A Fuzzy-based Multi-criteria Scheduler for Uniform Multiprocessor Real-time Systems

Vahid Salmani
Dept. of Computer Engineering, Ferdowsi University of Mashhad, Iran
salmani@um.ac.ir

Roya Ensafi
Dept. of Computer Engineering, Ferdowsi University of Mashhad, Iran
ensafi_roya@ieee.org

Narges Khatib-Astaneh
Dept. of Computer Engineering, Payame Noor University of Mashhad, Iran
narges.khatib@yahoo.com

Mahmoud Naghibzadeh
Dept. of Computer Engineering, Ferdowsi University of Mashhad, Iran
naghib@um.ac.ir

Abstract

It has been proved that there is no optimal online scheduler for uniform parallel machines. Despite its non-optimality, EDF is an appropriate algorithm to use in such environments. However, its performance significantly drops in overloaded situations. Moreover, EDF produces a relatively large number of migrations which may prove unacceptable for use on some parallel machines. In this paper a new algorithm based on fuzzy logic for scheduling soft real-time tasks on uniform multiprocessors is presented. The performance of this algorithm is then compared with that of EDF algorithm. It is shown that our proposed approach not only demonstrates a performance close to that of EDF in non-overloaded conditions but also has supremacy over EDF in overloaded situations in many aspects. Furthermore, it imposes much less overhead on the system.

1. Introduction

One of the research fields in the area of real-time systems to which more attention has been recently paid are *multiprocessor* real-time platforms that include several processors on which jobs can get executed. As multiprocessor systems are applied in real-time applications, scheduling of real-time tasks in such systems is of much significance. Two important types of multiprocessor systems are *identical* and *uniform* parallel machines. In the former the processing power of all processors is the same, whereas, each processor might have a different processing power in the latter case [1]. It has been shown that in general there is no optimal scheduling algorithm for multiprocessors [2].

Although many scheduling algorithms focus on timing constraints, there are other implicit constraints in the environment, such as uncertainty and lack of complete knowledge about the environment, dynamicity in the world, bounded validity time of information and other resource constraints. In real world situations, it would often be more realistic to find viable compromises between these parameters. For many problems, it makes sense to partially satisfy objectives. The satisfaction degree can then be used as a parameter for making a decision. One especially straightforward method to achieve this is the modeling of these parameters through fuzzy logic [3].

2. Related Work

In this part the scheduling algorithms which are served as the basis of our new approach are studied. This approach uses three criteria, namely *deadline*, *laxity* and *interval,* each of which is corresponded to EDF, LLF, and RM algorithms, respectively.

The Earliest Deadline First algorithm (EDF) [4] is a dynamic priority algorithm which uses the deadline of a task as its priority. The task with the earliest deadline has the highest priority, while the lowest priority belongs to the task with the latest deadline. This algorithm has been proved to be optimal on uniprocessors [5].

The Least Laxity First (LLF) algorithm [6] assigns higher priority to a task with the least laxity. The *laxity* of a real-time task T_i at time t, $L_i(t)$, is defined as in $L_i(t) = D_i(t) - E_i(t)$ where $D_i(t)$ is the deadline by which the task must be completed and $E_i(t)$ is the amount of computation remaining to be performed. In other words, laxity is a measure of the flexibility available for scheduling a task. A laxity of $L_i(t)$ means if the task T_i is delayed at most by $L_i(t)$ time units, it will still meet its deadline.

A task with zero laxity must be scheduled right away and executed without preemption or it will fail to meet its deadline. The negative laxity indicates that the task will miss the deadline, no matter when it is picked up for execution.

The Rate Monotonic (RM) algorithm [4] is a fixed priority scheduling algorithm which assigns the highest priority to the task with highest frequency (smallest interval) in the system, and lowest priority to the task

0-7695-3068-0/07 $25.00 © 2007 IEEE
DOI 10.1109/ICIT.2007.56

having lowest frequency. At any time, the scheduler chooses to execute the task with the highest priority. By specifying the period and computational time required by the task, the behavior of the system can be categorized *a priori*.

3. Background and definitions

Two important parameters affecting the performance of scheduling algorithms on parallel machines are *preemption* and *migration*. A job executing on a processor can be interrupted at any time, and its execution resumed later on the same or a different processor. If an interrupted task resumes execution on the same processor a preemption has occurred, and if its execution resumes on a different processor a migration has happened.

Based on the two aforementioned factors, there are two types of scheduling policies in multiprocessor environments named *global* scheduling and *partition* scheduling. Global scheduling algorithms put all the arrived tasks with non-zero remaining execution time into a queue that is common among the processing nodes. In a system with *m* processors, in every moment, *m* tasks having the highest priorities should be executing considering preemptions and migrations, if necessary. Partition scheduling algorithms divide the task set into partitions (subsets) such that all the tasks within a partition are assigned to a processor. In this policy task migrations are not allowed.

In [7] Baruah et al. came to the conclusion that despite its non-optimality, EDF is an appropriate algorithm to use for online scheduling on uniform multiprocessors. However, their implementation suffers from a rather large number of migrations due to vast priority fluctuations caused by terminating or arrival of jobs with relatively nearer deadlines. Task migration cost might be very high. For example, in loosely coupled systems such as a cluster of workstations a migration is performed so slowly that the overhead resulting from excessive migrations may prove unacceptable [6]. Another disadvantage of EDF is that its behavior becomes unpredictable in overloaded situations. That is, there is no guarantee on which jobs will meet their deadline. Therefore, the performance of EDF drops in overloaded conditions such that it can not be considered for use.

4. Fuzzy systems

A Fuzzy Inference System (FIS) tries to derive answers from a knowledgebase by using a fuzzy inference engine. It consists of an *input* stage, a *processing* stage, and an *output* stage.

The processing stage, which is called the inference engine, is based on a collection of logic rules in the form of *IF-THEN* statements, where the *IF* part is called the *antecedent* and the *THEN* part is called the *consequent*. A typical FIS has dozens of rules. These rules are stored in a knowledgebase. An example of fuzzy IF-THEN rules is "IF *deadline* IS *near* THEN *priority* IS *high*" in which *deadline* and *priority* are linguistics variables and *near* and *high* are linguistics terms.

Sugeno's fuzzy inference method has three advantages. Firstly, it is computationally efficient, which is an essential benefit to real-time systems. Secondly, it works well with optimization and adaptive techniques. The third advantage of Sugeno type inference is that it is well-suited to mathematical analysis [3]. Owing to the aforementioned benefits of Sugeno method, it has been applied in our proposed approach.

5. Real-time system model

This research is concentrated on uniform parallel machines in *soft* real-time environments. The algorithms being investigated are *on-line* and use up-to-date information for the scheduling activities during the systems execution.

We have focused on *periodic* tasks and each task's deadline is equal to its period. The reason for this choice is that it has been proved that a periodic task model is useful for modeling and analysis of majority of real-time systems [11]. Moreover, load factor measurement is easier and more accurate for periodic tasks. All tasks are *synchronous* i.e. their first request arrive simultaneously at the time zero. Such systems are common and applicable [12].

Tasks are *preemptable* and in each scheduling event a dispatcher decides which task to be performed next. In addition, a task is not allowed to run concurrently (on more than one processor at a time). Tasks must declare their characteristics and requirements such as interval, deadline and Worst Case Execution Time (WCET) at their arrival. The intervals and execution times are correct multiples of one time slice. The actual execution time of each task is equal to its WCET. Tasks are not removed from the local memory or cache before they migrate to another node.

Scheduling algorithms must prevent simultaneous access to resources and shared devices. We assume the tasks are independent and do not need to do I/O operations. Therefore, the concurrency control matters have not been considered.

6. Proposed approach

Our proposed algorithm is based on our previous contribution called Highest Fuzzy Priority First (HFPF)

Algorithm [14]. Although HFPF has supremacy over EDF in some aspects, it imposes more overhead to the system in terms of migrations than EDF does. Besides, it suffers from unpredictability, resulting in a relatively poor schedule. Therefore, we have applied some modifications to our method to decrease the number of task migrations and also to add predictability to its behavior.

In order to decrease the number of migrations we prevent a job from moving to another processor if it is among the *m* higher priority jobs. Therefore, a job will continue its execution on the same processor if possible. This concept is known as *processor affinity*. By scheduling tasks on the processor whose local memory or cache already contains the necessary data, we can significantly reduce the execution time and thus overhead of the system.

In order to give the scheduler a more predictable behavior we first perform a *feasibility check* to see whether a job has a chance to meet its deadline. If so, the job is allowed to get executed. Having known the deadline of a task and its remaining execution time it is possible to verify whether it has the opportunity to meet its deadline. More precisely, this verification can be done by measuring the tasks' laxity.

The block diagram of our fuzzy system is presented in Fig. 1. In the proposed model, the input stage consists of three linguistic variables, namely deadline, laxity and interval. The values of these input variables are obtained from the corresponding characteristics of tasks. Since the mentioned characteristics may vary a lot from task to task, they must be normalized. Therefore, we normalize the deadline, laxity and interval to numbers between 0 and 1.

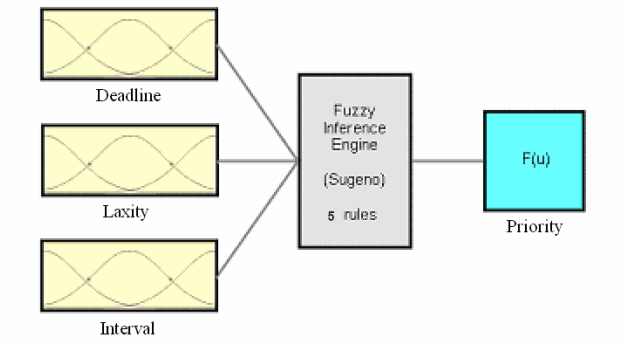

Figure 1. Inference system block diagram

Our proposed system has 5 rules which are shown in Fig. 2. In an FIS, the number of rules has a direct effect on the time complexity of the inference process. Therefore, having fewer rules may result in a better system performance.

1. IF *deadline* IS *near* THEN *priority* IS *very high*
2. IF *laxity* IS *near* THEN *priority* IS *urgent*
3. IF *interval* IS *small* THEN *priority* IS *high*
4. IF *deadline* IS *far* THEN *priority* IS *low*
5. IF *laxity* IS *far* THEN *priority* IS *very low*

Figure 2. Fuzzy rule-base

We use the aforementioned FIS to calculate the dynamic priority of tasks. Consequently, the following algorithm is performed at every scheduling event:

6.1. Highest Fuzzy Priority First (HFPF) Algorithm

Let *m* denote the number of processing nodes and *n*, $(n \geq m)$ denote the number of available tasks in a uniform parallel real-time system. Let $s_1, s_2, \ldots, s_m$ denote the computing capacity of available processing nodes indexed in a non-increasing manner: $s_j \geq s_{j+1}$ for all j, $1 < j < m$. We assume that all speeds are positive – i.e., $s_j > 0$ for all j.

In this section we present the six steps of the HFPF algorithm. Obviously, each task which is picked up for execution is not considered for execution by the other processors.

1. For each task T_i feed its corresponding deadline, laxity, and interval to fuzzy inference engine. Then consider the output as the priority of task T_i.
2. Perform a feasibility check to specify the tasks which have a chance to meet their deadline and put them in set A. Put the remaining tasks in set B.
3. Sort both task sets A and B according to their priorities in a non-ascending order.

Let *k* denote the number of tasks in set *A* – i.e. the number of tasks that have the opportunity to meet their deadline.

4. For all processor j, $(j \leq \min(k,m))$ check whether a task which was last running on the j^{th} processor is among the first min(k, m) tasks of set A. If so assign it to the j^{th} processor.

At this point there might be some processors to which no task has been assigned yet.

5. For all j, $(j \leq \min(k,m))$ if no task is assigned to the j^{th} processor, select the task with highest priority from remaining tasks of A and assign it to the j^{th} processor.

If $k \geq m$, each processor have a task to process and the algorithm is finished.

6. If k < m, for all j, $(k < j \leq m)$ assign the task with highest priority from B to the j^{th} processor.

The sixth step is optional and all of the tasks from *B* will miss their deadlines.

7. Performance evaluation

In this section, we study the performance of our proposed algorithm (HFPF) based on simulation and compare it with that of global EDF algorithm. Load factor is considered as main parameter and its influence on the performance metrics below as dependent variables is presented [21]:

- success ratio
- response time
- preemptions and migrations
- CPU utilization
- load balance

In the experiments that we present later, in order to minimize the influence of exceptional states, each experiment was repeated 100 times and the results were averaged out. The simulation time is equal to a *meta period* which is equivalent to the smallest common multiple of all tasks' periods. It should be considered that presented results are in fact the average of the obtained values from all processors.

Load factor of task T_i is defined as the ratio of its WCET (E_i) to its request period (P_i). For n periodic tasks, load factor is equal to:

$$L = \sum_{i=1}^{n} \frac{E_i}{P_i}$$

In multiprocessor environments, the overall load factor is the sum of all processors' load factor.

7.1. Success ratio

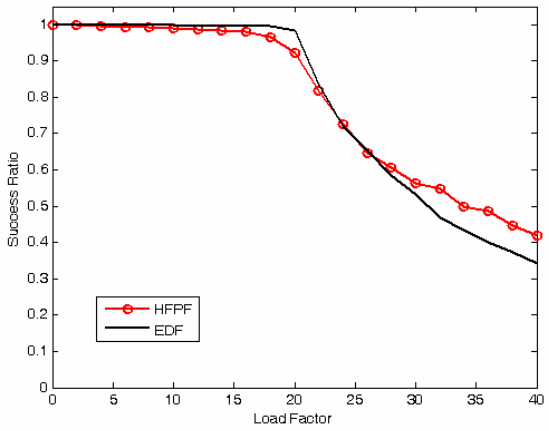

Figure 3. Success ratio

Success ratio is defined as the ratio of the jobs that have been successfully completed to the jobs that arrived to the system [13]. As illustrated in Fig. 3, both algorithms show a near optimal performance in non-overloaded situations. Nonetheless, in overloaded conditions, the performance of the both methods descends. Part of this performance drop is due to the fact that the system does not have the capacity to meet al deadlines. However, HFPF tries to fully utilize the computing capacity of available processing nodes, and shows a better performance.

7.2. Response time

Response time is defined as the time between arriving a request and completion of its processing. Obviously it is not influenced by the tasks which fail to meet their deadlines. A point to be considered is that due to presence of the tasks with different periods, the absolute numeric values are meaningless. Therefore, we use *response ratio* instead of response time and define it as the ratio of a task's response time to its period.

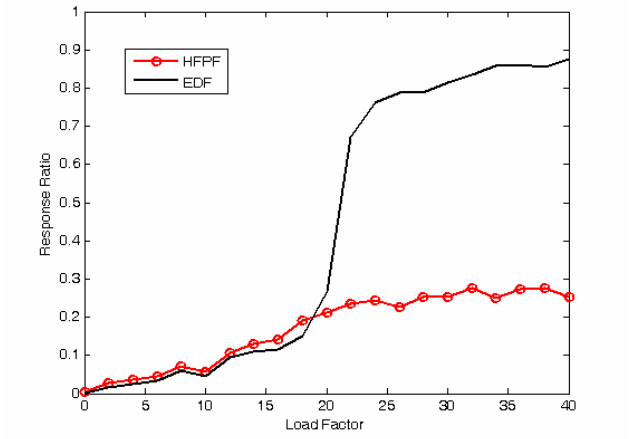

Figure 4. Response time

Fig. 4 depicts the observed response ratio. The two algorithms in non-overloaded conditions have close performances. In overloaded conditions, however, HFPF algorithm shows a better performance. The diagram suggests that in overloaded conditions tasks are completed relatively sooner when using HFPF.

7.3. Preemptions and migrations

One of the most significant factors influencing scheduling overload is the number of produced preemptions and migrations, and our aim here is to measure their values for each of the compared algorithms. In this case, due to different number of tasks in diverse conditions, applying absolute numeral values is meaningless. As a result, we use *preemption ratio* and *migration ratio*, instead of preemption and migration, and define them as the ratio of the total number of preemptions to the total number of arrived requests that have the chance to be executed, and the ratio of the total number of migrations to the total number of arrived requests that are picked up for execution, respectively.

Fig. 5 illustrates the produced (inter-processor) preemption ratio indicating that the supremacy is with EDF algorithm. A preemption ratio of 1.0 means that on average jobs are preempted once per release. Since preemption ratio for both algorithms is less than 1.0 it is acceptable and the two algorithms incur a very trivial overhead in term of preemptions.

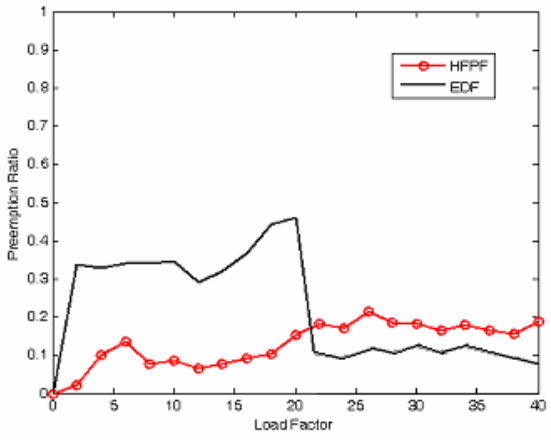

Figure 5. Preemption ratio

In Fig. 6, the produced migration ratio has been depicted. Unlike the previous diagram, HFPF algorithm produces much less migrations than EDF does. The migration ratio of our proposed algorithm is near 1.0 meaning that on average a typical task migrates once per release. Whereas the migration ratio of EDF is several times as much as that of HFPF.

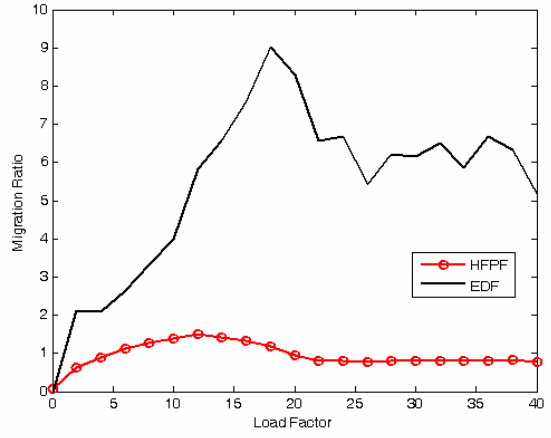

Figure 6. Migration ratio

The cost of each migration is several times as much as the cost of preemption. This is due to the fact that all of the necessary data for preemption exists in the local memory or perhaps in the cache of the same processor. While a migration means reading the required data from another processor's local memory or the shared memory. The above diagram shows that our proposed algorithm imposes much less overhead on the system.

7.4. CPU utilization

CPU utilization is the percentage of CPU time in which, the CPU has not been idle with respect to the time passed. Therefore, it does not include the times in which CPU has had idle processing or has been processing the jobs which have ultimately been missed.

In Fig. 7 the CPU utilization of the two algorithms has been illustrated. Both algorithms have approximately the same performance in low load factor conditions and use the maximum possible CPU resources. However, in overloaded conditions the HFPF algorithm almost fully utilizes the CPUs. This considerable improvement is due to performing feasibility check.

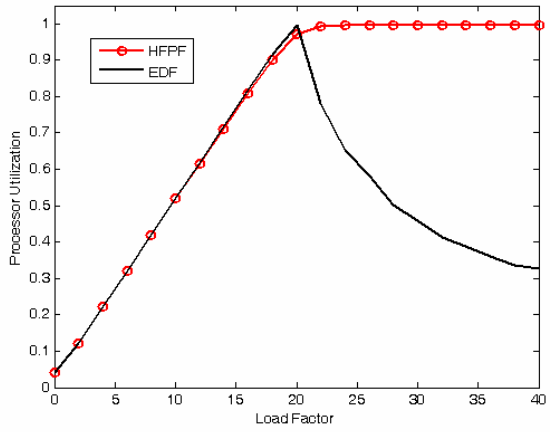

Figure 7. CPU utilization

7.5. Load balance

Load balance means steady distribution of load among processors in such a way that minimizes the load difference. Regular load balance among processors not only decreases the response time, but also increases system's reliability which is very significant in real-time systems. Another advantage of a balanced system is the minimized total power consumption. The length of schedule in balanced case is also minimized. We apply the formula below for defining the system's load balance [14]:

$$1-\frac{\sum_{j=1}^{m}\left|\overline{U}-U_j\right|}{m\times\overline{U}}$$

in which m is the number of processors. $\overline{U}$ denotes the average CPU utilization and U_j represents the j^{th} processor's utilization. Fig. 8 illustrates the load balance for both algorithms. Apparently, HFPF algorithm results in a balanced schedule in overloaded conditions.

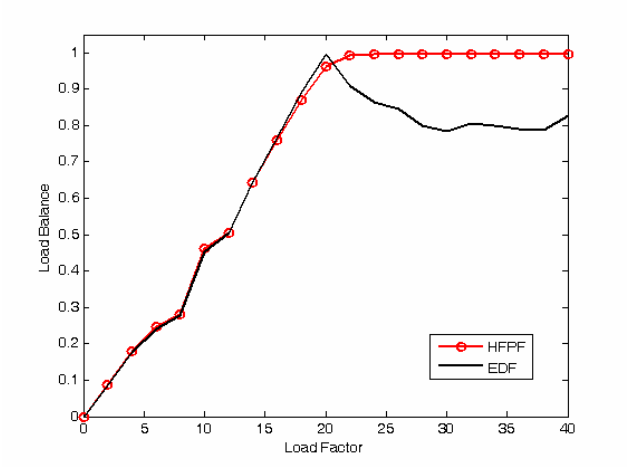

Figure 8. Load balance

8. Conclusion

In this paper a new fuzzy-based algorithm, called HFPF, for scheduling real-time tasks on uniform parallel machines is presented. The performance of this algorithm is then compared with that of EDF algorithm. It is shown than our proposed approach not only demonstrates a performance close to that of EDF in non-overloaded conditions but also it has supremacy over EDF in overloaded situations in many aspects. We show that traditional EDF algorithm which ignores the location of tasks when assigning them to processors, incurs a significant performance penalty on the system. Since HFPF imposes much less overhead in terms of migrations on the system, it could be more appropriate for use on parallel machines in which the cost of migrations is relatively high.

9. References

[1] J. Goossens, and P. Richard, Overview of real-time scheduling problems, *ninth international workshop on project management and Scheduling*, Nancy, France, April 2004, pp. 13-22.

[2] K. Mok, and M. L. Dertouzos, Multiprocessor scheduling in a hard real-time environment, *the 7th IEEE Texas Conference on Computing Systems*, November 1978, pp. 5-12.

[3] M. Sabeghi, M. Naghibzadeh, T. Taghavi, Scheduling Non-Preemptive Periodic Tasks in Soft Real-Time Systems Using Fuzzy Inference, *the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, April 2006.

[4] L. Liu, and J. W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, JACM, Vol. 20, No. 1, Jan. 1973, pp. 46-61.

[5] M. L. Dertouzos, Control robotics: the procedural control of physical processes, *IFIP Congress*, Stockholm, Sweden, August 5-10, 1974, pp. 807-813.

[6] A. Mok. "Fundamental Design Problems of Distributed Systems for Hard Real-time Environments", *PhD thesis*, Massachusetts Institute of Technology, Cambridge, MA, 1983.

[7] S. Baruah, S. Funk, and J. Goossens, Robustness Results Concerning EDF Scheduling upon Uniform Multiprocessors, IEEE Trans. Computers, VOL. 52, NO. 9, Sep 2003, pp. 1185-1195.

[8] L. Wang, A course in fuzzy systems and control, Prentice Hall, August 1996.

[9] E.H. Mamdani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, International Journal of Man-Machine Studies, Vol. 7, No. 1, pp1-13, 1975.

[10] M. Sugeno, Industrial applications of fuzzy control, Elsevier Science Inc., New York, NY, 1985.

[11] J. Goossens, S. Baruah, and S. Funk, Real-time scheduling on multiprocessor, *the 10th International Conference on Real-Time System*, 2002.

[12] J. Goossens, S. Funk, and S. Baruah, EDF scheduling on multiprocessor platforms: some (perhaps) counterintuitive observations, the International Conference on Real-Time Computing Systems and Applications (RTCSA 2002), Tokyo, Japan, March 18-20, 2002, pp. 321-329.

[13] B. Andersson, and J. Jonsson, Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition, *7th International Conference on Real-Time Computing Systems and Applications (RTCSA'2000)*, Cheju Island, South Korea, December 12-14, 2000, pp. 337-346.

[14] V. Salmani, M. Naghibzadeh, M. Kahani, and S. K. Nejad, "Multi-criteria Scheduling of Soft Real-time Tasks on Uniform Multiprocessors Using Fuzzy Inference," in *Proc. International Conference on Systems, Computing Sciences & Software Engineering (SCS^2 06)*, University of Bridgeport, December 4 – 14, 2006.