CSEE W4840 Embedded System
Design Report

The Evolution of a Smile
A Genetic Algorithm with FPGA Implementation

Jihua Li - jl4345
Wenbei Yu - wy2228
Yini Zhou - yz2719
Jian Jiao - jj2756

March 26, 2015

1. Overview

In this project, we will design an accelerator for Genetic Algorithm to generate Mona Lisa or any other images from circles or polygons which are generated by DNA sequence. Genetic algorithms, known as one of robust heuristic algorithms for complex optimization problems in various fields of engineering, find application in bioinformatics, computational science, economics and many other fields.

The goal of this project is to demonstrate Genetic Algorithm and hopefully to speed up the algorithm with the FPGA implementation when compared to running on a regular CPU.

2. Motivation

In recent years there has been a great interest in accelerating time consuming algorithms that solve large combinatorial optimization problems [1].

The Genetic Algorithm (GA) is a powerful technique that implements the principles nature uses in biological evolution to optimize a multidimensional nonlinear problem. GA provides robust capability of exploring in the solution space of a given problem. However, there are some notorious problems exists for GA. [2] One big problem is the computation time. Although, a GA can provide very good solutions for such problems, the amount of computations and iterations required for this method is enormous. As a result, software implementations of GA can become extremely slow for large circuit partitioning problems. To reduce the execution time of GA, hardware implementation of GA has been proposed[3].

So in this work, our target is to design an implementation on an FPGA for GA, trying to extract parallelism from the algorithm, to reduce its execution time.

3. Hill Climbing Algorithm

The Stochastic Hill Climbing algorithm is a Stochastic Optimization algorithm and is a Local Optimization algorithm (contrasted to Global Optimization). It is a direct search technique, as it does not require derivatives of the search space. Stochastic Hill Climbing is an extension of deterministic hill climbing algorithms such as Simple Hill Climbing (first-best neighbor), Steepest-Ascent Hill Climbing (best neighbor), and a parent of approaches such as Parallel Hill Climbing and Random-Restart Hill Climbing.

3.1 Strategy

The strategy of the Stochastic Hill Climbing algorithm is iterate the process of randomly selecting a neighbor for a candidate solution and only accept it if it results in an improvement. The strategy was proposed to address the limitations of deterministic hill climbing techniques that were likely to get stuck in local optima due to their greedy acceptance of neighboring moves.

3.2 Procedure

Algorithm (below) provides a pseudocode listing of the Stochastic Hill Climbing algorithm for minimizing a cost function, specifically the Random Mutation Hill Climbing algorithm described by Forrest and Mitchell applied to a maximization optimization problem Forrest1993[4].

Input: $Iter_{max}$, ProblemSize
Output: Current
Current <- RandomSolution(ProblemSize)
For ($iter_i$ from $Iter_{max}$)
  Candidate <- RandomNeighbor(Current)
  If (Cost(Candidate) >= Cost(Current))
    Current<-Candidate
  End
End
Return (Current)

Pseudocode for Stochastic Hill Climbing.

3.3 How our code works

(1) Set up a random DNA linked list.
(2) Copy the current DNA list and mutate it to generate a new generation.
(3) Pad every circles of the generated linked list onto an image.
(4) Calculate the difference between this image and the original image.

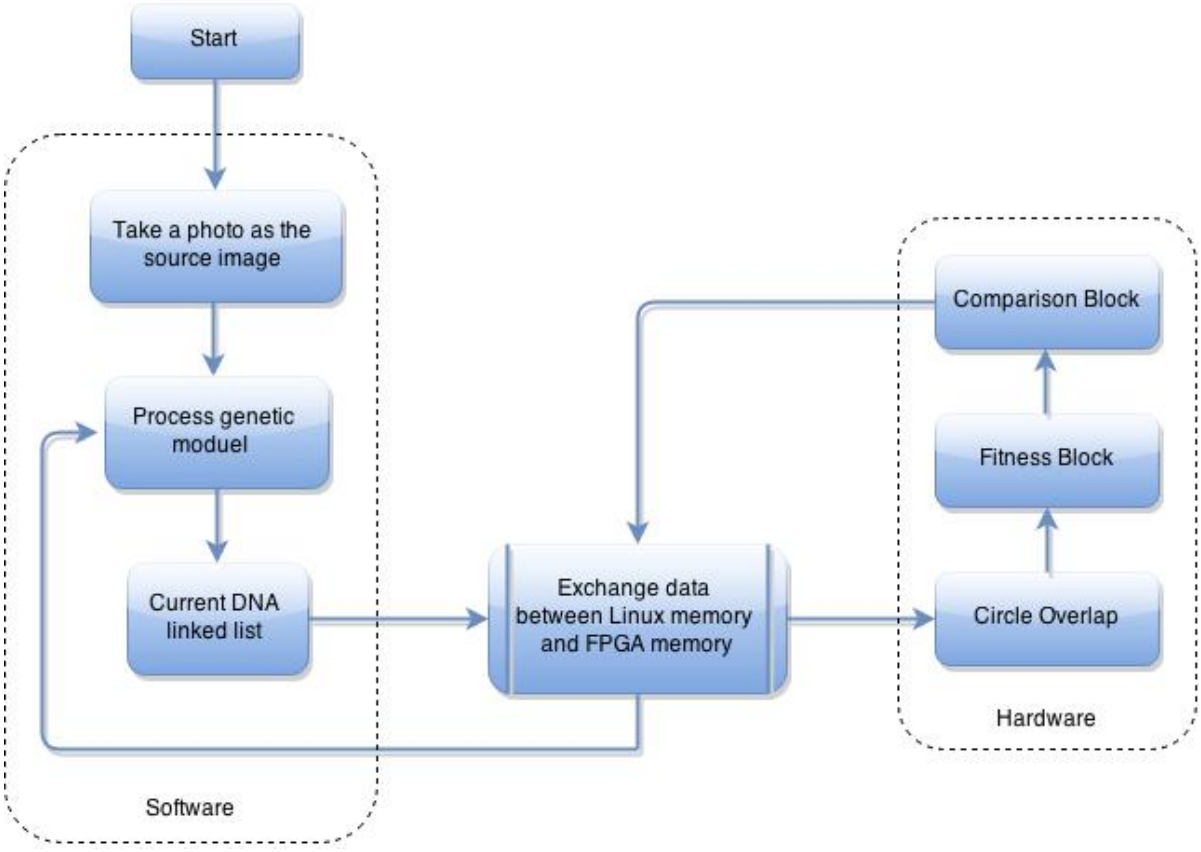
(5) If the new difference is smaller than the previous one, then overwrite the current DNA with the new DNA.
(6) Repeat from (2).

4. Software

Our software has two main sections. The first section is to generate linked lists and pass them to register for comparison. And clone the best generation's information returned by comparator to all other generations.

The second section is a camera driver. The functions are as follow. Take a photo using a camera and store it as the original image. Store the scene when process stops in order to use this image for the further processes. Implement control signals, like start and stop, to take control of the process and design a basic user interface.

5. Architecture



6. Hardware/Software Interface

6.1 Driver for camera and mouse

Linux kernel should load driver automatically when camera and mouse automatically when they are connected from USB.

6.2 Flags

There would be a flag set up when hardware completed fitting and comparator process and it tells software that current child have finished fitting and give me the next one. There would also be a flag set up when hardware is ready to read the next circle chain into its memory for further processing.

6.3 Struct vector R/W

Struct vector exist in both software and hardware, its structure contains the position, rgb color, radiance, opacity of the circle we are drawing. Software read and write such vector in hardware as memory write and read.

7. Hardware

7.1 Circle Overlap

The function for Circle Overlap module is to pad every circles of the generated linked list onto an image, calculate every pixel values of the generated image, which will be used to compare with the target image in the Fitness module.

The input for this module is all the information of the linked list, which is generated by the software and saved in a local register. For every circle, the register saves information of its radius, pixel values for R, G, B. Once the software send the linked list's data to the register, this module will read the information from the register, change the corresponding values of pixels in the result image, which is also saved in the register, until finish all the padding.

The output of this module is all the pixel information of the padded image, which will be saved in a register, for the Fitness module doing the comparison.

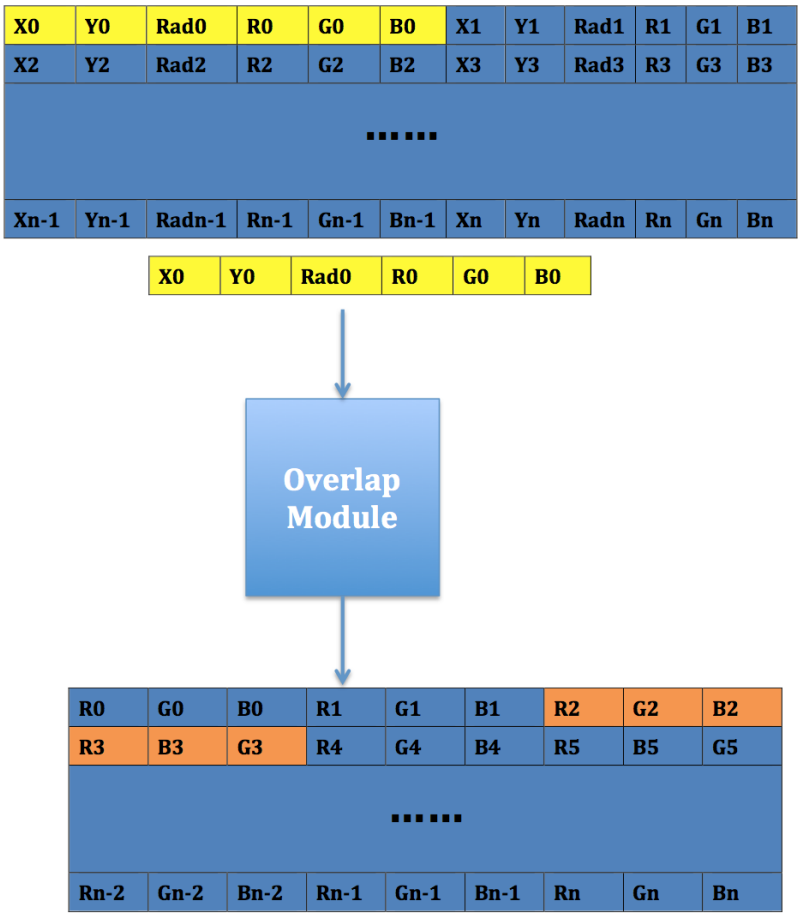


Fig. Data flow of the Overlap Module

7.2 Fitness

After the circles from the linked list overlap together and form a image in the circle-overlap module above, the fitness module will compare this image with the target image, which will calculate the absolute differences of pixel values from corresponding pixels in these two images and sum them together. This sum of absolute differences will store in the memory buffer for comparison in the next step, while the memory for the padded image and all the data of differences in the procedure will be cleared for fitness of the next padded image.

In detail, considering that the memory of the hardware may not support all these calculations including circle overlap, fitness and comparator, we may calculate the sum of absolute differences between the padded image and target image line by line. Store the result for each line and calculate the next one.

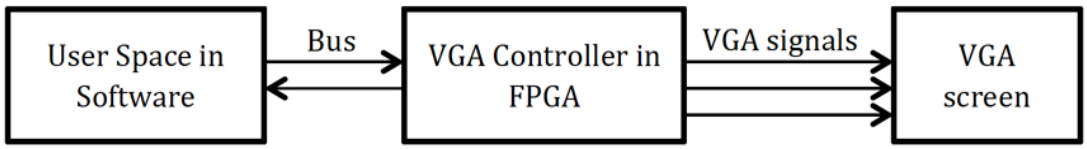
7.3 Comparator

The comparator gets its input from the fitness module, that is, the sum of absolute differences between one padded image and the target. This sum will be compared with the one generated before. If the new sum is smaller, it will remain and be used for comparison next time, and the order number will be remembered, while the old one will be deleted. If the new one is bigger, it will be removed and the old one remain.

When all mutations in one generation are generated, the user space (software) will send a signal (maybe a bit of flag) together with the last linked list of circle, to circle-overlap module. And this flag will finally send to comparator after the circle overlap and fitness module to mark the last cycle comparison in this generation. When the comparator finishes its work and gets the final value which is the smallest sum of absolute differences among all, it will send this result to the user space as well as the order number. Thus the user can find out the best one in this generation.

7.4 VGA Display

The VGA controller generates the VGA signals to control VGA display, just like what we did in lab3. In this module, we need to generate clock signals such as VGA_CLK, achieve display in any coordinates of the screen and control color values for each pixel. This module should communicate with user through Linux device driver, so the user can control the coordinates, shape and color displaying on the VGA screen.



8. Milestones

Completed Milestones

1. Block diagram design of the whole system
2. C code simulation on PC and analysis memory use and time cost
3. Determine how to balance hardware and software and make the most of hardware acceleration.

Milestone 1

1. Write C code of our own based on our system structure and run for generations to make sure it really works
2. Design software and hardware interface module
3. Design VGA part so that software can start with UI design and picture capture

Milestone 2

1. Hardware modules design and test separately with software
2. Timing diagrams for submodules

Milestone3

1. Full implementation of our system
2. Start the whole system and see results if it's faster and if it can evolve into the picture we want

Final Project Presentation

1. Close up the whole system
2. Write up final report and prepare for presentation

References

[1] D. Abramson, P. Logothetis, A. Postula, and M. Randall, "Application Specific Computers for Combinatorial Optimization," in Australian Computer Architecture Conference. Sydney, Australia: Springer-Verlag, 1997, pp. 29–44.

[2] S. Wakabayashiy, T. Koidez, N. Toshiney, M. Yamaney and H. Uenoy: "Genetic Algorithm Accelerator GAA-II," Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific, pp. 9-10 (June. 2000)

[3] S. D. Scott, A. Samal and S. Seth: "HGA: A hardware-based genetic algorithm," Proc. ACM/SIGDA 3rd International Symposium on FPGA, pp.53–59 (1995).

[4] S. Forrest and M.Mitchell, "Relative building-block fitness and the building-block hypothesis", in Foundations of Genetic algorithms 2, 1993