## System Partitioning

**Kris Kuchcinski**

**Krzysztof.Kuchcinski@cs.lth.se**



## Partitioning

"He who can properly define and divide is to be considered a god."

**Plato (ca 429-347 BC)**



## System Partitioning

- The functionality of a system is implemented with a set of interconnected system components, such as ASIC's, memories, CPU's, buses.
- The designer must solve two problems:
  - select a set of system components (**allocation**),
  - partition the system's functionality among these components (**partitioning**).
- The final implementation has to satisfy a set of design constraints, such as cost, performance and power consumption.



## Structural Partitioning

- First the system components are implemented using interconnected hardware components.
- Partitioning separates the objects into groups, where each group represents a system component.
- Mostly used at lower levels of abstraction for hardware partitioning.
- Satisfies certain constraints (for instance packaging).
- Problems:
  - size/performance trade-offs are difficult,
  - large number of objects.



## Functional Partitioning

- The system level *functionality* is partitioned in order to divide the behaviour of the system between multiple components.
- Usually executable model is partitioned and therefore the estimation of parameters and partitioning results is possible.
- Advantages:
  - size/performance trade-offs,
  - small number of objects,
  - hardware/software solutions.



## Partitioning Granularity

- Coarse granularity
  - deals with processes, subprograms, blocks of statements,
  - typical for system-level synthesis,
  - deals with a relatively small number of objects.
- Fine granularity
  - performed at operation level,
  - used during high-level synthesis,
  - high complexity.

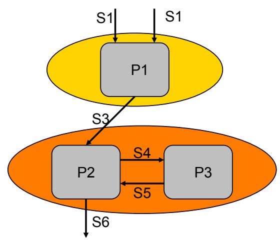
## Abstract Representation

- Structure.
- Register transfer.
- FSM with datapath.
- Control/data-flow graph (CDFG)
  - appropriate for operation level partitioning (HLS).
- Task
  - appropriate for system level partitioning.



## Task Partitioning

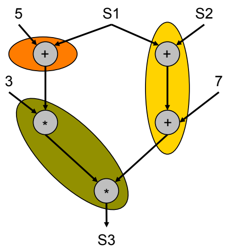
```
• • •
signal S1, S2, S3, S4, S5, S6: INTEGER;
• • •
P1: process
            variable A, B: INTEGER;
begin
            • • •
            A:=(S1+5)*3;
            B:=S1+S2+7;
            S3<=A*B;
            • • •
end process;

P2: process
            variable X, Y: INTEGER;
begin
            • • •
            wait on S3;
            S4<=S3+X;
            • • •
            wait on S5;
            S6<=S5*Y;
end process;

P3: process
            variable Z: INTEGER;
begin
            wait on S4;
            • • •
            S5<=S4+Z;
end process;
```






## CDFG Partitioning

x := S1 + 5;
y := S1 + S2;
t :=x * 3;
z := y + 7;
S2 <= t * z;




CDFG for process P1



## System Partitioning

- Purpose — to assign certain objects to clusters, so that a given objective function is optimized and design constraints are fulfilled.
- Given a set of $n$ objects $V=\{v_1, v_2, ..., v_n\}$, a $k$-way partitioning $P^k=\{C_1, C_2, ..., C_k\}$ consists of $k$ clusters, $C_1, C_2, ..., C_k$ so that $C_1 \cup C_2 \cup ... \cup C_k = V$, and $C_i \cap C_j = \varnothing$ for all $i,j$, $i \neq j$.
- The partitioning problem — find a partitioning $P^k$ of a set $V$ of $n$ objects, so that the cost determined by an objective function $ObjFunc(P^k)$ is minimal and a set of constraints $Cnstr(P^k)$, is satisfied.



## Metrics and Estimations

- Partitioning algorithms have to rely on a quantitative measure of a candidate solution's goodness.
- *Metrics* — attributes which characterise a given solution; they are expressed quantitatively.
- *Metrics* include cost, execution time, communication rates, power consumption, testability, reliability, program size, data size and memory size.
- Estimation determines a metric value from a *rough implementation*.
- Inaccuracy can be tolerated as long as the *relative goodness* of any two partitions is determined correctly.



## Objective Function and Closeness function

- *Objective function*: a combination of metrics which captures the *overall* quality of a certain partitioning.
- *Closeness function*: captures the benefit gained from grouping two objects into the same partition; it is based on a *local* view of the system.

## Partitioning Objective

- Partitioning quality is measured using an objective function (cost function).
- Objective function is a combination of metrics which captures the *overall* quality of a certain partitioning.

$$ObjFunc = \sum_i w_i \times M_i$$

- An example

$$ObjFunc = k_1 \cdot area + k_2 \cdot delay + k_3 \cdot power$$



## Objective Function Example

Objective function for hardware/software partitioning in VULCAN:

$$ObjFun = w_1 \cdot S_H - w_2 \cdot S_S + w_3 \cdot B - w_4 \cdot P + w_5 \cdot m$$

$S_H$: implementation cost of the hardware partition
$S_S$: implementation cost of the software partition
$B$: bus utilisation
$P$: processor utilisation
$m$: total size of variables transferred across hardware/ software boundary



## Design Constraints

- Considered separately by the partitioning algorithm,
  - need to check them during partitioning decisions,
  - rejection of infeasible solutions.
- Included into the cost function
  - can give an additional penalty to the objective function,
  - focus on a partitioning which satisfies constraints (ObjFunc=0)

*ObjFunc = $k_1$*F(area, area_constr) + $k_2$*F(delay,delay_constr) + $k_3$*F(power,power_constr)*



## Objective Function Example

- System level partitioning

$$ObjFunc = w_1 \cdot \sum_i \left(100 \cdot \frac{violate\_area(Cl_i)}{max\_area(Cl_i)}\right)^2 + w_2 \cdot \sum_i \left(100 \cdot \frac{violate\_pins(Cl_i)}{max\_pins(Cl_i)}\right)^2 + w_3 \cdot \sum_i \left(100 \cdot \frac{violate\_nrchips}{max\_nrchips}\right)^2 + w_2 \cdot \sum_i \left(100 \cdot \frac{violate\_exectime(b_i)}{max\_exectime(b_i)}\right)^2$$

$$violate\_area(Cl_i) = \begin{cases} area(Cl_i) - max\_area(Cl_i) & if \quad area(Cl_i) - max\_area(Cl_i) > 0 \\ 0 & otherwise \end{cases}$$



## Closeness Function

- Captures the *benefit* gained from grouping two objects into the same partition.
- It is based on a *local* view of the system.
- Closeness between two functions $f_i$ and $f_j$:

$$Close(f_i, f_j) = w_1 \cdot \frac{\cos t(f_i) + \cos t(f_j) - \cos t(f_i \cup f_j)}{\cos t(f_i \cup f_j)} - w_2 \cdot part(f_i, f_j)$$

$$part(f_i, f_j) = \begin{cases} 1 & if\ f_i\ and\ f_j\ can\ be\ executed\ in\ parallel \\ 0 & otherwise \end{cases}$$



## Partitioning Approaches

- Manually guided partitioning
  - Needs strong support from design environment:
    - estimation tools & schedulers,
    - facilities to interactively perform predefined transformations and to define new ones,
    - graphical interfaces.
- Automatic partitioning

## Automatic Partitioning

- The partitioning problem is *NP-complete.*
- The design space has to be explored according to a certain strategy which converges towards a solution close to one which yields the minimal cost.



## Automatic Partitioning Approaches

- Constructive (clustering)
  - bottom up approach: each object initially belongs to its own cluster, and clusters are then gradually merged until the desired partitioning is found;
  - does not require a global view of the system but relies only on local relations between objects (closeness metrics).



## Automatic Partitioning Approaches (cont’d)

- Iterative (transformation-based)
  - based on a design space exploration which is guided by an objective function that reflects the global quality of the partitioning; a starting solution is modified iteratively, by passing from one candidate solution to another based on evaluations of an objective function.



## Hierarchical clustering

- A constructive approach: performed in several iterations with final goal to group a set of objects into partitions according to some measure of closeness.
- At each iteration the two closest objects are grouped together; the process is iterated until a single cluster is produced.

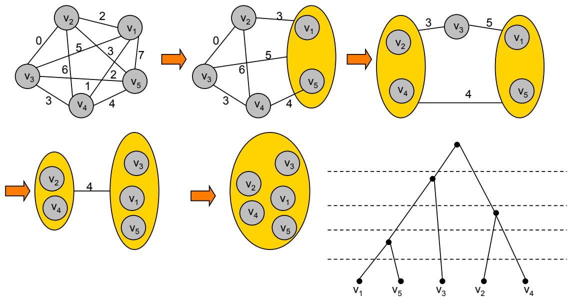


## Hierarchical cluster tree

- The cluster tree contains
  - *leafs*: original objects
  - *internal nodes*: clustered objects
  - *height*: associated to each non-terminal node; reflects the distance between the two objects that have been merged into the corresponding cluster.
- A certain partitioning is selected by cutting the cluster tree with a “cut line”; each sub-tree below the cut line becomes one resulting partition.
- The closeness function is defined between the initial objects; at successive iterations, closeness between different groups of objects have to be *estimated* based on the closeness between individual objects.



## Hierarchical Clustering - An Example

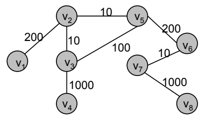


## Transformation Based Partitioning

- Transformation based approaches perform different variants of *neighbourhood search*.
- *Neighbourhood N(x)* of a solution *x* is a set of solutions that can be reached from *x* by a simple operation (*move*).
- Greedy partitioning algorithms have tendency to be trapped in local minima.
- There exist algorithms which help to escape from local minima (Kernighan-Lin, Simulated Annealing, Tabu Search, Genetic Algorithms, etc.).



## Kernighan-Lin Algorithm




$c_{56}=200$
$c_{67}=10$
:

$$ObjFunc = \sum c_{ij}, \quad for\ i,j\ so\ that\ v_i \in C_1, v_j \in C_2, e_{ij} \in E$$

**Moving $v_i$:**

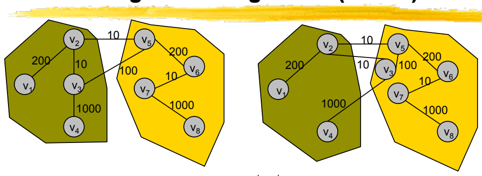
$$Ext_i = \sum_{v_j} c_{ij}, \quad where\ v_j\ belongs\ to\ a\ different\ cluster\ than\ v_i,\ and\ e_{ij} \in E$$

$$Int_i = \sum_{v_j} c_{ij}, \quad where\ v_j\ belongs\ to\ the\ same\ cluster\ as\ v_i,\ and\ e_{ij} \in E$$

$$ObjFunc(C_1, C_2) - ObjFunc(C_1', C_2') = D_i = Ext_i - Int_i$$



## Kernighan-Lin Algorithm (cont'd)




$$ObjFunc(C_1, C_2) - ObjFunc(C_1', C_2') = D_i = Ext_i - Int_i$$

**$D_3$ = 100 - 1010 = -910**

**Swap of two nodes $v_i$ and $v_j$**

$$G_{ij} = D_i + D_j, \quad if\ there\ is\ no\ connections\ between\ v_i\ and\ v_j$$
$$G_{ij} = D_i + D_j - 2 \cdot c_{ij}, \quad if\ there\ is\ an\ edge\ connecting\ v_i\ and\ v_j$$

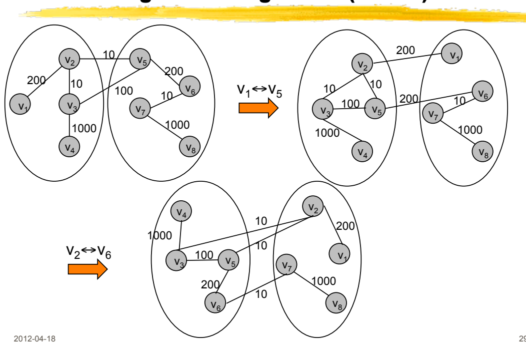


## Kernighan-Lin Algorithm (cont'd)

Construct initial configuration $x^{now} := (C_1, C_2)$, with $|C_1| = |C_2| = n$
***repeat***
**$S_0$** := 0
Unlock all nodes
**for** *k* := 1 **to** *n* **do**
Find the pair ($v_i \in C_1$, $v_j \in C_2$) so that $v_i$ and $v_j$ are unlocked and $G_{ij}$ is maximal
$S_k := S_{k-1} + G_{ij}$
$tentative_k := (v_i, v_j)$
Lock $v_i$ and $v_j$
Update *D* values for each node considering as if $v_i$ and $v_j$ are swapped
**end for**
Find *p* so that $S_p$ is maximum of all partial sums *S*
**if** $S_p > 0$ **then**
Generate new solution $\boldsymbol{x^{now}}$ starting from current solution $x^{now}$, by performing all the interchanges $tentative_l$, $1 \le l \le p$
**end if**
**until** maximal gain $S_p \le 0$
**return** $x^{now}$

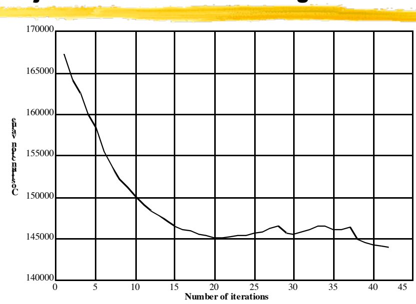


## Kernighan-Lin Algorithm (cont'd)






## Objective Function in KL Algorithm




Variation of the cost function during partitioning with KL algorithm.

## Neighbourhood Search

Construct initial configuration $x^{now}:=x_0$
**Repeat**
  Select new, acceptable solution $x' \in N(x^{now})$
  $x^{now} = x'$
**until** stopping criterion met
**return** solution corresponding to the minimum cost function

- Who is the neighborhood?
- Under which circumstances a new solution is accepted?
- What is the stopping criterion?

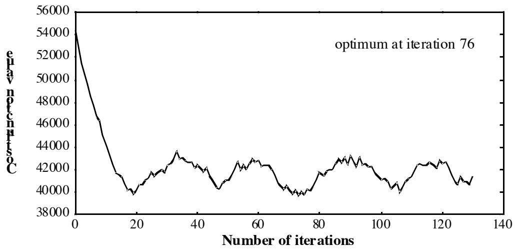


## Simulated Annealing

Select an initial solution $x^{now} \in \mathbf{X}$, an initial temperature $t_0 > 0$, and a temperature reduction function $\alpha$;
**repeat**
  **repeat**
    **randomly select** $x^{next} \in N(x^{now})$;
    $\delta := f(x^{next}) - f(x^{now})$;
    **if** $\delta < 0$ **then** $x^{now} := x^{next}$ **else**
    **begin**
      generate a random number $p$ uniformly in the range (0, 1);
      **if** $p < \exp(-\delta/t)$ **then** $x^{now} := x^{next}$;
    **end**
  **until** *iteration_count* = *nrep*;
  $t := \alpha(t)$;
**until** stopping_condition = true;
**return** $x^{now}$ as the solution.



## Simulated Annealing (cont' d)






## Hw/Sw Partitioning

- **Hardware/software partitioning** is very often treated as a particular two way partitioning in which performance has to be maximized and hardware size to be minimized;
- **Assumptions:**
  - microprocessor and ASIC working in parallel;
  - reducing the amount of communication between the microprocessor and the hardware coprocessor improves the overall performance of the system.
- **Objective**: Maximal performance at a given cost limit.



## Hw/Sw Partitioning (cont' d)

- Partitioning is based on metric values derived from profiling, static analysis of the specification, and cost estimation.
- Performance improvement based on assumption that better performance is obtained if
  - computation intensive processes are mapped into hardware,
  - parallelism is improved,
  - inter-domain communication is reduced.



## Summary

- The partitioning problem is NP-complete and has to be solved using optimization heuristics.
- Partitioning heuristics are *constructive* or *transformation-based*.
- Hierarchical clustering is one of the most used constructive approaches.
- Transformational approaches are based on neighborhood search.
- A hardware software partitioning for acceleration is done by placing computation intensive processes into hardware, improving parallelism and reducing inter-domain communication.

## Literature

- P. Eles, K. Kuchcinski and Z. Peng, *System Synthesis with VHDL*, Kluwer Academic Publisher, 1998.