Multiprocessor Systems

Ingo Sander
ingo@kth.se

Liu: Chapter 9

So far…

- many realistic properties of real-time systems have been ignored
- We have so far assumed
  - that systems consist of a single processor
  - that data dependencies impose precedence constraints between jobs
  - that timing constraints of jobs are usually not independent

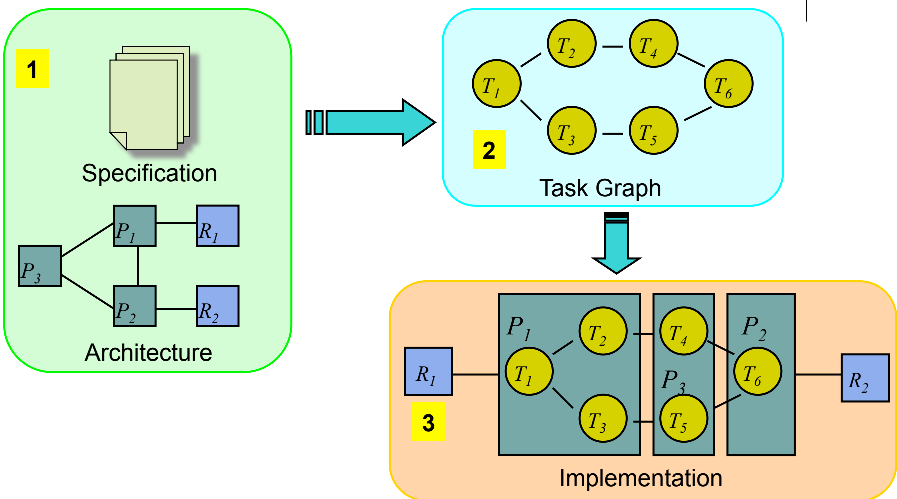
Multiprocessor Systems

- We switch now our focus to multiprocessor systems
  - Principles we learned give a solid base to tackle these systems
  - New problems arise
    - Task assignment problem
    - Multiprocessor protocols for resource access control
    - Interprocessor synchronization
- Again: overall goal is to meet all deadlines

Multiprocessor Systems

- Embedded systems often consist of several processors
  - Processors can be homogeneous (same type) or heterogeneous (different type)
    - Also hardware units performing a dedicated processing function (like an FIR-filter) can be viewed as processors
  - Embedded systems are often distributed
    - In cars several processors are connected by a bus

Simplified Design Flow

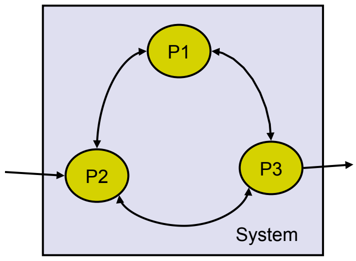


What can be parallelized?

- Multiprocessors allow for parallel execution
- But not all applications can be parallelized

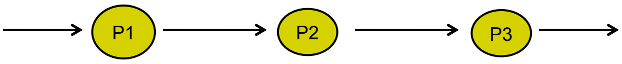
Concurrent Activities

- Concurrent activities can be assigned to different processors
- P1, P2, and P3 can run on different processors

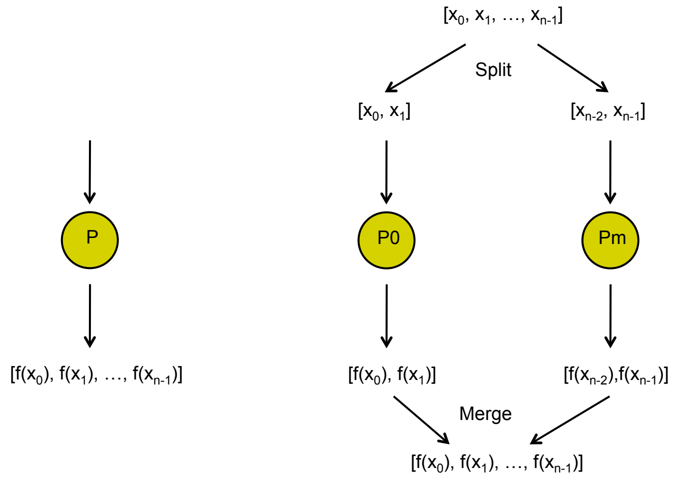


Pipeline

- Sequential programs that comply to data flow style can be parallelized using pipelining
- P1, P2, and P3 can run on different processors, in case the result of each process only depends on the input data



Data-Parallelism



Simplified Design Flow

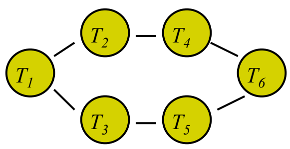
- The design of a multiprocessor embedded system is very challenging
  - architecture may not be fixed
  - many different implementations possible
  - remote access to resources
  - synchronization required
  - precedence constraints
- The design of such a system must be based on well-understood principles and techniques

Schedulers in Multiprocessor Systems

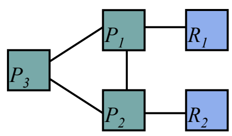
- Multiprocessor systems may have centralized or decentralized schedulers
- Here we assume that each processor has its own scheduler
- Also we do not distinguish multiprocessor systems from distributed systems and call both multiprocessor systems

Task Assignment

- In a static system, the application is partitioned into modules that are bound to processors
- This is called task assignement



Task Graph



Architecture

Task Assignment

- At some stage of the design process, the execution times resource requirements, data and control dependencies of all the tasks become known
- Task assignment determines
  - how many processors are needed
  - on which processor each task executes

Task Assignment

- Task assignment is a very complex (NP-hard) problem
  - often heuristics are used
  - task assignment is most often done off-line
  - in a dynamic system, task assignment can be used as an acceptance test

Task Assignment

- The simplicity of the model in the task assignment problem can vary in complexity
  1. Communication and placement of resources is ignored
  2. Only communication costs are considered
  3. Both communication costs and resource access costs are considered
- All models have their merits in the design flow

Task Assignment Based on Execution-Time Requirements

- It is often meaningful to ignore resources and communication in an early design phase
- In some shared-memory applications with few memory conflicts, the communication costs may be very small

Task Assignment Based on Execution-Time Requirements

- Task Assignment Problem
  - Given are the utilizations of n tasks
  - The system shall be partitioned into modules (set of tasks) in such a way that the tasks in each module are schedulable by themselves on a processor according to a uniprocessor scheduling algorithm of a given class
  - The task assignment is defined by a subset of tasks in every module

Task Assignment Based on Execution-Time Requirements

- Task assignment problem can be formulated as the simple uniform-size bin-packing problem
- Sizes of all bins is schedulable utilization U_{ALG} of the algorithm (EDF = 1; RMA = $ln\ 2$)
- Sizes of each item (task) is the utilization
- The number of bins required to pack the items is the number of processors required to feasibly schedule all tasks

First Fit Algorithm

- First-Fit is a simple heuristic algorithm for the bin-packing problem
  - Tasks are assigned one by one in arbitrary order
  - First task is assigned to processor P_1
  - After i-1 tasks, the i-th task T_i is assigned to processor P_k,
    - if the total utilization of T_i and the tasks already assigned to P_k is equal to or less than the schedulable utilization U_{ALG}
    - and assigning T_i to any of the processors P_1, P_2, …, P_{k-1}, would make the total utilization of tasks on the processor larger than U_{ALG}

Variable-Size Bin-Packing

- For the EDF algorithm U_{EDF} does not depend on the number of tasks
- But in the RM algorithm U_{RM} depends on the number of tasks
  - If a fixed size of a bin is assumed, we have to use $U_{RM} = ln\ 2$
  - Otherwise bin-size can be a function of the number of tasks
    - increasing the complexity of the problem!

Rate-Monotonic First Fit Algorithm

- In the RMFF algorithm tasks are first sorted in non-decreasing order according to their periods
- All tasks are assigned in the First Fit manner, starting from T_1
- A task can be assigned to a processor if the total utilization of T_i and the x tasks already scheduled on that processor is equal or less than $U_{RM}(x+1)$

Rate Monotonic Small Tasks Algorithm

- The RMFF algorithm does not exploit the fact that the schedulable utilization of tasks on a processor is higher, when the tasks are closer to being simply periodic
- The RMST algorithm exploits this fact, and is based on Theorem 6.14

Rate Monotonic Small Tasks Algorithm

- Theorem 6.14: The schedulable utilization $U_{RM}(\zeta,n)$ of the RM algorithm as a function of ζ and n is given by

$$U_{RM}(n,\zeta) = (n-1)(2^{\zeta/(n-1)} - 1) + 2^{1-\zeta} - 1 \quad \text{for } \zeta < 1 - 1/n$$
$$U_{RM}(n,\zeta) = n(2^{1/n} - 1) \quad \text{for } \zeta \geq 1 - 1/n$$

where

$$\zeta = \max_{1 \leq i \leq n} X_i - \min_{1 \leq i \leq n} X_i$$
$$X_i = \log_2 p_i - \lfloor \log_2 p_i \rfloor$$

Rate Monotonic Small Tasks Algorithm

- In the RMST algorithm periodic tasks are first sorted in non-decreasing order according to their parameters X_i
- Then tasks are assigned in this order to processors in the first-fit manner
- Here the following schedulability condition, which follows from Theorem 6.14 is used:

$$u_i + U_k \leq \max(\ln 2, 1 - \zeta_k \ln 2)$$

Task Assignment To Minimize Total Communication Costs

- Communication between tasks which run on same processor is usually significantly lower than for tasks that run on different processors
- When communication costs are considered objective of task assignment is twofold
  - Find minimum number of processors needed
  - Find minimum of communication costs for this number of processors

Task Assignment To Minimize Total Communication Costs

- Cost of communication depends on the volume of data exchanged and the bandwidth of the communication link
- Here we only discuss systems of homogeneous processors, which communicate via a shared communication channel (e.g. a bus)

Task Assignment To Minimize Total Communication Costs

- The *communication cost* between two tasks T_i and T_k is expressed by a single value $C_{i,k}$, if the tasks are placed on different modules
- $C_{i,k}$ usually reflects the amount of data exchanged between both tasks
- To account for memory contention an *interference cost* can be included if tasks T_i and T_k are placed on the same processor

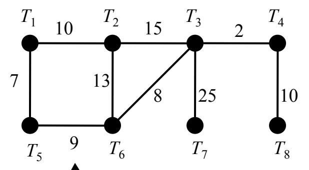
Task Assignment To Minimize Total Communication Costs

- In order to find a feasible solution different heuristic algorithms or other approaches such as integer linear programming can be used
- Given a cost function minimize
  - the total communication cost
  - the number of processors used

Example of Task Partitioning

i	T_i	u_i	i	T_i	u_i
1	(2,1)	0.50	5	(6,1)	0.17
2	(3,1)	0.33	6	(10,1)	0.10
3	(4,1)	0.25	7	(15,1)	0.07
4	(5,1)	0.20	8	(25,1)	0.04

(EDF scheduling)



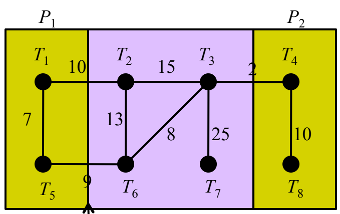
Communication Cost

Objective is to find a partitioning, which is feasible at minimal costs (here interference cost is neglected!)

Example of Task Partitioning

i	T_i	u_i	i	T_i	u_i
1	(2,1)	0.50	5	(6,1)	0.17
2	(3,1)	0.33	6	(10,1)	0.10
3	(4,1)	0.25	7	(15,1)	0.07
4	(5,1)	0.20	8	(25,1)	0.04

(EDF scheduling)



Communication Cost

Objective is to find a partitioning, which is feasible at minimal costs (here interference cost is neglected!)

Local vs. Remote Resources

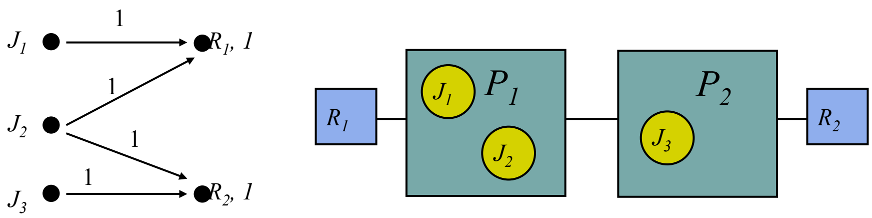
- We assume that each resource resides on a processor
  - Scheduler of that processor controls the access to the resource
  - If a job is using the resource, the critical section is executing on the processor belonging to the resource

Local vs. Remote Resources

- MPCP (Multiprocessor Priority-Ceiling Protocol) Resource Model
  - The processor on which each resource resides is called its *synchronization processor*
  - The processor on which each job is released and becomes ready is called the *local processor* of the job
  - A resource that resides on the local processor is a *local resource*
  - A resource that resides on another processor is a *global resource*
  - A *global resource* is a resource that is required by jobs that have different local processors

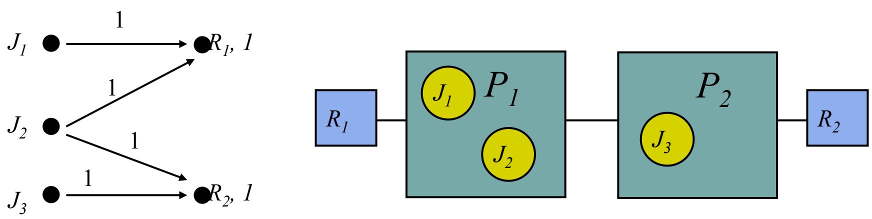
Local vs. Remote Resources

- MPCP
  - P_1 is the synchronization processor of R_1
  - P_2 is the synchronization processor of R_2
  - J_1 and J_2 are local jobs on P_1 and J_3 is a local job on P_2



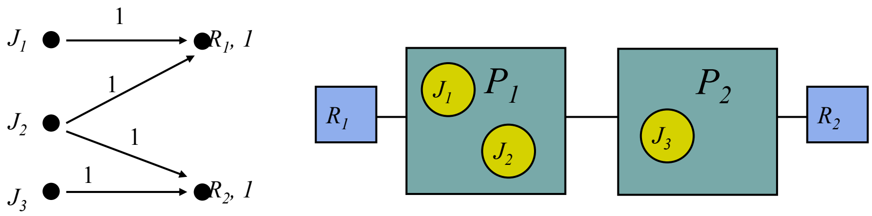
Local vs. Remote Resources

- MPCP
  - Since J_2 uses the remote resource R_2 on P_2 , R_2 is a global resource
  - During the remote global critical section during while J_2 uses the resource R_2, J_2 executes on P_2
  - When the global critical section of J_2 completes, the job J_2 returns to P_2



Local vs. Remote Resources

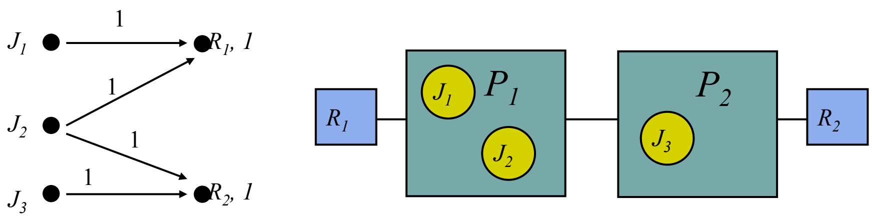
- End-to-End Resource Model
  - The job J_2 is an end-to-end job and consists of three component jobs
    - First component executes on P_1, the remote critical section executes on P_2 and the third component executes on P_1



IL2212 Embedded Software 35

Local vs. Remote Resources

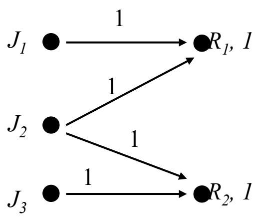
- End-to-End Resource Model
  - Since J_1 executes only on P_1 and J_3 executes only on P_2 they consist only of one component
  - Using this definition, each component job requires only resources on the processor on which the component jobs executes

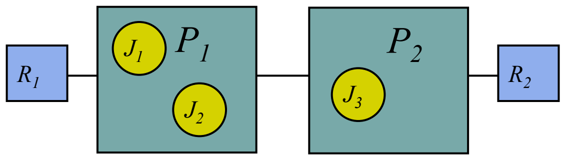


IL2212 Embedded Software 36

Local vs. Remote Resources

- End-to-End Resource Model
  - The scheduler of each processor can treat all requests for resources controlled by it as local requests
  - No need to distinguish jobs from component jobs
  - All jobs in an end-to-end task requires only resources on its local processor





Multiprocessor Priority-Ceiling Protocol (MPCP)

- Assumption
  - Tasks and resources have been assigned and statically bound to processors
  - Scheduler of every synchronization processor knows the priorities and resource requirements of all tasks requiring the global resources managed by the processor

Multiprocessor Priority-Ceiling Protocol (MPCP)

- The scheduler of each processor schedules all the local tasks and global critical sections on a processor on a fixed-priority basis
- Resource accesses are controlled by a modified priority-ceiling protocol

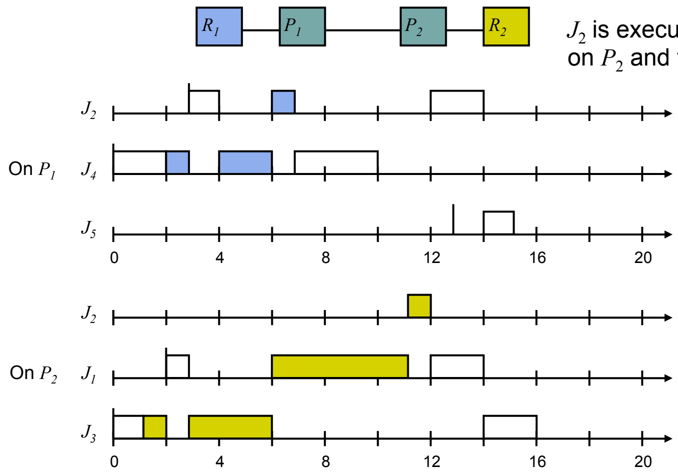
Multiprocessor Priority-Ceiling Protocol (MPCP)

- The multiprocessor priority-ceiling protocol schedules all global critical sections at higher priorities than all local tasks on every synchronization processor
- Idea is that local tasks shall not delay execution of global critical sections and prolong the blocking time of the remote task

Multiprocessor Priority-Ceiling Protocol (MPCP)

- This is implemented if the lowest priority π_{lowest} of all tasks in a system is known
- Then the global critical sections of a task with priority π_i are executed at a priority π_i - π_{lowest}

Multiprocessor Priority-Ceiling Protocol (MPCP)



J_2 is executed first on P_1 then on P_2 and then again on P_1

Multiprocessor Priority-Ceiling Protocol (MPCP)

- The blocking time $b_i(rc)$ for a task consist of the following contributions
  1. *local blocking time*: caused by contentions on the local processor
  2. *local preemption delay*: caused by global critical sections that belong to remote tasks executing on the local processor
  3. *remote blocking time*: caused by contention with lower-priority tasks that execute on the remote processor

Multiprocessor Priority-Ceiling Protocol (MPCP)

- The blocking time $b_i(rc)$ for a task consist of the following contributions
  4. *remote preemption delay*: caused by preemptions of other higher-priority remote sections executing on the same remote processor
  5. *deferred blocking time*: suspended execution of local higher-priority tasks

Formal Model in Liu, Section 9.3.2 – not discussed in this course

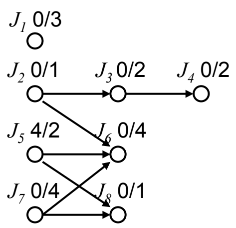
Timing Anomalies

Ingo Sander
ingo@kth.se

Liu: Chapter 4

Example: Priority-Driven Scheduling

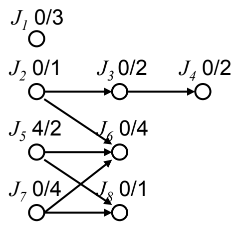
- Given are eight tasks with the following priorities: $J_1 > J_2 > \ldots > J_8$
- Jobs are scheduled on two processors P_1 and P_2
- Communication cost is negligible
- There is only one common priority queue



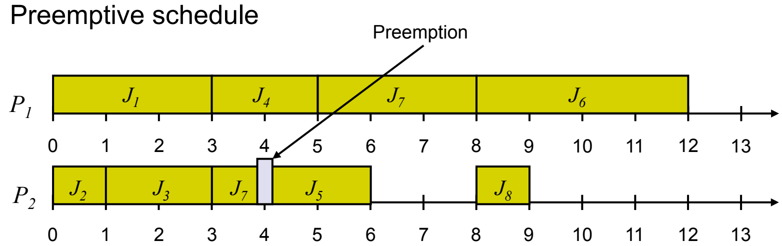
r/e = Release Time / Execution Time

Example: Priority-Driven Scheduling

r/e = Release Time/Execution Time

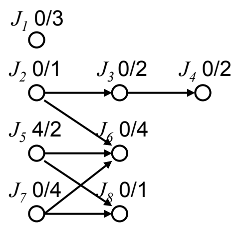


Preemptive schedule

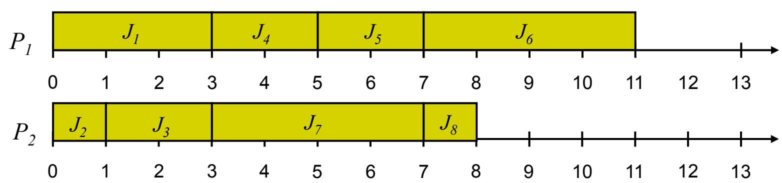


Example: Priority-Driven Scheduling

r/e = Release Time/Execution Time

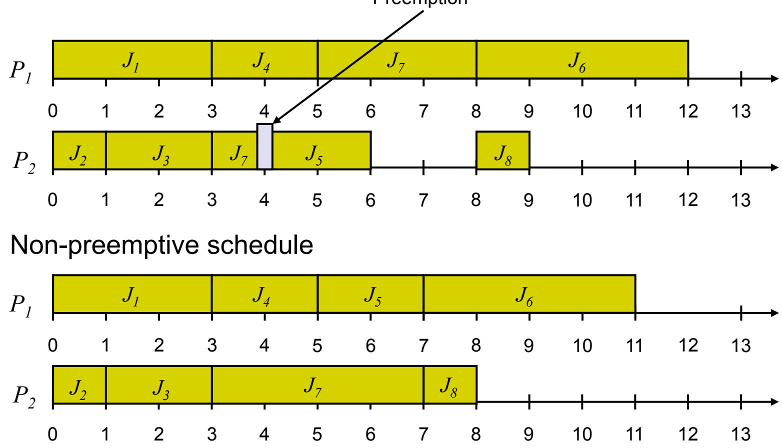


Non-preemptive schedule



Example: Priority-Driven Scheduling

Preemptive schedule



Example: Priority-Driven Scheduling

- Non-preemptive schedule gave better performance than preemptive schedule
  - No general rule, but priority scheduling results can be non-intuitive
  - Scheduling algorithms have different properties

Dynamic vs. static systems

- If jobs are scheduled on multiple processors and a job can be dispatched from the priority run queue to any of the processors, the system is *dynamic*
- A job *migrates* if it starts execution on a processor, is preempted, and later resumes execution on another processor
- If jobs are partitioned into subsystems and these subsystems are assigned to processors the system is *static*

Dynamic vs. static systems

- We expect that static systems have worse performance than dynamic systems
- But there do not exist reliable techniques to validate timing constraints of dynamic systems, while they exist for static systems

➢ Thus most hard real-time systems are static!

Summary

- The theory for uniprocessor forms the foundation for static multiprocessor systems
- New problems arise
  - Task Assignment
  - Multiprocessor protocols
  - Interprocessor
  - Timing Anomalies
- Naturally the problems get much more complex