Difference Operator

Augusto José de Oliveira Martins - 01656520

June 15, 2018

Contents

1. Problem Definition

The difference operator is an edge detector algorithm. It produces an image in which each pixel is the highest absolute difference between its neighbors and itself. As an example, Figure 2 shows the edges of Figure 1.

Figure 1 - Input to the difference operator

Figure 2 - Output image

The algorithm is quite simple. To every pixel, the absolute difference between its value and its nearest neighbors are calculated. The new value of the pixel is the biggest difference. For example, consider the

matrix with the values from 1 to 9. The central pixel is 5, therefore, the maximum difference is between left top corner |1 - 5| or the right bottom corner |1 - 9|:

Equation 1 – Difference operation example 1

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

$$\max \text{ of } \{ \; |5-1| \;\; |5-2| \;\; |5-3| \;\; |5-4| \;\; |5-6| \;\; |5-7| \;\; |5-8| \;\; |5-9| \; \} = 4$$

Equation 2 – Difference operation example 2

$$\begin{matrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 1 \end{matrix}$$

$$\max \text{ of } \{ \; |10-10| \;\; |10-10| \;\; |10-10| \;\; |10-10| \;\; |10-10| \;\; |10-10| \;\; |10-10| \;\; |10-1| \; \} = 9$$

2. Kernel Implementation

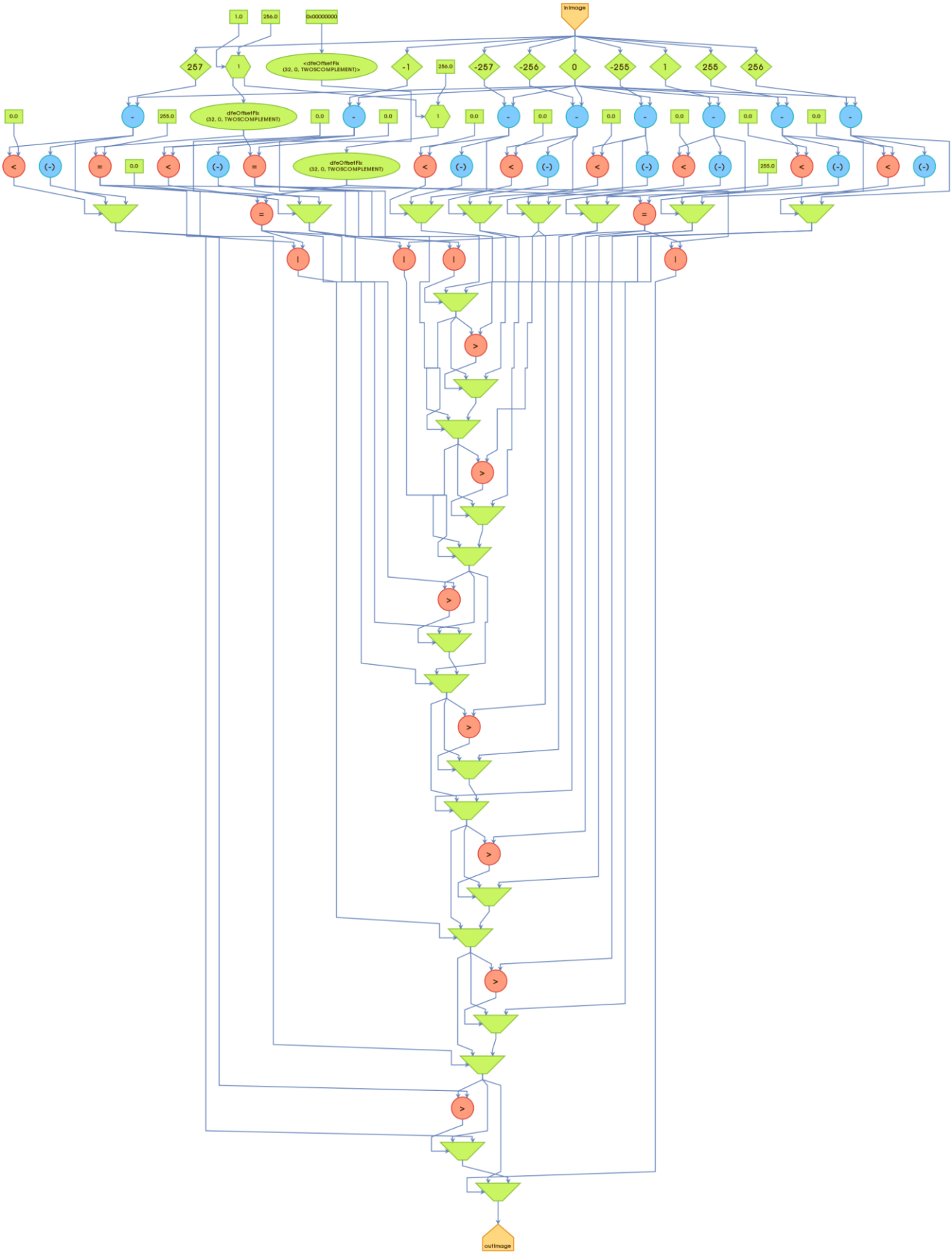
To each pixel is necessarily to find the maximum difference between its value and its neighbors. The algorithm does not have data dependency between its outputs. Therefore, each pixel can be independently calculated. Since the edges of the image don't have all the pixels in the 3x3 patch, it is necessary to keep track of the line and the column to adjust the calculations. A chain of counters takes care of this problem. The image is received as a stream from top to bottom and left to right. To read each pixel, the input stream is offset by the width of the image.

The kernel has two variables that define the width and height of the image. Therefore, the it isn't flexible to handle different sizes without recompilations. This isn't ideal. However, it wasn’t possible to simulate using scalar parameters. The resulting graph was very complicated and the simulation never finished. For the sake of simplicity, the size was restricted to 256x256 pixels.

Bellow, I list the code (Source 1) of the kernel presented in the DifferenceOperatorKernel.maxj. Graph 1 is the representation of this code.

```
int width = 256;
int height = 256;

DFEType TYPE = dfeInt(32);

DFEVar inImage = io.input("inImage", TYPE);

CounterChain chain = control.count.makeCounterChain();
DFEVar c = chain.addCounter(width, 1).cast(dfeInt(32));
DFEVar l = chain.addCounter(height, 1).cast(dfeInt(32));

DFEVar upLeft  = stream.offset(inImage, -(width + 1));
DFEVar up      = stream.offset(inImage, - width);
DFEVar upRight = stream.offset(inImage, -(width - 1));

DFEVar left  = stream.offset(inImage, -1);
DFEVar point = stream.offset(inImage,  0);
DFEVar right = stream.offset(inImage,  1);

DFEVar downLeft  = stream.offset(inImage, width - 1);
DFEVar down      = stream.offset(inImage, width);
DFEVar downRight = stream.offset(inImage, width + 1);

DFEVar firstLine = l === 0;
DFEVar lastLine  = l === (height - 1);
DFEVar firstCol  = c === 0;
DFEVar lastCol   = c === (width - 1);

DFEVar max = constant.zero(TYPE);

max = (firstLine | firstCol) ? max : KernelMath.abs(upLeft - point);
max =  firstLine             ? max : KernelMath.max(max, KernelMath.abs(up - point));
max = (firstLine | lastCol)  ? max : KernelMath.max(max, KernelMath.abs(upRight - point));

max = firstCol ? max : KernelMath.max(max, KernelMath.abs(left - point));
max = lastCol  ? max : KernelMath.max(max, KernelMath.abs(right - point));

max = (lastLine | firstCol) ? max : KernelMath.max(max, KernelMath.abs(downLeft - point));
max =  lastLine             ? max : KernelMath.max(max, KernelMath.abs(down - point));
max = (lastLine | lastCol)  ? max : KernelMath.max(max, KernelMath.abs(downRight - point));

io.output("outImage", max, TYPE);
```

Souce 1 - Difference Operator Kernel



Graph 1 – Graphical representation of the Kernel

3. Validation

I implemented a C function of the difference operator as a reference (Souce 2). This code is in the file DifferenceOperatorCpuCode.c.

I called the function with the Image 1 and Image 3 and I ran the kernel on them as well. Image 2 and 3 are the kernel's outputs. They are similar to reference function results.

```
void apply_difference_operator(int width, int height, int32_t *inImage, int32_t *outImage) {

  int idx = 0;
  int idx_up = idx - width;
  int idx_down = idx + width;

  for (int l = 0; l < height; ++l) {
    for (int c = 0; c < width; ++c) {
      int max = 0;

      int first_col = c == 0;
      int last_col = c == (width - 1);

      int first_lin = l == 0;
      int last_lin = l == (height - 1);

      if (!first_lin) {
        if (!first_col) max = abs_max(idx, idx_up - 1, max, inImage);
        max = abs_max(idx, idx_up, max, inImage);
        if (!last_col) max = abs_max(idx, idx_up + 1, max, inImage);
      }

      if (!first_col) max = abs_max(idx, idx - 1, max, inImage);
      if (!last_col) max = abs_max(idx, idx + 1, max, inImage);

      if (!last_lin) {
        if (!first_col) max = abs_max(idx, idx_down - 1, max, inImage);
        max = abs_max(idx, idx_down, max, inImage);
        if (!last_col) max = abs_max(idx, idx_down + 1, max, inImage);
      }

      outImage[idx] = max;

      idx++;
      idx_up++;
      idx_down++;
    }
  }
}
```

Souce 2 - Reference implementation in C

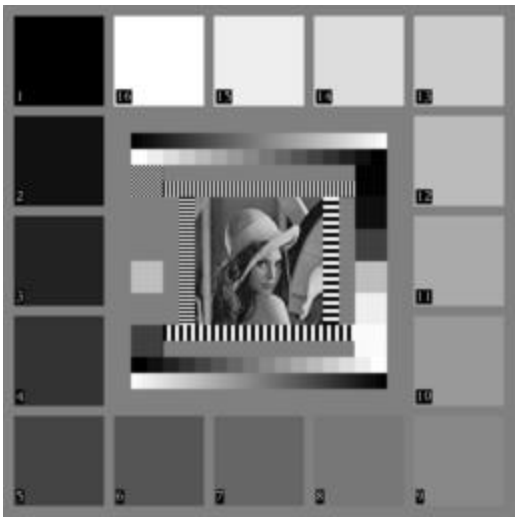


Figure 3 - Input example 2

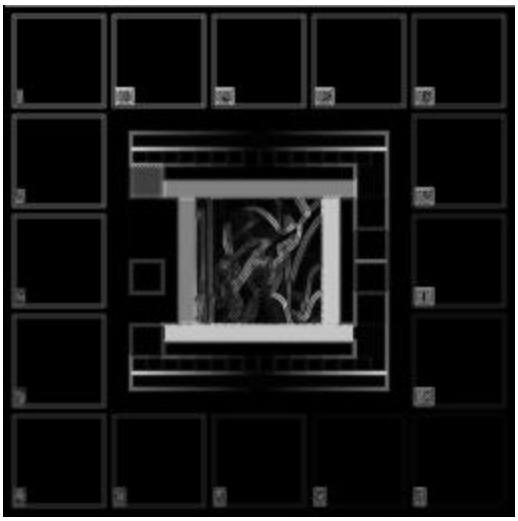


Figure 4 - Output example 2

4. Host C Interface

```
void DifferenceOperator(
        uint64_t ticks,
        const void *instream_inImage,
        size_t instream_size_inImage,
        void *outstream_outImage,
        size_t outstream_size_outImage);
```

Souce 4 – Kernel C interface

This is the declaration of the C function that wrapper the communication between host and accelerator. The kernel has one input stream and one output stream. The size of the stream must be informed as well. The host calls this function in the main (DifferenceOperatorCpuCode.c). The ppmIO.c provides routines to read and write images in the PPM format. The main function reads an image to an integer buffer. Then it calls the accelerator and providing another buffer to the result. The last operation is writing the output buffer to a PPM file.

```
int main(void) {
  printf("Loading image.\n");
  int32_t *inImage;
  int width = 0, height = 0;
  loadImage("lena.ppm", &inImage, &width, &height, 1);


  uint64_t n = width * height;
  size_t size = n * sizeof(int32_t);
  int32_t *outImage = malloc(size);

//  apply_difference_operator(width, height, inImage, outImage);
```

```
  printf("Running Kernel.\n");
  DifferenceOperator(n, inImage, size, outImage, size);

  printf("Saving image.\n");
  writeImage("lena_difference.ppm", outImage, width, height, 1);

  printf("Exiting\n");
  return 0;
}
```

Souce 5 – Host main function

5. References

Phillips, D. (1995). *Image processing in C*. BPB Publications.

Edge detection. (2018, June 10). Retrieved from https://en.wikipedia.org/wiki/Edge_detection