Department of Electrical, Computer, and Software Engineering

Part IV Research Project

Literature Review and

Statement of Research Intent

Project Number: 116

How much slack is in a multiprocessor schedule?

Prabhnandan Singh

Harrison Warahi

Oliver Sinnen

19/04/2021

Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person.

Signature: 

Name: Prabhnandan Singh

ABSTRACT: This paper explores the role of slack in increasing the efficiency of scheduling algorithms and techniques through a literature review. The review methodology is briefly explained, followed by a brief analysis of the literature. We find that multiple scheduling algorithms and techniques have been proposed over the years, utilizing slack to reduce makespan and power consumption. However, not much study has been done on how much slack is present in a multiprocessor schedule and how much of it is reclaimed by different algorithms, which will be the focus of our research project.

1. Literature Review

1.1. Introduction

The performance of modern processors has increased immensely in the past few years. With the increasing performance, the power consumption has also increased. This increase in power consumption raises many new problems like higher costs (electrical and management) [1]. Multiple hardware (for example, power-scalable clusters) and software (for example, slack reclaiming algorithms) solutions have been developed to reduce these problems.

Slack time is the time between tasks when the processor/s are sitting idle. There are different reasons why slack exists in a schedule. It can be present because of precedence constraints between tasks or the difference between the actual execution time(AET) and the estimated worst-case execution time(WCET). In multi-processor scheduling, the tasks can be assigned to different processors for parallel computing to increase the system's performance. However, in a precedence-constrained environment, some tasks cannot start execution until some other task/s that they depend on finish their execution. Furthermore, the inter-task communication overhead delays the tasks' start times and affect the schedule's efficiency [2]. This introduces slack in the schedule. Many researchers have come up with algorithms to reclaim this slack by slowing down some tasks' execution to reduce energy consumption. However, this raises the question of how much slack is present in a schedule and whether it is possible to reclaim all the slack present.

This literature review will look into some of the existing slack reclamation algorithms and help us develop a research intent to further contribute to the research done around the topic. We want to develop a formal definition of slack, discover the amount of slack in different models, and determine trends among the different models. We would also like to analyse different already existing algorithms and see if slack can be used as a performance metric.

1.2. Review Methodology and Analysis

This section describes how the literature search was performed and what was the selection process for the reviewed literature. A brief analysis is then performed on the selected literature.

1.2.1. Review Methodology

Google scholar was mainly used to find different papers, as it provides a wider database. ScienceDirect and Citeseer were also used using the same search terms. The terms that were used to find the papers were: "Slack time", "Slack Reclamation Algorithm", "DVFS slack Reclamation", "DVFS scheduling", "Optimal multiprocessor scheduling", "slack assessment in multiprocessor". Also, "filetype: pdf" was added to the search terms on Google Scholar to only get the results for papers that were available as PDFs. The papers were then selected by reading their abstracts and seeing if that relates to the research topic. We also looked at how often other papers cited a paper, which can be considered an indication of the paper's importance in the research topic. We also looked into the cited by link and used search terms (for example, slack) in only those resulting articles, allowing us to find newer information. For example, [3] is an article we found that was published in 2003, but by looking at the papers that cited this article, we found the article [4], which was published in 2017. Some of the papers were also found under the Related articles page.

We did not restrict the range of years the literature was selected from, as we wanted to look at different kinds of algorithms used to reclaim slack over the years.

1.2.2. Analysis

JabRef was used to keep track of the literature, as it provides an efficient way to collect and organize publications. It also allowed us to check for duplicates in the sources.

Figure 1 shows the different databases the literature was found from.

Figure 1: Distribution of publications over the databases

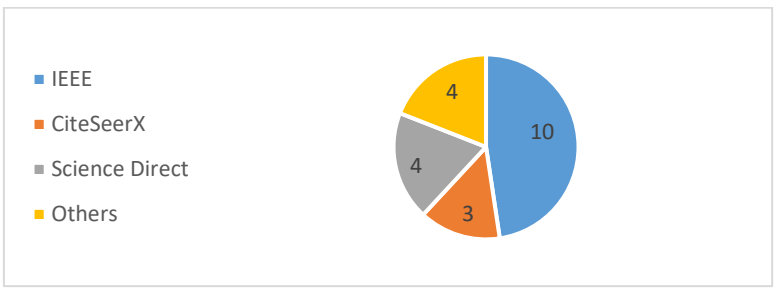
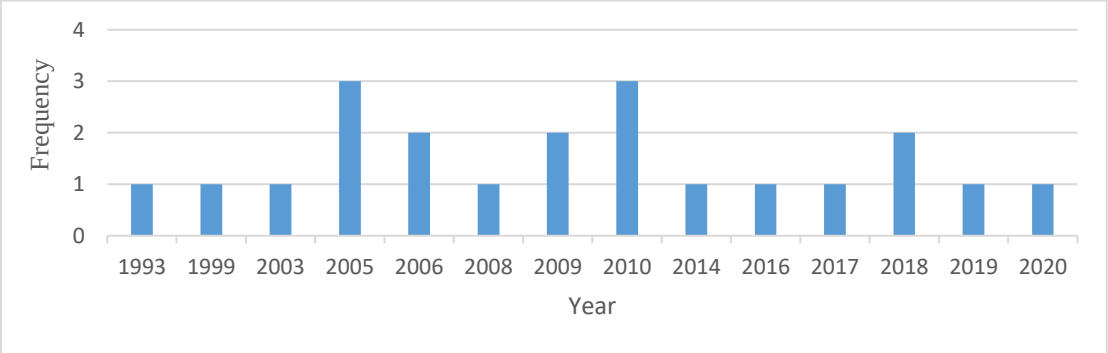


Figure 2 shows the publications distributions by the year. It shows that there was not much interest in the subject before the year 2000, which significantly increased around year 2005 .

Figure 2: Distribution by year



1.3. Classification

There has been a lot of research done around finding efficient and better algorithms that find and reclaim slack in a schedule. These algorithms each target a specific efficiency problem. Therefore, it would be beneficial to classify the algorithms according to the problems they address, giving us insight into what topics have been of higher interest.

1.3.1. Application models

The publications have used different application models, choosing the one better fitting to their problems. Most of the publications either use task models or task graphs (DAGs).

Generally, a precedence-constrained application can be modelled as a DAG, $G(V, E)$, where V is a set of n nodes $V = \{v_0, \dots, v_{n-1}\}$ and a set of directed edges E. The nodes represent the parallel tasks. An edge $e(i,j) \in E$ from task v_i to v_j represents a precedence constraint. The weight on task v_i is denoted by w_i which represents the computation cost of that task. The weight of an edge $e(i,j)$ is denoted as $c_{i,j}$ which represents the communication cost from v_i to v_j. The communication cost between two tasks is only needed if the tasks are assigned to different processors (model taken from [4] and [5]). The publications [1], [6], [4], [5], [2] and [7] all use an application model similar to the one stated above. The makespan is the finish time of the last task in the schedule. Minimizing the makespan and reducing the energy utilization is the primary purpose of modern scheduling algorithms [6]. They all present algorithms that reclaim slack to minimise energy utilization without delaying the makespan (or at least not by much).

[8] and [9] use a task model which consists of a set of n periodic real-time task, represented as $\Gamma = \{\tau_1, \dots \tau_n\}$. A task τ_i is a 3-tuple $\{T_i, D_i, C_i\}$ where T_i is the period of the task, D_i is the relative deadline and C_i is the WCET of the task at

maximum processor speed. Each invocation is called a job and the k^{th} instance of a task τ_i is referred to as $\tau_{i,k}$. [10], [11]and [12] use a similar model but for multicore processors or multiprocessors. [13] and [14] also uses a similar task model, but they schedule sporadic constrained-deadline tasks on DVFS-identical multiprocessor platforms.

1.3.2. Static vs Dynamic scheduling

In static algorithms, the parallel problem's characteristics are already known at compile-time. Therefore, scheduling decisions can be made before execution. In dynamic scheduling, only a few assumptions about the program can be made at compile time. Therefore, scheduling decisions are taken dynamically during run-time [15]. After analysing the algorithms, most of them use dynamic scheduling ([1] [16] [10] [3] [4] [5] [9] [11] [12] [13] [14] [17] [18]), which is self-explaining as more slack can be reclaimed if the task's timing characteristics are being checked dynamically.

1.3.3. Preemptive vs non-preemptive

In preemptive scheduling, the execution of a lower-priority task may get interrupted by a higher-priority task. The unfinished portion of the preempted task gets reallocated according to the preemption policy. In non-preemptive scheduling, the algorithm must let the task scheduled on a processor execute until completion. The algorithms proposed in [16] [9] [11] [13] [14] [17] use a preemptive approach. The algorithms proposed in [1] [10] [3] [4] [8] [12] [5] [6] [19] [18] use a non-preemptive approach. About 63% of the algorithms analysed in this literature review use a non-preemptive scheduling policy.

1.4. Key findings

This section will summarize some of the reviewed literature's key findings, which will help us evaluate the current state of knowledge in the field.

1.4.1. Dynamic Voltage/Frequency Scaling

When task scheduling is concerned, the primary objective is to execute a program fast. However, reducing energy consumption is becoming a significant objective too. One very well-known existing technique to reduce energy consumption is dynamic voltage/frequency scaling (DVFS). DVFS is an efficient power management system that enables processors to dynamically adjust voltage supply levels to reduce power consumption at the expense of clock frequencies [5]. The slack present in a schedule can slow down the execution of some tasks using DVFS, resulting in lower power consumption. Multiple algorithms have been proposed in the reviewed literature that use the DVFS technique. However,

using this technique implies a trade-off between the quality of schedules and energy consumption. [5] proposed an energy-conscious algorithm that explicitly balances the two performance metrics (makespan and energy consumption). [1] proposed an energy reduction algorithm that, using DVFS, selects the lower voltage and frequency uniformly along with the critical path in a task graph. After evaluation, the energy consumption was reduced by 16.8%, with only a minimal performance loss of 0.44% in a simple master-worker program. In a tree-based parallel program, the energy consumption is reduced by 25%.

1.4.2. Slack allocation algorithms

When tasks are scheduled statically, an estimated execution time is used. However, the estimated time in practice can be overestimated or underestimated, meaning some tasks may execute earlier or later than expected during actual execution. [19] presented novel slack allocation algorithms that use slack in the schedule to deal with such situations. For overestimations, extra available slack can be added to future tasks. In case of underestimation, this extra slack can reduce the possibility of missing the deadline. In [16], the authors have proposed a Least slack time rate first algorithm, which overcomes some of the existing scheduling algorithm's (EDF, LRT and LST) limitations in a multiprocessor environment. They have also demonstrated that their algorithm could show optimal possibility. [20] has presented a set of principles for effective slack management in EDF-based systems, which help reduce deadline miss ratio and tardiness.

1.4.3. Procrastination scheduling and leakage power

In addition to dynamic power consumed during the execution of instructions, [10] talks about another source of power consumption: leakage power. Leakage power is consumed as long as there is an electric current in the circuits. The suggested algorithms (Dynamic Repartitioning and Dynamic Core Scaling) in [10] are designed to reduce dynamic power consumption and leakage power consumption. Dynamic Core Scaling deactivates excessive cores by exporting their assigned task to other activated cores, thus reducing leakage power consumption. The evaluation done in [10] shows that Dynamic Core Scaling saves much more power than Dynamic Repartitioning. [8] uses procrastination scheduling and dynamic slowdown in conjunction to reduce the leakage power consumption. In procrastination scheduling, task execution is delayed maximizing the duration of idle intervals, therefore allowing the processors to sleep for longer intervals. The algorithm proposed in [8] distributes the slack between slowdown and procrastination, which instead of using the entire slack for just one, is more energy-efficient. The scheduling technique proposed in [11] (EAPSM) also uses a similar approach, but with more optimization to reduce the overhead from task migration and frequency switch.

EAPSM is integrated with Task processor affinity metric and Processor frequency affinity metric techniques to minimize the overhead effects.

1.4.4. Duplication-based scheduling

When tasks are scheduled on different processors, the inter-communication costs delay the dependent task's execution, introducing slack in the schedule. However, when on the same processor, the communication cost is considered negligible. One way to solve this problem is duplication-based scheduling. In duplication-based scheduling, the inter-processor communication is removed by duplicating the source task of the inter-processor communication to the processor where the dependent task is scheduled. [2] proposes a novel duplication-based MILP (Mixed Integer Linear Programming) formulation. The proposed MILP formulation optimizes the duplication strategy, serializes the execution of task execution on each processor and determine data precedence among different task instances, thus producing an optimal solution. The proposed algorithm is evaluated against other available duplication-based algorithms, and the results demonstrate that the proposed method outperforms the others in several aspects.

1.5. Conclusion

In this literature review, we looked at the research been done in the field of slack reclamation. Much research has been done around the developing algorithms, which, along with reducing the makespan of a schedule, focuses on minimizing energy consumption. The algorithms were then classified, giving us an insight into which types of problems have been popular in the research done around the topic. Task graph and task models have been the most popular application models. Most of the research is done around dynamic scheduling. We also looked at different scheduling approaches and algorithms proposed in the literature, the most popular ones being slack allocation algorithms, DVFS scheduling, duplication-based scheduling, and procrastination scheduling. They have used various performance metrics to evaluate their work like makespan, resource utilization, etc. Slack is used by most researchers to make their techniques more efficient. However, this leads to the question of how much slack is present in a multiprocessor schedule. Also, if specific graph models have more or less slack in general as compared to others. Another potentially important research topic is how much of the slack present in a schedule is reclaimed by the modern algorithms and scheduling techniques and if slack can be used as a metric to evaluate the efficiency of an algorithm. Not much research has been done around this topic. In the reviewed literature, only one paper [21] has evaluated the slack occurrence of scheduling algorithms. However, they only considered three algorithms (Rate Monotonic, Earliest Deadline First and Least Laxity First scheduling).

2. Research Intent

In the literature review, we looked at how various task scheduling algorithms and techniques use slack to increase efficiency. However, some aspects of slack have still not been formally defined. In our research project, we would like to contribute to the knowledge around the topic. Our contribution target is summarized in the following subsections.

2.1.1. How much slack is there in a multiprocessor schedule?

The algorithms we came across while reviewing the literature use slack for various purposes. However, the amount of slack in the schedules is still unknown. Most of the algorithms target different problems and use different types of task models. It is possible that the proposed algorithms only work efficiently for the problems that they are targeting. Therefore, we would like to analyse different graph models to determine if some specific graph models have more or less slack than other graph models. To achieve this, we will first have to define slack formally. By defining slack, we will determine if the times between tasks where the processors are idle, be defined as slack time or idle time. Idle time is the time between tasks that cannot be reclaimed as slack. We would then analyse graph models and determine if there is a trend between the graph models of the proportion of available slack time and idle time.

2.1.2. Slack as a metric

Most of the modern energy-efficient algorithms use slack reclamation. Since slack can be reclaimed to reduce the makespan of a schedule or the system's energy consumption, having more slack available in the schedule should allow more potential to achieve efficiency. However, it is still unclear how much of the slack is reclaimed back in the algorithms. Therefore, we will analyse various slack reclamation algorithms and determine how much slack each of these reclaim. We will have to find and compare the amount of slack before and after the slack reclamation process. Then, we will compare that with other performance metrics to see if there is a trend that could allow us to introduce slack as a metric for algorithm evaluation for future research.

2.1.3. Methodology

For the analysis discussed in *Section 2.1.1.*, we will use task graphs as the application model. Task graphs are the most suitable model for precedence-constrained applications in a multiprocessor environment, as seen in the literature review. In our research project, we will only consider only static slack in different graph models as it can be calculated before the scheduling starts, as opposed to dynamic slack. We will also be using a non-preemptive approach as it more popular among the reviewed papers. We aim to create an algorithm which when input a schedule, outputs the slack time and the

idle time present in the schedule. Using this algorithm, we will analyse any trends among the amount of slack between different graph models. Later, we will be using this algorithm to determine the amount of slack reclaimed by other scheduling algorithms and discover if slack can be established as a performance metric.

3. References

[1] H. Kimura, M. Sato, Y. Hotta, T. Boku and D. Takahashi, "Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster," in *2006 IEEE international conference on cluster computing*, 2006.

[2] Q. Tang, L.-H. Zhu, L. Zhou, J. Xiong and J.-B. Wei, "Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems," *Journal of Parallel and Distributed Computing,* vol. 138, pp. 115-127, 2020.

[3] D. Zhu, R. Melhem and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems,* vol. 14, pp. 686-700, 2003.

[4] Y. Hu, C. Liu, K. Li, X. Chen and K. Li, "Slack allocation algorithm for energy minimization in cluster systems," *Future Generation Computer Systems,* vol. 74, p. 119–131, 2017.

[5] Y. C. Lee and A. Y. Zomaya, "On Effective Slack Reclamation in Task Scheduling for Energy Reduction.," *JIPS,* vol. 5, p. 175–186, 2009.

[6] Z. Shi, E. Jeannot and J. J. Dongarra, "Robust task scheduling in non-deterministic heterogeneous computing systems," in *2006 IEEE International Conference on Cluster Computing*, 2006.

[7] M. Guzek, J. E. Pecero, B. Dorronsoro and P. Bouvry, "Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems," *Applied Soft Computing,* vol. 24, p. 432–446, 2014.

[8] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proceedings. 42nd Design Automation Conference, 2005.*, 2005.

[9] S. Midonnet, D. Masson and R. Lassalle, "Slack-time computation for temporal robustness in embedded systems," *IEEE embedded systems letters,* vol. 2, p. 119–122, 2010.

[10] E. Seo, J. Jeong, S. Park and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE transactions on parallel and distributed systems,* vol. 19, p. 1540–1552, 2008.

[11] P. Ramesh and U. Ramachandraiah, "Energy aware proportionate slack management scheduling for multiprocessor systems," *Procedia computer science,* vol. 133, p. 855–863, 2018.

[12] V. Kannaian and V. Palanisamy, "Energy-efficient scheduling for real-time tasks using dynamic slack reclamation," *Turkish Journal of Electrical Engineering & Computer Sciences,* vol. 27, p. 2746–2754, 2019.

[13] V. Nelis and J. Goossens, "MORA: An Energy-Aware Slack Reclamation Scheme for Scheduling Sporadic Real-Time Tasks upon Multiprocessor Platforms," in *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009.

[14] H. Baek, D. Lim and J. Lee, "Proof and Evaluation of Improved Slack Reclamation for Response Time Analysis of Real-Time Multiprocessor Systems," *IEICE TRANSACTIONS on Information and Systems,* vol. 101, p. 2136–2140, 2018.

[15] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR),* vol. 31, p. 406–471, 1999.

[16] M. Hwang, D. Choi and P. Kim, "Least slack time rate first: New scheduling algorithm for multi-processor environment," in *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, 2010.

[17] R. I. Davis, K. W. Tindell and A. Burns, "Scheduling slack time in fixed priority pre-emptive systems," in *1993 Proceedings Real-Time Systems Symposium*, 1993.

[18] T. A. AlEnawy and H. Aydin, "Energy-constrained scheduling for weakly-hard real-time systems," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, 2005.

[19] J. Kang and S. Ranka, "Dynamic slack allocation algorithms for energy minimization on parallel machines," *Journal of Parallel and Distributed Computing,* vol. 70, p. 417–430, 2010.

[20] C. Lin and S. A. Brandt, "Improving soft real-time performance through better slack reclaiming," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, 2005.

[21] P. Ramesh and U. Ramachandraiah, "Slack Assessment of the Real Time Scheduling Algorithms," 2016.