

Real-Time Operative Systems

Module: Multiprocessor Scheduling, V1.2

Paulo Pedreiras

DETI/UA/IT

November 28, 2022

Outline

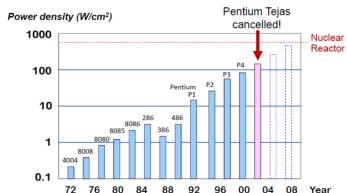
- 1 Introduction
- 2 Definitions, Assumptions and Scheduling Model
- 3 Scheduling for Multicore Platforms
- 4 Task allocation
- 5 Bibliography

- 1 Introduction
- 2 Definitions, Assumptions and Scheduling Model
- 3 Scheduling for Multicore Platforms
- 4 Task allocation
- 5 Bibliography

Why multicore?

Multicore systems are becoming increasingly popular for developing RT systems. **Why?**

- For years processing power increases arose (mainly) from using higher clock frequencies
- Higher clock frequencies and smaller transistors lead to higher dynamic and static power consumption
- Chip temperature eventually reached levels beyond the capability of cooling systems



Why multicore?

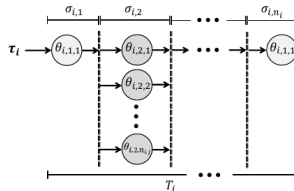
... and the demand for processing power in real-time applications never ceased to increase ...



Problems and Challenges

But ...

- Moving/developing applications to multicore platforms is not simple nor straightforward
- Simple use of sequential languages hides the **intrinsic concurrency** that must be exploited to allow benefiting from the hardware redundancy
- Programmers have to take approaches that allow to split the code in segments that can be executed in parallel, on different cores (special languages, annotations, ...)



Problems and Challenges

Some problems and challenges include:

- How to split the code into segments/jobs that can be executed in parallel?
- How to allocate code segments to different cores?
- How to assess the schedulability on multicore platforms?
- How to handle dependencies?
- How to cope with the impact of shared resources on the WCET
- etc.

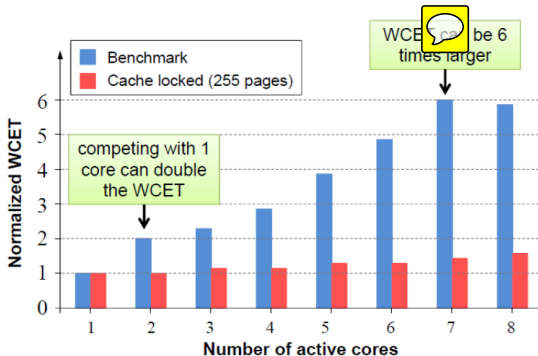
Is it that bad?

The WCET problem

- WCET is fundamental for RT analysis
- Existing RT analysis assumes that the **WCET of a task is constant** when it is executed alone or together with other tasks
- While this assumption is reasonable for single-core chips, it is **NOT true for multicore chips** !

The WCET problem

- Example: **Impact of shared resources on the WCET** of code on an 8-core platform, by Lockheed Martin Space Systems
- Shared resources include main memory, memory bus, cache, etc.



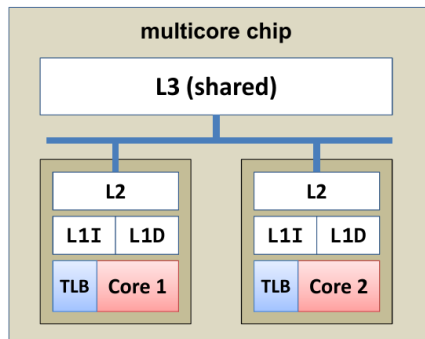
Example: impact on WCET

Reasons for the impact on the WCET

- In a **single core system**, concurrent tasks are **sequentially executed** on the processor.
 - Access to physical resources is implicitly serialized. E.g., two tasks can never cause a contention for a simultaneous memory access
 -
- In a **multicore platform**, different tasks can **run simultaneously** on different cores
 - several conflicts can arise while accessing physical resources
 -
- Important issue, as existing RT analysis assumes that WCET is constant and known!

Example: impact on WCET

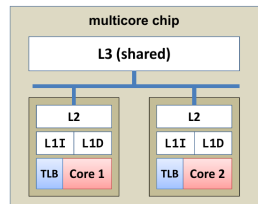
Cache in multicore systems



- L3 cache is typically shared by all cores \Rightarrow **cache conflicts**

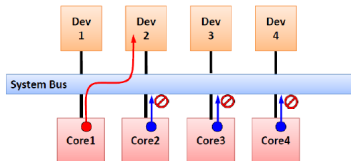
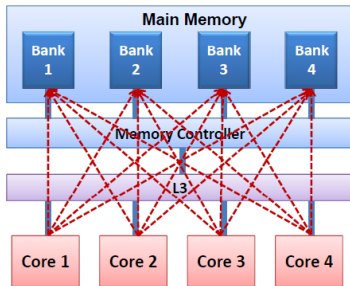
Example: impact on WCET

- In multicore systems, **L1 and L2** caches have the same problem seen in single-core systems
- **L3 cache lines can also be evicted by applications running on different cores**
- Possible approaches to attenuate the impact include e.g. partition the last level cache to simulate the cache architecture of a single-core chip. But the size of each partition becomes small...



Other sources of WCET growth

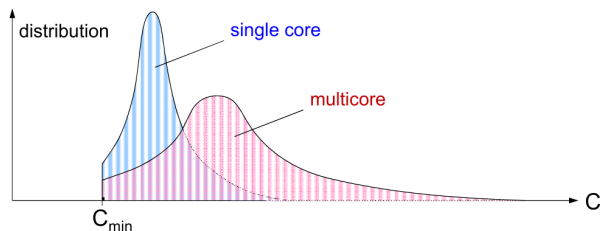
Similar situation with access to main memory, I/O devices, etc.



Note

Despite largely out of the control of the programmer, these issues have a significant impact on the scheduling strategies, as we will see!

Impact on WCET distribution



Note

In summary, the WCET uncertainty get much worse in Multicore systems!

- 1 Introduction
- 2 Definitions, Assumptions and Scheduling Model
- 3 Scheduling for Multicore Platforms
- 4 Task allocation
- 5 Bibliography

Definitions - Processor types

Multiprocessor types usually considered

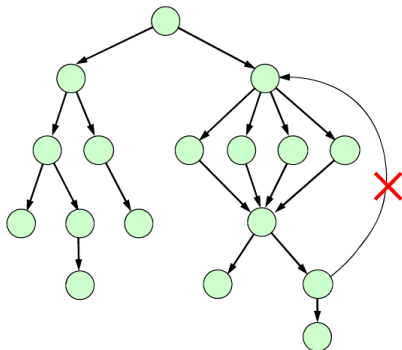
Identical Processors are of the same type and have the same speed.
Each task has the same WCET on each processor.

Uniform Processors are of the same type but may have different speeds. Task WCETs are smaller on faster processors.

Heterogeneous Processors can be of different type. The WCET of a task depends on the processor type and the task itself.

Task Models - DAG

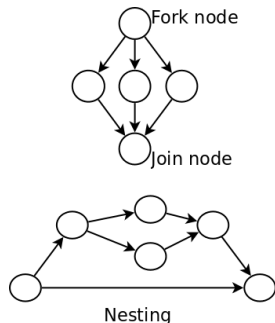
- Representing parallel code requires more complex structures than for sequential tasks.
- A Directed Acyclic Graphs (DAG) is a graph in which links have a direction and there are no cycles



In a DAG this connection is forbidden

Fork-Join Model Task Model

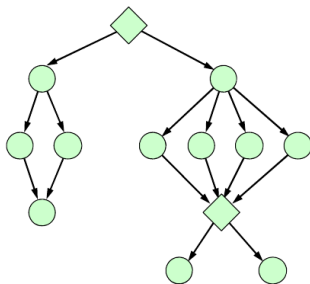
- Special type of Directed Acyclic Graph
- Nodes represent processing
- After a **Fork** node all successors have to be executed. The order is not relevant (nor defined).
- **Join** nodes only execute after the completion of all its predecessors
- Nesting is allowed



And-Or Graphs

The most general graph representation

- OR nodes represent conditional statements (◇)
- AND nodes represent parallel computations (○)

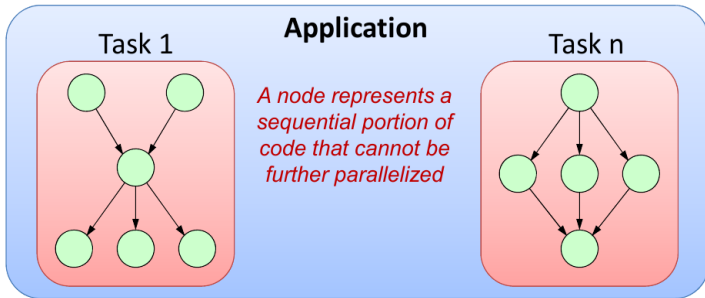


Note

There are other (less common) graph types ...


Application Model

An application can be modeled as a set of tasks, each described by a task graph.



Assumptions

Task and system characteristics

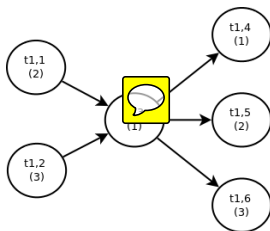
- Arrival pattern
 - Periodic or Sporadic (period or mit is T)
- Preemption
 - Allowed, but eventually with critical sections
- Task migration allowed?
 - May or may not. We will see ...
- Task parameters
 - $\tau_i = \{\{c_{i,1}, c_{i,2}, \dots, c_{i,m}\}, D_i, T_i\}$ 
 - c_i : WCET of a segment
 - m : number of segments of τ_i



Example

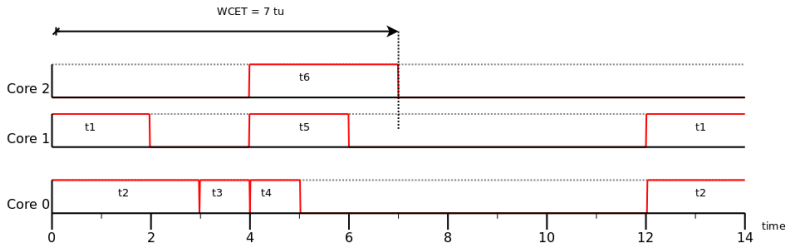
Consider the following example:

- Compute the minimum WCET of task τ_1 in isolation, considering the intra-task (segment) precedence relations.
- Number of cores is not restricted
- Task attributes
 - $\tau_1 = \{\{2, 3\}, 2, 3\}, 8, 12\}$
 - Dependencies: $\tau_{1,3} \leftarrow \{\tau_{1,1}, \tau_{1,2}\}$ and $\{\tau_{1,4}, \tau_{1,5}, \tau_{1,6}\} \leftarrow \tau_{1,3}$



Example (cont.)

- WCET = 7 tu for a number of cores greater or equal than 3.



Note

More processors are not always helpful ...

Additional definitions and observations

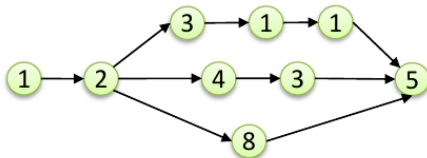
- Sequential Computation time: $C_i^S = \sum_{j=1}^{m_i} c_{i,j}$
- CPU utilization of a task: $U_i = \frac{C_i^S}{T_i}$

Immediate results:

- If $C_i^S \leq D_i$ the task is schedulable in a single core
- If $U_i > K$, where K is the number of cores, then the task **is not schedulable**

Critical path

Critical path length C_i^P length of the longest path in the graph



What is the critical path length?



Note

If follows immediately that if for any task $C_i^P > D_i$ then the application is not feasible in any number of cores

- 1 Introduction
- 2 Definitions, Assumptions and Scheduling Model
- 3 Scheduling for Multicore Platforms**
- 4 Task allocation
- 5 Bibliography

Assumptions

- Identical Multicore Systems (all cores are equal)
- WCET (upper bound) is known for any scenario
- Periodic or sporadic tasks

Main approaches:

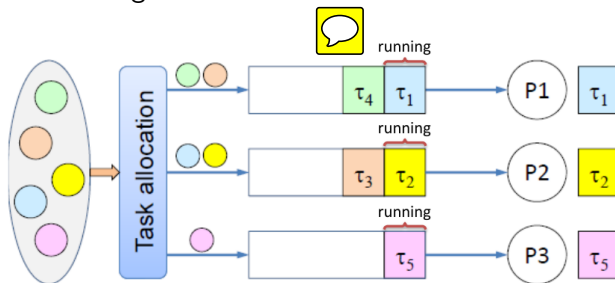
Partitioned scheduling Tasks are allocated a priori to a given core - individual queues

Global scheduling Tasks are allocated to cores at runtime - single queue

Partitioned scheduling



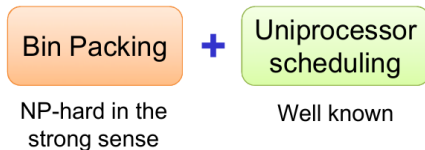
- Each processor has an independent ready queue
- The processor for each task is determined a priori
- Tasks cannot migrate



Partitioned scheduling

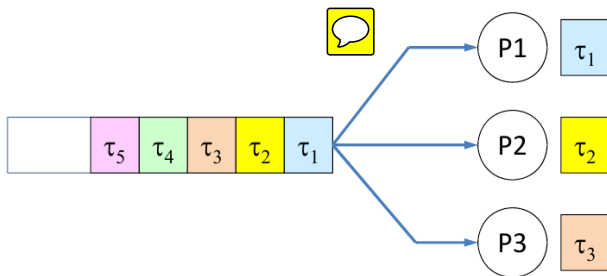
The scheduling problem reduces to:

- Bin-packing problem, for allocating tasks to processors
 - NP-hard in the strong sense.
 - Several heuristics: FirstFit, NextFit, BestFit, etc
- Uniprocessor scheduling problem (already studied and well known)



Global scheduling

- The system manages a **single queue** of ready tasks
- The processor is determined at run time
- During execution a **task can migrate** to another processor

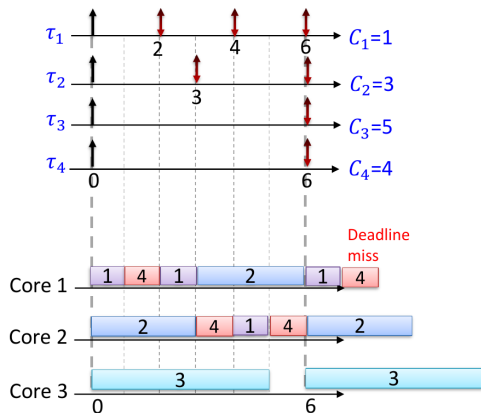


Global scheduling - Example

- Consider a Global Scheduler with RM policy and the task set below.
- Is the system schedulable?



τ_i	C_i	T_i
1	1	2
2	3	3
3	5	6
4	4	6

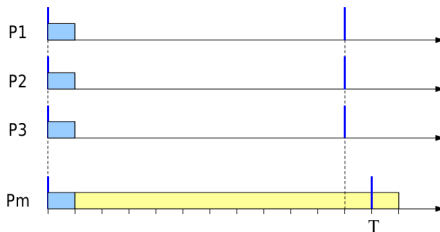


Global Scheduling Evaluation

Dhall's effect: Global RM and Global EDF produce unfeasible schedules for task set utilizations arbitrary close to 1.

- m processors and $m+1$ tasks

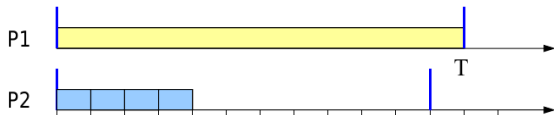
τ_i	C_i	T_i	U_i
τ_1	1	$T-1$	ϵ
τ_2	1	$T-1$	ϵ
...
τ_m	1	$T-1$	ϵ
τ_{m+1}	T	T	1



Global Scheduling Evaluation

But Partitioned Scheduling allows to schedule such system with just 2 processors!

τ_i	C_i	T_i	U_i
τ_1	1	$T-1$	ϵ
τ_2	1	$T-1$	ϵ
...
τ_m	1	$T-1$	ϵ
τ_{m+1}	1	T	1



Note

There are examples of task sets that one of the methods can schedule while the other cannot!

Global vs Partitioned Scheduling

Global

- + Automatic load balancing
- + Lower avg. response time
- + Easier re-scheduling
- + More efficient reclaiming and overload management
- + Lower number of preemptions
 - Migration costs
 - Inter-core synchronization
 - Loss of cache affinity
 - Weak scheduling framework

Partitioned

- + Supported by automotive industry (e.g., AUTOSAR)
- + No migrations
- + Isolation between cores
- + Mature scheduling framework
- + Low scheduling overhead (no need to access a global ready queue)
 - Cannot exploit unused capacity
 - Rescheduling not convenient
 - NP-hard allocation problem

Hybrid approaches

- Task/job migration can be costly
 - context must be moved, impact on cache, etc.
- There are other types of partitioning that consider **restrictions on task migration**

Job migration Tasks are allowed to migrate, but only at jobs boundaries.

Semi-partitioned scheduling Some tasks are statically allocated to processors, others are split into chunks (subtasks) that are allocated to different processors.

Clustered scheduling A task can only migrate within a predefined subset of processors (cluster).

- 1 Introduction
- 2 Definitions, Assumptions and Scheduling Model
- 3 Scheduling for Multicore Platforms
- 4 Task allocation**
- 5 Bibliography

Task allocation

How to partition tasks?

- By information sharing
- By functionality
- Minimizing resources: number of cores, frequency, energy, etc.
- and many others (e.g fault-tolerance)

Note:

These approaches do not aim at schedulability / real-time performance

Task allocation problem

Partitioning problem (bin packing problem):

Given a set of tasks $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$, and a multiprocessor platform with m processors, find an assignment from tasks to processors such that each task is assigned to one and only one processor

Solutions:


- Select a fitness criteria
 - Example: task utilization $U_i = \frac{C_i}{T_i}$ or task density $\alpha_i = \frac{T_i}{D_i}$
- Decide how to sort the tasks (decreasing, increasing, random)
- Decide what is the fitness evaluation method (assignment rejection rule)
- Use a fitting policy to assign tasks to processors:
 - FirstFit, BestFit, WorstFit, NextFit (there are others)


Bin packing heuristics - definitions

The Bin Packing problem

Pack n objects of different size a_1, a_2, \dots, a_n into the minimum number of bins (containers) of fixed capacity c .

Definitions

M_A  Number of bins used by an algorithm A

M_0 minimum number of bins used by the optimal algorithm 


M_{lb} (Lower bound) Number of bins required for sure by any algorithm

M_{ub} (Upper bound) Number of bins that cannot be exceeded for sure by any algorithm

Bin packing heuristics - definitions

Simple and immediate bounds

Given a set of n items of volume $V = \sum_i^n a_i$

- No algorithm can use less than $M_{lb} = \lceil \frac{V}{c} \rceil$ 
- No algorithm can use more than $M_{ub} = \lceil \frac{2 \cdot V}{c} \rceil$

- “ c ” is the capacity of each bin
- Optimality implies clairvoyance for online sequences \rightarrow heuristics

Bin packing heuristics

Partitioning heuristics (some examples)



First fit (FF) Place each item in the first bin that can contain it.

Best fit (BF) Place each item in the bin with the smallest empty space.

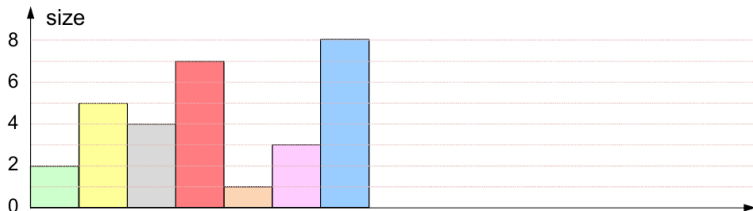
Worst fit (WF) Place each item in the used bin with the largest empty space.

Next fit (NF) Place each item in the same bin as the last item.



Bin packing heuristics

First fit : Place each item in the first bin that can contain it.

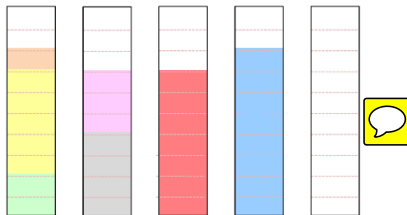


$V = 30$

$c = 10$

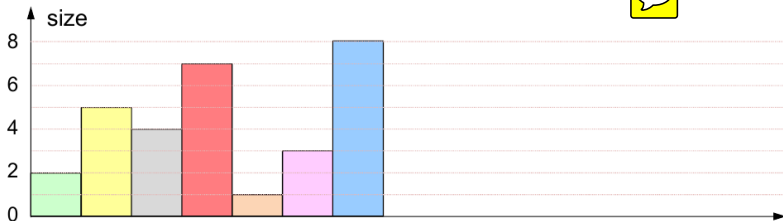
$M_0 = 3$

$M_{FF} = 4$



Bin packing heuristics

Best fit : Place each item in the bin with the smallest empty space.

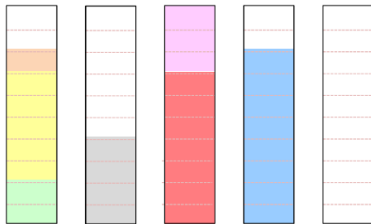


$$V = 30$$

$$c = 10$$

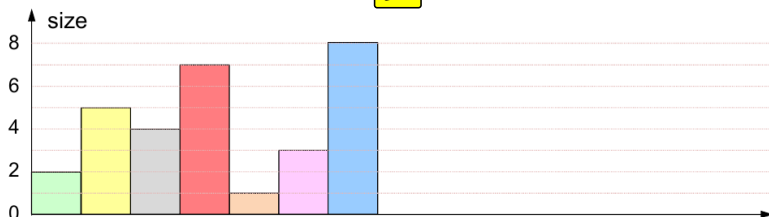
$$M_0 = 3$$

$$M_{BF} = 4$$



Bin packing heuristics

Worst fit : Places each item in the used bin with the largest empty space, otherwise start a new bin.

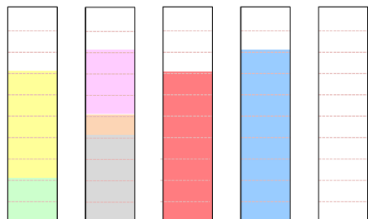


$$V = 30$$

$$c = 10$$

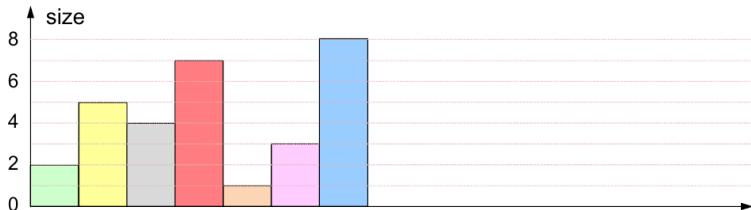
$$M_0 = 3$$

$$M_{WF} = 4$$



Bin packing heuristics

Next fit : Place each item in the same bin as the last item. If it does not fit, start a new bin.

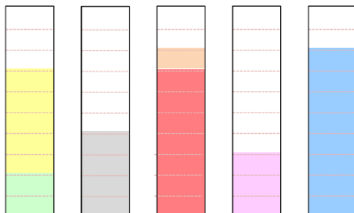


$$V = 30$$

$$c = 10$$

$$M_0 = 3$$

$$M_{NF} = 5$$



Bin packing heuristics

The **performance** of each algorithm strongly **depends on the input sequence** .

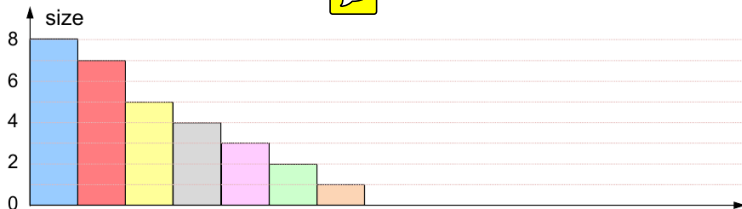


However:

- NF** has a poor performance since it does not exploit the empty space in the previous bins
- FF** improves the performance by exploiting the empty space available in all the used bins.
- BF** tends to fill the used bins as much as possible.
- WF** tends to balance the load among the used bins.

First Fit Decreasing

In this case sorting tasks by decreasing execution times would allow to minimize the number of bins

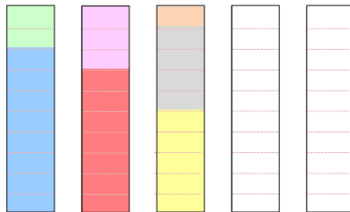


$$V = 30$$

$$c = 10$$

$$M_0 = 3$$

$$M_{FFD} = 3$$



Some theoretical results

Theoretical results

- Any online algorithm uses at least $4/3$ times the optimal number of bins:

$$M_{on} \geq \frac{4}{3} M_o$$

- NF and WF never use more than $2 \cdot M_0$ bins
- FF and BF never use more than $(1.7 \cdot M_0 + 1)$ bins
- FFD never uses more than $(\frac{4}{3} M_0 + 1)$ bins
- FFD never uses more than $(\frac{11}{9} M_0 + 4)$ bins

Some utilization bounds

- Any task set with total utilization $U \leq \frac{m+1}{2}$ is schedulable in a multiprocessor made up of m processors using FF allocation and EDF scheduling on each processor. (Lopez-Diaz-Garcia)
- Any task set with total utilization $U \leq m(2^{\frac{1}{2}} - 1)$ is schedulable in a multiprocessor made up of m processors using FF allocation and RM scheduling on each processor. (Oh and Baker)
- When each task has utilization $u_i \leq \frac{m}{3m-2}$, the task set is feasible by global RM scheduling if $U \leq \frac{m^2}{3m-2}$. (Andersson, Baruah, Jonsson)

- 1 Introduction
- 2 Definitions, Assumptions and Scheduling Model
- 3 Scheduling for Multicore Platforms
- 4 Task allocation
- 5 Bibliography**

Bibliography

Book

- Multiprocessor Scheduling for Real-Time Systems (Embedded Systems) 2015 Edition, Sanjoy Baruah, Marko Bertogna, Giorgio Buttazzo
Available at <https://www.springer.com/gp/book/9783319086958>

Papers

- A survey of hard real-time scheduling for multiprocessor systems, By Robert I. Davis and Alan Burns from University of York, U.K.
Available at <https://dl.acm.org/citation.cfm?id=1978814>

Slides

- Giorgio Buttazzo
On multiprocessor systems (part1):
<http://retis.sssup.it/~giorgio/slides/cbsd/mc1-intro-6p.pdf>
On multiprocessor systems (part2):
<http://retis.sssup.it/~giorgio/slides/cbsd/mc2-sched-6p.pdf>
Note: These slides have been largely adapted from Buttazzo's slides