Partitioned scheduling

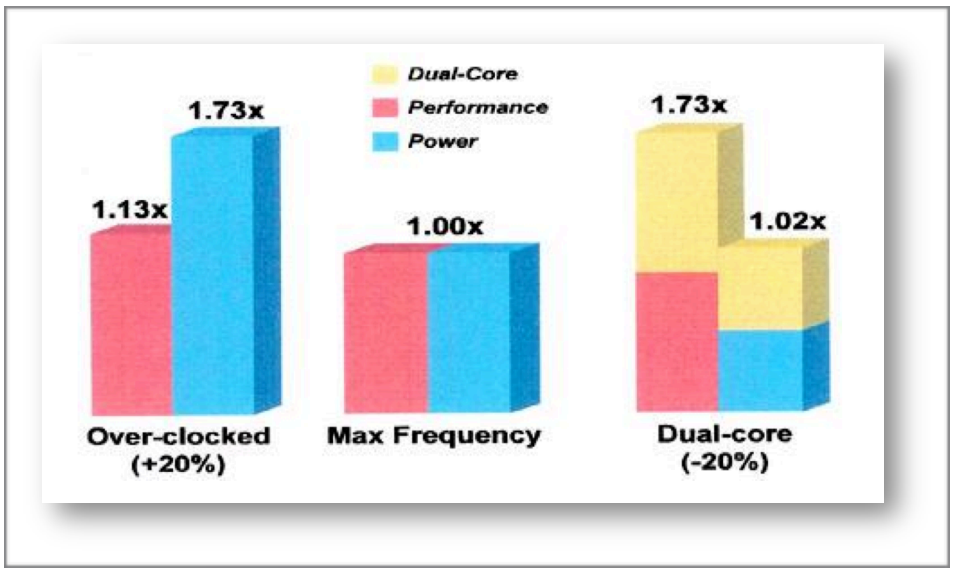
for multiprocessor systems

Cyril Trosset

Outline

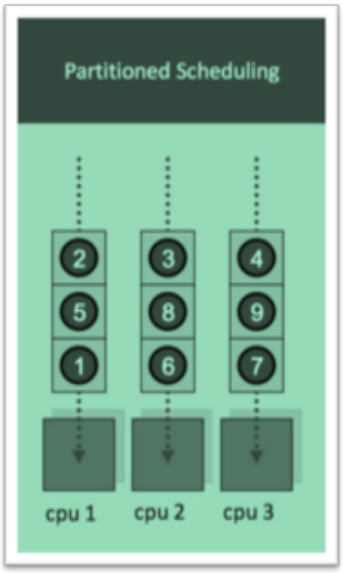
Introduction

In recent years multiprocessor architectures have become mainstream, and multi-core processors are found in products ranging from small portable cell phones to large computer servers.
As you can see on the graph below, the main goal is too increase performances avoiding a huge increase in power consumption.



In parallel, research on real-time systems has mainly focused on traditional single-core processors. Hence, in order for real-time systems to fully leverage on the extra capacity offered by new multi- core processors, new design techniques, scheduling approaches, and real-time analysis methods have to be developed.
In the multi-core and multiprocessor domain there are mainly two scheduling approaches, global and partitioned scheduling. Under global scheduling each task can execute on any processor at any time while under partitioned scheduling tasks are statically allocated to processors and migration of tasks among processors is not allowed. Besides simplicity and efficiency of partitioned scheduling protocols, existing scheduling and synchronization methods developed for single-core processor platforms can more easily be extended to partitioned scheduling. This also simplifies migration of existing systems to multi-cores. An important issue related to partitioned scheduling is distribution of tasks among processors which is a bin-packing problem.

The following schemas resume the concept of partitioned and global scheduling :



This project focuses on partitioned scheduling. More particularly, its goal is to compare and analyze how different bin packing algorithms can affect the schedulability on multiprocessors and more generally, how global performances are affected.
My study uses rate monotonic (RM) as the single queue scheduling algorithm. The different bin packing technics compared in this project are : First Fit (FF), Next Fit (NF), Best Fit (BF).

As a consequence, the partitioned scheduling algorithms compared are :
- RMFF (Rate Monotonic First Fit)
- RMNF (Rate Monotonic Next Fit)
- RMBF (Rate Monotonic Best Fit)

Motivations

Nowadays there are different algorithms for multiprocessors real time scheduling. I found out that there is no optimal algorithm but some figures are given. As partitioned scheduling takes advantages of single core studies, I am wondering how different bin packing technics with a fixed single processor algorithm could affect the performance of the systems. And then, how to choose one algorithm rather than another according to the needs. The main goal of my project is to answer these questions by being able to highlight the pros and cons of each solutions.

Bin Packing technics

The bin packing problem raises the following question: given a finite collection of n weights w_1, w_2, w_3, ... , wn, and a collection of identical bins with capacity C (which exceeds the largest of the weights), what is the minimum number k of bins into which the weights can be placed without exceeding the bin capacity C? Stripped of the mathematical formulation, we want to know how few bins are needed to store a collection of items. This problem, known as the 1-dimensional bin packing problem, is one of many mathematical packing problems which are of both theoretical and applied interest in mathematics. Moreover, it is a NP-Hard problem and that is why so many algorithms are proposed.

First Fit

First fit is probably the most common bin packing algorithm. It works as followed :

1. Order tasks in non-decreasing order
2. Starting from first processor, assign task as soon as the task fits
3. Restart with next task

Next Fit

Next Fit is the second technic I have implemented. It works as follow :

1. Order tasks in non-decreasing order, current processor is the first one at the beginning.
2. Assign the task to the current processor if it fits, it not assign to the next one, and current processor becomes the next one
3. Restart with next task

This algorithm seems to be less optimal as, when a task does not fit into one processor, no other task will be tested with this processor.

Best Fit

Best Fit uses a different approach. It works as follow :

1. Order tasks in non-decreasing order
2. Test all processors (calculate the space left after assigning the task)
3. Choose the processor with the less space left
4. Restart with the next task

This algorithm looks for the processor for which the task fills as much as possible.

Survey

Implementation

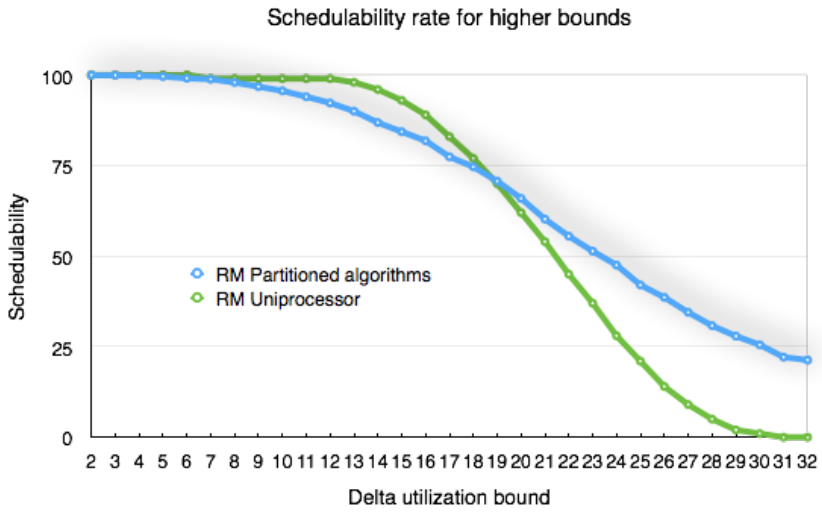
My implementation consists of a C++ simulation that can simulate a schedule on N processors using RMFF, RMNF, RMBF with the same task set.
The tasks are generated randomly using UUnifast.

Delta in utilization bound

To determine if a task can fit into a processor we use the utilization bound formula :

$$U = \sum_{i=1}^{n} C_i/T_i \leq n(\sqrt[n]{2} - 1)$$

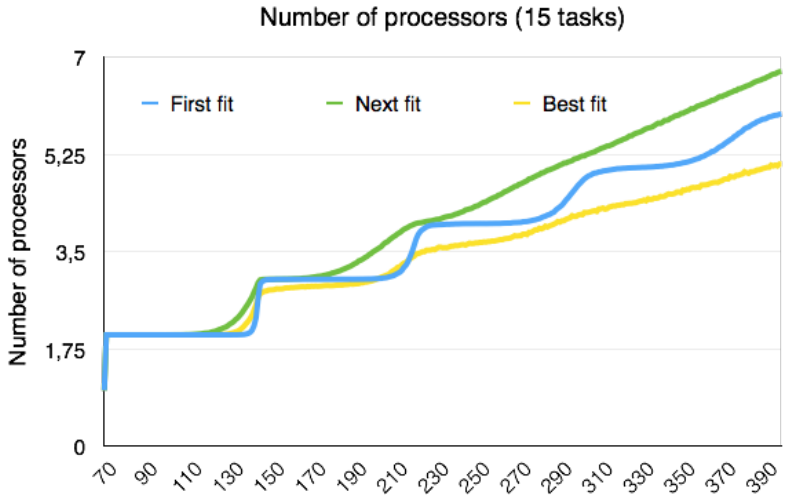
My first experiment consists of increasing this bound by increasing the value and observe whether or not a task-set is schedulable.
The following graph presents the results compared to RM on uniprocessor schedulability



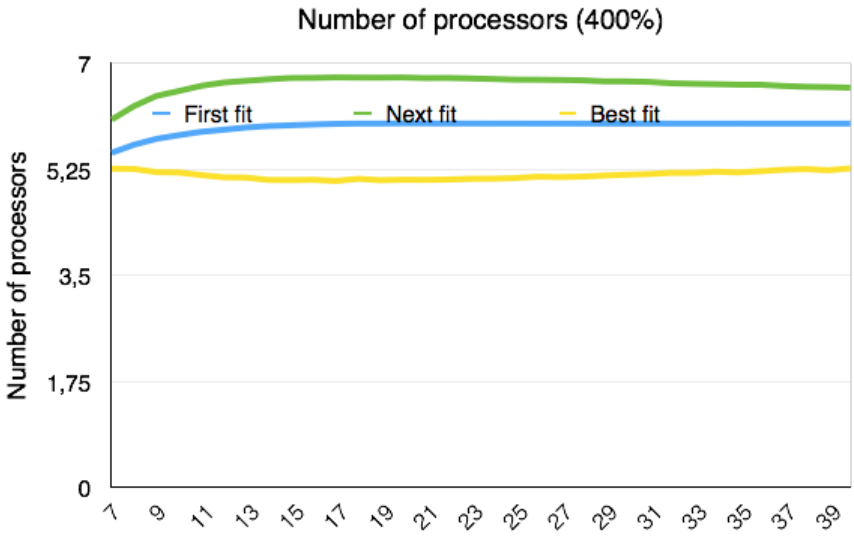
The schedulability decreases rapidly at the beginning for the following reason : if only one processor is not schedulable, the whole set is not schedulable. However the schedulability is better at the end because some big task can be scheduled on one processor alone meaning that the max utilization for this processor is 1.
We can conclude that if you can afford some missed deadlines and want to fill the processor more than the utilization bound, it can be a good idea to apply this delta even on multiprocessor scheduling.

Processors requirements

I have started the comparaison between the three algorithms by determining how many processors are required to schedule a same task set with FF, NF and BF.

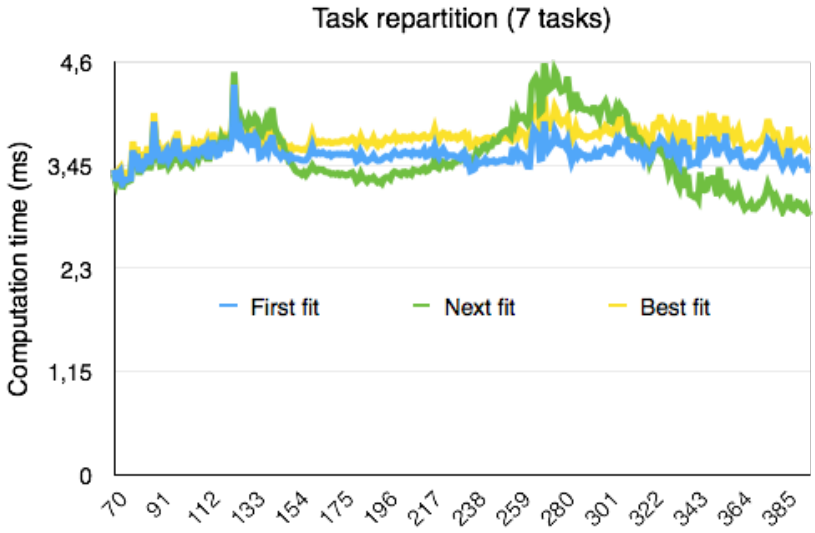


As you can see on the graph, Best Fit is the most optimal algorithm for the repartition as it requires fewer processors than others. The next one is First Fit, there are steps each multiple of 70. This can be explained by the fact that 70 is the utilization bound with more than 4 tasks (approximately), so first fist is able to schedule correctly before the utilization bound but when the total utilization of the task set is beyond this value, it rapidly needs a new processor. Finally Next Fit is the worst algorithm and requires always the same or more processors than the two previous ones.
However when the total utilization is fixed, the number of processors does not evolve. We can affirm that the number of tasks has very little effect on the number of processors required.



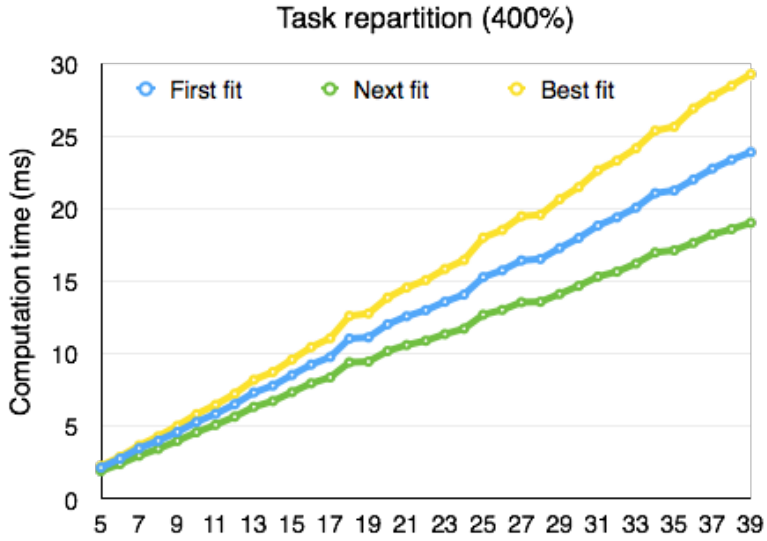
Computation time

For the computation time, the first experiment consists of a fixed number of tasks (seven) and increasing utilization.



We can see that in this case the three algorithms perform equally by having a constant computation time of an average of 3.45 ms.

However, when the total utilization is fixed (to 400%) and the number of tasks is increasing, the results are interesting.



I found out that Best Fit has the worst computation time then First Fit, and finally the fastest is Next Fit.

Conclusion

To conclude my project, I can confidently say that I was able to draw the tendencies of how the bin packing part affects the performances of a system on partitioned scheduling. As I used Rate Monotonic as the single core scheduling algorithm, the schedulability guarantee is still bounded by the Rate Monotonic bound per core (≈ 0.69). However we have seen that increasing slightly this bound can result in a low decrease of schedulability. Moreover the different tendencies revealed about the different bin packing technics are the following. Generally, only the total utilization affects the number of processors required, not the number of tasks, whereas the computation time is affected only by the number of tasks, not the total utilization. Next Fit algorithm appears to be the fastest one, but results in lower schedulability. First Fit is slightly better in term of schedulability but requires more time to distribute the tasks among the different processors. Finally, Best Fit is the best algorithm as it fills the processors in the more optimal way, however it is the longest one at computing the repartition.

Now it could be interesting to compare more bin packings algorithms and then compare different single core algorithms associated to all these bin packing algorithms. Then most of the partitioned scheduling algorithms would be compared and we would be able to choose which algorithm best fits our needs.