A novel task scheduling approach for dependent non-preemptive tasks using fuzzy logic

Heba E. Hassan | Gihan Nagib | Khaled Hosny Ibrahim

Faculty of Engineering, Fayoum University, Fayoum, Egypt

Abstract

Multiprocessor task scheduling problem is a pressing problem that affects systems' performance and is still being investigated by the researchers. Finding the optimal schedules is considered to be a computationally hard problem. Recently, researchers have used fuzzy logic in the field of task scheduling to achieve optimal performance, but this area of research is still not well investigated. In addition, there are various scheduling algorithms that used fuzzy logic but most of them are often performed on uniprocessor systems. This article presents a new proposed algorithm in which the priorities of the tasks are derived from the fuzzy logic and bottom level parameter. This approach is designed to find task schedules with optimal or sub-optimal lengths in order to achieve high performance for a multiprocessor environment. With respect to the proposed algorithm, the precedence constraints between the non-preemptive tasks and their execution times are known and described by a directed acyclic graph. The number of processors is fixed, the communication costs are negligible and the processors are homogeneous. The suggested technique is tested and compared with the Prototype Standard Task Graph Set.

1 | INTRODUCTION

Task scheduling is one of the main factors which affects the multiprocessor systems' performance. Task scheduling problem is considered to be NP-complete [1], that is, it takes a lot of time to find the optimal schedule. In multiprocessor systems, the tasks must be allocated into the processors such that the total Makespan must be as low as possible. There are many approaches that have been developed to resolve the task scheduling problem on the multiprocessor system. Some of them are heuristic-based approaches [2–4]; some depend on evolutionary approaches [5–12] and some follow the hybrid methods [13–16].

There are various heuristic-based procedures for solving the multiprocessor task scheduling approach. The best of them are based on the task list technique, which can be called list scheduling [17]. It is commonly affirmed as a promising approach as it pairs low complexity with good results. It depends on making a list of tasks which are ordered in a descending manner according to their priorities. Then, the first task in the list is selected and assigned to a processor. This approach can be categorized into static and dynamic scheduling approaches. List-scheduling techniques schedule tasks in the order of their priorities. Most list-scheduling techniques consist of two phases. The former is the task prioritizing phase in which the priority is calculated and given to each task of the Directed Acyclic Graph (DAG), and the latter is the processor selection phase in which each task is assigned a processor in the order of its priority. The scheduling Algorithm is considered as static if the processor selection phase begins subsequent to end of the task prioritizing phase and it is considered dynamic if the two phases are interleaved. In another meaning, Static scheduling of tasks occurs throughout the compiled time prior to running the parallel application. On the other hand, in dynamic scheduling algorithms, scheduling decisions are made within run time. Assigning priority is an important issue which affects the performance of the scheduling techniques. There are many attributes that can be used. Some of the scheduling
techniques use the top level (t-level) and bottom level (b-level) for assigning priorities to tasks. There are several level-based heuristics [18] such as Highest Level First with Estimated Times (HLFET), Highest Levels First with No Estimated Times (HLFNET), Smallest Co-levels First with Estimated Times (SCFET), Smallest Co-levels First with No Estimated Times (SCFNET), Critical Path/most Immediate Successors First (CP/MISF) [3] and Heavy Node First (HNF) [17]. All these attributes perform based on the level concept in the DAG without taking the communication cost into account. Table 1 summarizes the contributions of these algorithms and the proposed algorithm.

In recent years, numerous researchers have made use of several fuzzy techniques in the field of task scheduling so as to obtain an optimal performance, but this area of research is still not well investigated. On the other hand, multiple scheduling algorithms have been considered to guarantee the time constraints of real-time processes. The way in which these algorithms make a decision is usually based on parameters that are considered to be crisp [19,20]. Nevertheless, in many situations, the values of those parameters are vague. This vagueness proposes that we can benefit from fuzzy logic to determine the order in which the requests have to be executed for proper utilization of the system and to diminish the chance of missing

| Algorithm | Contribution |
|-----------|--------------|
| Highest level first with estimated times (HLFET) [18,25,26] | • It is considered to be the first list-scheduling algorithm which is designed for homogeneous multiprocessor systems. It is a priority-based algorithm that uses the static b-level for assigning priorities to tasks.  
• No communication costs are regarded in this algorithm.  
• The drawback of the HLEFT algorithm is that it does not utilise the idle time slot between the two processors which degrades its performance. |
| Highest levels first with No estimated times (HLFNET) [18,25,26] | • In this algorithm, all tasks are assumed to have the same execution time and their priorities are their levels.  
• It can be applied when HFLET is desirable, but the tasks’ execution times cannot be known in advance. |
| Smallest co-levels first with estimated times (SCFET) [18,25,26] | • It is a list scheduling algorithm in which the task’s priority is dependent on its co-level.  
• The co-level of a task is the summation of the longest path from the entry node to the desired node. The smaller the task’s co-level the higher its priority. |
| Smallest co-levels first with no estimated times (SCFNET) [18,25,26] | • It is the same as SCFET except that all the tasks are assigned equal weights assuming them to have the same execution time. |
| Critical path/most immediate successors first (CP/MISF) [3,26,27] | • The CP/MISF is proposed as a variation of the HLFet algorithm.  
• It assigned high priorities to the tasks of high-level values and the number of their immediate successors is large. |
| Heavy node first (HNF) [17,27] | • The general idea behind HNF is to use local analysis of DAG nodes at each level. Tasks are assigned level by level and the heaviest node is assigned first.  
• It produces an optimal solution when tasks are assigned a unit weight and the number of nodes is small.  
• Since the HNF uses local information from the DAG, the DAG can be divided into small levels and each one can be processed independently. |
| New fuzzy scheduling algorithms (NFSA) [23] | • It is designed for independent periodic tasks and their periods are equal to their deadlines.  
• This algorithm assigns priorities to tasks using a single fuzzy inference engine (FIS). The inputs to the FIS are the task’s arrival time, deadline, and computation time. While the output is the runtime priority of the task which is used to schedule tasks on a multiprocessor. |
| [24] | • The author has not given a name to this algorithm. It is designed to be implemented on non-periodic independent tasks for uniprocessor soft real-time systems.  
• This algorithm depends on a couple of fuzzy inference engines in decision making. The first FIS is used to calculate the task’s priorities, while the second FIS is used to modify these priorities in the case of a new task arrival taking their deadlines into account. |
| Priority-fuzzy-B-level algorithm (PFB) [proposed] | • A scheduling algorithm that uses a single of fuzzy inference engine along with the bottom level parameter to solve the scheduling algorithm for real-time systems with non-preemptive tasks that have precedence relations and timing constraints. The communication costs between tasks are negligible.  
• Evaluating the performance metrics of the developed algorithm and comparing it with that of a selected online benchmark. The comparison is done in terms of Makespan, minimum schedule length, speedup, and efficiency. |
a request [21,22]. There are numerous scheduling algorithms that use fuzzy logic in scheduling such as [23] but some of them are often performed on uniprocessor systems such as [24], as shown in Table 1.

Consider a Directed Acyclic Graph $G$ that can be simply defined as $G = (V, E)$ where $V$ is a set of nodes (tasks) and $E$ is the set of directed edges which represent the precedence constraints between the tasks. Each Edge has a source node which is the parent node and a sink node which is the child node. The node which has no parent is called 'Entry node', while the node which has no children is called 'Exit node'. So, for any edge $E_{ij}$ between the two nodes $T_i$ and $T_j$, $T_i$ precedes $T_j$, the height of any task $T_i$ can be calculated using equation:

\[
\text{height}(T_i) = \begin{cases} 
0 & \text{if } \text{PRED}(T_i) = \emptyset \\
1 + \max\{\text{height}(T_i)\} & \text{Otherwise}
\end{cases}
\]

where $\text{PRED}(T_i)$ is the set of all nodes which precedes $T_i$, and $T_j \in \text{PRED}(T_i)$. The previous equation indicates that the height $(T_i)$ must be less than the height $(T_j)$, and $T_i$ must be executed before $T_j$ so that the precedence constraints can be preserved. Figure 1 represents an example of DAG with 11 tasks with their execution times and heights respectively, and the node label stands for the task ID. Nodes 0 and 11 are dummy nodes with zero execution times.

This article aims to propose a new algorithm called ‘Priority – Fuzzy- B-Level’ (PFB). This approach uses a fuzzy logic engine and bottom level parameter for assigning priorities to tasks. The motivation behind this work is to find a task schedule which achieves high performance for the multiprocessor system by finding optimal or sub-optimal schedule lengths. Also, the proposed algorithm is based on a deterministic model which means that precedence constraints between tasks and their execution times are known. The number of processors is fixed, and the communication costs are negligible. The precedence constraints between tasks and their execution times are described by means of a DAG. The tasks should be non-preemptive, in which the task must complete its execution before another task starts its execution on the same processor. The processors are homogeneous which means that the processors are of the same speeds or processing capabilities and they are wholly connected with each other through identical links.

The rest of this article is organized as follows: The proposed system model is explained in Section 2. Section 3 is dedicated to the simulation results and discussion. Finally, the paper is concluded in Section 4 in addition to future work directions.

2 | THE PROPOSED SYSTEM MODEL

2.1 | Scheduler model and fuzzy model

Figure 2 describes a new scheduler model for $m$ homogeneous processors. These processors are fully connected with a negligible communication cost. $P = \{p_i, i = 1, 2, 3 \ldots m\}$.

A dedicated processor called the ‘Central Scheduler’ is responsible for assigning tasks to their appropriate processors. When tasks arrive, they will stay in a queue called Task Queue in which tasks wait to be scheduled. There are two additional queues which are the Waiting Queue and the Ready Queue. The Waiting Queue is reserved for the tasks which need to be scheduled but they have to wait because of their precedence constraints. On the other hand, the Ready Queue is dedicated for the tasks which finished waiting in the Waiting Queue and need to be assigned to a processor. In addition, there is a fuzzy inference engine embedded with them, which is responsible for assigning priorities to tasks depending on their heights and execution times as stated before. All these parts are controlled by the Central Scheduler according to the proposed algorithm as it will be explained in the following section. In the end, the Scheduler will stop working when the three queues become empty.

---

**Figure 1** An example of DAG

**Figure 2** Scheduler model block diagram
Figure 3a shows the block diagram of the proposed fuzzy inference system. Its input stage contains two variables which are the height of the node in the DAG (HEIGHT) and its execution time (EXT). The output stage consists of one variable which is the priority (PRIORITY). The membership functions map these input variables to their corresponding output variables. These membership functions and the surface of the FIS are given in Figure 3b–e. Nine rules are created and a Mamdani’s type fuzzy Inference system is built. Some of these rules are mentioned below:

1. If (HEIGHT is L) and (EXT is L) then (PRIORITY is H)
2. If (HEIGHT is L) and (EXT is M) then (PRIORITY is H)
3. If (HEIGHT is L) and (EXT is H) then (PRIORITY is M)

2.2 The proposed algorithm

The main concept of PFB algorithm is making a list of tasks, assigning them some priorities and sorting them according to these priorities in a descending order. This is achieved as in the following steps:

1. Once the tasks arrive at a queue called Task Queue, the Scheduler extracts their heights and execution times and passes them to the fuzzy inference engine to assign them priorities, as stated before.
2. Then, it calculates their b-level, as shown in Equation (2) [28], is the length of the longest path from task \( T_i \) to an exit task, taking the execution cost of \( T_i \) into consideration.

\[
blevel(T_i) = w_i + \max\{C_{ij} + blevel(T_j)\}
\]  

(2)

where \( w_i \) is the execution cost of \( T_i \), \( c_{ij} \) is the communication cost, and \( b\)-level \( (T_j) \) is the \( b\)-level value of the successor node of \( T_i \). By ignoring \( c_{ij} \), the \( b\)-level Equation (2) will be:

\[
blevel(T_i) = w_i + \max\{blevel(T_j)\}
\]  

(3)

3. Then, it sorts the tasks of the Task Queue, according to their \( b\)-level in a descending order.
4. Finally, it sorts the tasks of the Task Queue, with the same \( b\)-level value according to their priorities in a descending order.

Table 2 shows priorities given by FIS, the \( b\)-level values and the order of execution of each task according to these steps.

After assigning priorities to tasks and sorting them in the Task Queue, the tasks will be assigned to the processors according to the following procedure:

Step 1. While the Ready Queue is empty and the Task Queue is not empty, check the dependency of the first node in it.

- If all of its predecessors are done, then remove it from the task queue and assign it to the first available processor.
- Otherwise, move it to the waiting queue until they finish their execution.

![FIS](image)

**Figure 3**  (a) Block diagram of the proposed FIS. (b) Membership function of Height. (c) Membership function of Execution Time. (d) Membership function of Priority. (e) Surface view of FIS
### Table 2 b-level values of each task and order of execution of each task

| Task ID | Execution Time | Height | Priority | B-Level | Order of Execution |
|---------|----------------|--------|----------|---------|--------------------|
| 0       | 0              | 0      | 8.3667   | 11      | 1                  |
| 1       | 4              | 1      | 4.7216   | 11      | 3                  |
| 2       | 2              | 1      | 6.1446   | 11      | 2                  |
| 3       | 2              | 2      | 5.2784   | 7       | 6                  |
| 4       | 1              | 2      | 5.3262   | 9       | 4                  |
| 5       | 1              | 3      | 5.000    | 8       | 5                  |
| 6       | 3              | 3      | 4.3795   | 5       | 8                  |
| 7       | 2              | 3      | 5.000    | 4       | 9                  |
| 8       | 4              | 4      | 1.7784   | 7       | 7                  |
| 9       | 1              | 4      | 4.6738   | 2       | 11                 |
| 10      | 2              | 5      | 3.8554   | 3       | 10                 |
| 11      | 0              | 6      | 5.000    | 1       | 12                 |

**Step 2.** If the Waiting Queue is not empty, then

- Move the successors of the finished task from the Waiting Queue to the Ready Queue.
- Update the status of the processor to Idle.
- If the Ready Queue is updated with a new task, sort it according to the b-level values of the tasks and sort the tasks with the same b-level values according to their priorities.

**Step 3.** While the Ready Queue is not empty,

- Remove the first task from the Ready Queue and assign it to the first idle processor.
- Repeat step 2.

**Step 4.** Repeat all the previous steps until all of Task Queue, Waiting Queue and Ready Queue become empty.

Figure 4 indicates the Gantt chart of the resulting schedule according to the proposed algorithm.

### 3 SIMULATION RESULTS AND DISCUSSION

The Prototype Standard Task Graph Set, which is available online at [29], was used as a benchmark to test the proposed PFB algorithm. It is composed of 300 task graphs, each with 50 to 2500 tasks. Table 4 shows 50 selected task graphs out of the 300 tasks. Several optimization objectives can be considered for this problem, especially this problem is multi-objective in its general form. The fundamental criterion here is minimizing the Makespan, which is, the time taken to finish the last job. This criterion can be defined using Equation (4).

\[
\text{Makespan} = \min_{S_{i \in \text{Sched}}} \left\{ \max_{j \in \text{jobs}} F_{j} \right\} \tag{4}
\]

where \( F_{j} \) is the time when job \( j \) finishes, \( \text{Sched} \) is the set of all possible schedules and \( \text{jobs} \) is the set of all jobs to be scheduled.

The simulation was carried out, using MATLAB, on all the 300 prototypes to measure the performance of the proposed PFB algorithm in terms of Makespan, Minimum Schedule Length, Speedup (\( S \)) and Efficiency (\( E \)). These metrics are obtained from Equations (4)–(7), respectively [28,30]. The following lines discuss the performance of the proposed algorithm:

\[
\text{Minimum Schedule Length} = \frac{\text{Makespan on a uniprocessor}}{\text{Number of processors}(m)} \tag{5}
\]

\[
E = \frac{\text{Makespan on a uniprocessor}}{\text{Makespan on } (m) \text{ processors}} \tag{6}
\]

\[
E = \frac{100 \times S}{m} \tag{7}
\]

A comparison was done between the resulting schedule lengths from the proposed PFB algorithm and the optimal Schedules for the Prototype Standard Task Graph Set [29]. These optimal schedules were obtained by a practical parallel optimization algorithm PDF/IHS (Parallelized Depth First/
Implicit heuristic Search) on a shared main memory multiprocessor system Sun Ultra Enterprise 3000 (E3000). The E3000 has 6 Ultra SPARC 167 MHz CPUs and 384 MB shared main memory. Because of NP-completeness of the scheduling problem, the upper limit on the search time was set to 10 min. The word ‘Time Over’ means that PDF/IHS algorithm using six processor elements could not find an optimal schedule within 10 min. The practical parallel optimization algorithm, PDF/IHS, found optimal schedules for 255 prototypes out of 300 while it could not find optimal schedules for the other 45 prototypes within 10 min, so their values were set to ‘Time Over’. The results of the PFB algorithm are shown in Table 3.

Table 3 and Figure 5 show that the PFB algorithm achieved 205 optimal schedules out of 255. It obtained a better solution once, for ‘proto293’, with schedule length 1299 [u.t.], while the optimal schedule is 1472 [u.t.] which is better than the optimal by 173 [u.t.] (11.7525%), while the minimum schedule length for this task graph is 1298 [u.t.] as shown in Table 4 and Figure 5. For the remaining 49 prototypes, it missed the optimal schedules with error values less than 0.5% 45 times, twice for error values between 0.5% and 1%, and finally, twice for error values between 1% and 2.5%. The error value is calculated from Equation (8).

**Table 3** Schedule lengths with respect to benchmark optimal schedules

| Quality of the Solution (Error) | Number of times |
|---------------------------------|-----------------|
| <0 (better than optimum)        | 1               |
| 0% (optimum)                   | 205             |
| 0%-0.5%                        | 45              |
| 0.5%-1%                        | 2               |
| 1%-2.5%                        | 2               |
| Total                          | 255             |

**Figure 5** Performance according to Makespan

**Table 4** Schedule results for 50 prototype task graph sets

| No. | Task Graph File Name | No. of Tasks | No. of Processors Length [u.t.] | Proposed Algorithm [u.t.] | Error | Minimum Schedule Length | Speedup (S) | Efficiency (E) |
|-----|----------------------|--------------|---------------------------------|---------------------------|-------|-------------------------|-------------|----------------|
| 1   | proto000             | 452          | 20                              | 537                       | 0     | 227.1                   | 8.4581005   | 42.290502      |
| 2   | proto001             | 473          | 4                               | 1179                      | 0.0848896 | 1177.75                 | 3.9957591   | 99.893977      |
| 3   | proto002             | 499          | 14                              | 341                       | 0.2932551 | 340.071428              | 13.921052   | 99.436090      |
| 4   | proto003             | 164          | 13                              | 556                       | 0     | 123                     | 2.8758992   | 22.122302      |
| 5   | proto004             | 457          | 20                              | Time over                | 222   | 218.75                  | 19.707207   | 98.536036      |
| 6   | proto005             | 404          | 7                               | 724                       | 742   | 2.4861878               | 5.1868792   | 74.123989      |
| 7   | proto006             | 273          | 19                              | Time over                | 141   | 139.473684              | 18.794526   | 98.917506      |
| 8   | proto007             | 499          | 18                              | 489                       | 0     | 264.222222              | 9.7259713   | 54.033174      |
| 9   | proto008             | 399          | 7                               | 571                       | 0.1751313 | 571                    | 6.9877622   | 99.825174      |
| 10  | proto009             | 438          | 18                              | 625                       | 0     | 233.888888              | 6.736       | 37.422222      |
| 11  | proto010             | 539          | 16                              | 334                       | 0     | 333.75                  | 15.988024   | 99.925149      |
| 12  | proto050             | 338          | 15                              | 738                       | 0     | 230.933333              | 4.6037669   | 31.291779      |
| 13  | proto051             | 416          | 16                              | 516                       | 0     | 281.5                   | 8.7286821   | 54.554263      |
| 14  | proto052             | 169          | 8                               | 298                       | 0     | 220.75                  | 5.9261745   | 74.077181      |
| 15  | proto053             | 455          | 3                               | 1581                      | 0     | 1581                    | 3           | 100            |
| 16  | proto054             | 158          | 12                              | 630                       | 0     | 134.416666              | 2.5603174   | 21.335978      |
| 17  | proto055             | 349          | 17                              | 713                       | 0     | 211.470588              | 5.0420757   | 29.659269      |
| 18  | proto056             | 349          | 12                              | 386                       | 0     | 308.166666              | 9.5803108   | 79.835924      |
| 19  | proto057             | 279          | 11                              | 575                       | 0     | 269.181818              | 5.1495652   | 46.814229      |

(Continues)
where $L_{OPTIMAL}$ is the optimal schedule length and $L_{PROPOSED}$ is the proposed algorithm schedule length.

Figure 6a shows the difference between the schedule lengths obtained by the proposed PFB algorithm and the minimum schedule lengths for each task graph from the 255 prototypes. It also shows the efficiency of the PFB algorithm for them. From this figure, it can be concluded that as the difference between the minimum schedule length and the schedule length obtained from the proposed algorithm decreases, the efficiency increases. If they are equal, the efficiency will be 100% which means that all the processors are fully utilized during all time periods of the programme.

\[ S = \frac{L_{OPTIMAL} - L_{PROPOSED}}{L_{OPTIMAL}} \times 100 \] (8)

**TABLE 4 (Continued)**

| No. | Task Graph File Name | No. of Tasks | No. of Processors (m) | Optimal Schedule Length [u.t.] | Proposed Algorithm [u.t.] | Error | Minimum Schedule Length | Speedup (S) | Efficiency (E) |
|-----|---------------------|--------------|-----------------------|-------------------------------|---------------------------|-------|-------------------------|-------------|----------------|
| 20  | proto058            | 137          | 19                    | 543                           | 543                       | 0     | 76.3615789             | 2.6813996   | 14.112629      |
| 21  | proto059            | 421          | 7                     | 618                           | 619                       | 0.161812 | 617.142857             | 6.9789983   | 99.699976      |
| 22  | proto240            | 2264         | 12                    | 1985                          | 1985                      | 0     | 1984.58333             | 11.997481   | 99.979009      |
| 23  | proto241            | 2187         | 12                    | 1949                          | 1949                      | 0     | 1948.25                | 11.995382   | 99.961518      |
| 24  | proto242            | 2131         | 12                    | 1880                          | 1880                      | 0     | 1879.41666             | 11.996276   | 99.968971      |
| 25  | proto243            | 2166         | 20                    | 1126                          | 1126                      | 0     | 1125.3                 | 19.987566   | 99.93783       |
| 26  | proto244            | 2426         | 17                    | 1470                          | 1470                      | 0     | 1469.41176             | 16.993197   | 99.959984      |
| 27  | proto245            | 2156         | 10                    | 2290                          | 2290                      | 0     | 2289.2                 | 9.9965065   | 99.965065      |
| 28  | proto246            | 2199         | 3                     | 7572                          | 7572                      | 0     | 7571.66666             | 2.9998769   | 99.995979      |
| 29  | proto247            | 2351         | 14                    | 1778                          | 1779                      | 0.0562429 | 1777.64285             | 13.989319   | 99.923713      |
| 30  | proto248            | 2215         | 19                    | Time over                     | 1233                      | –     | 1231.78947             | 18.981346   | 99.901822      |
| 31  | proto249            | 2080         | 9                     | 2407                          | 2407                      | 0     | 2406.66666             | 8.9987536   | 99.986151      |
| 32  | proto250            | 447          | 10                    | 560                           | 560                       | 0     | 443.7                  | 7.9232142   | 79.232142      |
| 33  | proto251            | 160          | 11                    | 276                           | 276                       | 0     | 177.72727              | 7.0833333   | 64.393939      |
| 34  | proto252            | 132          | 14                    | 195                           | 195                       | 0     | 90.3571428             | 6.4871794   | 46.336096      |
| 35  | proto253            | 342          | 7                     | 495                           | 495                       | 0     | 495                    | 7           | 100            |
| 36  | proto254            | 250          | 8                     | 345                           | 345                       | 0     | 5797.101               | 7.9510086   | 99.387608      |
| 37  | proto255            | 138          | 12                    | 360                           | 360                       | 0     | 122.416666             | 4.0805555   | 34.004629      |
| 38  | proto256            | 163          | 11                    | 292                           | 292                       | 0     | 161.72727              | 6.0924657   | 55.386052      |
| 39  | proto257            | 157          | 5                     | 460                           | 460                       | 0     | 347.8                  | 3.7804347   | 75.680695      |
| 40  | proto258            | 145          | 14                    | 359                           | 359                       | 0     | 106                    | 4.1337047   | 29.524662      |
| 41  | proto290            | 2133         | 16                    | 1428                          | 1428                      | 0     | 1428                   | 16          | 100            |
| 42  | proto291            | 2122         | 9                     | 2591                          | 2591                      | 0     | 2590.55555             | 8.9984561   | 99.982846      |
| 43  | proto292            | 2333         | 3                     | 8009                          | 8009                      | 0     | 8008.33333             | 2.997502    | 99.99167       |
| 44  | proto293            | 2089         | 17                    | 1472                          | 1299                      | –11.75271 | 1298.23529             | 16.989992   | 99.941131      |
| 45  | proto294            | 2014         | 17                    | 1257                          | 1257                      | 0     | 1256.82352             | 16.997613   | 99.98596       |
| 46  | proto295            | 2168         | 18                    | 1318                          | 1318                      | 0     | 1318                   | 18          | 100            |
| 47  | proto296            | 2162         | 9                     | 2563                          | 2563                      | 0     | 2562.33333             | 8.9976589   | 99.973988      |
| 48  | proto297            | 2136         | 3                     | 7703                          | 7703                      | 0     | 7702.66666             | 2.9998701   | 99.995672      |
| 49  | proto298            | 2399         | 10                    | 2471                          | 2471                      | 0     | 2471                   | 10          | 100            |
| 50  | proto299            | 2205         | 20                    | 1141                          | 1141                      | 0     | 1140.95                | 19.999123   | 99.995617      |
execution, for example, Proto053, Proto253, Proto295 and Proto298 in Table 4. The quality of any algorithm is strongly relevant to the speedup and efficiency. Figure 6b shows the speedup values for the 255 prototypes. It shows that the PFB algorithm has achieved satisfying speedup values which reached 19.99.

For more clarification about the performance of the PFB algorithm, Figure 7a shows the difference between the minimum schedule lengths and the schedule lengths obtained from the PFB algorithm for the 45 prototypes with ‘Time Over’ values obtained by the benchmark. It can be observed that the PFB algorithm has also achieved high-efficiency values for them reached 99.934%. While Figure 7b shows the speedup values of them which reached 19.937 in some cases.

4 | CONCLUSION

This paper proposed an innovative algorithm for scheduling tasks on multiprocessor systems. According to that algorithm, priorities of tasks are calculated depending on their bottom level values and the priorities given by the fuzzy inference engine. The fuzzy engine gives priorities to tasks according to their execution times and heights. After that, the tasks are assigned to the appropriate processors with the aid of the scheduler processor. The motivation behind the proposed algorithm is to increase the performance of the system by finding optimal or near-optimal schedule lengths with high efficiency and speedup values.

The Prototype Standard Task Graph Set was used as a benchmark to test the proposed algorithm. The simulation was
carried out to measure the performance of the proposed algorithm in terms of Makespan, Speedup and Efficiency. A comparison was accomplished between the resulting schedule lengths from the proposed algorithm and the optimal Schedules for Prototype Standard Task Graph Set. The quality of any algorithm is well related to the speedup and efficiency values and the proposed algorithm achieved efficiency values that reached 100% and speedup values reached that 19.99.

In the future, artificial intelligence tools like the Genetic Algorithm may be used to modify the proposed algorithm to cover more types of scheduling issues to make the proposed algorithm more general and flexible.

REFERENCES

1. Sai, R.V., Lavanya, M., Srinivasan, B.: A hybrid algorithm for multiprocessor scheduling. Int. J. Pure Appl. Math. 118(20), 3149–3155 (2018)
2. Hwang, J.J., et al.: Scheduling precedence graphs in systems with interprocessor communication times. SIAM J. Comput. 18(2), 244–257 (1989)
3. Kasahara, H., Narita, S.: Practical multiprocessor scheduling algorithms for efficient parallel processing. IEEE Trans. Comput. 11, 1023–1029 (1984)
4. Kwook, Y.K., Ahmad, I.: Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Trans. Parallel Distrib. Syst. 7(5), 506–521 (1996)
5. Hwang, R., Gen, M., Katayama, H.: A performance evaluation of multiprocessor scheduling with genetic algorithm. Asia Pac. Manag. Rev. 11(2) (2006)
6. Wu, A.S., et al.: An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Trans. Parallel Distrib. Syst. 15(9), 824–834 (2004)
7. Lee, Y.H., Chen, C.: A modified genetic algorithm for task scheduling in multiprocessor systems. In: The Ninth Workshop on Compiler Techniques for High Performance Computing (2003)
8. Ashouraei, M., Navimipour, N.J.: Priority-Based Task Scheduling on Heterogeneous Resources in the Expert Cloud. Kybernetes (2015)
9. Azad, P., Navimipour, N.J.: An energy-aware task scheduling in the cloud computing using a hybrid cultural and ant colony optimization algorithm. Int. J. Cloud Appl. Comput. 7(4), 20–40 (2017)
10. Dordaei, N., Navimipour, N.J.: A hybrid particle swarm optimization and hill climbing algorithm for task scheduling in the cloud environments. ICT Express. 4(4), 199–202 (2018)
11. Keshanchi, B., Navimipour, N.J.: Priority-based task scheduling in the cloud systems using a memetic algorithm. J. Circ. Syst. Comput. 25(10), 1650119 (2016)
12. Keshanchi, B., Souri, A., Navimipour, N.J.: An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing. J. Syst. Software. 124, 1–21 (2017)
13. Sivanandam, S.N., Visalakshi, P., Bhuvaneswari, A.: Multiprocessor scheduling using hybrid particle swarm optimization with dynamically varying inertia. Indian J. Chem. Sect. A. 4(3), 95–106 (2007)
14. Chen, H., Cheng, A.M.: Applying ant colony optimization to the partitioned scheduling problem for heterogeneous multiprocessors. ACM SIGBED Rev. 2(2), 11–14 (2005)
15. Ercan, M.E.: A hybrid particle swarm optimization approach for scheduling flow-shops with multiprocessor tasks. In: IEEE 2008 International Conference on Information Science and Security (ICISS), pp. 13–16., (Jan 2008)
16. Moghaddam, M.E., Bonyadi, M.R.: An immune-based genetic algorithm with reduced search space coding for multiprocessor task scheduling problem. Int. J. Parallel Program. 40(2), 225–257 (2012)
17. Shirazi, B., Wang, M., Pathak, G.: Analysis and evaluation of heuristic methods for static task scheduling. J. Parallel Distrib. Comput. 10(3), 222–232 (1990)
18. Sarangal, A., Kaur, N., & Kaur, R.: List scheduling algorithms classification: an analytical study. Int. J. Adv. Res. Comput. Sci. 9(2), pp. 385, (2018)
19. Josephson, J., Ramesh, R.: A novel algorithm for real time task scheduling in multiprocessor environment. Cluster Comput. 22(6), 13761–13771 (2019)
20. Rajaram, V., Srividhya, S., & Kumararathnam, N.: Impact of fuzzy inference system for improving the network lifetime in wireless sensor networks—a survey. In: IEEE 2018 International Conference on Communication and Signal Processing (ICCP), pp. 993–0937 (Apr 2018)
21. Hassan, H.E., Nagib, G., Ibrahim, K.H.: An evolutionary fuzzy based scheduling algorithm for soft real-time multiprocessor systems. In: 2019 7th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC), pp. 174–177., (Dec 2019)
22. Omwuachu Uzochukwu, C., Enyindah, P.: Application of fuzzy logic to predictive job shop scheduling in an interconnected system. Int. J. Comput. Appl. 975, 8887 (2016)
23. Ogedengbe, M.T., Agana, M.A.: New fuzzy techniques for real-time task scheduling on multiprocessor systems. Int J Comput Trends Technol. 47(3), 189–196 (2017)
24. Fahmy, M.M.M.: A fuzzy algorithm for scheduling non-periodic jobs on soft-real-time single processor system. Am. Shams Eng. J. 1(1), 31–38 (2010)
25. Homaifar, A., Joulapour, M.: Comparison of scheduling algorithms for efficient parallel processing of Red-time linear Control systems. In: The 24th Southeastern Symposium on System Theory and the 3rd Annual Symposium on Communications, Signal Processing Expert Systems, and ASIC VLSI Design, IEEE, pp. 120–124 (Mar 1992)
26. Salleh, S., Zomaya, A.Y.: Scheduling in Parallel Computing Systems, Fuzzy and Annealing Techniques, vol. 510. Springer Science & Business Media (2012)
27. Roy, P., et al.: Heuristic based task scheduling in multiprocessor systems with genetic algorithm by choosing the eligible processor (2012). arXiv preprint arXiv:1208.1922
28. Kaur, G., Chhabra, A., Singh, G.: Heuristics based genetic algorithm for scheduling static tasks in homogeneous parallel system. Int. J. Comput. Sci. Secur. 4(2), 183–198 (2010)
29. Standard task graph set. [Online]. http://www.kasahara.elec.waseda.ac.jp/schedule
30. Almalaethinam, D.G., Mary, G.J.: A new DAG based dynamic task scheduling algorithm (DYTAS) for multiprocessor systems. Int. J. Comput. Appl. 19(8), 24–28 (2011)

How to cite this article: Hassan HE, Nagib G, Ibrahim KH. A novel task scheduling approach for dependent non-preemptive tasks using fuzzy logic. IET Comput. Digit. Tech. 2021;15:214–222. https://doi.org/10.1049/cdt2.12018