Parallel Parametric Optimisation with Firefly Algorithms on Graphical Processing Units

A.V. Husselmann and K.A. Hawick
Computer Science, Institute for Information and Mathematical Sciences, Massey University, North Shore 102-904, Auckland, New Zealand
e-mail: {a.v.husselmann, k.a.hawick}@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181
March 2012

Abstract

Parametric optimisation techniques such as Particle Swarm Optimisation (PSO), Firefly algorithms (FAs), genetic algorithms (GAs) are at the centre of attention in a range of optimisation problems where local minima plague the parameter space. Variants of these algorithms deal with the problems presented by local minima in a variety of ways. A salient feature in designing algorithms such as these is the relative ease of performance testing and evaluation. In the literature, a set of well-defined functions, often with one global minimum and several local minima is available to evaluate the convergence of an algorithm. This allows for simultaneously evaluating performance as well as the quality of the solutions calculated. We report on a parallel graphical processing unit (GPU) implementation of a modified Firefly algorithm, and the associated performance and quality of this algorithm. We also discuss spatial partitioning techniques to dramatically reduce redundant entity interactions introduced by our modifications of the Firefly algorithm.

Keywords: optimisation; firefly algorithm; GPU; CUDA; spatial partitioning.

1 Introduction

Research towards metaheuristic optimisation algorithms has begun as far back as 1975, in which John Holland introduced Genetic Algorithms [1]. Several years after this discovery, Simulated Annealing followed in 1983, which is inspired by the annealing process in metallurgy [2]. It was another 12 years before Kennedy and Eberhart developed the Particle Swarm Optimiser (PSO), which lead to the development of the Firefly Algorithm (FA) by Yang [3], which was introduced in

Figure 1: 65536 fireflies attempting to optimise a 3-parameter generalised Rosenbrock Function. The global minimum is at coordinates (1, 1, 1), which is near the centre of the box. The Rosenbrock function is characterised by a low-lying valley, which is easy to find, but the minimum inside this valley is more difficult to find. 2008. Yang observed better performance in the Firefly algorithm than the standard PSO, as did Aungkulanon and Chai-ead [4]. As can be seen by the great milestones of computational metaheuristic optimisation, algorithms in this domain generally share a natural or biological inspiration.

Parametric optimisation has been an area of interest for decades however, with early techniques such as linear programming. As computational power increased in latter years, interest and scientific inquiry has increased exponentially in the search for algorithms to effectively exploit this computing power. This is even more so with
Figure 2: A visualisation of a uniform grid datastructure, the development of multi-core chips and indeed a massively parallel environment such as NVidia’s Compute Unified Device Architecture (CUDA). Graphical Processing Units (GPUs) [5] have shown excellent ability in accelerating agent-based simulations, and continues to impress with newer algorithms.

Problems involving parametric optimisation are plentiful in the areas of image compression [6, 7], manufacturing improvement [4], structure design [8], scheduling problems [9], cryptanalysis [10], object clustering/recognition [11], economics [12–14], structure design [8, 15], Antenna design [16, 17], isospectral systems [18] and more. Generally, in parameter-based optimisation, one attempts to obtain a vector $x$ with $d$ dimensions (parameters) which minimises a scalar-valued function $f(x)$. In practice, the function $f(x)$ is not normally known. This makes parametric optimisation very versatile. However, when evaluating the performance of algorithms such as these, the function $f(x)$ is known exactly. An example of this is the Rosenbrock function, given in Equation 1. For this equation, the global minimum is $f(x, y) = 0$ for $x = y = 1$.

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$  

(1)

To facilitate the evaluation of $n$-parameter optimisers, these functions are normally generalised in some way. For Rosenbrock’s function, a common generalisation is shown in Equation 2. Similar generalisations exist for most of the other evaluation functions.

$$f(x) = \sum_{i=0}^{n-2} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$$  

(2)

For the purpose of visualisation, we restricted the number of parameters to 3. This is a severely limiting attribute in real-world applications, but the visualisation

Figure 3: A surface plot of the 3-parameter Schwefel function.

of the algorithm is extremely useful.

The Firefly algorithm [3] is a homogeneous, metaheuristic, evolutionary optimisation algorithm, and a recent addition to the particle-based optimiser family. Macroscopic effects of this algorithm is reminiscent of emergence in agent-based systems. It finds several similarities with the standard PSO algorithm, but in the original article, Yang states that the PSO is simply a special case of the Firefly algorithm. In nature, the displays of flashing lights from fireflies is associated with mating habits. This served as the source of inspiration for Yang, who idealised the biological phenomenon with a few assumptions. Firstly, the algorithm would have unisex fireflies which would be attracted to any other fireflies regardless. Secondly, the attractiveness of a firefly must be essentially proportional to the objective function. For minimisation problems which we discuss, the attractiveness of a firefly is inversely proportional to the objective function value.

The use of these homogeneous spatial agents involves each firefly performing a movement calculation based on a deterministic part and a stochastic part in conjunction with other spatially local agents. While not always local, distant fireflies have a degraded influence. Depending on the parameter count, the parameter space is easily modified (albeit less able to be visualised).

By making use of what can be seen as the de facto standard naming convention, the $\alpha$-step and the $\beta$-step, every firefly is influenced by every other firefly in a deterministic and stochastic way respectively. The $\alpha$-step refers to the stochastic space exploration, and the $\beta$-step refers to the deterministic bias towards other fireflies with better solutions in the parameter space.

As described by Yang, the update formula for any two fireflies $x_i$ and $x_j$ is shown in equation 3.
Apart from advancing the firefly based on its previous position, the equation begins with the $\beta$-step. $\beta_0$ refers to an attractiveness coefficient, while the exponential $e^{-\gamma r_{ij}^2}$ degrades the attractiveness of firefly $x_j$ over distance $r_{ij}$. The second term is the $\alpha$-step. The $\alpha$ variable is the random step size, where $d$ refers to a random vector.

As can be seen from this equation, the algorithm has an inherent $O(n^2)$ complexity, since every firefly $x_i$ must evaluate this equation $n$ times, for every other firefly $x_j$. This complexity is not normally reduced because of its scaling problems, but because of diversity loss in fireflies in the parameter space [19]. There are a handful of algorithms which attempt to rectify this problem, most of which involve maintaining multiple independent swarms [19, 20]. Our algorithm differs from these approaches by very loosely maintaining separate swarms, and allowing full interaction between these swarms. By limiting the interaction distance, we also permit ourselves to use acceleration data-structures to dramatically increase performance and hence allow the use of much, much larger systems which greatly increase accuracy and reduces computation time necessary.

Some report that the FA responds well to hybridisation, especially with algorithms from the realm of artificial intelligence [19, 21]. As with PSO, the FA does require tuning, but far less than the standard PSO.

Some popular functions for evaluating the performance of optimisation algorithms include the Rosenbrock function, Ackley’s Path function, the Schwefel Function and the Rastrigin function. Each of these have vastly different appearances, and their 3-parameter counterparts are shown in Figures 2, 4, 5 and 3. For visualisation, these functions are best constrained to 3 parameters (dimensions) or less.

It is important to note that these have very different characteristics. As we observed from our results in Section 3, unimodal and multimodal test functions can have staggering performance implications.

2 CUDA Fixed-Interaction Firefly Algorithm

There have been various attempts to parallelise swarm intelligence and particularly the PSO [20, 22–24]. A few of these include CUDA implementations.

Our algorithm differs greatly in two aspects from the original FA. Firstly, our algorithm can support large numbers of fireflies due to optimised data-structures. The second difference is that we impose a maximum interaction distance, which is dramatically smaller than the values some authors have suggested. In this limited interaction distance, we still degrade attractiveness of fireflies that are more distance, but at roughly the same rate as the original algorithm would. These two differences go hand-in-hand as the interaction distance allows us to accelerate the simulation. According to authors who implement multi-swarm modifications in the FA, smaller interaction distances improves accuracy when there are local minima to avoid, as is often the case [19].

Following from Equation 3, we first modified the update to incorporate our smaller interaction distance. This is shown in Equation 4.

$$x_{i+1} = x_i + \beta e^{-\gamma r_{ij}^2} (x_j - x_i) + \alpha(d)$$ (4)
Figure 6: A surface plot of the 3-parameter version of Ackley’s Path function.

Where we specifically choose $\gamma$ to be

$$\gamma = \frac{1}{g^2(1 + \ln(\beta))}$$

with $g$ being size of a grid box. This will ensure that the $\beta$ step will give less notice to fireflies that are at the edge of visibility to a particular firefly. In our experience, this does not seem to provide far better results, but it will preserve some diversity. Instead of this, for simplicity and speed, we multiply the $\beta$-step with $N(r)$, as defined below.

$$N(r) = \begin{cases} 
0 & r > m \\
r & r < m 
\end{cases}$$

This is very simple, and by choosing a suitable $\gamma$, we need not add extra complexity as indicated above.

When optimising a function, it is beneficial to have as many fireflies or agents as possible to avoid convergence to local minima. We are able to simulate large numbers of fireflies with this algorithm by means of acceleration data-structures which dramatically reduce redundant entity interactions. We use a method named grid-boxing parallelised for CUDA (sometimes known as uniform-grid space partitioning) [25] to localise interactions of fireflies, in order to improve performance greatly. Our single-GPU implementation of this algorithm is similar to the data-structure used by the Particle simulation shipped with the CUDA SDK [26]. This allows separate (yet fully interacting) swarms to search in parallel, while still allowing complete interaction between these swarms. The simulation is not initialised with a random set of independent swarms. Rather, these seemingly separate swarms spontaneously form as a side effect of using grid-boxing with a suitably chosen interaction distance.

Figure 7: 4096 fireflies solving a 3-parameter De Jong function.

Figure 8: 262,244 fireflies optimising a 3-parameter Rosenbrock function.

The $\alpha$-step in the update formula requires a random vector $d$. This is a significant problem, considering that this code executes almost entirely on GPU. Fortunately, a library named CURAND is available directly from NVidia, distributed within the CUDA SDK. This library provides a high performance Mersenne Twistor(MT), which we use to generate enough $d$ vectors to satisfy the $\alpha$-steps for all the fireflies in the simulation. There is a complex tradeoff between quality and raw performance of random number generators [27], but we believe the MT is a suitable compromise for both criteria for teh work reported here.

3 Performance Evaluation

In our performance evaluations, we primarily used an Intel Core i7 server running at 3.4GHz, configured with two NVidia GTX 590 graphics cards. The implementation of the original firefly algorithm was obtained from [28]. This program includes a hybridised Firefly algorthm, which we did not use in our evaluations. This modified algorithm by Mancuso modifies the size of the $\alpha$ step size before every simulation step.
Table 1: CPU vs GPU Parallel Firefly algorithm in optimising the 3-parameter Rosenbrock function.

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rosenbrock 3D</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (msec)</td>
<td>368455.2</td>
<td>9488.725</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.000071</td>
<td>0.000045</td>
</tr>
<tr>
<td><strong>Rastrigin 3D</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (msec)</td>
<td>367329.1</td>
<td>966.5312</td>
</tr>
<tr>
<td>Minimum</td>
<td>1.445014</td>
<td>1.174144</td>
</tr>
<tr>
<td><strong>Schwefel 3D</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (msec)</td>
<td>369934.9</td>
<td>848.6163</td>
</tr>
<tr>
<td>Minimum</td>
<td>73.26784</td>
<td>25.47458</td>
</tr>
<tr>
<td><strong>Ackley’s Path 3D</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (msec)</td>
<td>368384.2</td>
<td>949.9359</td>
</tr>
<tr>
<td>Minimum</td>
<td>1.138163</td>
<td>3.060646</td>
</tr>
</tbody>
</table>

Table 2: CPU vs GPU Parallel Firefly algorithm in optimising the 3-parameter Rosenbrock function.

Table 3: The optima for each of the four algorithms is as follows:

1. Rosenbrock function:
   for \( f(x, y, z) \), \( f(1, 1, 1) = 0 \).

2. Ackley’s Path function:
   for \( f(x, y, z) \), \( f(0, 0, 0) = 0 \).

3. Rastrigin function:
   for \( f(x, y, z) \), \( f(0, 0, 0) = 0 \).

4. Schwefel function:
   for \( f(x, y, z) \), \( f(420.9, 420.9, 420.9) = 0 \).

We modified our Schwefel function by adding \( 3(418.929) \) to the function, in order to yield a minimum of 0.

The algorithms were both configured to randomly distribute particles in the ranges described in Table 3 in each parameter range. We removed boundary checks in the CPU Firefly algorithm to more closely resemble the implementation of our GPU algorithm, but the CPU algorithm is implemented in double precision, whereas the GPU algorithm is implemented in single precision. This makes comparison more difficult, but with the great margin of performance difference obtained, the difference becomes more pronounced.

The results show a clear difference in performance, both in computation time, and accuracy of the solution, albeit, the latter is not true for Ackley’s Path function. The accuracy of the algorithm can be attributed to the difference in interaction distance, making it more resistant to converging to a local minimum instead of the global minimum. The decreased computation time can be immediately attributed to the parallel environment. The local minima of the Ackley’s Path function is not as pronounced as the Schwefel function or Rastrigin function, so it seems possible that global interaction is indeed more suitable for this function. Another consideration is that all particles in the CPU Firefly algorithm will eventually reach the area around the global minimum, which means there will be more fireflies in that area.

So far we have compared the quality of the optimisers and the performance to some extent, but good per-
formance is possible to achieve with far less fireflies, depending on the application (unimodal or multimodal functions). Our testing revealed that the CPU Firefly algorithm can easily outperform the GPU algorithm with a simple unimodal function such as the Rosenbrock function, by simply having 16 fireflies which obtains an error of less than 0.00005 in approximately 25msec (10-run average). The GPU counterpart is only effective from 4096 fireflies up, and this takes on a 10-run average approximately 338msec for an error less than 0.00005. A clear advantage. However, when facing multimodal functions such as the Schwefel function, Rastrigin function and Ackley’s path function with low numbers of fireflies, the CPU algorithm either fails to achieve the global minimum, or takes an inordinate amount of time to compute it (days, weeks).

The GPU algorithm can easily optimise multimodal functions such as the Schwefel function in a fraction of the time it would take the CPU algorithm. Roughly 2048 or more fireflies would be needed to ensure the algorithm saturates the parameter space enough to obtain the global minimum. As seen above, the GPU speed-up over the CPU firefly algorithm is 39 times for 4096 fireflies, this would mean the GPU algorithm is far superior in multi-modal optimisation, whereas the CPU firefly algorithm is far superior in unimodal optimisation, where large numbers of fireflies are not necessary. It is difficult to compare with absolute certainty in meta-heuristic optimisation, but in our observations, we could not use the CPU firefly algorithm for optimising the multimodal functions with less than 2000 fireflies - Schwefel, Rastrigin, and Ackley’s Path in reasonable time at all, whereas the GPU algorithm could optimise all three within one second.

4 Discussion

Our algorithm seems to perform more accurately and faster than the original Firefly algorithm. There is however, a clear difference when comparing multimodal functions to unimodal functions. The original Firefly algorithm is very well suited to optimising unimodal functions, as very few fireflies are required, and frame calculation times are dramatically lower. However, for multimodal functions, it is imperative to have a larger number of fireflies to avoid local minima. This results in an $O(n^2)$ complexity which our algorithm reduces to $O(n \log n)$ and also parallelises. The acceleration data-structure allows this, thanks to a small modification we made to the original firefly algorithm, namely the smaller and fixed interaction distance.

Using ideas contained within this new GPU algorithm, it is possible to dramatically increase system sizes, and efficiently optimise multimodal functions. However, it is well worth noting that we only discuss 3-parameter functions in this article. In practice, it is common to find functions which require hundreds of parameters. The Firefly algorithm easily adapts to this by simply moving through n-dimensional space towards other fireflies, and having an n-dimensional random step. A CUDA implementation which allows n-dimensional optimisation will require extra considerations to be made, as it will require far more storage among others.

We observe that the greater the number of fireflies, the more likely it is that the global minimum will be obtained. This is certainly the case with Ackley’s Path function, as the GPU algorithm could only obtain the global minimum ($0$) consistently when run with 266,144 fireflies. This serves to saturate the parameter space to the extent that the global minimum is obtained. It is noteworthy however, that with 262,144 fireflies, it takes substantially longer to compute 600 frames, but it is still within two minutes on average.

5 Conclusions and Future Work

We have presented a GPU-based Firefly algorithm with a fixed-interaction distance and a uniform-grid acceleration data-structure. We compared this algorithm in accuracy and performance to the original single-grid acceleration Firefly algorithm, and found a vast performance increase, but only for multimodal test functions. Global interaction in very few numbers of fireflies could still outperform the GPU algorithm in unimodal function such as Rosenbrock’s function. However, we observed that the multimodal test functions could only be optimised with the GPU algorithm, and it consistently does so in less than one second for 4096 fireflies. This is approximately a 39-fold speedup over the same number of fireflies simulated by the CPU-based algorithm.

We conclude that for massively multimodal functions such as the Schwefel function, the GPU algorithm is by far the better choice. This is especially applicable to functions which also have geometrically distant global minima.

In future we will explore methods of effectively parallelising this algorithm across several GPUs in order to increase system sizes with a much smaller computational cost. We will also aim to support n-dimensional optimisation problems. These data-parallelisable optimisation algorithms show great promise across a range of complex systems applications.
References