Lab 1 Report

Professor Gunnar Schirner
EECE 7368
Aly Sultan 001057787
Northeastern University

Table of content

Abstract

This lab focuses on analysing canny.c's source code in an attempt to get an overview of the algorithms characteristics. Characteristics considered were primary functions and their dependencies, HW/SW partitioning, and memory behavior. Each section in this report will be dedicated to answering the questions presented in the lab pdf. The tutorial in [2] was used as a reference when writing the Makefile for this Lab. Several visualization tools were used, these tools [1][4] were not installed on the remote server gateway.coe.neu, rather they were run locally for analysis purposes. All makefile targets ran successfully on the remote server except "make graph" which generated the call graph included below and relied on [1] which could not be installed on the coe servers due to insufficient priveladges. Basic makefile requirements requested were satisfied in the included makefile in the root directory of the submitted project.

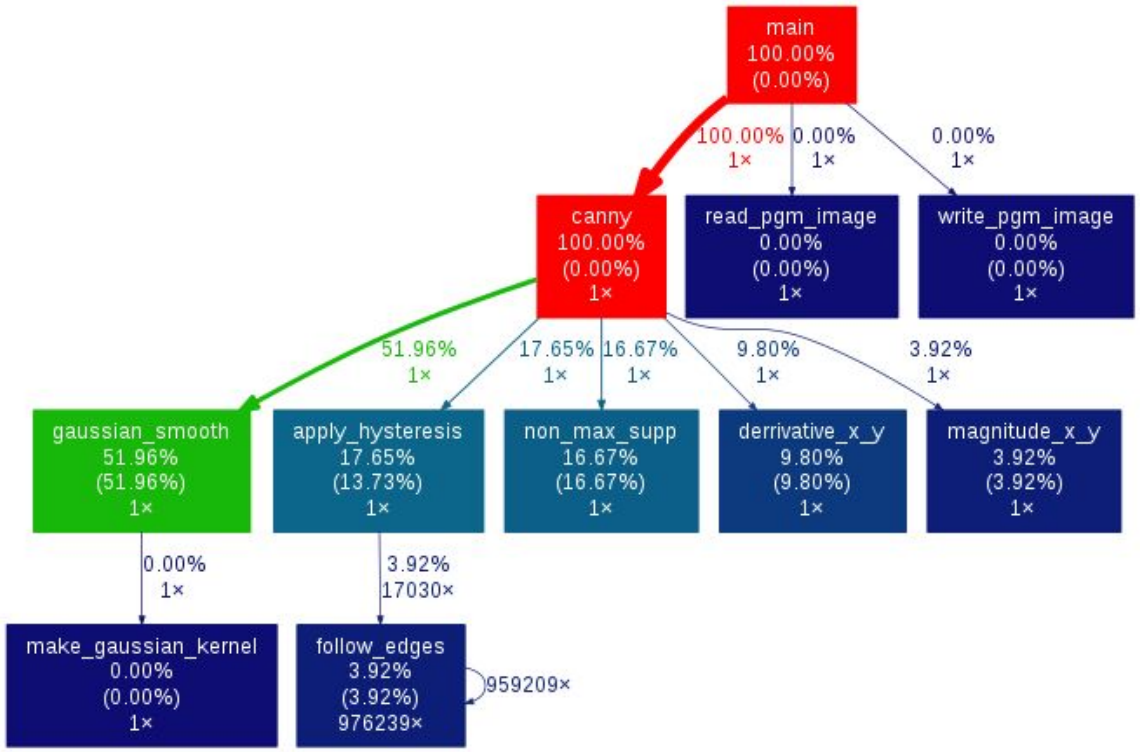
What are the main functions of the algorithm and where does the actual computation occur?

Gprof output

Flat profile:

% time	cumulative seconds		self seconds	self	total calls	name
52.51	0.53	0.53	1	0.53	0.53	gaussian_smooth
16.84	0.70	0.17	1	0.17	0.17	non_max_supp
13.87	0.84	0.14	1	0.14	0.17	apply_hysteresis
10.90	0.95	0.11	1	0.11	0.11	derrivative_x_y
2.97	0.98	0.03	17030	0.00	0.00	follow_edges
2.97	1.01	0.03	1	0.03	0.03	magnitude_x_y
0.00	1.01	0.00	6	0.00	0.00	write_pgm_image
0.00	1.01	0.00	1	0.00	1.01	canny
0.00	1.01	0.00	1	0.00	0.00	make_gaussian_kernel
0.00	1.01	0.00	1	0.00	0.00	read_pgm_image

Call graph from gprof and gprof2dot.py [1]



Analysis

After running gprof it is evident that the gaussian_smooth function takes the longest time to execute. Additionally from the Call graph above (generated from python source in [1], ran locally, not installed on server) the most significant function were computation primarily occurred was the canny function called from main. Other notable functions include the input/ output read/ write functions for image input and output, and helper functions for canny: gaussian_smooth, apply_histerisis, non_max_supp, derivative_x_y and magnitude_x_y.

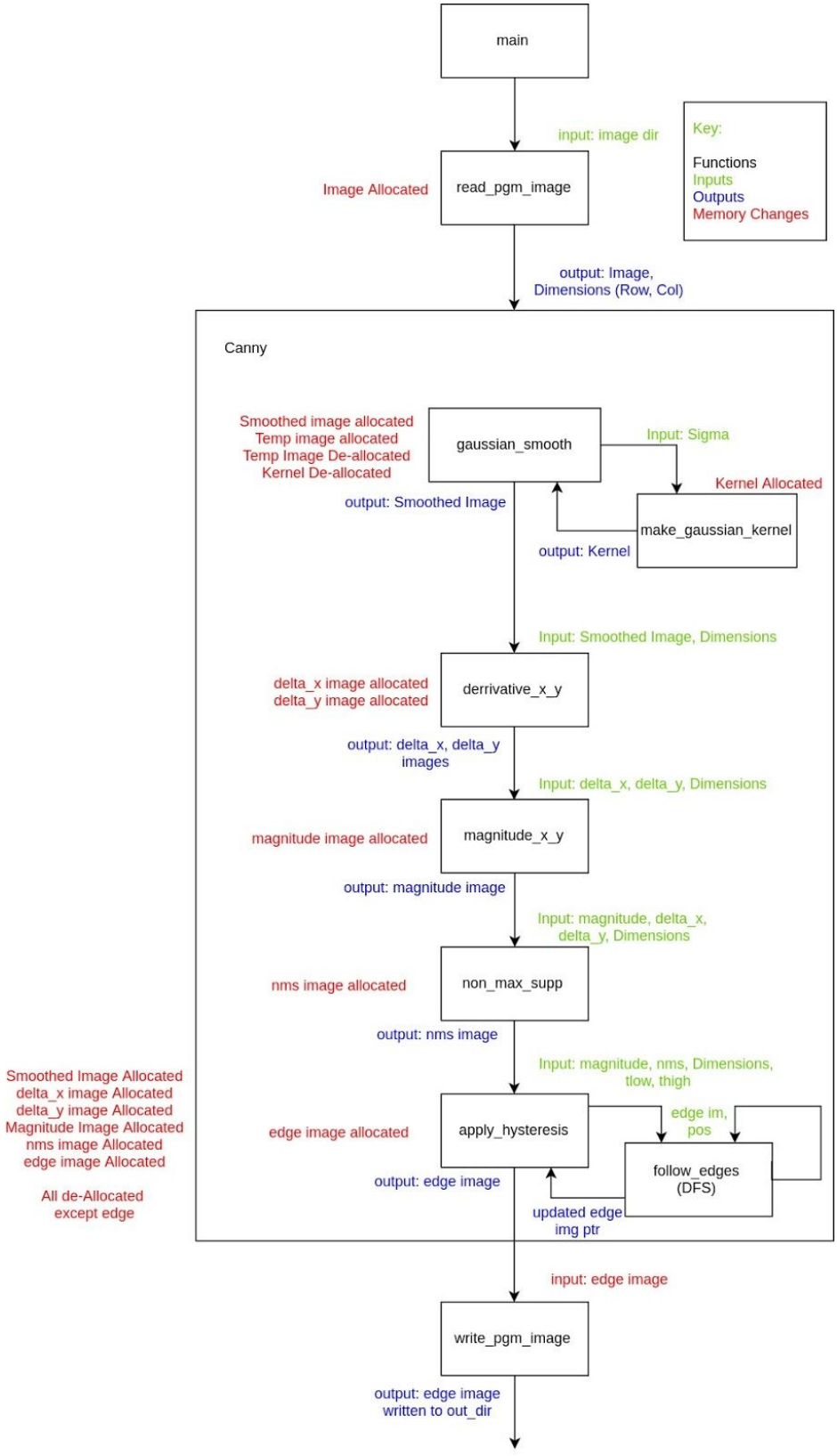
Which functions are used for data input and data output?

The functions read_pgm_image and write_pgm_image were used for input and output of image data to and from the executable.

Function hierarchy and communication dependencies

Analysis

Presented on the right is a detailed function hierarchy and is used to highlight function dependencies and memory changes that occur throughout the life cycle of the program. Since the canny function is were most of the primary computation resides, a description of each function called in canny is given below.



Functions

Gaussian_smooth
Applies a blur effect to the image to increase aliasing. Effectively a convolution with a 1D kernel, a stride of 1, and padding to guarantee that input image size is retained in the output

Derivative_x_y
Calculated derivative at each pixel relative to surrounding pixels.

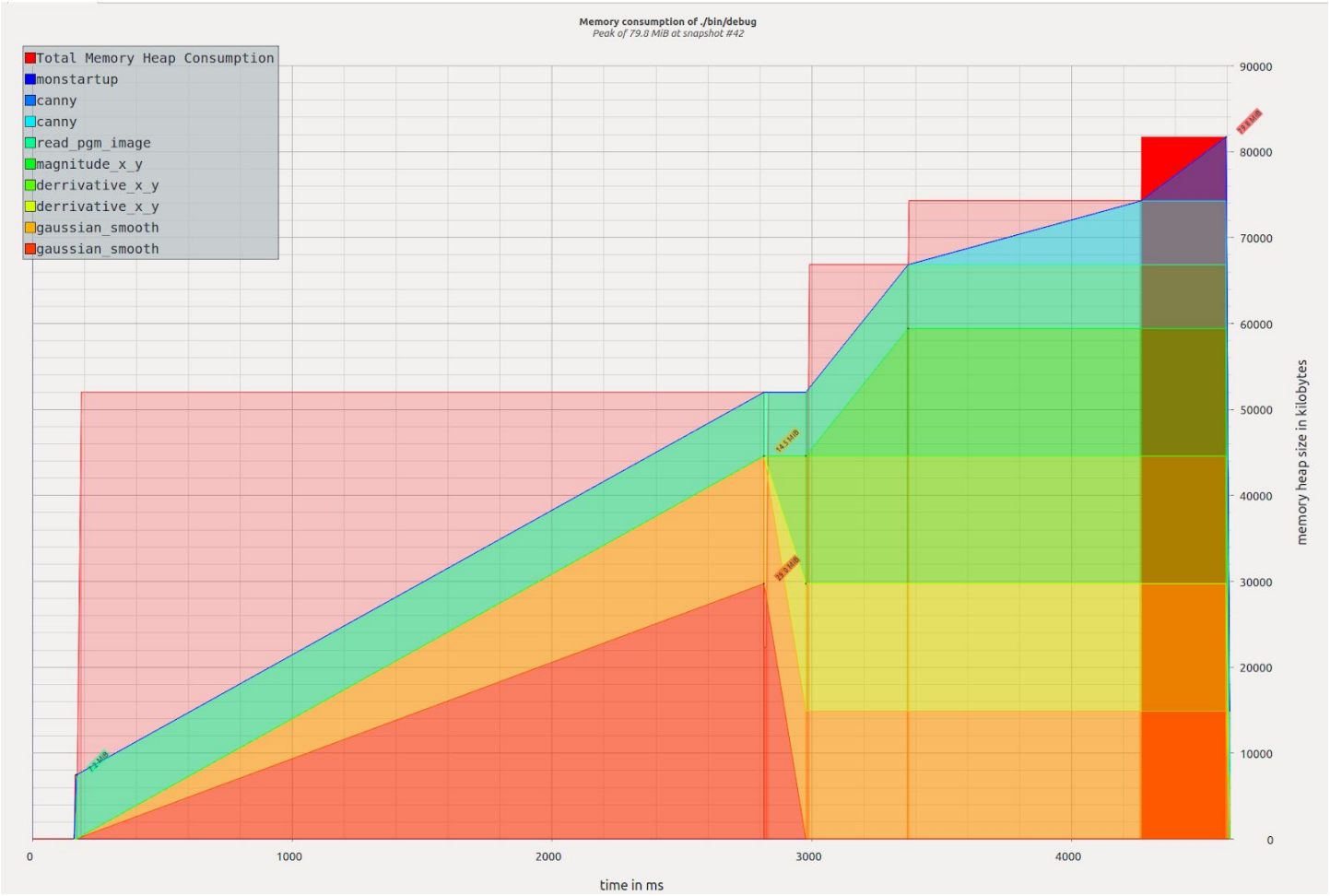
Magnitude
Calculates the magnitude of the gradient

Non_max_supp
Determines pixels where likely edges exist. Disregards border pixels to saves computation during apply hysteresis.

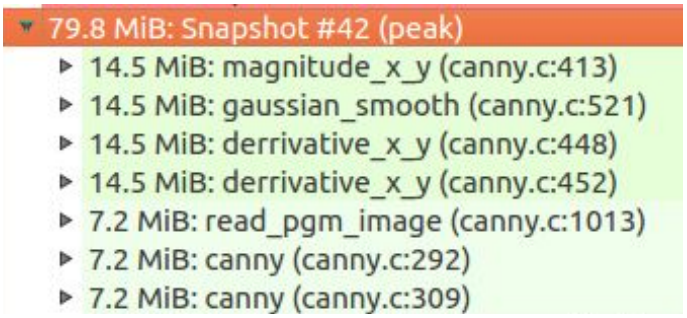
Apply hysteresis
Finalizes edge location via follow_edges function.

Follow Edges
Similar to Depths first Search for adjacent edges

How much memory is needed when the algorithm runs?



Analysis



The above image (generated from Valgrind massif and massif-visualizer [4]) shows heap consumption over the lifetime of the program. Heap consumption grows linearly as few deAllocations occur during the program's lifecycle. Many allocations occur within canny and it's called functions. Most are not de-allocated until the end of canny. Memory usage scales with input size. The above results were generated with an input image of size 4461x1704 which consumes 7.2Mib when loaded into memory as an unsigned character array. Memory usage peaks at 79.8Mib as a result of several intermediate image allocations coexisting between function calls. The intermediates are as follows: delta_x delta_y magnitude smoothedim, and nms. Input is the image array and output is the edge array. It must be noted that the representation between the intermediates is not consistent. Some intermediates like delta_x, delta_y, magnitude, and smoothedim, represent each pixel as a short int (2 bytes). This results in some intermediate representations being larger than others as in the snapshot presented of peak memory usage. For hardware acceleration purposes representation needs to be taken into consideration due to memory

constraints. Often lower bit representations for pixels yield negligible loss in accuracy combined with greater speedup and lower memory movement. Memory accesses are generally non sequential for most functions. Given the nature of the operations being applied on the image, many operations operate on the image vertically which results in non-sequential memory access due to how the image is stored. Memory accesses greatly affect the choice of functions for HW partitioning which will be discussed in the following section.

Are there any obvious candidates for hardware acceleration and What should be performed in software?

Analysis

Considering the size of the input image, the choice of functions that can be accelerated is limited to functions were operations on the input image can be streamed and parallized to an accelerator core. That relies heavily on the nature of memory accesses within the function and the operations being done.

Functions to accelerate in order of importance

- **Gaussian_smooth**

  Gaussian smooth is a simple convolution that convolves the input image with a 1D kernel. The function breaks down convolution into the x and y direction and accumulates the sum of the dot product for both directions to produce one output image. Provided that the transpose of the image is available, the image can be streamed to an accelerator core which can convolve across rows and across x and y directions in parallel, thus significantly shortening the time taken to execute the function. Transposition of the image allows memory accesses to remain sequential and allows contiguous memory allocation to occur without having to reset DMA controllers to new addresses and thus preserving high throughput. Considering that this function took half the execution time of the program, any speedup achieved here will significantly affect the overall program execution time.

- **Non_max_suppression**

  Non max suppression attempts to estimate the probable presence of an edge in a pixel based on a calculation that depends on the magnitudes and gradients of the 8 pixels boarding the pixel in question. While memory accesses are generally non sequential this function, it can be accelerated if array partitioning is applied before segments are sent out to the core.

- **Derivative_x_y**

  Similar arguments for accelerating **Gaussian_smooth** apply here. Derivative calculation can be both parallelized across rows and across directions provided that the transpose of the image is present.

- **Magnitude**

  Magnitude calculation can also be parallelized across derivative directions and streamed provided the transpose of the image is present. However, the percentage of execution time dedicated to the Magnitude function isn't significant. So in terms of developer effort, it might not be necessary to accelerate this function despite it being possible in the absolute.

It must be noted that the above statements do not factor in single threaded performance of a general purpose core the exists on the same die as the accelerator. The degree of parallelism allowed by the cores may not be sufficient to beat a general purpose processor in terms of speed especially for the functions like Magnitude and Derivative_x_y.

Functions to avoid attempting to accelerate

- **Apply_histerisis and Follow Edges**

  The above functions access the image array non-sequentially. Follow Edges is similar to the DFS algorithm which complicates hardware implementations for it significantly because there is no predictable pattern to memory accesses so no core can operate on multiple pixels simultaneously. These two functions should be left on the processor implemented in software.

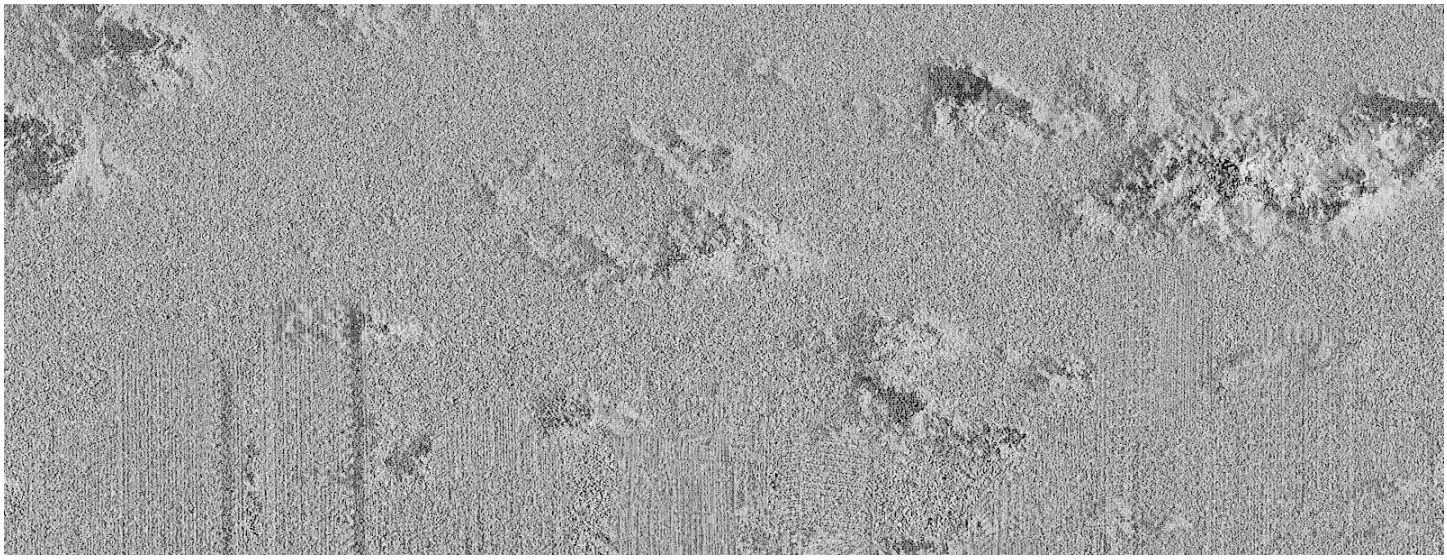
Image Data

Below is a list of the images that existed in memory of the program at different stages of its lifecycle. The final output is presented below the input as required by the Lab.pdf

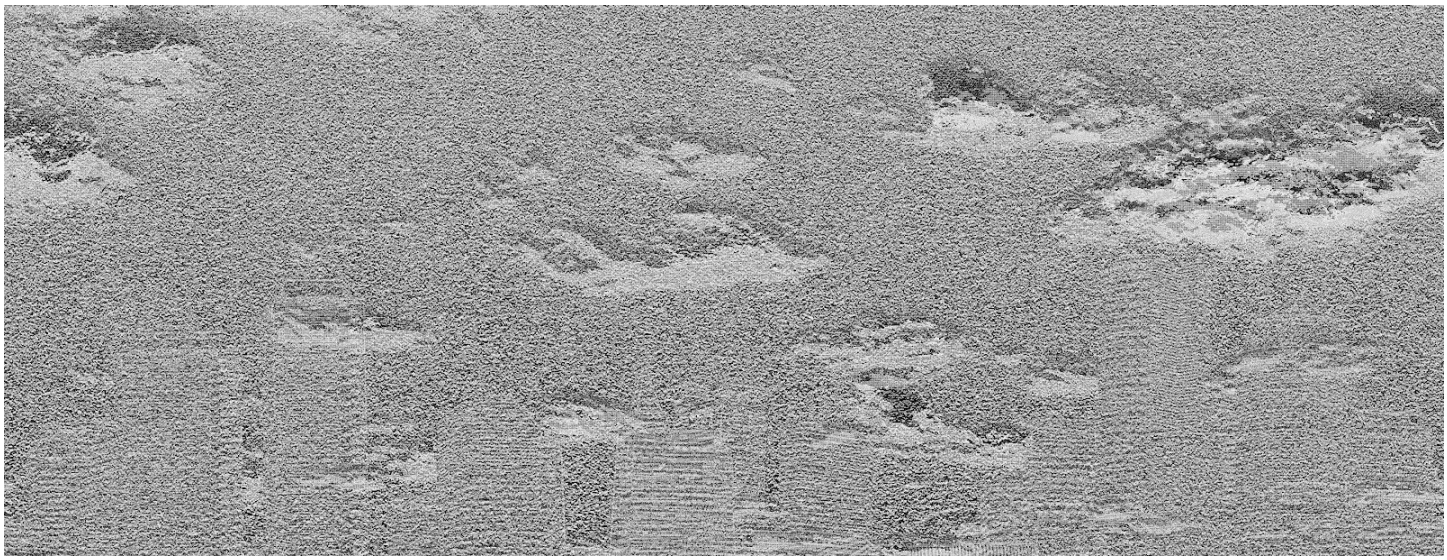
Canny input [3]

Canny Final Output

Delta_x



Delta_y



Magnitude

nms

References

[1] gprof output call graph https://github.com/jrfonseca/gprof2dot/blob/master/gprof2dot.py
[2] Makefile tutorial http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
[3] Boston skyline https://rew-online.com/wp-content/uploads/2019/03/BOSTON.jpeg
[4] massif-vsualizer https://github.com/KDE/massif-visualizer