M. Naeem Shehzad[1], A.M. Déplanche[2], Yvon Trinquet[2], Umer Farooq[1]

CIIT Lahore, Pakistan (1), LUNAM Université, Université de Nantes, IRCCyN UMR CNRS, 6597, Nantes, France (2)

Efficient data generation for the testing of real-time multiprocessor scheduling algorithms.

Abstract. *In conducting the performance evaluation tests of real-time multiprocessor scheduling algorithms, synthetic input data is mandatory for obtaining reliable results and to draw strong conclusions. The results of the statistical evaluation highly depend on the data chosen for the experimentation. The data generation process is required to be efficient while the resultant data should comply with the user requirements. This article discusses two established data generation techniques used in the literature, explains their advantages and disadvantages, and finally proposes few extensions in one of the techniques, which is considered quite efficient, to be made compatible with discrete time simulator.*

Streszczenie. *W artykule przedyskutowano dwie techniki generacji danych w multiprocesorze czasu rzeczywistego. Zaproponowano modyfikacje algorytmu Stafforda.**(Efektywny proces generacji danych w multiprocesorze czasu rzeczywistego)***

Keywords: Simulation data, Multiprocessor scheduling, Performance evaluation.
Słowa kluczowe: kolejkowanie w układzie multiprocesora, algorytm Stafforda.

doi:10.12915/pe.2014.09.36

Introduction

Real-time embedded systems have become very significant in the recent times and are seen everywhere from mobile phones, computer tablets up to the automobiles and space systems. Multiprocessors in real-time embedded systems emerged to fulfill the demand of enhanced processing power. Another topic that is directly linked to multiprocessing is multiprocessor scheduling. The multiprocessor scheduling got researcher's attention primarily with the advent of multi-core architecture. This is true both for symmetric [1] and heterogeneous processor [2]. Performance evaluation of scheduling algorithms is an essential part of algorithm development which helps them to analyze the characteristic of algorithms.

In symmetric multiprocessor environment, there is a constant growth towards proposition of new scheduling algorithms with some new features. Although a number of optimal multiprocessor scheduling algorithms have been already proposed but they are not considered practical due to a large amount of overhead produced. To show the strength of new proposed algorithms over the older ones, testing of scheduling algorithms have been and continue to be a useful tool for determining the capacity of an algorithm to solve the given problems. There are two methods to determine the performance of a scheduling algorithms: Either to prove it mathematically or to show it statistically with the help of simulation study. In the first evaluation method, it is difficult to compute various performance parameters and this is achievable only under simple assumptions. To study the performance evaluation by statistical method requires testing of the scheduling algorithm over a significant amount of data. The results help to analyze the behavior and efficiency of scheduling algorithms, their strengths and weaknesses. The output also helps to do a comparison with other scheduling algorithms and suggests some improvements if possible. The data generation process plays a key role in the evaluation by statistical method.

A typical data generator meant to produce data for evaluating a multiprocessor scheduling algorithm starts with prior basic knowledge of hardware and software of the desired model. The hardware information includes the number of processors while the software configuration includes the number of tasks, their total utilization factor and the scheduling policy. The output of a data generator is a file that includes the precise definition of task parameters in accordance with input data. The generated data sets should have the following characteristics:

- All the task parameters should be independent of each other up to acquired possible level;
- The data generator must provide user with a complete control of parameters so that they can be fixed or varied according to the testing requirements;
- The generated data should be unbiased and should properly cover all the critical areas;
- The process of data generation should be efficient so that it requires a little time to generate a huge amount of data.

The use of data lacking the above mentioned characteristics may lead to a wrong conclusion. For example, a generator which produces a data which has a tendency to miss some critical areas may result in unrealistic conclusions about the algorithms that may be weak or strong in those areas.

In literature, various data generators have been proposed, each one with its own properties. They can be categorized as: one used for the evaluation of mono-processor scheduling algorithms and the other used in multi-processor environment for the same purpose. The data generation for multi-processor environment is comparatively complex because of the fact that a single dimensional problem is converted into a multi-dimensional problem. Similar to the uniprocessor environment, the output of data generator in multi-processor is a set of utilization factors of the specified number of tasks with a given accumulative value of their utilization factors. However, instead of taskset utilization factor, most of the real-time simulators require discrete values of task period and worst case execution time. The utilization factors generated by generators give rise to non-discrete values of task parameters. Theoretically, the non-discrete values of parameters do not create any problem. However, the theoretical conclusions drawn on the basis of such ideal data do not necessarily remain valid in a practical situation. The motivation behind this article is to modify an existing efficient data generators for multiprocessor environment according to the requirement of real-time simulator without compromising the original properties of the data generator. The proposed modifications in a data generator make it compatible with practical simulators.

The next section presents the system model to be analyzed.

In the very next section, an overview of main data generation tools is given. In the section after that, some modifications in the existing generation tools are suggested. The characteristics of resultant generator are given in the following section along with conclusion in last section.

System Model

We study a data generator used for evaluating a scheduling algorithm model. The scheduling algorithm model consider τ, a set N independent, synchronous and periodic tasks with implicit deadlines to be scheduled on a set of M identical processors. Each task T_i should be characterized by a worst case execution time $T_i.e$ and a period $T_i.p$. The utilization factor of T_i is defined as $T_i.u = T_i.e/T_i.p$. The sum of total utilization factor of tasks in the dataset should be equal to the given input U. The necessary and sufficient feasibility[1] conditions for τ are assumed to be met, i.e. $U = \sum_{i=1}^{N}(T_i.u) \leq M$ and $max_{i=1}^{N}(T_i.u) \leq 1$.

The data generator model, used for the evaluation of above discussed scheduling algorithm, takes U and N as input and gives N couples of $T_i.u$ according to the system load conditions.

Related work

The most known results in generation of taskset for a multiprocessor scheduling problem are *UUnifast-Discard* algorithm [3] and the *Randfixedsum algorithm* [4, 5]. The UUnifast-Discard is an extension of UUnifast which is a data generator for uniprocessor scheduling.

UUniFast: In the case of uniprocessor scheduling, an efficient data generator *UUniFast* was proposed by Bini and Buttazo [6] in 2005. It generates the taskset configurations with a fixed number of tasks (N) and specified value of sum of their individual utilization factor (U). *UUniFast* produces independent tasks with randomly generated unbiased utilization factors. In this process, the utilization factor of each task is sampled and then only the tasks with correct total utilization are kept in the configurations. The work of Bini and Buttazo was motivated by a previous work of Lehoczky et al. [7] where biased results were produced because of concentration of utilization factor of tasks in the center of utilization factor plane (starts at 0 and ends at 1) and consequently some biased results were produced for a specific algorithm which was less efficient in that region. Other uniprocessor dataset generation algorithms like UScaling, UFitting and UUniform all suffer from problem of biased data generations and are not discussed here [6].

The use of *UUniFast* algorithm is not very successful in case of multiprocessor scheduling. This is because there is a possibility of generation of task with utilization factor more than 1 which is an invalid value. *UUniFast* is usually used in multiprocessor scheduling after following two modifications:

- For a M processor system, an initial taskset of $(M+1)$ tasks are generated and then individual tasks with randomly generated utilization factors are added to the taskset unless the specified value of total utilization factor is achieved. The disadvantage of this type of modification is that it can result in unbiased distribution of utilization factor because of involvement of number of taskset length in addition to the task utilization factor. [8], [1] and [9] used this technique.
- The second type of modification was proposed by Davis and Burn [3] which they name as *UUnifast-Discard*. According to this proposition, the invalid taskset are discarded. The invalid tasksets are the one which have at least one task with utilization factor more than 1. The disadvantage of this type of generation is that there are certain values of *U* and *N* where the ratio of discarded taskset is too much. For example, when $N \approx 2U$, it starts becoming inefficient. The phenomenon becomes more visible with large values of *N*.

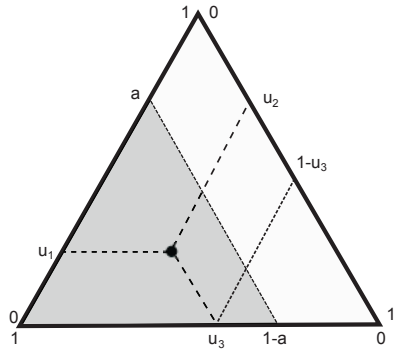

Fig. 1. Idea behind the Stafford's algorithm.

Stafford's RandomFixed Sum The *Stafford's Random FixedSum* algorithm [4, 5] efficiently generates dataset without producing invalid tasksets. Hence there is no rejection of generated tasksets. It generates a set of vectors which are uniformly distributed in *(N-1)* dimensional space with a constant given sum. The vectors represent the utilization factors of the tasks and are limited between 0 and 1[2]. The operation is mentioned with the help of figure 1 [5]. It explains the generation of a taskset with 3 tasks and their sum equal to 1 i.e. *N=3* and *U=1*. Each side of triangle has a unit length which represents the utilization factor. A taskset with three tasks can be represented somewhere inside the triangle.

According to Stafford, if the points are uniformly distributed in the triangle, then the number of points inside a region are proportional to the area of that region. To illustrate the idea, a small triangle can be created by drawing a line between *(a,0,1-a)* and *(0,a,1-a)* as shown in figure 1. The area of small triangle is $\frac{\sqrt{3}}{4}a^2$ and that of large triangle is $\frac{\sqrt{3}}{4}$. Hence the probability of a point to lies inside small triangle is a^2. This is similar to $P(u_3 > (1-a)) = P((1-u_3) < a)$ where P represents the probability. The data generation process starts by taking a random number r_2 and setting $u_3 = 1 - r_2^{\frac{1}{2}}$ similar to that of UUniFast. The u_2 is chosen any where between 0 and $(1-u_3)$ as is done in *UUniFast* $u_2 = (1-u_3) - r_1$ where r_1 is a random uniform value. The sum of three values come out to be 1.

The stafford's algorithm breaks down valid area into (*N-1*) dimensional simplexes and applies an algorithm similar to that of *UUniFast* within randomly chosen simplex. The Matlab implementation of algorithm is freely available [4].

Although Stafford's algorithm efficiently generates task utilization factors for all the tasks in a taskset but, as discussed earlier, the scheduling simulators are based on discrete time require integer values of all the task parameters rather than the task utilization factor. This requirement is due to the hardware characteristics of processors which states that execution time of a task should be an integral multiple of the highest precision time unit in the system and the

[1] A taskset is feasible if there exists an algorithm that can schedule all the jobs generated by the taskset without missing deadlines.

[2] The limits *a* and *b* are general in Stafford's algorithm but are fixed to 0 and 1 because these are extreme values of a task utilization factor

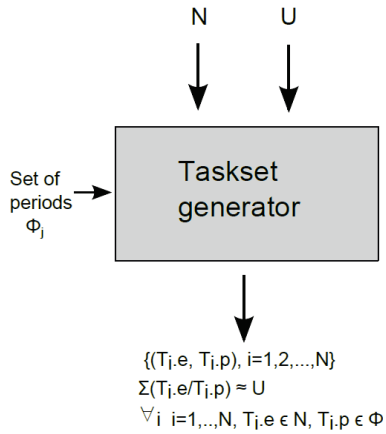

Fig. 2. Block diagram of SRFS modified.

decisions are made at discrete times. In this article, we have tried to take in account this practical constraint of scheduler and to modify the output of the existing Stafford's algorithm in order to make it compatible with the needs of simulator used for the analysis of multiprocessor scheduling algorithms. The article suggests some modifications which fulfills the requirements without compromising the key characteristics of Stafford's algorithm. The article also highlights some characteristics of *Stafford's* algorithm useful for finding new aspects of scheduling algorithms.

SRFS Modified

When we introduce some modifications in Stafford's RandomFixed Sum algorithm, we call the resulting algorithm as Stafford's RandomFixed Sum Modified or *SRFS Modified*. The modification assures the production of integer values of task period and worst case execution time from the utilization factor generated by Stafford's RandomFixed Sum algorithm. Figure 2 shows the block diagram while Figure 3 shows the functional diagram of SRFS Modified taskset generator. Figure 3 shows that Roger Stafford's randfixedsum algorithm is used at the heart of the taskset generator. It takes total utilization factor *U*, the number of tasks *N* in each taskset and a set of task periods ϕ_j as input while it gives *N* couples of $(T_i.e, T_i.p)$ such that $\sum_{i=1}^{N}(\frac{T_i.e}{T_i.p})$ is approximately equal to U. In case of acquired sum less than U, it should be as close as possible to U. For example, the upper bound of this difference can be set as 0.1 % of the U.

Computation of task period: For each task, a value of period is selected and then task's worst case execution time is computed using task period and task utilization factor. There are two methods to set the values of task period.

- An integer value of task period $T_i.p$ can be randomly chosen with in a specific limit. However this technique can result in very huge values of *hyper period* which specifies the duration of simulation. For example, if there is a taskset comprising 20 tasks with 100 as an upper limit of task period. If three or four time periods are prime numbers then the resulting *hyper period* can be very large which is not feasible. In addition, the scheduling intervals[3] are defined by the values of task period. In random selection of task period, if some small values are selected, then scheduling intervals can be very small. For example, if there is a task with time period 2, all the scheduling intervals will either be 1 or 2.
- In the second technique, the time periods $T_i.p$ are chosen from a set of pre-defined values of time periods in any specific order. Round robin way is one of the examples. Using fix values of task periods helps to limit the hyper period of the taskset and thus bounds reasonably the simulation duration. Moreover, the utilization of different sets of pre-defined task period give rise to various distributions of scheduling intervals and frequency. For example, ϕ_j = {30, 35, 40, 50, 100} is a set of specific task period with a *hyper period* = 4200. The length of scheduling intervals found within the hyper period and their frequency are given in Table 1. The set produces intervals with 6 different lengths with relatively higher proportion of smaller lengths.

[3]For the sake of convenience, the scheduling time is divided into small time intervals which are easy to handle. These are defined by the period of the tasks in the taskset

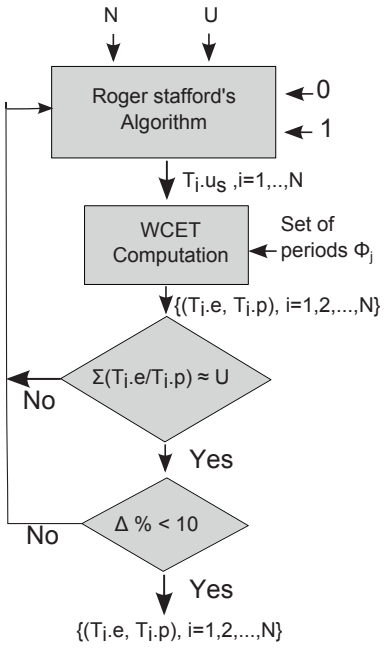

Fig. 3. Flow diagram of SRFS modified.

Algorithm 1 WCET computation

Suppose
$T_i.u_s$ = Utilization factor of task T_i produced by Stafford's algorithm
$T_i.u$ = Utilization factor of task T_i after modification
ϕ_j= {ϕ_{j0}, ϕ_{j1},... ϕ_{jk}}
$\Delta\%$ = Percentage error

1. $\Delta_0 = 0$;
2. $\Delta\% = 0$
3. ***for*** $(i = 1...N)$
4. $\quad T_i.u' = min\{(T_i.u_s + \Delta_{i-1}), 1\}$
5. $\quad T_i.p = \phi_{j(i\,\%\,k+1)}$
6. $\quad T_i.e = max\{\lfloor T_i.p \times T_i.u' \rfloor, 1\}$
7. $\quad \Delta_i = T_i.u_s - \frac{T_i.e}{T_i.p}$
8. $\quad \Delta\% = \Delta\% + \frac{|\Delta_i|}{T_i.u_s}$
9. ***end for***
10. $\Delta\% = \frac{\Delta\%}{N}$

Computation of WCET: The value of $T_i.e$ is calculated by taking a time period $T_i.p$ value from period set using round robin way and using $T_i.e$ = $T_i.u_s$ x $T_i.p$. However, the obtained value of $T_i.e$ may not be an integer value. Therefore, an algorithm is used to convert it into the closest lower in-

teger value while taking into account the possible rounding off error in the next steps. The algorithm 1 explains it. The taskset is discarded if the total utilization factor finally calculated is more than the required *U*. Due to restricting $T_i.e$ to an integer, a difference or error is produced between the utilization factor obtained by Stafford's algorithm $T_i.u_s$ and final utilization factor obtained $T_i.u$. The taskset is not valid if this average error is greater than 10 %. This value is a good compromise between the initial Stafford's distribution and a moderate time to generate a large number of configurations.

Length of interval	Frequency of interval	%
5	72	21.42
10	144	42.85
15	36	10.71
20	60	17.86
25	12	3.57
30	12	3.57
	$\sum$ = 336	100

Table 1. Scheduling intervals of ϕ_j.

Properties of SRFS Modified

It is mentioned in the previous section that due to digitizing all the task parameters to the integer values, a rounding off error is produced in the output of classical Stafford's algorithm. The digitizing error should not disturb the unbiased nature of the data produced in SRFS-Modified. In our case, this is done by discarding the taskset with a percentage error of more than 10 %. This value of percentage error may be limited to any desired value like 5 % etc.

The bar graphs in the figures 4, 5 and 6 show the average number of tasks produced by *SRFS Modified* over a range of utilization factors with variable values of U and N and digitizing error of 10 %. The vertical axis shows the frequency of the tasks while the horizontal axis represents the task utilization factor. Each value in the graph represents an average of 30 taskset generations. The number of tasks in each taskset were 100.

At *U/N = 0.5,* there is a uniform distribution of tasks across different values of task utilization i.e. the task utilization factors lie uniformly between 0 and 1. At *U/N = 0.25,* the number of tasks with relatively lower utilization factors dominate but still there exist some tasks with high value of utilization factors. At *U/N = 0.75*, there is an opposite trend as that of *U/N = 0.25* and tasks with relatively higher utilization factor dominate while the tasks with lower utilization factor are present in minority.

In *SRFS Modified*, limits of utilization factor can be varied as per requirement similar to that of original algorithm. For example, utilization factor of the tasks in the taskset can be limited to a range between 0 and 0.5 or between 0.25 and 0.75 etc.

Conclusion

This paper presents some modifications and extensions in Stafford's randfixedsum algorithm which is considered as very efficient data generation tool. The purpose of the modifications is to generate discrete values of task's parameters to make it compatible for the use practical discrete time simulator. The modification does not compromise the originality of Stafford's randfixedsum algorithm. The generator provides user with a variety of generating options i.e. variable total load, task set with controlled individual utilization factors and specified simulation duration indicated by set of user defined task period.

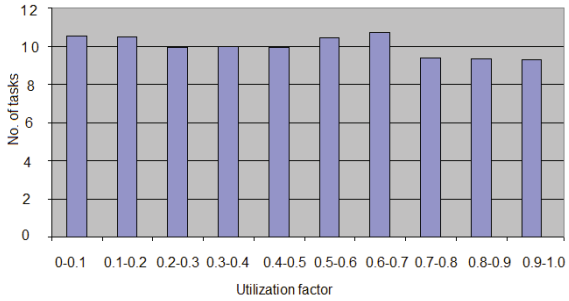

Fig. 4. Distribution of utilization factor for U/N = 0.5.

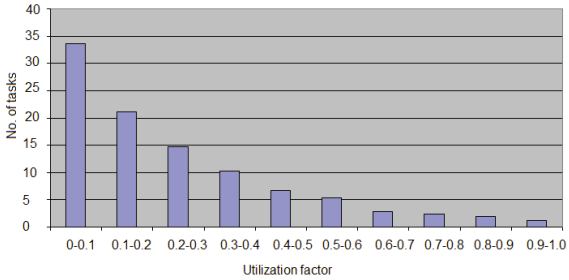

Fig. 5. Distribution of utilization factor for U/N = 0.25.

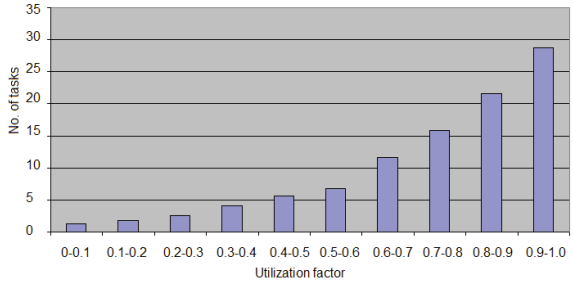

Fig. 6. Distribution of utilization factor for U/N = 0.75.

REFERENCES

[1] M. Bertogna, M. Cirinei, Response-time analysis for globally scheduled symmetric multiprocessor platforms, In the Proceedings of the 28th IEEE International Real-Time Systems Symposium (2007) pp. 149–160.

[2] Z. Handzel, Implementing the rate monotonic scheduling algorithm for heterogeneous processors, Przeglad Elektrotechniczny (2012) pp.143–145.

[3] R. I. Davis, A. Burns, Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, In the Proceedings of the 30th IEEE Real-Time Systems Symposium RTSS 2009 (December 2009) pp. 398–409.

[4] R. Stafford, Random vectors with fixed sum., http://www.mathworks.com/matlabcentral/fileexchange/9700.

[5] P. Emberson, R. Stafford, R. I. Davis, Techniques for the synthesis of multiprocessor tasksets, 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) (2010) pp. 6–11.

[6] E. Bini, G. Buttazzo, Measuring the performance of schedulability tests, Real-Time Systems 30 (May 2005) pp. 129–154.

[7] J. P. Lehoczky, L. Sha, Y. Ding, The rate monotonic scheduling algorithm: Exact characterization and average case behavior, in: RTSS, 1989, pp. 166–171.

[8] M. Bertogna, Real-time scheduling for multiprocessor platforms, Ph.D. thesis, PhD. dissertation, Scuola Superiore SantAnna (Pisa 2007).

[9] M. Bertogna, M. Cirinei, G. Lipari, Schedulability analysis of global scheduling algorithms on multiprocessor platforms, IEEE Trans. Parallel Distrib. Syst. 20 (4) (2009) pp. 553–566.

Corresponding Author

Dr. M. Naeem Shehzad, Asst. Prof. CIIT Lahore, Pakistan
email: naeem.shehzad@ciitlahore.edu.pk