Minimal Task Allocation in Multiprocessors using improved next-fit for RM scheduling

Muthu Kumar B
Amrita Vishwa Vidyapeethem
Coimbatore, India

Anju S Pillai
Amrita Vishwa Vidyapeethem
Coimbatore, India

ABSTRACT

Task-processor allocation in multiprocessors can be accomplished efficiently for reducing the required number of processors for the given task set, accounting reduced power consumption with maximum processor utilization. This work is based on next fit algorithm using Rate Monotonic Algorithm (RMA) for a fixed priority system. The work proposes a minimal task allocation algorithm for multiprocessor environment. The proposed method reduces the number of processors required for a given task set using improved next fit algorithm and the same has been evaluated and tested. The proposed algorithm gives better results when there is large number of tasks in the system.

General Terms

Task-processor allocation algorithm, Fixed priority system, Rate Monotonic, Scheduling, Improved Next-fit algorithm, Task allocation.

Keywords

Multiprocessor systems, Task-processor allocation algorithm, multiprocessor allocation, Rate Monotonic algorithm, Next fit Algorithm, Improved Next-fit algorithm.

1. INTRODUCTION

Recent high-end technologies in embedded systems have lead to the requirement of more efficient usage of processors with zero loss. Traditional uniprocessor systems have become outdated and are not able to cope up with the current technological balance. To catch up with the improving performance requirements, there is a need for multiprocessors. Task allocation and scheduling is a challenging problem. The scheduling part is evolved to its most efficient way and in the allocation bin-packing [1] is one of the most efficiently used algorithms. Though the distributed systems and multiprocessors are highly advantageous over uniprocessors, their scheduling algorithms [2] are not yet fully evolved to the ultimate. In this paper, a motivation to design a heuristic algorithm for decreasing the number of required processors, thereby increasing processor utilization and reducing the power consumption is dealt with. Many simple heuristics [3], [4] algorithms and complex algorithms [5], [6] are proposed; out of which the current work falls into the category of complex algorithms.

Scheduling of multiprocessors can be basically implemented in two ways: global scheduling [7] and partitioned scheduling. In partitioned scheduling the task-processor allocation is done so as to utilize the platform fully. Moreover scheduling is of two types, static scheduling and dynamic scheduling. The latter case uses Earliest Deadline First (EDF) algorithm for total processor utilization, while the former case has its most efficient usage of processors by applying RMA [8]. Static scheduling has its optimal solution in uniprocessor and has poor solutions in multiprocessor environment. Allocating the tasks in an optimal way and scheduling using uniprocessor algorithms will lead to a better scheduling in multiprocessors. Many simple allocation algorithms [2], [3] are proposed. The RMA has an upper bound [9] which makes it unsuitable for complete processor utilization. This paper implements a bin-packing algorithm with higher processor utilization and less time complexity for allocating tasks among the shared memory multiprocessor systems. The system has a static scheduling algorithm (RM) which is implemented in a multiprocessor system. In the first phase the next-fit for RM is used to allocate tasks to the processors and the second phase reallocates the tasks by partitioning. Often the processors are described as bins. So the bin-packing is done by reallocation of tasks. Thus, the bin-packing implemented in static scheduling decreases the number of processors required to allocate the tasks, enhancing maximum utilization of the processors.

2. RELATED WORK

In the literature there are available many task allocation algorithms for multiprocessor system. The performance of the system is improved by using dynamically reconfigurable accelerator cores. A heuristic algorithm to assign tasks to a set of processing sites by balancing the load over different processors is described in [10]. An Energy-efficient task allocation and scheduling schemes with deterministic fault-tolerance capabilities are proposed for symmetric multiprocessor systems when executing tasks with hard real-time constraints is described in [11]. This is achieved by optimally balancing the workload in the system. A novel lifetime reliability-aware task allocation and scheduling algorithm based on simulated annealing technique is presented [12] to estimate the lifetime reliability of multiprocessor platforms when executing periodical tasks. In [13], the authors propose an adaptive energy efficient task allocation scheme for a multiprocessor System-on-Chip (SoC). In the work, an offline task schedule is generated which is prolonged to adapt the energy availability at run time.

Normally, the optimizations are performed for each individual task upon activation for better resource utilization. But, these optimizations are sub-optimal from the system point of view. A task allocation that considers the system level resource management is proposed [14]. The proposed task allocator has a run time self adaptability with respect to change in application. Energy consumption is largely influenced by task allocation algorithms especially in Network-on-Chip (NoC) heterogeneous multiprocessor systems. An Integer Linear Programming (ILP) approach is described [15] with Simulated Annealing with Timing Adjustment (SA-TA) heuristic to accelerate the optimization process. System on a programmable chip design for a multicore embedded system is proposed in [16]. The optimization criteria of the task allocator is made to adapt itself according to the relative scarcity of different types of resources, by dynamically adjusting a set of parameters at run time is described in [17]. By this approach the resource bottlenecks can be effectively mitigated. The work uses self adaptability in task allocation to obtain the desired performance.

The rest of the paper is arranged as follows: In section 3, the multiprocessor environment used in this work is explained. In section 4, task processor allocation and task scheduling is introduced. In section 5, the proposed model of the algorithm is given. In Section 6, the implementation of the proposed method is described. In section 7 the results of improved next-fit is given. Finally section 8 concludes with the results of the proposed method.

3. MULTIPROCESSOR SYSTEM

The multiprocessor systems are of two types: distributed multiprocessor systems and shared multiprocessor systems. The distributed systems have their own memory and individual processors but in the shared multiprocessor systems, there is a common memory which is accessed by all the processors. The system model in the current work is a shared or common memory multiprocessor system. The shared memory has the task allocation algorithm which decides according to the given task set, the required number of processors to be present in the system.

4. TASK-PROCESSOR ALLOCATION

4.1 Task Allocation

Task allocation is about allocating the given tasks to minimum number of processors. There are many allocation algorithms such as in static scheduling: utilization balancing, next-fit algorithms and in dynamic scheduling: bin-packing, focused addressing and bidding and buddy algorithms.

The advantage of allocation in dynamic scheduling based algorithm is that it provides fault tolerant scheduling and makes use of redundant processor if any of the processors fail when executing a task or a job.

When it comes to static scheduling based allocations where priorities are fixed, handling fault tolerant mechanisms are not very possible. Because, it is a fixed scheduling based algorithm. The next-fit algorithm is used for RM and is better than utilization balancing algorithm but due to the upper bound of RM, it does not pack the bin fully. The proposed algorithm ensures that the bins are packed completely leading to a reduction in number of processors or bins.

The bin-packing in RM has less number of required bins more or less equal to bin-packing in EDF. So the disadvantage of upper bound in RM is reduced by using an effective task allocation algorithm in this work.

4.2 Task scheduling

Scheduling in multiprocessors is done by RM or EDF according to the application for which the processor employed. The RM is a static scheduling algorithm and the priority assignment is based on the periods of the tasks. It is fixed priority scheduling and when the task allocation is done, RM will be executed in each processor similar to a uniprocessor scheduling.

The RM requires the execution time or computation time and time period of the task. Consider a task set given in Table 1.

Table 1. Task set for RM

Task	Time Period	Execution Time
A	3	1
B	6	1
C	9	2

The tasks are prioritized according to the shortest period first in RM as in Fig.1. Task A is the highest priority task, Task B the middle priority and Task C the least priority task.

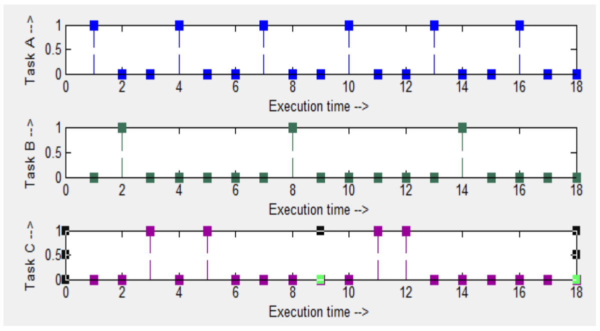

Fig 1: Execution trace of the task set by RM

5. PROPOSED ALGORITHM MODEL

Given '*n*' number of tasks, where *n* is large. The proposed algorithm has basically three phases. In phase one, task allocation by next-fit algorithm is done. In phase two, the algorithm checks whether the bins can be fully packed and if not task reallocation is performed so as to completely fill the bins. This overcomes the disadvantage of next-fit algorithm of not packing the bins completely. In phase three, task splitting is achieved for high utilization tasks. The tasks are split and packed into the bins as close to the respective upper bounds of the RM algorithm which may be implemented at the later stage similar to uniprocessor scheduling.

The minimum and maximum number of required processors and the effective use of the two phases are formulated in Table 2. It is seen that when both the phases are implemented the algorithm offers the maximum effective output by reducing the number of processors.

Table 2. Algorithm model results

Bin Packing	Task Split	Result
N	N	No reduction in bins
N	Y	Reduction of bins in less amount
Y	N	Better Reduction
Y	Y	Maximum Reduction of bins as possible

Fig. 2 elaborates the task allocation strategy by the proposed algorithm.

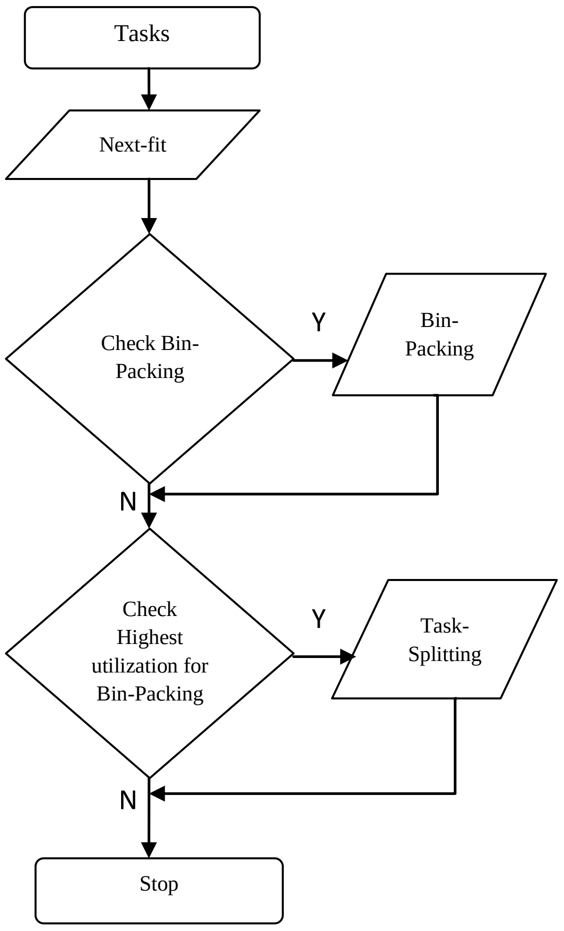

Fig 2: Proposed allocation model

5.1 Phase one: Next-Fit Allocation

The next-fit allocation algorithm [18] is formulated for fixed priority system which uses RM. It helps in fixed distribution of the tasks when it is allocated to a particular processor or bin. The scheduling then follows the RM.

When the tasks are to be fed into the multiprocessor environment, it is to be divided into *m* classes, a task T_i belongs to a CLASS *j*, so that $0 \leq j < m$.

$$2^{\frac{1}{j+1}} - 1 < \frac{C_i}{T_i} < 2^{\frac{1}{j}} - 1$$

Where; C_i is the Computation time or execution time of a task and T_i the time period of a task.

The tasks are divided into the following classes and each class is based upon the utilization factor of the task ($U_i = C_i/T_i$). Each class corresponds to a processor needed for allocating the whole task set.

CLASS 1: $2^{\frac{1}{2}} - 1 < C_1 f\ T_1 < 2^{\frac{1}{1}} - 1$

CLASS 2: $2^{\frac{1}{3}} - 1 < C_2 f\ T_2 < 2^{\frac{1}{2}} - 1$

CLASS 3: $2^{\frac{1}{4}} - 1 < C_3 f\ T_3 < 2^{\frac{1}{3}} - 1$

CLASS 4: $0 < C_4 f\ T_4 < 2^{\frac{1}{4}} - 1$

When the above allocation is done, the upper bound is also checked so that it lies within the feasibility of RM scheduling. If more tasks are assigned in the same CLASS, exceeding the upper bound of scheduling, it is to be accommodated in a new processor of the same CLASS. This algorithm has a disadvantage that some processor will have less number of tasks or even with single task, leading to more idle time.

Thus, in phase one the task allocation with next-fit algorithm is done. The next phases will overcome the above mentioned disadvantage.

5.2 Phase two: Bin-Packing of Lower Utilization Tasks

After allocating the tasks, there are two phases introduced in this minimal task allocation algorithm in which the second phase is about checking the lower utilization tasks whether they can be accommodated in the other processors not taking into account the rules of next-fit algorithm. A processor can accommodate tasks of different classes to only see that it packs the bins [1] fully to the upper bound of RM according to the number of tasks accommodated.

5.3 Phase three: Task-Splitting of Higher Utilization Tasks

The previous phase eliminates the disadvantage of lower utilization tasks and their class allocation strategy leaving to eliminate the remaining higher utilization tasks which are reallocated in this final phase of the allocation strategy.

The final phase involves the task splitting of higher utilization tasks and allocating the tasks in possible remaining idle space in other processors.

Shinpei Kato et.al. [19] have proposed a task splitting strategy for multiprocessors. Unlike the traditional way of allocating tasks to processors, in this strategy the tasks are split and the partitioned tasks are executed in different processors. The system model is a memory shared multiprocessor and the code and data are shared by each processor. The tasks are fixed and new tasks are not either added or deleted from the present task set. The task splitting strategy adopted is explained below:

First step of the task splitting strategy is that it checks for total utilization of task in an arbitrary task set allocated by bin-packing whether the utilization of the processor is within the schedulable bound. If so, there is no need of task splitting.

If the above condition fails, it checks next whether there are processors available to accommodate the partitioned tasks. Then on successful availability of processors, the algorithm splits the tasks and calculates the remaining utilization (U_{REM}), and the partitioned computation times: C_i', C_i'', … and the corresponding time periods: T_i, $T_{i+1\ldots}$. It then checks for the upper bound whether the addition of the task will make the schedule infeasible. If the addition of a partitioned task makes it to exceed the upper bound, the task is moved to the next processor after checking the feasibility.

In this phase of the algorithm the tasks with higher utilizations are considered one by one in the decreasing order of the utilization for task splitting. The tasks are partitioned into appropriate numbers depending upon the available idle time in the processors. Let a task τ_i be partitioned into $\tau_{i'}$ and $\tau_{i''}$ which is the last possible task for splitting, then, there is no need to calculate for upper bound for the new processor P_{m+1} because there is only τ_i'' portion to be assigned and it always will meet the deadline. When the task τ_i is split into τ_i' and τ_i'' its corresponding computation time is changed as C_i' and C_i'', with time period $T_{i'}$ and T_i''. The tasks τ_i' and τ_i'' does not really mean that the task τ_i is split but stands for a pseudo codes which reserve the processor time in P_m and P_{m+1}. The task splitting strategy is described in Fig. 3.

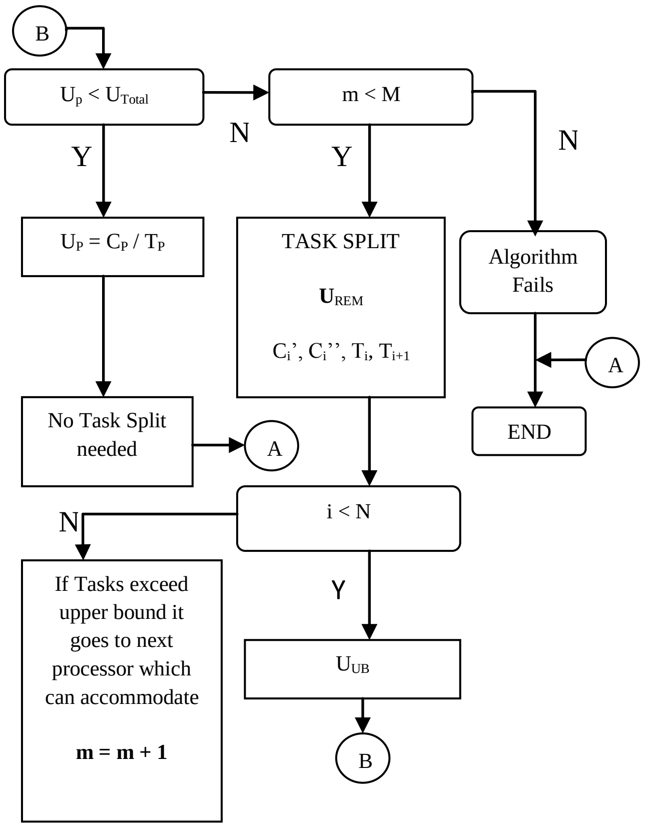

Fig 3: Task Split Strategy

6. IMPLEMENTATION OF THE MODEL WITH AN EXAMPLE

Table 3. Task set

TASKS	COMPUTATION TIME	TIME PERIOD	UTILIZATION
τ_1	3	4	0.75
τ_2	4	6	0.6667
τ_3	6	10	0.6
τ_4	7	14	0.5
τ_5	6	15	0.4
τ_6	5	18	0.2778
τ_7	4	22	0.1818
τ_8	3	25	0.12
τ_9	5	30	0.1667
τ_{10}	10	35	0.2857
τ_{11}	18	40	0.45
τ_{12}	21	42	0.5
τ_{13}	11	45	0.2444
τ_{14}	4	50	0.08
τ_{15}	8	53	0.1509
τ_{16}	20	58	0.3448
τ_{17}	41	62	0.6612
τ_{18}	20	65	0.3077
τ_{19}	15	72	0.2083
τ_{20}	14	75	0.1867

The table 3 consists of a set of synthetically generated tasks by UUnifast algorithm [19] and is implemented by the proposed method. The UUnifast algorithm [19] generates tasks of uniform distribution with O(n) complexity. In phase one, by using next-fit algorithm; the tasks are allocated to the four defined classes. But, when the assigned tasks in a particular class fail to meet the upper bound criteria, additional processors are added in the system with the same class. Thus, finally for the above 20 tasks in the system, there are 7 processors required in class 1, 3 processors in class 2, 1 processor in class 3 and 2 processors in class 4. Therefore the next-fit algorithm for scheduling the above tasks with RM policy requires 13 processors totally. It can be noted that out of these 13 processors many processors have large idle time. These details are summarized in table 4.

Table 4. Next-fit allocation

Tasks allocated	Total Utilization
τ_1 – Class 1	0.75
τ_2 – Class 1	0.6667
τ_3 – Class 1	0.6
τ_4 – Class 1	0.5
τ_5, τ_6 – Class 2	0.6778
τ_7, τ_8, τ_9, τ_{14}, τ_{15} – Class 4	0.6994
τ_{11} – Class 1	0.45
τ_{12} – Class 1	0.5
τ_{13}, τ_{19} – Class 3	0.4527
τ_{10}, τ_{16} – Class 2	0.6305
τ_{17} – Class 1	0.6612
τ_{18}– Class 2	0.3077
τ_{20} – Class 4	0.1867

The processors are not fully utilized up to the upper bound of feasibility by RM. They are feasible with RM but require a large number of processors. This disadvantage can be overcome by eliminating the lower and higher utilization tasks. The lower utilization tasks are directly packed into the processors available with more ideal time irrespective of classes.

Table 5. Bin packing reallocation

Tasks allocated	Total Utilization
τ_1	0.75
τ_2	0.6667
τ_{17}	0.6612
τ_3, τ_{19}	0.8083
τ_4, τ_{10}	0.7857
τ_5, τ_{16}	0.7448
τ_{12}, τ_6	0.7778
τ_{11}, τ_{18}	0.7577
$\tau_{13}, \tau_{20}, \tau_7$	0.6129
$\tau_9, \tau_8, \tau_{14}, \tau_{15}$	0.5176

The lower utilization tasks are now allocated to the bins regardless of the classes and 10 processors are required in the second phase of implementation of the proposed algorithm.

The next phase is to eliminate the idle time created by the higher utilization tasks. The higher utilization tasks cannot be packed as the lower utilization tasks; because the idle time of other processors may not be feasible to accommodate due to the upper bounds of RM. Hence the tasks are split using the task split strategy explained in section 5.3 and packed in different processors. The task τ_i is split into τ_i1 and τ_iN and allocated in different processors according to the feasibility.

Table 6. Task split reallocation

Tasks allocated	Total Utilization
τ_2, τ_14	**0.8283**
τ_{17}, τ_12	**0.8283**
τ_3, τ_{19}	0.8083
τ_4, τ_{10}	0.7857
$\tau_5, \tau_{16}, \tau_15$	**0.7796**
τ_{12}, τ_6	0.7778
$\tau_{11}, \tau_{18}, \tau_16$	**0.7747**
$\tau_{13}, \tau_{20}, \tau_7, \tau_13$	**0.7567**
$\tau_9, \tau_8, \tau_{14}, \tau_{15}, \tau_11$	**0.7433**

The number of processors after the higher utilization task is checked and split (τ_11 to τ_16) into six parts so that it can be allocated within the feasibility of RM in 9 processors. The utilization which is improved than the previous phase nearing to the upper bound is shown above. The same set of tasks when implemented using bin-packing and scheduled with EDF which has an upper bound of 100% requires a minimum of 8 processors. The improved next-fit algorithm minimizes the number of processors approximately equal to EDF feasible scheduling.

7. RESULTS

The improved next fit algorithm gives better result when implemented with large number of tasks. After eliminating the least utilization tasks by bin-packing and high utilization tasks by task split, the required numbers of processors have considerably reduced with the proposed algorithm.

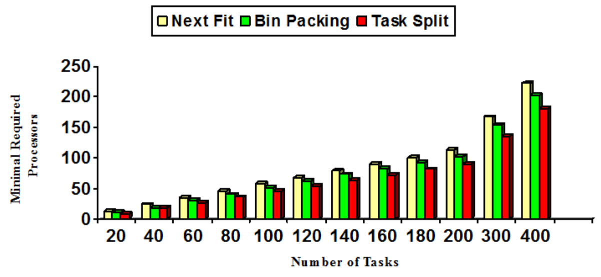

Fig 4: Minimal processors comparison

It is seen that there is shoot of 1.1 times in total utilization rate by using improved next-fit algorithm than standard next-fit algorithm. Thus, by implementing the proposed algorithm, minimum number of processors is only used. Fig.4 depicts the reduction in number of required processors, which decreases drastically with larger number of tasks in the system.

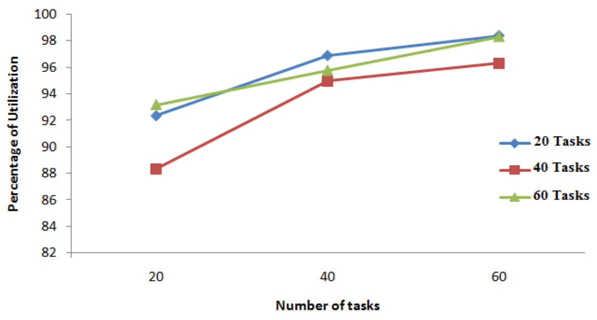

Fig 5: Tasks Vs Utilization

The improved next fit algorithm works efficiently with large number of tasks. By figure 5 it is observed that the utilization of the total system is increased by each phase, finally resulting in an improved utilization rates with maximum elimination of idle time of processors.

Table 7. Improved next-fit results

Number of tasks/Algorithm	20	40	60
Standard Next-fit	92.37%	88.36%	93.2%
Improved Next-fit	98.42%	96.32%	98.35%

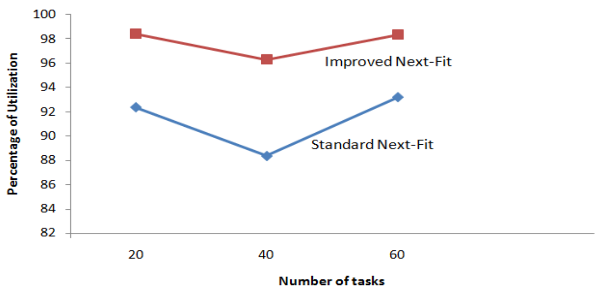

Fig 6: Standard Next-fit Vs Improved Next-fit

8. CONCLUSION

In this paper, an improved next-fit task allocation algorithm is proposed. The algorithm is tested for the particular task set and it is proved that it reduces the required number of processors by 1.44 times than using the standard next-fit algorithm. Also, the total processor utilization is improved by 1.1 times than when using standard next-fit algorithm, eliminating maximum idle time. The RM inspite of being the optimal scheduling algorithm for fixed priority system has the disadvantage of upper bounds for feasibility, which is minimized in the multiprocessor environment by the use of the improved next-fit algorithm. For optimal performance evaluation, the proposed algorithm is to be tested with large number of task sets.

9. REFERENCES

[1] WANG Tao and LIU Da-Xin. “The Performance Evaluation of Rate Monotonic Tasks Assignment Algorithms on Multiprocessor,” Computer Science, China, Vol.34, pp.272-276, 2007.

[2] A. Abraham, R. Buyya, and B. Nath. Nature’s heuristics for scheduling jobs on computational grids. In The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), India, 2000.

[3] S.K. Dall and C.L. Liu, “On a Real-Time Scheduling Problem,”Operations Research, vol. 6, no. 1, pp. 127-140, 1978.

[4] A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son, “New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems,” IEEE Trans. Computers, vol. 44, no. 12, pp. 1429-1441, Dec. 1995.

[5] D. Peng, K. Shin, and T. Abdelzaher, “Assignment and Scheduling Communicating Periodic Tasks in distributed Real-Time Systems,” Trans. Software Eng., vol. 23, no. 12, pp. 745-758, Dec. 1997.

[6] Satoshi Fujita, ”A Branch-and-Bound Algorithm for solving the Multiprocessor Scheduling Problem with Improved Lower Bounding Techniques,” in IEEE Transactions on computers, Vol.60, No.7, July 2011.

[7] S.K. Baruah and J. Goossens, “Rate-Monotonic Scheduling on Uniform Multiprocessors,” IEEE Trans. Computers, vol. 52, no. 7, pp. 966-970, July 2003.

[8] C.L. Liu and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” J. ACM, vol. 20, no. 1, pp. 46-61, 1973.

[9] Jose´ M. Lo´pez, Jose´ L. Dı´az, and Daniel F. Garcı´a, “Minimum and Maximum utilization bounds for multiprocessor Rate Monotonic scheduling,” IEEE Transactions on Parallel and Distributed Systems, Vol.15, No.7, 2004.

[10] BEAUVAIS, Jean-Pime; DEPLANCHE, Anne-Marie; , "A Task Allocation Algorithm In a Multiprocessor Real-Time System," *Parallel Processing, 1993. ICPP 1993. International Conference on* , vol.2, no., pp.130-133, 16-20 Aug. 1993.

[11] Wei, T.; Mishra, P.; Wu, K.; Liang, H.; , "Fixed-Priority Allocation and Scheduling for Energy-Efficient Fault Tolerance in Hard Real-Time Multiprocessor Systems," *Parallel and Distributed Systems, IEEE Transactions on* , vol.19, no.11, pp.1511-1526, Nov. 2008.

[12] Lin Huang; Feng Yuan; Qiang Xu; , "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.* , vol., no., pp.51-56, 20-24 April 2009.

[13] Tongquan Wei; Yonghe Guo; Xiaodao Chen; Shiyan Hu;, "Adaptive task allocation for multiprocessor SoCs," *Quality Electronic Design (ISQED), 2010 11th International Symposium on* , vol., no., pp.538-543, 22-24 March 2010.

[14] Jia Huang; Raabe, A.; Buckl, C.; Knoll, A.; , "Runtime adaptive allocation of dynamically mixed tasks on a heterogeneous MPSoC platform," *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on* , vol., no., pp.34-41, 26-28 Oct. 2010.

[15] Jia Huang; Buckl, C.; Raabe, A.; Knoll, A.; , "Energy-Aware Task Allocation for Network-on-Chip Based Heterogeneous Multiprocessor Systems," *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on* , vol., no., pp.447-454, 9-11 Feb. 2011.

[16] Suganya, K.; Nagarajan, V.; , "Efficient run-time task allocation in reconfigurable multiprocessor System-on-Chip with Network-on-Chip," *Computer, Communication and Electrical Technology (ICCCET), 2011 International Conference on* , vol., no., pp.12-17, 18-19 March 2011.

[17] Jia Huang; Raabe, A.; Buckl, C.; Knoll, A.; , "A workflow for runtime adaptive task allocation on heterogeneous MPSoCs," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011* , vol., no., pp.1-6, 14-18 March 2011.

[18] Sudharshan K Dhall, ”Approximation algorithms for scheduling Time-critical jobs on Multiprocessor systems,” University of oklahoma, by CRC press,2004.

[19] Shinpei Kato and Nobuyuki Yamasaki, “Real Time Scheduling with Task Splitting on Multiprocessors,” in 8th IEEE International conference on Embedded and Real-Time computing systems and Applications, RTCSA 2007.

[20] Enrico Bini and Giorgio C.Buttazzo, “Measuring the Perfomance of Schedulability Tests,” in Springer science, Real-Time systems, 30, pp.145-147, 2005.