Supporting Real-Time Applications on Multi-core Platforms

Giorgio Buttazzo

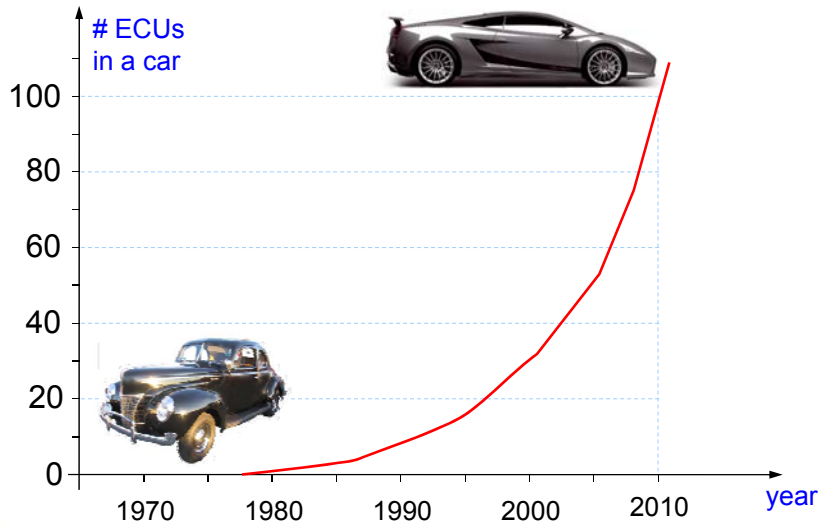
Retis
Real-Time System Laboratory

Scuola Superiore Sant'Anna, Pisa

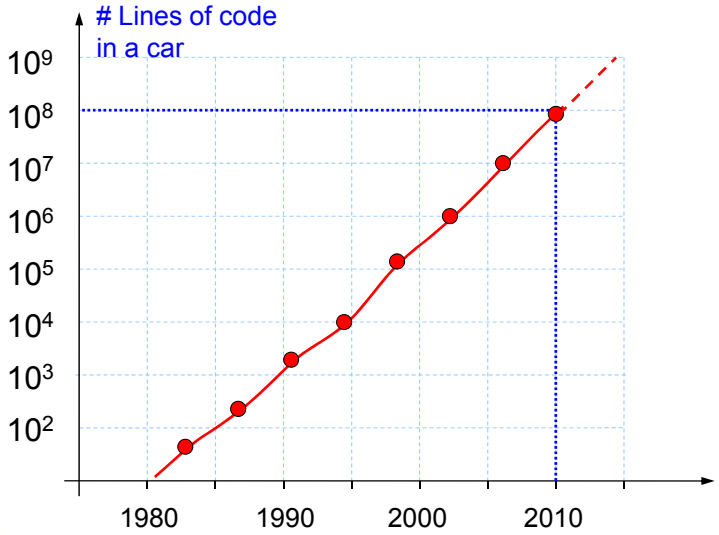
Evolution of Embedded Systems

The complexity of embedded systems has grown exponentially in several application domains:

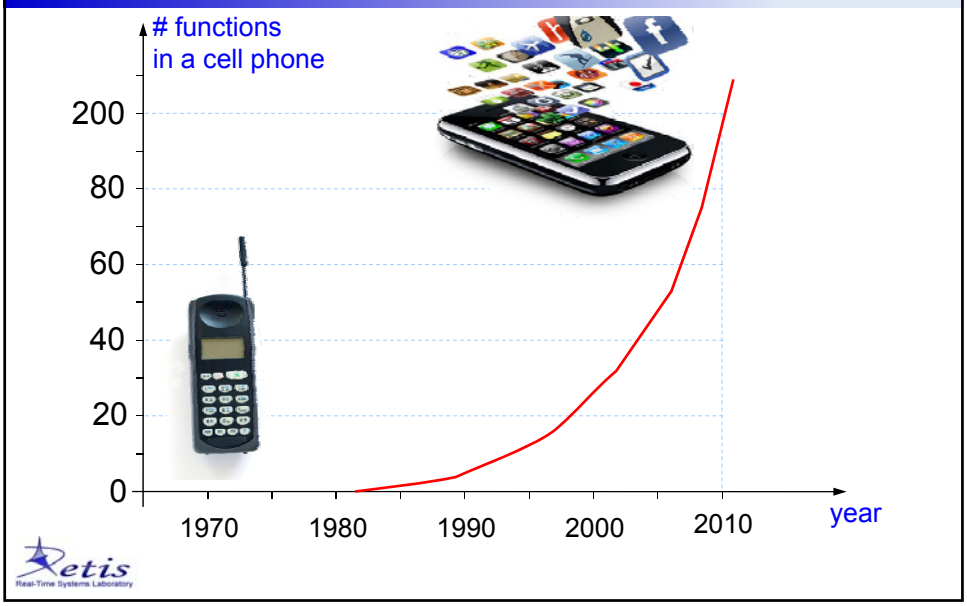
ECU growth in a car



Software evolution in a car

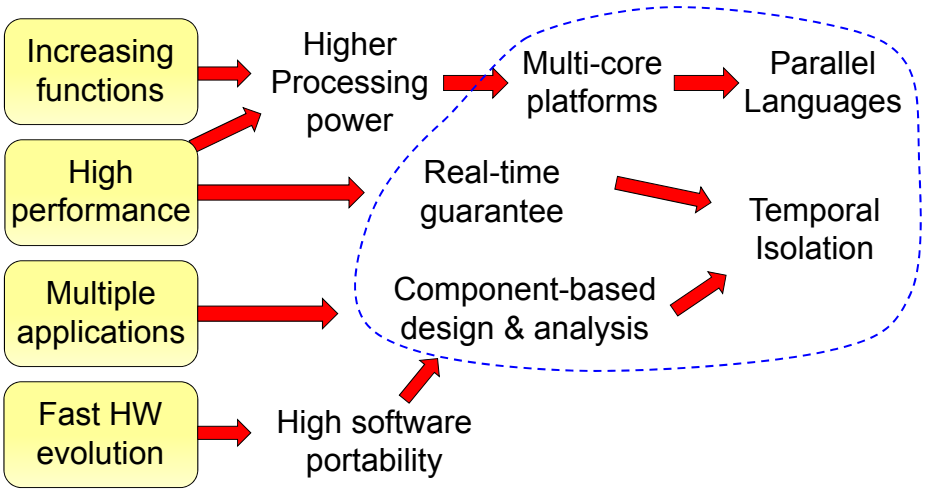


Functions in a cell phone



Cell phone applications

Coping with increasing requirements



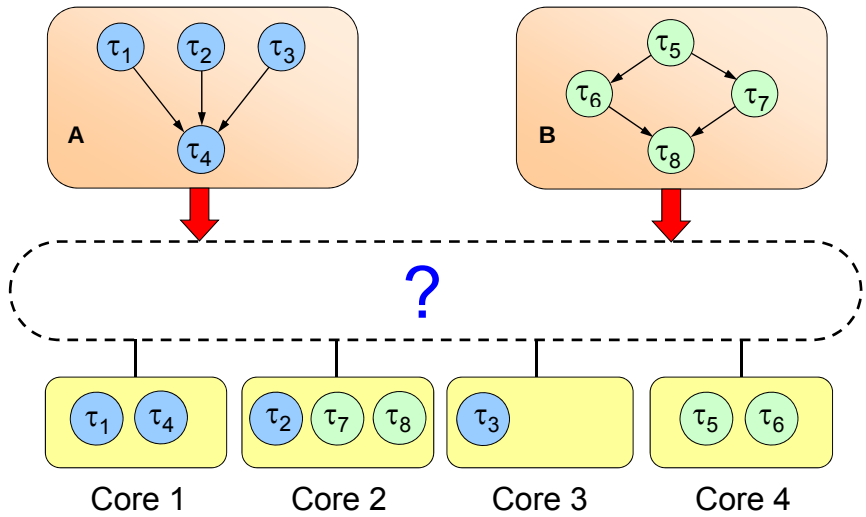
Multiple Objectives

- Exploit parallelism to improve performance
- Execute multiple applications in isolation
- Simplify portability by a proper abstraction layer
- Provide offline guarantee for real-time applications
- Optimize resource usage

This talk

- Illustrates a general framework to address such multiple objectives;
- Presents a number of consolidated methodologies for handling specific problems;
- States a number of open problems for further research.

System model



Outline

- How to express parallelism
- How to achieve temporal isolation
- How to perform real-time guarantees
- How to achieve portability by a proper abstraction (the virtual multiprocessor)
- How to optimize resource usage

Exploiting parallelism

- As we are entering the multicore era, sequential languages (as C/C++) are no longer the most appropriate way to specify programs targeted to run on parallel platforms.
- In fact, a sequential language hides the intrinsic concurrency that must be exploited to improve the performance of the system.

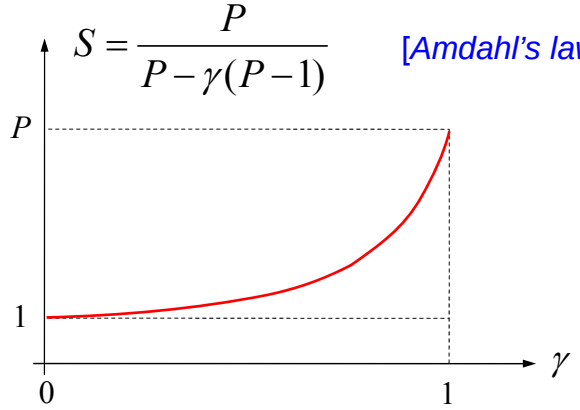
Speed-up factor

γ fraction of parallel code

$S(P)$ speed-up factor on P processors

$$S = \frac{P}{P - \gamma(P-1)}$$

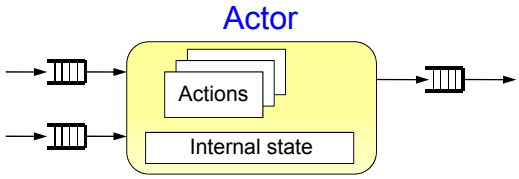
[*Amdahl's law*]



Expressing parallelism

Parallelism can be expressed by using a suitable dataflow language, like CAL [UC Berkeley, 2003].

- It describes algorithms through a set of modular components (actors), communicating through I/O ports:



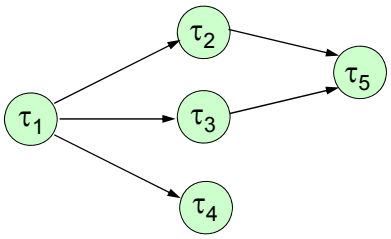
- Actions read input tokens, modify the internal state, and produce output tokens.

Application model

- An application can be modeled as a task graph with precedence relations:

Task τ_i

A sequential portion of code that cannot be further parallelized



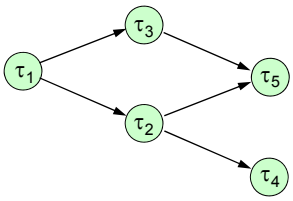
A task graph specifies the maximum level of parallelism

Assumptions and parameters

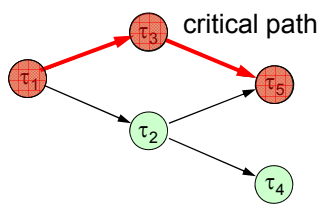
- Each application is sporadic:
  - Minimum interarrival time (period) T
  - relative deadline D
- Tasks are fully preemptive
- Task migration is not allowed

Application parameters:

$\{C_1, C_2, C_3, C_4, C_5\}$, D, T

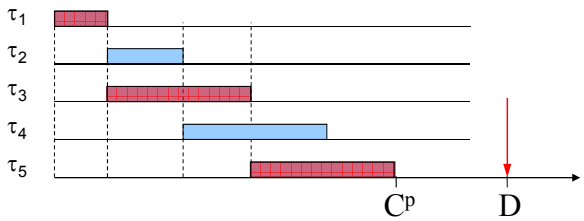


Application model



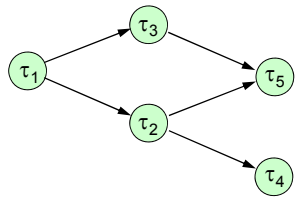
Sequential Computation time $C^s = \sum C_i$

Parallel Computation time C^p = length of a critical path



$$\begin{cases} (D < C^p) \Rightarrow \text{A is not feasible in any number of cores} \\ (D \geq C^s) \Rightarrow \text{A is feasible on a single core} \end{cases}$$

Application model



Sequential Computation time $C^s = \sum C_i$

Parallel Computation time C^p = length of a critical path

$$U = \frac{C^s}{D}$$ required bandwidth

$$\left\lceil \frac{C^s}{D} \right\rceil \leq \text{Number of required cores} \leq \left\lceil 2\frac{C^s}{D} \right\rceil$$

Outline

- How to express parallelism
- How to achieve temporal isolation
- How to perform real-time guarantees
- How to achieve portability by a proper abstraction (the virtual multiprocessor)
- How to optimize resource usage

Achieving Temporal Isolation

Temporal Isolation

Property of a multi-application system in which the performance of an application does not depend on the execution of the others.

The performance of an application only depends on:

- Its own computational demand;
- The amount of allocated resources.

→ Resource Reservation

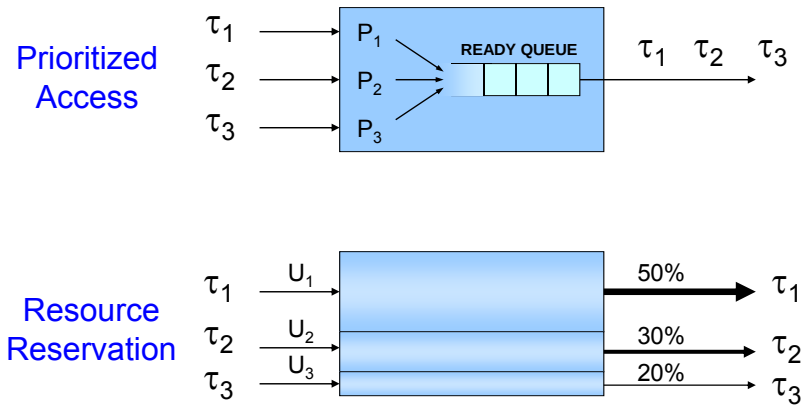
Achieving Temporal Isolation

An isolated application executes as it were executing alone on a slower dedicated processor of speed s equal to the reserved fraction.

Advantages

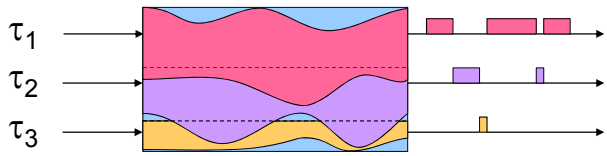
- Predictability: A misbehavior of an application does not affect the others.
- Modular analysis: RT constraints can be verified independently of the knowledge of other applications.

Priorities vs. Resource Reservations

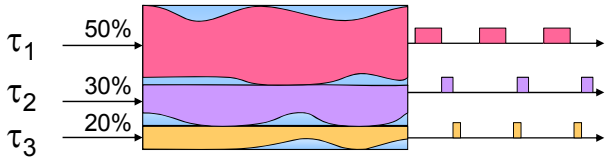


Priorities vs. Resource Reservations

Prioritized Access

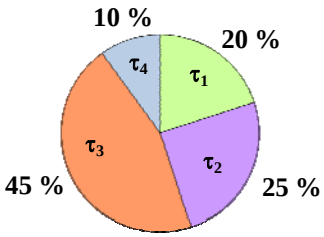


Resource Reservation



Resource Reservation

Resource partition

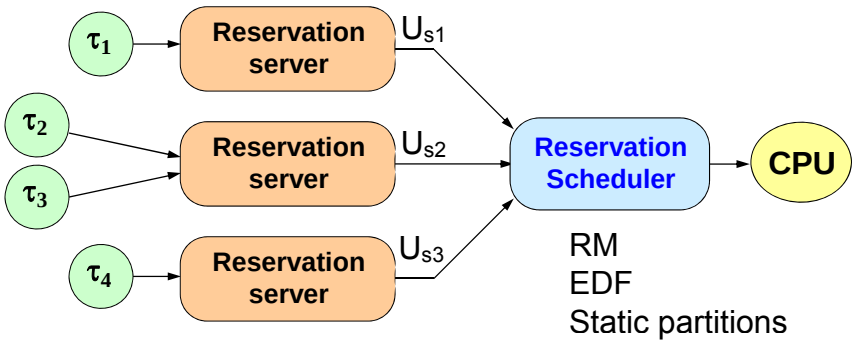


Each task receives a fraction U_i of the resource (bandwidth)

Resource enforcement

- Prevents tasks to consume more than what reserved.
- If a task executes more, it is delayed, preserving the resource for the other tasks.

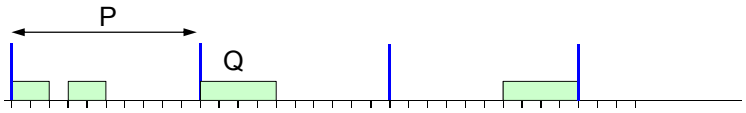
Implementing Resource Reservation



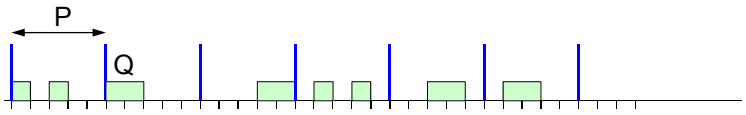
Polling
Deferrable Server
CBS

Reservation server

A way to implement a reservation is through a periodic server providing a budget **Q** every period **P**:



Reserved bandwidth: α = Q/P = 4/10

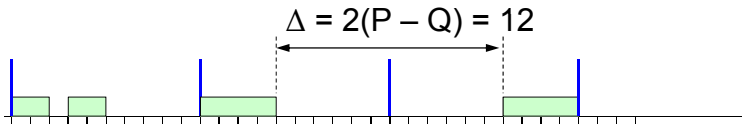


Reserved bandwidth: α = Q/P = 2/5

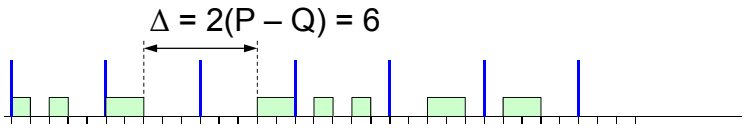
Which one is better?

Reservation server

Observe the worst-case delay:



Reserved bandwidth: $\alpha = Q/P = 4/10$

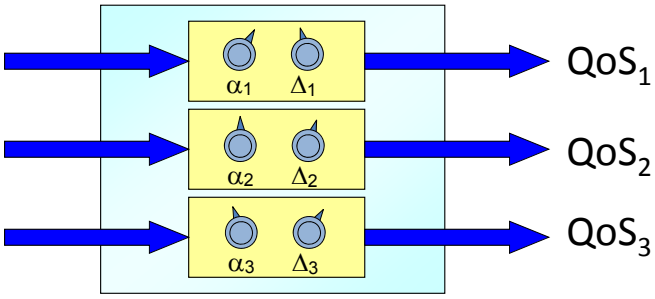


Reserved bandwidth: $\alpha = Q/P = 2/5$

Reservation server

Hence, two key parameters to describe a reservation are:

- Bandwidth: α
- Worst-case delay: Δ

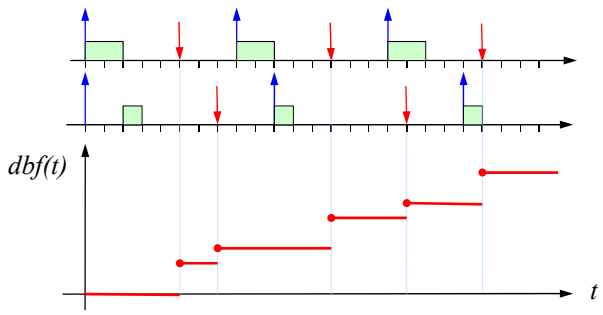


Outline

- How to express parallelism
- How to achieve temporal isolation
- How to perform real-time guarantees
- How to achieve portability by a proper abstraction (the virtual multiprocessor)
- How to optimize resource usage

Schedulability Analysis

- Application demand is described through the demand bound function:

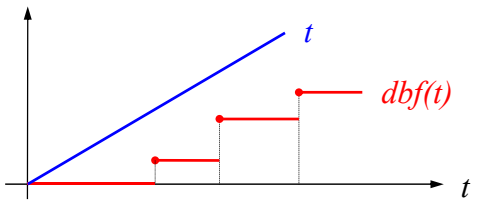


$$dbf(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

Schedulability Analysis

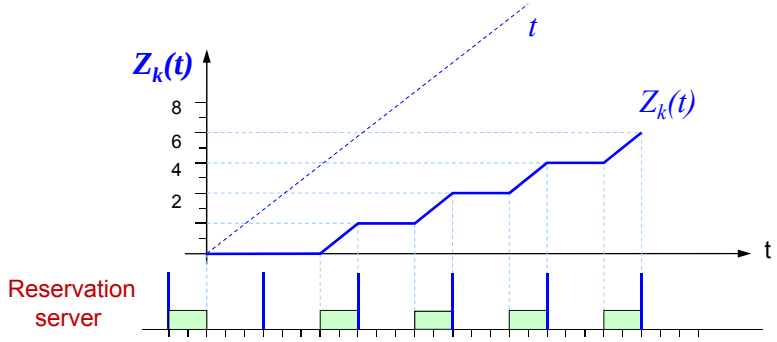
- On a single processor, feasibility is guaranteed under EDF if and only if:

$$\forall t > 0 \quad dbf(t) \le t$$



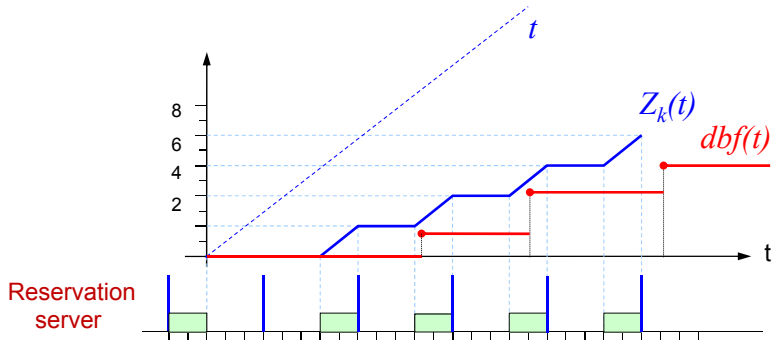
Analysis under reservations

Given a reservation R_k, the supply function $Z_k(t)$ gives the minimum amount of service time available in any interval of length t:



Analysis under reservations

$$\forall t > 0 \quad dbf(t) \le Z_k(t)$$

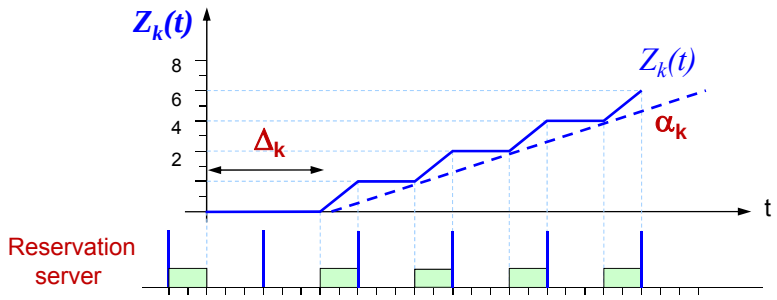


Analysis under reservations

Linear bound: This function can be lower bounded by:

$$S_k(t) = \max\{0,\ \alpha_k(t - \Delta_k)\}$$

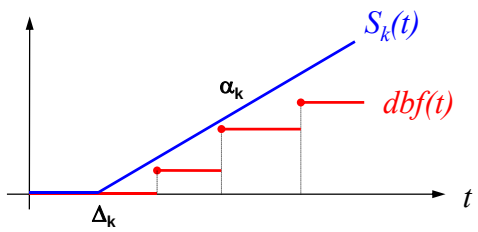
- α_k bandwidth
- Δ_k service delay



Schedulability Analysis

- On a reservation R_k, feasibility is guaranteed under EDF if:

$$\forall t > 0 \quad dbf(t) \le S_k(t)$$



$$dbf(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

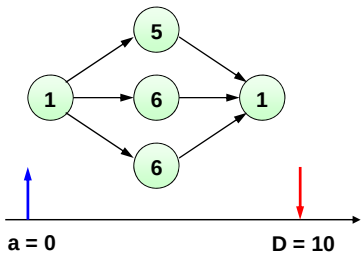
Outline

- How to express parallelism
- How to achieve temporal isolation
- How to perform real-time guarantees
- How to achieve portability by a proper abstraction (the virtual multiprocessor)
- How to optimize resource usage

Multicore Reservations

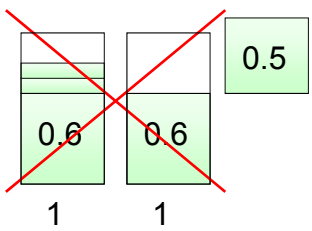
- How to extend reservations on multicore platforms?
- Does it make sense to define reservations with bandwidth $\alpha > 1$?

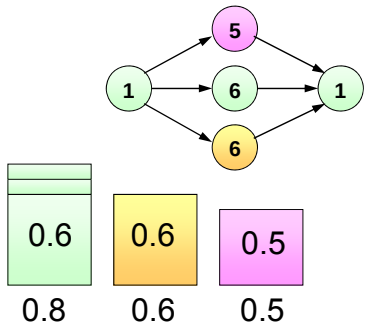
Partitioning is not trivial



$$C_s = \sum C_i = 19$$

$$U = \frac{C^s}{D} = \frac{19}{10} = 1.9 < 2$$



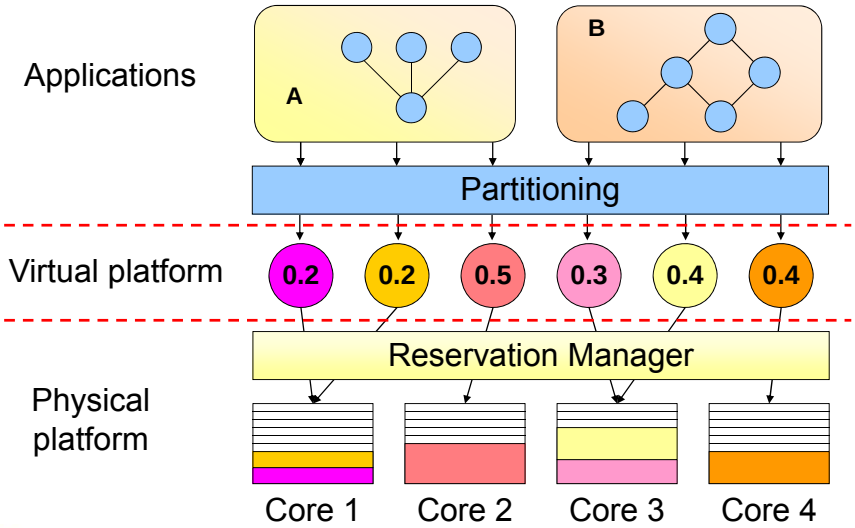


Multicore Reservations

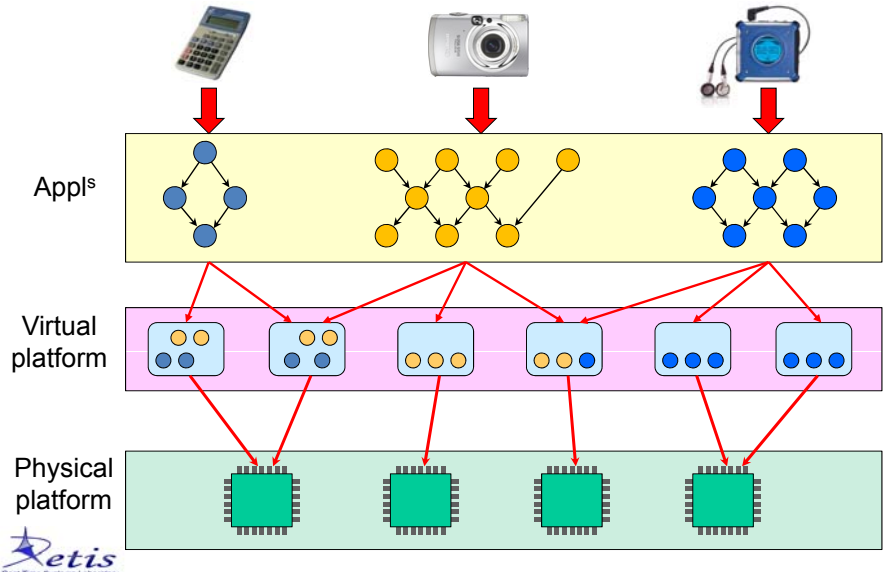
- Hence, a multicore reservation cannot be specified by the overall supplied bandwidth.

A multicore reservation must be specified as a set of uniprocessor reservations

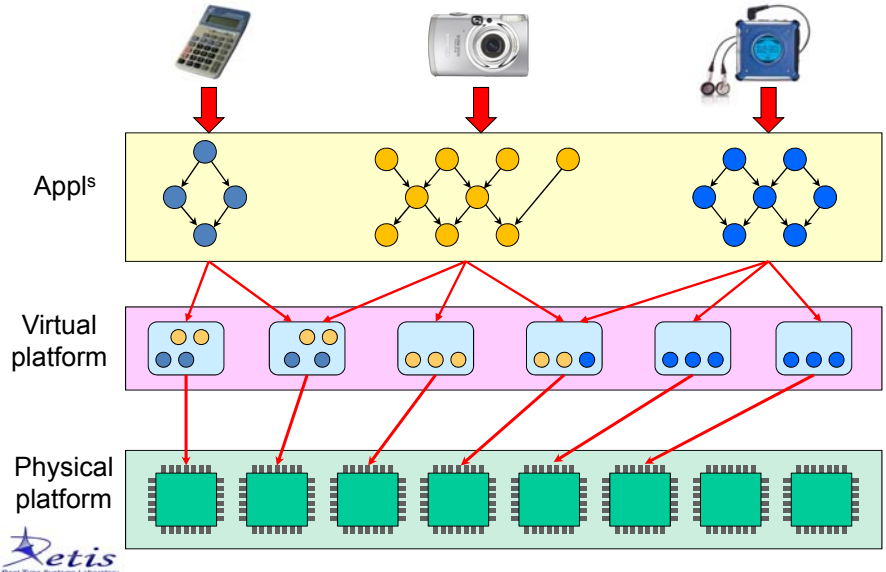
Abstracting the platform



Abstracting the platform



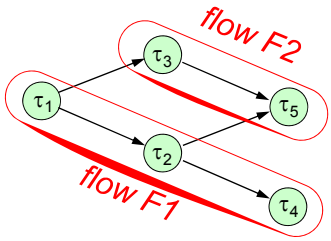
Abstracting the platform



Partitioning

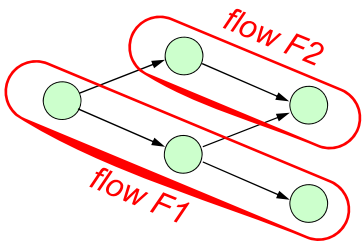
- To partition an application into a set of reservations, we identify a set of **flows**.
- Each flow is a sequential execution to be allocated on a virtual uniprocessor (i.e., a reservation)

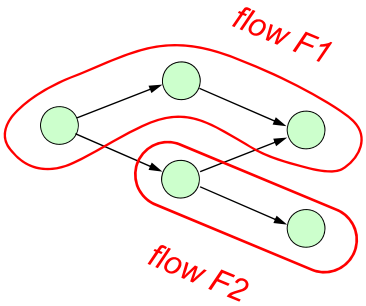
For instance:



Selecting the best flows

- Different partitions have different bandwidth consumption
- Which one gives the best results?



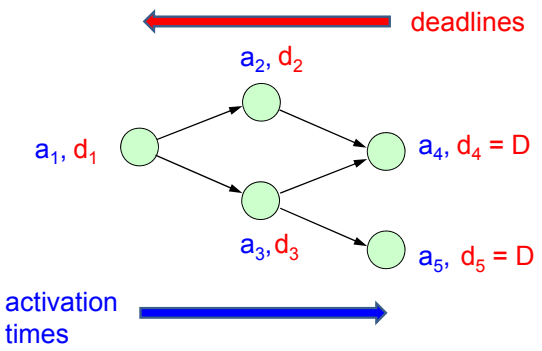


Outline

- How to express parallelism
- How to achieve temporal isolation
- How to perform real-time guarantees
- How to achieve portability by a proper abstraction (the virtual multiprocessor)
- How to optimize resource usage

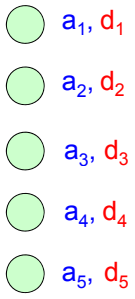
Dealing with precedence constraints

To compute the bandwidth required by a flow, we have to assign intermediate activation times and deadlines:



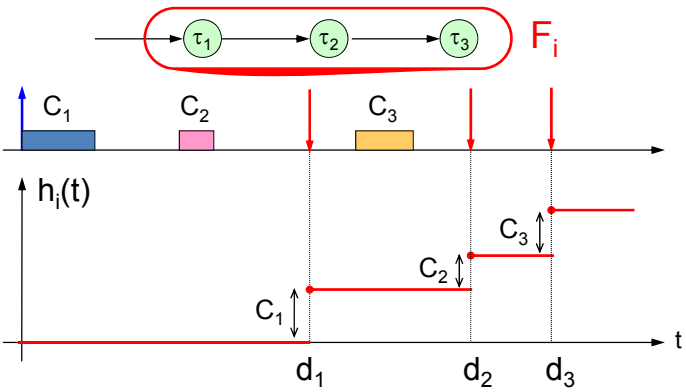
Dealing with precedence constraints

Once activation times and deadlines are assigned to each task, we can execute them according EDF, forgetting the precedence relations:



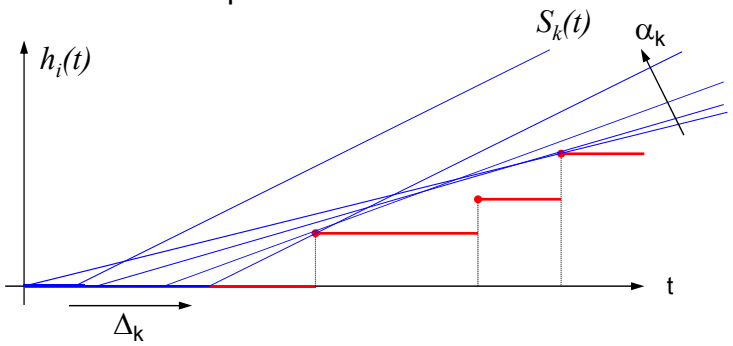
Computational demand of a flow

- Then, the processor demand required by a flow F_i is computed in each interval of time:



Real-time guarantee

- To guarantee flow F_i on VP_k it must be $\forall t \;\; h_i(t) \leq S_k(t)$
- Several solutions are possible:



The higher the delay, the higher the bandwidth required.

Resource Optimization

$$\begin{cases} \alpha = Q/P \\ \Delta = 2(P - Q) \end{cases}$$

Overhead: σ / P

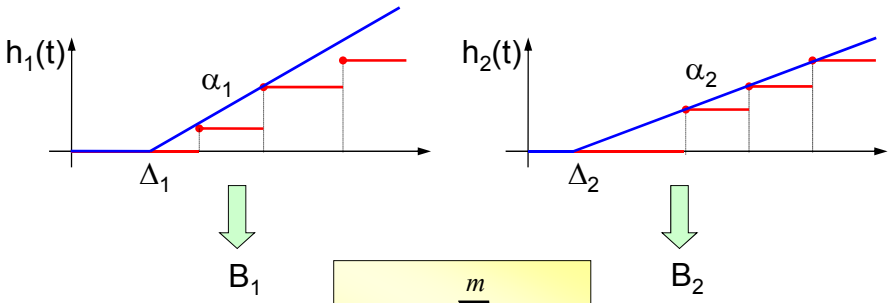
Actual Bandwidth: $B = \alpha + \sigma/P$

$$B = \alpha + 2\sigma \frac{1 - \alpha}{\Delta}$$

Taking overhead into account, it is possible to compute the (α,Δ) that minimizes B.

Optimal bandwidth

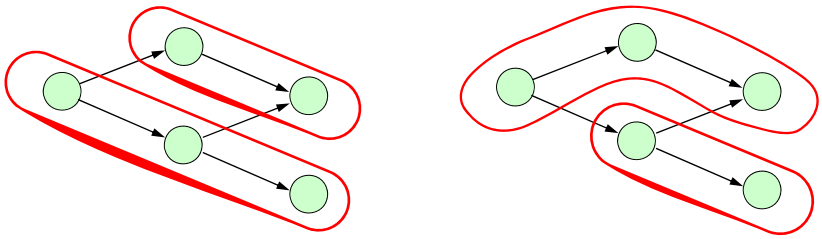
- Once the best (α,Δ) have been computed for each flow, the total bandwidth required by the application is:



$$B = \sum_{k=1}^{m} B_k$$

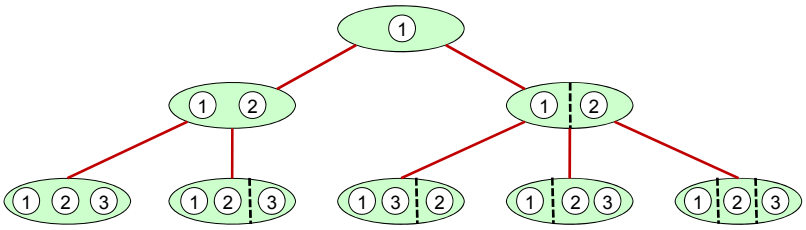
Selecting the best partition

- Different partitions require different bandwidth:



How to find the partition that minimizes B?

Search algorithm



Pruning is used to cut

- unfeasible branches ($B_k > 1$)
- redundant branches ($m > M$)

$$M = \left\lceil \delta \frac{C^s}{D} \right\rceil$$

Exponential complexity (tractable for n < 20)

Heuristic

Although the pruning, the branch and bound algorithm has exponential complexity and the method is not usable with more than 15 tasks.

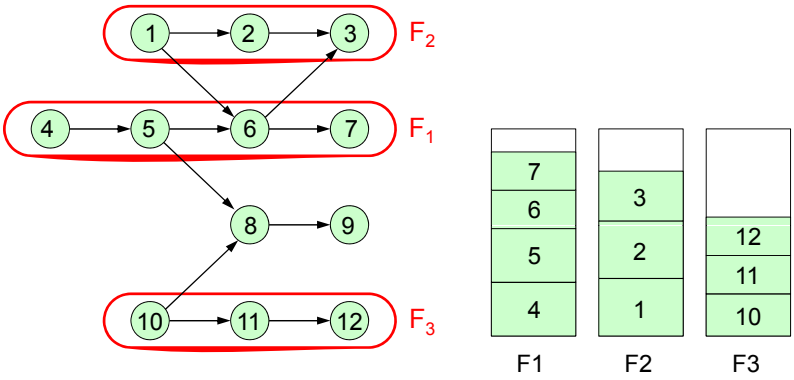
For this reason, we propose a heuristic algorithm to partition the applications into flows.

Let M_{low} be the lower bound on the number of cores needed by the application.

$$M_{low} = \left\lceil \frac{C^s}{D} \right\rceil$$

Heuristic

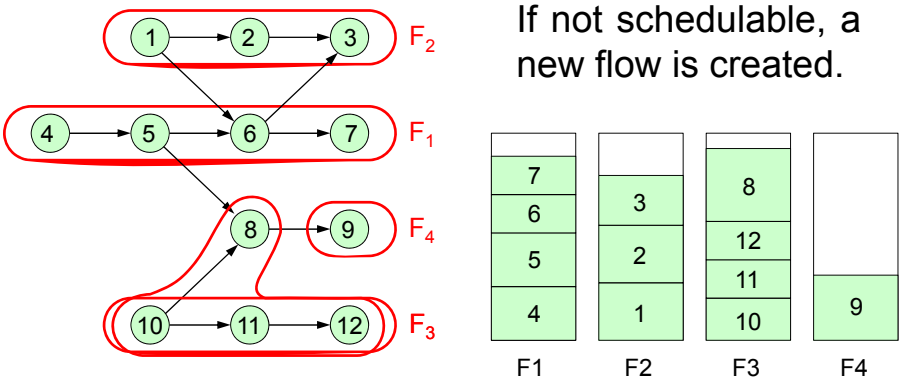
The algorithm starts by constructing M_{low} flows using the longest paths in the precedence graph:



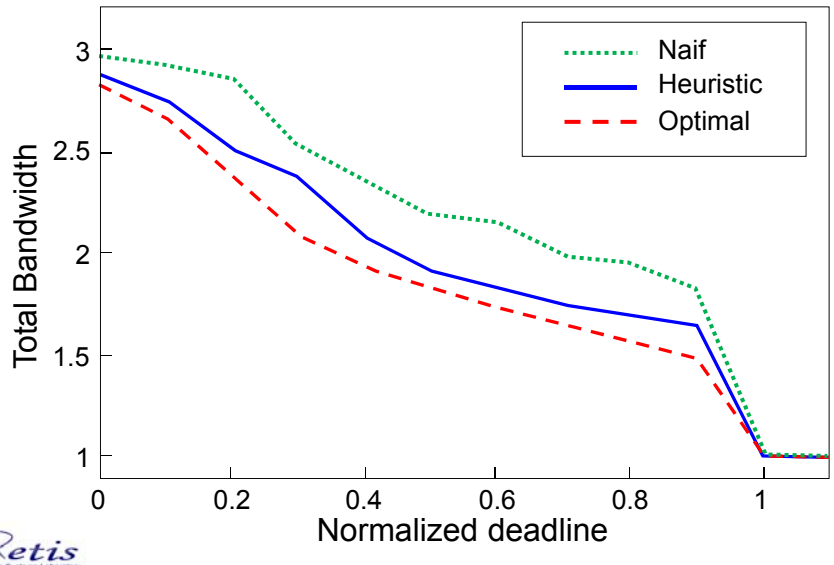
Heuristic

The remaining tasks (selected by decreasing C_i) are put in the existing flows using Best Fit.

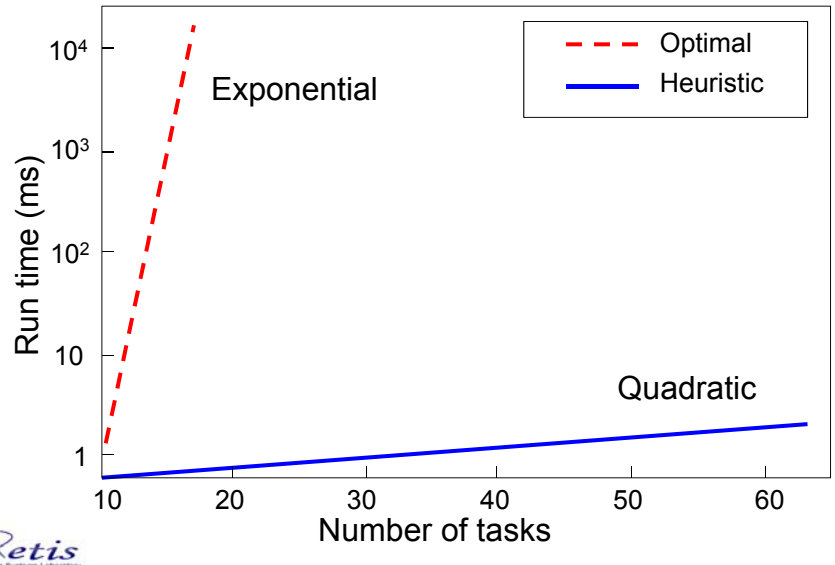
If not schedulable, a new flow is created.



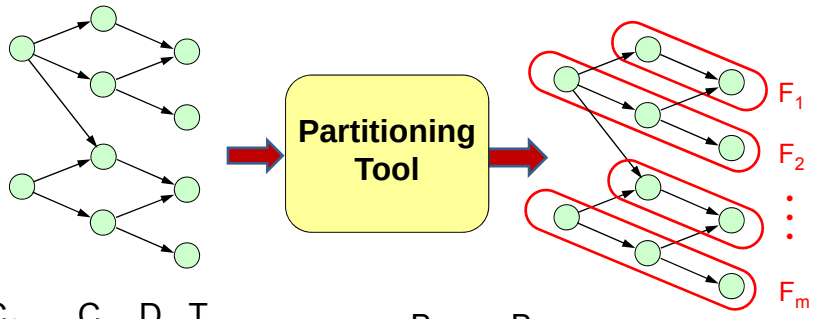
Experimental results



Experimental results



The Partitioning Tool

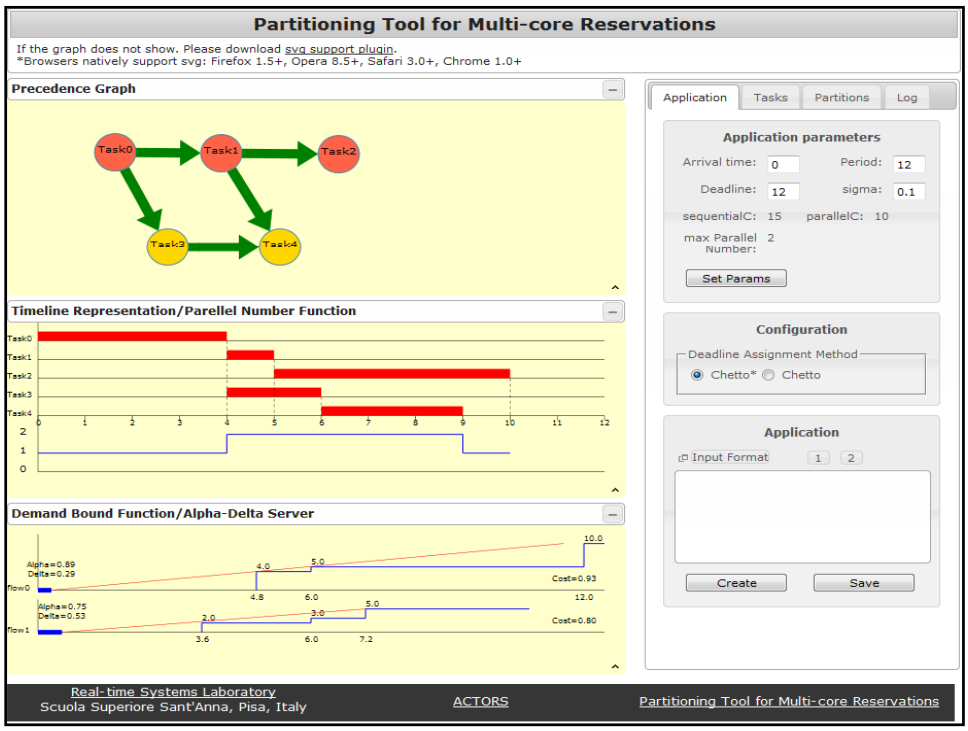


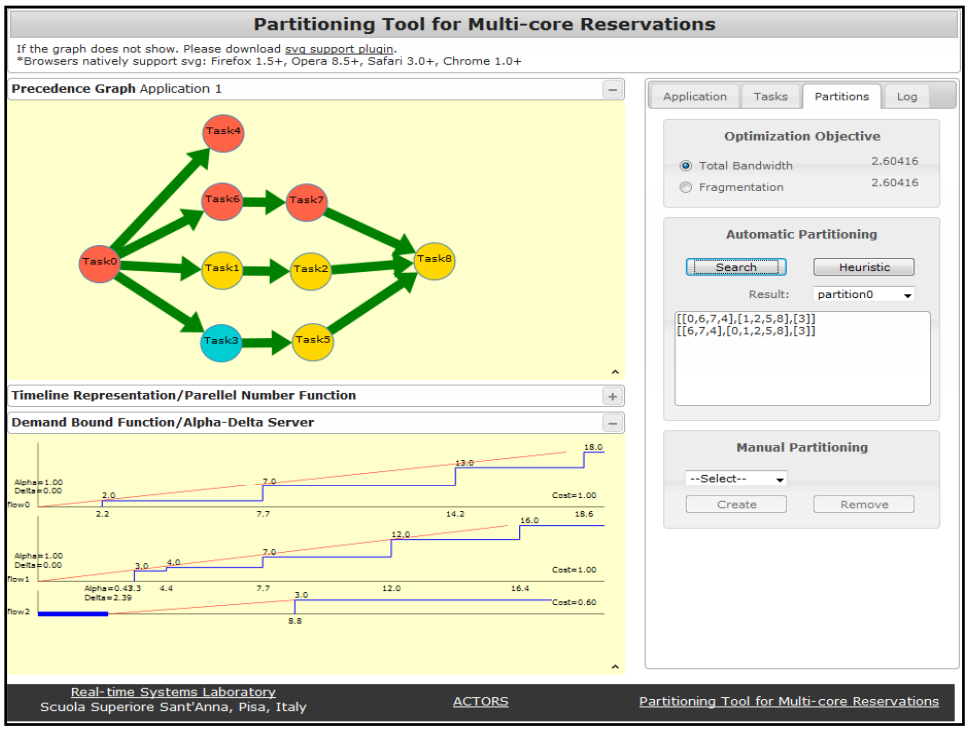
$C_1, \ldots C_n$, D, T

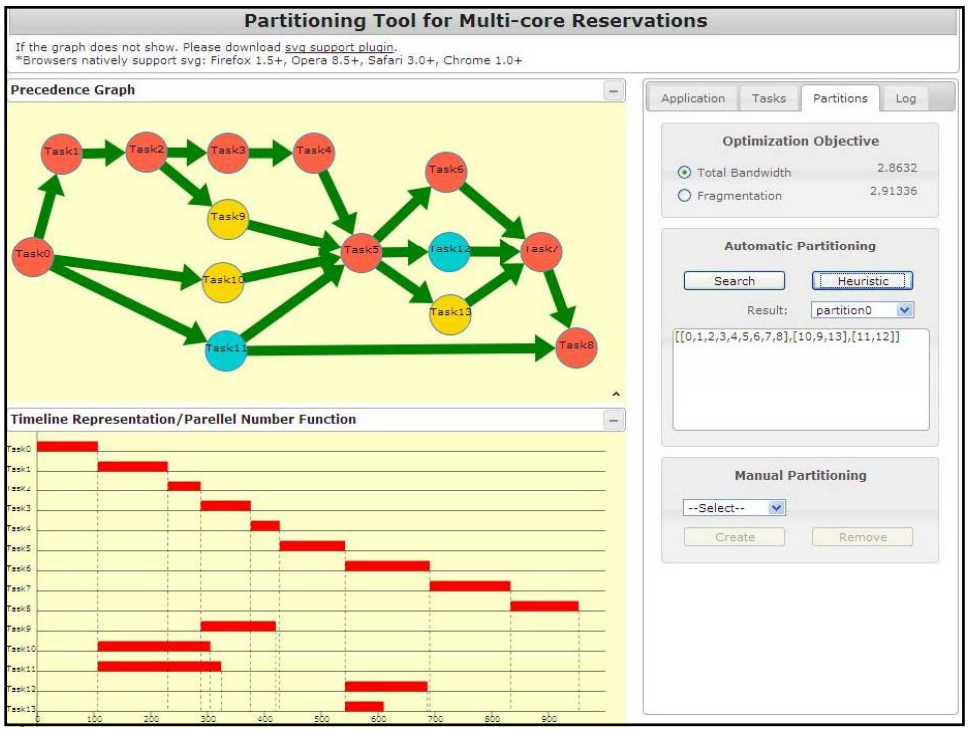
ctx-sw. overhead σ

$B_1, \ldots, B_m$

Such that $B = \sum B_k$ is minimum




Partitioning Tool for Multi-core Reservations
If the graph does not show. Please download svg support plugin.
*Browsers natively support svg: Firefox 1.5+, Opera 8.5+, Safari 3.0+, Chrome 1.0+
Precedence Graph Application 1
Task0
Task1
Task2
Task3
Task4
Task5
Task6
Task7
Task8
Timeline Representation/Parellel Number Function
Demand Bound Function/Alpha-Delta Server
Application
Tasks
Partitions
Log
Optimization Objective
Total Bandwidth 2.60416
Fragmentation 2.60416
Automatic Partitioning
Search
Heuristic
Result: partition0
[[0,6,7,4],[1,2,5,8],[3]]
[[6,7,4],[0,1,2,5,8],[3]]
Manual Partitioning
--Select--
Create
Remove
Real-time Systems Laboratory
Scuola Superiore Sant'Anna, Pisa, Italy
ACTORS
Partitioning Tool for Multi-core Reservations

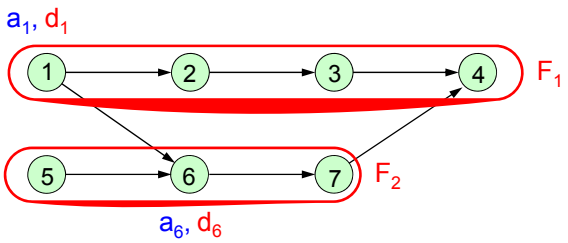

Partitioning Tool for Multi-core Reservations
If the graph does not show. Please download svg support plugin.
*Browsers natively support svg: Firefox 1.5+, Opera 8.5+, Safari 3.0+, Chrome 1.0+
Precedence Graph
Task0
Task1
Task2
Task3
Task4
Task5
Task6
Task7
Task8
Task9
Task10
Task11
Task12
Task13
Timeline Representation/Parellel Number Function
Application
Tasks
Partitions
Log
Optimization Objective
Total Bandwidth 2.8632
Fragmentation 2.91336
Automatic Partitioning
Search
Heuristic
Result: partition0
[[0,1,2,3,4,5,6,7,8],[10,9,13],[11,12]]
Manual Partitioning
--Select--
Create
Remove

Open Problems

- Communication delays
- Shared Resources
- Adaptivity

Accounting for communication delays

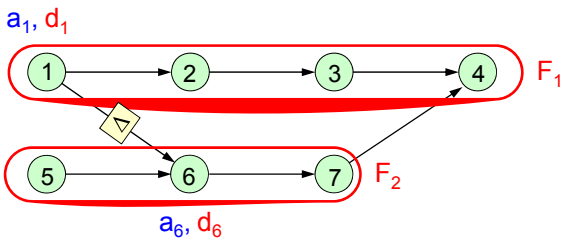
- Delays depend on the allocation on the physical cores.
- They affect task activation times



$a_6 = \max(a_5, d_1)$

Accounting for communication delays

- Delays depend on the allocation on the physical cores.
- They affect task activation times

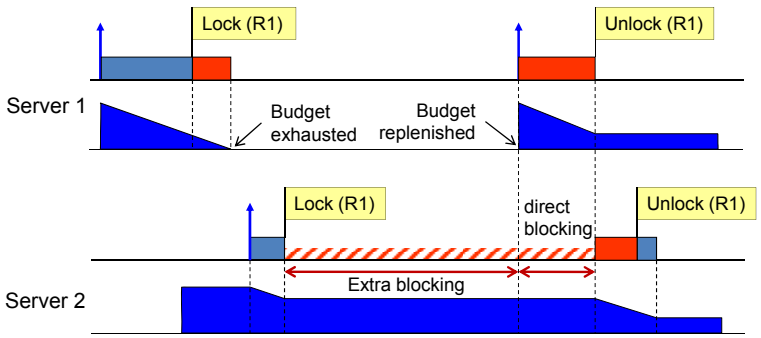


$a_6 = \max(a_5, d_1+\Delta)$

Δ is larger if F_1 and F_2 are allocated on ≠ cores

Problems with shared resources

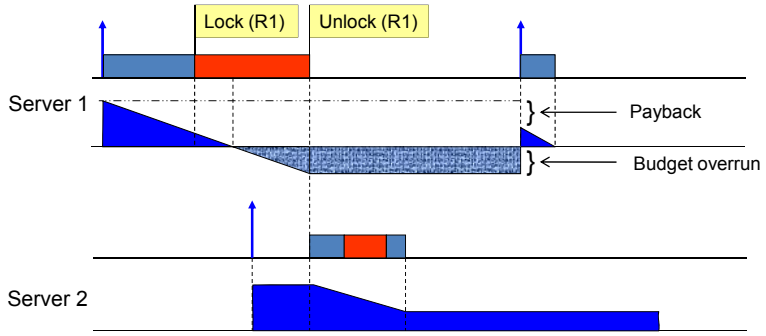
If a server exhausts its budget while locking a shared resource, extra blocking can be introduced in other servers needing the same resource:



Possible solution

Budget overrun (with payback)

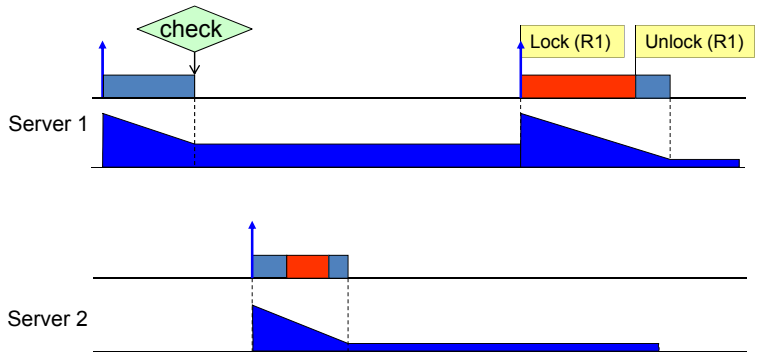
Effective, but it breaks isolation and requires using extra bandwidth.



A better solution

Budget check [Behnam et al. 2007]

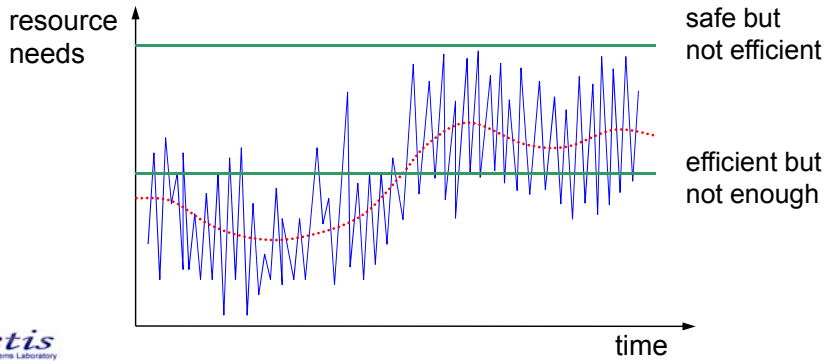
Before locking a resource, the budget is checked: if it is not sufficient, the server waits for replenishment.



Dynamic applications

- Some applications have highly variable behavior difficult to predict (e.g. multimedia players, visual tracking)

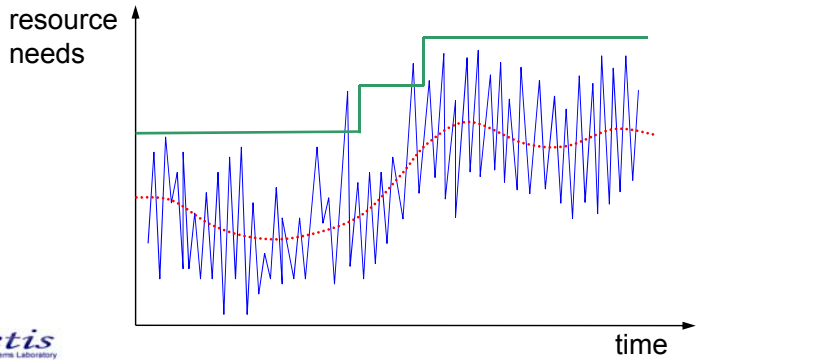
→ any reservation is not appropriate



Need for adaptivity (modes)

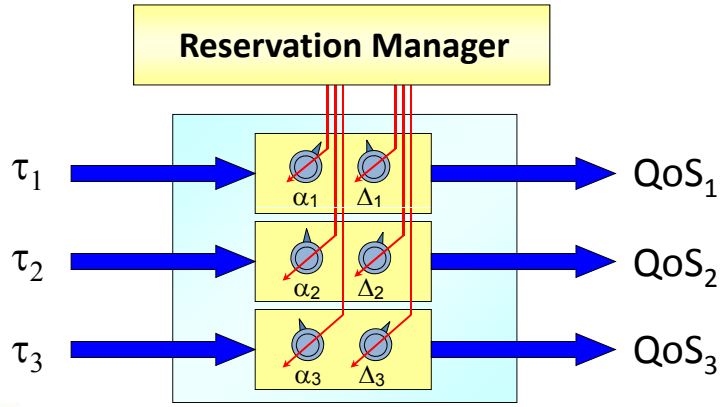
- Reservations should be adapted to the application needs based on runtime requirements:

Off line guarantee → { Feasible modes, Feasible transitions }



Adaptive QoS Management

- A Reservation Manager must decide how and when changing a reservation to satisfy the application needs, as well as to preserve the other reservations.



Thank you