8. Multicore systems

Credits to A. Burns and A. Wellings

RTS York

to B. Andersson and J. Jonsson for their work in *Proc. of the the IEEE Real-Time Systems Symposium*, WiP Session, 2000, pp. 53–56

and to a student of this class a few years back

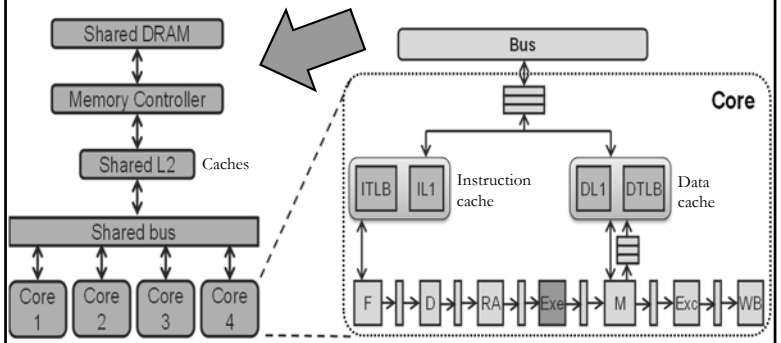
Hardware architecture taxonomy

- A multiprocessor (or multi-core) is *tightly coupled*
  - Global status and workload information on all processors (cores) can be kept current at low cost
  - The system may use a centralized dispatcher and scheduler
  - When each processor (core) has its own scheduler, the decisions and actions of all schedulers are coherent
    - Scheduling in this model is an NP-hard problem
- A distributed system is *loosely coupled*
  - It is too costly to keep global status
  - There usually is a dispatcher / scheduler per processor

Fundamental issues

- Hardware architecture taxonomy
  - Homogeneous vs. heterogeneous processors
    - Research focused first on SMP (*symmetric multiprocessors*) that make a much simpler problem
- Scheduling approach
  - Global or partitioned or alternatives between these extremes
    - Partitioning = allocation problem followed by single-CPU scheduling
- Optimality criteria are shattered
  - EDF no longer optimal and not always better than FPS
  - Global scheduling not always better than partitioned

Understanding the hardware /3



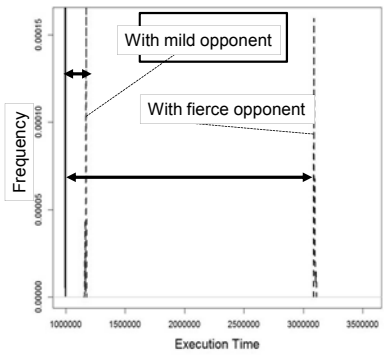
Courtesy of PROXIMA

Hardware interference /1

- Parallel execution on a multiprocessor causes vast opportunities of contention for hardware resources that are shared among the cores
- This phenomenon increases the execution time of running threads by causing them to hold the CPU *without* progressing (**!**)
  - Unlike software interference, which prevents a ready thread from running

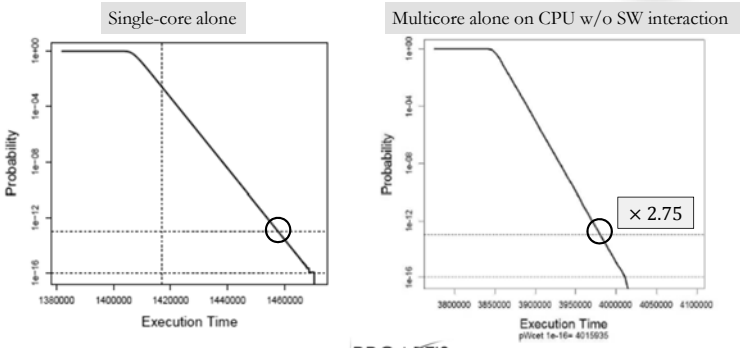
Hardware interference /2

- The WCET of a simple single-path program running alone does <u>not</u> stay the same when other programs happen to execute on other CPUs



Courtesy of PROARTIS

A big anomaly



Courtesy of PROARTIS

State of the art: what a loss!

- Some task sets may be deemed unschedulable even though they have low utilization
  - Much less than linear with the number of processors
  - This is known as the Dhall's effect [Dhall & Liu, 1978]
- The known *exact* schedulability tests have exponential time complexity
  - The known sufficient tests have polynomial time complexity but obviously are pessimistic
- Rate-monotonic priority assignment is not optimal
- No optimal priority assignment scheme with polynomial time complexity has been found yet

Simplifying assumptions

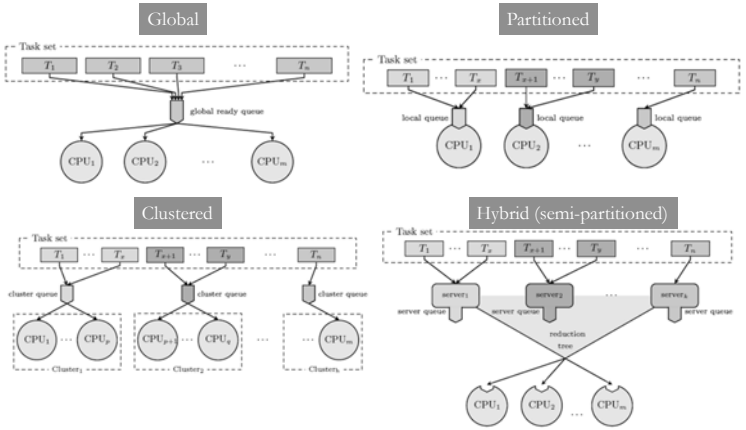
- *Processor (CPU) identity*
  - All processors are equivalent
- *Task independence*
  - Tasks are logically independent of one another
- *Task unity*
  - Tasks have no internal parallelism: they can run only on one CPU at any one time
- *Task migration*
  - Tasks can run on different CPUs at different times
- *No overhead*
  - Context switch and migration costs are built into WCET estimates

Predictability [Ha & Liu, 1994]

- For arbitrary job sets on multiprocessors, if the scheduling algorithm is ***work-conserving***[1], preemptive, global (with migration), with fixed job priorities is predictable
  - Job completion times monotonically related to job execution times
- Hence it is safe to consider only upper bounds for job execution times in schedulability tests
- This is <u>not true</u> for non-preemptive scheduling

1) A scheduling algorithm is *work conserving* if processors are not idle while tasks eligible for execution are not able to execute on other processors

The solution space for scheduling



Software interference /1

- We know what is the interference I_i suffered by a task τ_i for single-processor scheduling
  - How does this change for multiprocessors?
- For *global* multiprocessor scheduling with m processors interference only occurs for tasks from $m + 1$ onward
- Multiprocessor interference can be computed as the sum of all intervals when m higher-priority tasks execute <u>in parallel</u> on all m processors

Software interference /2

- A very pessimistic bound considers all higher-priority tasks to always fully interfere
  - $R_k^{max} = C_k + \frac{1}{m}\sum_{\tau_j \in hp(k)}\left(\left\lceil \frac{R_k^{max}}{T_j} \right\rceil C_j + Cj\right)$
- This naive bound can be improved, and has been, but for great computational complexity and still without becoming exact

Dhall's effect /1

Task	T	D	C	U
a	10	10	5	0.5
b	10	10	5	0.5
c	12	12	8	0.67

On 2 processors $\sum_i U_i = 1.67 < 2$

- Under global scheduling, EDF and FPS would run tasks **a** and **b** first on each of the 2 processors
- But this would leave no time for task **c** to complete
  - 7 time units on each processor, 14 in total, but 8 on neither
- Even if the total system is underutilized (**!**)

Dhall's effect /2

Task	T	D	C	U
d	10	10	9	0.9
e	10	10	9	0.9
f	10	10	2	0.2

On 2 processors $\sum_i U_i = 2$

- Partitioned scheduling does not work here either
- After tasks **d** and **e** are allocated, task **f** cannot reside on just one processor
  - It needs to migrate from one to the other to find room for execution
- And it also needs that tasks **d** and **e** are willing to use cooperative scheduling for it complete in time

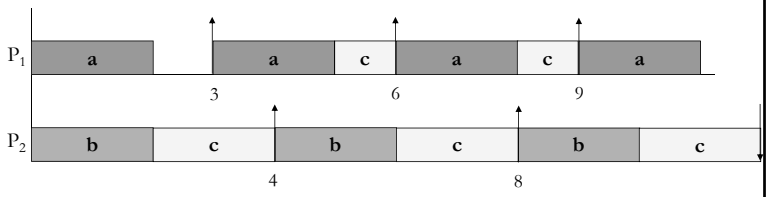
Global scheduling anomalies

- In single-processor real-time scheduling the deadline miss ratio often highly depends on the system load
  - This suggests that increasing the period should decrease the utilization and thus decrease the deadline miss ratio
- **Anomaly 1**
  - A *decrease* in processor demand from higher-priority tasks can *increase* the interference on lower-priority tasks because of the change in the time when tasks execute
- **Anomaly 2**
  - A *decrease* in processor demand of a task causes an *increase* in the interference suffered by that task

Anomaly 1: decrease in hp demand

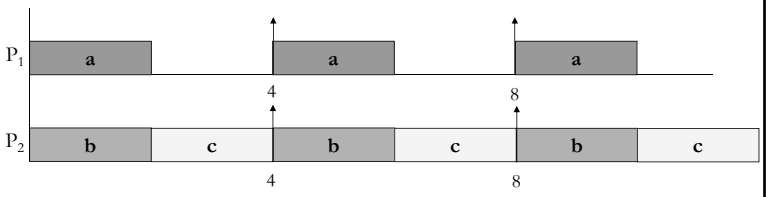
Task	T	D	C	U
a	3	3	2	0.67
b	4	4	2	0.50
c	12	12	8	0.67

$m = 2$ processors and $\sum_i U_i = 1.83$ but τ_c is *saturated* because $C_c + I_c = D_c$ hence any increase in I_c would make it unschedulable



Anomaly 1 (cont'd)

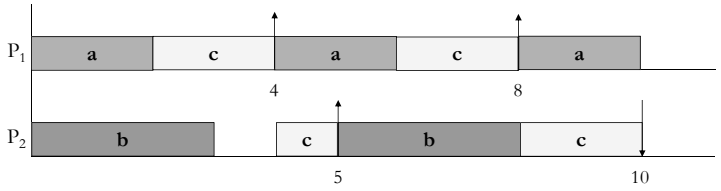
- If we reduce T_a to 4 we *decrease* system load to $U = 1.67$
- But in this way I_c *increases* from 4 to 6 and τ_c misses its deadline (**!**)



Anomaly 2: decrease in own demand

Task	T	D	C	U
a	4	4	2	0.5
b	5	5	3	0.6
c	10	10	7	0.7

$m = 2$ processors and $U = 1.8$ but τ_c with $I_c = 3$ is *saturated*



Anomaly 2 (cont'd)

- If we extend T_c to 11 we *decrease* system load to $U = 1.74$
- But in this way I_c *increases* from 3 to 5 (**!**) as it becomes visible in the second job of τ_c