ECE382N.23: Embedded System Design and Modeling

Lecture 10 – Mapping & Exploration

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

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

Lecture 10: Outline

- **Hardware/software co-design**
  - Separate partitioning & scheduling definitions
  - Traditional partitioning & scheduling algorithms
- **System-level design**
  - Combined partitioning & scheduling
  - MPSoC mapping algorithms
- **Design space exploration**
  - Multi-objective optimization
  - Exploration algorithms

Partitioning

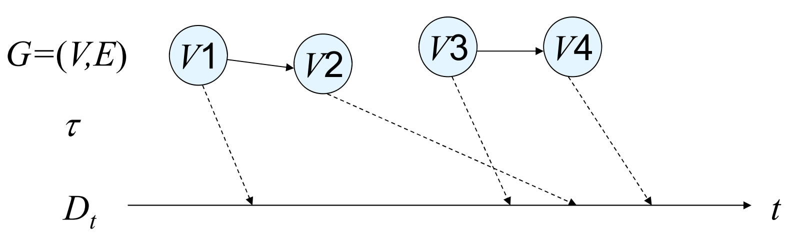
- **The partitioning problem is to assign *n* objects $O = \{o_1, ..., o_n\}$ to *m* blocks (also called partitions) $P = \{p_1, ..., p_m\}$, such that**
  - $p_1 \cup p_2 \cup ... \cup p_m = O$
  - $p_i \cap p_j = \{\ \}\ \forall i,j: i \neq j$ and
  - cost $c(P)$ is minimized

➢ **In system-level design:**
  - o_i = processes/actors
  - p_j = processing elements (hardware/software processors)
  - $c(P) = \sum$ cost of processor p_j (zero if unused) and/or communication cost between partitions
  - Constrain processor load and/or number of partitions
  - ➢ Bin packing and/or graph partitioning (both NP-hard)

Source: L. Thiele

Scheduling

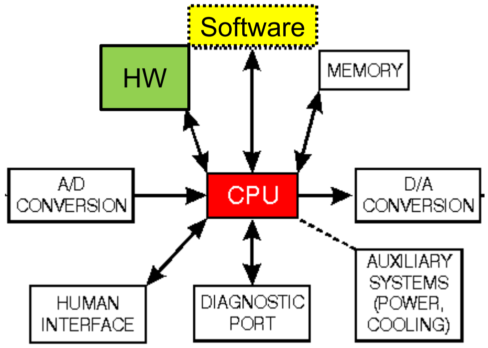
- **Assume that we are given a specification graph $G=(V,E)$**
- **A *schedule* τ of G is a mapping $V \rightarrow D_t$ of a set of tasks V to start times from domain D_t, such that none overlap**



➢ **In system-level design:**
  - Static vs. dynamic vs. quasi-static (static order)
  - Preemptive vs. non-preemptive (atomic)
  - Optimize throughput (rate of G), latency (makespan of G)
  - Resource, real-time (deadline) constraints
  - ➢ Implicit or explicit multi-processor partitioning (NP-hard)

Source: P. Marwedel

Traditional Hardware/Software Co-Design



- **Limited target architecture model**
  - Single CPU plus *N* hardware accelerators/co-processors
  - Often limited to single optimization objective
    - Minimize cost under performance constraints
    - Maximize performance under resource constraints
  - ➢ Classical approaches for partitioning & scheduling

Hardware/Software Partitioning

- **Constructive heuristics**
  - Hierarchical clustering
    - Minimize notion of communication cost between partitions
- **Iterative heuristics**
  - Kernighan-Lin (min-cut)
    - Minimize notion of communication cost between partitions
- **Meta-heuristics**
  - Simulated annealing
    - Generic optimization approach
    - ➢ Extends to multi-processor system-level design
  - …

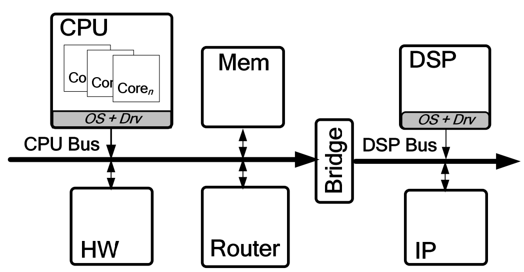
Hardware/Software Scheduling

- **Uni-processor scheduling**
  - General-purpose OS schedulers
    - – Balance average performance, fairness, responsiveness
  - Exact real-time scheduling methods
    - – Throughput/makespan fixed, minimize response (= meet deadlines)
    - – Analytical cost models based on estimated task execution times
    - ➢ EDD, RMS, EDF for independent periodic real-time task sets
    - ➢ LDF, EDF* for dependent task graphs
  - KPN, SDF scheduling of generalized task graphs
    - – Buffer/code sizing, completeness, ..
- **Uni-processor extensions**
  - Hardware accelerators as special cases
  - Extensions for (homogeneous) multi-cores

ECE382N.23: Embedded Sys Dsgn/Modeling, Lecture 10 © 2022 A. Gerstlauer 7

Multi-Processor Systems-on-Chip (MPSoCs)

- **Multi-processor**
  - Heterogeneous
  - Asymmetric multi-processing (AMP)
  - Distributed memory & operating system



- **Multi-core**
  - Heterogeneous or homogeneous or identical
  - Symmetric multi-processing (SMP)
  - Shared memory & operating system
  - ➢ Multi-core processors in a multi-processor system
- **Many-core**
  - > 10 processors/cores …

ECE382N.23: Embedded Sys Dsgn/Modeling, Lecture 10 © 2022 A. Gerstlauer 8

MPSoC Mapping

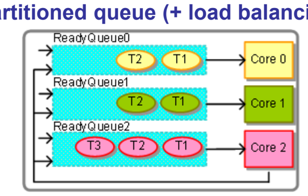
- **Partitioning**
  - Possible extensions of classical two-partition approaches
    - Min-cut, clustering, annealing
  - ➢ Truly parallel execution (not just accelerators)
    - Need to consider effect on scheduling
- **Scheduling**
  - Restricted multi-core scheduling
    - Periodic, independent tasks
    - Homogeneous processors/cores
    - ➢ Real-time extensions [G-EDF, P-Fair, …]
  - General multi-processor scheduling
    - General task graphs
    - Heterogeneous processors
    - ➢ Schedule & partitioning inter-dependent!

➢ **Integrated (allocation &) partitioning & scheduling**

Multi-Core Mapping

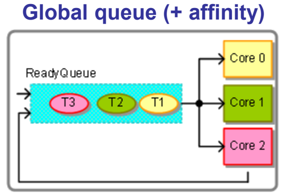
- **Partitioned scheduling**
  - Partition tasks to cores
  - Apply uni-processor scheduling on each core
  - Optional load-balancing

Partitioned queue (+ load balancing)



- **Global scheduling**
  - Fixed priorities [G-RMS]
  - Fixed job priority [G-EDF]
  - Dynamic [P-Fair]

Global queue (+ affinity)



➢ **Can not account for heterogeneity & dependencies**

Multi-Processor Mapping (1)

- **Models of computation**
  - Set of tasks (processes/actors) $\{ T_1, T_2, \ldots \}$
    - Independent
    - Task graph = data-flow/precedence graph (DFG/HSDF) = directed, acylic graph (DAG)
    - Generalized task models (KPN, SDF)
  - Timed models
    - Arrival/release times a_i (periods t_i), soft/hard deadlines d_i $(= t_i)$
- **Models of Architecture**
  - Set of processing elements (processors) $\{ P_1, P_2, \ldots \}$
    - Number and type fixed, constrained, or flexible
    - With or without migration, homogeneous or heterogeneous
  - Set of communication media (busses) $\{ B_1, B_2, \ldots \}$
    - Shared, point-to-point, fully connected
  - Set of storage elements (memories) $\{ M_1, M_2, \ldots \}$
    - Shared, distributed

Multi-Processor Mapping (2)

- **Optimization problems**
  - Cost models
    - Analytical: execution times e_i (best/worst/average?), real-time calc.
    - Simulation (dynamic scheduling, timing variations)
  - Objectives/constraints
    - Latency: response time r_i = finish time $f_i - a_i$, lateness $l_i = r_i - d_i$
    - Throughput: 1 / makespan (schedule length)
    - Cost: chip area, code/memory size, …

➢ **Examples (all at least NP-complete):**
  - General job-shop scheduling
    - Minimize makespan of independent task set on m processors
    - Classical multi-processor scheduling: atomic jobs, no migration
  - General task graph (DAG) scheduling
    - Minimize makespan for dependent task graph on m resources
    - Minimize resources under makespan constraint
    - Pipelined variants for periodic task graph invocations
  - KPN, SDF scheduling
    - Optimize latency, throughput, buffers, cost, … under *x* constraints

Multi-Processor Mapping Approaches

- **Exact methods**
  - Integer linear programming (ILP)
- **Constructive heuristics**
  - List schedulers to minimize latency/makespan
    - Hu's algorithm as optimal variant for uniform tasks & resources
  - Force-directed schedulers to minimize resources
- **Generic iterative heuristics**
  - Simulated annealing
  - Set-based multi-objective DSE approaches

➢ **Many of these adapted from other domains**
  - DAG/DFG scheduling in compilers & high-level synthesis
  - Production planning, operations research, …

Multi-Processor Mapping Approaches

- **Exact methods**
  - Integer linear programming (ILP)
- **Constructive heuristics**
  - List schedulers to minimize latency/makespan
    - Hu's algorithm as optimal variant for uniform tasks & resources
  - Force-directed schedulers to minimize resources
- **Generic iterative heuristics**
  - Simulated annealing
  - ➢ Set-based multi-objective DSE approaches

➢ **Many of these adapted from other domains**
  - ➢ DAG/DFG scheduling in compilers & high-level synthesis
  - ➢ Production planning, operations research, …

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:**

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

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

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

Source: L. Thiele

Integer Linear Program for Partitioning (2)

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

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

- **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, $1 \leq i \leq n$ with edges (t_i, t_j)
  - Discrete time window: $0 \leq t < T_{max}$
- **Decision variables**
  - $s_{i,t} \in \{0,1\}$: task t_i executes at time t
- **Constraints**
  - Single task execution: $\sum_t s_{i,t} = 1, \quad 1 \leq i \leq n$
  - Sequential task execution: $\sum_i s_{i,t} \leq 1, \quad 0 \leq t < T$
  - Task dependencies $t_i \rightarrow t_j$: $\sum_t t \cdot s_{j,t} \geq \underbrace{\sum_t t \cdot s_{i,t}}_{\text{Start time of task } t_i} + 1$
- **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: $\min \mathrm{w}_1 \sum_k \sum_i x_{i,k} \cdot c_{i,k} + \mathrm{w}_2 (\sum_t t \cdot s_{n,t} + \sum_k x_{n,k} \cdot e_n)$

ILP for Partitioning & Scheduling (2)

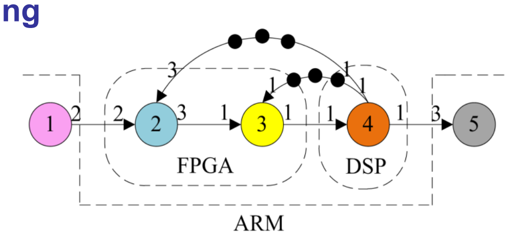
- **Inputs**
  - Tasks t_i, $1 \le i \le n$, edges (t_i,t_j), time window: $0 \le t < T_{max}$
  - Processors p_k, $1 \le k \le m$, cost $c_{i,k}$ if task t_i in processor p_k
  - Execution time $e_{i,k}$ of task t_i on processor p_k
- **Decision variables**
  - $s_{i,k,t} \in \{0,1\}$: task t_i starts 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:
    $$\text{minimize } w_1\left(\sum_k \sum_i \sum_t c_{i,k} \cdot s_{i,k,t}\right) + w_2\left(\sum_k \sum_t t \cdot s_{n,k,t} + \sum_k \sum_t s_{n,k,t} \cdot e_{n,k}\right)$$

SDF Partitioning & Scheduling

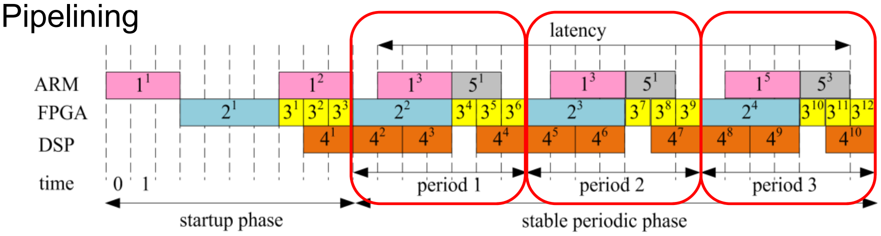
- **Inputs**
  - Actors a_i, $1 \le i \le n$, channels (a_i,a_j), time window: $0 \le t < T_{max}$
  - Production, consumption, initial rates/tokens on (a_i,a_j): $c_{i,j}$, $p_{i,j}$, $o_{i,j}$
  - Repetitions for actor a_i: r_i
  - Processors p_k, $1 \le k \le m$, cost $c_{i,k}$ if actor a_i in processor p_k
  - Execution time $e_{i,k}$ of actor a_i on processor p_k
- **Decision variables**
  - $s_{i,k,t} \in \{0,1\}$: actor t_i starts at time t on processor p_k
- **Constraints**
  - Single & unique actor mapping: $\sum_k \sum_t s_{i,k,t} = r_i, \quad 1 \le i \le n$
  - Sequential, non-overlapping execution on each processor:
    $$\sum_i \sum_{\tau=t-e_{i,k}+1}^{t} s_{i,k,\tau} \le 1, \quad 0 \le t < T,\ 1 \le k \le m$$
  - Token balance equations for each channel $a_i \rightarrow a_j$:
    $$\sum_k \sum_{\tau=0}^{t} c_{i,j} \cdot s_{j,k,\tau} \le \sum_k \sum_{\tau=0}^{t-e_{i,k}} p_{i,j} \cdot s_{i,k,\tau} + o_{i,j}, \quad 0 \le t < T$$
- **Objective**
  - Weighted cost & latency (unique sink a_n with $r_n = 1$):
    $$\text{minimize } w_1\left(\sum_k \sum_i c_{i,k} \cdot \underbrace{\mathrm{I}\{\textstyle\sum_t s_{i,k,t} > 0\}}_{\text{Indicator function}}\right) + w_2\left(\sum_k \sum_t t \cdot s_{n,k,t} + \sum_k \sum_t s_{n,k,t} \cdot e_{n,k}\right)$$

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

Multi-Processor Mapping Approaches

- Exact methods
  - Exhaustive search
  - Integer linear programming (ILP)
- Constructive heuristics
  - Random mapping
  - List schedulers to minimize latency/makespan
    - Hu's algorithm as optimal variant for uniform tasks & resources
  - Force-directed schedulers to minimize resources
- Generic iterative heuristics
  - Random search
  - Iterative improvement/hill climbing
  - Simulated annealing

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;
    }
}
```


Multi-Processor Mapping Approaches

- Exact methods
  - Exhaustive search
  - Integer linear programming (ILP)
- Constructive heuristics
  - Random mapping
  - List schedulers to minimize latency/makespan
    - Hu's algorithm as optimal variant for uniform tasks & resources
  - Force-directed schedulers to minimize resources
- **Generic iterative heuristics**
  - Random search
  - Iterative improvement/hill climbing
  - Simulated annealing

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 (single objective)
    - 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

Lecture 10: Outline

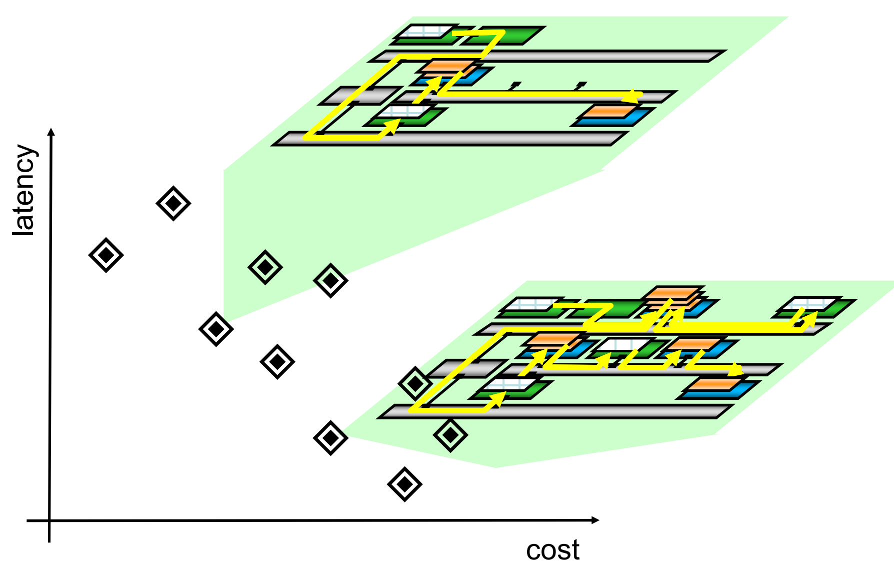
- ✓ Partitioning & scheduling
  - ✓ Problem definitions
- ✓ Hardware/software co-design
  - ✓ Traditional partitioning & scheduling algorithms
- ✓ System-level design
  - ✓ MPSoC mapping algorithms
- **Design space exploration**
  - Multi-objective optimization
  - Exploration algorithms

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 (EA, ACO, PSO)**
  - ➢ Randomized, problem independent (black box)
  - ➢ Often inspired by processes in nature (evolution, ant colonies, diffusion)

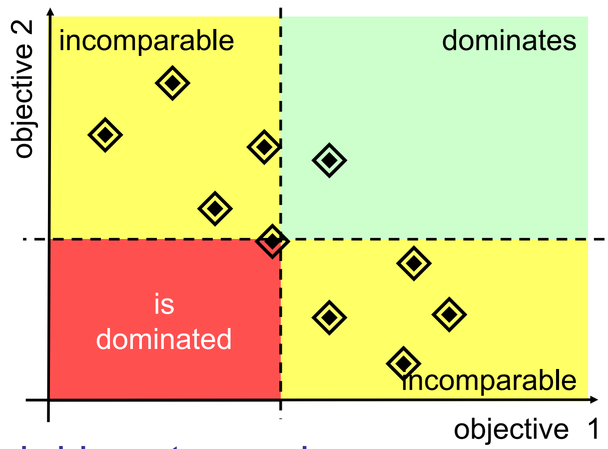
Source: C. Haubelt, J. Teich

Objective Space



Source: C. Haubelt, J. Teich

Pareto Dominance

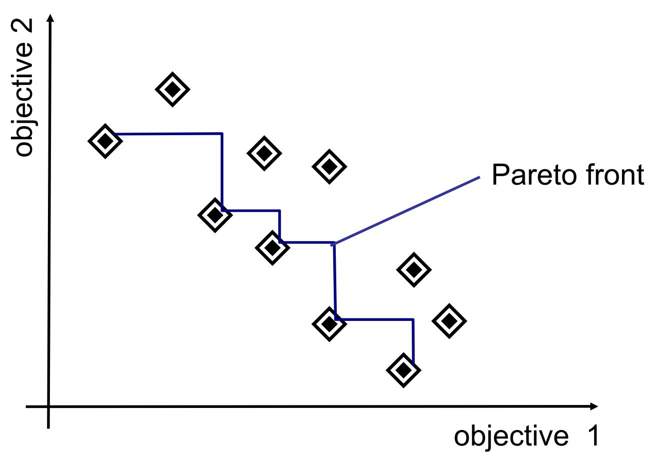


- **Given: two decision vectors x_1 and x_2**
  - $x_1 \succ\succ x_2$ (strongly dominates) if $\forall i: f_i(x_1)<f_i(x_2)$
  - $x_1 \succ x_2$ (dominates) if $\forall i: f_i(x_1)\leq f_i(x_2) \wedge \exists j: f_j(x_1)<f_j(x_2)$
  - $x_1 \sim x_2$ (indifferent) if $\forall i: f_i(x_1)=f_i(x_2)$
 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

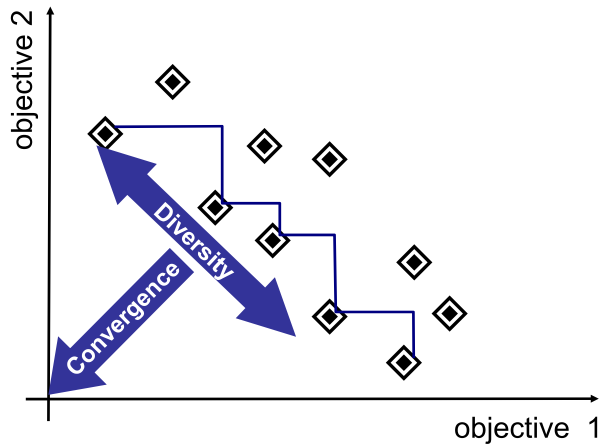
- **Set of all solutions *X***
- **A decision vector $x \in X$ is said to be *Pareto-optimal* if $\nexists y \in X: y \succ x$**



Source: C. Haubelt, J. Teich

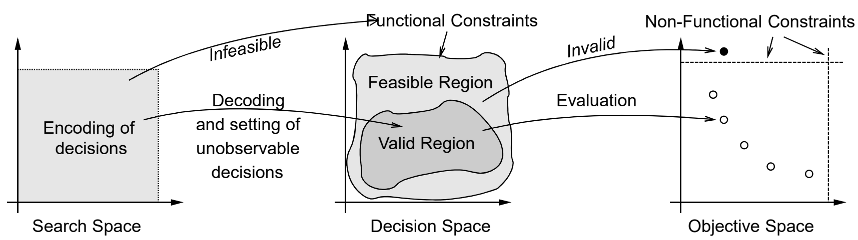
Optimization Goals

- **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

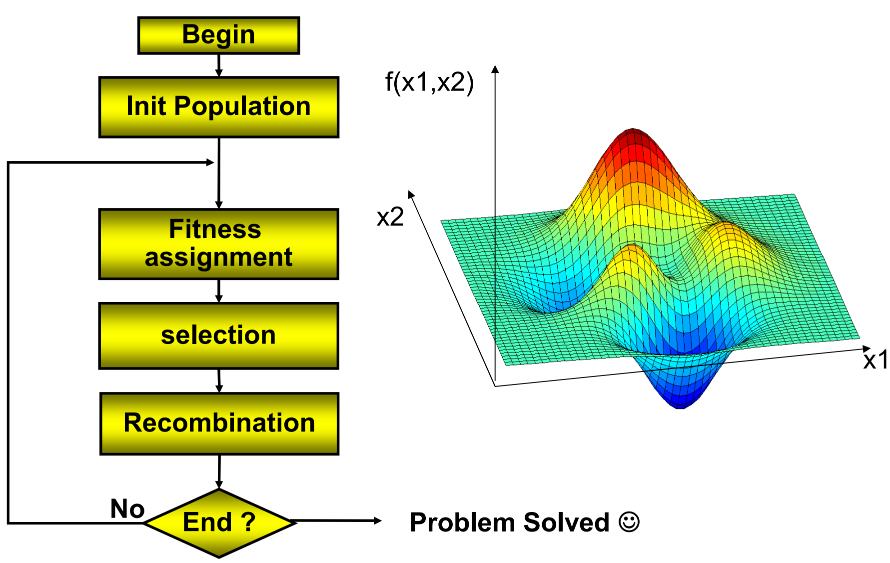
Evolutionary Algorithms (EAs)

- **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 10 © 2022 A. Gerstlauer 41

Multi-Objective Evolutionary Algorithm

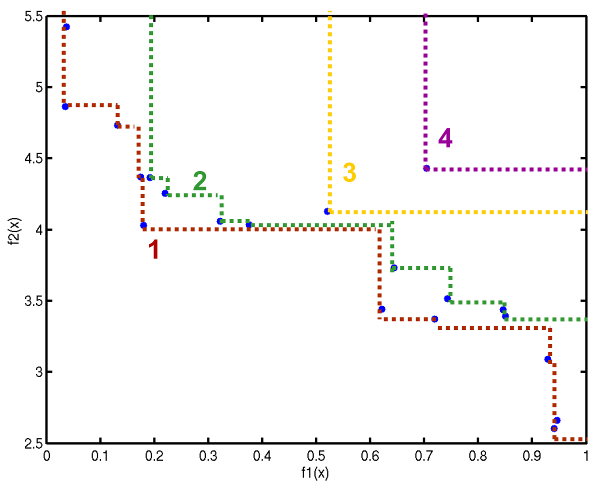


Source: C. Haubelt, J. Teich

ECE382N.23: Embedded Sys Dsgn/Modeling, Lecture 10 © 2022 A. Gerstlauer 42

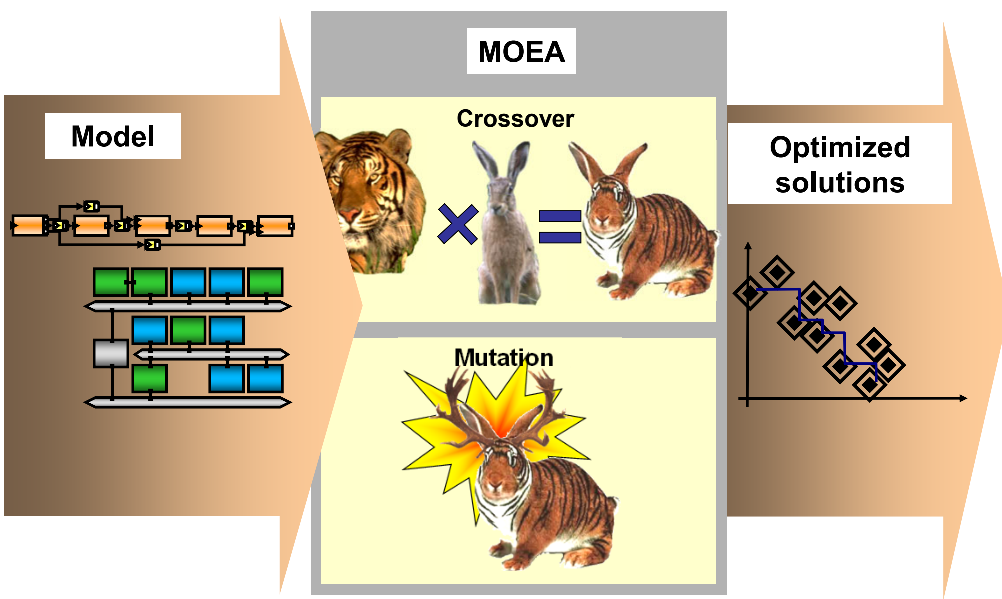
Fitness Selection

- **Pareto ranking**



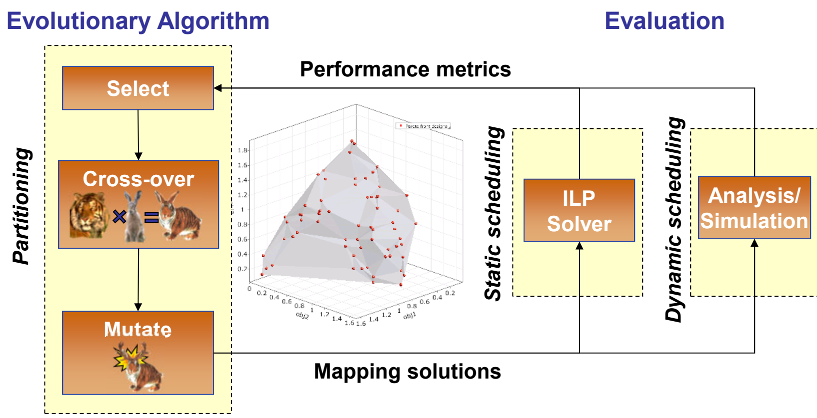
Source: C. Haubelt, J. Teich

Recombination



Source: C. Haubelt, J. Teich

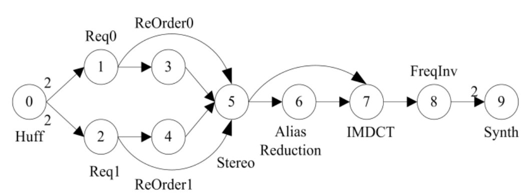
Hierarchical Optimization



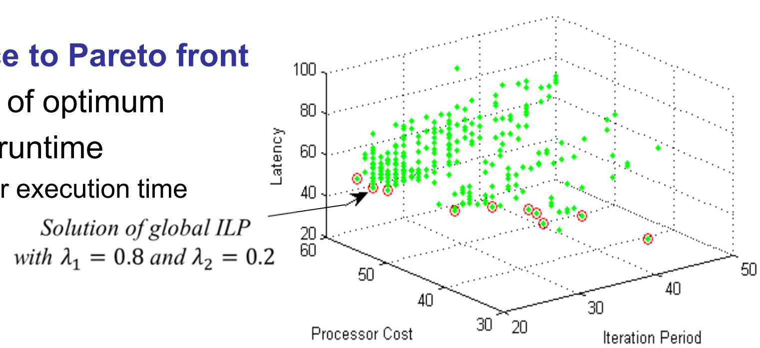
- **SDF mapping heuristics**
  - Multi-objective evolutionary algorithm (MOEA) + ILP
    - Partitioning + scheduling

SDF Mapping Results

- **Design space exploration for an MP3 decoder**



- **Convergence to Pareto front**
  - Within 10^{-6} of optimum
  - 12x better runtime
    - <1 hour execution time



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

Lecture 10: 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)