Informatik
Professur Technische Informatik
Prof. Dr. Wolfram Hardt

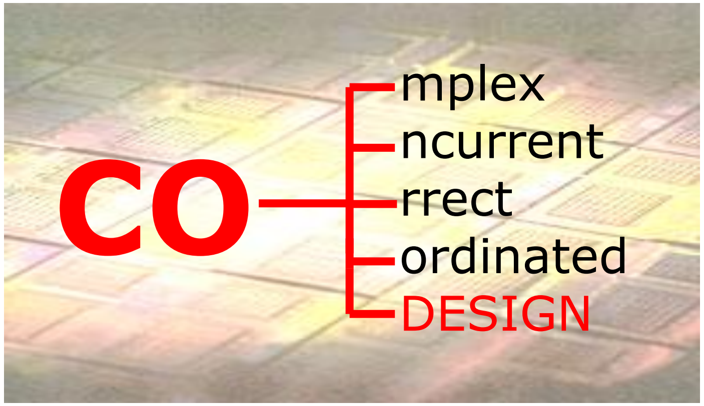
Hardware /Software Codesign I

System Partitioning

Prof. Dr. Wolfram Hardt
Dipl.-Inf. Michael Nagler

Contents

- Partitioning
- General Partitioning Algorithms
- HW/SW Partitioning Algorithms
- Examples



Partitioning - Abstraction

- ____________________ partitioning
  - register transfer level, net lists
  - system parameters are known quite good (area, delay, …)
  - no comparison of design alternatives
  - e.g. map a digital circuit onto two chips (FPGA, ASIC)
- ____________________ partitioning
  - system level
  - system parameter are not known → estimation required
  - compare different design alternative → design space exploration

Cost Functions

- measure quality of a design point
  - may include C … system cost (in [$])
    L … latency (in [sec])
    P … power consumption (in [W])
  - requires estimation to find C, L, P

- example

$$f(C,L,P) = k_1 \cdot h_C(C, C_{\max}) + k_2 \cdot h_L(L, L_{\max}) + k_3 \cdot h_P(P, P_{\max})$$

h_C, h_L, h_P … denote how strong C, L, P violate the design constraints $C_{max}, L_{max}, P_{max}$

k_1, k_2, k_3 … weighting and normalisation

General Partitioning Problem

- **definition**

  The general partitioning problem is to assign n objects $O = \{o_1, ..., o_n\}$ to m blocks (partitions) $P = \{p_1, ..., p_m\}$, such that

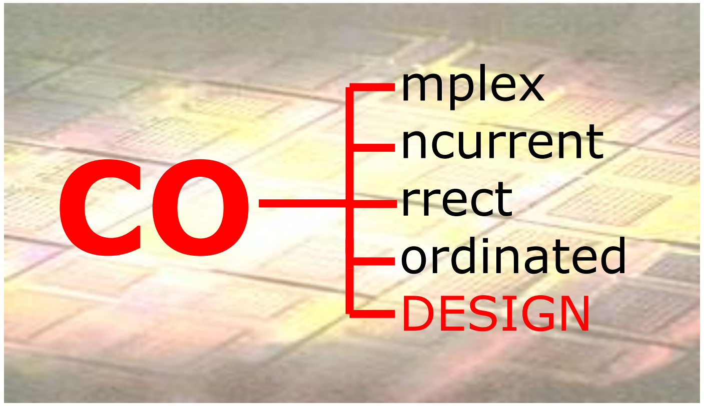
  – ____________________________
  – ____________________________
  – ____________________________

- the general partitioning problem is NP-complete

- in case of system synthesis:
  – objects O = *problem graph* nodes
  – blocks P = *architecture graph* nodes

Contents

- Partitioning
- General Partitioning Algorithms
- HW/SW Partitioning Algorithms
- Examples



Classification

- ______________________ algorithms
  - constructive algorithms
    - random mapping
    - hierarchical clustering
  - iterative algorithms
    - Kerninghan-Lin
    - simulated annealing
    - evolutionary algorithms (design space exploration)

accuracy?
local minimum?

- ______________________ algorithms
  - enumeration of solutions
  - Integer Linear Programs (ILP)

very high
computing effort!

Constructive Algorithms

- often used to generate a valid start partition for iterative algorithms (initial partition)

- possibly difficult to find a suitable closeness function

- algorithms
  - random mapping (each object is randomly assigned to some block)
  - hierarchical clustering

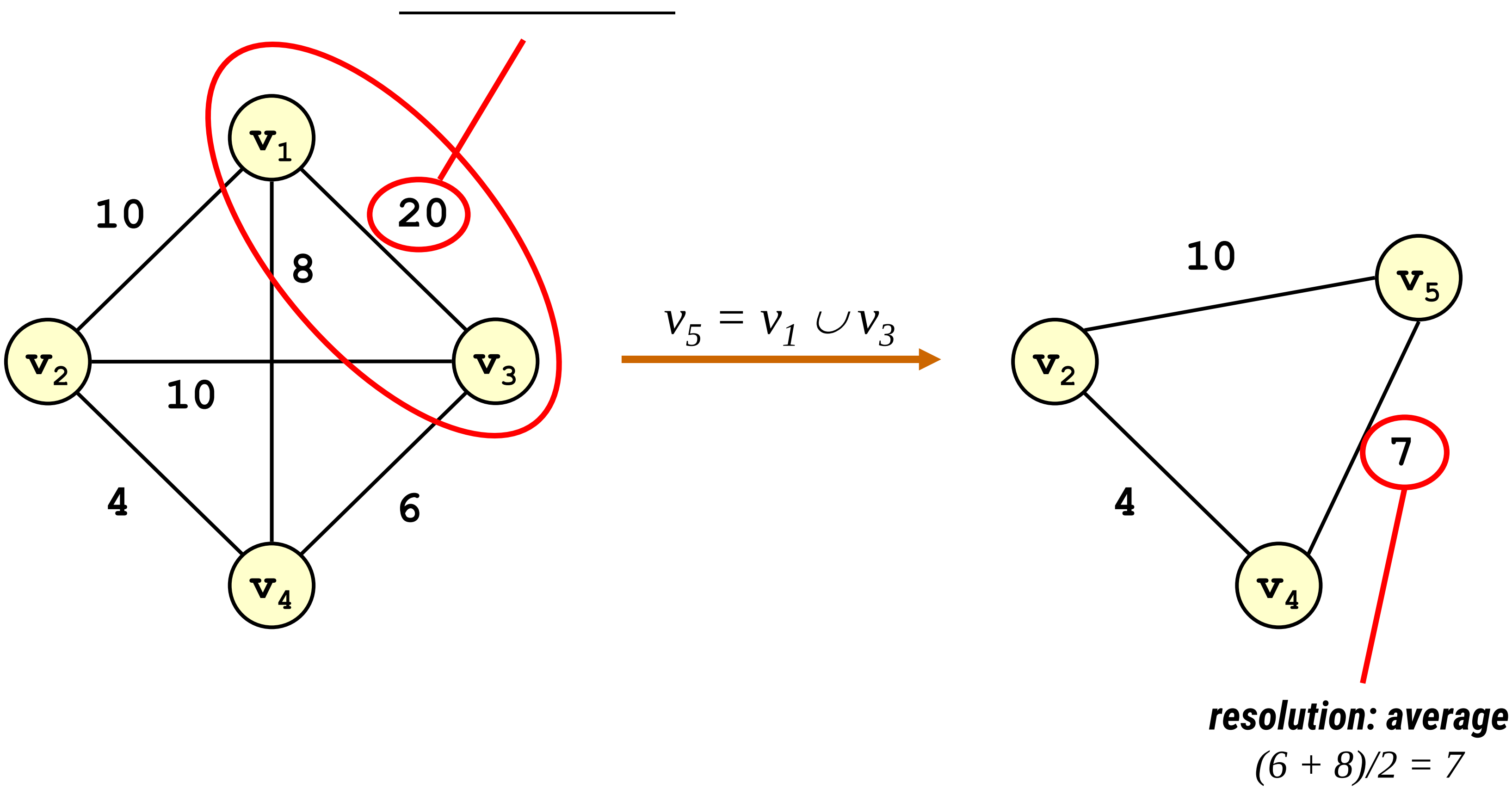
Hierarchical Clustering

- complete connected graph $G = (V, E)$
  - V … set of partitions
  - E … relation of each partition pair
- ______________________ $f : E \rightarrow \Re$
  - assigns real number to each edge of G
  - determines how desirable the grouping of two partitions is
- stepwise grouping of two appropriate partitions
- time complexity: $O(n^2)$

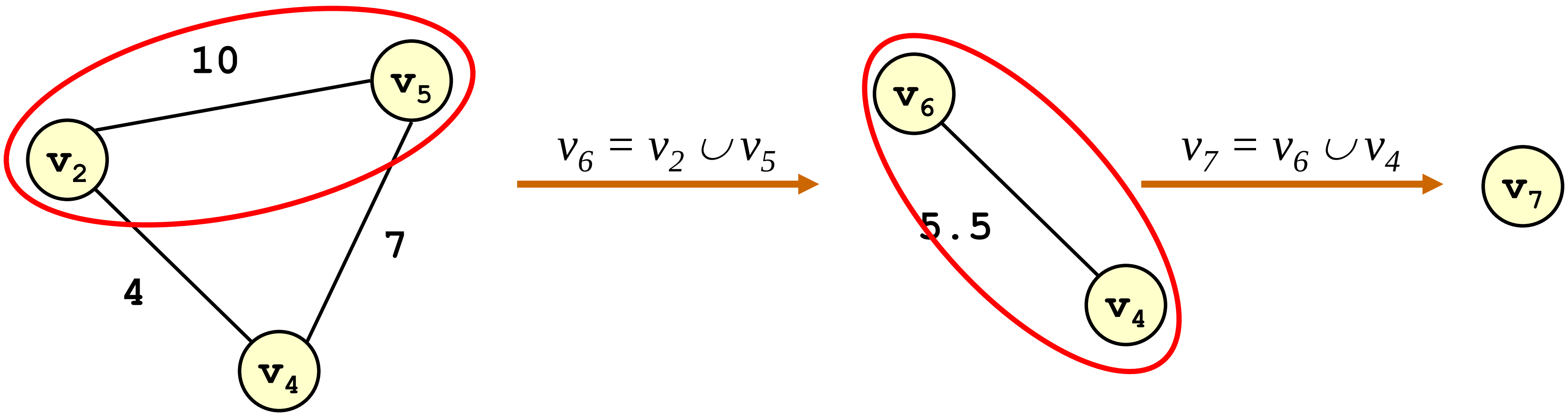
Algorithm

```
HierachicalClustering(O,f)
{
    // put every object to its own block
    for (int i = 0; i < N; i++) p[i] += {o[i];

    // calculate closeness between the objects
    for (int i = 0; i < N; i++)
          for (int j = 0; j < N; j++)
                   CalculateCloseness(p[i],p[j]);

    // combine of objects and recalculation of closeness
    k := N + 1;
    while (BreakCondition(p) == false)
    {
          (x,y) := BestPair(p);          // get partitions p[x], p[y] to combine
          p[k] := Union(p[x], p[y]);
          p[x] := null; p[y] := null;    // "delete" old partitions
          N := N - 1;                    // 1 partition created, 2 deleted

          RecalculateCloseness(p);
          k := k + 1;
    }
}
```

Example (I)


10
20
8
10
4
6
v1
v2
v3
v4
$v_5 = v_1 \cup v_3$
10
4
7
v2
v4
v5
resolution: average
$(6 + 8)/2 = 7$

Example (II)

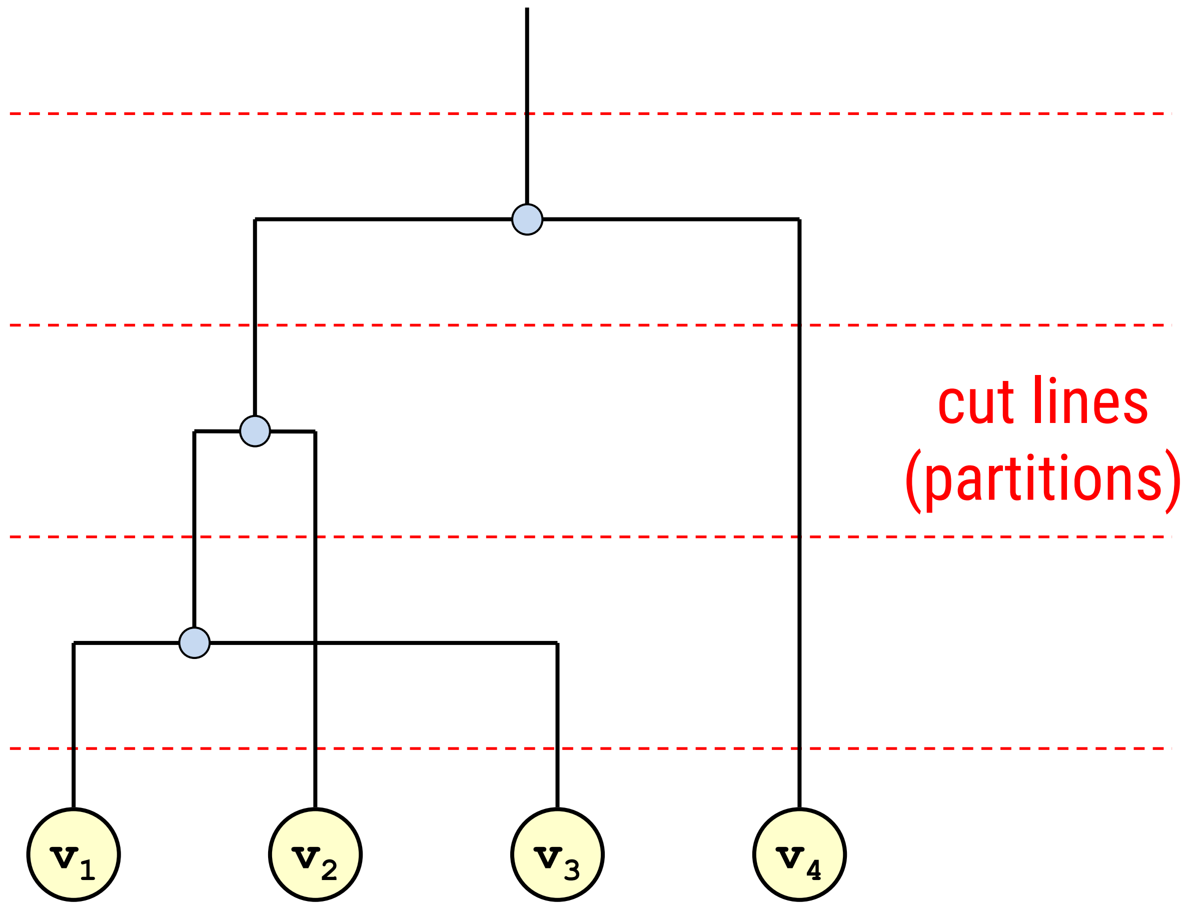


Example (III)

step 3:
$v_7 = v_6 \cup v_4$

step 2:
$v_6 = v_2 \cup v_5$

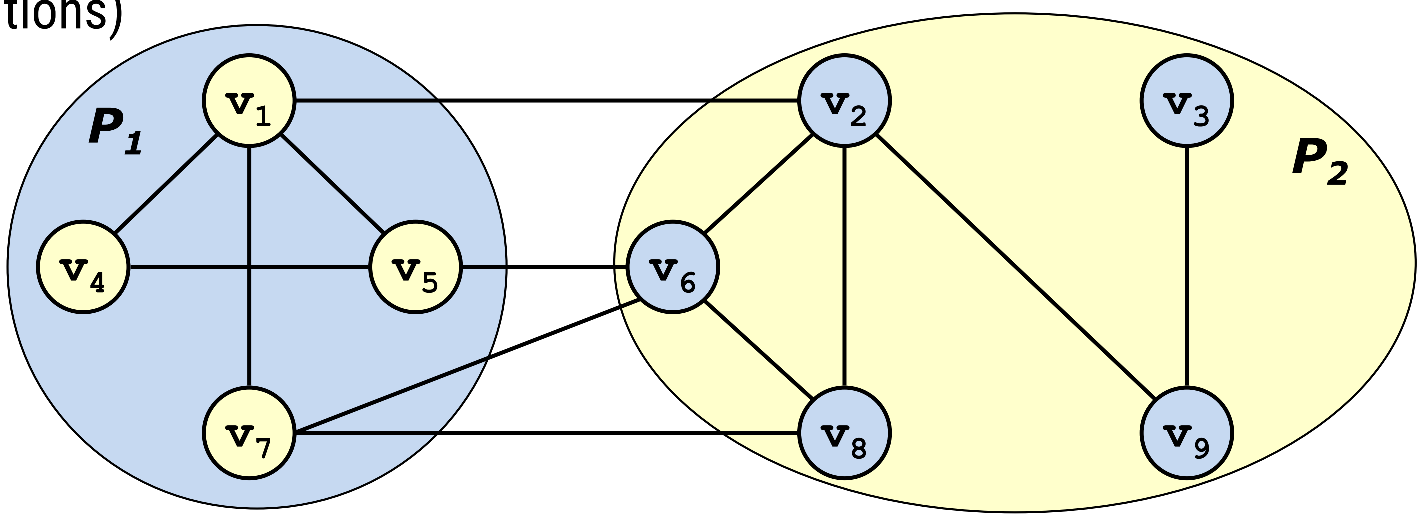
step 1:
$v_5 = v_1 \cup v_3$



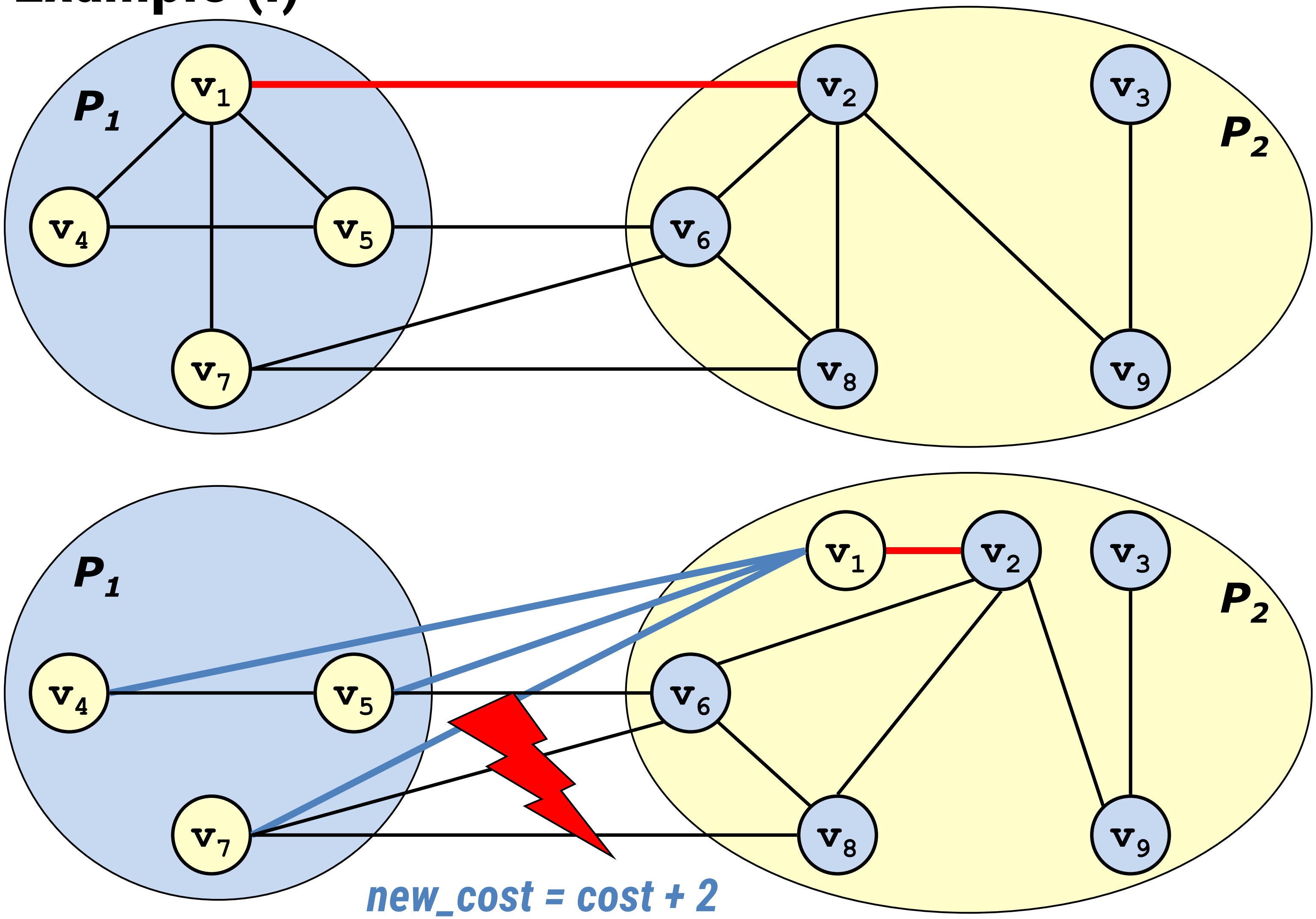
Kernighan-Lin Algorithm

- generation of bi-partitions
  - calculate cost benefit for all objects, ______________________________
    ____
  - move object with most benefit

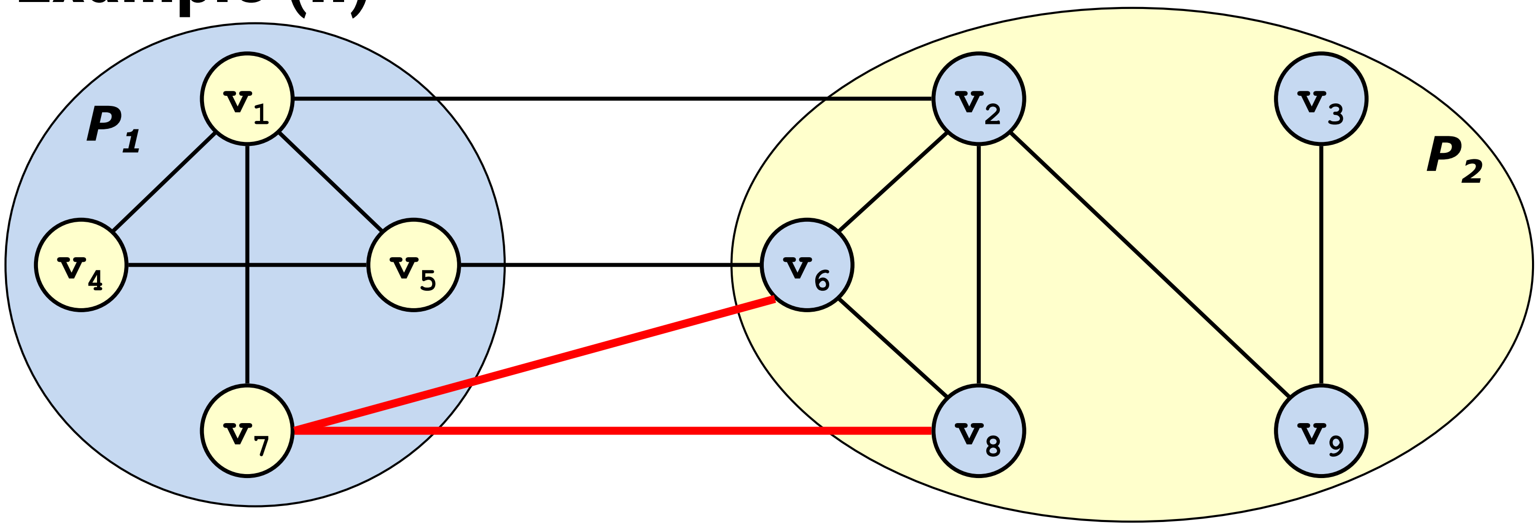
- example: minimum cut (minimise number of edges between partitions)

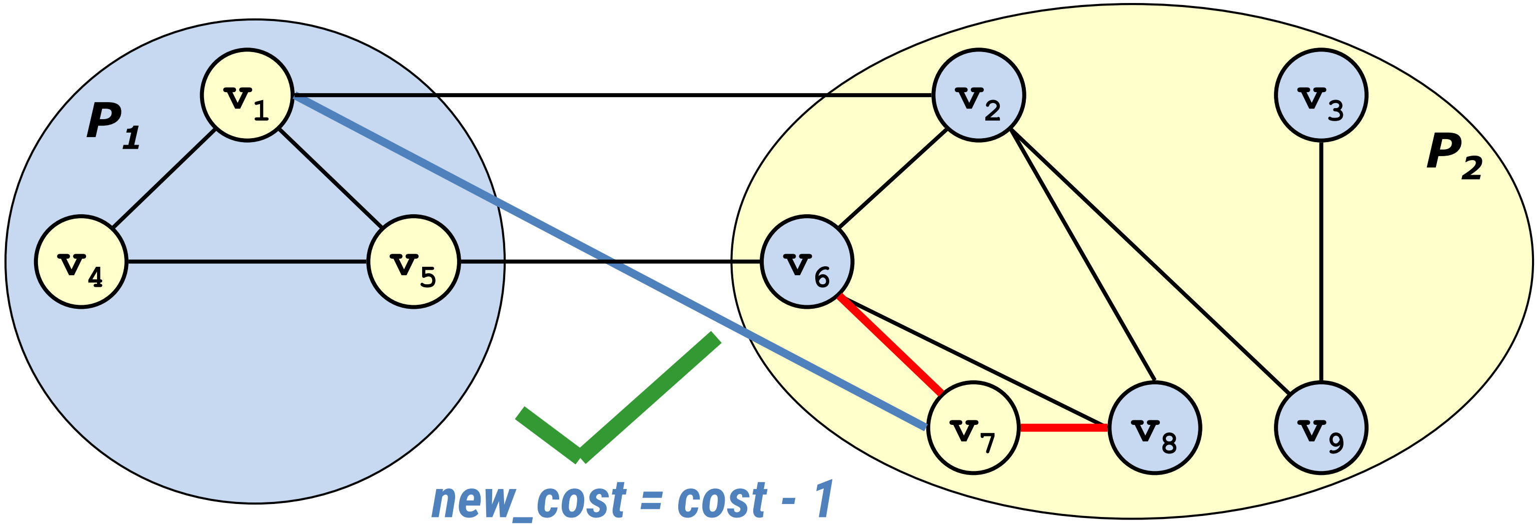


Example (I)


P1
v1
v4
v5
v7
v2
v3
v6
v8
v9
P2
P1
v4
v5
v7
v1
v2
v3
v6
v8
v9
P2
new_cost = cost + 2

Example (II)


P1
v1
v4
v5
v7
P2
v2
v3
v6
v8
v9


P1
v1
v4
v5
P2
v2
v3
v6
v7
v8
v9
new_cost = cost - 1

Kerninghan-Lin Extension

- move the object into the other partition that leads to the highest cost reduction ____

  → leave the local minimum

- algorithm
  - as long as a better partition is found:
    - move the best one of the n objects by trial and error
    - continue with all other n-1 objects
    - chose from this n partitions the cheapest one
    - activate the relevant movement

- time complexity: _______
- partitioning into m blocks: _______

Simulated Annealing (I)

- metal and glass take on minimal energy states when they are cooled down under certain conditions
  - for each temperature, thermodynamic equilibrium is reached
  - the temperature is decreased arbitrarily slow
- generalisation
  - temperature is fixed → change parameters
  - wait until balance is established → optimisation based on this new parameters
- time complexity:
  - from ________________________________, depending on the implementation of **`Equilirium(), DecreaseTemp(), Frozen()`**
  - the longer the runtime, the better the result
  - usually functions are constructed to get polynomial runtime

Simulated Annealing (II)

```
temp = temp_start;
cost = c(P);
while (Frozen() == false)
{
  while (Equilibrium() == false)
  {
    P_n = RandomMove(P);
    cost_n = c(P_n);
    deltacost = cost_n - cost;

    if (Accept(deltacost, temp) > random(0,1))
    {
      P = P_n;
      cost = cost_n;
    }
  }

  temp = DecreaseTemp(temp);
}
```

$$Accept() = \min(1, e^{-\frac{deltacost}{k \cdot temp}})$$

Simulated Annealing (III)

- `DecreaseTemp()` `(temp_start = 1.0)`
  – $temp = \mu * temp$ (typical: $0.8 \leq \mu \leq 0.99$)
- `Frozen()`
  – gets `true` when ______________________________ or if there is no more improvement
- `Equilibrium()`
  – gets true after certain number of iterations or if there is no more improvement
- `RandomMove()`
  – moves randomly objects between partitions

(Integer) Linear Programs

- **LP are a method to optimise a objective function of a set**
- the set is constricted by ______________________________
- used to solve problems without a specialised solving method
- ILP is special case: only whole numbers are allowed as solutions
- problem has to be mathematically modelled and can be solved with existing (I)LP algorithms (solver)
- (I)LP problem is in principle NP-complete

Integer Linear Programs (I)

- mathematically modelling of partition problem:
  - binary variables $x_{i,k} = 1 \leftrightarrow$ object o_i in block p_k
  - cost $c_{i,k}$, if object o_i in block p_k
  - integer linear program:

$$x_{i,k} \in \{0,1\} \qquad 1 \le i \le n,\ 1 \le k \le m$$

$$\sum_{k=1}^{m} x_{i,k} = 1 \qquad 1 \le i \le n$$

objective function:

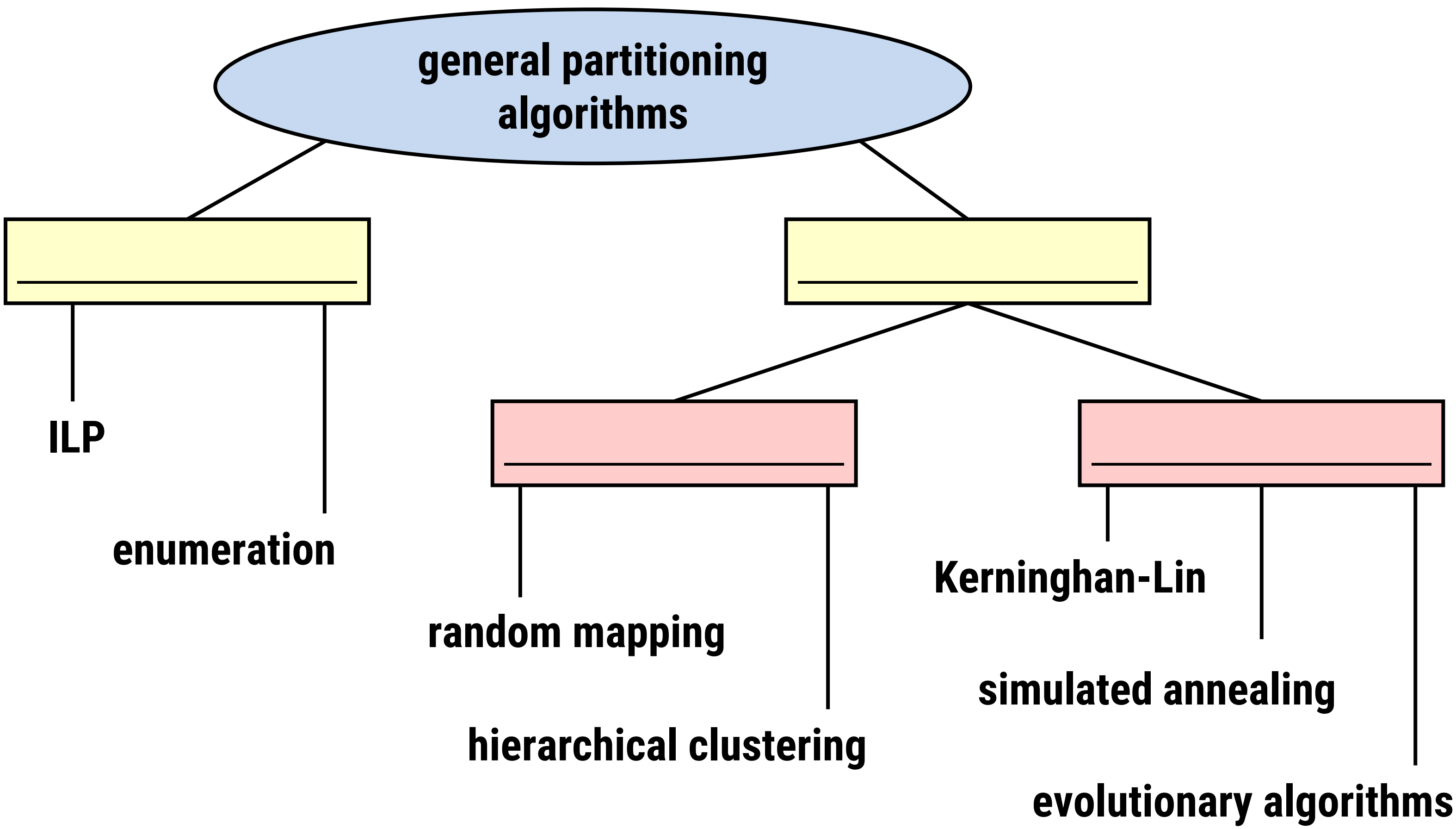
Integer Linear Programs (II)

- limits modelled by constraints
- maximal numbers of h_k objects in block p_k

$$\sum_{i=1}^{n} x_{i,k} \le h_k \qquad 1 \le k \le m$$

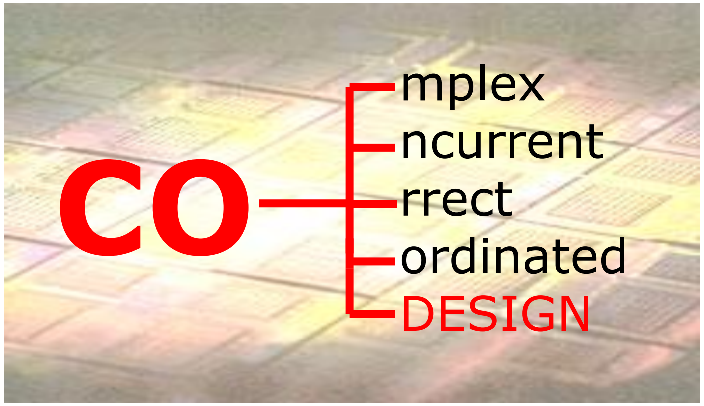
- maximal cost H_k of objects in block p_k

Summary


general partitioning algorithms
ILP
enumeration
random mapping
hierarchical clustering
Kerninghan-Lin
simulated annealing
evolutionary algorithms

Contents

- Partitioning
- General Partitioning Algorithms
- HW/SW Partitioning Algorithms
- Examples



HW/SW Partitioning Problem

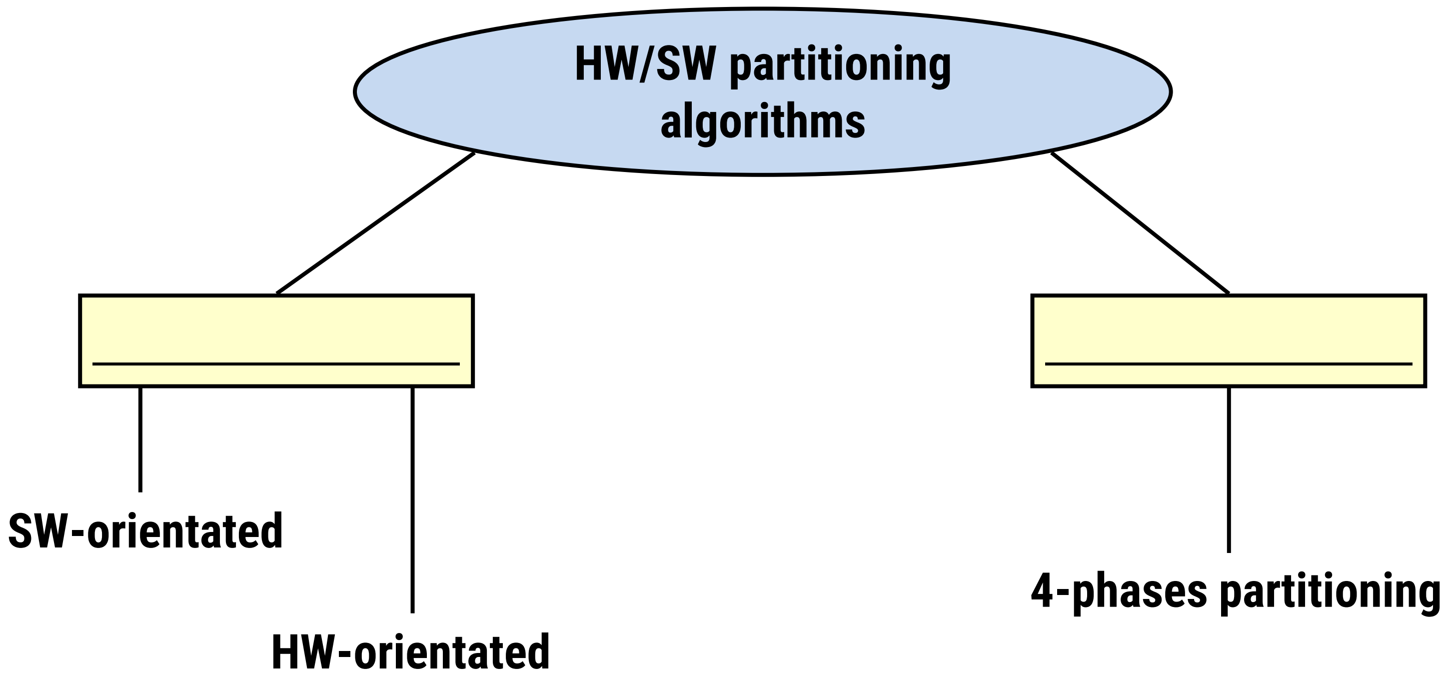
- *remember*: general partitioning problem

  The partitioning problem is to assign n objects $O = \{o_1, \ldots, o_n\}$ to m blocks (partitions) $P = \{p_1, \ldots, p_m\}$, such that

  - $p_1 \cup p_2 \cup \ldots \cup p_m = O$
  - $\forall i, j \colon i \neq j \Rightarrow p_i \cap p_j = \varnothing$
  - cost $c(P)$ are minimised

- **HW/SW partitioning is special case: bi-partitioning**

Classification



Greedy Partitioning

- migration (***move***) of objects into the other block (HW or SW) until ________________________________ (***costs***)

- algorithm:

```
do
{
  partition_old = partition;

  for (int i = 1; i <= n; i++)
  {
    if (costs(move(partition,o[i]) < costs(partition))
      partition = move(partition, o[i]);
  }

} while (partition != partition_old)
```

Start Partitioning

- software-orientated approach
  - **start greedy with partitioning: ____________**
  - all functions can be realised in SW
  - performance might be too low → migrate objects to HW

- hardware-orientated approach
  - **start greedy with partitioning: ____________**
  - the performance is sufficient in HW
  - costs might be too high → migrate objects to SW

4-Phases Partitioning (I)

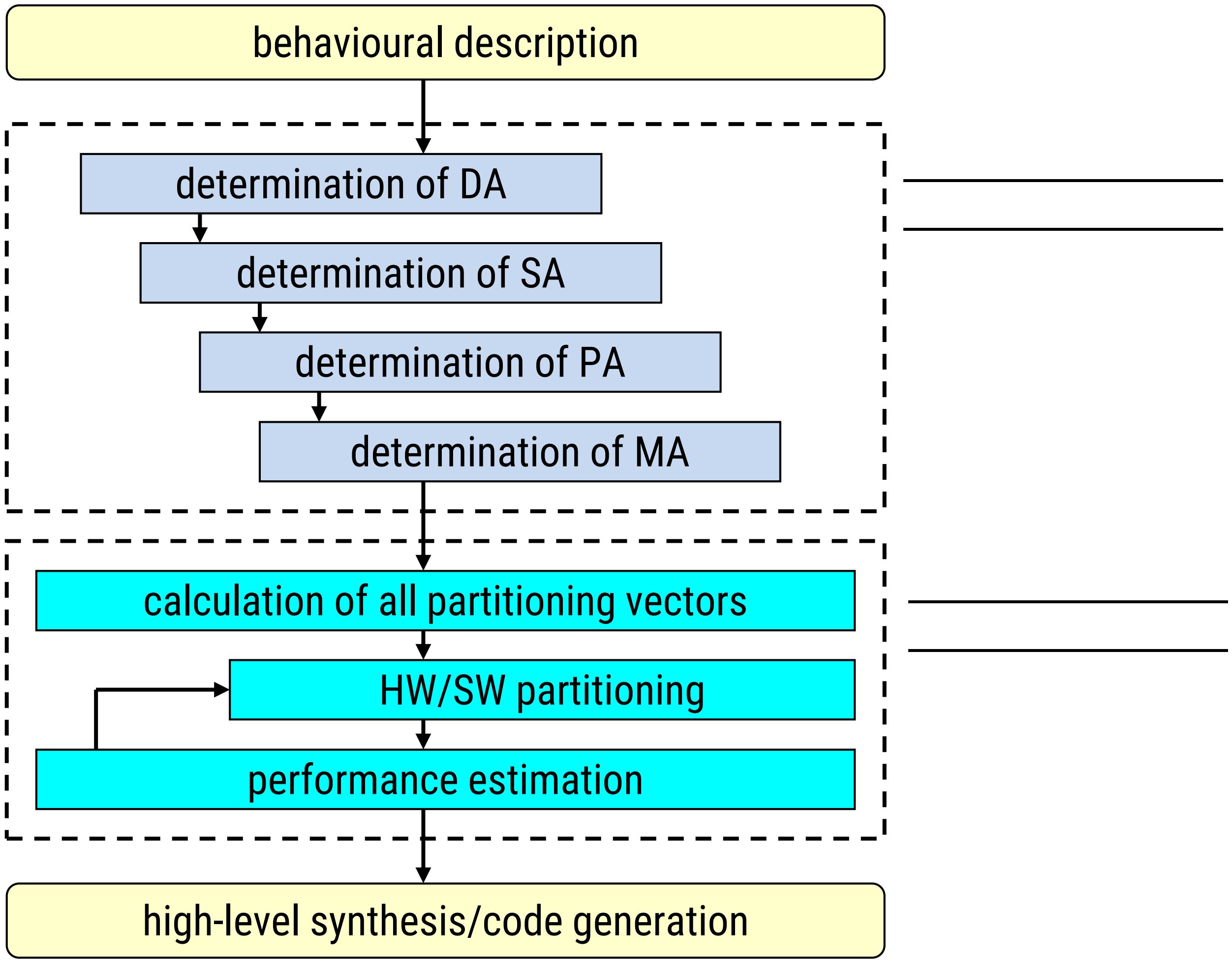
- input: program in ANSI C or C++
  - no HW specific extensions
  - many applications
- abstraction level: module (= C function/method)
- method: ______________________________
  - automatic determination of graph weightings
  - 4 partitioning criterions
  - designer can insert experiences by different weighting constants
- time complexity: ________

4-Phases Partitioning (II)

- partitioning criterions for modules
  - dynamic execution time (DA)
  - statically determinable execution time (SA)
  - interface parameter (PA)
  - memory access (MA)
- **formalism**
  - characterisation of each module mod by partitioning vector
    $$P(mod) = (DA(mod), SA(mod), PA(mod), MA(mod))$$
  - weighting vector for criterions
    $$Cut_{HW/SW} = (W_{DA}, W_{SA}, W_{PA}, W_{MA})$$
  - partitioning function Φ for module mod
    $$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = P(mod) - Cut_{HW/SW}$$
    $$\Phi(mod) = \begin{cases} 1, & \forall i \in [1,4]: \ \alpha_i > 0 \\ 0, & else \end{cases}$$

mod to HW: $\Phi(mod) = 1$
mod to SW: $\Phi(mod) = 0$

Algorithm


behavioural description
determination of DA
determination of SA
determination of PA
determination of MA
calculation of all partitioning vectors
HW/SW partitioning
performance estimation
high-level synthesis/code generation

Determination of DA

- execution time of a C-function (module) varies
  - __________ = determined runtime depending on number of execution of *mod*
  - __________ = relative frequency of runtime to runtime of whole system
  - __________ = average runtime depending on number of execution
- many measurements necessary for reliable determination
- cost function

normalisation constants

$$DA(mod) = \frac{RT_{abs}^{SW}(mod)}{SW\_ABS\_RT} + \frac{RT_{rel}^{SW}(mod)}{SW\_REL\_RT} + \frac{RT_{ave}^{SW}(mod)}{SW\_AVE\_RT}$$

Determination of SA

- different characteristics in SW or HW for different instructions
  - jumps ($Jump$) → interferes pipelining in SW
  - bit manipulating instructions ($Bitop$) → usually only one bit / cycle

- count of all instructions ($Inst$) for calculation of relative frequency of instructions

- cost function

estimated complexity of a SW implementation

$$SA(mod) = \frac{Jump(mod) + Bitop(mod)}{Inst(mod)} \cdot 100\% \cdot \frac{RT_{approx}^{SW}(mod)}{SW\_APPROX\_RT}$$

Determination of PA

- ________________________
  - data type (implicit data size, e.g. **`type = char`** → **`size = 8 bit`**)
  - access type (read, write, r/w)

- pointer and global variables not correctly determined

- cost function

given interface width of HW implementation

$$PA(mod) = \frac{SW\_INTERF\_WIDTH}{max(Width_{IN}(mod), Width_{OUT}(mod), Width_{InOut}(mod))}$$

Determination of MA

- ______________________________
  - local and global data
  - direction of data transfer (load, store)
  - "distance" between source and destination (→ access times may varies)

- compare efficiency of data transfers in SW and HW

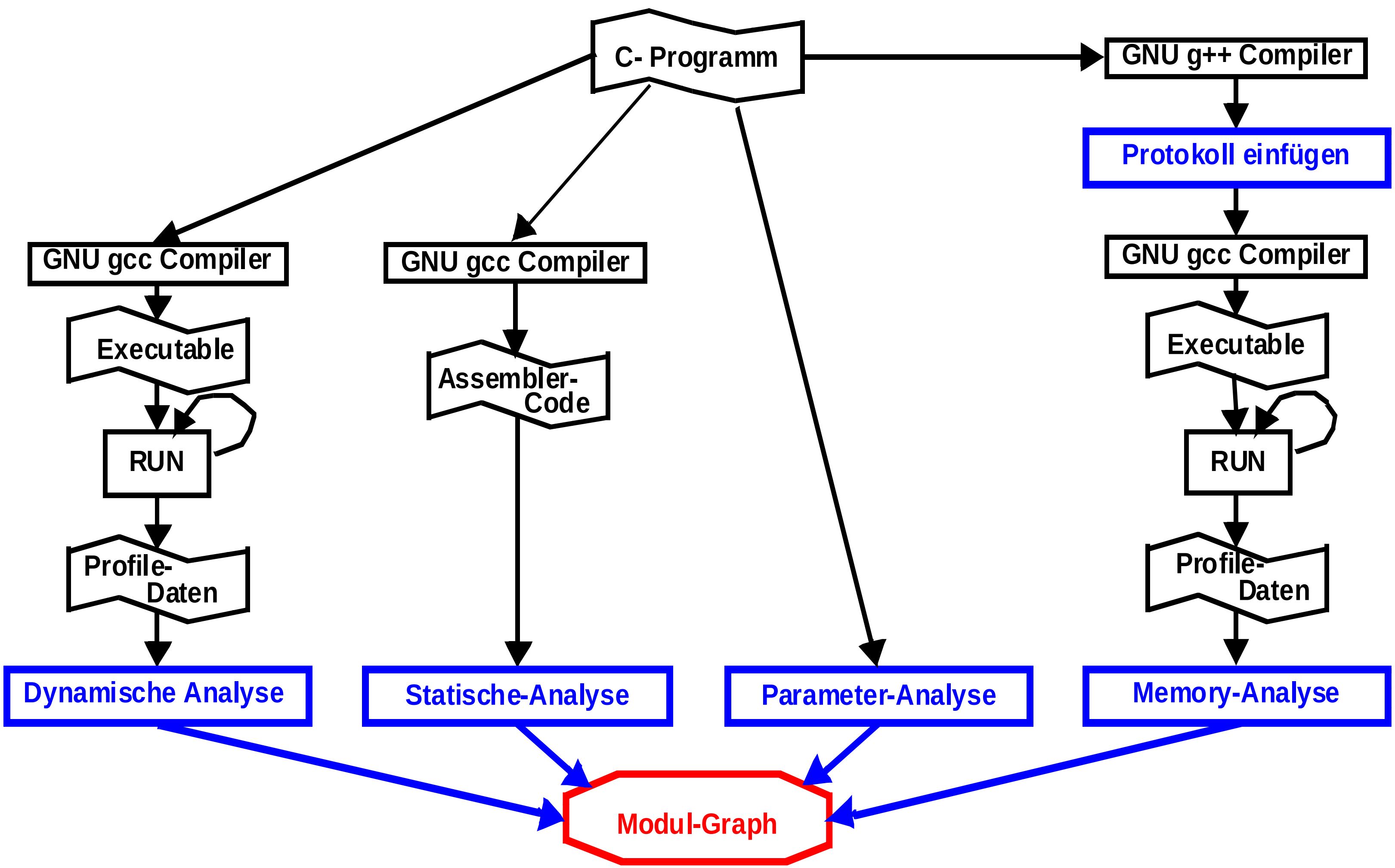
$$\eta_{DT}(mod) = \frac{DT^{SW}(mod)}{DT^{HW}(mod)}$$

costs for data transfers of mod in SW implementation

- cost function

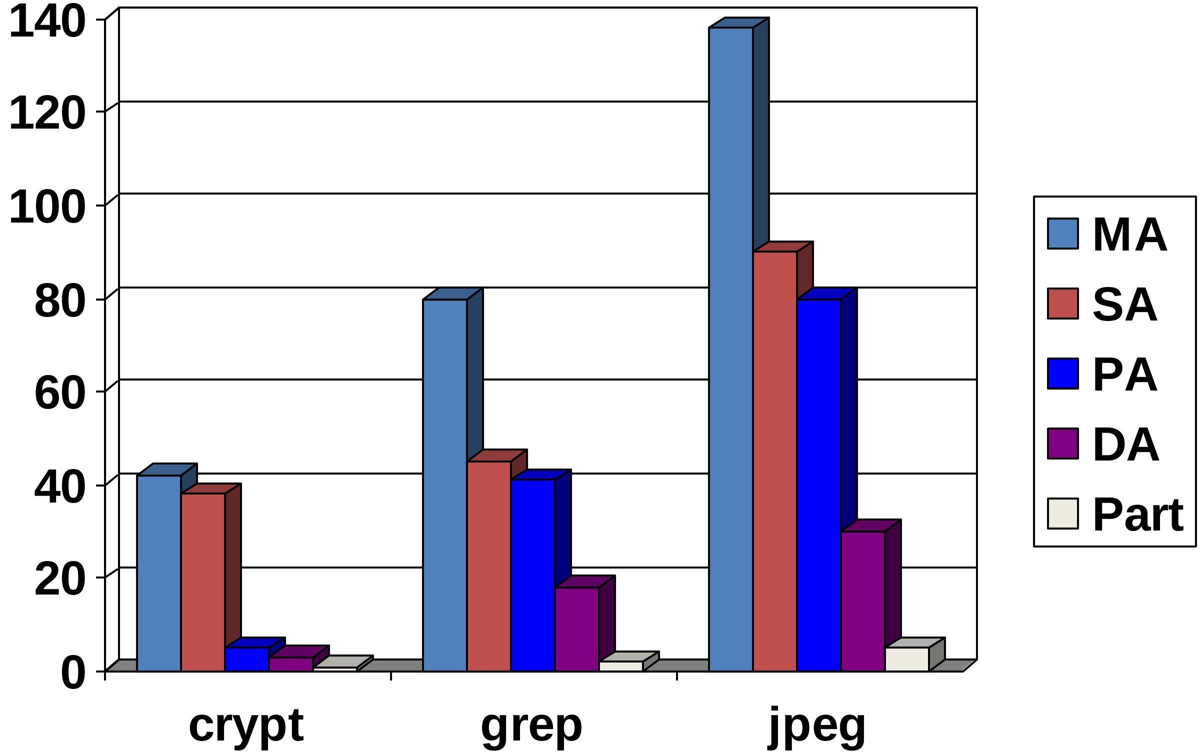
$$MA(mod) = \begin{cases} \eta_{DT}(mod), & \eta_{DT}(mod) > 1 \\ 0, & else \end{cases}$$

Implementation

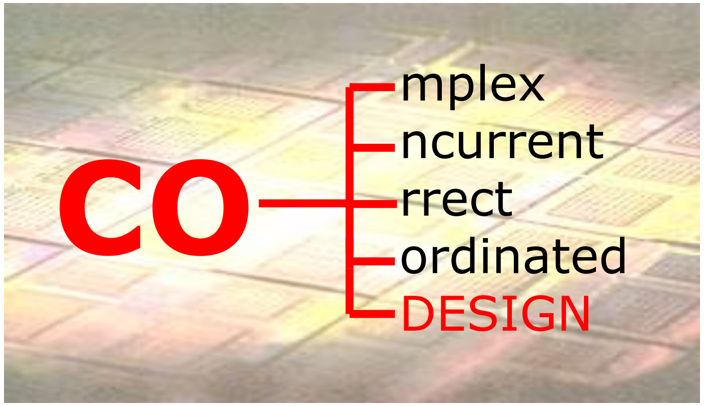


Results

- example



Contents



Yorktown Silicon Compiler (YSC)

- functional partitioning of hardware
  - *input:* functional description on the level of arithmetic and logical expressions
  - *target:* partitioning to several chips
  - *abstraction level:* functional units of data paths (ALUs, registers)
  - method: ________________________
    - closeness function:

constant

$$closeness(p_i, p_j) = \left(\frac{sharedwires(p_i, p_j)}{maxwires(P)}\right)^{c_2} \cdot \left(\frac{maxsize}{min(size(p_i), size(p_j))}\right)^{c_3} \cdot \left(\frac{maxsize}{size(p_i) + size(p_j)}\right)$$

Vulcan

- HW/SW bi-partitioning
  - *input:* program in *HardwareC*
    - C code, extended by a process concept and interprocess communication
    - specification with constraints (min/max-times, data rates)
  - *target architecture:* 1 processor, 1 ASIC
    - 1 global bus (processor is master) and 1 global memory
  - *abstraction level:* basic blocks and operations
    - deterministic execution times
  - *method:* HW orientated greedy
    - cost function includes HW costs, memory requirements, performance and synchronisation effort

Cosyma

- HW/SW bi-partitioning
  - *input:* program in C^x
    - C code, extended by a process concept and interprocess communication
    - specification of min/max times
  - *target architectures:* processor, coprocessor
    - coupled by a shared memory
    - computations on the processor and the coprocessor may not overlap in time
  - *abstraction level:* basic blocks
  - *method:* 2 loops
    - inner loop: simulated annealing with cost function that gives the estimated time gain for a HW realisation of a block
    - outer loop: synthesis to improve the estimations for the inner loop