Allocation, Assignment and Scheduling

Kris Kuchcinski

Krzysztof.Kuchcinski@cs.lth.se

Allocation of System Components

- Defines an architecture by selecting hardware resources which are necessary to implement a given system.
- The components can be, for example, microprocessors, micro-controllers, DSP's, ASIP's, ASIC's, FPGA's, memories, buses or point-to-point links.
- Usually made manually with a support of estimation tools.
- In simple cases can be performed automatically using optimization strategy.

Assignment of System Components

- After allocation the partitioning of system functionality to selected components can be done.
- The partitioning defines the *assignment* of tasks to particular components.
- If there is number of tasks assigned to the same component, which does not support parallel execution, the execution order need to be decided — task *scheduling*.

Scheduling

- Depending on the *computation model* scheduling can be done off-line or during run-time.
- *Static* vs. *dynamic* scheduling.
- RTOS support for dynamic scheduling.
- Scheduling can address advanced execution techniques, such as software pipelining.
- Can be applied to tasks allocated to hardware, software as well as hardware operations and software instructions.

Scheduling

- Data-flow scheduling (SDF, CSDF)
  - static assignment of the instants at which the execution takes place,
  - *time-constrained* and *resource-constrained*,
  - typical for DSP applications (hw and sw).
- Real-time scheduling
  - periodic, aperiodic and sporadic tasks,
  - independent or data-dependent tasks,
  - based on priorities (static or dynamic).

Scheduling Approaches

- Static scheduling
  - static cycling scheduling
- Dynamic scheduling
  - fixed priorities — e.g., rate monotonic
  - dynamic priorities — e.g., earliest deadline first

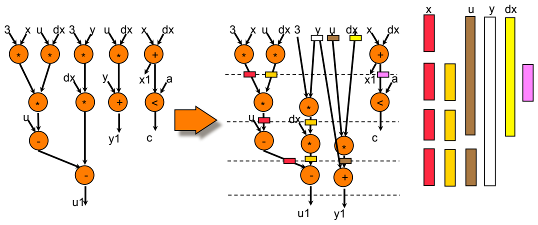
High-Level Synthesis Scheduling

Synthesis of the following code

(inner loop of differential equation integrator)

```
while c do
    begin
    x1 := x + dx;
    u1 := u - ( 3 * x * u * dx) - (3 * y * dx);
    y1 := y + ( u * dx);
    c = x < a;
    x := x1; u := u1; y := y1;
    end;
```

High-Level Synthesis Scheduling (cont'd)



data-flow graph

scheduled data-flow graph

register allocation

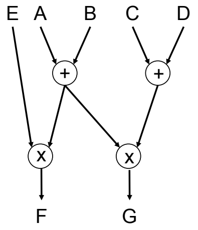
HLS typical process

- Behavioral specification
  - Language selection
  - Parallelism
  - Synchronization

```
procedure example;
    var a, b, c, d, e, f, g : integer;
    begin
        read(a, b, c, d, e);
        f := e * (a + b);
        g := (a + b) * (c + d);
        ...
    end;
```

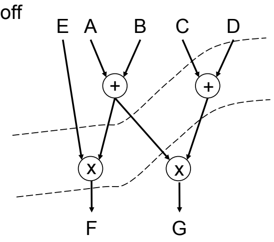
HLS typical process (cont'd)

- **Design Representation**
  - parsing techniques
  - data-flow analysis
  - parallelism extraction
  - program transformations
    - elimination of high-level constructs
    - loop unrolling
    - subexpression detection



HLS typical process (cont'd)

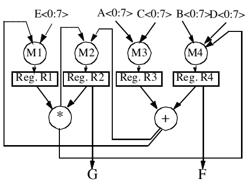
- **Operation Scheduling**
  - Parallelism/cost trade-off
  - Performance measure
  - Clocking strategy



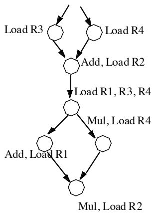
HLS typical process (cont'd)

- **Data Path Allocation**
  - Operation selection
  - Register/Memory allocation
  - Interconnection Generation
  - Hardware Minimization

Data Path Allocation



Data part

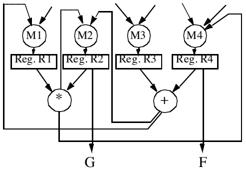


Control part

HLS typical process (cont'd)

- **Control Allocation**
  - Selection of control style (PLA, random logic, etc.)
  - Clock implementation
  - Signal/condition design

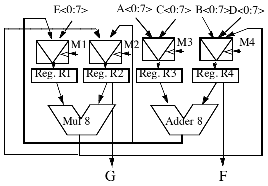
Control Allocation



S0: Start **next** S1;
S1: M3, Load R3, M4=0, Load R4 **next** S2;
S2: Add, ¬M2, Load R2, ¬M1, Load R1, ¬M3, Load R3, M4=1, Load R4 **next** S3;
S3: Add, M1, Load R1, Mul, M4=2, Load R4 **next** S4;
S4: Mul, M2, Load R2 **next** ...

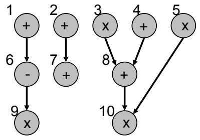
Control description (FSM)

Module Binding

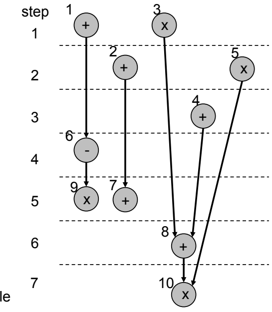


Control ROM
0000: 11000000 0001
0001: 00100000 0010
0010: 00011000 0011
0011: 01000000 0100
...

ASAP Scheduling

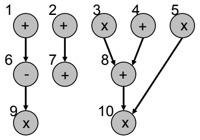


DFG

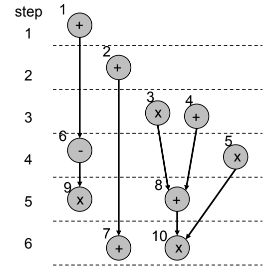


ASAP schedule

ALAP Scheduling



DFG

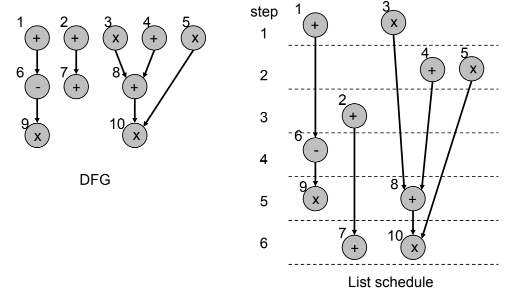


ALAP schedule

List Scheduling

- Constructive scheduling algorithm which selects operation to be assigned to control steps based on a *priority function*.
- Priority function can be different in different versions of list scheduling algorithms:
  - higher priority to operations with low mobility, or
  - higher priority to operations with more immediate successors,
  - length of the path from the operation to the end of the block,
  - ...

List Scheduling



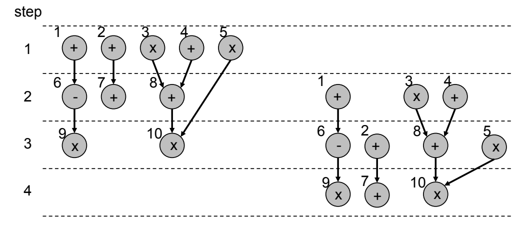
DFG

List schedule

Force-Directed Scheduling

- The basic strategy is to place similar operations in different control steps to balance the concurrency of the operations without increasing the total execution time.
- By balancing the concurrency of operations, it is ensured that each functional unit has a high utilization and therefore the total number of units required is decreased.
- Three main steps:
  - determine the time frame of each operation,
  - create a distribution graph, and
  - calculate the force associated with each assignment.

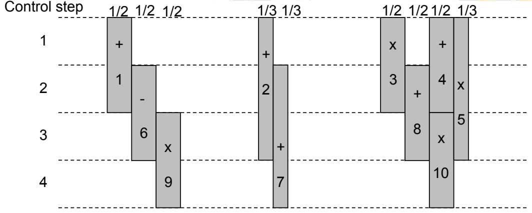
Time Frame of Each Operation



ASAP schedule

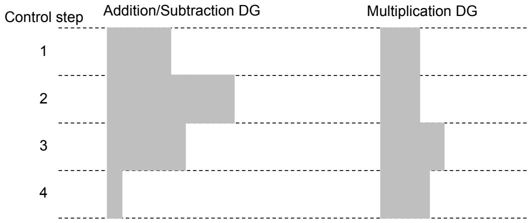
ALAP schedule

Distribution Graph



$DG_{mult}(1) = 1/2 + 1/3 = 0.833$
$DG_{mult}(2) = 1/2 + 1/ 3 = 0.833$
$DG_{mult}(3) = 1/2 + 1/2 + 1/3 = 1.333$
$DG_{mult}(4) = 1/2 + 1/2 = 1$

Distribution Graph (cont'd)



Force Calculation

- the force associated with the tentative assignment of an operation to c-step *j* is equal to the difference between the distribution value in that c-step and the average of the distribution values for the c-steps bounded by the operation's time frame.

$$Force(j) = DG(j) - \sum_{i=f}^{t} \frac{DG(i)}{t-(f+1)}$$

  - assignment of operation 10 to control step 3

Force(3) = DG_{mult}(3) - average DG_{mult} value over time frame of operation 10 = 1.333 - (1.333 + 1)/2 = 0.167, *Force*(4) = -0.167

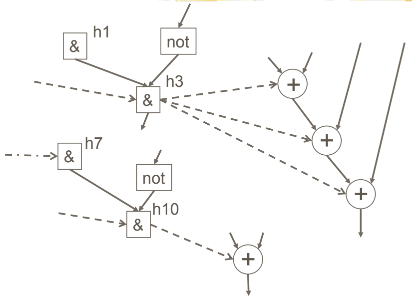
Forced Directed Scheduling Algorithm

- Once all the forces are calculated, the operation-control step pair with the lowest force is scheduled.
- The distribution graphs and forces are then updated and the above process is repeated until all operations are scheduled.

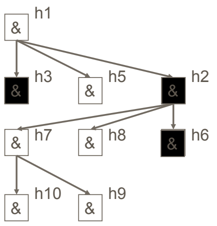
Advanced Scheduling Topics

- Control constructs
  - conditional sharing — sharing of resources for mutually exclusive operations,
  - speculative execution.
- Chaining and multi-cycling.
- Pipelined components.
- Pipelining.
- Registers and memory allocation.
- Low-power issues.

Conditional Dependency Graph



Guard Hierarchy and Mutually Exclusive Guards

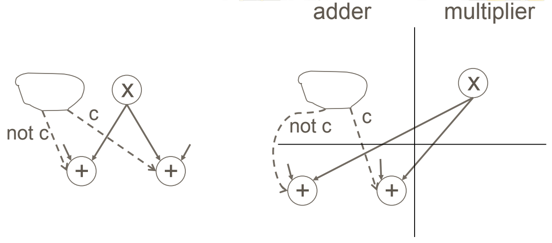


Guard Hierarchy

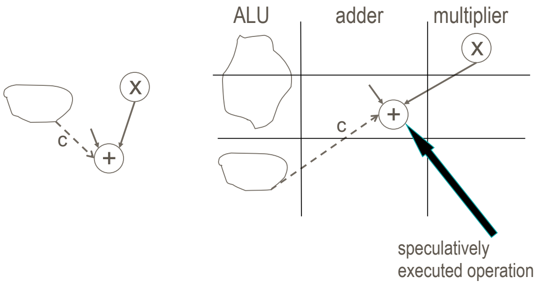
h10	h3, h6, h9
h1	
h2	h3
h3	h10, h2, h6, h7, h8, h9
h5	h9
h6	h10, h3, h7, h9
h7	h3, h6
h8	h3, h9
h9	h10, h3, h5, h6, h8

Mutually Exclusive Guards

Conditional Resource Sharing

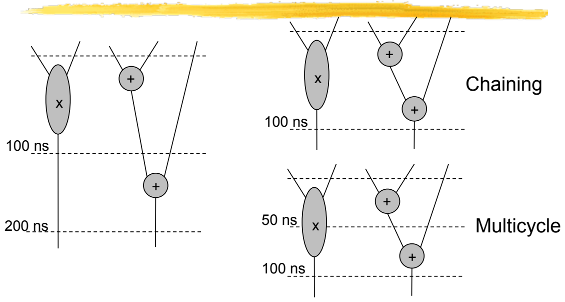


Speculative Execution

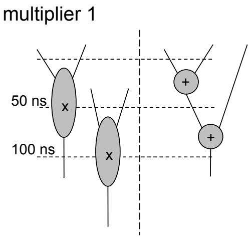


speculatively
executed operation

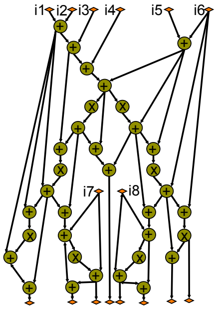
Chaining and multicycling



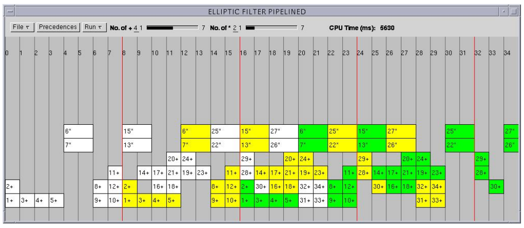
Pipelined Components



Fifth Order Elliptic Wave Filter DFG



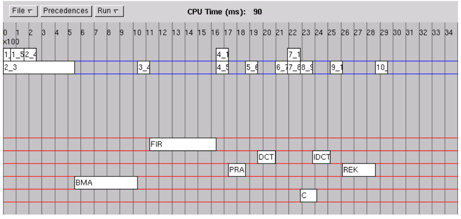
Pipelining — Example


ELLIPTIC FILTER PIPELINED

No. of + 4 1 7 No. of * 2 1 7

CPU Time (ms): 5630

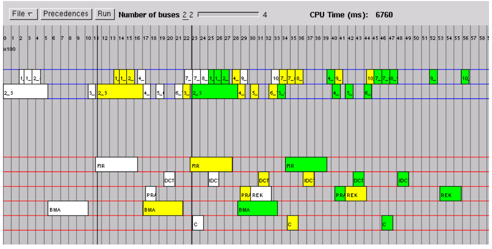
Scheduling of H.261 Video Coding Algorithm



CPU Time (ms): 90

$T_{exec} = 2963$, $T_{10exec} = 29630$

Scheduling of H.261 Video Coding Algorithm



$T_{exec} = 3373$, $T_{pipe\ init} = 1154$, $T_{10exec} = 13759$

Real-Time Scheduling

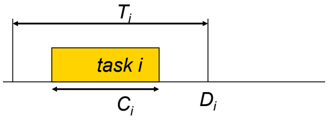
- System is a set of tasks
  - tasks have known execution times (usually WCET)
- Tasks are enabled by repeating events in the environment
  - some known timing constraints
- Task executions must satisfy timing requirements
- Single processor (hard)
  - multiprocessors — much harder, mostly negative results (e.g. NP-hardness)

Scheduling Policies

- Static (pre-run-time, off-line)
  - round-robin,
  - static cyclic.
- Dynamic (run-time, on-line)
  - static priority,
  - dynamic priority,
  - preemptive or not.

Assumptions and Definitions

- Fixed number of periodic and independent tasks.
- Parameters:
  - execution period — *T*
  - worst-case execution time — *C*
  - execution deadline — *D*
  - usually assumed that *T=D*



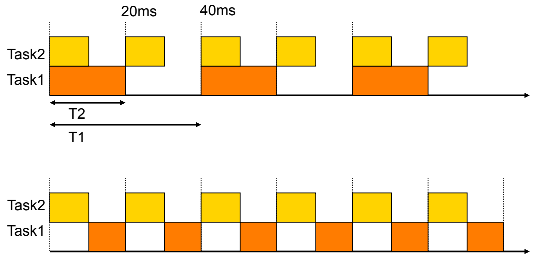
Static Scheduling

- **Round-robin**:
  - pick an order of tasks, e.g. A B C D,
  - execute them forever in that order.
- **Static cyclic**:
  - pick a sequences of tasks, e.g. A B C B D,
  - execute that sequence forever.
- Much like static data-flow.
- If there are arbitrary task periods the schedule duration is equal *least common multiplier* (LCM) of task periods (MPEG 44.1 kHz * 30 Hz requires 132 300 repetitions).
- Problem with sporadic tasks.

Dynamic Scheduling — Static Priorities

- Priorities are assigned to tasks off-line.
- A task with the highest priority is always executed among enabled tasks.
- Single processor execution.
- Preemptive schedule.
- **Rate Monotonic Scheduling (RMS)** — a task with a shorter period is assigned a higher priority.
- RMS is *optimal* — no fixed priority scheme does better.

Rate Monotonic Scheduling- An Example



Priority(Task1) < Priority(Task2)

Rate Monotonic Scheduling

- Utilization

$$Utilization = \sum_i \frac{C_i}{T_i}$$

- All deadlines met when *utilization* ≤ $n(2^{1/n} - 1)$
  - For n=2, $2*(2^{1/2}-1) = 0.83$,
  - For $n \rightarrow \infty$, $n(2^{1/n} - 1) \rightarrow \ln(2) = 0.69$
- An example

Task	Period (T)	Rate (1/T)	Exec Time (C)	Utilization(U)
1	100 ms	10 Hz	20 ms	0.2
2	150 ms	6.67 Hz	40 ms	0.267
3	350 ms	2.86 Hz	100 ms	0.286

- 0.753 < 0.779 (=$3*(2^{1/3} -1)$) => tasks are schedulable

Rate Monotonic Scheduling Implementation

- Table of tasks with a task priority and its state (enables, etc.).
- At context switch select the task with the highest priority.
- Linear complexity, *O*(*n*) where *n* = number of tasks.

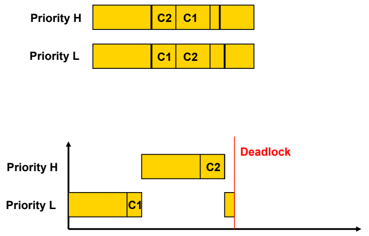
Critical sections

- Access to a shared resource should be mutually exclusive to access a resource
  - lock the resource → critical section starts
    - may fail and block the task
  - process the resource
  - unlock the resource critical section ends

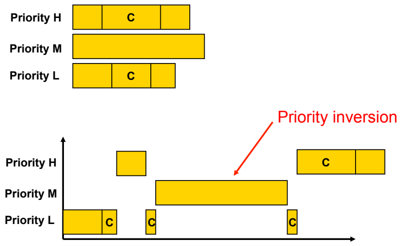
Rate Monotonic Scheduling — Problems

- Static cyclic scheduling is better if possible.
- Critical sections in tasks and communication create problems
  - Deadlock
  - Priority inversion
- Methods for solving priority inversion problem
  - priority inheritance,
  - priority ceiling protocol.

Deadlock



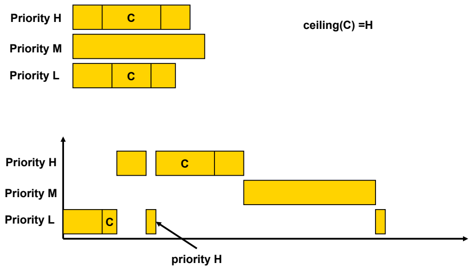
Priority Inversion



Priority Inheritance

- When a job J_i tries to enter a critical section and it is already locked by a lower priority task J_k then J_i waits and J_k inherits the priority of J_i
- The queue of jobs waiting for a resource is ordered by decreasing priority
- Priority inheritance is transitive.
- At any time, the priority at which a critical section is executed is always equal to the highest priority of the jobs that are currently blocked on it.
- When a job exits critical section it usually resumes the priority it had when it entered critical section.
- When released, a resource is granted to the highest priority job, if any waiting for it.

Priority Inheritance Protocol



Priority Inversion — problems

- **Chained blocking**:
  - a job can have several critical sections,
  - it can be blocked whenever it wants to enter a critical section
  - this generates overhead in terms of task switching.
- The main idea is to reduce the occurrence of priority inversions by preventing multiple priority inversions; a job will be blocked at most once before it enters its first critical section.
- The solution prevents deadlock.

Priority ceiling protocol

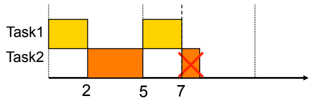
- Assumptions
  - a task cannot voluntarily suspend itself,
  - semaphores cannot be held between invocations,
  - semaphores must be locked in a nested manner.
- Protocol
  - Every CS has a ceiling: priority of a highest task that may enter it,
  - A task is allowed into a CS only if its priority is higher than ceilings of all active CS's,
  - If task A is blocking some higher priority task B, then A gets the priority of B while in CS.

Dynamic Scheduling — Dynamic Priorities

- Dynamic priorities needed since sometime static priorities might not meet deadlines
- An Example

  $C_1=2,\ T_1=5$

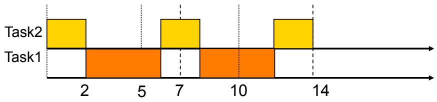
  $C_2=4,\ T_2=7$



Earliest Deadline First

- Minimizes number of missed deadlines.
- Tasks with earliest deadline has priority.
- An Example

$C_1=2$, $T_1=5$
$C_2=4$, $T_2=7$

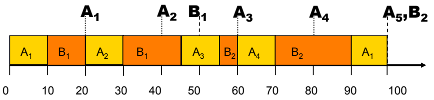


- Any sequence is optimal that puts the jobs in order of non-decreasing deadlines

Earliest Deadline First

- EDF can achieve 100% utilization until overload occurs.
- Cannot guarantee meeting deadlines for arbitrary data arrival times.
- An example

Task	Period (T)	Rate (1/T)	Exec Time (C)	Utilization (U)
A	20 ms	50 Hz	10 ms	0.5
B	50 ms	20 Hz	25 ms	0.5



Earliest Deadline First

- Additional assumptions
  - arbitrary release times and deadlines, and
  - arbitrary and unknown (to the scheduler) execution times.
- The EDF algorithm is optimal in that if there exist any algorithm that can build a valid (feasible) schedule on a single processor, then the EDF algorithm also builds a valid (feasible) schedule.

Earliest Deadline First Implementation

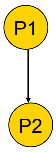
- At each preemption, sort tasks by time-to-deadline.
- Need for efficient sorting algorithm — $O(n \log n)$
- Choose ready task closest to the deadline.

What Is Missing

- Data dependencies between tasks
- Context switching time (jitter)
- Multiprocessor scheduling
- Memory considerations
- ...

Data dependencies

- Data dependencies allow us to improve utilization.
  - Restrict combination of processes that can run simultaneously.
- P1 and P2 can't run simultaneously.



Context-switching time

- Non-zero context switch time can push limits of a tight schedule.
- Hard to calculate effects -- depends on order of context switches.
- In practice, OS context switch overhead is small.

Literature

- P. Eles, K. Kuchcinski and Z. Peng, *System Synthesis with VHDL*, Kluwer Academic Publisher, 1998.
- Any book on real-time scheduling, e.g.,
  Alan Burns and Andy Wellings, *Real-Time Systems and Programming Languages*, Addison Wesley, 1996.