8.a Multicore systems – initial reckoning

Credits to A. Burns and A. Wellings

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

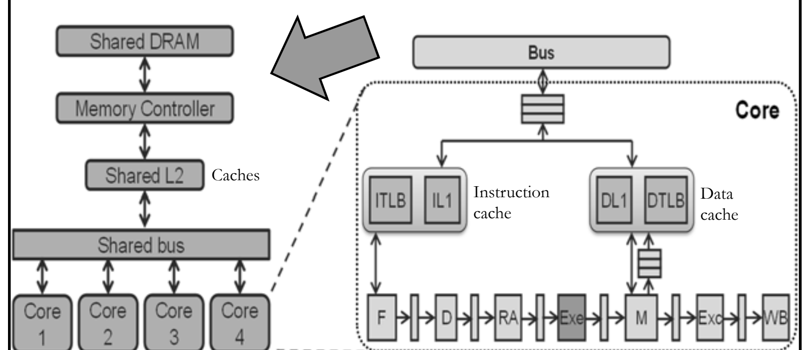
Hardware architecture taxonomy

- 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



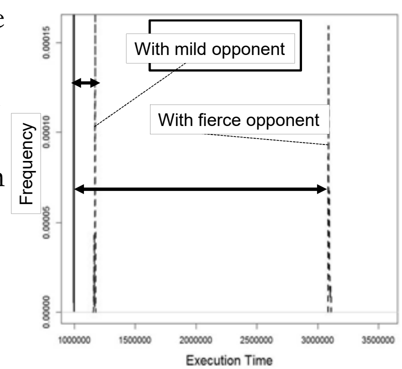
Courtesy of PROXIMA

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

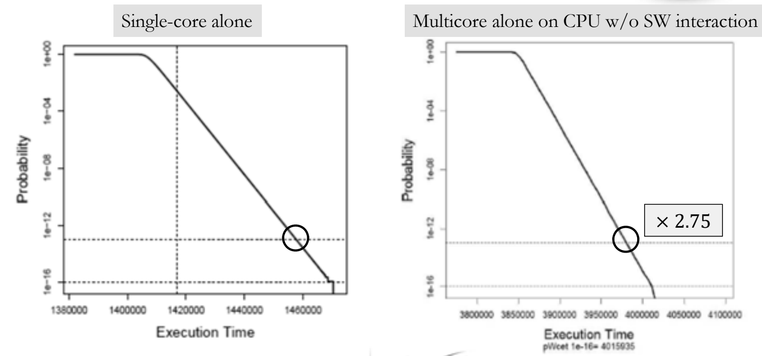
Hardware interference /2

- 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

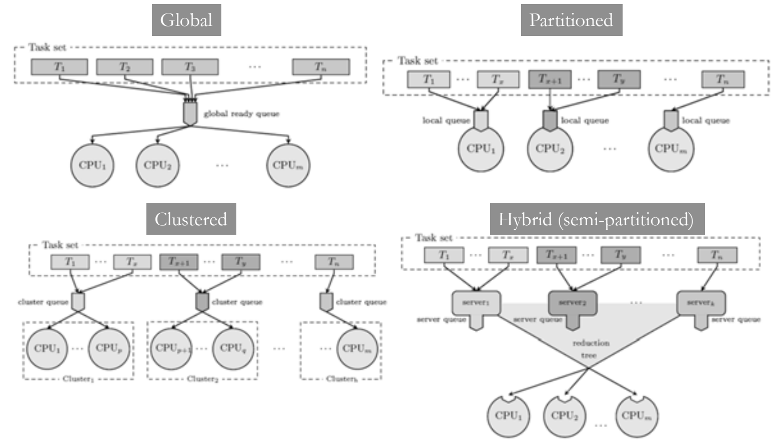
- *Processor (CPU) identity*
  - All processors are equivalent
- *Task independence*
  - Tasks are logically independent of one another
- *Task unity*
  - Tasks have no internal parallelism: they can run only on one CPU at any one time
- *Task migration*
  - Tasks can run on different CPUs at different 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 not true 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 in parallel on all m processors

Software interference /2

- A very pessimistic bound considers all higher-priority tasks to always fully interfere
  - $R_k^{max} = C_k + \frac{1}{m}\sum_{\tau_j \in hp(k)}(\left\lceil \frac{R_k^{max}}{T_j} \right\rceil C_j + Cj)$
- 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

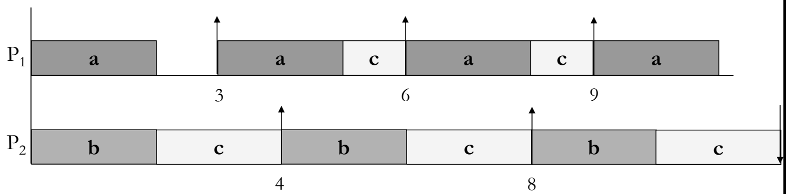
Global scheduling anomalies

- 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

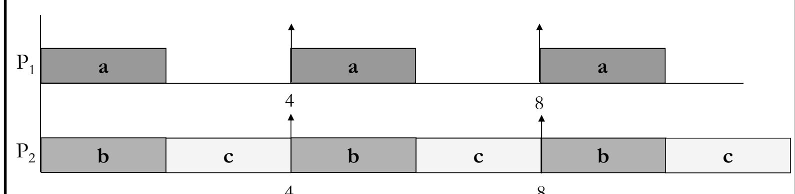
Task	T	D	C	U
a	3	3	2	0.67
b	4	4	2	0.50
c	12	12	8	0.67

$m = 2$ 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)

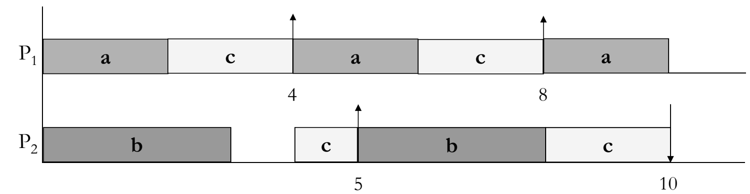
- If we reduce T_a to 4 we *decrease* system load to $U = 1.67$
- But in this way I_c *increases* from 4 to 6 and τ_c misses its deadline (!)



Anomaly 2: decrease in own demand

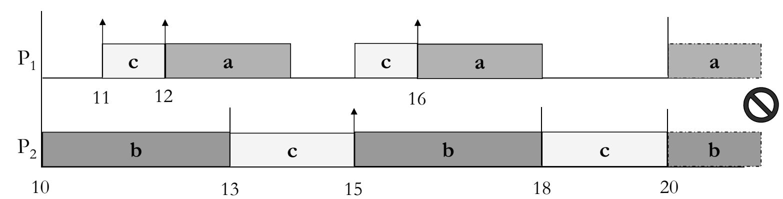
Task	T	D	C	U
a	4	4	2	0.5
b	5	5	3	0.6
c	10	10	7	0.7

$m = 2$ 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



The defeat of greedy schedulers

- Greedy algorithms are easy to explain, study, and implement
  - They work very well on single-core processors
  - EDF [1] and LLF [2] are optimal for single-core processors
- *They collapse the urgency of a job into a single value and use it to greedily schedule jobs*
- Unfortunately (and surprisingly) greedy algorithms fail when used on multiprocessors
  - EDF and LLF are no longer optimal

Why do greedy schedulers fail?

Theorem 1 (stating the obvious)

When the total utilization of a periodic task set is equal to the number of processors, then no feasible schedule can allow any processor to remain idle for any length of time

P-fair scheduling [Baruah et al. 1996]

- *Proportional progress* is a form of proportionate fairness also known as ***P-fairness***
  - Each task τ_i is assigned resources in proportion to its *weight* $W_i = \frac{C_i}{T_i}$ so that it progresses proportionately
  - Useful e.g., for real-time multimedia applications
- At every time t task τ_i must have been scheduled either $\lfloor W_i \times t \rfloor$ or $\lceil W_i \times t \rceil$ time units
  - Without loss of generality, preemption is assumed to only occur at integral time units
  - The workload model is assumed to be periodic

P-fair scheduling /2

- $\boldsymbol{lag}(S, \tau_i, t)$ is the difference between the total resource allocation that task τ_i should have received in $[0, t)$ and what it received under schedule S
- For a P-fair schedule S at time t
  - τ_i is *ahead* iff $\boldsymbol{lag}(S, \tau_i, t) < 0$
  - τ_i is *behind* iff $\boldsymbol{lag}(S, \tau_i, t) > 0$
  - τ_i is *punctual* iff $\boldsymbol{lag}(S, \tau_i, t) = 0$

P-fair scheduling /3

- $\boldsymbol{\alpha}(x)$ is the *characteristic* (infinite) *string* of task τ_x over $\{-, 0, +\}$ for $t \in \mathbb{N}$ with
  - $\boldsymbol{\alpha}_t(x) = \boldsymbol{sign}(W_x \cdot (t+1) - \lfloor W_x \cdot t \rfloor - 1)$
    - Distance from the integral approximation of fluid curve
  - $\boldsymbol{\alpha}(x, t)$ is the *characteristic substring* $\boldsymbol{\alpha}_{t+1}(x)\boldsymbol{\alpha}_{t+2}(x) \dots \boldsymbol{\alpha}_{t'}(x)$ of task τ_x at time t where $t' = min\, i: i > t: \boldsymbol{\alpha}_i(x) = 0$
- For a P-fair schedule S at time t, task τ_i is
  - *Urgent* iff τ_i is *behind* and $\boldsymbol{\alpha_t}(\tau_i) \neq -$
  - *Tnegru* iff τ_i is *ahead* and $\boldsymbol{\alpha_t}(\tau_i) \neq +$
  - *Contending* otherwise

Properties of a P-fair schedule S

- For task τ_i *ahead* at time t under S
  - *tnegru* { If $\boldsymbol{\alpha_t}(\tau_i) = -$ and τ_i not scheduled at t then τ_i is *ahead* at $t+1$
  - *tnegru* { If $\boldsymbol{\alpha_t}(\tau_i) = 0$ and τ_i not scheduled at t then τ_i is *punctual* at $t+1$
  - If $\boldsymbol{\alpha_t}(\tau_i) = +$ and τ_i not scheduled at t then τ_i is *behind* at $t+1$
  - If $\boldsymbol{\alpha_t}(\tau_i) = +$ and τ_i scheduled at t then τ_i is *ahead* at $t+1$
- For task τ_i *behind* at time t under S
  - If $\boldsymbol{\alpha_t}(\tau_i) = -$ and τ_i scheduled at t then τ_i is *ahead* at $t+1$
  - If $\boldsymbol{\alpha_t}(\tau_i) = -$ and τ_i not scheduled at t then τ_i is *behind* at $t+1$
  - *urgent* { If $\boldsymbol{\alpha_t}(\tau_i) = 0$ and τ_i scheduled at t then τ_i is *punctual* at $t+1$
  - *urgent* { If $\boldsymbol{\alpha_t}(\tau_i) = +$ and τ_i scheduled at t then τ_i is *behind* at $t+1$

P-fair scheduling /4

- General principle of P-fairness
  - Every task *urgent* at time t must be scheduled at t to preserve P-fairness
  - No task *tnegru* at time t can be scheduled at t without breaking P-fairness
- Problems with n_0 *tnegru*, n_1 *contending*, n_2 *urgent* tasks at time t, with m resources and $n = n_0 + n_1 + n_2$
  - If $n_2 > m$ the scheduling algorithm cannot schedule all *urgent* tasks
  - If $n_0 > n - m$ the scheduling algorithm is forced to schedule some *tnegru* tasks

P-fair scheduling /5

- The **PF** scheduling algorithm
  - Schedule all *urgent* tasks
  - Allocate the remaining resources to the highest-priority *contending* tasks according to the total order function $\supseteq$ with ties broken arbitrarily
    - $x \supseteq y$ iff $\boldsymbol{\alpha}(x, t) \geq \boldsymbol{\alpha}(y, t)$
    - And the comparison between the characteristics substrings is resolved lexicographically with $- < 0 < +$
- With PF we have $\sum_{x \in [0,n]} W_x = m$
  - A dummy task may need to be added to the task set to top utilization up
- No problem situation can occur with the PF algorithm

Example (PF scheduling) /1

Task	C	T	W
v	1	3	0.333…
w	2	4	0.5
x	5	7	0.714…
y	8	11	0.727…
z	335	462	3-U

- $m = 3$ processors
- $n = 4$ tasks
- τ_z is a dummy task used to top system utilization up
- In general its period is set to the system hyperperiod
  - This time we halved it
- With PF we always have $n_2 > m$ and $n_0 \leq n - m$

Example (PF scheduling) /2

These tasks are scheduled and they become ahead

t	lag × period v	w	x	y	z	characteristic string v	w	x	y	z	urgent tasks	contending tasks	tnegru tasks
0	0	0	0	0	0	–	–	–	–	–	$\{\}$	$y > z > x > w > v$	$\{\}$
1	1	2	−2	−3	−127	–	0	+	+	+	$\{w\}$	$y > z > x > v$	$\{\}$
2	2	0	3	−6	−254	0	–	+	+	+	$\{v, x\}$	$w > y > z$	$\{\}$
3	0	−2	1	2	81	–	0	–	–	–	$\{\}$	$y > z > x > v$	$\{w\}$
4	1	0	−1	−1	−46	–	–	+	+	+	$\{\}$	$y > z > x > v = w$	$\{\}$
5	2	2	−3	−4	−173	0	0	+	+	+	$\{v, w\}$	$y > z > x$	$\{\}$
6	0	0	2	−7	162	–	–	0	+	+	$\{x, z\}$	$w > y > v$	$\{\}$
7	1	−2	0	1	35	–	0	–	–	–	$\{\}$	$y > z > x > v$	$\{w\}$
8	2	0	−2	−2	−92	0	–	+	+	+	$\{v\}$	$y > z > x > w$	$\{\}$
9	0	2	3	−5	−219	–	0	+	+	+	$\{w, x\}$	$y > z > v$	$\{\}$
10	1	0	1	−8	116	–	–	–	0	–	$\{\}$	$z > x > v = w$	$\{y\}$
11	−1	2	−1	0	−11	0	0	+	–	+	$\{w\}$	$y > z > x$	$\{v\}$
12	0	0	4	−3	−138	–	–	+	+	+	$\{x\}$	$y > z > w > v$	$\{\}$
13	1	2	2	−6	−265	–	0	0	+	+	$\{w, x\}$	$v > y > z$	$\{\}$
14	−1	0	0	2	70	0	–	–	–	–	$\{\}$	$y > z > x > w$	$\{v\}$
15	0	2	−2	−1	−57	–	0	+	+	+	$\{w\}$	$y > z > x > v$	$\{\}$
16	1	0	3	−4	−184	–	–	+	+	+	$\{x\}$	$y > z > v = w$	$\{\}$
17	2	2	1	−7	−311	0	0	–	+	+	$\{v, w\}$	$x > y > z$	$\{\}$
18	0	0	−1	1	24	–	–	+	–	–	$\{\}$	$y > z > x > w > v$	$\{\}$
19	1	2	−3	−2	−103	–	0	+	+	+	$\{w\}$	$y > z > v = x$	$\{\}$