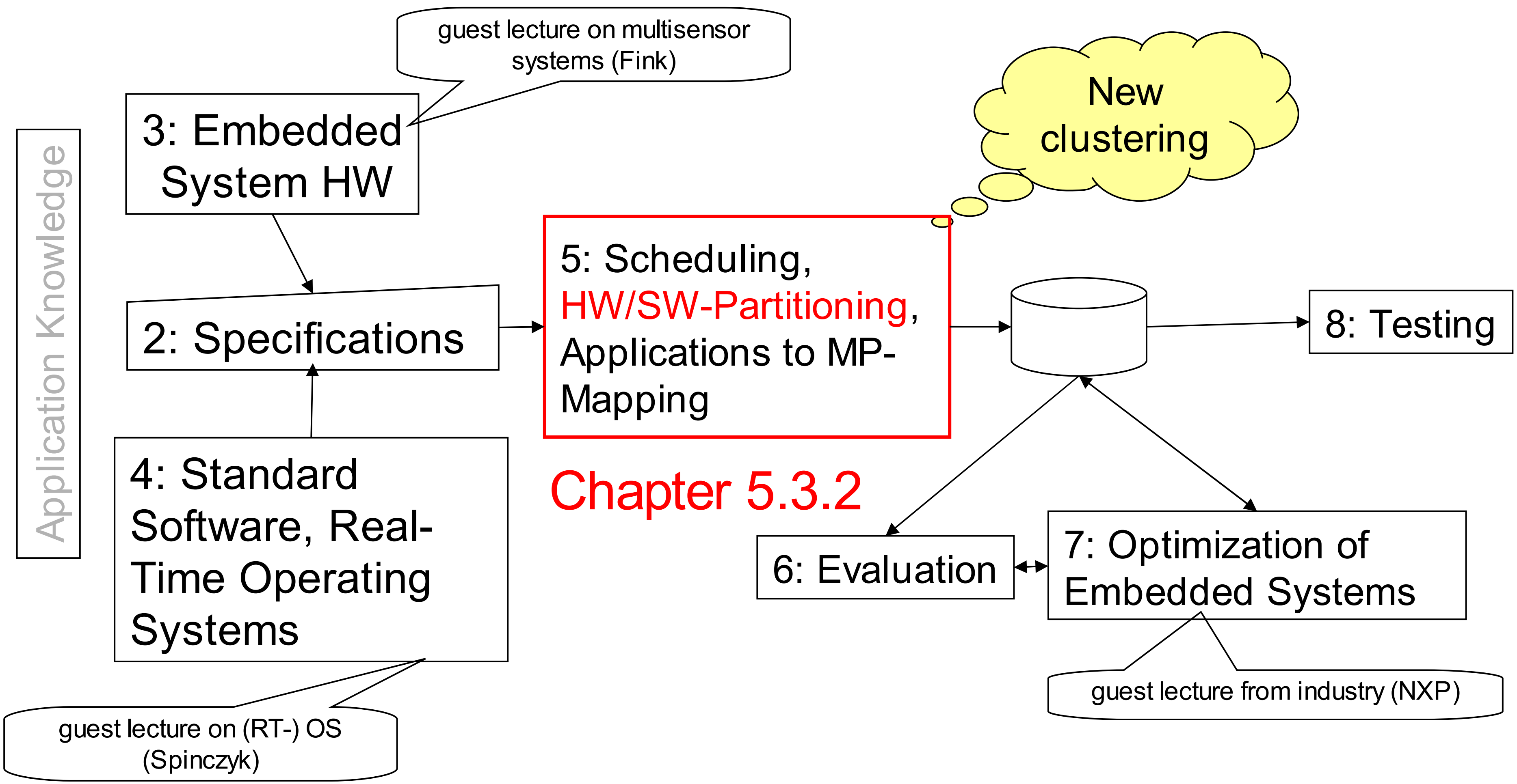
Hardware/Software Partitioning

Peter Marwedel
Informatik 12
TU Dortmund
Germany

Chapter 5.3.2

Structure of this course



Integer (linear*) programming models

*"linear" sometimes included, sometimes not

Ingredients:
- Cost function
- Constraints

Involving linear expressions of integer variables from a set X

Cost function $$C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in R, x_i \in \mathbb{N} \quad (1)$$

Constraints: $$\forall j \in J : \sum_{x_i \in X} b_{i,j}\, x_i \geq c_j \text{ with } b_{i,j}, c_j \in \mathbb{R} \quad (2)$$

Def.: The problem of minimizing (1) subject to the constraints (2) is called an **integer (linear) programming (I(L)P) problem**.

If all x_i are constrained to be either 0 or 1, the IP problem said to be a **0/1 integer (linear) programming problem**.

Example

$$C = 5x_1 + 6x_2 + 4x_3$$

$$x_1 + x_2 + x_3 \geq 2$$

$$x_1, x_2, x_3 \in \{0,1\}$$

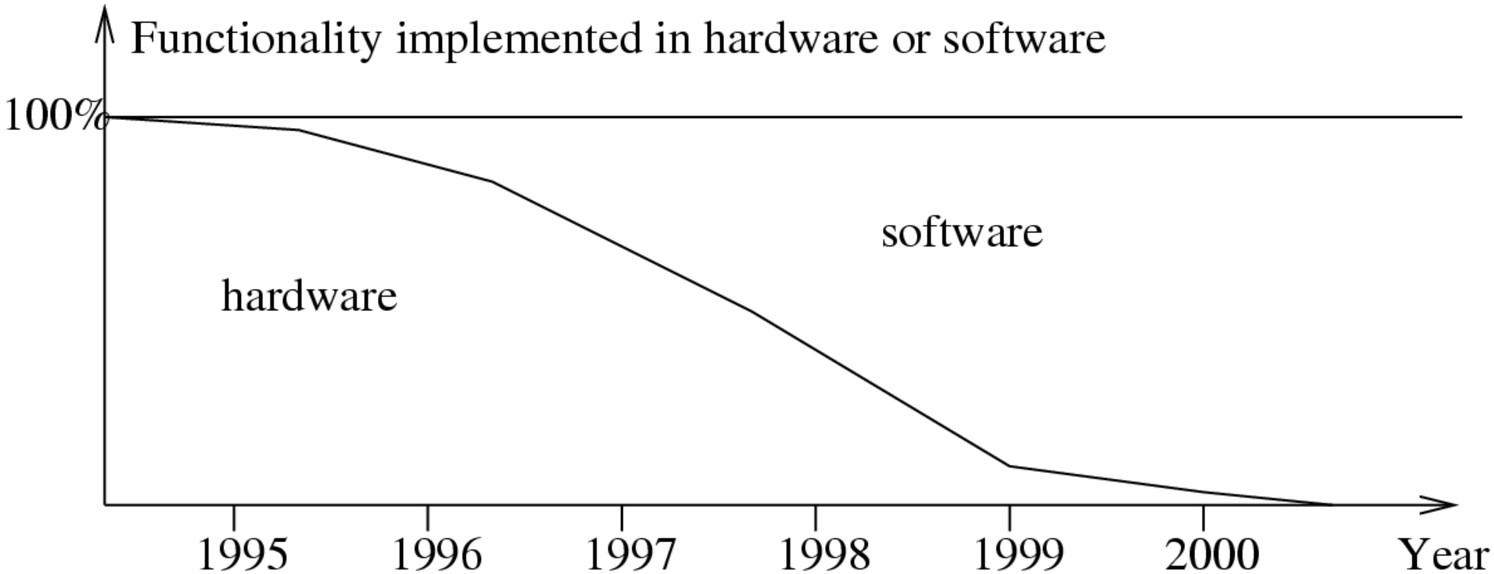
x_1	x_2	x_3	C
0	1	1	10
1	0	1	9
1	1	0	11
1	1	1	15

← Optimal (row with $C = 9$)

Remarks on integer (linear) programming

- Maximizing the cost function: just set C'=-*C*
- Integer programming is NP-complete.
- Running times depend exponentially on problem size, but problems of >1000 vars solvable with good solver (depending on the size and structure of the problem)
- The case of $x_i \in \mathbb{R}$ is called *linear programming* (LP). LP has polynomial complexity, but most algorithms are exponential, still in practice faster than for IP problems.
- The case of some $x_i \in \mathbb{R}$ and some $x_i \in \mathbb{N}$ is called *mixed integer-linear programming.*
- ILP/LP models can be a good starting point for modeling, even if in the end heuristics have to be used to solve them.

Hardware/software partitioning

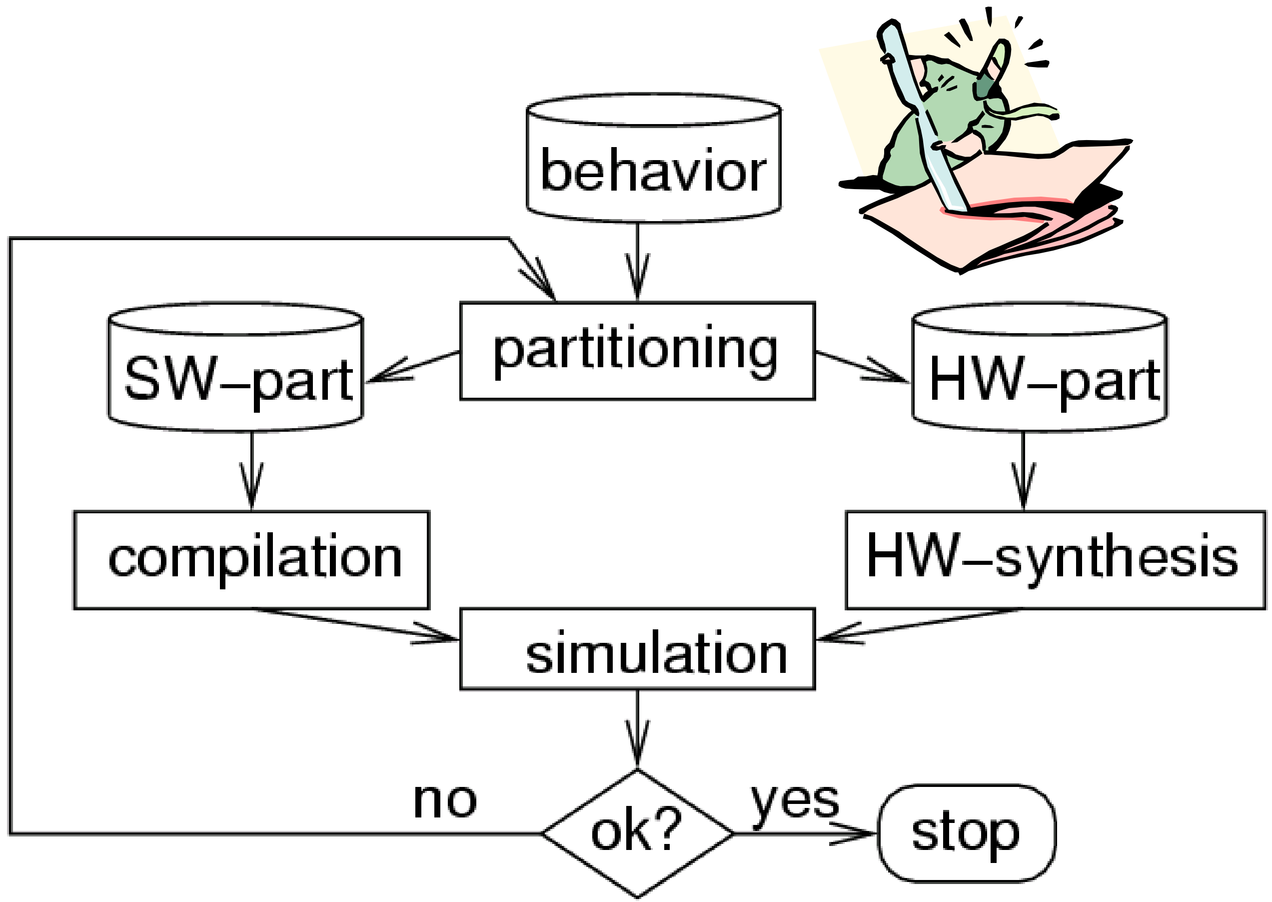


No need to consider special purpose hardware in the long run?
Correct for fixed functionality, but wrong in general, since
"By the time MPEG-n can be implemented in software, MPEG-n+1 has been invented" [de Man]

☞ Functionality to be implemented in software or in hardware?

Functionality to be implemented in software or in hardware?

Decision based on hardware/ software partitioning, a special case of hardware/ software codesign.



Codesign Tool (COOL) as an example of HW/SW partitioning

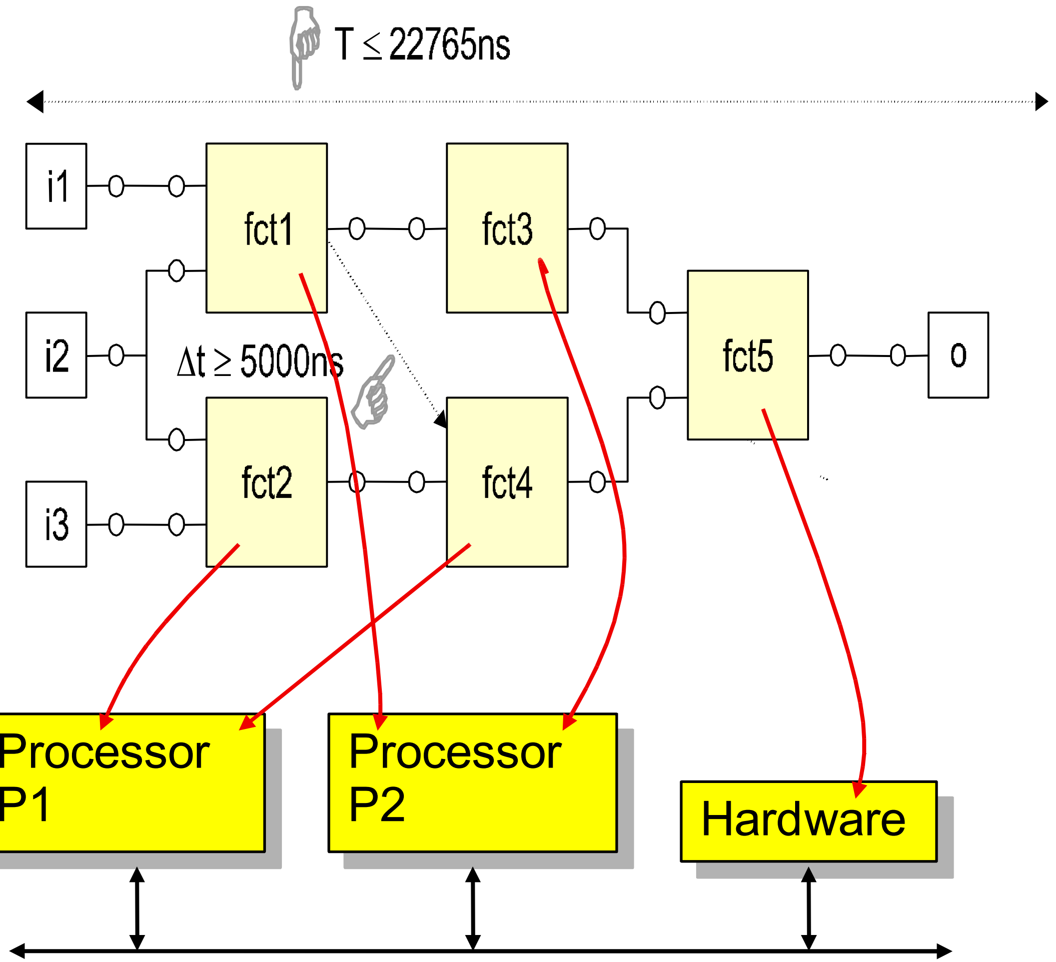
Inputs to COOL:

1. Target technology
2. Design constraints
3. Required behavior

Hardware/software codesign: approach

Specification

Mapping



[Niemann, Hardware/Software Co-Design for Data Flow Dominated Embedded Systems, Kluwer Academic Publishers, 1998 (Comprehensive mathematical model)]

Steps of the COOL partitioning algorithm (1)

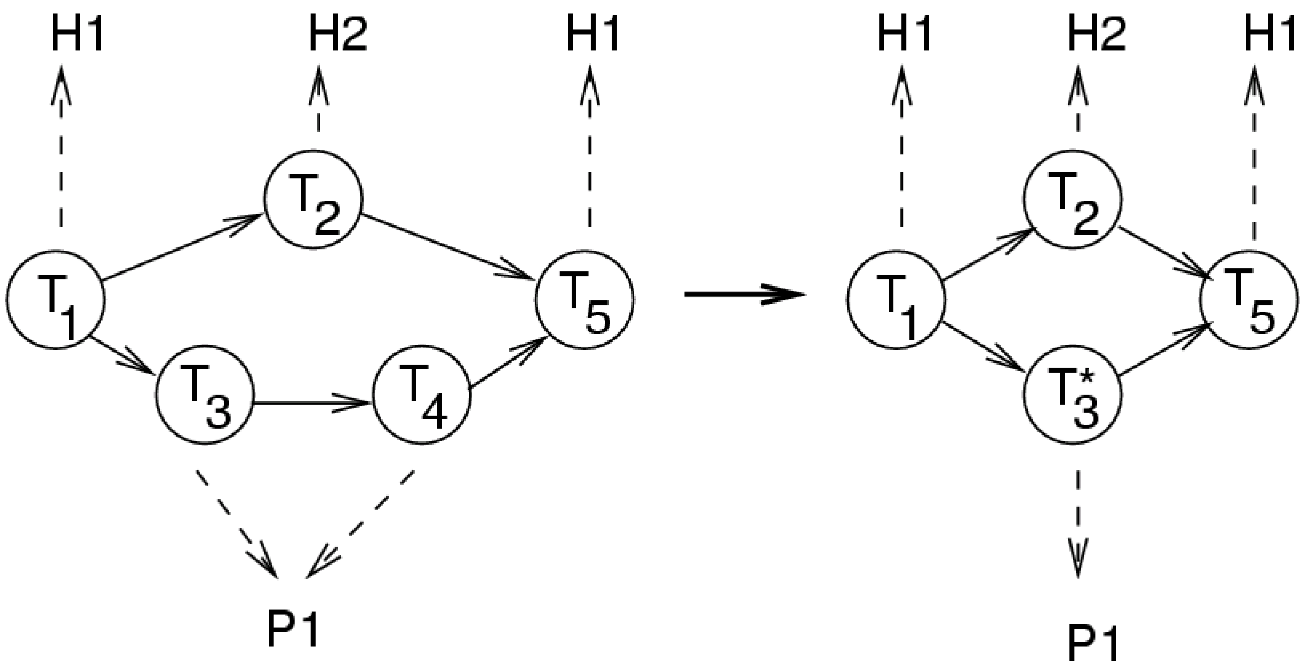
1. **Translation of the behavior into an internal graph model**
2. **Translation of the behavior of each node from VHDL into C**
3. **Compilation**
   - All C programs compiled for the target processor,
   - Computation of the resulting program size,
   - estimation of the resulting execution time (simulation input data might be required)
4. **Synthesis of hardware components**:
   ∀ leaf nodes, application-specific hardware is synthesized.
   High-level synthesis sufficiently fast.

Steps of the COOL partitioning algorithm (2)

1. **Flattening of the hierarchy**:
   - Granularity used by the designer is maintained.
   - Cost and performance information added to the nodes.
   - Precise information required for partitioning is pre-computed
2. **Generating and solving a mathematical model of the optimization problem**:
   - Integer programming IP model for optimization. Optimal with respect to the cost function (approximates communication time)

Steps of the COOL partitioning algorithm (3)

1. **Iterative improvements**:
   Adjacent nodes mapped to the same hardware component are now merged.



Steps of the COOL partitioning algorithm (4)

1. **Interface synthesis**:
   After partitioning, the glue logic required for interfacing processors, application-specific hardware and memories is created.

An integer programming model for HW/SW partitioning

Notation:

- Index set *I* denotes task graph nodes.
- Index set *L* denotes task graph node **types**
  e.g. square root, DCT or FFT
- Index set *KH* denotes hardware component **types**.
  e.g. hardware components for the DCT or the FFT.
- Index set *J* of hardware component instances
- Index set *KP* denotes processors.
  All processors are assumed to be of the same type

An IP model for HW/SW partitioning

- $X_{i,k}$: =1 if node v_i is mapped to hardware component type $k \in KH$ and 0 otherwise.
- $Y_{i,k}$: =1 if node v_i is mapped to processor $k \in KP$ and 0 otherwise.
- $NY_{\ell,k}$ =1 if at least one node of type ℓ is mapped to processor $k \in KP$ and 0 otherwise.
- T is a mapping from task graph nodes to their types: $T: I \rightarrow L$
- The cost function accumulates the cost of hardware units:

  C = cost(processors) + cost(memories) + cost(application specific hardware)

Constraints

Operation assignment constraints

$$\forall i \in I : \sum_{k \in KH} X_{i,k} + \sum_{k \in KP} Y_{i,k} = 1$$

All task graph nodes have to be mapped either in software or in hardware.

Variables are assumed to be integers.

Additional constraints to guarantee they are either 0 or 1:

$$\forall i \in I : \forall k \in KH : X_{i,k} \leq 1$$

$$\forall i \in I : \forall k \in KP : Y_{i,k} \leq 1$$

Operation assignment constraints (2)

$\forall \ell \in L, \forall i: T(v_i)=c_\ell, \forall k \in KP: NY_{\ell,k} \geq Y_{i,k}$

For all types ℓ of operations and for all nodes *i* of this type:
if *i* is mapped to some processor *k*, then that processor must implement the functionality of ℓ.

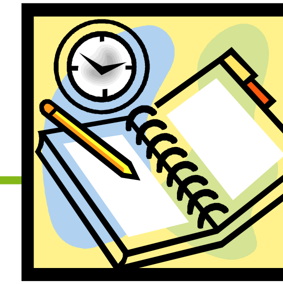
Decision variables must also be 0/1 variables:

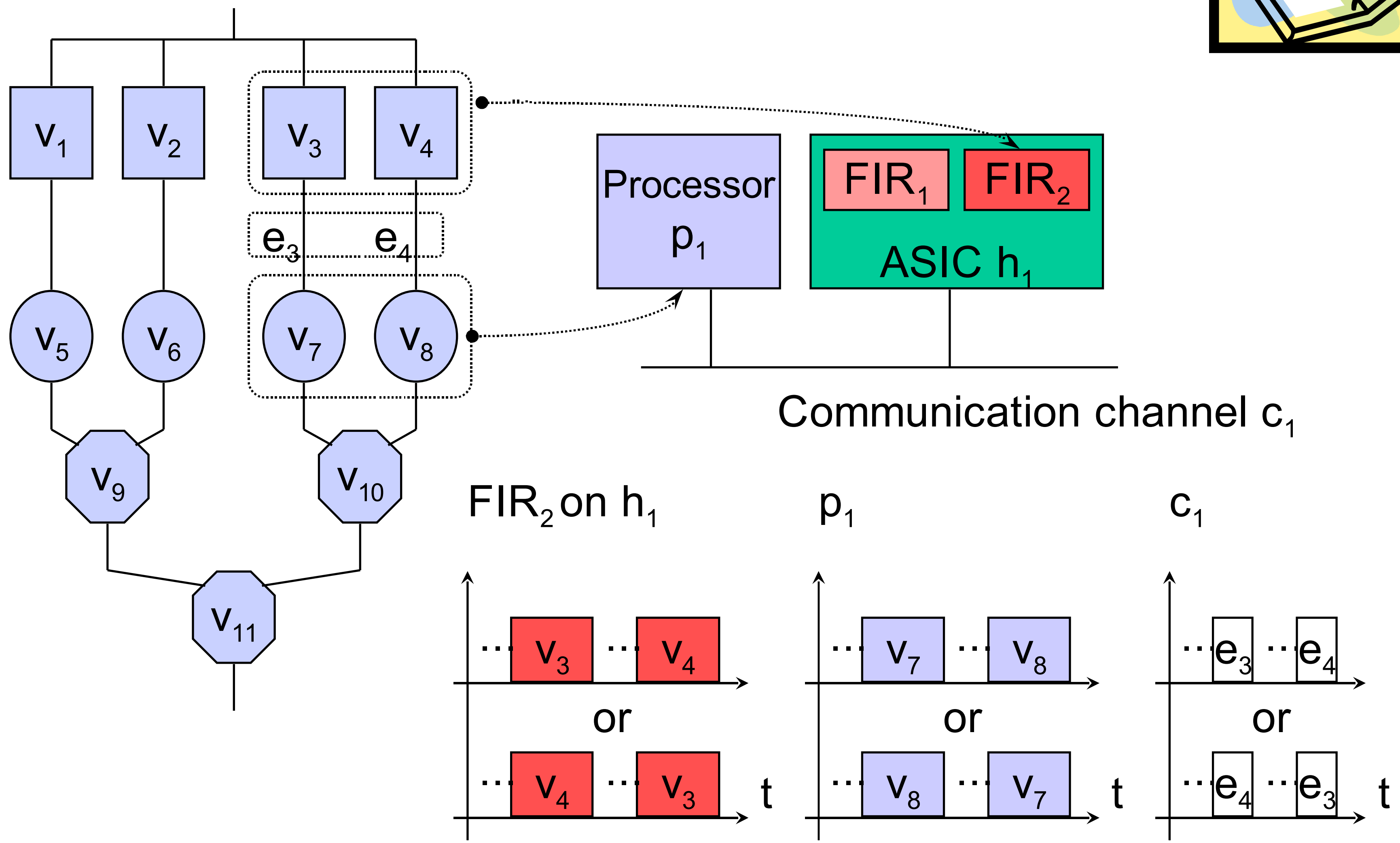
$\forall \ell \in L, \forall k \in KP: NY_{\ell,k} \leq 1.$

Resource & design constraints

- $\forall\, k \in KH$, the cost (area) used for components of that type is calculated as the sum of the costs of the components of that type. This cost should not exceed its maximum.
- $\forall\, k \in KP$, the cost for associated data storage area should not exceed its maximum.
- $\forall\, k \in KP$ the cost for storing instructions should not exceed its maximum.
- The total cost ($\Sigma_{k \in KH}$) of HW components should not exceed its maximum
- The total cost of data memories ($\Sigma_{k \in KP}$) should not exceed its maximum
- The total cost instruction memories ($\Sigma_{k \in KP}$) should not exceed its maximum

Scheduling





v1
v2
v3
v4
e3
e4
v5
v6
v7
v8
v9
v10
v11
Processor p1
FIR1
FIR2
ASIC h1
Communication channel c1
FIR2 on h1
p1
c1
v3
v4
or
v4
v3
t
v7
v8
or
v8
v7
t
e3
e4
or
e4
e3
t

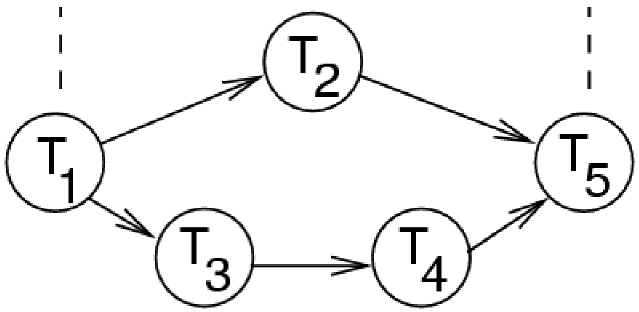
Scheduling / precedence constraints

- For all nodes v_{i1} and v_{i2} that are potentially mapped to the same processor or hardware component instance, introduce a binary decision variable $b_{i1,i2}$ with
  $b_{i1,i2}$=1 if v_{i1} is executed before v_{i2} and
  = 0 otherwise.
  Define constraints of the type
  (end-time of v_{i1}) $\leq$ (start time of v_{i2}) if $b_{i1,i2}$=1 and
  (end-time of v_{i2}) $\leq$ (start time of v_{i1}) if $b_{i1,i2}$=0
- Ensure that the schedule for executing operations is consistent with the precedence constraints in the task graph.
- Approach just fixes the order of execution and avoids the complexity of computing start times during optimization.

Other constraints

- **Timing constraints**
  These constraints can be used to guarantee that certain time constraints are met.
- Some less important constraints omitted ..

Example



HW types H1, H2 and H3 with costs of 20, 25, and 30.
Processors of type P.
Tasks T1 to T5.
Execution times:

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Operation assignment constraints (1)

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

$$\forall i \in I: \sum_{k \in KH} X_{i,k} + \sum_{k \in KP} Y_{i,k} = 1$$

$X_{1,1}+Y_{1,1}=1$ (task 1 mapped to H1 or to P)

$X_{2,2}+Y_{2,1}=1$

$X_{3,3}+Y_{3,1}=1$

$X_{4,3}+Y_{4,1}=1$

$X_{5,1}+Y_{5,1}=1$

Operation assignment constraints (2)

Assume types of tasks are ℓ =1, 2, 3, 3, and 1.

$\forall \ell \in L, \forall i{:}T(v_i)=c_\ell, \forall k \in KP{:}\ NY_{\ell,k} \geq Y_{i,k}$

$$NY_{1,1} \geq Y_{1,1}$$
$$NY_{2,1} \geq Y_{2,1}$$
$$NY_{3,1} \geq Y_{3,1}$$
$$NY_{3,1} \geq Y_{4,1}$$
$$NY_{1,1} \geq Y_{5,1}$$

Functionality 3 to be implemented on processor if node 4 is mapped to it.

Other equations

Time constraints leading to: Application specific hardware required for time constraints under 100 time units.

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Cost function:

C=20 #(H1) + 25 #(H2) + 30 # (H3) + cost(processor) + cost(memory)

Result

For a time constraint of 100 time units and cost(P)<cost(H3):

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Solution (educated guessing) :

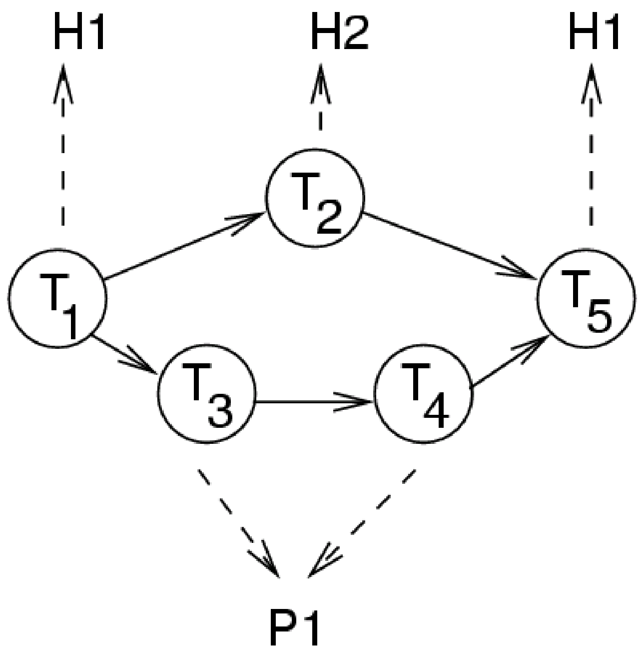
T1 → H1

T2 → H2

T3 → P

T4 → P

T5 → H1

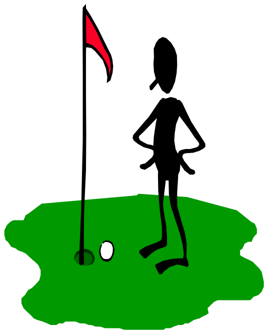


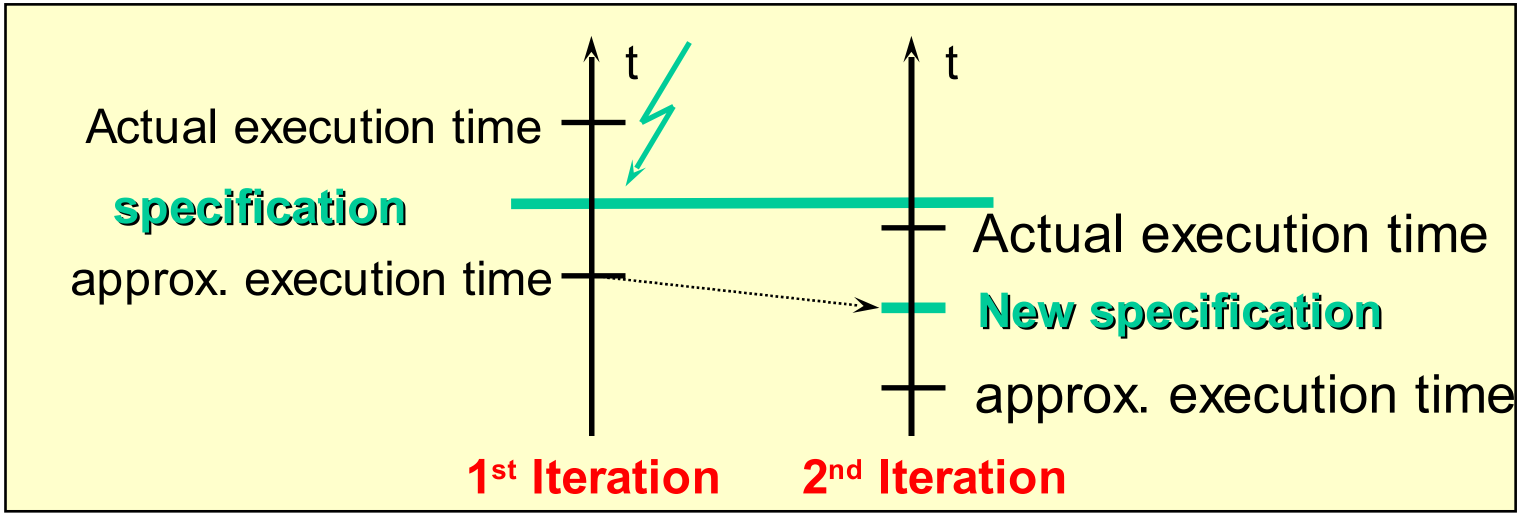
Separation of scheduling and partitioning

Combined scheduling/partitioning very complex;

☞ Heuristic: Compute estimated schedule

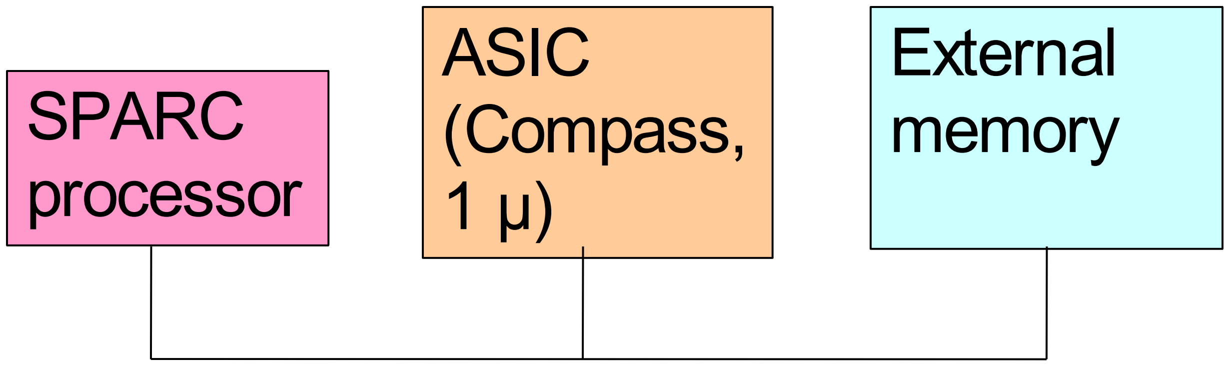
- Perform partitioning for estimated schedule
- Perform final scheduling
- If final schedule does not meet time constraint, go to 1 using a reduced overall timing constraint.





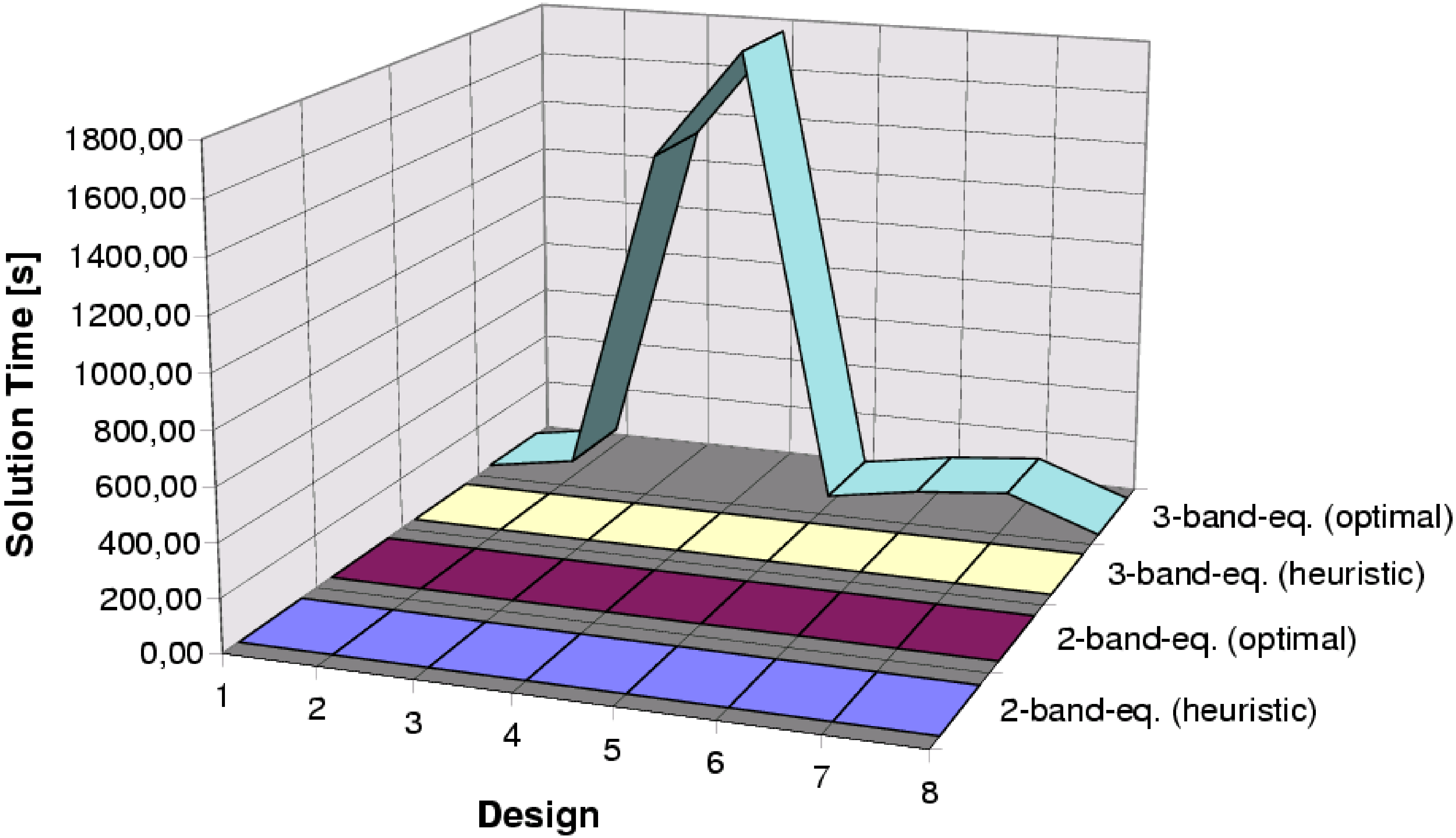
Application example

Audio lab (mixer, fader, echo, equalizer, balance units); slow SPARC processor
1µ ASIC library
Allowable delay of 22.675 µs (~ 44.1 kHz)



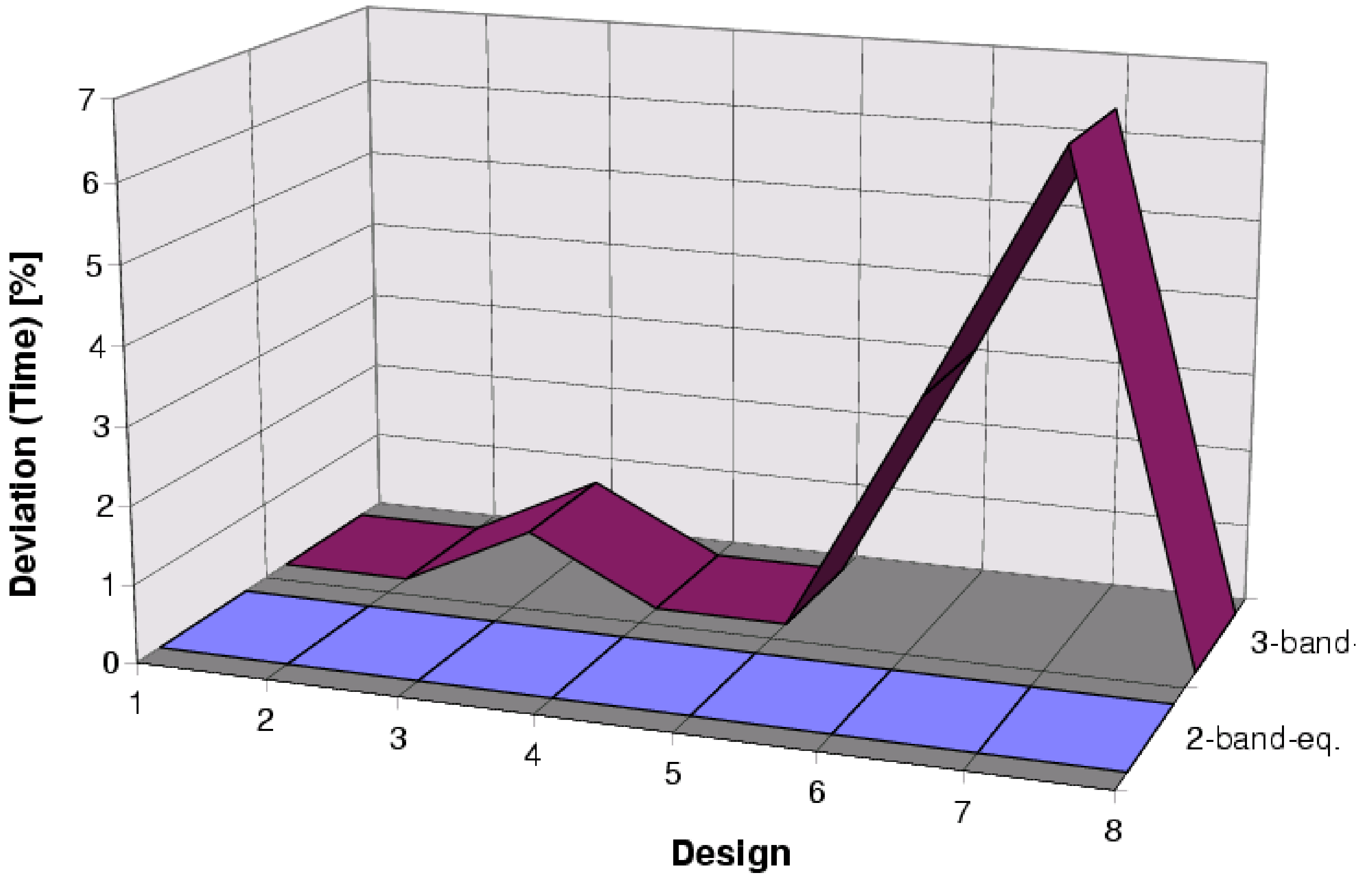
Outdated technology; just a proof of concept.

Running time for COOL optimization



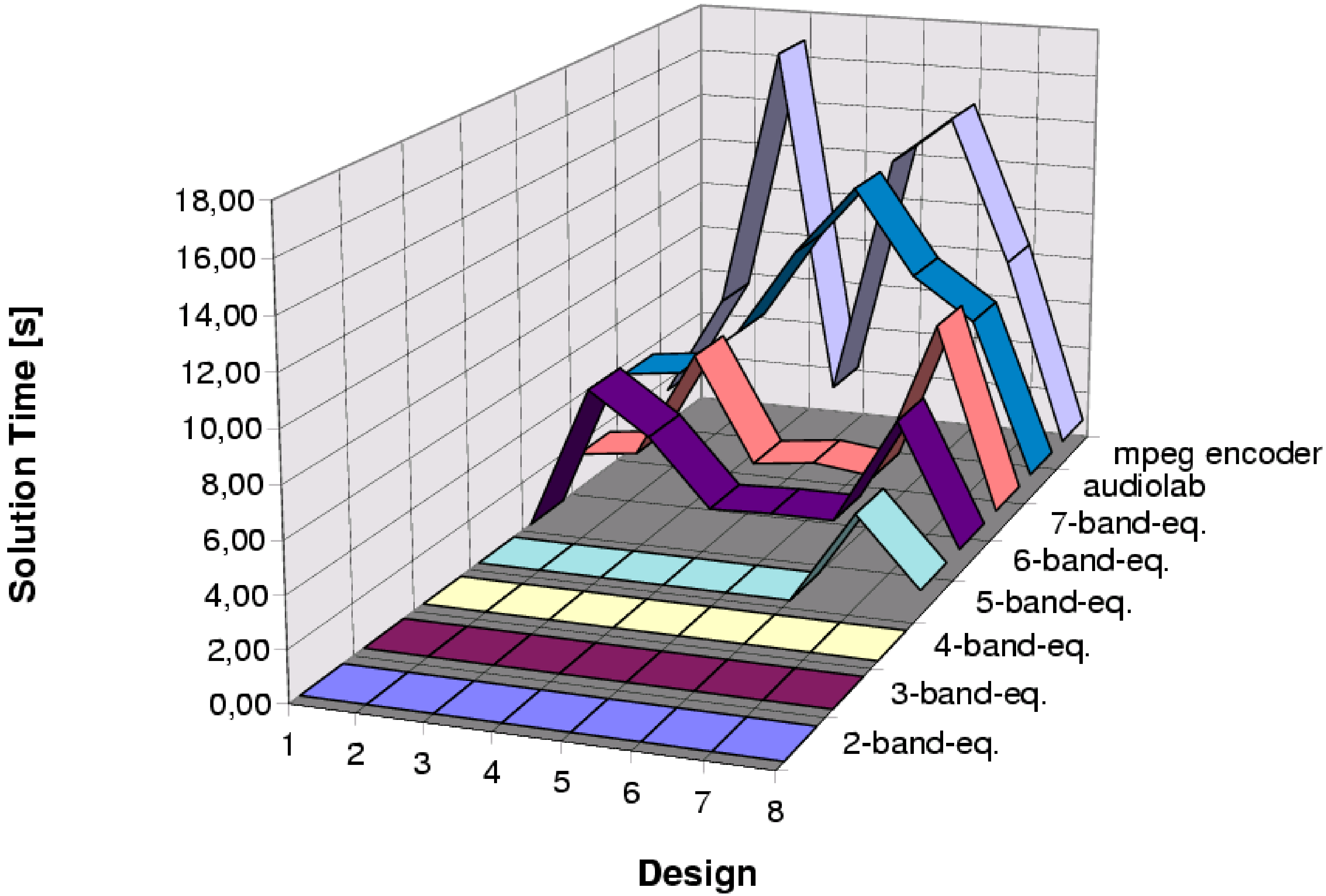
☞ Only simple models can be solved optimally.

Deviation from optimal design

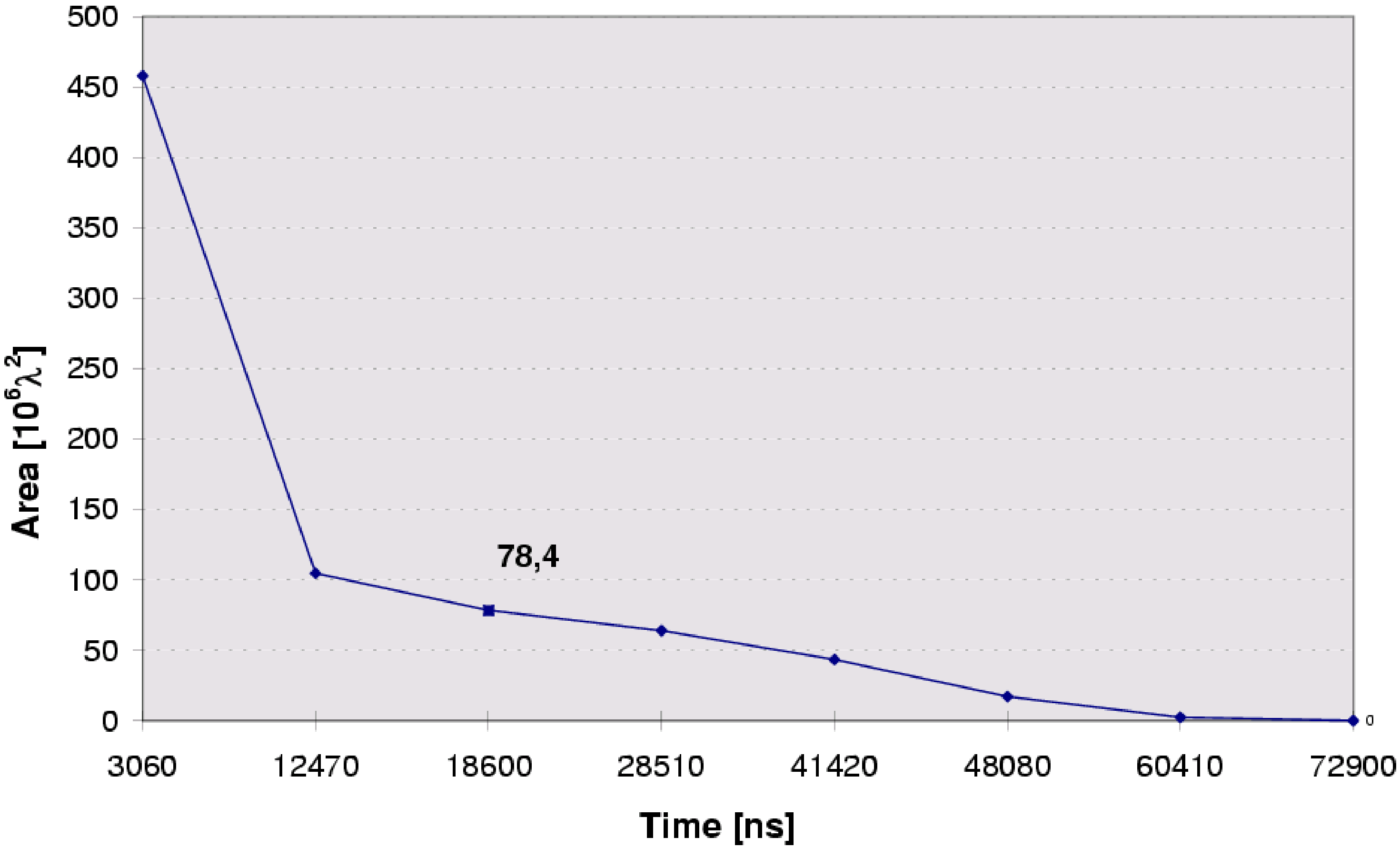


☞ Hardly any loss in design quality.

Running time for heuristic



Design space for audio lab



Everything in software: 72.9 µs, 0 λ^2

Everything in hardware: 3.06 µs, $457.9 \times 10^6\ \lambda^2$

Lowest cost for given sample rate: 18.6 µs, $78.4 \times 10^6\ \lambda^2$,

Positioning of COOL

COOL approach:

- shows that formal model of hardware/SW codesign is beneficial; IP modeling can lead to useful implementation even if optimal result is available only for small designs.

Other approaches for HW/SW partitioning:

- starting with everything mapped to hardware; gradually moving to software as long as timing constraint is met.
- starting with everything mapped to software; gradually moving to hardware until timing constraint is met.
- Binary search.

HW/SW partitioning in the context of mapping applications to processors

- Handling of heterogeneous systems
- Handling of task dependencies
- Considers of communication (at least in COOL)
- Considers memory sizes etc (at least in COOL)
- For COOL: just homogeneous processors
- No link to scheduling theory