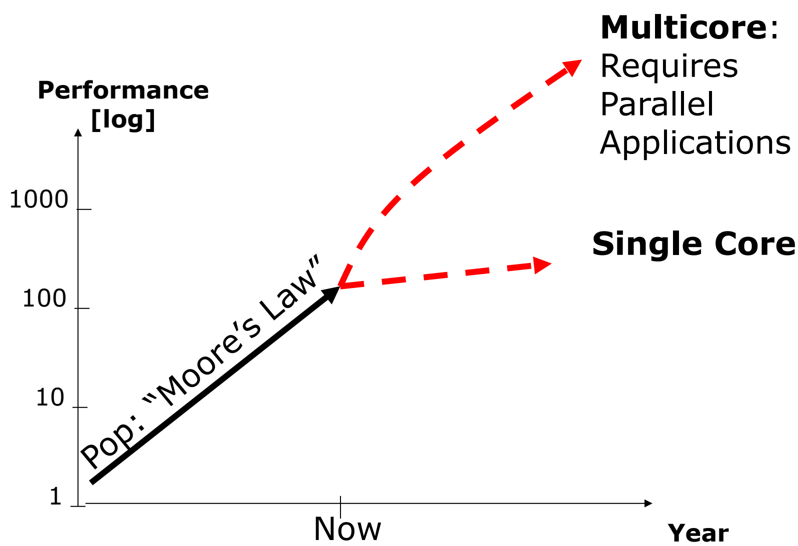
Interesting topics

- Why multiprocessor?
  - energy, performance and predictability
- What are multiprocessor systems
  - Architectures, OS etc
- Design RT systems on multiprocessors
  - Task Assigment
- Multiprocessor scheduling
  - (semi-)partitioned scheduling
  - global scheduling

Why multiprocessor systems?

To get high performance and to reduce energy consumption

Hardware: Trends



Theoretically you may get:

- Higher Performance
  - Increasing the cores -- unlimited computing power ∞ !
- Lower Power Consumption
  - Increasing the cores, decreasing the frequency
    - Performance (IPC) = Cores * F ➔ 2* Cores * F/2 ➔ Cores * F
    - Power = C * V^2 * F ➔ 2* C * $(V/2)^2$ * F/2 ➔ C * V^2 /4 * F

➔ Keep the "same performance" using ¼ of the energy (by doubling the cores)

This sounds great for embedded & real-time applications!

What's happening now?

- General-Purpose Computing
  (Symposium on High-Performance Chips, Hot Chips 21, Palo Alto, Aug 23-25, 2009)
  - 4 cores in notebooks
  - 12 cores in servers
    - AMD 12-core Magny-Cours will consume less energy than previous generations with 6 cores
  - 16 cores for IBM servers, Power 7
- Embedded Systems
  - 4 cores in ARM11 MPCore embedded processors

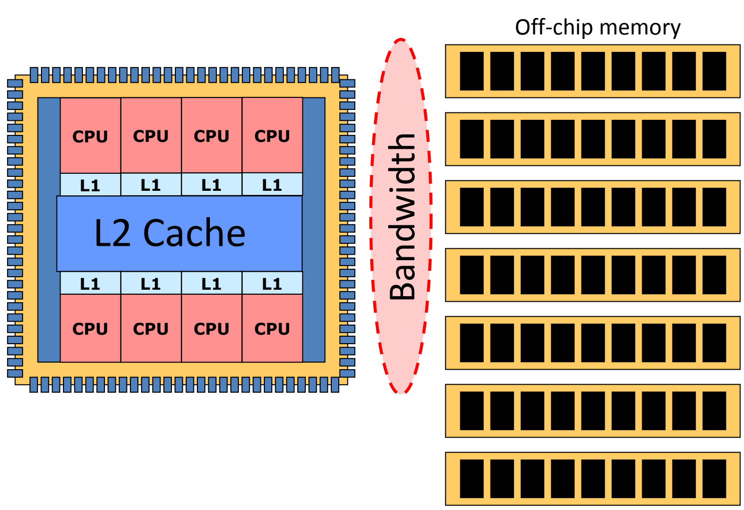
What next?

- Manycores (>100's of cores) predicted to be here in a few years – e.g. Ericsson

What are multiprocessor systems?

“Tightly connected” processors by a “high-speed” interconnect e.g. cross-bar, bus, NoC etc.

Typical Multicore Architecture



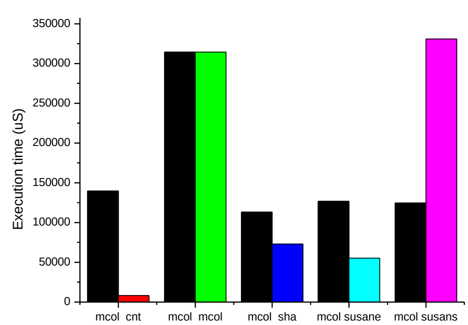
8

Key Challenges

- New programming models, languages
  - Ada, Erlang run on multiprocessor systems
- Legacy software migration
  - Parallelization, synchronization, memory layouts
- New operating systems
  - Different strategies
- Real-time guarantees for embedded apps
  - Cache interferences
  - Multiprocessor scheduling

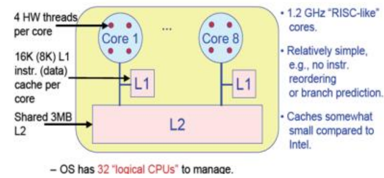
An Experiment on a LINUX machine with 2 cores

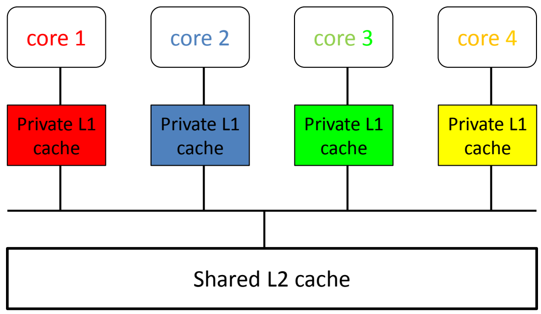
WCET (vary 10 – 50%)



mcol runs with different programs

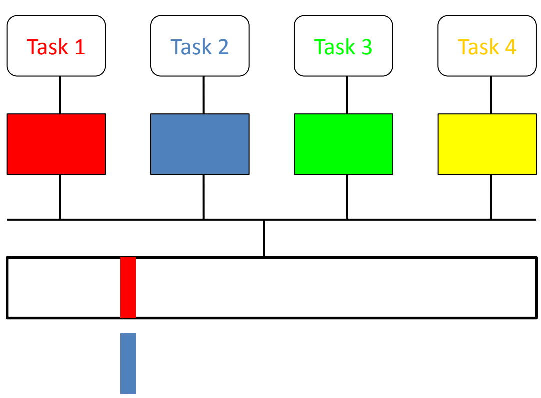
An Example Architecture





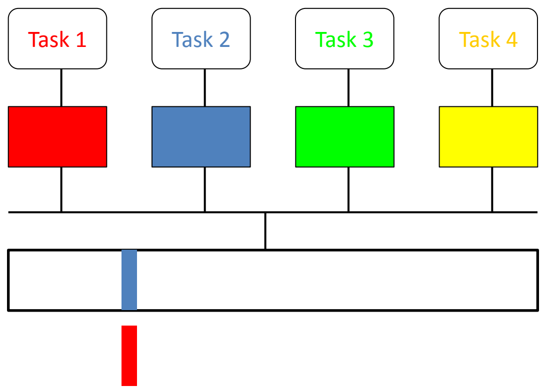
Cache analysis on multicore

- **L2 cache contents of task 1 may be over-written by task 2**



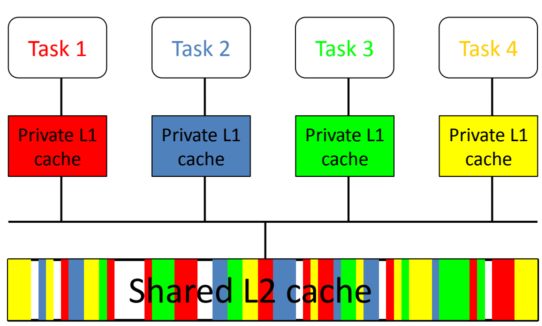
Cache analysis on multicore

- L2 cache contents of task 1 may be over-written by task 2



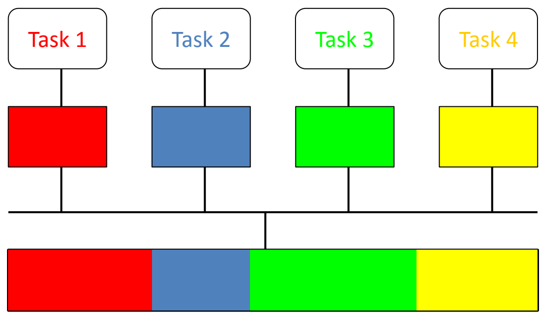
13

Cache analysis on multicore

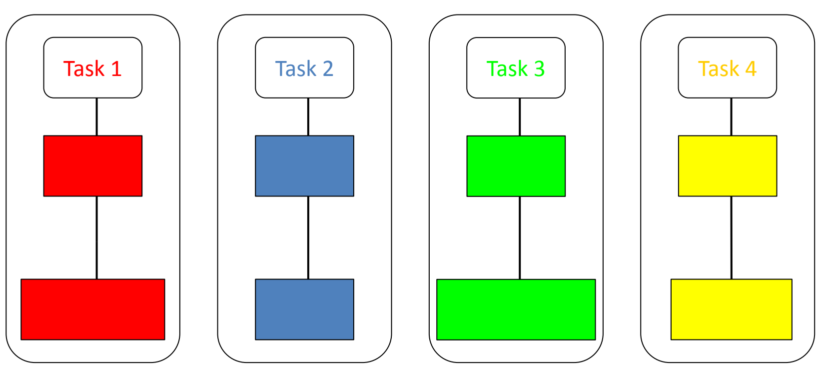


14

Cache-Coloring: partitioning and isolation



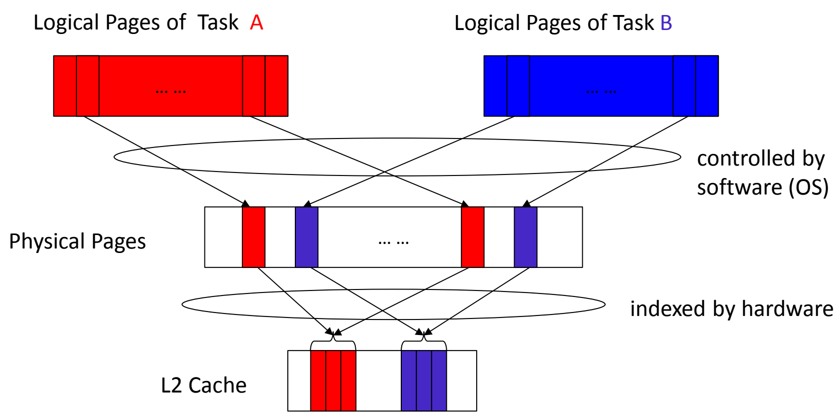
Cache-Coloring: partitioning and isolation



WCET can be estimated using static techniques for single processor platforms (for the given portion L2 cache)

Cache-Coloring: partitioning and isolation

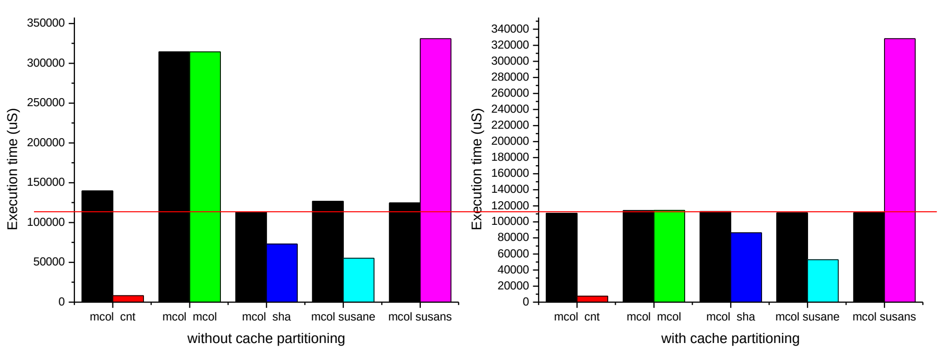
- E.g. LINUX – Power5 (16 colors)



17

An Experiment on a LINUX machine with 2 cores

with Cache Coloring/Partitioning [ZhangYi et al]

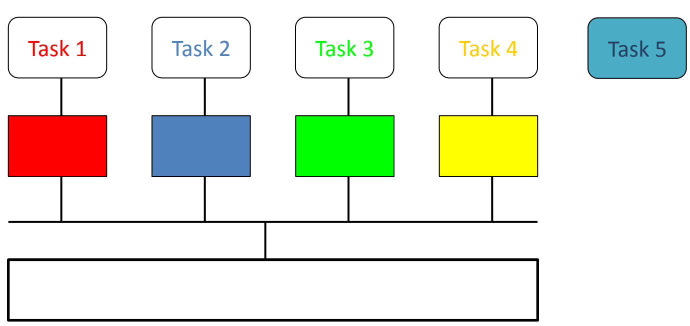


18

What to do when #tasks > #cores ?

The multicore challenge: Schedulability analysis

- #cores < #tasks



Multiprocessor scheduling

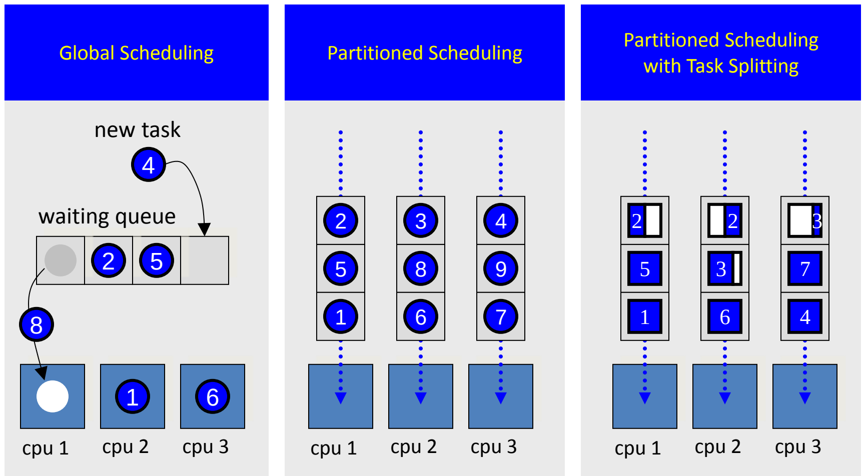
- *"Given a set J of jobs where job j_i has length l_i and a number of processors m_i, what is the minimum possible time required to schedule all jobs in J on m processors such that none overlap?"* – Wikipedia
  - That is, design a schedule such that the response time of the last tasks is minimized

- The problem is NP-complete
- It is also known as the "load balancing problem"

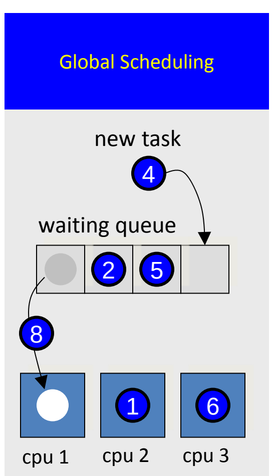
Multiprocessor scheduling
– static and dynamic task assignment

- Partitioned scheduling
  - Static task assignment
    - Each task may only execute on a fixed processor
    - No task migration
- Semi-partitioned scheduling
  - Static task assignment
    - Each instance (or part of it) of a task is assigned to a fixed processor
    - task instance or part of it may migrate
- Global scheduling
  - Dynamic task assignment
    - Any instance of any task may execute on any processor
    - Task migration

Multiprocessor Scheduling



Global Scheduling



Global scheduling

- All ready tasks are kept in a global queue
- When selected for execution, a task can be dispatched to any processor, even after being preempted

Global scheduling Algorithms

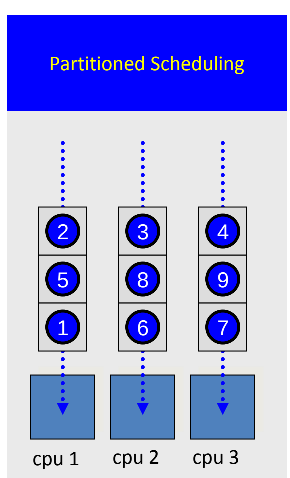
Any algorithm for single processor scheduling may work, but schedulability analysis is non-trivial

- EDF – Unfortunately not optimal!
  - No simple schedulability test known (only sufficient)
- Fixed Priority Scheduling e.g. RM
  - Difficult to find the optimal priority order
  - Difficult to check the schedulability

Global Scheduling: + and -

- Advantages:
  - Supported by most multiprocessor operating systems
    - Windows NT, Solaris, Linux, ...
  - Effective utilization of processing resources (if it works)
    - Unused processor time can easily be reclaimed at run-time (mixture of hard and soft RT tasks to optimize resource utilization)
- Disadvantages:
  - Few results from single-processor scheduling can be used
  - No "optimal" algorithms known except idealized assumption (Pfair sch)
  - Poor resource utilization for hard timing constraints
    - No more than 50% resource utilization can be guaranteed for hard RT tasks
  - Suffers from scheduling anomalies
    - Adding processors and reducing computation times and other parameters can actually decrease optimal performance in some scenarios!

Partition-Based Scheduling



Bin-packing algorithms

- The problem concerns packing objects of varying sizes in boxes (”bins”) with the objective of minimizing number of used boxes.
  – Solutions (Heuristics): Next Fit and First Fit
- Application to multiprocessor systems:
  – Bins are represented by processors and objects by tasks.
  – The decision whether a processor is ”full” or not is derived from a utilization-based schedulability test.

Partitioned scheduling

- Two steps:
  – Determine a mapping of tasks to processors
  – Perform run-time scheduling
- Example: Partitioned with EDF
  – Assign tasks to the processors such that no processor’s capacity is exceeded (utilization bounded by 1.0)
  – Schedule each processor using EDF

Rate-Monotonic-First-Fit (RMFF): [Dhall and Liu, 1978]

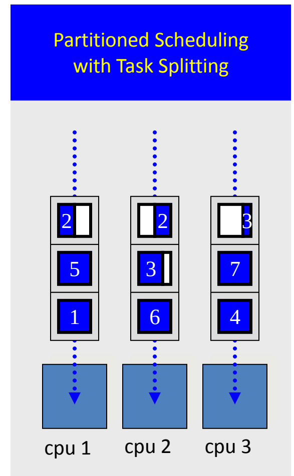
- First, sort the tasks in the order of increasing periods.
- Task Assignment
  - All tasks are assigned in the First Fit manner starting from *the task with highest priority*
  - A task can be assigned to a processor if all the tasks assigned to the processor are RM-schedulable i.e.
    - the total utilization of tasks assigned on that processor is bounded by $n(2^{1/n}-1)$ where n is the number of tasks assigned.

      (One may also use the Precise test to get a better assignment!)
  - Add a new processor if needed for the RM-test.

Partitioned scheduling

- Advantages:
  - Most techniques for single-processor scheduling are also applicable here
- Partitioning of tasks can be automated
  - Solving a bin-packing algorithm
- Disadvantages:
  - Cannot exploit/share all unused processor time
  - May have very low utilization, bounded by 50%

Partition-Based Scheduling with Task-Splitting



Partition-Based scheduling with Task Splitting

- High resource utilization
- High overhead (due to task migration)

Fixed-Priority Multiprocessor Scheduling