Implementation of Parallel Firefly Algorithm

Juby Mathew¹, R Vijayakumar²

¹AmalJyothi College of Engineering, Kanjirapally, Kerala
²School of Computer Sciences, Mahatma Gandhi University, Kottayam, Kerala

Abstract
Clustering is the process of assigning data objects into a set of disjoint groups called clusters so that objects in each cluster are more similar to each other than objects from different clusters. We try to exploit computational power from the multicore processors. Firefly algorithm is one of the metaheuristic algorithms which are used for solving optimization problems. This research work analyzes the performance of Parallel FA algorithm using Java Join and Fork Method. It is the most effective design method for achieving good parallel performance. Here, the parallel architecture will be developed by including the process like, splitting the input data, clustering each subset of data and merging to optimal final clustering. Firefly-based clustering is a recent method which proves better for optimal clustering finding. In order to utilize the intrinsic capabilities of a multi-core processor the software application must be able to execute tasks in parallel using all available CPUs.

IndexTerms:- Clustering, Firefly algorithm, join and fork.

1. INTRODUCTION
Firefly algorithm is one of the powerful population–based algorithms inspired by nature which is used for solving optimization problems[1]. The field of nature inspired computing and optimization techniques have evolved to solve the difficult optimization problems in diverse fields of engineering, science and technology. The Firefly algorithm is one of the several nature inspired algorithms that have been developed in the recent past and is inspired from the flashing behaviour of the fireflies. The flashing behaviour of the fireflies is to attract other fireflies in the group for mating. The less bright firefly will be attracted by the brighter one. As all the fireflies are assumed to be unisexual, each firefly is attracted to the other. This process is mimicked in the algorithm to find the solution to objective function.

This swarm intelligence optimization technique is based on the assumption that solution of an optimization problem can be shown as a firefly which glows proportionally to its quality in a considered problem setting. Consequently, each brighter firefly attracts its partners, which makes the search space being explored efficiently. Yang used the FA for nonlinear design problems [2] and multimodal optimization problems [3] and showed the efficiency of the FA for finding global optima in two dimensional environments.

Fork/Join parallelism [4] is a style of parallel programming useful for exploiting the parallelism inherent in divide and conquer algorithms. Fork-join executor framework has been created which is responsible for creating one new task object which is again responsible for creating new sub-task object and waiting for sub-task to be completed. Internally it maintains a thread pool and executor assign pending task to this thread pool to complete when one task is waiting for another task to complete.

Fork/Join programs can be run using any framework that supports construction of subtasks that are executed in parallel, along with a mechanism for waiting out their completion.

However, the java.lang.Thread class (as well as POSIX pthreads, upon which Java threads are often based) are suboptimal vehicles for supporting fork/join programs:

- Fork/join tasks have simple and regular synchronization and management requirements. The computation graphs produced by fork/join tasks admit much more efficient scheduling tactics than needed for general-purpose threads.
  - For example, fork/join tasks never need to block except to wait out subtasks. Thus, the overhead and bookkeeping necessary for tracking blocked general-purpose threads are wasted.
  - Given reasonable base task granularities, the cost of constructing and managing a thread can be greater than the computation time of the task itself. While granularities can and should be subject to tuning when running programs on particular platforms, the extremely coarse granularities necessary to outweigh thread overhead limits opportunities for exploiting parallelism.

In short, standard thread frameworks are just too heavy to support most fork/join programs. But since threads form the basis of many other styles of concurrent and parallel programming as well, it is impossible (or at least impractical) to move overhead or tune scheduling of threads themselves just for the sake of supporting this style. The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents the proposed implementation. Section 4 shows experimental results and evaluations. Finally, the conclusions and future work are presented in Section 5.

2. RELATED WORK
To obtain acceptable computational speed on huge datasets, most researchers turn to parallelizing scheme. Here, we review some of the techniques presented for
Firefly algorithm

The firefly algorithm (FA) is a metaheuristic algorithm, inspired by the flashing behaviour of fireflies. The primary purpose for a firefly's flash is to act as a signal system to attract other fireflies. [7]

a. All fireflies are unisexual, so that one firefly will be attracted to all other fireflies regardless of their sex.
b. Attractiveness is proportional to their brightness, and for any two fireflies, the less bright one will be attracted by the brighter one.
c. If there are no fireflies brighter than a given firefly, it will move randomly.

The brightness should be associated with the objective function.

Assume that there exists a swarm of n agents (fireflies) and Xi represents a solution for a firefly i, whereas f (Xi) denotes its fitness value[8]. Here the brightness I of a firefly is selected to reflect its current position x of its fitness value f (x).

\[ I_i = f (X_i), 1 \leq i \leq n \]  

(1)

Firefly attractiveness is proportional to the light intensity seen by adjacent fireflies [1]. Each firefly has its distinctive attractiveness \( \beta \) which implies how strong it attracts other members of the swarm. However, the attractiveness \( \beta \) is relative, it will vary with the distance \( r \) between two fireflies i and j at locations \( x_i \) and \( x_j \) respectively, is given as

\[ r_g = ||x_i - x_j|| \]  

(2)

The degree of attractiveness of a firefly is determined by

\[ \beta(r) = \beta_0 e^{-\gamma r^2} \]  

(3)

where \( \beta_0 \) is the attractiveness at \( r = 0 \) and \( \gamma \) is the light absorption coefficient at the source.

The movement of a firefly i at location \( x_i \) attracted to another more attractive (brighter) firefly j at location \( x_j \) is determined by

\[ x_i(t + 1) = x_i(t) + \beta e^{-\gamma r^2}(x_j - x_i) \]  

(4)

Algorithm 1: firefly algorithm

Initialize algorithm parameters.

Define light absorption co efficient \( \gamma \)
Define maxGenerations

While(t<maxGenerations)

for i = 1 to no.of fireflies

for j = 1 to no.of fireflies

If (I_i < I_j)

Move firefly i toward j using Eq. (4)

end if

Evaluate new solutions and update light intensity using Eq. (1)

end for j

end for i

Rank the fireflies and find the current best

End while

Optimization using Multi core

Multi-core processors can deliver significant performance benefits for multi-threaded software by adding processing power with minimal latency, given the proximity of the processors. Multicore processors are now widespread across server, desktop, and laptop hardware. They are also making their way into smaller devices, such as smartphones and tablets. They open new possibilities for concurrent programming because the threads of a process can be executed on several cores in parallel. One important technique for achieving maximal performance in applications is the ability to split intensive tasks into chunks that can be performed in parallel to maximize the use of computational power. When designing software to run on a multi-core or multiprocessor system, the main consideration is how to allocate the work that will be done on the available processors. The most common way to allocate this work is by using a threading model where the work can be broken down to separate execution units that can run on different processors in parallel. If the threads are completely independent of one another, their design...
does not have to consider how they will interact. For example, two programs running on a system as separate processes each on its own core do not have any awareness of each other. Performance of the programs is not affected unless they contend for a shared resource such as system memory or the same I/O device. The Fork/Join Framework enhances multithreaded programming in two important ways. First, it simplifies the creation and use of multiple threads. Second, it automatically makes use of multiple processors.

3. PROPOSED WORK

The aim of the proposed method is to cluster a large dataset efficiently. Here a scalable parallel clustering algorithm is used to overcome the problem in clustering large dataset with high dimension.

We developed a new method for clustering the merge data by firefly algorithm. In general, the firefly base clustering algorithm can reach the optimum point of the function very quickly. In the first step of our firefly method each firefly is randomly generated. The proposed algorithm concentrates on distance calculation between each point and the k centers, performs these calculations in parallel and optimized way. If we have m cores and n data points, then each core will approximately calculate the distances between n/m points and k centers. As m increase, the amount of calculation per each core will decrease.

Algorithm 2: parallel firefly algorithm

1. Initialize a population of fireflies
2. Define light absorption coefficient γ
3. Find number of cores P
4. Partition data to P subgroups
5. for each P
6. for i=1 to no.of fireflies
   for j=1:i
   If (Ij > Ii) then
      Move firefly i towards j in all d dimensions
   End if
   End for j
   End for i
7. Receive cluster members of k cluster from P Process
8. Recalculate new centroid
9. If k stable centroid reached
10. Else go to step 4

Our approach consists of parallelizing the first phase of each step, where we calculate the nearest centroid to each point. Following code can be used to calculate cluster centers using fireflies.

```java
public class flies {
    public static int numflies = 2;
    public static double dist[] = new double[2];
    public static ArrayList<Double> min = new ArrayList<Double>();
    public static ArrayList<Double> max = new ArrayList<Double>();
    public static void assign() {
        ArrayList<Integer> ch = new ArrayList<Integer>pta;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < numflies; j++) {
                ArrayList<Double> temp = new ArrayList<Double>();
                int v = r.nextInt(data.size());
                if (ch.contains(v)) {
                    j--;
                } else {
                    ArrayList<Double> temp = new ArrayList<Double>();
                }
            }
        }
    }
}
```

The Fork/Join Framework enhances multithreaded programming in two ways. First, it simplifies the creation and use of multiple threads. Second, it automatically makes use of multiple processors. Fork/Join Framework adds two main classes to the java.util.concurrent package:

- ForkJoinPool
- ForkJoinTask

The execution of ForkJoinTask takes place within a ForkJoinPool, which manages the execution of the tasks. ForkJoinTask objects support the creation of subtasks and waiting for the subtasks to complete. Advantage of the ForkJoinPool, is that it can 'steal' work. It means that it allows one thread that has finished a task to immediately execute another task with much less overhead than the ExecutorService. [9]

The RecursiveAction and RecursiveTask are the only two direct, known subclasses of ForkJoinTask. The only difference between these two classes is that the RecursiveAction does not return a value while RecursiveTask does have a return value and returns an object of specified type.

**ForkJoinTask objects feature two specific methods:**

The fork () method allows a new ForkJoinTask to be launched from an existing one. In turn, the join () method allows a ForkJoinTask to wait for the completion of another one. A ForkJoinTask instance is very light weight when compared to a normal Java thread. [4]

Fork/join parallelism is implemented by means of a fixed pool of worker threads. We set the number of worker threads in the fork/join pool to a maximum of four, which is the number of cores in a node. Each worker thread can execute one task at a time. Tasks waiting to be executed are stored in a queue, which is owned by a particular worker thread. Currently executing tasks can dynamically generate (i.e. fork) new tasks, which are then enqueued for subsequent execution.
4. EXPERIMENTAL RESULTS

The code is executed on dell inspiron N4030 Laptop, Intel(R) Core(TM) i5 Processor 2.67 GHz, 2 MB cache memory, 3GB RAM, 64-bit Windows 7 Home and NetBeans IDE 8.0.

For testing purposes, we used following well-known unconstrained benchmark functions:
- Ackley
- Griewank
- Rastrigin

Because of the fact that these are standard benchmark functions, their definition is omitted. I am using same parametersused by Milos Subotic[10].

The CPU usage shows that all CPU power is maxed out 100% while the Fork/Join method runs. Fig 1 shows that Task Manager Performance view of the proposed algorithm is running:

Fig.1: Task manager Performance view

If we compare the results shown in Table 2 with that of [10] we get better results from our proposed approach.

Table 1: Parameter settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of function evaluation</td>
<td>1000000</td>
</tr>
<tr>
<td>calls</td>
<td></td>
</tr>
<tr>
<td>Colony size</td>
<td>40</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.01</td>
</tr>
<tr>
<td>Number of runs with different</td>
<td>30</td>
</tr>
<tr>
<td>seeds</td>
<td></td>
</tr>
<tr>
<td>Number of function parameters</td>
<td>5,10,50,500</td>
</tr>
</tbody>
</table>

We can apply this approach; speed comparison is shown in Table 2. Comparison is done for different number of parameters of objective function.

Table 2: Speed test results

<table>
<thead>
<tr>
<th>Function</th>
<th>Number of parameters</th>
<th>Multithreading</th>
<th>J/F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ackley</td>
<td>5</td>
<td>20</td>
<td>18.45</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>18.2</td>
<td>17.39</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>20.4</td>
<td>18.6</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>215.9</td>
<td>210.7</td>
</tr>
</tbody>
</table>

Above results shows that new approach is efficient in compared with multithreading approach.

5. CONCLUSION

In this paper, we propose the design and implementation ofparallel firefly using Java join and fork method. The performance of this approach was measured through four tests on standard benchmark functions. We compared speed results with existing multithread approach. From the results obtained, we can conclude that the Fork and Join method is an efficient method, which can be applied successfully to generate optimal cluster centers. Fork/join method overcomes deficiencies of multithreaded execution. In our future work it will apply different cores in different cluster size.

References

[9] Robert D Blumofe, The University of Texas at Austin, Scheduling Multithreaded Computations by Work stealing

[10] Parallelization of the Firefly Algorithm for Unconstrained Optimization Problems, Milos SUBOTIC, Milan TUBA, Nadezda STANAREVIC, Faculty of Computer Science, University Megatrend Belgrade, Latest Advances in Information Science and Applications

AUTHOR

Dr. JUBY MATHEW is a dynamic, resourceful teaching professional. He received his Ph.D. in Computer Science from Mahatma Gandhi University, Kottayam. He pursued his MCA from Periyar University Salem, MPhil in Computer Science from Madurai Kamaraj University and MTech from MS University Thirunelveli. So far he has published ten international journals and presented papers in more than twenty National and International conferences. Over twelve years of diversified teaching and corporate experience made him actively involved in all areas of education including curriculum development, student mentoring, student career preparation and community work. At present he is working as an Assistant Professor in MCA Department of Amal Jyothi College of Engineering Kanjirapally, Kerala. He won Best Faculty award as a result of his proven ability to enhance students’ performance, promising to shape a better world for the students and empower them with knowledge. He reviewed many paper publications and journals within an incredible short span of time.

Dr. R. VIJAYAKUMAR, Professor and Dean is a dedicated, ambitious and goal-driven educator and Professor of School of Computer Sciences, Mahatma Gandhi University, Kottayam. Being Graduate of College of Engineering, Trivandrum he pursued his M Tech in Computer Science from IIT Bombay and PhD from Kerala University. He started his career as Lecturer in NSS College of Engineering, Palakkad, Kerala; and he is a diligent and success driven educator with a working experience of 30 years. He holds several positions in the Universities of Kerala, Dean-Engineering and Technology, Director of College Development Council (DCDC). He has guided almost 14 students for PhD programme and another 8 students are under his guidance at present. He had published research materials in more than 25 International and National Journals.