System Partitioning

SS 2009
Hw/Sw Codesign

Christian Plessl
Overview

- Graph models for system synthesis
- The partitioning problem
- Partitioning methods
- Design space exploration
Models for System Synthesis

• Allocation + Binding = Partitioning

• Problem graph
  – nodes: functional and communication objects
  – edges: dependencies

• Architecture graph
  – nodes: functional and communication resources
  – edges: directed communication paths

• Specification graph
  – problem graph + architecture graph + possible mappings
Problem Graph

DFG

1
2
3
4

problem graph

1
2
5
3
6
7
4

communication nodes
Architecture Graph

architecture

architecture graph

RISC

HWM1

HWM2

bus

point-to-point link

\[ V_{RISC} \]

\[ V_{bus} \]

\[ V_{HWM1} \]

\[ V_{ptp} \]

\[ V_{HWM2} \]
Specification Graph
Allocation, Binding

1 -> V_{RISC}
2 -> V_{bus}
5 -> V_{HWM1}
6 -> V_{HWM2}
3 -> V_{ptp}
7 -> V_{ptp}
Example: Homogeneous Multiprocessor

- Optimization goals
  - minimize latency
  - meet deadlines
  - ...

![Homogeneous Multiprocessor Diagram]

M
PE1
M
PE2
M
PE3

bus
Example: Hw/Sw Bi-Partitioning

- Only two blocks: SW and HW (accelerator)
**Synthesis and Constraints**

- **Modeling**
  1. model problem
  2. define architectural template
  3. identify possible bindings
     - refinements: communication, memory, ..

- **System synthesis with constraints** \( C_{max}, L_{max} \)
  1. allocation - gives cost \( C \) (first approximation)
  2. binding
  3. scheduling - gives latency \( L \)

  - feasible schedule: \( L \leq L_{max} \)
  - feasible binding: leads to at least one feasible schedule
  - feasible allocation: \( C \leq C_{max} \) and leads to at least one feasible binding
Overview

- Graph models for system synthesis
- The partitioning problem
- Partitioning methods
- Design space exploration
Partitioning Problem

• **Definition:** The partitioning problem is to assign \( n \) objects \( O = \{ o_1, ..., o_n \} \) to \( m \) blocks (partitions) \( P = \{ p_1, ..., p_m \} \), such that
  
  - \( p_1 \cup p_2 \cup ... \cup p_m = O \)
  - \( p_i \cap p_j = \{ \} \) \( \forall \) \( i, j: i \neq j \) and
  - the cost \( c(P) \) is minimized.

  the general partitioning problem is NP-complete

• **In system synthesis:**
  - objects = problem graph nodes
  - blocks = architecture graph nodes
Partitioning – Abstraction Levels

• **Structural** partitioning
  – on the RTL- or netlist level
    ▪ eg: map a digital circuit onto two chips (FPGAs, ASICs)
    ▪ system parameters are relatively well known (area and delay of functional units, registers, gates)
    ▪ no comparison of design alternatives

• **Functional** partitioning
  – on the system level
    ▪ comparison of design alternatives → design space exploration
    ▪ system parameters are not known → estimation required
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: linear cost function with penalty

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

  - \( h_C, h_L, h_P \) … denote how strong \( C, L, P \) violate the design constraints \( C_{\text{max}}, L_{\text{max}}, P_{\text{max}} \)
  - \( k_1, k_2, k_3 \) … weighting and normalization
Overview

- Graph models for system synthesis
- The partitioning problem
- Partitioning methods
- Design space exploration
 Partitioning Methods - Overview

• Exact methods
  – enumeration of solutions
  – Integer Linear Programs (ILP)

• Heuristic methods
  – constructive methods
    ▪ random mapping
    ▪ hierarchical clustering
  – iterative methods (refinement methods)
    ▪ greedy partitioners
    ▪ Kernighan-Lin
    ▪ simulated annealing
    ▪ evolutionary algorithms (design space exploration)
**Integer Linear Programs (1)**

- Binary variables $x_{i,k} = 1$: object $o_i$ in block $p_k$
- Cost $c_{i,k}$, if object $o_i$ is in block $p_k$
- Integer linear program:

\[
x_{i,k} \in \{0,1\} \quad 1 \leq i \leq n, 1 \leq k \leq m
\]
\[
\sum_{k=1}^{m} x_{i,k} = 1 \quad 1 \leq i \leq n
\]
\[
\text{minimize} \quad \sum_{k=1}^{m} \sum_{i=1}^{n} x_{i,k} \cdot c_{i,k} \quad 1 \leq k \leq m, 1 \leq i \leq n
\]
Integer Linear Programs (2)

- Constraints are modeled by inequations
  eg.: maximal number of $h_k$ objects in block $k$

\[ \sum_{i=1}^{n} x_{i,k} \leq h_k \quad 1 \leq k \leq m \]

- ILP is NP-complete
  - in the worst-case exponential runtime
  - solved by branch&bound algorithms
  - modeling difficult when constraints are non-linear
Constructive Methods

• Examples
  – random mapping
    ▪ each object is randomly assigned to some block
  – hierarchical clustering
    ▪ stepwise grouping of objects
    ▪ closeness function determines how desirable the grouping of two objects is

• Constructive methods …
  – are often used to generate a valid start partition for iterative (refinement) methods
  – clustering often shows the difficulty of finding suitable closeness function
Hierarchical Clustering (1)

The closeness between hierarchical objects: arithmetic mean.

Mathematically, this can be expressed as $v_5 = v_1 \cup v_3$.
Hierarchical Clustering (2)

\[ v_6 = v_2 \cup v_5 \]
Hierarchical Clustering (3)

\[ v_7 = v_6 \cup v_4 \]
Hierarchical Clustering (4)

step 1:
\[ v_5 = v_1 \cup v_3 \]

step 2:
\[ v_6 = v_2 \cup v_5 \]

step 3:
\[ v_7 = v_6 \cup v_4 \]

cut lines (partitions)
Greedy Hw/Sw Partitioning (1)

- Bi-partitioning (simplest case): \( P = \{ p_{SW}, p_{HW} \} \)

- Software oriented approach: \( P = \{ O, \emptyset \} \)
  - in sw all functions can be realized
  - performance might be too low ➜ migrate objects to hw

- Hardware oriented approach: \( P = \{ \emptyset, O \} \)
  - in hw the performance is ok (assumes hw is always faster than sw)
  - cost might be too high ➜ migrate objects to sw
Greedy Hw/Sw Partitioning (2)

- Migration of objects into the other block (HW/SW), until there is no more improvement

```plaintext
REPEAT {
    P_old = P;
    FOR i = 1 TO n {
        IF (f(Move(P, o_i)) < f(P)) {
            P = Move(P, o_i);
        }
    }
    UNTIL (P == P_old)
```

cost function $f()$
Kernighan-Lin (1)

- Generation of bi-partitions
  - regroup the object which gives the biggest gain in cost under constraints on the balance of partition sizes
    (cmp. minimum cut set and maximum cut set)
**Kernighan-Lin (2)**

- **Extensions**
  - regroup the object which gives the **biggest cost gain or the smallest cost loss**
    - as long as there is a better partition:
      - from the \( n \) objects, tentatively regroup the “best” one, then from the remaining \( n-1 \) objects again the “best”, and so on until all objects have been regrouped
      - from these \( n \) partitions select the one with the smallest cost and actually perform the regroup operations
    - can escape from a local minima
  - also for partitioning into \( m \) blocks
Simulated Annealing (1)

- Simulated annealing
  - 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
  - probability for a particle jumping into a higher energy state:

\[
P(e_i, e_j, T) = e^{\frac{e_i - e_j}{k_b T}}
\]

- Application to combinatorial optimization
  - energy = cost of a solution
  - reduction of cost with simulated temperature, sometimes increases in cost are accepted
Simulated Annealing (2)

temp = temp_start;
cost = c(P);
WHILE (Frozen() == FALSE) {
    WHILE (Equilibrium() == FALSE) {
        P' = RandomMove(P);
cost' = c(P');
deltacost = cost' - cost;
        IF (Accept(deltacost, temp) > random[0,1)) {
            P = P'
cost = cost'
        }
    }
temp = DecreaseTemp(temp);
}
Simulated Annealing (3)

• **Annealing schedule**: `DecreaseTemp()`, `Frozen()`
  - `temp_start = 1.0`
  - `temp = \alpha \cdot temp` (typical: `0.8 \leq \alpha \leq 0.99`)
  - stop at `temp < temp_min`
    or if there is no more improvement

• **Equilibrium**: `Equilibrium()`
  - after certain number of iterations or if there is no more improvement

• **Complexity**
  - from exponential to constant, depending on the implementation of the functions `Equilibrium()`, `DecreaseTemp()`, `Frozen()`
  - the longer the runtime, the better the results
  - usually functions are constructed to get polynomial runtime
Case Study: YSC

• Yorktown Silicon Compiler: 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 datapaths (ALUs, registers)
  – **method**: hierarchical clustering, closeness function:

\[
Closeness(p_i, p_j) = \left( \frac{\text{sharedwires}(p_i, p_j)}{\text{maxwires}(P)} \right)^{c_2} \cdot \left( \frac{\text{maxsize}}{\min\{\text{size}(p_i), \text{size}(p_j)\}} \right)^{c_3} \cdot \left( \frac{\text{maxsize}}{\text{size}(p_i) + \text{size}(p_j)} \right)
\]
Case Study: Vulcan (Hw/Sw Bi-Part.)

- **Input**: program in HardwareC
  - C extended by a process concept and interprocess communication
  - specification with constraints (min/max-times and data rates)

- **Target architecture**: 1 processor / 1 ASIC
  - one global bus and one global memory
  - the processor is always the bus master

- **Abstraction level**: basic blocks and operations
  - deterministic execution times
  - internal/external non-deterministic execution times

- **Method**: HW-oriented greedy
  - cost function includes hw cost, memory requirement, performance and synchronization effort
Case Study: Cosyma (Hw/Sw Bi-Part.)

- **Input:** Programm in C^x
  - C extended by a process concept and interprocess communication
  - specification of min/max-times

- **Target architecture:** processor + coprocessor
  - coupled by a shared memory
  - computations on the processor and on the coprocessor may not overlap in time

- **Abstraction level:** basic blocks

- **Method:** SW-oriented, 2 loops:
  - inner loop: simulated annealing with cost function that gives the time gain for a HW realization of a block
  - outer loop: synthesis to improve the estimations for the inner loop
Overview

- Graph models for system synthesis
- The partitioning problem
- Partitioning methods
- Design space exploration
Basic Principles - Evolution

1 selection

2 crossover

3 mutation
Evolutionary Algorithms (EA)

minimize $g(x) = x^2$

- Fitness calculation
- Selection
- Crossover
- Mutation
- Next generation

- $0011$ = one solution
- Fitness $= 9$

- $0011$

- $0100$

- $0000$

- $1011$

- $0011$
**Evolutionary Algorithms (EA)**

- **Evolutionary algorithms are randomized search heuristics**
  - problem-independent (meta heuristics)
  - population based
  - use variation (crossover, mutation) and selection

- **Application domains**
  - when optimization problem is complex and 'diffuse'
    - examples: system synthesis, path planning in robotics
  - multiobjective optimization
    - several conflicting criteria, eg. performance vs. cost vs. power consumption
    - EAs find Pareto fronts (set of Pareto points)
Dominance, Pareto Points

• **Definition:** A (design) point $J_i$ is dominated by point $J_k$, if $J_k$ is equal or better than $J_i$ in all criteria and better in at least one criterion.

\[ J_i < J_k \]

• **Definition:** A (design) point is Pareto-optimal or a Pareto point, if it is not dominated by any other point.
Multiobjective Optimization (1)

Decision space

\((x_1, x_2, \ldots, x_n)\)

Objective space

\((y_1, y_2, \ldots, y_k)\)

Minimize \(f\)

Difficulties:

1. Large search space
2. Multiple optima

Pareto optimal

Not dominated

dominated
Multiobjective Optimization (2)

- Classic single-objective methods
  - eg. simulated annealing, ILP, hierarchical clustering, ...
  - decision making before optimization
    - weighted cost function
    - multi-stage optimization
      - eg. hierarchical clustering with different closeness functions
  - decision making after optimization
    - multiple optimization runs with varying weights

- Population-based methods
  - evolutionary algorithms
  - decision making after optimization
    - the goal is to explore the design space
Weighted Cost Function

Multiple objectives
(y₁, y₂, ..., yₖ)

Transformation

Parameters

Single objective

y

Example: weighting approach

y = w₁y₁ + … + wₖyₖ

Maximization problem
EAs for Multiobjective Optimization

EA operations
1. selection
2. recombination
3. mutation

diversity

goals

distance
Selection by Pareto Ranking

- Fitness function:

\[ F'(J) = \sum_{i=1..N, \text{\scriptsize } J \neq J_i} \left\{ \begin{array}{ll} 1: & J_i \prec J \\ 0: & \text{else} \end{array} \right. \]

 execution time

\[ \begin{align*}
F'(1) &= 3 \\
F'(2) &= 1 \\
F'(3) &= 1 \\
F'(4) &= 2 \\
F'(5) &= 1 \\
F'(6) &= 0
\end{align*} \]
Example: Strength Pareto EA (1)

1. Save nondominated solutions (elitism)
2. Reduce nondominated set by means of clustering
3. Assign fitness values (Pareto-based)
4. Perform binary tournament selection
5. Recombination
   Mutation
Example: Strength Pareto EA (2)

Clustering: reduce nondominated set but do not destroy characteristics

1. group
2. select
3. remove
Example: Strength Pareto EA (3)

fitness assignment scheme:

1. nondominated solutions:
   fitness = #dominated solutions

2. dominated solutions:
   fitness =
   \[ \text{#non-Pareto solutions} + \sum \text{fitness of dominators} \]

- “lighter better than darker” → guidance towards Pareto-optimal set
- “few better than many” → maintenance of diversity
Design Space Exploration with EAs

EA

1. selection
2. recombination
3. mutation

“chromosome” = encoded allocation + binding

individual

allocation

bindi

decode allocation

decode binding

scheduling

design point

(implementation)

fitness evaluation

fitness

user constraints
Challenges

• Encoding of (allocation+binding)
  – simple encoding
    ▪ eg. one bit per resource, one variable per binding
    ▪ easy to implement
    ▪ many infeasible partitionings
  – encoding + repair
    ▪ eg. simple encoding and modify such that for each \( v_p \in V_p \) there exists at least one \( v_a \in V_A \) with \( \beta(v_p) = v_a \)
    ▪ reduces number of infeasible partitionings

• Generation of the initial population, mutation

• Recombination
Case Study - Video Coder (1)

behavioral specification of a video codec for video compression
Case Study - Video Coder (2)

problem graph of the video coder
Case Study - Video Coder (3)

h261 architecture template

frame memory
dual ported frame memory
block matching module
input module
subtract/add module
DCT/IDCT module
Huffman encoder

interconnected modules:
- RISC1
- FM
- DPFL
- BMM
- DSP
- INM
- RISC2
- SAM
- DCTM
- HC
- OUTM

connections:
- RISC1 → FM
- FM → DPFL
- DPFL → BMM
- BMM → DSP
- DSP → INM
- INM → RISC2
- RISC2 → SAM
- SAM → DCTM
- DCTM → HC
- HC → OUTM

modules and numbers:
- RISC1: 3 150
- FM: 5
- DPFL: 6 40
- BMM: 7
- DSP: 8 200
- INM: 1 0
- RISC2: 4
- SAM: 9 50
- DCTM: 10 100
- HC: 11 50
- OUTM: 2 0

SBM: 13 20
SBF: 14 30
EA Design Space Exploration Tool

[Diagram showing synthesis configuration panel and Gantt chart with resource utilization and cost analysis.]
Case Study - Solution 1

[Diagram showing resource allocation over time with labels for INM, OUTM, FM, RISC2, and SBS, along with a Gantt chart indicating task durations and dependencies.]
Case Study - Solution 2

INM

OUTM

DPFM

HC

DCTM

BMM

SAM

SBF
Case Study - Code Synthesis (1)

synchronous data flow graph

1 1 2 3 2 7 8 7 5 1
A → B → C → D → E → F
CD → DAT

software implementation

decisions

schedule
ABABABCCABABA...

inlining

ICODE (A)
CODE (B)
CODE (A)
CODE (B)
CODE (C)

subroutines

CALL (A)
CALL (B)
CALL (A)
CALL (B)
CALL (C)

looping

FOR 1 TO 2
CODE (A)
CALL (B)
CODE (C)
CODE (A)
Case Study - Code Synthesis (2)

Trade-offs

Data memory

Program memory

Execution time

Looping, subroutines

May increase

Schedule

Saves

Looping

Save

Increase
Case Study - Code Synthesis (3)

Trade-off surface for TI TMS320C40
Case Study - Code Synthesis (4)

TI TMS320C40

Motorola DSP56k

ADSP 2106x