STREAM: A Simulation Tool for Energy Efficient Real Time Scheduling and Analysis

Mayuri Digalwar Pravin Gahukar Sudeept Mohan Biju K. Raveendran

Department of Computer Science and Information Systems
Birla Institute of Technology and Science, Pilani, India

Abstract: This paper proposes an automation tool - STREAM, for simulation, testing and analysis of energy aware real time scheduling algorithms for periodic as well as mixed task sets on multi-core processor. The key features of STREAM include implementation of scheduling algorithms, synthetic task set generation, modules for doing performance analysis and generation of execution traces. The simulator design is made simple, understandable and flexible such that addition of new algorithms or modification to existing algorithms can be done with minimum efforts. Testing of modules is carried out on randomly generated task sets. STREAM facilitates the comparative performance analysis of different scheduling algorithms on uniprocessor, multiprocessor or multi-core processor platforms. A graph plotter is provided to visualize the performance analysis of different scheduling algorithms.

Keywords: real time scheduling, simulation tool, dynamic voltage and frequency scaling, mixed task set.

I. INTRODUCTION

Real time systems are deterministic by nature where the real time tasks need to meet hard deadlines. The real time scheduling algorithms play major role for maintaining the timing constraints of the tasks. Therefore, correctness of these scheduling algorithms should be tested before deploying them on the real system. This validation can be carried out either theoretically, experimentally or using simulation. Simulation is one of the methods eventually used in real time systems community for validation of real time scheduling algorithms when formal analysis is not possible.

Majority of the researchers in real time systems community evaluate correctness of new algorithms by using simulation on randomly generated task sets. As a results, number of simulation tools have been developed in last few years, some of which are discussed in next section. Most of these simulators simulate scheduling algorithms that are energy aware and schedule independent periodic task sets [1, 2, 4, 14, 16, 17, 19, 20]. A few simulators have capability of simulating energy efficient scheduling algorithm for mixed task sets containing a mix of periodic and aperiodic tasks on multi-core processor. Also in most of the simulators, the quality of service (QoS) parameters such as good documentation, flexibility to add new modules, performance analyzer, task set generator etc are not incorporated.

This paper presents an easy to use and well documented simulation tool called STREAM that can simulate energy aware real time scheduling algorithms for mixed work load on multi-core processor by incorporating majority of QoS parameters. Energy optimization is carried using dynamic voltage and frequency scaling (DVFS) technique. STREAM stands for "Simulation Tool for Real time Energy efficient scheduling and Analysis for Multi-core processors". It includes implementation of Earliest Deadline First (EDF), EDF with Deferrable Server (DS), EDF with Total Bandwidth Server (TBS), DVFS based EDF with DS (EEDFVS [25]) and DVFS based EDF with TBS algorithms for uniprocessor and multi-core processor platforms. It has modules to generate synthetic task set and to calculate various performance metrics such as energy consumption, aperiodic task's response times etc. The analytical results of the algorithms can be visualized by plotting different graphs.

STREAM is written in java programming language that makes use of object oriented paradigm. The modules are organized in such a way that highly specific or similar objects are grouped inside a single package. This helps new programmers to quickly track the required module by navigating the group hierarchy. The Graphical User Interface (GUI) of simulator is very user friendly and is easy to explore and use. The use of abstract classes facilitate addition of new modules in the current version of simulator.

Rest of the paper is organized as follows. Section II summarizes existing simulations tools. Section III briefly describes the system model which includes task, processor and energy models. Section IV presents the proposed simulator architecture that explains overall simulator design, task set generation method, details of simulation core and performance analyzer. Section V provides description of all the entities and related modules. Finally, section VI concludes the paper and states the future directions.

II. RELATED WORK

Real time scheduling is well established in literature. Many of these algorithms are validated using simulation. This section presents the survey of existing simulation tools.

Table I summarizes over 21 simulation tools existing in the literature. Some of these simulators support only periodic task model [1,2,5,6,7,8,10,11,13,14,15,17,18,20,26] and others [3,4,9,12,16,19] support mixed task model. Similarly simulators in [1,2,3,6,7,8,11,15,19] are made for uni-processor platform and in [4,5,9,10,12,13,14,16,17,18,20,26] are made for multiprocessor/distributed platform. Task set generation

module implemented in FORTAS [14], YARTISS [16], RTMultiSim [17], ERTSim [19], GEN4MAST [20] and SimSo [26] made use of UUniFast and UUniFast-Discard algorithms [21] for generating task sets. SimDVS [8], STORM [13], SPARTS [14] and YARTISS [16] support energy optimization as well. Simulators in [5,6,9,10,11,13,14,15,16,18,20,26] are open source and rest are not open source. Table I shows additional details about these simulators such as programming language, design methodology, processor model, performance analysis etc. It can be observed from table I that none of the simulators takes care of all the parameters. The proposed simulator, STREAM, deals with all the parameters except that it is not open source.

TABLE I. SUMMARY OF EXISTING SIMULATORS

Sr. No.	Simulator	Year	Language	Design	Performance Analysis	Scheduler Profiling
1	STRESS [1]	1994	Domain Specific Language	Pseudo code design	N	N
2	Generic Simulator [2]	1996	C++	Complex, Non-Modular, Redundant , No flexibility to add new module	N	N
3	GHOST [3]	1997	C	Simple, Modular design, Flexibility to add new module	N	Trace Generator
4	MAST [4]	2001	ADA-ayacc	Modular, flexible to add new component	Y	Y
5	RTSim [5]	2001	C++	-	N	N
6	Java Simulator [6]	2002	Java	Modular, Concurrent programming, Independent Component design	N	N
7	YASA [7]	2003	ANSI C	Modular and Flexible, supports RT-Linux and RTEMS	Y	N
8	SimDVS [8]	2003	-	Modular design, Flexible	Energy Management	N
9	Cheddar [9]	2004	ADA	Modular, Flexible, allows integration of third party components	N	N
10	TORSCHE [10]	2006	Matlab/ Simulink	Routine based design	Timing Analysis	N
11	Realtss [11]	2007	C/C++/TCL	Modular, flexibility to add new scheduler	N	N
12	STORM [12]	2010	Java	OOPS design, Modular, Flexible, Extensible, Reusability	N	Statistical information: deadline miss count, power consumption etc
13	FORTAS [13]	2011	Java	Modular, flexible, Follows OOPS paradigm	Comparison between different scheduler performance	N
14	SPARTS [14]	2011	Java	Modular, flexible, Follows OOPS paradigm	Execution time Vs. Task Set size, Simulation time analysis	Scheduler overhead statistics
15	omNET [15]	2012	C++	Modular	CPU Utilization Vs. deadline size and deadline stolerance	N
16	Yartiss [16]	2012	Java	OOPS design, Modular, Flexible, Extensible, Reusability	N	Tracking Preemption points
17	RTMultiSim [17]	2012	-	Follows STORM like design	CPU utilization, parallelism degree	N
18	RealtssMP [18]	2013	C/C++/TCL	Modular, flexibility to integrate new scheduler	Y	Tracking Preemption points, migration points, Deadline miss points
19	ERTSim [19]	2013	C/C++	Modular and Structural paradigm of C/C++	Utilization upper bound test, Response Time Analysis, Processor Demand Analysis	N
20	GEN4MAST [20]	2014	Python	Modular, flexible	CPU Utilization Analysis	N
21	SimSo [26]	2014	Python	Modular	Comparison between different scheduler performance	Overhead calculation such as context switches, scheduling decisions, cache related preemption delays

III. System Model

The system model includes task, processor and energy models used by different energy efficient real time scheduling algorithms. The real time scheduling algorithms that are implemented in STREAM consider multi-core system composed of *m* homogeneous processor cores and mixed task model which consist of preemptive and independent periodic as well as aperiodic tasks. Each periodic task is described by four attributes: arrival time, worst case execution time, period or inter-release time and deadline. All periodic tasks have implicit deadlines. An aperiodic task is described by two attributes: arrival time and worst case execution time.

Energy consumption of a DVFS enabled CMOS-based processor consists of two components: dynamic energy ($E_{dynamic}$) due to switching activities and static energy (E_{static}) due to leakage current. $E_{dynamic}$ is a convex function of processor speed and contributes to the largest part of total energy consumed during instruction execution [22]. It can be expressed as follows:

$$P_{dynamic} = C_{ef} \times {V_{dd}}^2 \times f \quad (1)$$

$$E_{dynamic} = P_{dynamic} \times f^{-1} \quad (2)$$

where, $P_{dynamic}$ is dynamic power consumption, C_{ef} is the effective switching capacitance, V_{dd} is the supply voltage and f is the clock frequency. For the DVFS scheduler, the frequency and voltage transition overheads are assumed to be negligibly small. The sum of dynamic energy consumption of the individual cores is taken as the total dynamic energy consumption of the multi-core processor. Table II shows the voltage and frequency settings of a synthetic processor which is used for the simulations. The voltage and frequency values of this synthetic processor are derived based on values taken from Intel Core i3-530 dual core processor [23] whose maximum frequency is 2.93 GHz, voltage variation ranges from 0.6500V to 1.400V and maximum power consumption is 101W. However, STREAM provides the flexibility to change the frequency/voltage settings as per user's requirement.

TABLE II. FREQUENCY/VOLTAGE SETTINGS OF SYNTHETIC PROCESSOR

Level	Frequency (GHz)	Voltage (V)
0	2.93	1.4
1	2.64	1.3
2	2.05	1.1
3	1.47	0.9
4	1.17	0.8
5	0.73	0.65

IV. Software Architecture of STREAM

Software architecture of STREAM provides the overall view of the system. It describes modeling and relationship between the different software entities in the system. These software entities are categorized into four subsystems: system modeler, task set generator, scheduler/controllers and performance analyzer/scheduling profiler. System modeler provides necessary working environment to facilitate and bring together the sound and flexible execution behavior of the simulator. It encapsulates task model, multi-core processor model, partition manager and energy model. Task set generator is a separate sub system that is responsible for the generation of periodic and aperiodic tasks. It makes use of basic building blocks of system modeler. Performance analyzer/scheduling profiler provides the facility to analyze and view various statistical results of the scheduler such as scheduler log traces, analytical graphs etc. Scheduler/controller is surrounded by other three subsystems. Task set generator produces a task set which is given as input to scheduler/controller subsystem. It schedules the task set, generates the output and calculates the energy consumption. The results are shown with the help of performance analyzer. Figure 1 shows the architecture of STREAM in which relationship between different entities is broadly shown to visualize the overall flow of the system.

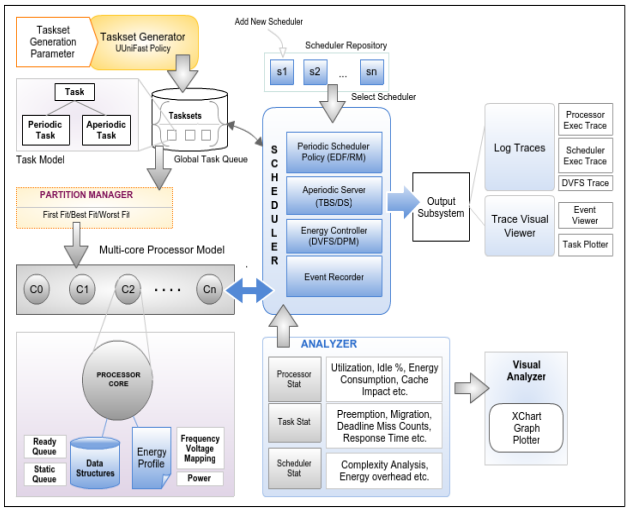

Figure 1. Software Architecture of STREAM

V. Entity and Module Description:

The subsystems mentioned in the previous section are composed of different entities. This section presents the details of all these entities including their attributes, modules and relationships among them.

A. System Modeler:

This subsystem includes task model, processor model, energy profiler model and partition manager. The algorithms implemented by STREAM use a mix of periodic and aperiodic

tasks. Hence task model includes the characteristics of both types of tasks and state transition details. The task state diagram shown in figure 2 gives the details of all the states in which a task may exist during execution. Tasks begin with a state called Zombie where it lives in static list of suspended tasks outside the execution environment. Upon arrival, it advances to a *Ready* state where it enters the ready queue of the system. Upon getting a turn, scheduler makes it eligible for execution on the processor (*Running*). This is depicted by a transition from *Ready* state to *Running* state. While running, a task may get preempted by arrival of a high priority task in which case the scheduler brings it back to *Ready* state and gives chance to a high priority task. On termination, a task is moved back to a *Zombie* state, where it waits until its next invocation (in case of periodic task) or it marks its finished status (in case of aperiodic task). Few scheduling algorithms may allow tasks to migrate from one processor core to another on preemption, for reasons such as reducing the response time. This state is depicted in dotted lines (indicating that it's a scheduler specific implementation) and is called as *Migrated Ready*. On migration, it follows the same fundamental state behavior as explained above.

The task model is represented as generic class which is extended to add specific task type such as periodic and aperiodic task type. In order to perform the state transitions during execution, different functionalities are implemented. They are getArivalTime(), getWCET(), getPeriod(), getNextDeadline(), isRunning(), isPreempted(), isFinished() etc.

Processor model represents virtual processor entity, the instance of which represents one processor core. It maintains a static list of periodic tasks and dynamic list of periodic as well as aperiodic jobs. It performs important functionalities such as calculation of core utilization at each scheduling point (here scheduling point refers to either arrival of a higher priority job or completion event), maintaining the record of execution frequency, count of idle and busy time etc. Some of these functionalities are named as: getTaskset(), getReadyList(), populateReadyList(), getLocalTime(), getCoreUtil() and getCurrentFrequency(). It also includes an abstract energy profile class which is a Map like structure providing <key, value> construct. The key is represented as frequency level while the value is represented by the corresponding voltage at that frequency level. This class can be extended to provide an energy profile of a real energy specification or synthesized energy specification of standard processor model of intel, ARM, AMD etc. The functionalities of energy profile class include getVoltage(), getNearAbsoluteFrequency(), getEnergyProfile(), getCapacitance(), getUnitEnergy() etc.

In order to apply partition scheme, partition manager class is written that provides different partitioning schemes such as first fit, worst fit, best fit [24] etc. Algorithms implemented in STREAM follow partition approach for periodic tasks and global approach for aperiodic tasks in which aperiodic job can migrate to other core upon its preemption. In addition, one more abstract method is written which can be overridden to provide custom implementation of the partition scheme.

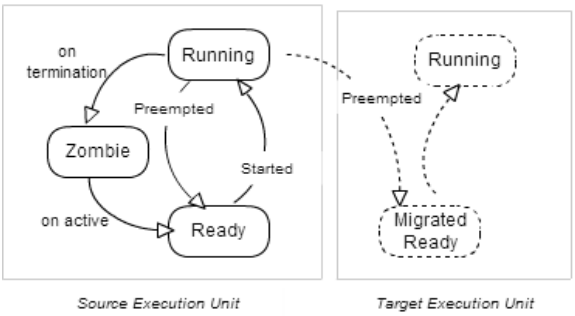

Figure 2. Task State Transition Diagram

B. Task set Generator:

Task set generator is a separate subsystem which is responsible for generating synthetic task sets. Figure 3 shows the implementation of synthetic task set generation. The attributes of task set generator class are shown in a separate box in the flow chart in figure 3. The task set generation algorithm is responsible for generating periodic and aperiodic tasks. Each iteration in the flow chart generates one mixed task set. For generating periodic/aperiodic tasks, it makes use of two algorithms: RandFixedSum and UUniFast-Discard [21]. RandFixedSum algorithm generates a vector of uniformly distributed utilization values such that the sum of those utilization values is equal to periodic/aperiodic load. UUniFast-Discard algorithm is used to discard those utilization values that are exceeding 1 in case of periodic task. Period of each periodic task is chosen to be a random number such that it is a natural factor of a given hyper period value. Rest of the task attributes are calculated using UUniFast algorithm. Arrival time of the aperiodic task is obtained as a random number over a hyper period and the minimum inter-arrival time between two aperiodic tasks is not more than 5% to 10% of hyper period. The WCET of aperiodic task is calculated as a product of utilization value and the time left till the hyper period from its arrival. The synthetic task generator generates periodic tasks for a wide range of utilization: 30% to 80% and aperiodic tasks utilization is based on the remaining utilization of processor. The number of periodic tasks in a task set are ranging from 2 to 20 and aperiodic tasks in a task set ranges from 2 to 5 for the purpose of simulation but STREAM has the flexibility to generate even more number of tasks. For each class of fixed number of tasks and utilization, 100 task sets are generated.

Task set generator is implemented as a generic interface that provides the flexibility of adding new task set generator algorithms.

C. Schedulers, Servers and Controllers:

In the proposed simulation tool, various flavors of dynamic priority schedulers have been implemented. Though the current version of STREAM do not focus on fixed priority scheduling techniques, it provides the flexibility to add new schedulers. The schedulers currently implemented in STREAM are all based on EDF scheduling policy. The aperiodic servers currently implemented are TBS and DS. To

make these schedulers energy efficient, a dynamic energy optimization technique namely, DVFS is implemented.

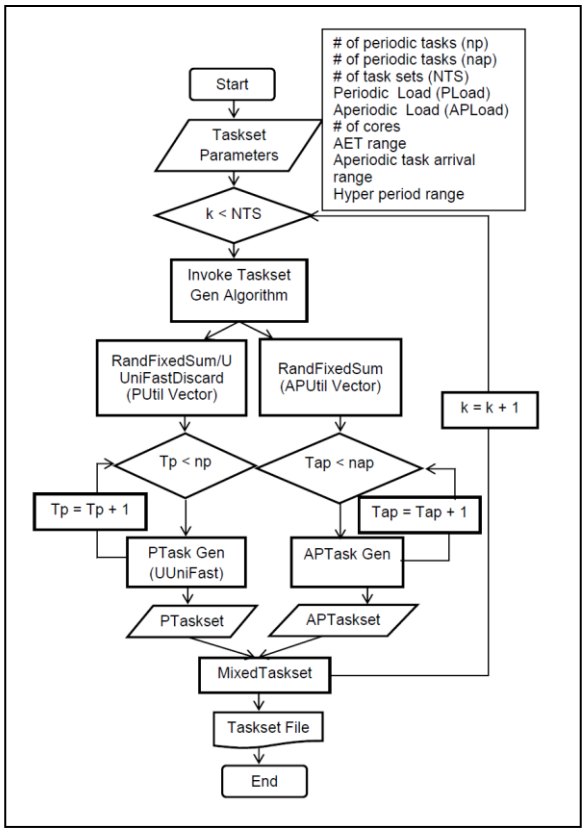

Figure 3. Task Set Generator Flow Chart

The flexible design of simulator allows execution of different schedulers separately as well as they can execute in combination with different aperiodic servers and energy controllers depending upon the user requirement. For example, if a user wants to execute non energy aware scheduler for hard real time tasks, then EDF can be selected or if a user wants to execute energy aware scheduler for mixed task set, then a combination of EDF, TBS and DVFS can be selected. The scheduler, aperiodic server and energy controller are implemented as java interface that makes them easy to add new scheduling policy, another aperiodic server or any other energy optimization policy (e.g. Dynamic Power Management).

Figure 4 shows the detailed class diagram of the simulation core which includes scheduler, server and energy controller. EDF scheduling policy is implemented with two different aperiodic servers: TBS and DS. Utilization update [27], a DVFS technique is implemented for optimizing dynamic energy of cores. Aperiodic jobs are executed at maximum frequency in case of DVFS and are allowed to migrate to a processor core with minimum utilization in order to achieve minimum response time.

D. Results, Performance Analyzer and Profiler:

A separate subsystem is made for displaying the results, scheduler profiling and performance analysis. Scheduler profiler is used to test the validity and correctness of the scheduler. It generates the log traces in the background of the scheduler and records the details of every single scheduler event. This log trace provides the detailed action of events which happened at a particular time within one hyper period. Besides this, it generates per-core execution flow by tracking down all the state transitions of tasks at every unit of time.

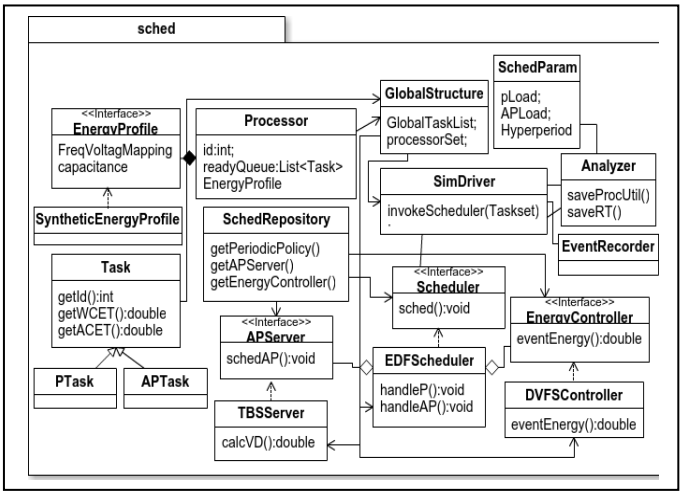

Figure 3. Simulation Core

This per-core execution flow is useful to find out various statistical information such as idle/busy percentage of processor time. It also generates energy statistics according to the employed energy scheme. For example, in case of DVFS, detailed information of frequency-voltage scaling is recorded for every scheduling decision over a hyper period.

Analyzing the performance of scheduler output is very important in order to study behavior and working of schedulers. STREAM provides many run time analysis options that makes the whole design attractive and easy to handle. It also provides a visual plotter tool to display the analysis graphically. Figure 5 shows sample output from STREAM. Following are the different runtime analysis options that can be switched on or off as per requirement.

- Task runtime execution and behavior analysis: This includes tracking state of the tasks at various points of execution, counting task's decision points such as arrival points, completion points, preemption points etc., measuring the response time of tasks etc.
- Processor Runtime Statistics: This includes tracking of processor busy/idle time statistics, logging the operational state (e.g. powered, shutdown, suspended etc.) of individual core at various points of schedule, logging run time energy consumption statistics for individual core as calculated by specified energy model, logging per-core cache memory behavior for its hot access and cold access over the entire schedule.

- Scheduler overhead analysis: This includes measuring the extra cost of execution incurred by a scheduler program in addition to above two analysis. This extra cost includes the time taken by scheduler program to invoke various scheduling decisions such as time required for task's state transition, to handle preemption, to change processor parameters like frequency/voltage settings, cache impact detection, ready job queues etc.

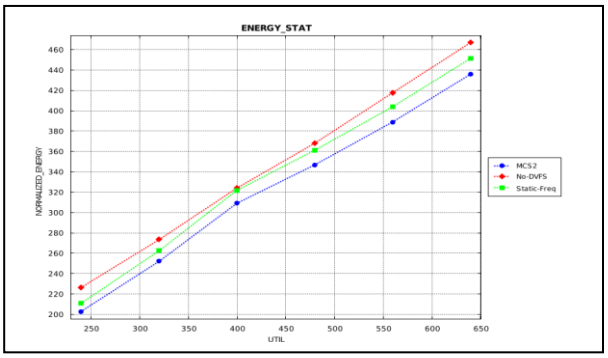

Figure 5. Example graph produced by plotter tool: Normalized Energy Consumption Vs. Utilization

VI. CONCLUSION

This paper proposed a simulation tool - STREAM, for simulation of energy efficient real time scheduling algorithms for mixed task set on multi-core processors. It provides simulation of different real time scheduling techniques for periodic as well as mixed task set. It also has a separate module for generation of mixed task set. Efficient visual display is provided to view scheduler log traces and profiling results. A graph plotter is provided to visualize the performance analysis of different schedulers. The energy efficient schedulers implemented in this simulator currently does not take care of static energy optimization. The future work focuses on implementation of leakage aware scheduling algorithms for mixed task set.

REFERENCES

[1] N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings, "STRESS: a simulator for hard real-time systems", Software Practice and Experience, 24(6), pp. 543–564, June 1994.

[2] S. D. Vroey, J. Goossens, C. Hernalsteen, "A generic simulator of real-time scheduling algorithms", Proceedings of the 29th Annual Simulation Symposium, pp. 242-249, 8-11 Apr 1996.

[3] Sensini, Fabrizio, G. Buttazzo, and P. Ancilotti. "Ghost: A tool for simulation and analysis of real-time scheduling algorithms", Proceedings of the IEEE Real-Time Educational Workshop, 1997.

[4] M. G. Harbour, J. J. G. Garcia, J. C. P. Gutierrez, J. M. Drake, "MAST: Modeling and analysis suite for real time applications", 13th Euromicro Conference on Real-Time Systems, pp.125,134, 2001.

[5] A. Manacero, M. B. Miola, V. A. Nabuco, "Teaching real-time with a scheduler simulator", 31st Annual Frontiers in Education Conference, vol.2, no., pp.T4D,15-19, 2001.

[6] G. Jakovljevic, Z. Rakamaric, D. Babic, " Java Simulator of Real-Time Scheduling Algorithms ", 24th International Conference on Information Technology Interfaces, June 24-27, Cavtat, Croatia, 2002.

[7] Blumenthal, Jan, et al. "YASA-A Framework for Validation, Test, and Analysis of Real-Time Scheduling Algorithms." Proceedings of 5th Real-Time Linux Workshop. 2003.

[8] D.Shin, W. Kim, J. Jeon, J. Kim, and S. L. Min, "SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage and Frequency Scaling Algorithms", PACS 2002, pp. 141–156, Springer 2003.

[9] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a Flexible Real Time Scheduling Framework", in Proc. of SIGAda, 2004.

[10] Sucha, Premysl, et al. "Torsche scheduling toolbox for matlab", Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006.

[11] A. Diaz, R. Batista, O. Castro, "Realtss: a real-time scheduling simulator", 4th International Conference on Electrical and Electronics Engineering, pp. 165 - 168, 2007.

[12] R. Urunuela, A.-M. Déplanche, and Y. Trinquet, "STORM: a Simulation Tool for Real-time Multiprocessor Scheduling Evaluation", GDR SOC SIP, p. 1, 2009.

[13] P. Courbin and L. George, "FORTAS: Framework fOr Real-Time Analysis and Simulation", Proceedings of the 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, pp. 21–26, 2011.

[14] B. Nikolic, M.A. Awan, S.M. Petters, "SPARTS: Simulator for Power Aware and Real-Time Systems," 10th International Conference on Trust, Security and Privacy in Computing and Communications, pp.999,1004, 16-18 Nov. 2011.

[15] Z. Khalib, B. Ahmad, O. Bi, "Performance Analysis of a Non-preemptive Dynamic Soft Real Time Scheduler Using Discrete Event Simulator", Computational Intelligence, Modelling and Simulation, 2012 Fourth International Conference on , pp.182,187, 25-27 Sept. 2012.

[16] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, and M. Qamhieh, "YARTISS: A tool to visualize, test, compare and evaluate real-time scheduling algorithms," Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, Pisa (Italy), 2012.

[17] Hangan, Anca, and G. Sebestyen. "RTMultiSim: A Versatile Simulator for Multiprocessor Real-Time Systems." Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, Pisa (Italy). 2012.

[18] A. D. Ramirez, D. K. Orduno, P. M. Alvarez, "A multiprocessor real-time scheduling simulation tool," 22nd International Conference on Electrical Communications and Computers, pp.157,161, 27-29 Feb. 2012.

[19] A.S. Pillai, T.B. Isha, "ERTSim: An embedded real-time task simulator for scheduling", IEEE International Conference on Computational Intelligence and Computing Research, pp.1-4, 26-28 Dec. 2013.

[20] J. M. Rivas, J. J. Gutiérrez, and M. G. Harbour, "GEN4MAST: A Tool for the Evaluation of Real-Time Techniques Using a Supercomputer.", Proceedings of 3rd International Workshop on Real Time and Distributed Computing in Emerging Applications co-located with 34th IEEE Real Time Systems Symposium, Dec 2014.

[21] 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, 2010.

[22] T.D. Burd, R. W. Brodersen, "Energy efficient CMOS microprocessor design", System Sciences, Proceedings of the Twenty-Eighth Hawaii International Conference, pp.288-297, 3-6 Jan 1995.

[23] http://cpuboss.com/cpu/Intel-Core-i3-530.

[24] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems." ACM Computing Survey. 43, 4, Article 35, October 2011.

[25] Digalwar, M., Mohan, S., Raveendran, B.K., "Dynamic voltage and frequency scaling scheduling algorithm for mixed task set," 8th IEEE International Conference on Industrial and Information Systems, pp.643,648, 17-20 Dec. 2013.

[26] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche. "SimSo: A simulation tool to evaluate real-time multiprocessor scheduling algorithms." In Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, 2014.

[27] Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In Proc. ACM Symposium on Operating Systems Principles, 2001.