

Introduction to Scheduling

Radu Prodan
Institute of Computer Science
University of Innsbruck

What is Scheduling?

- Decision making process that deals with allocation of **resources to tasks**
 - Processing units in a computing environment
 - Routers (to handle packet traffic)
 - Disk drives (I/O scheduling)
 - Printers (print spooler)
 - Embedded systems
 - ...
- Goal is to optimise one or more **objectives**
- A **scheduling problem** can be described by a triplet (α, β, γ)
 - α describes the resource environment
 - β describes the processing characteristics and constraints
 - γ describes the objective
 - *[Pinedo, Scheduling: Theory, Algorithms, and Systems, 1994]*

Resource Environments α

- Single machine
- Identical machines in parallel
- Machines in parallel with different speeds
- Unrelated machines in parallel

- Flow shop
- Flexible flow shop
- Job shop
- Flexible job shop
- Open shop

18.01.2011

R. Prodan, Introduction to Scheduling

3

Processing Characteristics and Constraints

 β

- Release dates
- Preemptions
- Precedence constraints (workflows)
- Sequence dependent setup times (file transfers)
- Job families
- Batch processing

- Breakdowns
- Machine eligibility restrictions

- Permutation (FIFO in flow shops)
- Blocking (in flow shops)
- Nowait (in flow shops)
- Recirculation (in job shops)

18.01.2011

R. Prodan, Introduction to Scheduling

4

Objectives Functions γ

- Makespan (execution time)
- Turnaround (completion time)
- Response time
- Throughput
- Waiting time
- Communication time
- Load balance
- Processor utilisation
- Reliability
- Energy
- Economic cost

18.01.2011

R. Prodan, Introduction to Scheduling

5

Agenda

- **Introduction**
- Parallel and distributed applications
- Compiler construction
- Operating systems

18.01.2011

R. Prodan, Introduction to Scheduling

6

Agenda

- Introduction
- **Parallel and distributed applications**
- Compiler construction
- Operating systems

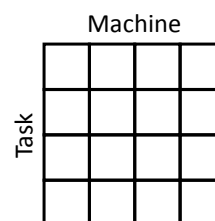
18.01.2011

R. Prodan, Introduction to Scheduling

7

Scheduling Problem

- Resource environment
 - M unrelated machines in parallel
- Processing restrictions and constraints
 - N tasks
 - Independent (parameter studies)
 - Workflow
- Objective function
 - Execution time, ...
- Prediction matrix
 - Objective function estimation for each task

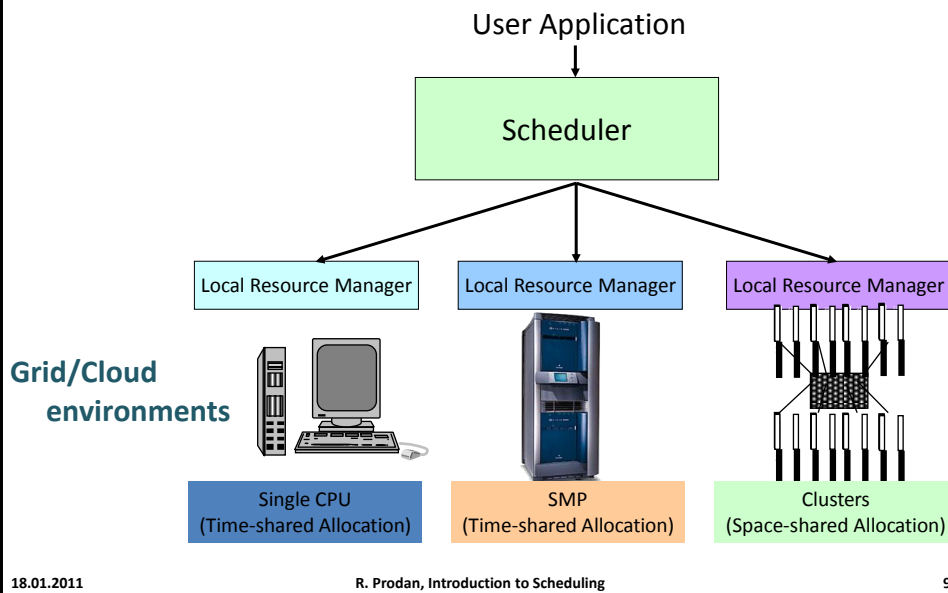


18.01.2011

R. Prodan, Introduction to Scheduling

8

Conventional Scheduling Architecture



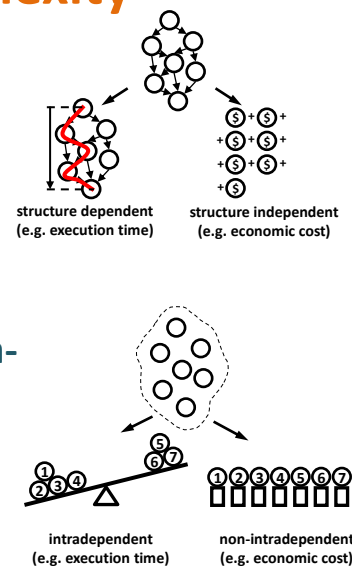
Problem Complexity

■ Structure dependent and intradependent objectives

- Makespan (execution time)
- NP-complete (Ullman, 75)
- $O(M^N)$ complexity

■ Structure independent and non-intradependent objectives

- Cost (simple models)
- Reliability
- Energy consumption



ETC Matrix

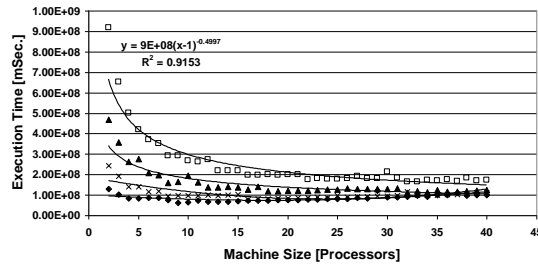
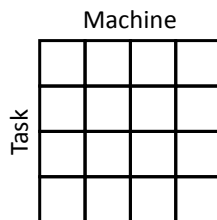
- Expected Time to Compute matrix

$$T = \frac{\text{Work}}{\text{Speed}}$$

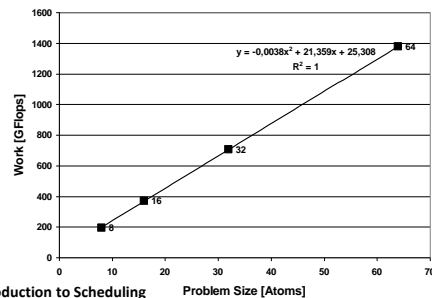
$$T_{\text{comp}} = \frac{\text{MIPS}}{\text{Clock_freq}}$$

$$T_{\text{comm}} = \frac{\text{Data_size}}{\text{Bandwidth}}$$

- Uncertainties, inaccuracies, incomplete information, ...



8 Atoms, $y = 83756(x-1)^2 - 3E+06(x-1) + 1E+08$ 16 Atoms, $y = 188062(x-1)^2 - 9E+06(x-1) + 2E+08$
 32 Atoms, $y = 4E+08(x-1)-0.3935$ 64 Atoms, $y = 9E+08(x-1)-0.4997$



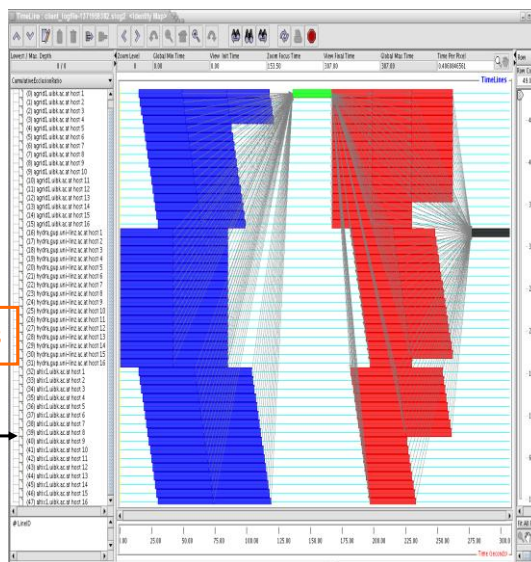
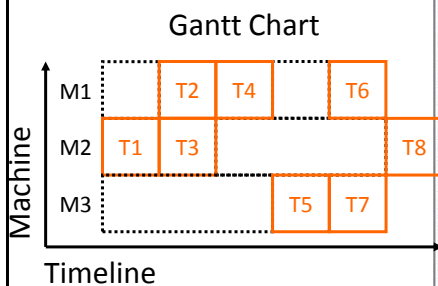
18.01.2011

R. Prodan, Introduction to Scheduling

11

Gantt Chart

- The execution time plan



18.01.2011

R. Prodan, Introduction to Scheduling

12

Independent Task Scheduling

- $O(N \cdot M)$ heuristics
 - Minimum Execution Time (MET)
 - Minimum Completion Time (MCT)
 - Switching Algorithm (SA)
 - K-Percent Best (K-PB)
 - Opportunistic Load Balancing (OLB)
- $O(N^2 \cdot M)$ heuristics
 - Min-min
 - Min-max
 - Sufferage

18.01.2011

R. Prodan, Introduction to Scheduling

13

Min-Min Heuristic — $O(N^2 \cdot M)$

- Determine the MCT for all the tasks
- Schedule only the task with the minimum MCT and reconsider the rest

algorithm min-min(task_list, machine_list)

GanttChart = Φ

while task_list $\neq \Phi$

foreach task \in task_list

 (machine, time) = MCT(task, machine_list, GanttChart)

 MCT_list = MCT_list \cup (task, machine, time)

end foreach

 (task_s, machine_s) = min_{time} MCT_list

 GanttChart = GanttChart \cup (task_s, machine_s)

 task_list = task_list – task_s

end while

return GanttChart

M1

2	2	2
---	---	---

M2

2	2	2	12
---	---	---	----

18.01.2011

R. Prodan, Introduction to Scheduling

14

Min-Max Algorithm — $O(N^2 \cdot M)$

algorithm min-max(task_list, machine_list)

GanttChart = Φ

while task_list $\neq \Phi$

foreach task \in task_list

 (machine, time) = MCT(task, machine_list, GanttChart)

 MCT_list = MCT_list \cup (task, machine, time)

end foreach

 (task_s, machine_s) = \max_{time} MCT_list

 GanttChart = GanttChart \cup (task_s, machine_s)

 task_list = task_list - task_s

end while

return GanttChart

M1	2	2	2	2	2	2
M2	12					

18.01.2011

R. Prodan, Introduction to Scheduling

15

Sufferage Heuristic — $O(N^2 \cdot M)$

algorithm sufferage(task_list, machine_list)

GanttChart = Φ

while task_list $\neq \Phi$

foreach task \in task_list

 (machine, time) = MCT(task, machine_list, GanttChart)

 (machine2, time2) = $2^{\text{nd_MCT}}$ (task, Grid, GanttChart)

 MCT_list = MCT_list \cup (task, machine, time2 - time)

end foreach

 { (task_s, machine_s) } = { \max_{time} MCT_list }

 GanttChart = GanttChart \cup { (task_s, machine_s) }

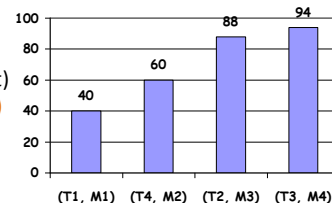
 task_list = task_list - { task_s }

end while

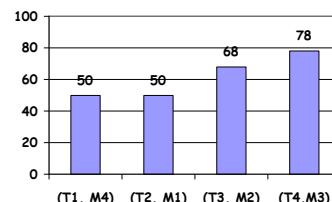
return GanttChart

	M1	M2	M3	M4
T1	40	48	134	50
T2	50	82	88	89
T3	55	68	94	93
T4	52	60	78	108

Min-Min



Sufferage



18.01.2011

R. Prodan, Introduction to Scheduling

16

Multi-Criteria Workflow Scheduling

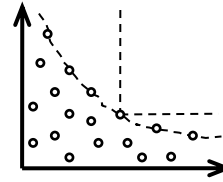
- Optimise multiple objectives simultaneously

- Makespan
- Reliability
- Energy
- Economic cost
- ...

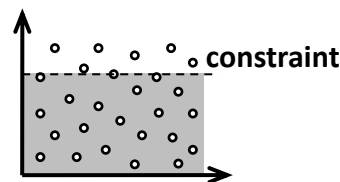
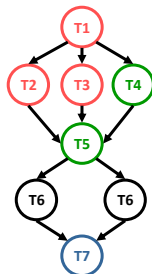
- Pareto curve

- Aggregate objective function

- Constrained optimization



$$\min(\text{clock} + \text{dollar} \cdot \%)$$



18/01/2011

R. Prodan, Introduction to Scheduling

17

Bi-Criteria Scheduling

- Goal:** optimise two contradictory criteria

- Two phase Dynamic Constraint Algorithm

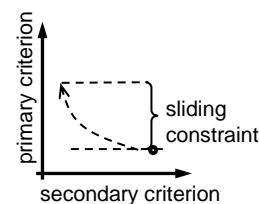
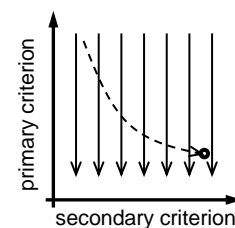
- **Primary scheduling** phase optimises the primary criterion

- ✓ Heterogeneous Earliest Finish Time (HEFT)

- ✓ Preliminary solution: (c_1^{PREL}, c_2^{PREL})

- **Secondary scheduling** phase optimises the secondary criterion

- ✓ Keep the primary objective within a **sliding constraint**



$$|c_1^{FINAL} - c_1^{PREL}| \leq L, \text{ where } L = f(c_1^{PREL}) \cdot p\% \cdot c_1^{PREL}$$

18.01.2011

R. Prodan, Introduction to Scheduling

18

Multiple Choice Knapsack Problem

Multiple Choice Knapsack Problem	Bi-Criteria Scheduling
n classes of items: $S = \{ S_1, \dots, S_n \}$	Workflow tasks
Set of items in each class: $S_i = \{ e_{i1}, \dots, e_{im_i} \}$	Set of machines
Price of an item: $p(e_{ij})$	Primary criterion
Weight of an item: $w(e_{ij})$	Secondary criterion
Find solution: $sol = \{ s_1, \dots, s_n \}, s_i \in S_i$	Workflow schedule
Maximise total price: $\sum_{i=1}^n p(s_i)$	Primary scheduling
Keep the weight below a limit L : $\sum_{i=1}^n w(s_i) \leq L$	Secondary scheduling

18.01.2011

R. Prodan, Introduction to Scheduling

19

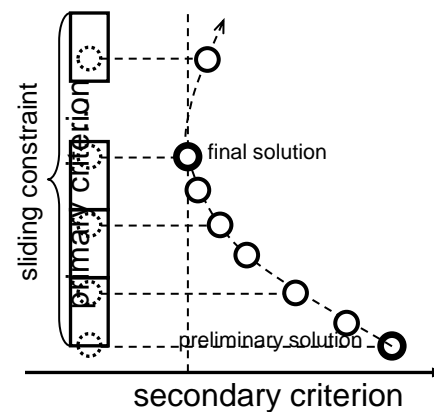
Secondary Scheduling

Dynamic programming

```

mem_table = { solPREL }
for each activity a ∈ workflow
  for each sol ∈ mem_table
    for each machine m ∈ machine_list
      sol' = modify(sol, a, m)
      if c1(sol') ∈ [c1PREL, c1PREL + L]
        ∧ sol' is not dominated
      then insert(sol', mem_table)
      remove all sol dominated by sol'
return minc2(sol')

```



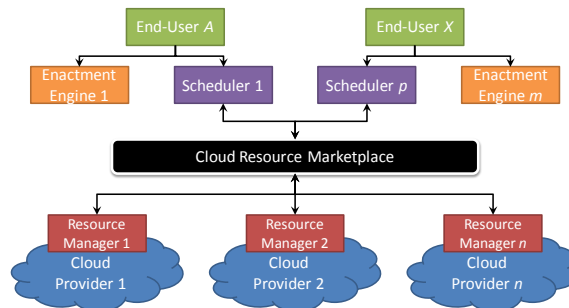
18.01.2011

R. Prodan, Introduction to Scheduling

20

Game Theoretic Approach?

- Multiple schedulers representing end-user's objectives
 - Minimise execution time, cost
 - Maximise reliability
- Multiple resource managers representing resource provider's objectives
 - Maximise income, throughput, utilisation, fairness
 - Minimise energy consumption
- Single or multi-criteria objective for each participant
- Game theoretic negotiation



18.01.2011

R. Prodan, Introduction to Scheduling

21

Agenda

- Introduction
- Parallel and distributed applications
- **Compiler construction**
- Operating systems

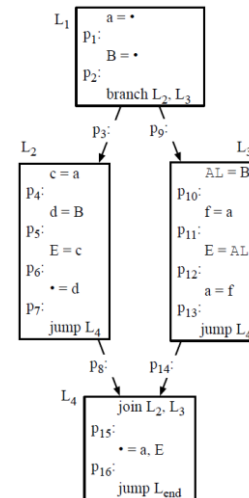
18.01.2011

R. Prodan, Introduction to Scheduling

22

Register Allocation

- Map program variables to machine registers
- Build a control flow graph of basic program blocks
- Two types of variable references
 - Use variables
 - Define variables
- Program point between two consecutive instructions
- v is **alive** at program point p if
 - There is a path from p to a use variable v
 - v is not redefined in between
- **Live range** = set of points where a variable is alive
 - **b**: p_2, p_3, p_4, p_9
 - **a**: $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{13}, p_{14}, p_{15}$



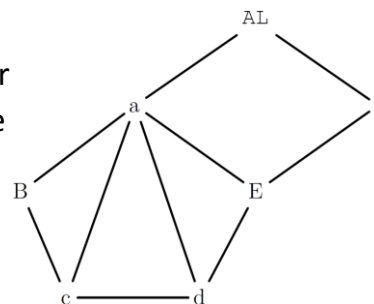
18.01.2011

R. Prodan, Introduction to Scheduling

23

Interference Graph

- Two variables interfere if their live range is not empty
- Nodes of the graph are variables
- Edges connect variables that interfere with one another
- Graph colouring problem
 - Each register has a different colour
 - Colour each node according to the register assigned to the corresponding variable
 - Two adjacent colours are forbidden



18.01.2011

R. Prodan, Introduction to Scheduling

24

Interference Graph

Instructions Live vars

$b = a + 2$

$c = b * b$

$b = c + 1$

return $b * a$

18.01.2011

R. Prodan, Introduction to Scheduling

25

Interference Graph

Instructions Live vars

$b = a + 2$

$c = b * b$

$b = c + 1$

return $b * a$

b, a

18.01.2011

R. Prodan, Introduction to Scheduling

26

Interference Graph

Instructions	Live vars
--------------	-----------

$b = a + 2$	
-------------	--

$c = b * b$	
-------------	--

	a,c
--	-----

$b = c + 1$	
-------------	--

	b,a
--	-----

return $b * a$	
----------------	--

18.01.2011

R. Prodan, Introduction to Scheduling

27

Interference Graph

Instructions	Live vars
--------------	-----------

$b = a + 2$	
-------------	--

	b,a
--	-----

$c = b * b$	
-------------	--

	a,c
--	-----

$b = c + 1$	
-------------	--

	b,a
--	-----

return $b * a$	
----------------	--

18.01.2011

R. Prodan, Introduction to Scheduling

28

Interference Graph

Instructions	Live vars
	a
$b = a + 2$	b,a
$c = b * b$	a,c
$b = c + 1$	b,a
return $b * a$	

18.01.2011

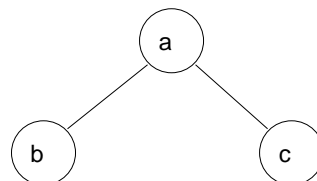
R. Prodan, Introduction to Scheduling

29

Interference Graph

Instructions	Live vars
	a
$b = a + 2$	a,b
$c = b * b$	a,c
$b = c + 1$	a,b
return $b * a$	

colour	register
	eax
	ebx



18.01.2011

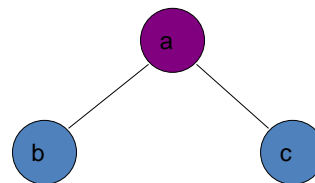
R. Prodan, Introduction to Scheduling

30

Interference Graph

Instructions	Live vars
	a
$b = a + 2$	a,b
$c = b * b$	a,c
$b = c + 1$	a,b
return $b * a$	

colour	register
	eax
	ebx



18.01.2011

R. Prodan, Introduction to Scheduling

31

Graph Colouring


- NP-complete problem
 - Fast and good heuristics required
- Kempe's algorithm [1879]
- **Step 1: Simplify**
 - Find a node with at **most K-1 edges**, remove it from the graph, push it onto a stack, and recurse
 - If the graph cannot be coloured, it will be simplified to graph in which every node has at least K neighbours (sometimes still K-colourable)
- **Step 2: Colour**
 - After the simplified sub-graph has been coloured, add back the node on the top of the stack and assign it a colour not taken by the adjacent nodes

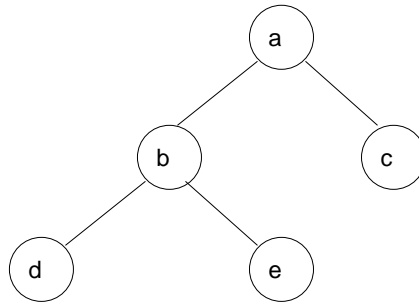
18.01.2011

R. Prodan, Introduction to Scheduling

32

Colouring

colour	register
	eax
	ebx



stack:

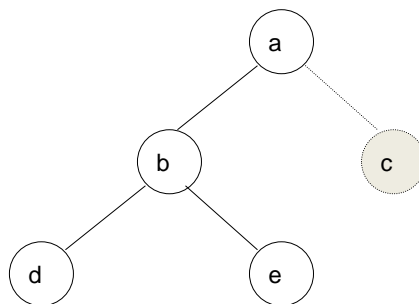
18.01.2011

R. Prodan, Introduction to Scheduling

33

Colouring

colour	register
	eax
	ebx



stack:

c

18.01.2011

R. Prodan, Introduction to Scheduling

34

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

a

b

c

d

e

stack:

e
c

18.01.2011

R. Prodan, Introduction to Scheduling

35

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

a

b

c

d

e

stack:

a
e
c

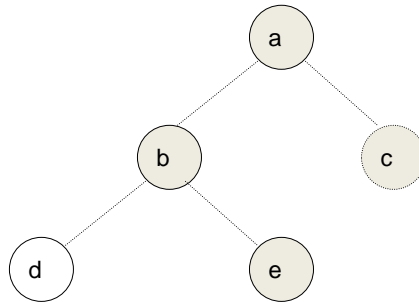
18.01.2011

R. Prodan, Introduction to Scheduling

36

Colouring

colour	register
	eax
	ebx



stack:
b
a
e
c

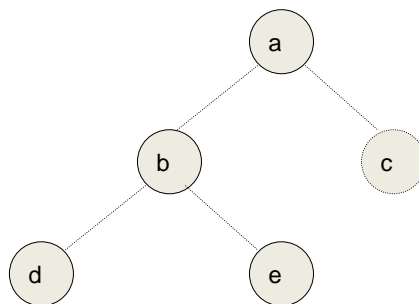
18.01.2011

R. Prodan, Introduction to Scheduling

37

Colouring

colour	register
	eax
	ebx



stack:
d
b
a
e
c

18.01.2011

R. Prodan, Introduction to Scheduling

38

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

a

b

c

d

e

stack:
b
a
e
c

18.01.2011

R. Prodan, Introduction to Scheduling

39

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

a

b

c

d

e

stack:
a
e
c

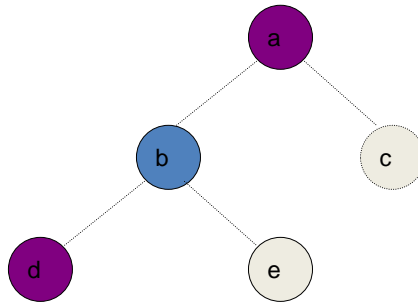
18.01.2011

R. Prodan, Introduction to Scheduling

40

Colouring

colour	register
	eax
	ebx



stack:

e
c

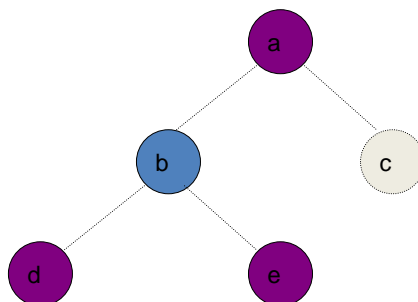
18.01.2011

R. Prodan, Introduction to Scheduling

41

Colouring

colour	register
	eax
	ebx



stack:

c

18.01.2011

R. Prodan, Introduction to Scheduling

42

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

a

b

c

d

e

stack:

18.01.2011

R. Prodan, Introduction to Scheduling

43

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

a

b

c

d

e

stack:

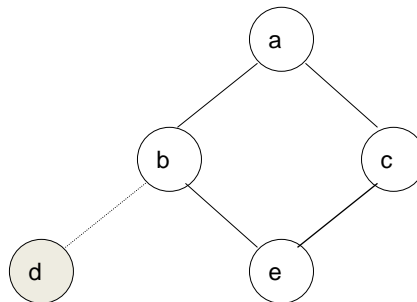
18.01.2011

R. Prodan, Introduction to Scheduling

44

Colouring

colour	register
	eax
	ebx



stack:
d

all nodes have
2 neighbours!

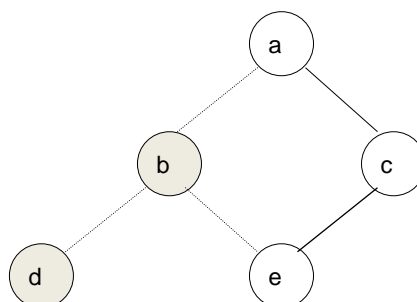
18.01.2011

R. Prodan, Introduction to Scheduling

45

Colouring

colour	register
	eax
	ebx



stack:

b
d

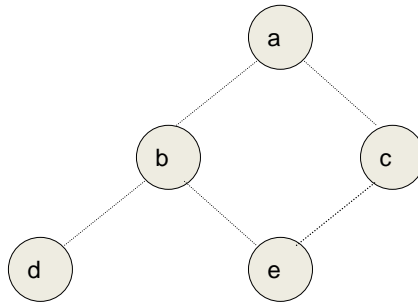
18.01.2011

R. Prodan, Introduction to Scheduling

46

Colouring

colour	register
	eax
	ebx



stack:
c
e
a
b
d

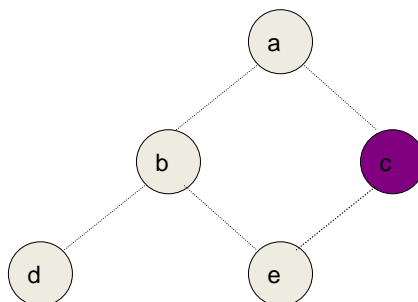
18.01.2011

R. Prodan, Introduction to Scheduling

47

Colouring

colour	register
	eax
	ebx



stack:
e
a
b
d

18.01.2011

R. Prodan, Introduction to Scheduling

48

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

stack:
a
b
d

18.01.2011

R. Prodan, Introduction to Scheduling

49

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

stack:
b
d

18.01.2011

R. Prodan, Introduction to Scheduling

50

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

stack:
d

18.01.2011

R. Prodan, Introduction to Scheduling

51

Colouring

colour	register
<div></div>	eax
<div></div>	ebx

stack:

We got lucky!

18.01.2011

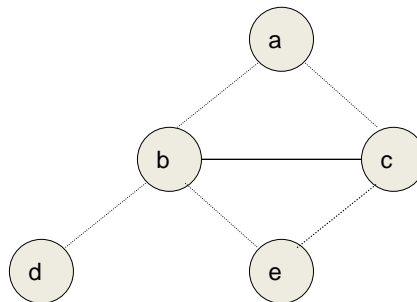
R. Prodan, Introduction to Scheduling

52

Colouring

colour	register
	eax
	ebx

Some graphs cannot be coloured:



stack:

c
b
e
a
d

18.01.2011

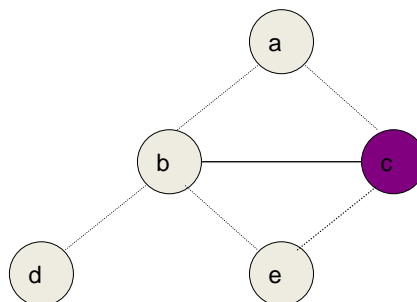
R. Prodan, Introduction to Scheduling

53

Colouring

colour	register
	eax
	ebx

Some graphs cannot be coloured:



stack:

b
e
a
d

18.01.2011

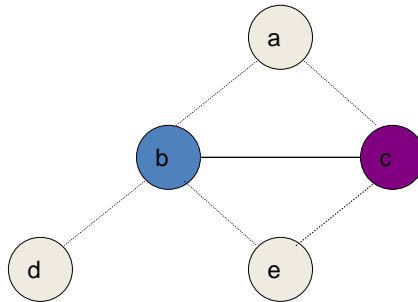
R. Prodan, Introduction to Scheduling

54

Colouring

colour	register
	eax
	ebx

Some graphs cannot be coloured:



stack:

e
a
d

18.01.2011

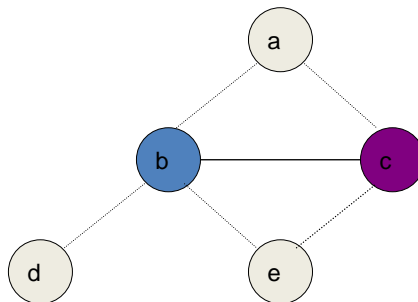
R. Prodan, Introduction to Scheduling

55

Colouring

colour	register
	eax
	ebx

Some graphs cannot be coloured:



stack:

e
a
d

no colours left for e!

18.01.2011

R. Prodan, Introduction to Scheduling

56

Chaitin's Algorithm

- **Renumber**
 - Discover live range information in the source program
- **Build**
 - Build the interference graph.
- **Coalesce**
 - Merge the live ranges of non-interfering variables related by copy instructions ($a:=b$)
- **Spill cost**
 - Compute the spill cost of each variable
- **Simplify**
 - Kempe's colouring algorithm
- **Spill Code**
 - Insert loads and stores to commute values between registers and memory

18.01.2011

R. Prodan, Introduction to Scheduling

57

Agenda

- Introduction
- Parallel and distributed applications
- Compilers
- **Operating systems**

18.01.2011

R. Prodan, Introduction to Scheduling

58

Operating Systems Scheduling

- **Goal**
 - Load balancing on **shared memory systems** (SMP and multi-core)
 - Multiplex a **single CPU** for multiple processes
- **Scheduling policy**
 - When it is time for a process to be removed from the CPU?
 - Which ready process should be allocated to the CPU next?
- **Enqueuer**
 - Places a pointer to a process descriptor into the ready list
- **Context switcher**
 - Saves the contents of all processor registers for the process removed from the CPU in its process descriptor
- **Dispatcher**
 - Selects one of the ready processes enqueued in the ready list and allocates the CPU

Preemptive versus Non-Preemptive Scheduling

- **Non-preemptive scheduling**
 - Process calls the **yield()** system call to release the CPU
 - Multiple processes could simultaneously yield to the scheduler
 - Scheduler selects and yields to one of the processes
- **Preemptive scheduling**
 - Interrupt system enforces periodic involuntary interruption of any process using an **interval timer**
 - Invokes the scheduler and **yield()**
 - Each process gets to run in units of **time slices** (may be less than the interval time)

```
yield(r, s) {
    memory[r] = PC;
    PC = memory[s];
}
```

- *r* and *s* are usually a function of the process's id

Simple Algorithms

Scheduling algorithm	CPU Utilization	Throughput	Turnaround time	Response time	Deadline handling	Starvation free
First In First Out	Low	Low	High	Low	No	Yes
Shortest Job First	Medium	High	Medium	Medium	No	No
Priority-based scheduling	Medium	Low	High	High	Yes	No
Round-robin scheduling	High	Medium	Medium	High	No	Yes
Multilevel queue scheduling	High	High	Medium	Medium	Low	Yes

18.01.2011

R. Prodan, Introduction to Scheduling

61

Multiple Level Feedback Queue

- Linux 2.6
- Minimise response and turnaround times **without knowing the service time**
- 140 priority queues
 - Static priorities 0—99 for real-time processes
 - Static priorities 100—139 for normal processes set via **nice** system call
 - Round-robin scheduling within each queue level
- **O(1)** insertion and selection time
 - A new process is positioned at the end of the top-level queue
 - Execute the process with the highest priority
- Dynamic priority calculation and adjustment
 - Preempt and move jobs at the end of next priority queue if they consume their time slice
 - Move I/O blocking jobs to higher priority queues
- Drawbacks
 - CPU share not easy to calculate
 - Possible to cheat the scheduler

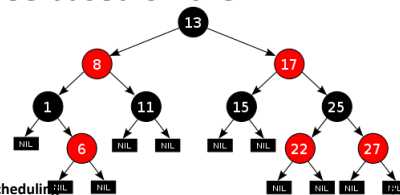
18.01.2011

R. Prodan, Introduction to Scheduling

62

Completely Fair Scheduler

- Linux 2.6.23
- One single run queue
- ***“wait_runtime”*** value for each task
 - Time the task should run to become completely fair and balanced
 - Always zero on “ideal” hardware
- Always run the task with the largest *wait_runtime* for its time slice
- Subtract the executed time slice from *wait_runtime* (minus the “fair share”)
- Organise the tasks in a red-black tree based on their *wait_runtime*
 - $O(\log n)$ worst case search time



18.01.2011

R. Prodan, Introduction to Scheduling

Shared Memory Multiprocessor Scheduling

- Linux 2.6
- Separate run queue for each processor
- Each processor only selects processes from its own queue to run
- Queues are periodically rebalanced (every 200 ms)
- No cache awareness
- Processes with core affinity

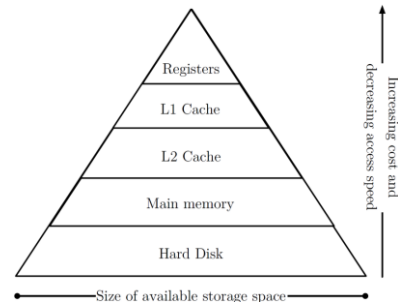
18.01.2011

R. Prodan, Introduction to Scheduling

64

Game Theoretic Scheduling?

- Different processes as game players with different resource requirements
 - Thousands to millions
- Heterogeneous hardware
 - Hundreds of heterogeneous cores
 - Different latencies to memory modules
 - ✓ On-chip shared memories
 - ✓ Distributed shared memory



18.01.2011

R. Prodan, Introduction to Scheduling

65

Conclusions

- Parallel and distributed applications
- Compilers
- Operating systems
- **Starting point for some joint research?**

18.01.2011

R. Prodan, Introduction to Scheduling

66