

# Evolving a GPU kernel to perform complex low-level image-processing tasks

David H Jones, Adam Powell, Christos Bouganis, Peter Cheung\*

## Abstract

We present a new technique for the design of GPU-based algorithms for complex, low-level, image processing tasks. Cellular automata (CA) are a form of distributed computing architecture that can perform complex tasks determined by the local interactions between its cells. We show that an evolutionary algorithm can be used to determine the local interactions of the CA from certain desired global functionality then we demonstrate this by evolving a salient point detector. Finally we show that both the CA and the evolutionary algorithm can be accelerated on a GPU, making for a powerful technique for the design of GPU kernels for complex parallel processing tasks.

## 1 Introduction

Of the applications showcased by the Nvidia cuda community, those accelerated the most fall into two classes: modelling with multi-agent systems and signal processing. Multi-agent systems analysis normally involves studying the global properties of a distributed system of which the local interactions are already known, making the coding of an appropriate kernel simple. This is not true of a signal processing task, where typically we need to determine the local behaviour of our filters such that globally they demonstrate some desired function. Designing large-scale, locally interacting parallel systems such that they perform some function or display certain complex properties is computationally difficult. Our approach to this problem builds on existing techniques for the design of cellular automata, another form of locally interacting distributed system.

Cellular automata (CA) are dynamic systems in which space and time are discrete. CA consist of a number of identical cells in an array. Each cell can be in one of a number of states. The next state of each cell is determined at discrete time intervals according to the current state of the cell, the current state of the neighbouring cells and a next-state rule that is identical for each cell. That a CA is a homogeneous array of processes makes them appropriate for GPU implementation, however because the update stage of every cell must be synchronised the potential performance is limited. Also we know certain CA are capable of performing complex calculations[4], but designing the local rules each cell obeys such that the CA performs some specific calculation, the so-called *inverse* problem, is largely beyond the capabilities of deterministic design algorithms. An alternative is to use a stochastic, evolutionary, design algorithm[4].

In this paper we will show that a large array of low-level processes can be effectively designed to perform some signal processing task using an evolutionary algorithm and a GPU. We will then demonstrate this technique by evolving a GPU to perform a signal processing task of the human vision system: determining the most salient, or conspicuous, points in a given scene.

## 2 The evolutionary design of cellular automata for image processing

The low-level vision task we choose to evolve on a CA is the determination of a saliency map for a given input scene. Saliency is the measure of object conspicuity within the scene. This is determined by the V4 region of the primary visual cortex and is used to direct the rapid movements of the eye (saccades). Most attempts to artificially determine the saliency of a scene have used different combinations of banks of Gaussian, Gabor, color and intensity filters [1]. Saliency filters have been used in image and video compression, rapid scene analysis, object recognition and tracking. As the visual cortex uses a large number of locally interconnected components to determine the saliency of a scene it is an appropriate task to imitate on a CA.

As saliency is such a subjective measure there may be one or more effective solutions, making it an interesting task for evolutionary algorithms. We need an evolutionary algorithm appropriate for exploring large search spaces with many possible solutions. Here we describe the three components of our design algorithm: a genome structure, a fitness function and some means of evolving the genome.

### 2.1 Training set and fitness function

To train the next-state rule of our CA we create a set of test input grayscale images and use their pixel saturations to set the initial states of each cell within the CA. We then repeatedly iterate the CA, each cell using the evolved next-state rule to determine its next state. After  $N$  iterations, the states  $c_{x,y,t=N}$  of each cell at position  $(x,y)$  within the CA form a map of the salient points in the image.

For each image in our training set we calculate its saliency map using a classic saliency algorithm[1]. The fitness of every evolved genome is determined as a sum of squared differences between the CA's final, normalised, state and the output of this normalised saliency map.

### 2.2 Genome structure

We need a form of genome that can describe a CA next-state rule and be implemented on a GPU. Cartesian genetic programs [3] use arrays of integers to describe the function and interconnections of a collection of functional components. We use a cartesian genetic program formed of  $3 \times 20$  integers to describe the CA next-state rule. Each set of 3 integers describes a

\*Thanks to EPSRC for funding and NVIDIA for assistance.

component of the next-state logic of each cell. The first integer ( $I_1 \in [0, 7]$ ) chooses the type of component from the following functions: Pass through, Compare, Bit shift left, Bit shift right, Subtract, Add, OR, AND. The remaining two integers ( $I_{2,3} \in [0, 25]$ ) select the inputs for this component. These refer to the output of previously executed components or the current state of the cell or its neighbours.

### 2.3 Mutation algorithm

We use a variant of the HereBoy algorithm [2] for the mutation of this CA because its version of constrained simulated annealing is particularly suited to exploring large search spaces with many possible solutions.

To start the process, a genome is initialised as a list of “pass through” components, each taking their input from the previous circuit; thus  $c_{x,y,t+1} = c_{x,y,t}$ . This genome is interpreted by each cell of the CA as a next-state rule. The next-state rule is used to synchronously update the state of each cell 20 times. 20 iterations is sufficient for each cell to determine its final state according to the inputs of 1% of a 300x300 input image. The final state of each cell in the CA forms a map of the saliency of image. The fitness,  $f$ , of this genome is then evaluated.

The HereBoy algorithm uses a population of two genomes: the best solution so far and a mutation of this solution. In most cases, if the mutated genome performs better than the current solution, the current is replaced with the mutated. Otherwise the mutated genome is discarded and another mutation of the best solution is evaluated. First the algorithm attempts to determine the general structure of the solution with high mutation rates, then tries to refine it with lower mutation rates. Thus the probability of mutating each integer  $p_m$  of the genome is function of pre-defined limits ( $p_{m,min}, p_{m,max}$ ), the fitness of the solution and the expected maximum fitness,  $f_{max}$ :

$$p_m = \frac{f_{max} - f}{f_{max}} \times (p_{m,max} - p_{m,min}) + p_{m,min} \quad (1)$$

In order to ensure the evolutionary algorithm doesn't settle on a local maxima of the search space, occasionally the algorithm will select the sub-optimal genome to mutate. The probability  $p_r$  of this occurring is determined according to the same formula for determining  $p_m$ .

## 3 Implementation of cellular automata

As well as being an appropriate medium on which to mimic biological systems, CA are particularly suited for implementation on graphics processing units (GPU). The GPU architecture allows us to execute many thousands of parallel threads, but each thread must run the same code. This is comparable to a CA, in which each of many thousands of cells run the same program synchronously. Figure 3 shows the sequence of tasks and their delegation between the CPU and GPU of our proposed design algorithm<sup>1</sup>.

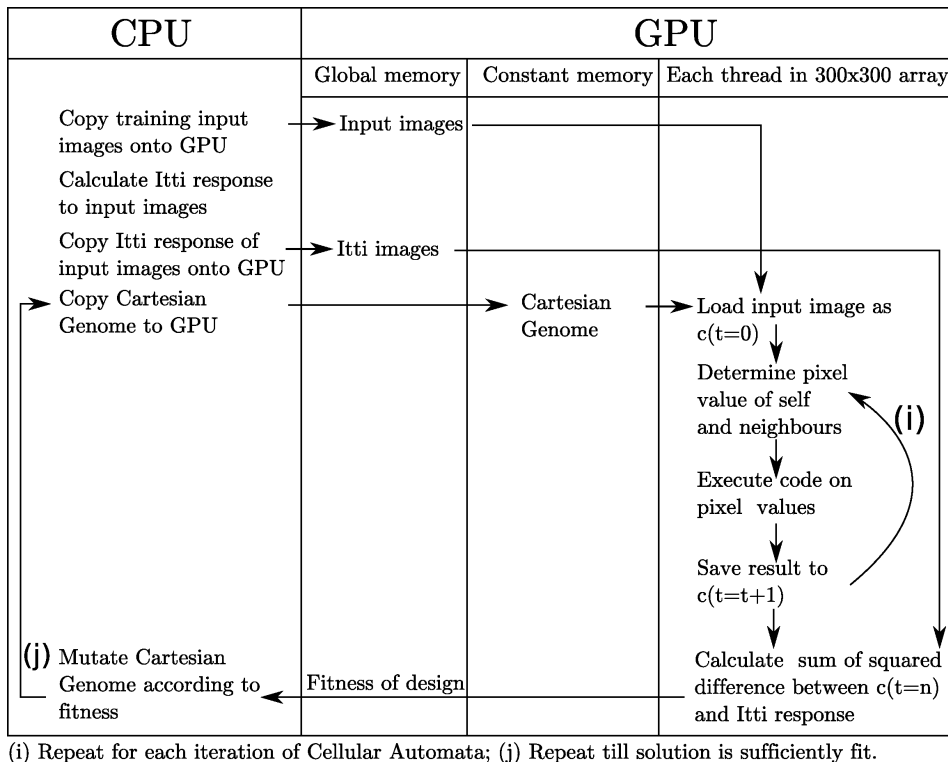


Figure 1: Delegation of tasks between the CPU and GPU

### 3.1 Performance analysis of GPU over CPU implementations

We benchmarked our GPU implementation against an equivalent single CPU implementation and tested various CA configurations of differing size, number of components that form the next-state rule and the number of iterations. The default CA

<sup>1</sup>Code available at [cas.ee.ic.ac.uk/people/dhjones](http://cas.ee.ic.ac.uk/people/dhjones)

was a 300x300 array of cells using a next-state rule made up of 100 components and iterated 20 times. Figure 2(a) shows that the GPU implementation runs up to 97 times faster than a CPU equivalent as the array size increases. Figure 2(b) shows that the GPU implementation runs up to 82 times faster than a CPU equivalent as the number of iterations of the CA increases. Figure 2(c) shows that the GPU implementation runs up to 45 times faster than a CPU equivalent as the length of the cartesian genome determining the next-state rule of each cell uses increases.

In all three cases the performance increases with the system complexity before rolling-off towards a constant. This suggests the overhead of a GPU implementation is significant at lower system complexities but negligible as they increase. That the performance increase is most significant for larger arrays is due to their better utilizing the available parallelism of the GPU.

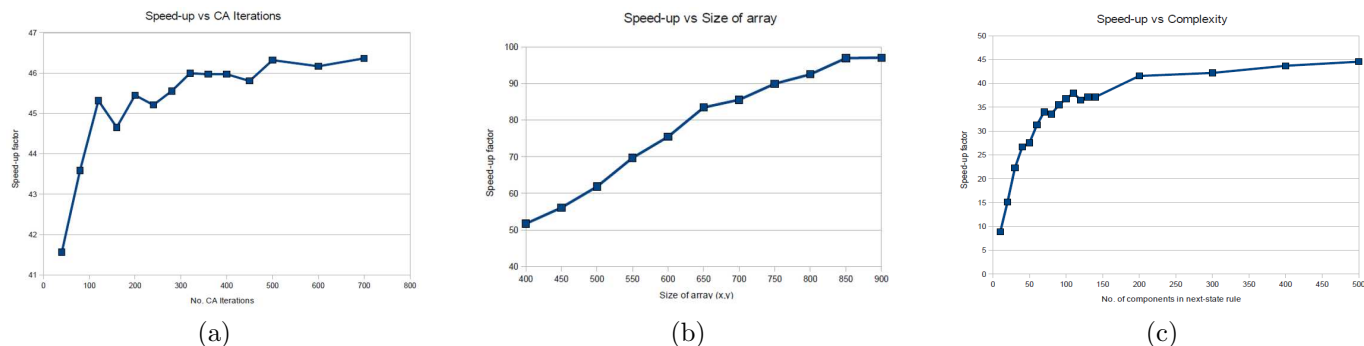


Figure 2: Improvement of GPU over CPU implementations

## 4 Results

We used the evolutionary algorithm described in section three to evolve a 300x300 cell CA with a next-state rule made up of 20 components that was iterated 20 times. This with the purpose of creating a distributed salient point detector. The evolutionary algorithm tested over a million possible solutions in two days. An equivalent CPU implementation would have required over 71 days to run.

Figure 3 shows the results of the evolved design trying to detect the salient points of various 300x300 pixel images. The salient points of each image were determined by a GTX260 graphics card in 0.14s.

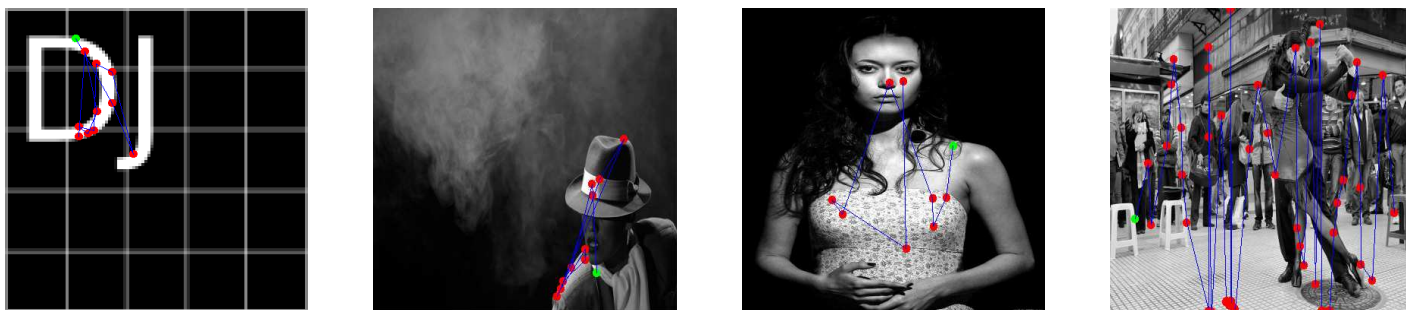


Figure 3: The salient points of four images

## 5 Conclusions

The CA on a GPU architecture is an effective platform for image processing and runs considerably faster than CPU alternatives. We have selected and adapted an evolutionary design algorithm to design the code that governs the local interactions of each cell of the CA. This design algorithm is appropriate for exploring large search spaces with multiple solutions. The designed CA has been shown to rapidly locate salient points within four images. However more work is needed to determine the invariance of the algorithm when the input is subject to noise, translation or rotation. Also the algorithm needs to be evaluated against a larger test set.

## References

- [1] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:1254 – 1259, Nov 1998.
- [2] D. Levi. Hereboj: a fast evolutionary algorithm. *Proceedings of the 2nd NASA/DoD workshop on evolvable hardware*, pages 17–24, 2000.
- [3] J. Miller. Principles in the evolutionary design of digital circuits, part 1. *Journal of genetic programming and evolvable machines*, 1, 2000.
- [4] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 1994.