System Synthesis of Digital Systems

Delft University of Technology

System Synthesis

- Input: an implementation independent specification of the system; this includes: functionality and constraints.
- The synthesis tasks:
  - To select the architecture
  - To partition functionality over the components of the architecture
  - To schedule activities

TU Delft

System Synthesis (2)

- To generate behavioral modules corresponding to the hardware and software domain of the implementation, including interface modules.
- The behavioral modules resulted from the previous steps are further synthesized into the actual hardware and/or software implementation.

From Algorithm to Design Representation

xl = x + dx;

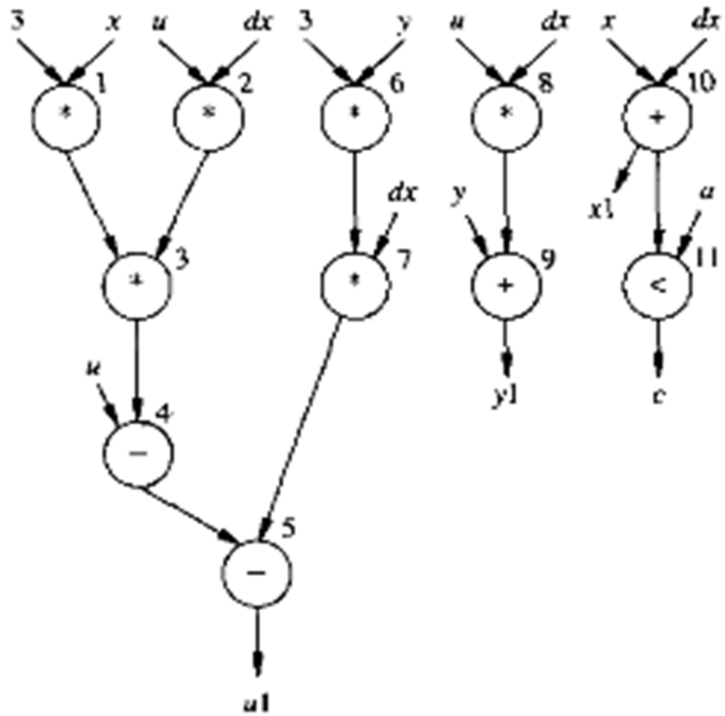
ul = u - (3*x*u*dx) – (3*y*dx);

yl = y+u*dx;

c = xl < a;

x = xl; u=ul; y=yl;

See also Fig. 3.11



TU Delft

High –Level Synthesis

1. Basic definition
2. A typical HLS process
3. Scheduling techniques
4. Allocation and binding techniques
5. Advanced issues

Introduction

- Definition: HLS generates register-transfer level designs from behavioral specifications, in a automatic manner.
- Input:

  - The behavioral specification.

  - Design constraints (cost, performance, power consumption, pin-count, testability, etc.).

  - An optimization function.

  - A module library representing the available components at RTL.

Introduction (2)

- Output
  - RTL implementation structure (net list).
  - Controller (captured usually as a symbolic FSM).
  - Other attributes, such as geometrical information.
- Goal: to generate a RTL design that implements the specified behavior while satisfying the design constraints and optimizing the given cost function.

TU Delft

A typical HLS Process (1)

1. Behavioral specification:

Which language to use?

- Procedural languages
- Functional languages
- Graphics notations

Explicit parallelism?

Input behavioral specification

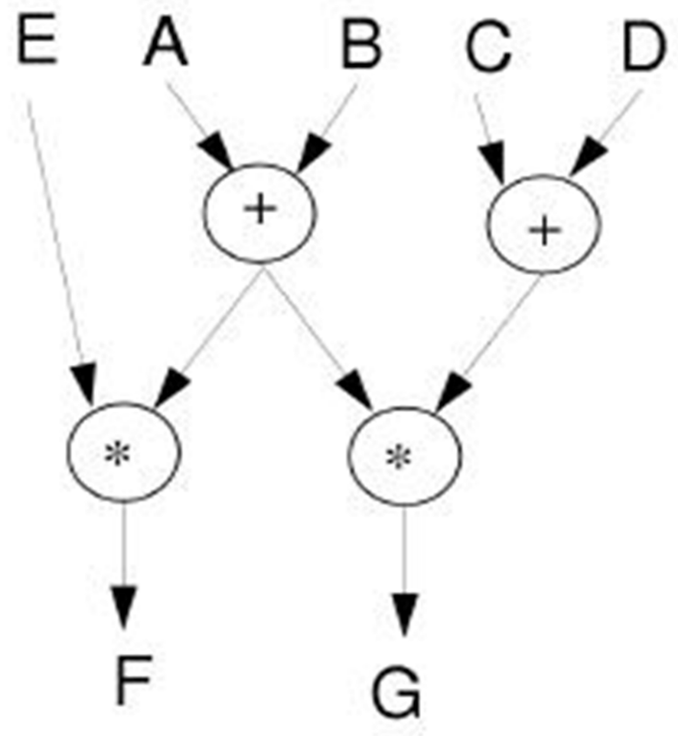
```
Procedure Test;
  VAR A, B, C, D, E, F, G: integer;
     BEGIN
     Read (A,B,C,D,E);
           F:= E*(A+B);
           G:=(A+B)*(C+D);
           .....
    END
```


TU Delft

A Typical HLS Process (2)

2. Dataflow analysis:

- •Parallelism extraction.
- •Eliminating high-level language constructs.
- •Loop unrolling.
- •Program transformation.
- •Common sub expression detection.



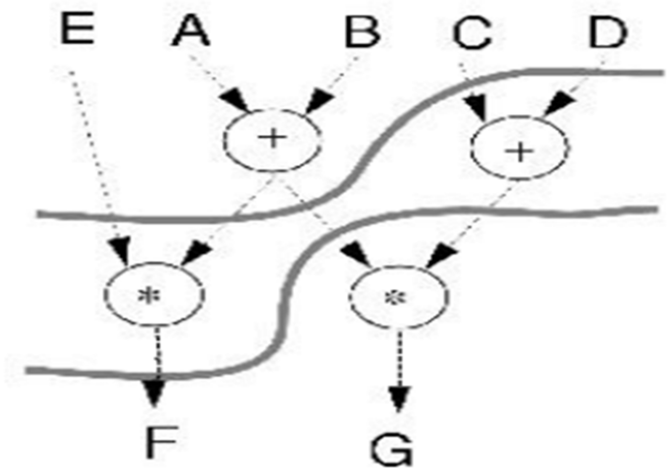
Dataflow description

TU Delft

A Typical HLS Process (3)

3. Operating scheduling

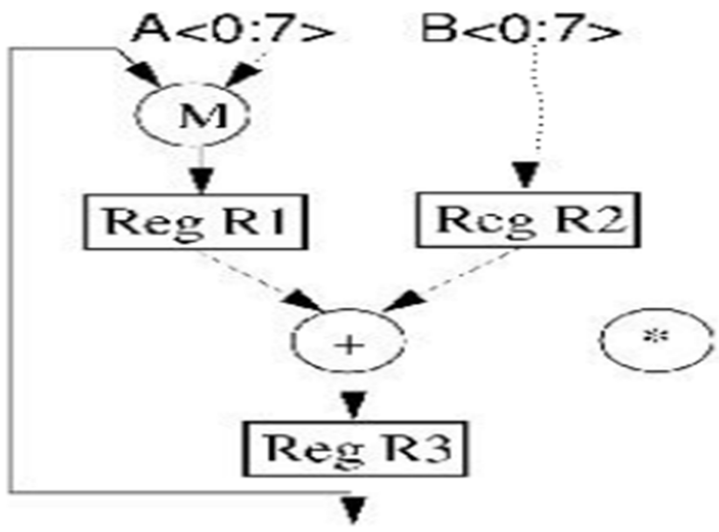
- Performance/cost trade-offs.
- Performance measure.
- Clocking strategy.



Scheduled dataflow description

A Typical HLS Process (4)

4. Data-path allocation:
   - Operator selection.
   - Register/memory allocation.
   - Interconnection generation.
   - Hardware minimization



Partial data-path

A Typical HLS Process (5)

5. Control allocation:

- Selection of control style (PLA, microcode, random logic, etc.).
- Controller generation.

Optimization Need to know

NP, NP complete, NP hard

•NP-hard (Non-deterministic Polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP".

•In computational complexity theory, a decision problem is NP-complete when it is both in NP and NP-hard. The set of NP-complete problems is often denoted by NP-C or NPC.

Bounds of a solution

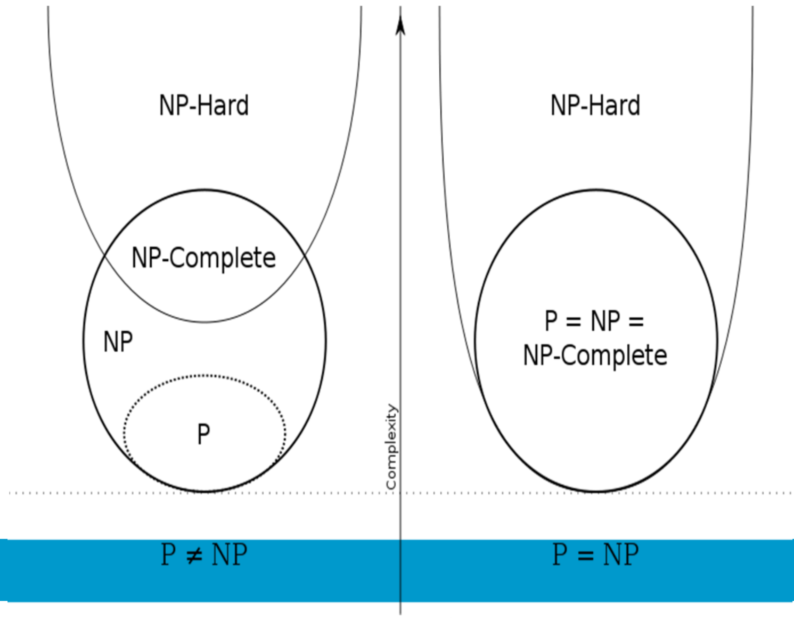
Exact algorithms: Mathematical proof of correctness

Heuristic algorithms: 'Work Best'

Compute complexity: Order O(....)

Local minimum

Global Minimum



The Basic Issues (1)

- Scheduling – Assignment of each operation to a time slot corresponding to a clock cycle or time interval.
- Resource Allocation – Selection of the types of hardware components and the number for each type to be included in the final implementation.
- Module Binding – Assignment of operation to the allocated hardware components.
- Controller Synthesis – Design of control style and clocking scheme.

TU Delft

The Basic Issues (2)

- Compilation of the input specification language to the internal representation must be done.
- Parallelism Extraction – To extract the inherent parallelism of the original solution, which is usually done with data flow analysis techniques.
- Operation Decomposition – Implementation of complex operations in the behavioral specification.
- ....

TU Delft

The scheduling Problem (1)

- Resource-constrained (RC) scheduling:

  - Given a set O of operations with a partial ordering which determines the precedence relations, a set K of functional unit types, a type function, $\tau: O \rightarrow K$, to map the operations into the functional unit types, and resource constraints m_k for each functional unit type.

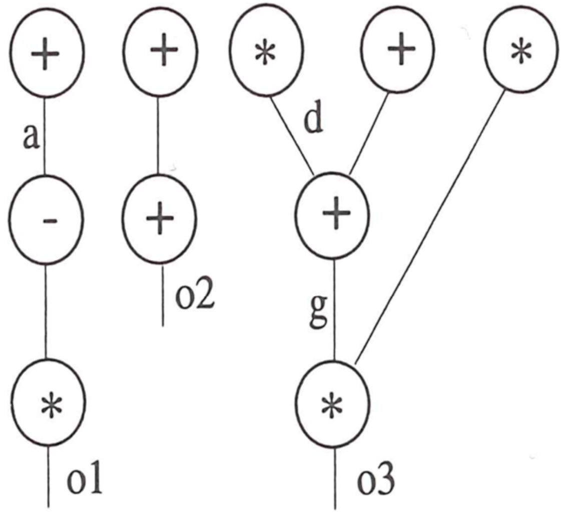
  - Find a (optimal) schedule for the set of options that obeys the partial ordering and utilizes only the available functional units.

TU Delft

The Scheduling Problem (2)

```
a  := i1 + i2;
o1  := (a - i3) * 3;
o2  := i4 + i5 + i6;
d  := i7 * i8;
g  := d + i9 + i10;
o3  := i11 * 7 * g;
```

(a) Behaviroal specification



(b) DFG

1 adder, 1 multiplier

+- → adder

* → multiplier

Scheduling

- Scheduling:
  - Determine the start times for the operations
  - Satisfying all the sequencing (timing and resource) constraints
- Goal:
  - Determine *area/latency* trade-off

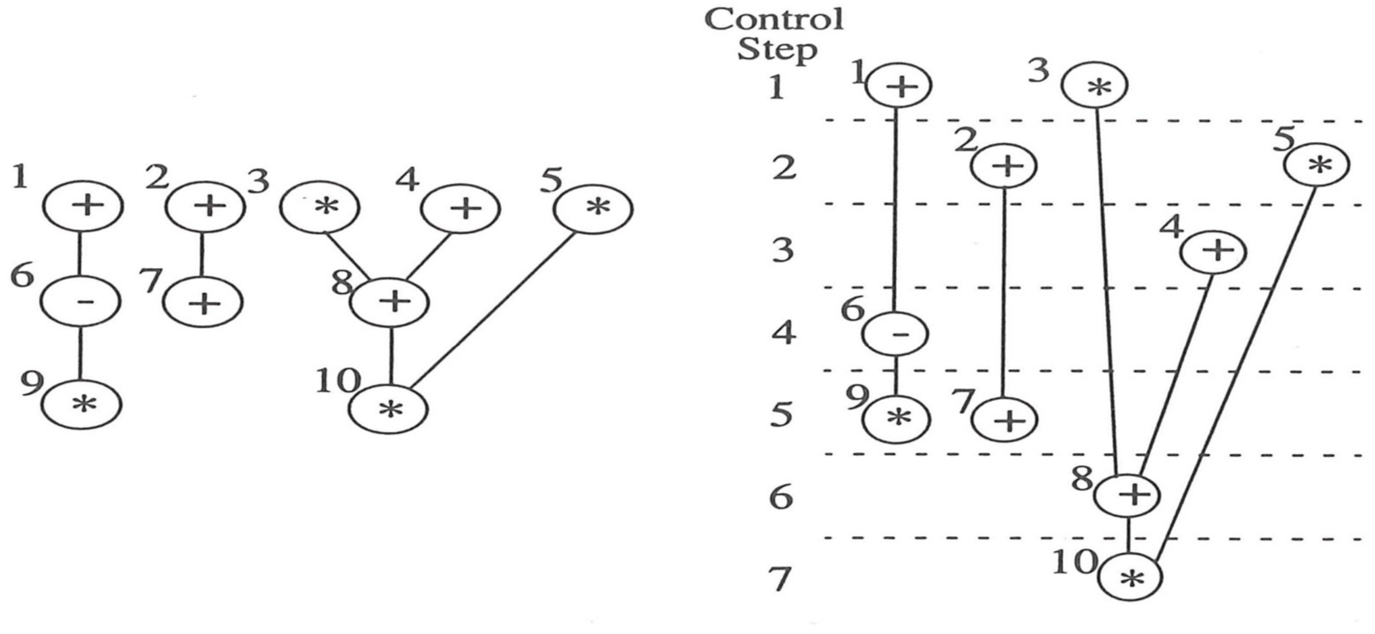
TU Delft

RC Scheduling Techniques (Resource Constrained)

ASAP: As soon as possible

- Sort the operations topologically according to their data/control flow;
- Schedule operations in the sorted order by placing them in the earliest possible control step.

ASAP



(a) Sorted DFG

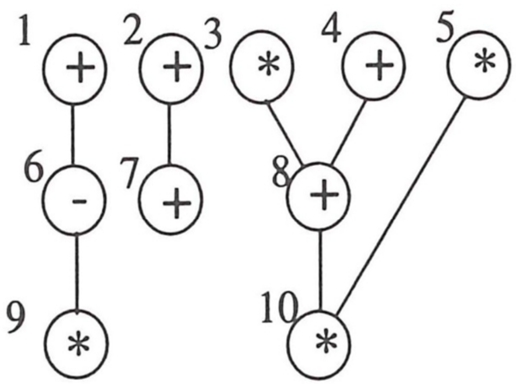
(b) ASAP schedule

TUDelft

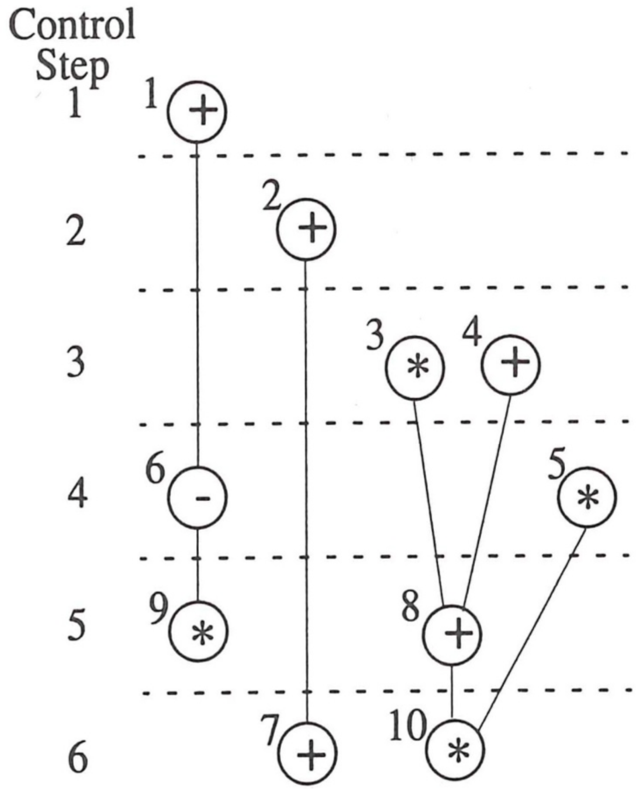
RC Scheduling Techniques (Cont'd)

- ALAP: As late as possible
  - Sort the operations topologically according to their data/control flow;
  - schedule operations in the reversed order by placing them in the latest possible control step.

ALAP



(a) Sorted DFG



(b) ALAP schedule

Remarks

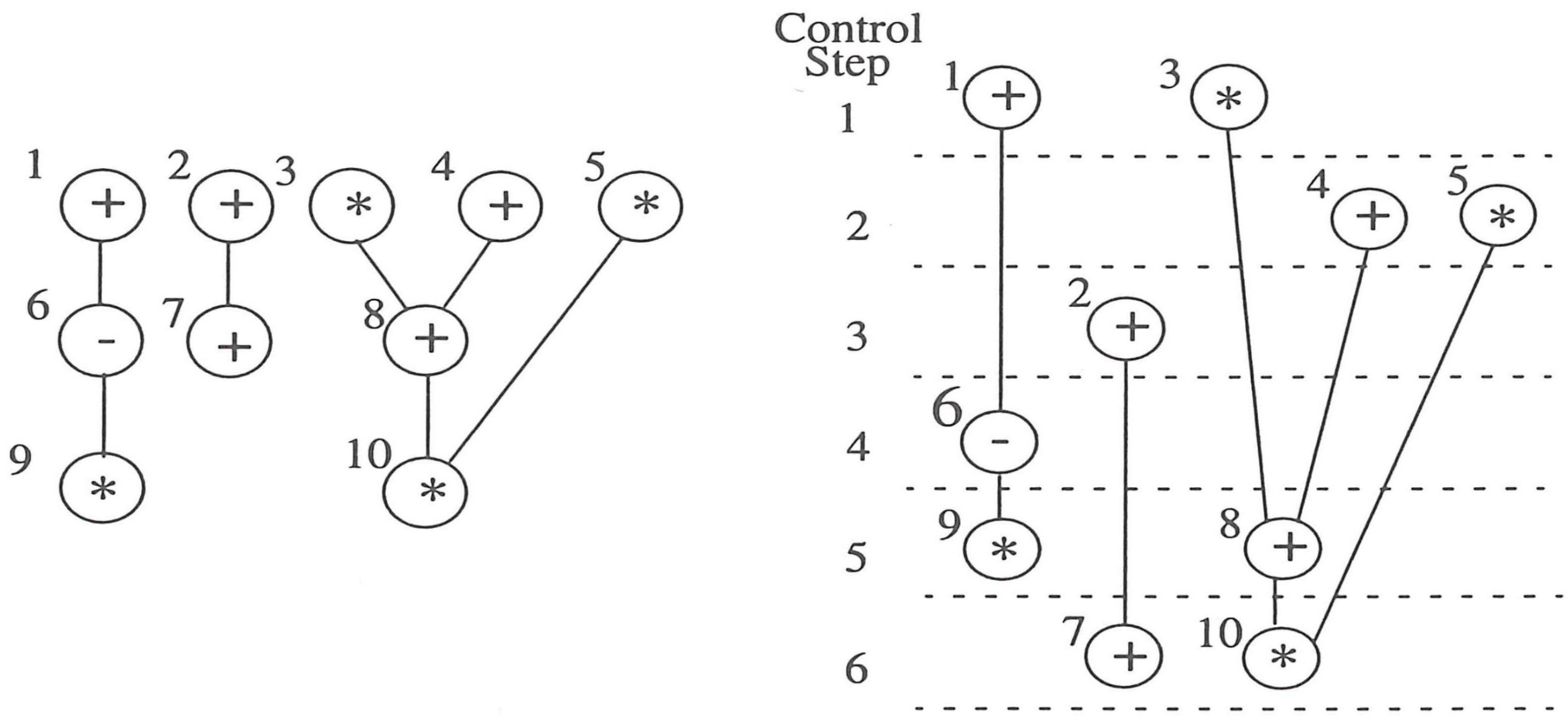
- ALAP solves a latency-constrained problem
- Mobility:
  - Defined for each operation
  - Difference between ALAP and ASAP schedule
- Slack on the start time

RC Scheduling Techniques (Cont'd)

- List Scheduling
  - For each control step, the operations that are available to be scheduled are kept in a list;
  - The list is ordered by some priority function:
    - 1. The length of path from the operation to the end of the block;
    - 2. Mobility: the number of control steps from the earliest to the latest feasible control step.
  - Each operation on the list is scheduled one by one if the resources it needs are free: otherwise it is deferred to the next control step.

TU Delft

LIST



(a) DFG (b) List schedule

TU Delft

Scheduling under resource constraints

- Intractable problem
- Algorithms:
  - Exact:
    - Integer linear program
    - Hu (restrictive assumptions)
  - Approximate/Heuristic :
    - List scheduling
    - Force-directed scheduling

ILP

- Linear programming (LP) is a mathematical method for determining a way to achieve the best outcome (such as maximum profit or lowest cost) in a given mathematical model for some list of requirements represented as linear relationships.

More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints.

- An integer linear program, variables are forcibly constrained to be integers, and this problem is NP-hard in general.

ILP formulation

- Binary decision variables:

  $X = \{ x_{il}, \quad i = 1,2,\ldots.\ n; \ \ l = 1,2,\ldots, \lambda + 1\}$

  x_{il} is TRUE only when operation v_i starts in step l of the schedule (i.e. $l = t_i$)

  λ is an upper bound on latency

- Start time of operation v_i : $\Sigma_l \ l \cdot x_{il}$

ILP formulation constraints

- Operations start only once

  $\sum_l x_{il} = 1 \quad i = 1, 2, \ldots, n$

- Sequencing relations must be satisfied

  $t_i \geq t_j + d_j \quad \rightarrow \quad t_i - t_j - d_j \geq 0 \quad \text{for all } (v_j, v_i) \in E$

  $\sum_l l \bullet x_{il} - \sum_l l \bullet x_{jl} - d_j \geq 0 \quad \text{for all } (v_j, v_i) \in E$

- Resource bounds must be satisfied

  Simple case (unit delay)

  $\sum_{i:T(v_i)=k} \sum_{m=l-d_i+1}^{l} x_{im} \leq a_k \quad , \quad k = 1,2,\ldots n_{res} ; \quad \text{for all } l$

See also page 199, De Micheli

TU Delft

SAT solver

A SAT solver is a program that automatically decides whether a propositional logic formula is satisfiable.

If it is satisfiable, a SAT solver will produce an example of a truth assignment that satisfies the formula.

SAT solvers have proved to be an indispensable component of many formal verification and (more recently) program analysis applications.

Hu's algorithm

- Assumptions:
  - Graph is a forest
  - All operations have unit delay
  - All operations have the same type
- Algorithm:
  - Greedy strategy
  - Exact solution

TC Scheduling Techniques (1)

- Force-Directed Scheduling: The basic idea is to balance the concurrency of operations.
  - ASAP and ALAP schedules are calculated to derive the time frames for all operations.
  - For each type of operations, a distribution graph is built to denote the possible control steps for each operation. If an operation could be done in k steps, then 1/k is added to each of these k steps.

Force-directed scheduling

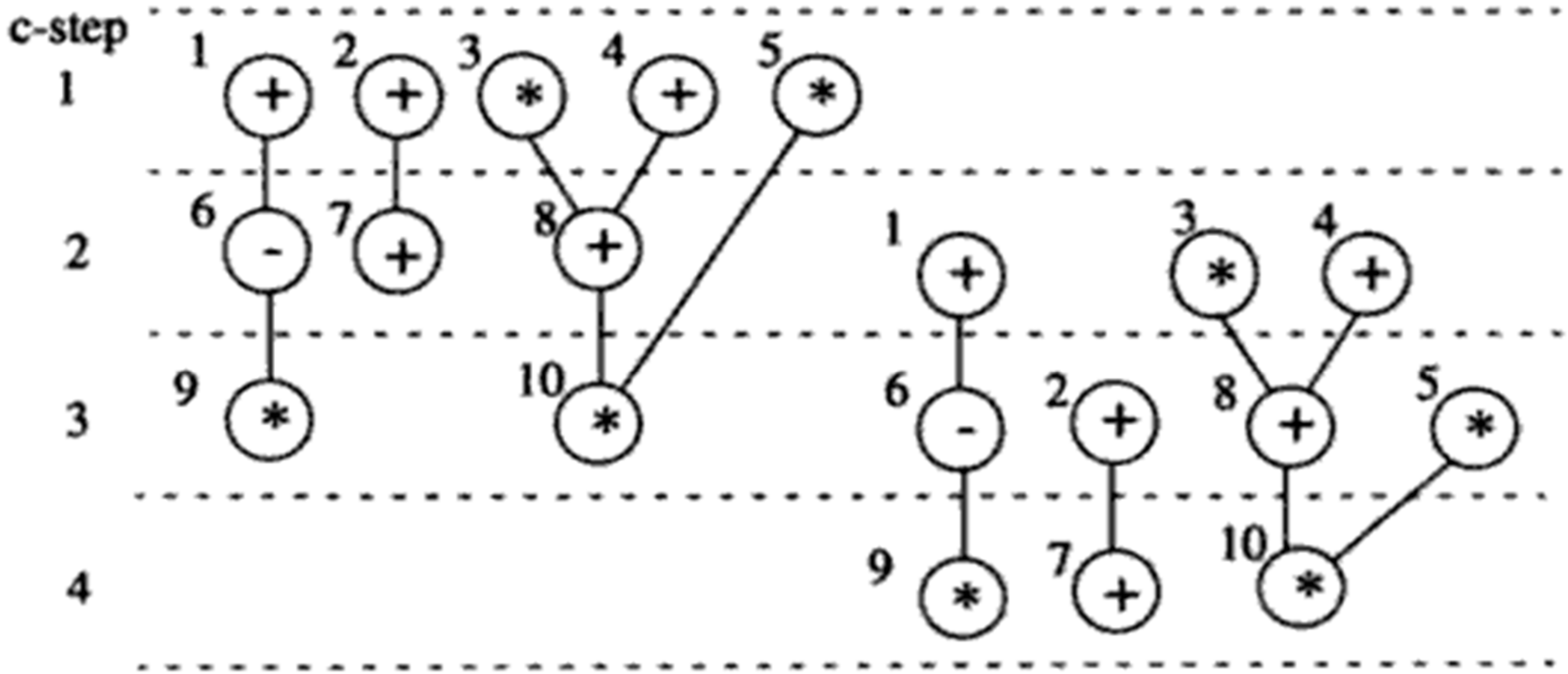
- Heuristic scheduling methods [Paulin]:
  - Min *latency* subject to *resource bound*
    - *Variation* of list scheduling : FDLS
  - Min *resource* subject to *latency bound*
    - Schedule one operation at a time
- Rationale:
  - Reward *uniform distribution* of operations across schedule steps

Force

- Used as *priority* function
- Force is related to concurrency:
  - Sort operations for least force
- Mechanical analogy:
  - Force = constant x displacement
    - Constant = operation-type distribution
    - Displacement = change in probability

TU Delft

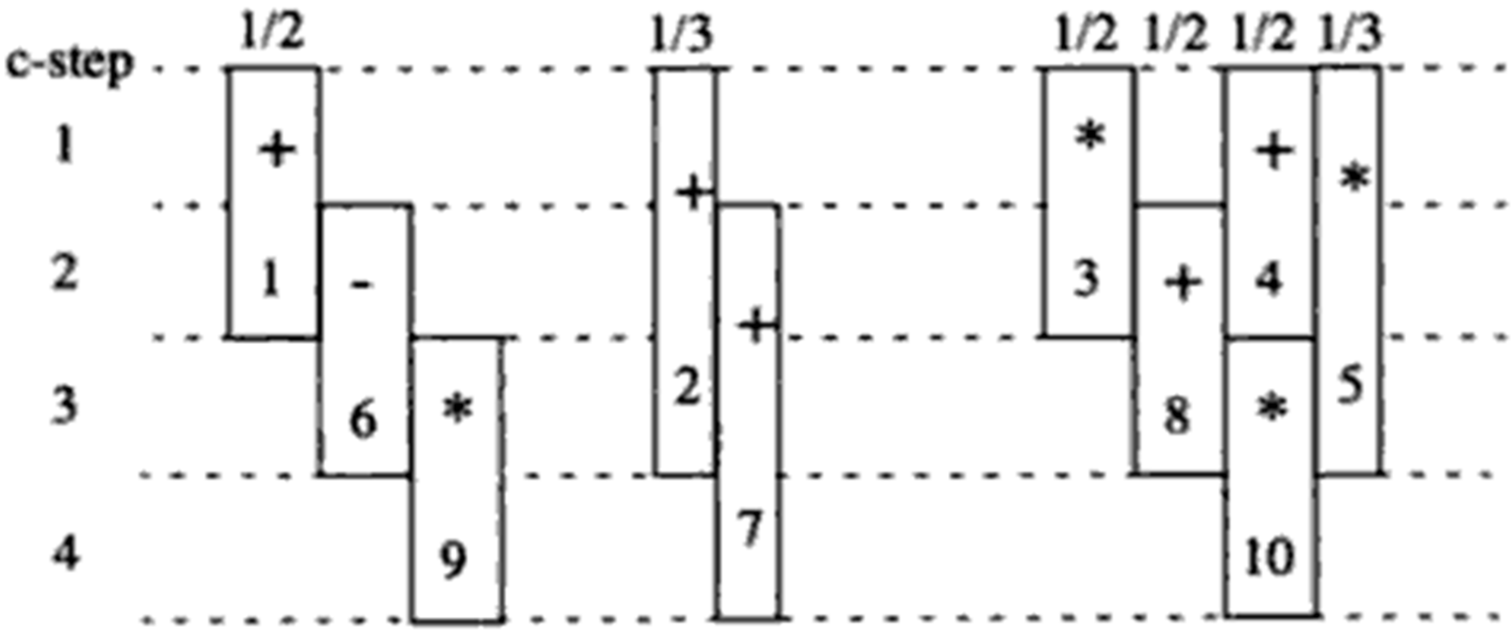
ASAP and ALAP



(a) ASAP schedule (b) ALAP schedule

TU Delft

C-Steps



Distribution

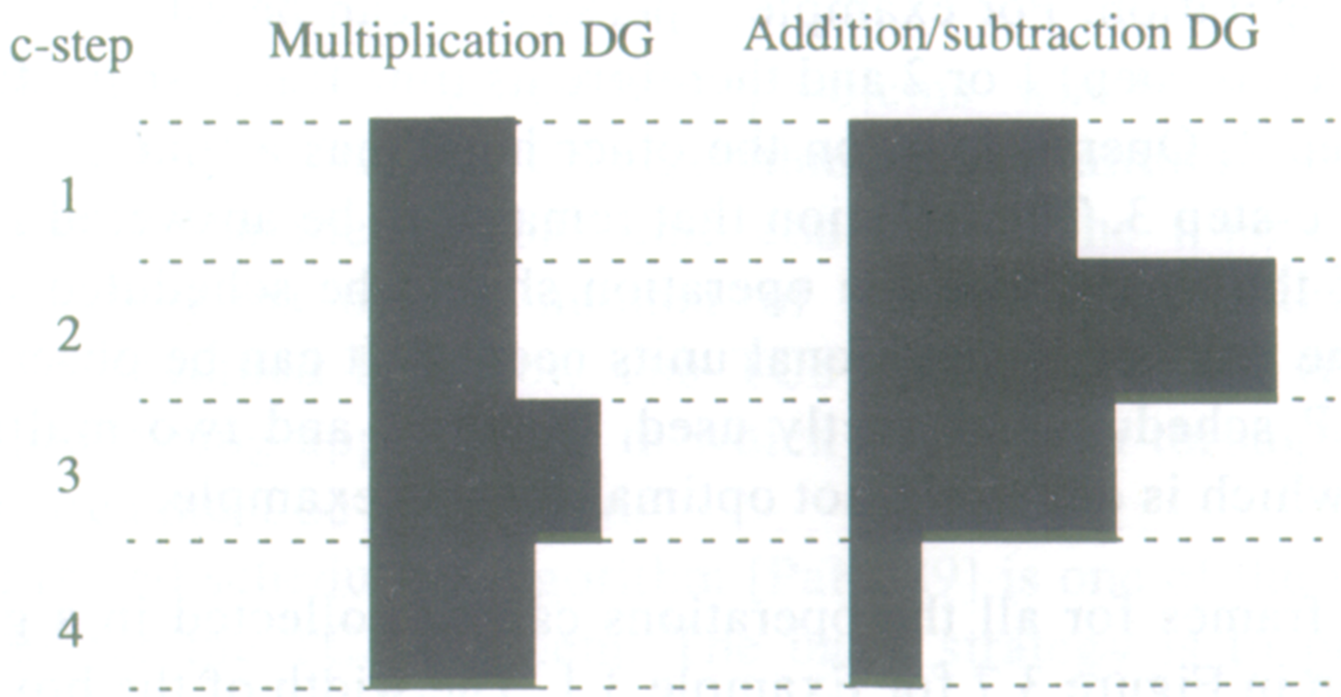
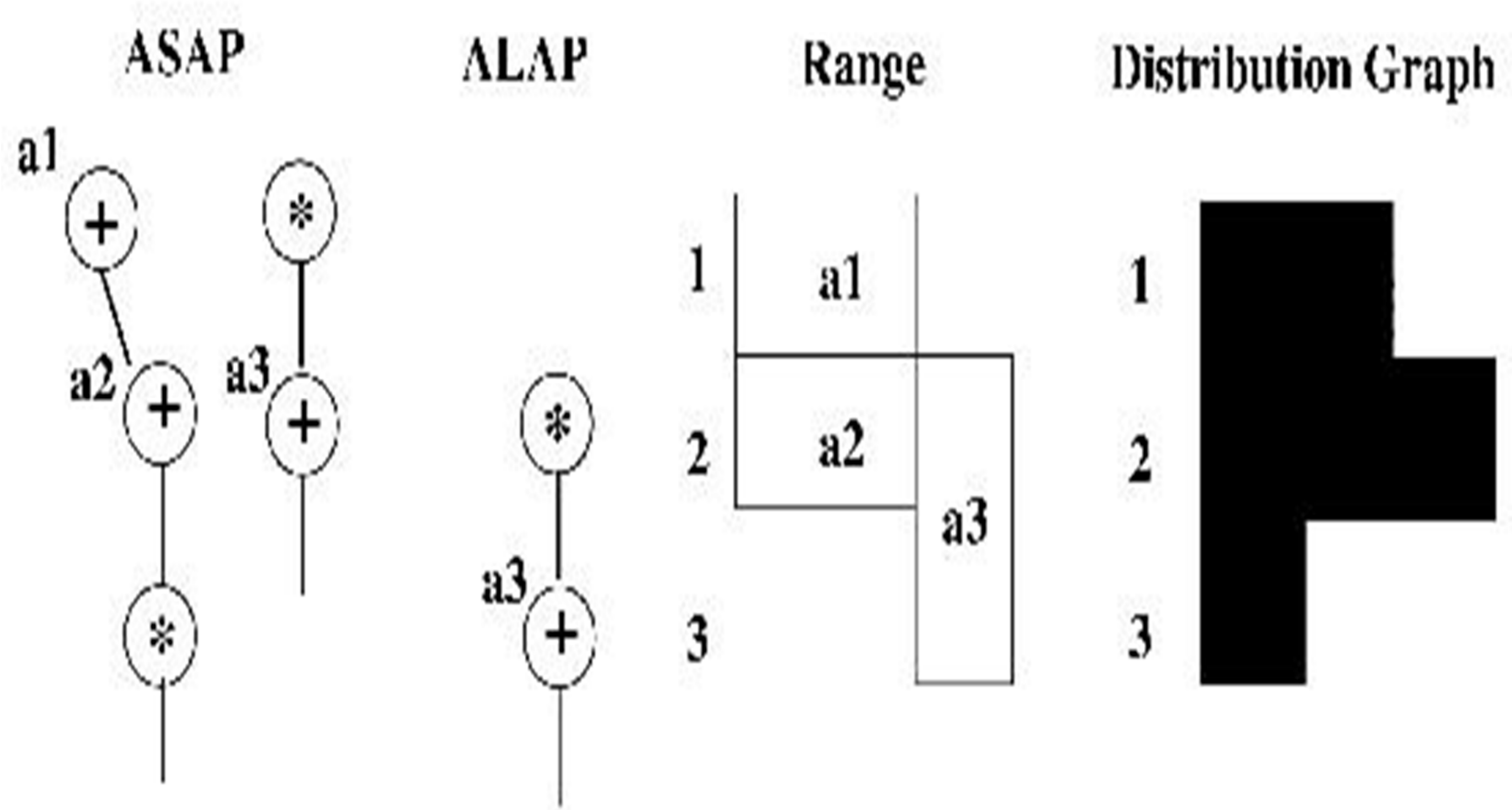


Figure 3.8: Distribution graphs for Example 3.1.

FDS



ASAP
ALAP
Range
Distribution Graph
a1
a2
a3
a3
1
2
3
a1
a2
a3
1
2
3

TU Delft

TC Scheduling Techniques (2)

- The algorithm tries to balance the distribution graph by calculate the force of each operation-to-control step assignment and select the smallest force:

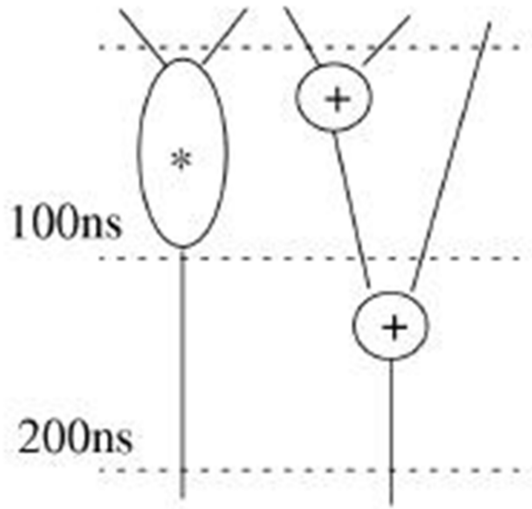
$$Force_{(\sigma(o_i) = s_j)} = DG(s_j) - \frac{1}{\Delta T(o_i)} \cdot \sum_{s = \sigma_{ASAP}(o_i)}^{\sigma_{ALAP}(o_i)} DG(s)$$

Advanced Scheduling Issues

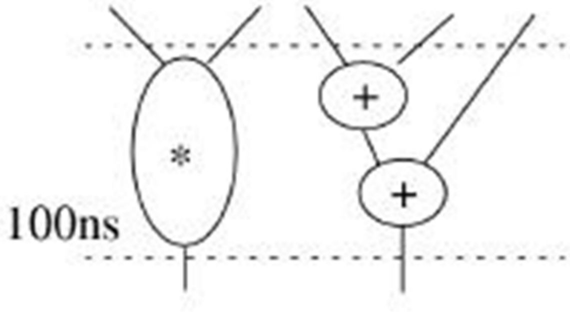
- Control construct consideration.
  - Conditional branches
  - Loops
- Chaining and multicycling.

- Scheduling with local timing constraints.

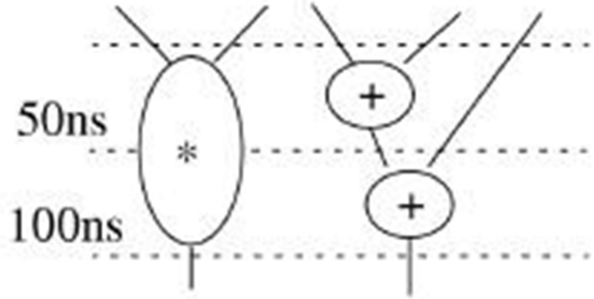
Advanced Scheduling Issues (2)



(a) No chaining or multicycling



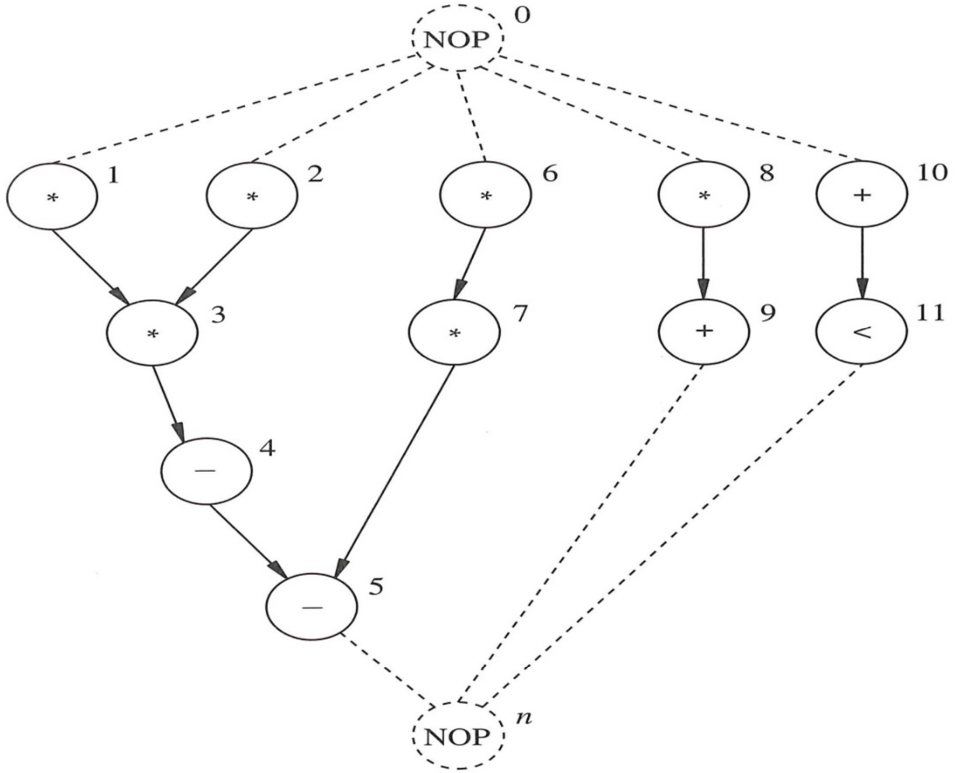
(b) Two chained additions



(c) A multicycle multiplication

TU Delft

Exercise



Consider the graph of Figure 5.1. Assume the execution delays of the multiplier and of the ALU are 2 and 1 cycle respectively. Schedule the graph using the ASAP algorithm. Assuming a latency bound of $\overline{\lambda} = 8$ cycles, schedule the graph using the ALAP algorithm. Determine the mobility of the operations.

TU Delft