ECE382N.23: Embedded System Design and Modeling

Lecture 9 – System-Level Synthesis

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Lecture 9: Outline

- **Exact methods**
  - Integer linear programming (ILP)
- **Constructive heuristics**
  - Random mapping
  - List schedulers
- **Generic iterative DSE heuristics**
  - Brute-force or random search
  - Simulated annealing
  - Evolutionary algorithms

Integer Linear Programming

- **Linear expressions over integer variables**
  - Cost function $C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in R, x_i \in N \quad (1)$
  - Constraints $\forall j \in J : \sum_{x_i \in X} b_{i,j}\, x_i \geq c_j \text{ with } b_{i,j}, c_j \in R \quad (2)$

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

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

Source: L. Thiele

Integer Linear Program for Partitioning (1)

- **Inputs**
  - Tasks t_i, $1 \leq i \leq n$
  - Processors p_k, $1 \leq k \leq m$
  - Cost $c_{i,k}$, if task t_i is in processor p_k
- **Binary variables $x_{i,k}$**
  - $x_{i,k} = 1$: task t_i in block p_k
  - $x_{i,k} = 0$: task t_i not in block p_k
- **Integer linear program:**

$$x_{i,k} \in \{0,1\} \quad 1 \leq i \leq n, 1 \leq k \leq m$$

$$\sum_{k=1}^{m} x_{i,k} = 1 \quad 1 \leq i \leq n$$

$$\text{minimize} \sum_{k=1}^{m} \sum_{i=1}^{n} x_{i,k} \cdot c_{i,k} \quad 1 \leq k \leq m, 1 \leq i \leq n$$

Source: L. Thiele

Integer Linear Program for Partitioning (2)

- **Additional constraints**
  - example: maximum number of h_k objects in block k

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

- **Popular approach**
  - Various additional constraints can be added
  - If not solving to optimality, run times are acceptable and a solution with a guaranteed quality can be determined
  - Can provide reference to provide optimality bounds of heuristic approaches
  - Finding the right equations to model the constraints is an art… (but good starting point to understand a problem)
  - ➢ Static scheduling can be integrated (SDFs)

Source: L. Thiele

Integer Linear Program for Scheduling

- **Inputs**
  - Task graph TG: tasks t_i, $1 \le i \le n$ with edges (t_i, t_j)
  - Discrete time window: $0 \le t < T_{max}$
- **Decision variables**
  - $s_{i,t} \in \{0,1\}$: task t_i executes at time t
- **Constraints**
  - Single task execution: $\sum_t s_{i,t} = 1, \quad 1 \le i \le n$
  - Sequential task execution: $\sum_i s_{i,t} \le 1, \quad 0 \le t < T$
  - Task dependencies $t_i \rightarrow t_j$: $\sum_t t \cdot s_{j,t} \ge \underbrace{\sum_t t \cdot s_{i,t}}_{\text{Start time of task } t_i} + 1$
- **Objective**
  - Minimize latency (task t_n is sink): $\text{minimize} \sum_t t \cdot s_{n,t}$

Integer Linear Program for Scheduling (2)

- **Inputs**
  - Task graph TG: tasks t_i, $1 \le i \le n$ with edges (t_i, t_j)
  - Execution time e_i of task t_i, $1 \le i \le n$
  - Discrete time window: $0 \le t < T_{max}$
- **Decision variables**
  - $s_{i,t} \in \{0,1\}$: task t_i starts execution at time t
- **Constraints**
  - Single task execution: $\sum_t s_{i,t} = 1, \quad 1 \le i \le n$
  - Sequential task execution: $\sum_i \underbrace{\sum_{\tau=t-e_i+1}^{t} s_{i,\tau}} \le 1, \quad 0 \le t < T$

    Is task t_i executing at time t ? ⇨ Did it start in t, t-1, … ?
  - Task dependencies $t_i \rightarrow t_j$: $\sum_t t \cdot s_{j,t} \ge \sum_t t \cdot s_{i,t} + e_i$
- **Objective**
  - Minimize latency (task t_n is sink): $\text{minimize } \sum_t t \cdot s_{n,t} + e_n$

ILP for Partitioning & Scheduling (1)

- **Inputs**
  - Tasks t_i, $1 \le i \le n$, edges (t_i, t_j), time window: $0 \le t < T_{max}$
  - Processors p_k, $1 \le k \le m$, cost $c_{i,k}$ if task t_i in processor p_k
  - Execution time $e_{i,k}$ of task t_i on processor p_k
- **Decision variables**
  - $x_{i,k} \in \{0,1\}$: task t_i mapped to processor p_k
  - $s_{i,t} \in \{0,1\}$: task t_i starts execution at time t
- **Constraints**
  - Unique task mapping: $\sum_k x_{i,k} = 1, \quad 1 \le k \le m$
  - Single task execution: $\sum_t s_{i,t} = 1, \quad 1 \le i \le n$
  - Sequential task execution on each processor:

    $$\sum_i \sum_{\tau=t-e_i+1}^{t} x_{i,k} \cdot s_{i,\tau} \le 1, \quad 0 \le t < T, \; 1 \le k \le m$$
  - Task dependencies $t_i \rightarrow t_j$: $\sum_t t \cdot s_{j,t} \ge \sum_t t \cdot s_{i,t} + \sum_k x_{i,k} \cdot e_{i,k}$

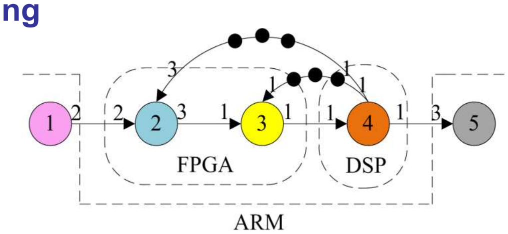
    Non-linear!
- **Objective**
  - Weighted cost & latency: $\min \; w_1 \sum_k \sum_i x_{i,k} \cdot c_{i,k} + w_2 (\sum_t t \cdot s_{n,t} + \sum_k x_{n,k} \cdot e_n)$

ILP for Partitioning & Scheduling (2)

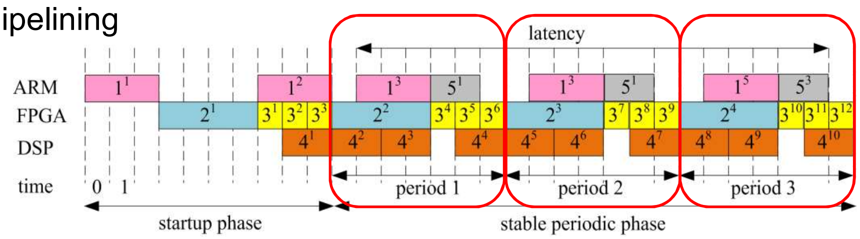
- **Inputs**
  - Tasks t_i, $1 \le i \le n$, edges (t_i, t_j), time window: $0 \le t < T_{max}$
  - Processors p_k, $1 \le k \le m$, cost $c_{i,k}$ if task t_i in processor p_k
  - Execution time $e_{i,k}$ of task t_i on processor p_k
- **Decision variables**
  - $s_{i,k,t} \in \{0,1\}$: task t_i starts at time t on processor p_k
- **Constraints**
  - Single & unique task mapping: $\sum_k \sum_t s_{i,k,t} = 1, \quad 1 \le i \le n$
  - Sequential, non-overlapping execution on each processor:
    $$\sum_i \sum_{\tau=t-e_{i,k}+1}^{t} s_{i,k,\tau} \le 1, \quad 0 \le t < T, \; 1 \le k \le m$$
  - Task dependencies $t_i \rightarrow t_j$:
    $$\sum_k \sum_t t \cdot s_{j,k,t} \ge \sum_k \sum_t t \cdot s_{i,k,t} + \sum_k \sum_t s_{i,k,t} \cdot e_{i,k}$$
- **Objective**
  - Weighted cost & latency:
    $$\text{minimize } w_1\left(\sum_k \sum_i \sum_t c_{i,k} \cdot s_{i,k,t}\right) + w_2\left(\sum_k \sum_t t \cdot s_{n,k,t} + \sum_k \sum_t s_{n,k,t} \cdot e_{n,k}\right)$$

Pipelined Scheduling

- **Allocation and partitioning**
  - Resource sharing



- **Static scheduling**
  - Pipelining



Throughput = 1 / Period

Latency = (End of the *n*-th exec. of sink) – (Start of the *n*-th exec. of source)

Pipelined Scheduling ILP

- **Multi-objective cost function**
  - Minimize: $w_1 \cdot Throughput + w_2 \cdot Latency + w_3 \cdot Cost$
- **Decision variables**
  - Actor to processor binding for time window (period)
  - Actor start times within time window (period)
- **Constraints**
  - Execution precedence according to SDF semantics
  - Single & unique actor mapping
  - Sequential execution on each processor
  - Stable periodic phase

➢ **Optimize partition and schedule simultaneously**
➢ **Incorporate communication mapping**

J. Lin, A. Srivasta, A. Gerstlauer, B. Evans, "Heterogeneous Multiprocessor Mapping for Real-time Streaming Systems," ICASSP'11
J. Lin, A. Gerstlauer, B. Evans, "Communication-Aware Heterogeneous Multiprocessor Mapping for Real-time Streaming Systems," JSPS'12

Lecture 9: Outline

✓ **Exact methods**
  ✓ Integer linear programming (ILP)

- **Constructive heuristics**
  - Random mapping
  - List schedulers
- **Generic iterative DSE heuristics**
  - Brute-force or random search
  - Simulated annealing
  - Evolutionary algorithms

Constructive Methods – List Scheduling

- **Greedy heuristic**
  - Process graph in topology order (source to sink)
  - Process ready nodes in order of priority (criticality)
  - ➢ List scheduling variants only differ in priority function
    - – Highest level first (HLF), i.e. distance to the sink
    - – Critical path, i.e. longest path to the sink
- **Widely used scheduling heuristic**
  - Operation scheduling in compilation & high-level synthesis
    - Hu's algorithm for uniform delay/resources (HLF, optimal)
    - Iterative modulo scheduling for software pipelining
  - Job-shop/multi-processor scheduling
    - Graham's algorithm (optimal online algorithm for ≤ 3 processors)
    - Heterogeneous earliest-finish time first (HEFT)
  - ➢ Natural fit for minimizing makespan/latency
    - ➢ $O(n)$ complexity

Constructive Methods – List Scheduling

```
l = 0;
i = 0…n: p_i ← Idle;
Ready ← Initial tasks (no dependencies);
while (!empty(Ready)) {
   forall p_i: status(p_i) == Idle {
      t = first(Ready, p_i);   // by priority
      p_i ← (t, l, l + exec_time(t));
   }
   l = min(l + 1, finish_time(p_i));
   forall p_i: finish_time(p_i) == l {
      Ready ← successors(current(p_i));
      p_i ← Idle;
   }
}
```

Lecture 9: Outline

✓ **Exact methods**
- ✓ Integer linear programming (ILP)

✓ **Constructive heuristics**
- ✓ Random mapping
- ✓ List schedulers

- **Generic iterative DSE heuristics**
  - Brute-force or random search
  - Simulated annealing
  - Evolutionary algorithms

Iterative Methods

- **Basic principle**
  - Start with some initial configuration (e.g. random)
  - Repeatedly search *neighborhood* (similar configuration)
    - Select *neighbor* as candidate (make a *move*)
  - Evaluate *fitness* (cost function) of candidate
    - Accept candidate under some rule, select another neighbor
  - Stop if quality is sufficient, no improvement, or end time
- **Ingredients**
  - Way to create an initial configuration
  - Function to find a *neighbor* as next candidate (make *move*)
  - *Cost* function
    - Analytical or simulation
  - *Acceptance* rule, stop criterion
  - ➢ No other insight into problem needed

Source: L. Thiele

Iterative Improvement

- **Greedy "hill climbing" approach**
  - Always and only accept if cost is lower (fitness is higher)
  - Stop when no more neighbor (move) with lower cost
- **Disadvantages**
  - Can get trapped in local optimum as best result
    - Highly dependent on initial configuration
  - Generally no upper bound on iteration length
- ➢ **How to cope with disadvantages?**
  - Repeat with many different initial configurations
  - Retain information gathered in previous runs
  - Use a more complex strategy to avoid local optima
  - ➢ Random moves & accept cost increase with probability > 0

Source: L. Thiele

Iterative Methods - Simulated Annealing

- **From Physics**
  - Metal and gas take on a minimal-energy state during cooling down (under certain constraints)
    - At each temperature, the system reaches a thermodynamic equilibrium
    - Temperature is decreased (sufficiently) slowly
  - Probability that a particle "jumps" to a higher-energy state:

$$P(e_i, e_{i+1}, T) = e^{\frac{e_i - e_{i+1}}{k_B T}}$$

- **Application to combinatorial optimization**
  - Energy = cost of a solution (cost function)
    - Can use simulation or any other evaluation/estimation model
  - Iteratively decrease temperature
    - In each temperature step, perform random moves until equilibrium
    - Increases in cost are accepted with certain probability (depending on cost difference and "temperature")

Source: L. Thiele

Iterative Methods - Simulated Annealing

```
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);
}
```

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

Source: L. Thiele

Iterative Methods - Simulated Annealing

- **Random moves: `RandomMove(P)`**
  - Choose a random solution in the neighborhood of P
- **Cooling Down: `DecreaseTemp()`, `Frozen()`**
  - Initialize: temp_start = 1.0
  - DecreaseTemp: temp = α • temp (typical: $0.8 \leq \alpha \leq 0.99$)
  - Terminate (frozen): temp < temp_min or no improvement
- **Equilibrium: `Equilibrium()`**
  - After defined number of iterations or when there is no more improvement
- **Complexity**
  - From exponential to constant, depending on the implementation of the cooling down/equilibrium functions
  - The longer the runtime, the better the quality of results

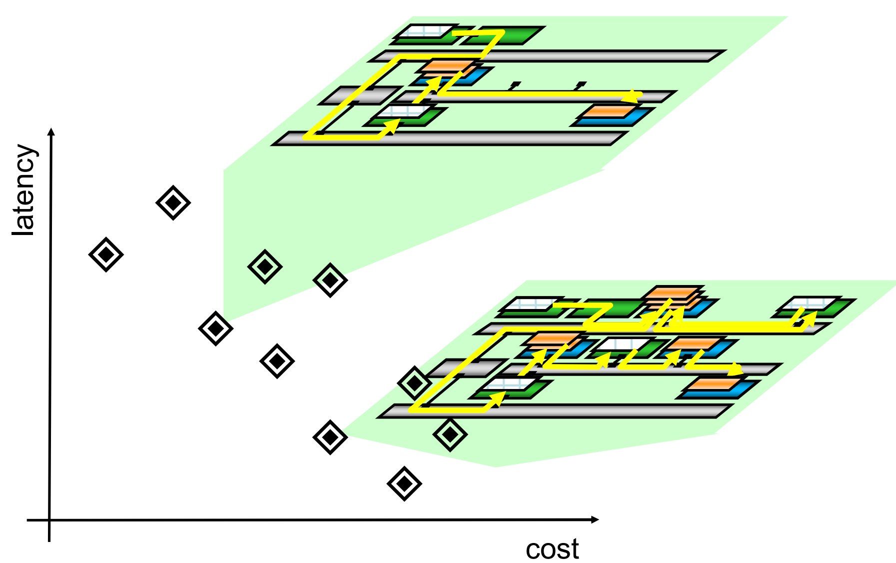
Source: L. Thiele

Multi-Objective Exploration

- **Multi-objective optimization (MOO)**
  - Implementations are optimized with respect to many (conflicting) objectives
  - Several optimal solutions exist with different tradeoffs among properties
- **Exact, constructive methods are prohibitive**
  - Large design space, dynamic behavior
- **Iterative single-objective methods**
  - Only return a single solution
- ➢ **Set-based iterative approaches**
  - ➢ Randomized, problem independent (black box)
  - ➢ Often inspired by processes in nature (evolution, ant colonies, diffusion)

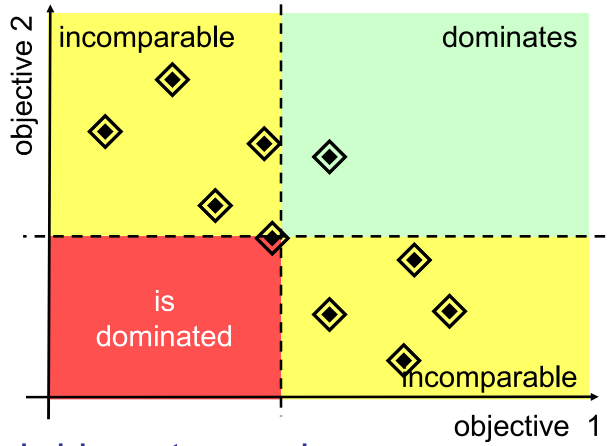
Source: C. Haubelt, J. Teich

Objective Space



Source: C. Haubelt, J. Teich

Pareto Dominance

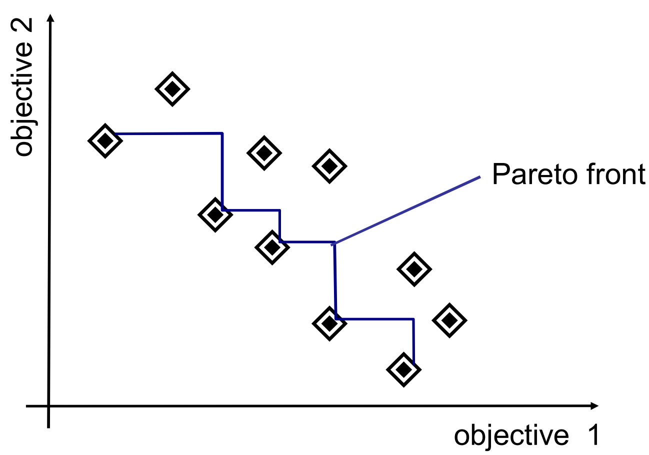


- **Given: two decision vectors x_1 and x_2**
  - $x_1 \succ\succ x_2$ (strongly dominates) if $\forall i: f_i(x_1) < f_i(x_2)$
  - $x_1 \succ x_2$ (dominates) if $\forall i: f_i(x_1) \leq f_i(x_2) \wedge \exists j: f_j(x_1) < f_j(x_2)$
  - $x_1 \sim x_2$ (indifferent) if $\forall i: f_i(x_1) = f_i(x_2)$
  - $x_1 \| x_2$ (incomparable) if $\exists i,j: f_i(x_1) < f_i(x_2) \wedge f_j(x_2) < f_j(x_1)$

Source: C. Haubelt, J. Teich

Pareto Optimality

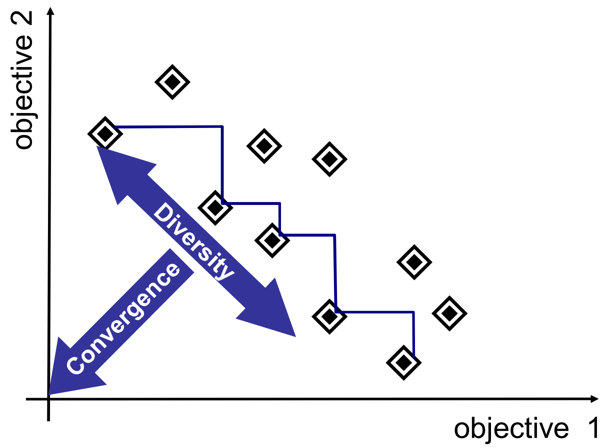
- **Set of all solutions *X***
- **A decision vector $x \in X$ is said to be *Pareto-optimal* if $\nexists y \in X: y \succ x$**



Source: C. Haubelt, J. Teich

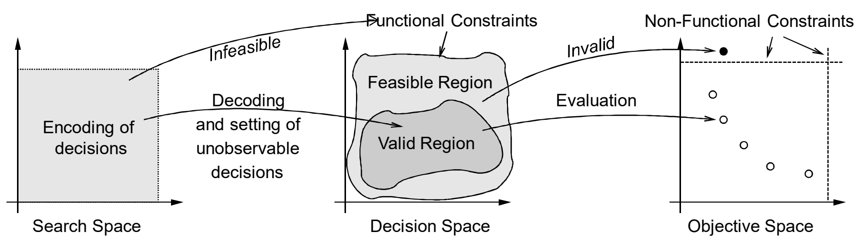
Optimization Goals

- **Find Pareto-optimal solutions (Pareto front)**
- **Or a good approximation (convergence, diversity)**
- **With a minimal number of iterations**



Source: C. Haubelt, J. Teich

Design Space Exploration (DSE)



- **Search space vs. decision space vs. design space**
  - Encoding of decisions defines search space
    - Focus on observable decisions, hardcode unobservable ones
  - Functional & architecture constraints define decision space
    - Quickly prune & reject infeasible decisions
  - Quality constraints restrict objective space
    - Invalid solutions outside of valid quality range

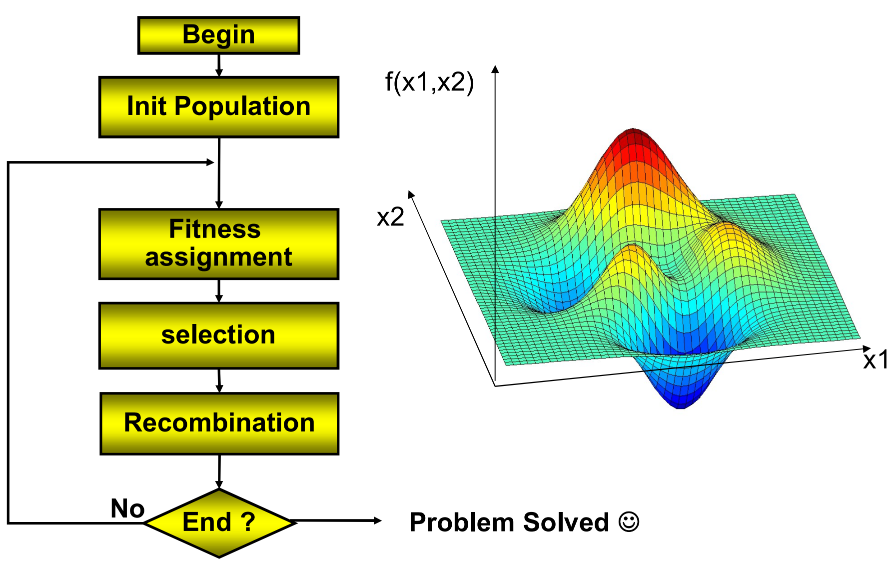
Evolutionary Algorithms (EAs)

- **Multi-objective evolutionary algorithms (MOEAs)**
  - Capable to explore the search space very fast, i.e., they can find some good solutions after a few iterations (generations)
  - Explore high dimensional search spaces
  - Can solve variety of problems (discrete, continuous, …)
  - Work on a population of individuals in parallel
  - Black box optimization (generic evaluation model)
- **Fitness evaluation**
  - Simulation, analysis or hybrid
    - Tradeoff between accuracy and speed
  - Hierarchical optimization
    - Combination with second-level optimization

Source: C. Haubelt, J. Teich

ECE382N.23: Embedded Sys Dsgn/Modeling, Lecture 9 © 2024 A. Gerstlauer 27

Multi-Objective Evolutionary Algorithm

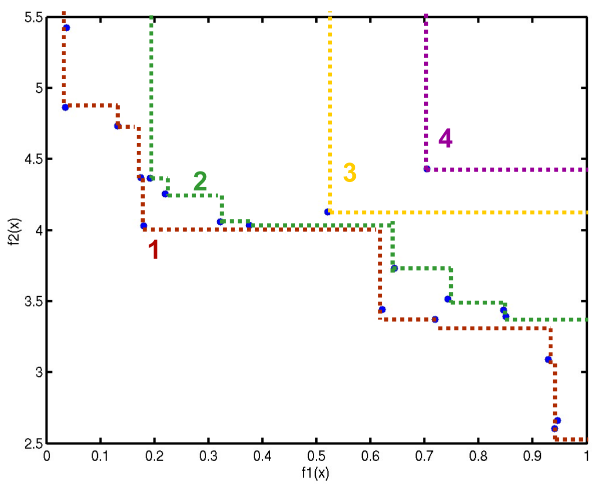


Source: C. Haubelt, J. Teich

ECE382N.23: Embedded Sys Dsgn/Modeling, Lecture 9 © 2024 A. Gerstlauer 28

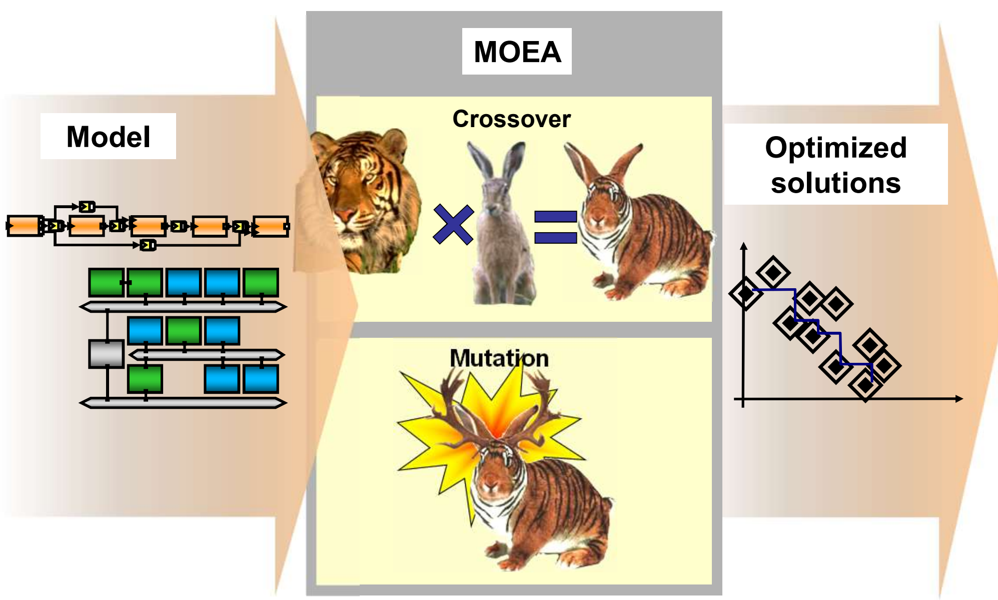
Fitness Selection

- **Pareto ranking**



Source: C. Haubelt, J. Teich

Recombination



Source: C. Haubelt, J. Teich

Lecture 9: Summary

- **System-level synthesis & decision making**
  - Formalization as a basis for automation
  - Partitioning (allocation, binding) & scheduling
- **Classical HW/SW co-design approaches**
  - Single processor + co-processors
- **Multi-processor mapping heuristics**
  - ILPs, list scheduling, simulated annealing
- **Design space exploration (DSE)**
  - Multi-objective optimization, MOEAs
- ➢ **Machine-learning based methods**
  - ➢ Reinforcement learning (robotics, game play)

