System Partitioning

Kris Kuchcinski
Krzysztof.Kuchcinski@cs.lth.se

"He who can properly define and divide is to be considered a god."
Plato (ca 429-347 BC)

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

```
• 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;
```

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 $P = \{C_1, C_2, ..., C_k\}$ consists of $k$ clusters, $C_1$, $C_2$, ..., $C_k$ so that $C_i \cap C_j = \emptyset$ for all $i \neq j$.
- The partitioning problem — find a partitioning $P$ of a set $V$ of $n$ objects, so that the cost determined by an objective function $\text{ObjFunc}(P)$ is minimal and a set of constraints $\text{Cnstr}(P)$, 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.

\[ \text{ObjFunc} = \sum_{i} W_i \times M_i \]

- An example

\[ \text{ObjFunc} = k_1 \cdot \text{area} + k_2 \cdot \text{delay} + k_3 \cdot \text{power} \]

Objective Function Example

Objective function for hardware/software partitioning in VULCAN:

\[ \text{ObjFun} = w_1 \cdot S_H - w_2 \cdot S_S + w_3 \cdot B - w_4 \cdot P + w_5 \cdot m \]

- \( S_H \): 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 (\( \text{ObjFunc} = 0 \))

\[ \text{ObjFun} = \sum_{i} w_i \cdot \text{area} + \sum_{i} w_i \cdot \text{delay} + \sum_{i} w_i \cdot \text{power} \]

Objective Function Example

System level partitioning

\[ \text{ObjFunc} = w_1 \cdot \sum_{C} \text{vis_area}(C) - \sum_{C} \text{area}(C) + \sum_{C} \text{忙}(C) - \sum_{C} \text{area}(C) + \% \cdot \text{max_area} + \% \cdot \text{max_nrchip} \]

Design Constraints

- 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 \( f \) and \( f' \):

\[ \text{Close}(f, f') = \frac{\text{cost}(f) \cdot \text{cost}(f') - \text{cost}(f \cup f')}{\text{cost}(f) \cdot \text{cost}(f')} - w_1 \cdot \text{part}(f, f') \]

\[ \text{part}(f, f') = \begin{cases} 1 & \text{if } f \text{ and } f' \text{ can be executed in parallel} \\ 0 & \text{otherwise} \end{cases} \]

Closeness Function

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

Partitioning Approaches
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:
  - Leaf: original objects
  - Internal node: 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.
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

\[ \text{ObjFunc} = \sum_{i,j} c_{ij} \quad \text{for } i,j \text{ such that } v_i \in C_i, v_j \in C_j, v_i, v_j \in E \]

Moving \( v_i \):
- \( \text{Int} = \sum_{v_j \in \text{Int}} \), where \( v_j \) belongs to a different cluster than \( v_i \), and \( v_i, v_j \in E \)
- \( \text{Ext} = \sum_{v_j \in \text{Ext}} \), where \( v_j \) belongs to the same cluster as \( v_i \), and \( v_i, v_j \in E \)

\[ \text{ObjFunc}^{'}(C_i,C_j) = \text{ObjFunc}(C_i,C_j) - \text{Int}_i + \text{Int}_j \]

Swap of two nodes \( v_i \) and \( v_j \):
- \( S_j = S_j + D_{ij} \) if there is no connection between \( v_i \) and \( v_j \)
- \( S_i = S_i + D_{ij} - 2c_{ij} \) if there is an edge connecting \( v_i \) and \( v_j \)

Construct initial configuration \( x^{\text{init}} = (C_i, C_j) \), with |\( C_i \) - \( C_j \)| = \( n \)

repeat
- \( S_j = 0 \)
- Unlock all nodes for \( k = 1 \) to \( n \) do
  - Find the pair \((v_i \in C_i, v_j \in C_j)\) so that \( v_i \) and \( v_j \) are unlocked and \( G_j \) is maximal
  - \( S_j = S_j + c_{ij} \)
  - \( \text{tentative} = (v_i, v_j) \)
  - 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 \( j \) so that \( S_j \) is maximum of all partial sums \( S_k \)
- If \( S_j > 0 \) then
  - Generate new solution \( x^{\text{new}} \) starting from current solution \( x^{\text{init}} \), by performing all the interchanges \( \text{tentative} \), 1xswap
end if
- until maximal gain \( S_j = 0 \)
- return \( x^{\text{new}} \)

Objective Function in KL Algorithm

Variation of the cost function for 40 problem partitions in the KL algorithm.
Neighbourhood Search

Construct initial configuration \( x^{(0)} := x_0 \)

```
Repeat
  Select new, acceptable solution \( x' \in N(x^{(0)}) \)
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 \( x^{(0)} \in X \), 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 \( p < e^{-\delta/T} \) then \( x^{(now)} := x^{(next)} \);
  end
```

until iteration_count = nrep;

```
t := \alpha(t);
```

until stopping_condition = true;

return \( x^{(now)} \) as the solution.

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