, and $c_f(v_2, v_1)$ is increased.



Relabel(u)

- Applicable condition
  - Vertex u is active.
  - Push(u, v) cannot be applied to vertex u.
- Operation
  - $h(u)$ is heightened so that push(u, v) can be applied.
- Example
  - Push(v_1, v_2) can not applied to v_1 because $h(v_1) < h(v_2)$.
  - $h(v_1)$ is heightened more than $h(v_2)$ so that push(v_1, v_2) can be applied to v_1.



Heuristics for the Push-relabel Method

- The computational complexity of the push-relabel method is $O(V^2E)$.
- To reduce the computational complexity, two heuristics are widely used.

- "global relabeling" changes $h(u)$ by calculating the minimum distance from u to t by the breadth first search. We need to traverse the graph by dereferencing, so it is NOT suitable for hardware implementation.
- "gap relabeling" heightens $h(u)$ to $|V|+1$ if u belongs to S. This method can be implemented using a histogram.

Heuristics for the Push-relabel Method

- The computational complexity of the push-relabel method is $O(V^2E)$.
- To reduce the computational complexity, two heuristics are widely used.

- “global relabeling” changes $h(u)$ by calculating the minimum distance from u to t by the breadth first search. We need to traverse the graph by dereferencing, so it is NOT suitable for hardware implementation.
- “gap relabeling” heightens $h(u)$ to $|V|+1$ if u belongs to S. This method can be implemented using a histogram.

Heuristics for the Push-relabel Method

- The computational complexity of the push-relabel method is $O(V^2E)$.
- To reduce the computational complexity, two heuristics are widely used.

- “global relabeling” changes $h(u)$ by calculating the minimum distance from u to t by the breadth first search. We need to traverse the graph by dereferencing, so it is NOT suitable for hardware implementation.
- “gap relabeling” heightens $h(u)$ to $|V| + 1$ if u belongs to S. This method can be implemented using a histogram.

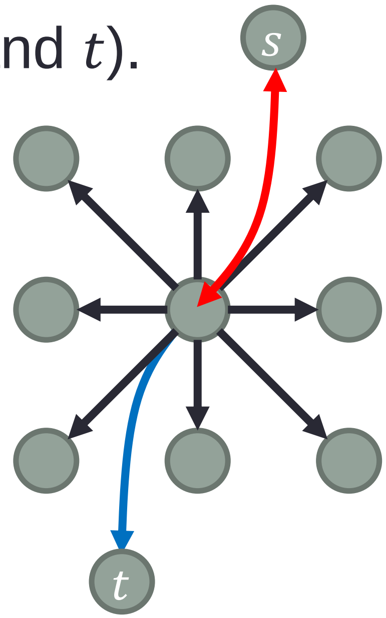
Main Features in Our Approach

- Major operations are “push” and “ralabel”.
- Operations are applied to the active vertices.
- Relabel is applied first if necessary, and then push is applied.
- A FIFO is used to manage active vertices, because the order of the processing is arbitrary.

- We can obtain max-flow of the flow network when there exists no active vertex.

Hardware Implementation

Data Format of Each Pixel

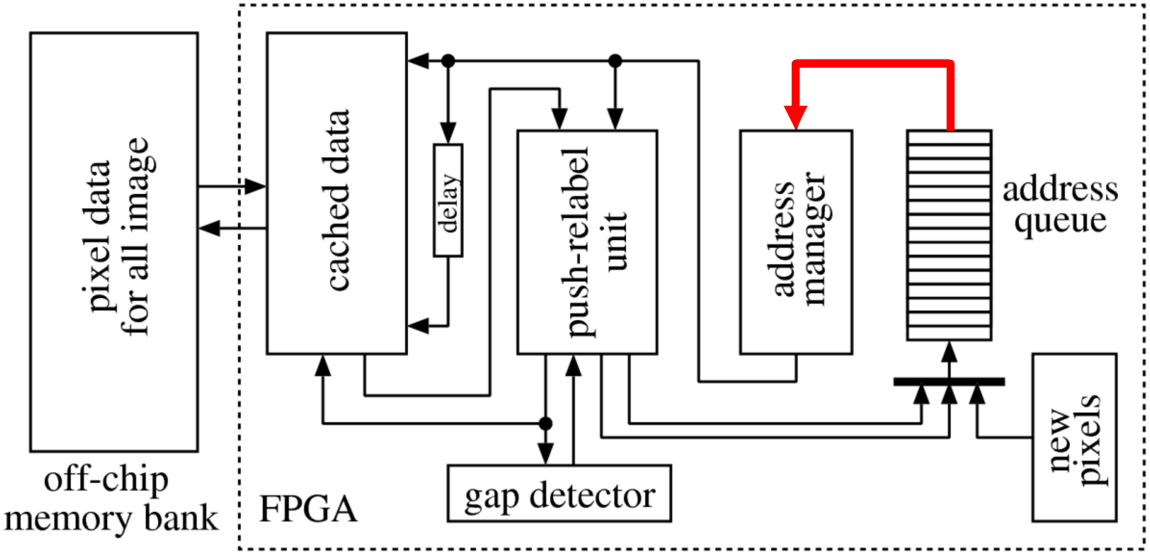


- Each vertex u has 10 links (eight neighbors, and s and t).
- Each link has residual capacity $c_f(u,*)$ from u.
- Vertex u also has residual capacity $c_f(s,u)$ from s to u.
- Excess flow $e(u)$ and height $h(u)$ are required for each vertex u.
- The total data width is 126b.

e(u)	h(u)	cf(u,n0)	cf(u,n1)	·····	cf(u,n7)	cf(u,s)	cf(u,t)	cf(s,u)
13	10	8	8		8	13	13	13

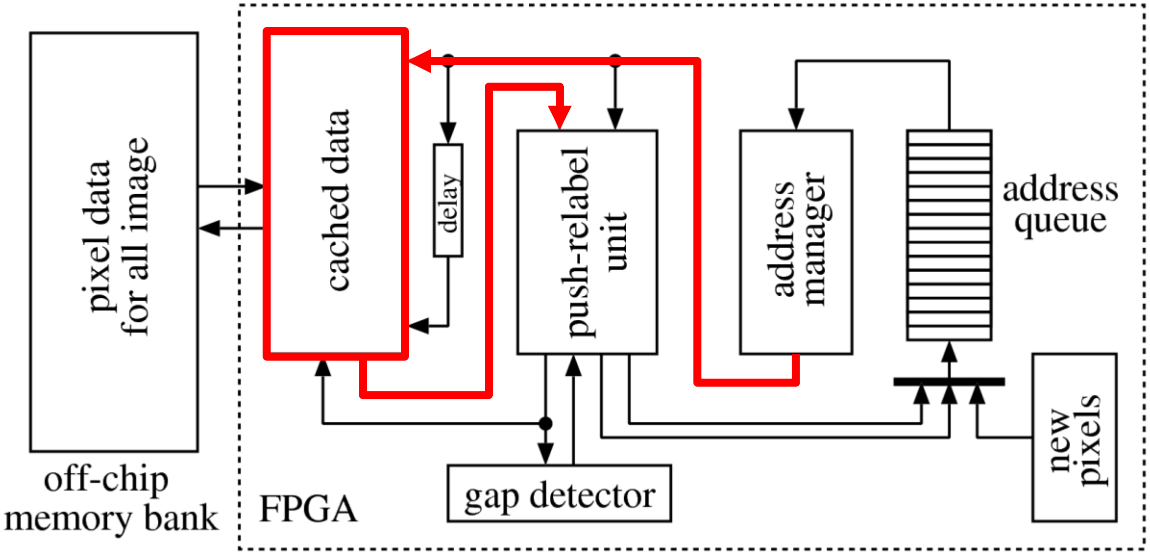
A Block Diagram of the Circuit

1. The address of an active vertex is popped up from the address queue.
2. The data of the nine pixels are read out from the cache memory.
3. Relabel operation is applied if necessary, and push operation is applied in the push-relabel unit.
4. If new active vertex is generated, put it in the address queue.
5. The result is written back to the cache memory.



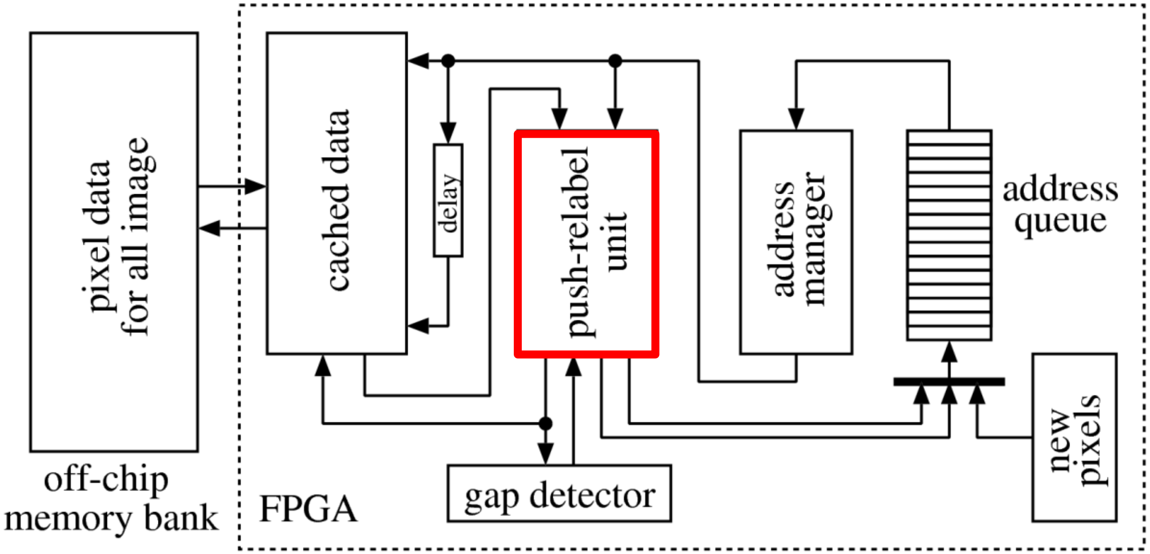
A Block Diagram of the Circuit

1. The address of an active vertex is popped up from the address queue.
2. The data of the nine pixels are read out from the cache memory.
3. Relabel operation is applied if necessary, and push operation is applied in the push-relabel unit.
4. If new active vertex is generated, put it in the address queue.
5. The result is written back to the cache memory.



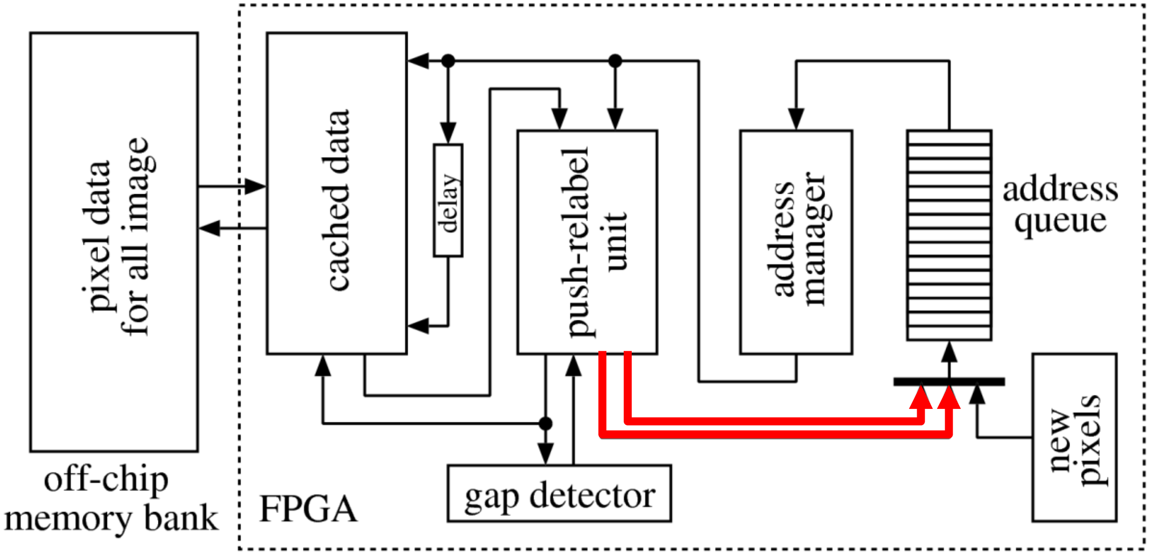
A Block Diagram of the Circuit

1. The address of an active vertex is popped up from the address queue.
2. The data of the nine pixels are read out from the cache memory.
3. Relabel operation is applied if necessary, and push operation is applied in the push-relabel unit.
4. If new active vertex is generated, put it in the address queue.
5. The result is written back to the cache memory.



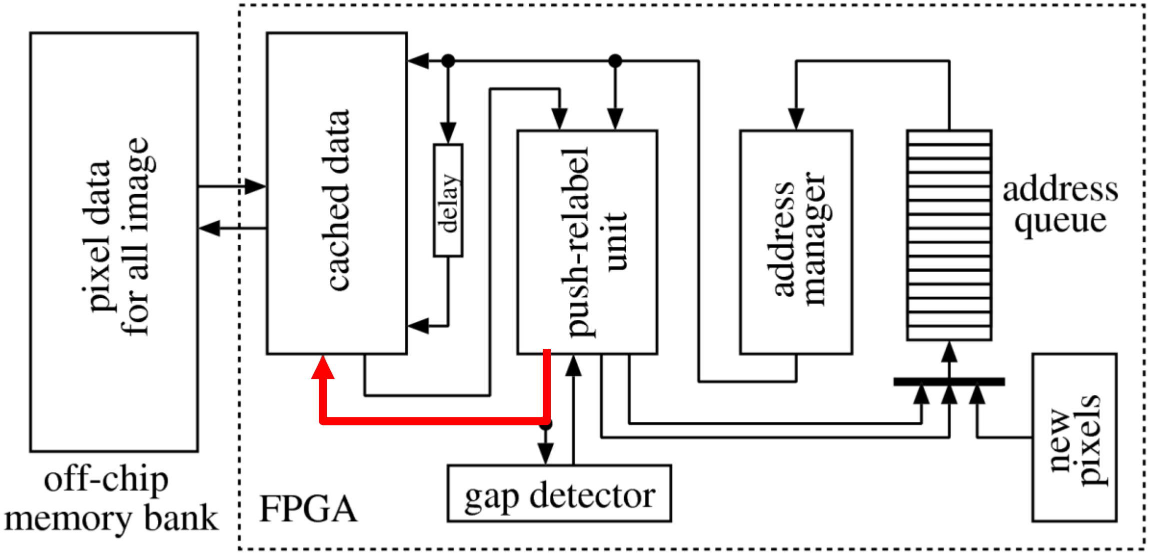
A Block Diagram of the Circuit

1. The address of an active vertex is popped up from the address queue.
2. The data of the nine pixels are read out from the cache memory.
3. Relabel operation is applied if necessary, and push operation is applied in the push-relabel unit.
4. If new active vertex is generated, put it in the address queue.
5. The result is written back to the cache memory.



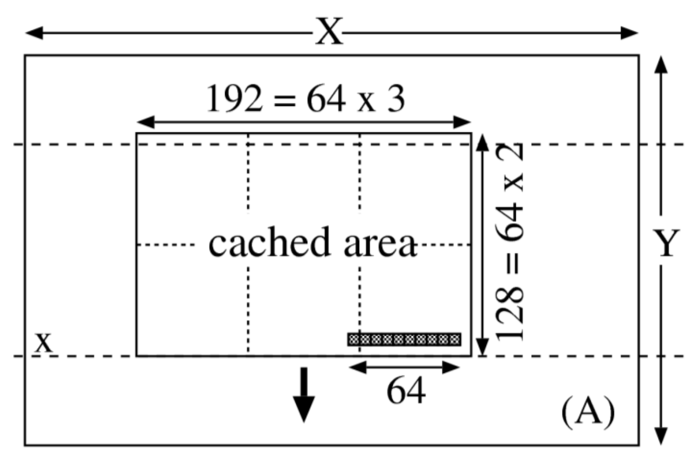
A Block Diagram of the Circuit

1. The address of an active vertex is popped up from the address queue.
2. The data of the nine pixels are read out from the cache memory.
3. Relabel operation is applied if necessary, and push operation is applied in the push-relabel unit.
4. If new active vertex is generated, put it in the address queue.
5. The result is written back to the cache memory.



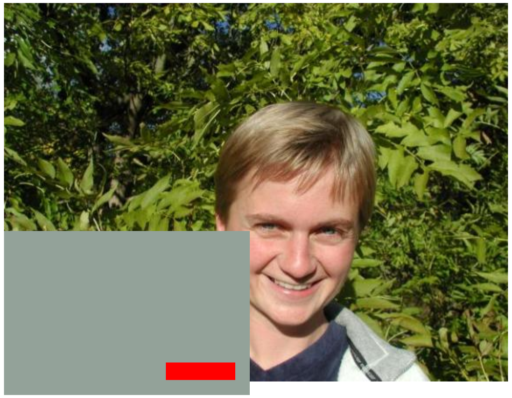
Data Caching Method

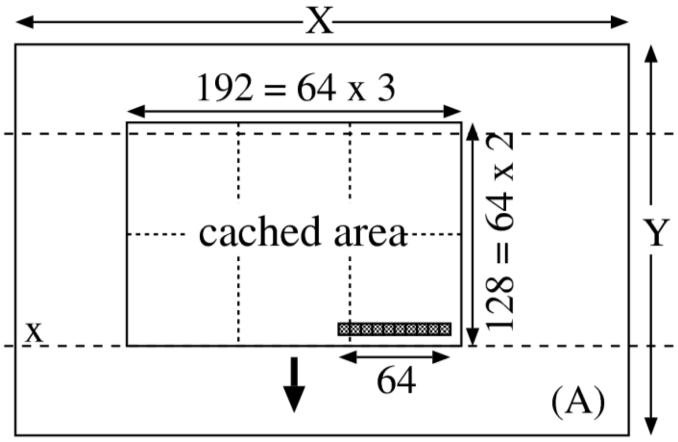
- 192 x 128 pixels are cached on block RAMs.
- The cached area is changed.
- Among the cached pixels, 64 pixels are newly processed.



Data Caching Method

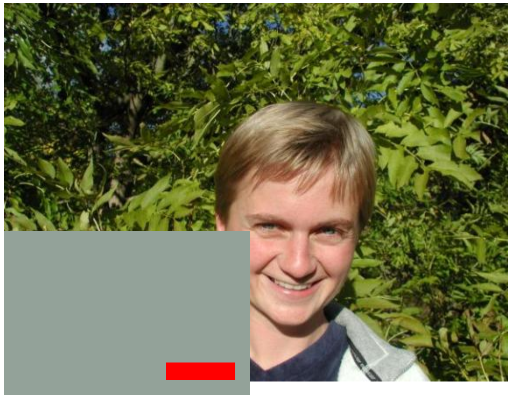
- 192 x 128 pixels are cached on block RAMs.
- The cached area is changed.
- Among the cached pixels, 64 pixels are newly processed.

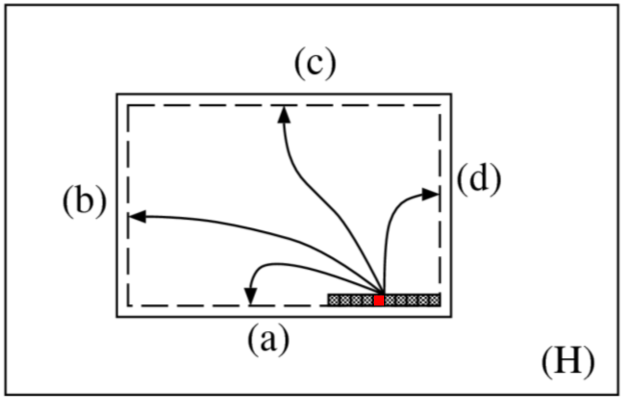




Data Caching Method

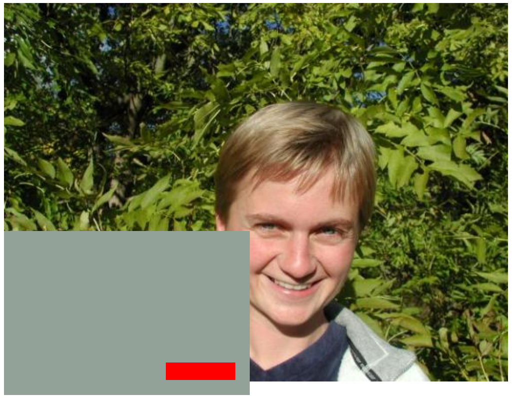
- By applying the push operation to an active pixel, its neighbor pixels may become active from one to another.
- There are four possibilities that active pixels go out of the cached area.
  - In case of (a), those pixels are pushed in a queue and processed afterward.
  - In case of (b) or (c), a control flag is set, and vertical scan is rewound.
  - In case of (d), those processed in the next vertical scan.

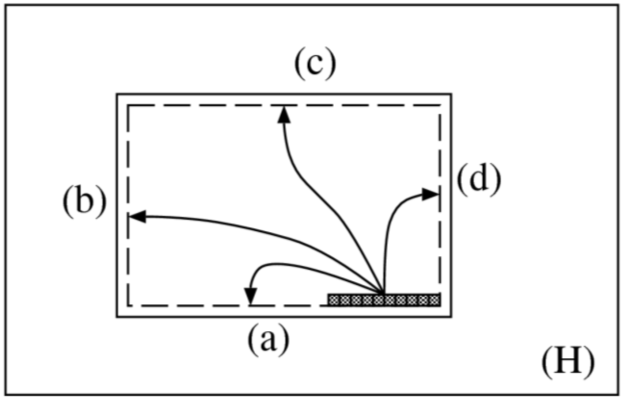




Data Caching Method

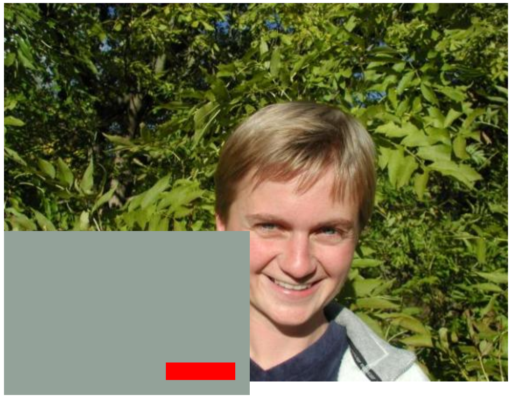
- By applying the push operation to an active pixel, its neighbor pixels may become active from one to another.
- There are four possibilities that active pixels go out of the cached area.
  - In case of (a), those pixels are pushed in a queue and processed afterward.
  - In case of (b) or (c), a control flag is set, and vertical scan is rewound.
  - In case of (d), those processed in the next vertical scan.

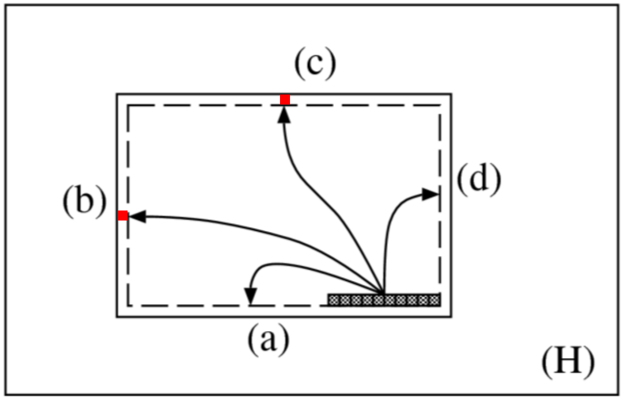




Data Caching Method

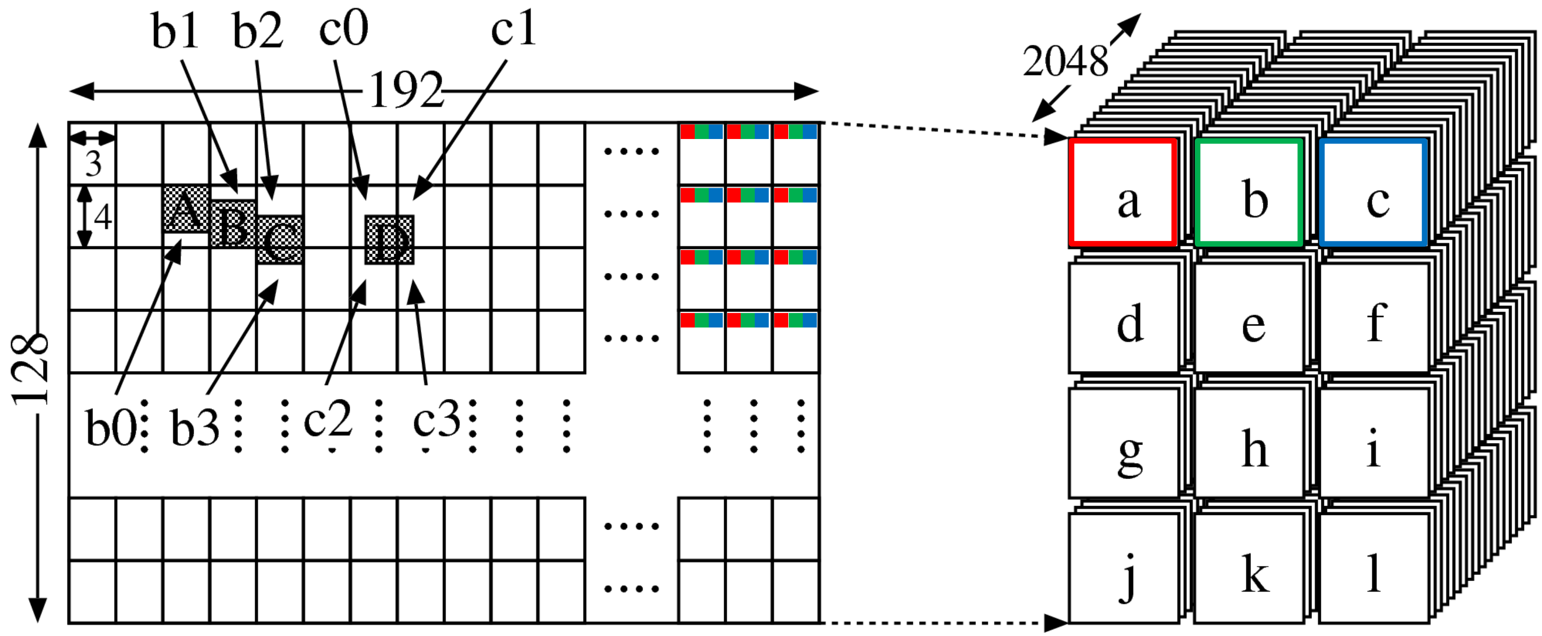
- By applying the push operation to an active pixel, its neighbor pixels may become active from one to another.
- There are four possibilities that active pixels go out of the cached area.
  - In case of (a), those pixels are pushed in a queue and processed afterward.
  - In case of (b) or (c), a control flag is set, and vertical scan is rewound.
  - In case of (d), those pixels are processed in the next vertical scan.





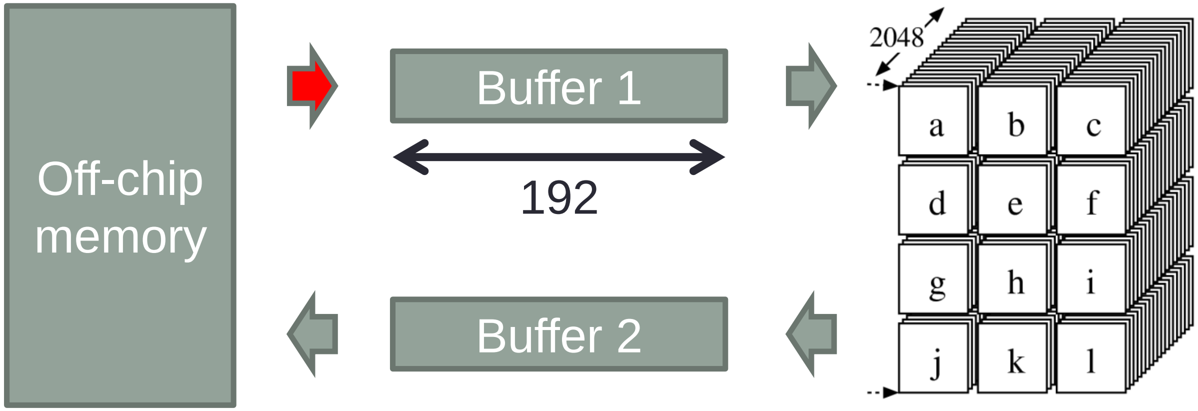
Data Mapping Method

- 192 x 128 pixels in the target area are mapped onto 12 banks (arranged 3 x 4) to allow parallel accesses to them.
- 12 pixels around any coordinate can be read out in parallel.
- 9 of the 12 pixels are selected by the selectors, and given to the push-relabel unit.



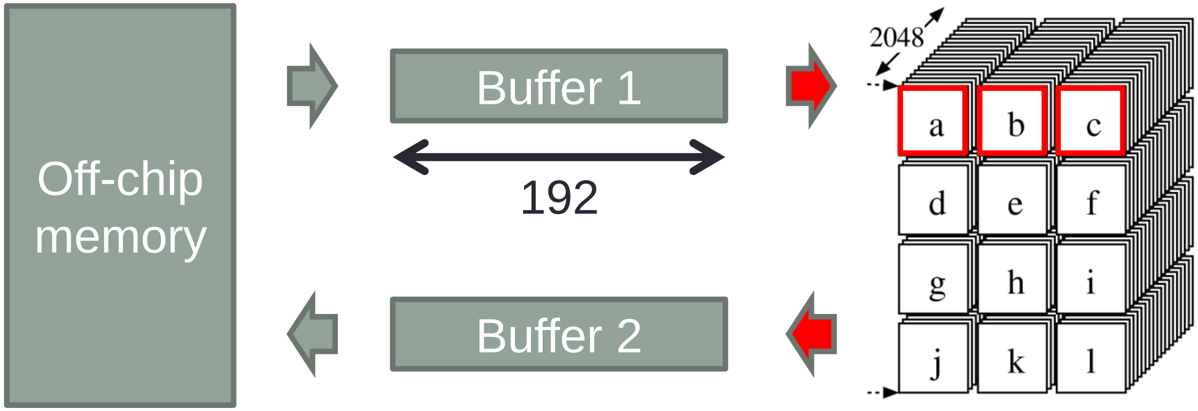
Updating the Cached Area

- The data of the next 192 pixels are read into a set of buffers which consist of distributed RAMs, while the pixels are being processing.
- When the number of active pixels becomes less than the given threshold, the push-relabel unit is stopped.
- The data of 3 of the 12 banks are updated in parallel.
- Old data are written back to the off-chip memory, while the pixels are being processing.



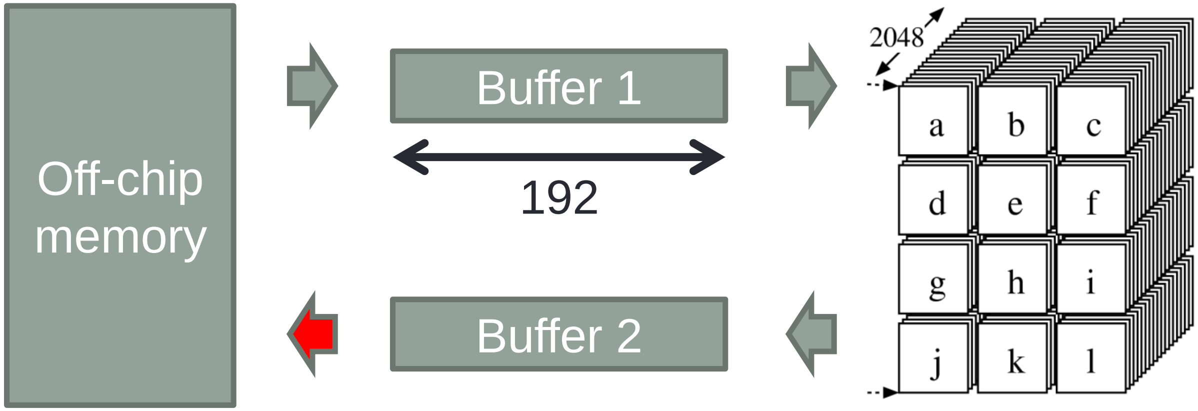
Updating the Cached Area

- The data of the next 192 pixels are read into a set of buffers which consist of distributed RAMs, while the pixels are being processing.
- When the number of active pixels becomes less than the given threshold, the push-relabel unit is stopped.
- The data of 3 of the 12 banks are updated in parallel.
- Old data are written back to the off-chip memory, while the pixels are being processing.



Updating the Cached Area

- The data of the next 192 pixels are read into a set of buffers which consist of distributed RAMs, while the pixels are being processing.
- When the number of active pixels becomes less than the given threshold, the push-relabel unit is stopped.
- The data of 3 of the 12 banks are updated in parallel.
- Old data are written back to the off-chip memory, while the pixels are being processing.

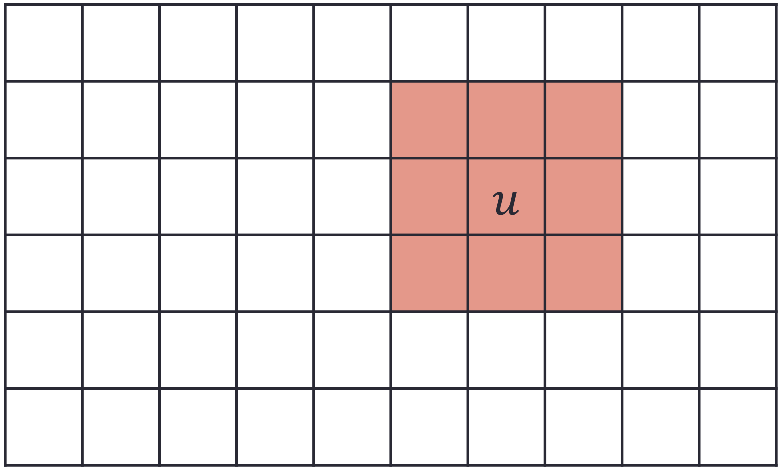


Filling the Pipeline Stages

- Push-relabel unit has 10 stages.
- In order to achieve higher performance, we need to fulfill all the pipeline stages.
- However, while a pixel u is being processed in this unit, its neighbor pixel v can not be put into the unit, because $c_f(v, u)$ may be changed by the processing of u.

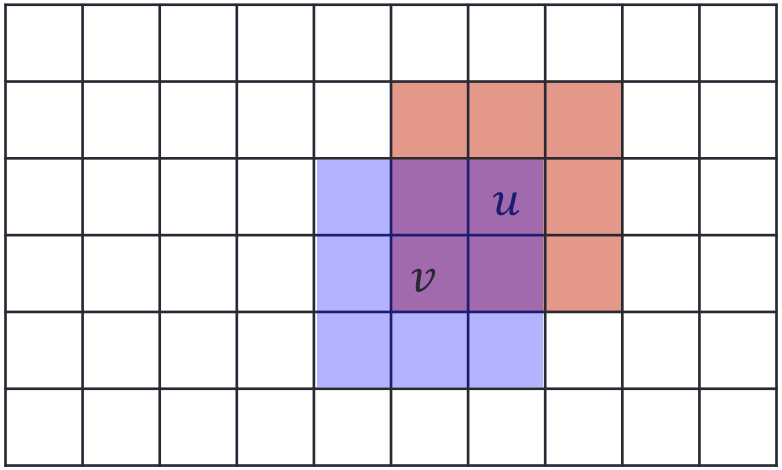
Filling the Pipeline Stages

- Push-relabel unit has 10 stages.
- In order to achieve higher performance, we need to fulfill all the pipeline stages.
- However, while a pixel u is being processed in this unit, its neighbor pixel v can not be put into the unit, because $c_f(v, u)$ may be changed by the processing of u.



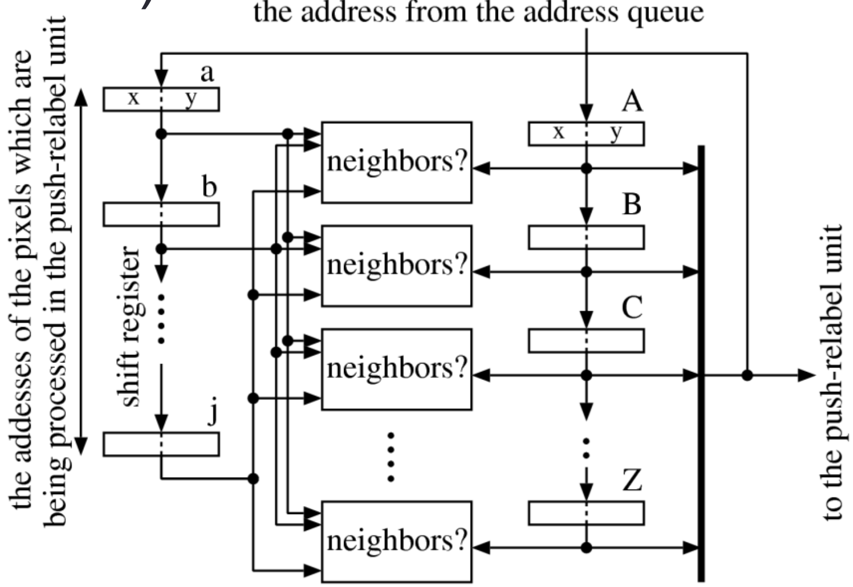
Filling the Pipeline Stages

- Push-relabel unit has 10 stages.
- In order to achieve higher performance, we need to fulfill all the pipeline stages.
- However, while a pixel u is being processed in this unit, its neighbor pixel v can not be put into the unit, because $c_f(v,u)$ may be changed by the processing of u.



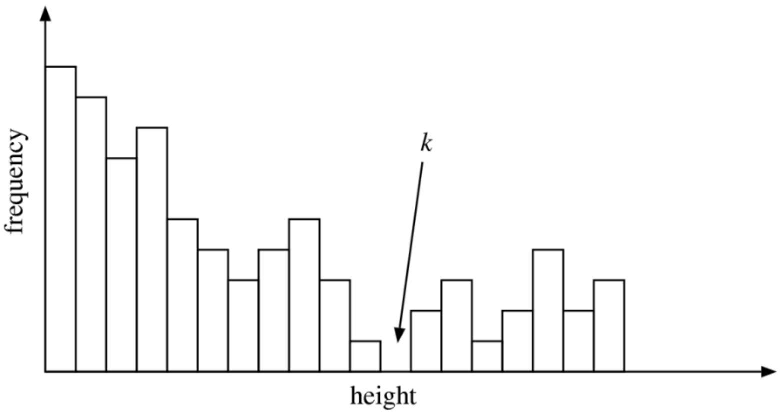
Management of the Pixels

- Suppose that pixels “a” to “j” are being processed.
- First, new pixels are put into the shift register “A” to “Z”.
- If one of “a” to “j’ is a neighbor of the pixel on “C”, the data on “C” continues to stay on the shift register.
- If several pixels can be processed, the older one is chosen (piriority is “Z”>…>“B”>“A”).



Detecting Gaps Using a Histogram

- $h(u)$ is heightened to $|V| + 1$ by gap relabeling heuristics if vertex of height = k does NOT exist and $h(u) > k$.
- In our implementation, k is looked up using a histogram of the height of all pixels.
- In our experiments,
  - The maximum value of each bin is less than 10000, so the data width of counters is 15b.
  - The maximum k is less than 512.
    We used 800 instead of $|V|$.



Experimental Results

- We have implemented the circuit on Xilinx XC6VLX130T-3.
- The circuit uses 33.3 KLUTs (41%) and 97 36Kb block RAMs (36%).
- Operational frequency is 201.1 MHz.

- We have compared the performance with
  - Software program (maxflow-v3.01) on Intel Core 2 Duo E8500 @3.16 GHz.
  - GPU program on GeForce GTX280.
- The graph is generated on the host computer.

Performance Comparison 1

Seeds (background)

Seeds (foreground)

Flower Stone2 Person2

Segmentation results

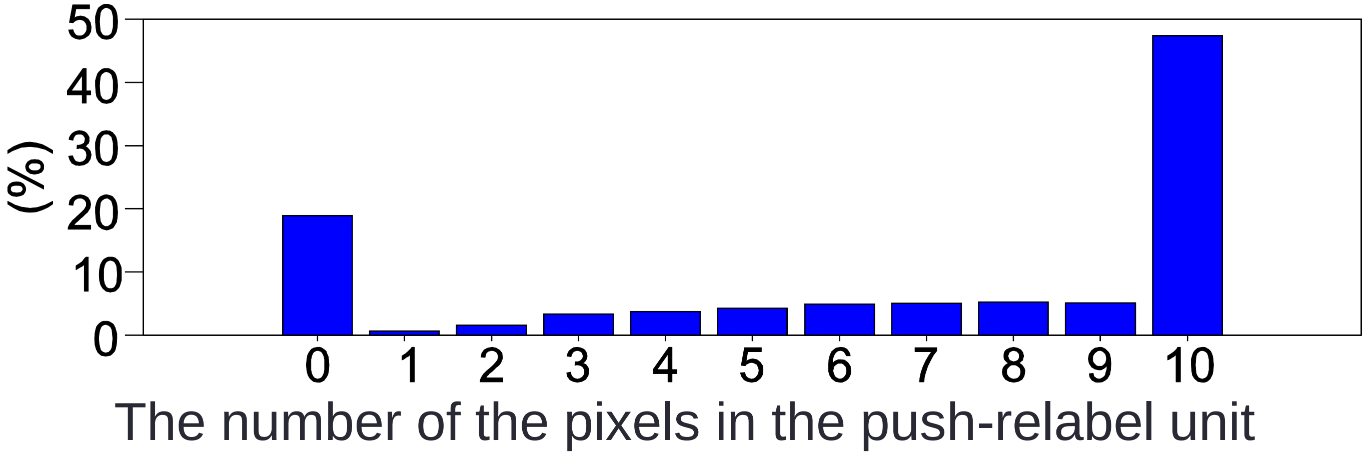
Performance Comparison 1

- The performance of the FPGA is almost comparable with GPU (20 -- 30 fps).
- Proposed system is about 3 to 5 times faster than CPU.

Benchmark		Exec. Time (msec)			Speedup
Image	Size	CPU	GPU	FPGA	
Flower	600 x 450	161.1	37	30.7	5.2
Stone2	640 x 480	117.2	44	45.8	2.6
Person2	600 x 450	118.5	61	36.7	3.2

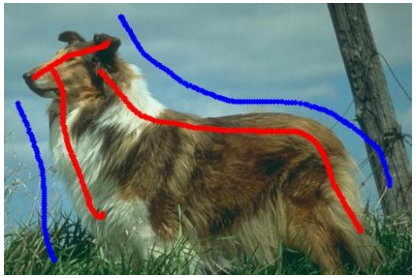
Performance Comparison 1

- This figure shows the number of the pixels in the push-relabel unit when processing Person2.
- All stages are fully filled during about 50% of the execution time.
- But the idle time occupies about 20%.

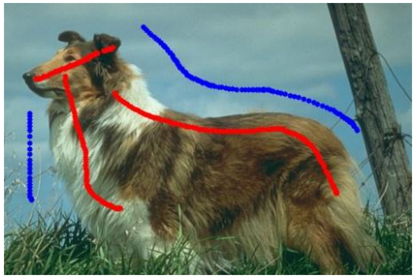


Performance Comparison 2

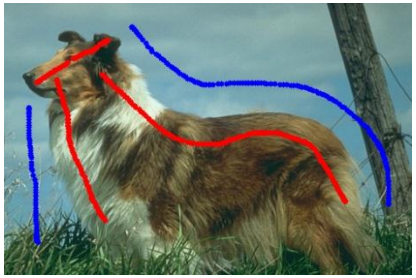
- Four different seeds are given to “dog”.
- The speedup depends on the seeds, but fast enough for real-time processing.



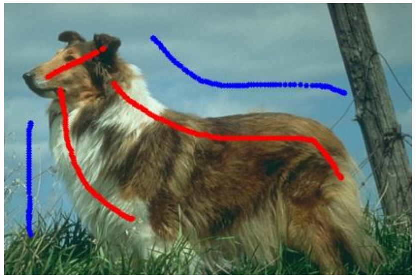
Dog / seed1



Dog / seed2



Dog / seed3



Dog / seed4

Benchmark		Exec. Time (msec)		Speedup
Image	Size	CPU	FPGA	
Dog / seed1	482 x 321	182.1	31.7	5.7
Dog / seed2	482 x 321	300.0	30.3	9.9
Dog / seed3	482 x 321	185.7	26.4	7.0
Dog / seed4	482 x 321	363.4	33.1	11.0

Performance Comparison 3

- Images with small object are segmented.
- Worse speedup because of higher idle ratio of the pipeline stages (processing of the background pixels finishes faster than the pixels on the foreground).
- However, it is fast enough for real-time processing.

Wolf

Sheep

Benchmark		Exec. Time (msec)		Speedup
Image	Size	CPU	FPGA	
Wolf	482 x 321	17.8	7.6	2.3
Sheep	450 x 600	33.6	16.9	2.0

Conclusions and Future Work

- We have proposed an acceleration method of the max-flow problem with FPGA.
- The performance gain compared with a software library on CPU is about 3 to 5.

- For more speedup,
  - We need to fill the pipeline stage of the push-relabel unit more.
  - Several push-relabel units can be implemented
    (the size of the unit is small enough).

Thank you for your kind attention