EE382V: Embedded System Design and Modeling

Lecture 8 – Mapping & Exploration

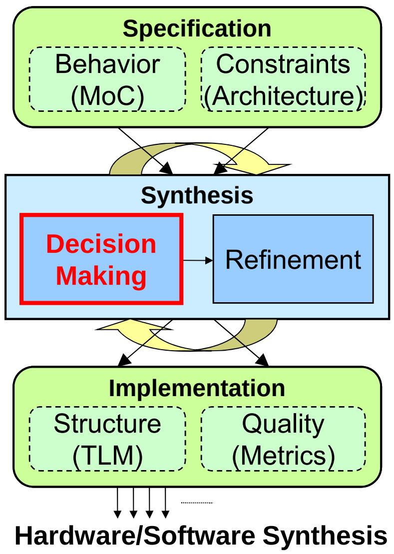
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

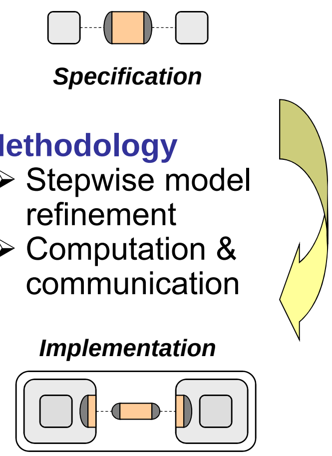
Lecture 8: Outline

- **Automated decision making**
  - Problem formulation
  - Optimization approaches
- **Partitioning & scheduling**
  - Traditional hardware/software co-design
  - System-level design
- **Design space exploration**
  - Multi-objective optimization
  - Exploration algorithms

System-Level Synthesis

- X-Chart





- ➢ Methodology
  - ➢ Stepwise model refinement
  - ➢ Computation & communication

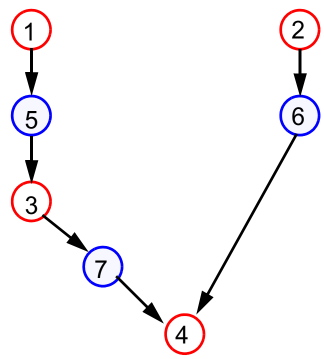
Automated Decision Making

- Map specification onto architecture
  - Functionality + constraints ⇨ structure + metrics
- Synthesis tasks
  - Allocation
    - Select resources from a platform/architecture template (database)
  - Binding
    - Map processes onto allocated computational resources
    - Map variables onto allocated storage units
    - Route channels over busses, gateways and address spaces
  - Scheduling
    - Determine order of processes bound to the same resource
    - Determine order of transaction routed over the same (arbitration)
  - ➢ Partitioning = (allocation +) binding
  - ➢ Mapping = (allocation +) binding + scheduling
- ➢ Formalization of decision making process

Example (1)

- **Basic model with a task graph MoC and static scheduling**
  - Task graph = homogeneous, acyclic SDF

Application task graph $G_P(V_P, E_P)$



Interpretation:

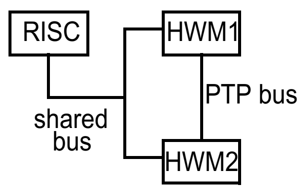
- V_P consists of functional nodes V_P^f (task, procedure) and communication nodes V_P^c .
- E_P represent data dependencies

Source: L. Thiele

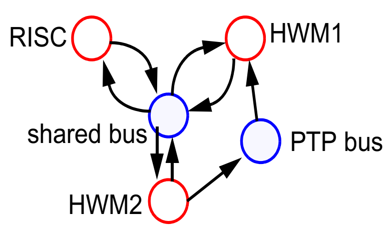
EE382V: Embedded Sys Dsgn and Modeling, Lecture 8 © 2014 A. Gerstlauer 5

Example (2)

Architecture graph $G_A(V_A,E_A)$:



Architecture



Architecture graph

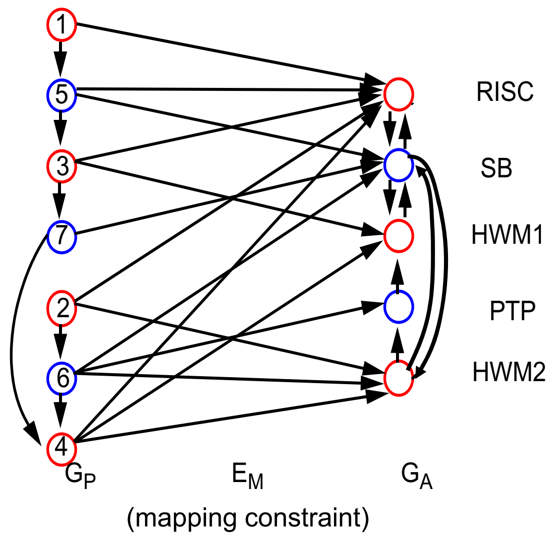
- V_A consists of functional resources V_A^f (RISC, ASIC) and bus resources V_A^c. These components are potentially allocatable.
- E_A model directed communication.

Source: L. Thiele

EE382V: Embedded Sys Dsgn and Modeling, Lecture 8 © 2014 A. Gerstlauer 6

Example (3)

Definition: A specification graph is a graph $G_S=(V_S,E_S)$ consisting of a problem graph G_P, an architecture graph G_A, and edges E_M. In particular, $V_S=V_P \cup V_A$, $E_S=E_P \cup E_A \cup E_M$



Source: L. Thiele

Example (4)

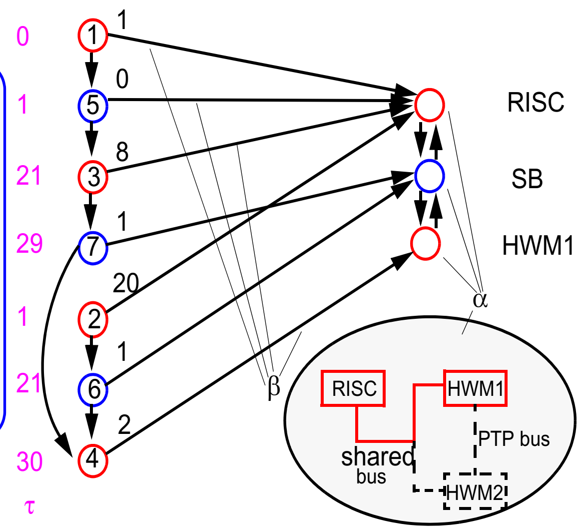
Three main tasks of synthesis:

- Allocation α is a subset of V_A.
- Binding β is a subset of E_M, i.e., a mapping of functional nodes of V_P onto resource nodes of V_A.
- Schedule τ is a function that assigns a number (start time) to each functional node.

Source: L. Thiele

Example (5)

Definition: Given a specification graph G_S an implementation is a triple (α,β,τ), where α is a feasible allocation, β is a feasible binding, and τ is a schedule.



Source: L. Thiele

Optimization

- **Decision making under optimization objectives**
  - Single- vs. multi-objective optimization
  - Couple with refinement for full synthesis
- **General optimization formulation**
  - Decision variables: $x \in$ *Domain*
  - Constraints: $g_i(x) \le G_i$, $h_j(x) = H_j$
  - Objective function: $f(x)$: *Domain* $\rightarrow \mathbb{R}$
  - Single-objective optimization problem:

$$\min_x f(x) \text{ subject to } g_i(x) \le G_i, h_j(x) = H_j$$

- **System-level optimization**
  - Allocation (α), binding (β), scheduling (τ) decisions
  - Under functional and non-functional constraints/objectives
    - Architecture & mapping constraints (G_A, E_m)
    - Design quality constraints & objectives

Cost Functions

- **Measure quality of a design point as optimization objective**
  - May include C … system cost in [$]
    L … latency in [sec]
    P… power consumption in [W]
- **Example: linear weighted cost function with penalty**

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

  - h_C , h_L , h_P … denote how strong C, L, P violate the design constraints $C_{max}, L_{max}, P_{max}$
  - k_1 , k_2 , k_3 … weighting and normalization
- **Requires *estimation or evaluation* to find C, L, P**
  - Analytical quality/cost model (estimation)
  - Refinement + simulation (evaluation)

Source: L. Thiele

Optimization Methods

- **Exact (optimal) methods**
  - Enumeration, exhaustive search
  - Convex optimizations
  - (Integer) linear programming
  - ➢ Prohibitive for exponential problems (large design spaces)
- **Heuristics (non-optimal)**
  - Constructive
    - Random assignment, list schedulers
  - Iterative
    - Random search, simulated annealing
  - Set-based iterative
    - Evolutionary/genetic Algorithms (EA/GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO)
    - ➢ Multi-objective optimization (MOO), Design space exploration (DSE)

➢ **Exact & constructive methods imply analytical cost models**

Source: C. Haubelt, J. Teich

Lecture 8: Outline

- ✓ **Automated decision making**
  - ✓ Problem formulation
  - ✓ Optimization approaches
- **Partitioning & scheduling**
  - Traditional hardware/software co-design
  - System-level design
- **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

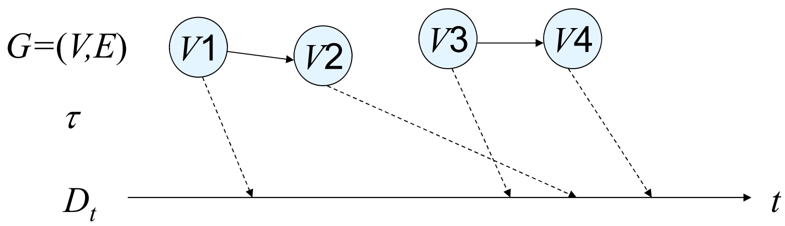
➢ **In system-level design:**
  - o_i = processes/actors
  - p_j = processing elements (hardware/software processors)
  - $c(P)$ = $\sum$ cost of processor p_j (zero if unused) and/or communication cost between partitions
  - Constrain processor load and/or fixed number of partitions

  ➢ 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, dependency, real-time (deadline) constraints

➢ Implicit or explicit multi-processor partitioning (NP-hard)

Source: P. Marwedel

Hardware/Software Co-Design (1)

➢ **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

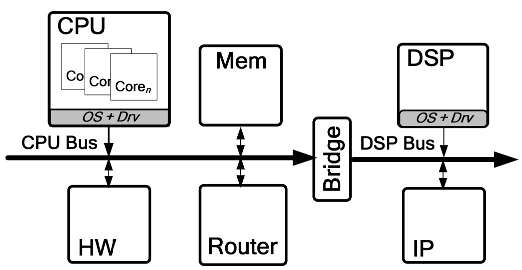
- **Constructive or iterative HW/SW partitioning**
  - Hierarchical clustering, Kernighan-Lin (min-cut)
    - Minimize notion of communication cost between partitions
  - Simulated annealing
    - Generic optimization approach
    - ➢ Extends to multi-processor system-level design
  - …

Hardware/Software Co-Design (2)

- **Uni-processor scheduling**
  - General-purpose OS schedulers
    - Balance average performance, fairness, responsiveness
  - Exact real-time scheduling methods
    - RMS, EDF for independent periodic real-time task sets
      - » Schedulability (maximize utilization while guaranteeing deadlines)
    - EDD, EDF for independent aperiodic real-time task sets
    - LDF, EDF* for dependent (real-time) task graphs
      - » Minimize maximal lateness (response time minus deadline)
    - Mix of (hierarchical) schedulers for indep. concurrent task graphs
    - ➢ Throughput/makespan fixed, minimize latency (= meet deadlines)
    - ➢ Analytical cost models based on estimated task execution times
  - KPN, SDF scheduling of generalized task graphs
    - Constructive methods, focus on buffer/code sizing, completeness, ..
  - ➢ Hardware accelerators as special cases
  - ➢ Extensions for (homogeneous) multi-cores

Multi-Processor Systems-on-Chip (MPSoCs)

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

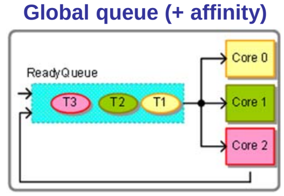


- **Multi-core**
  - Heterogeneous or homogeneous or identical
  - Symmetric multi-processing (SMP)
  - Shared memory & operating system
  - ➢ Multi-core processors in a multi-processor system
- **Many-core**
  - > 10 processors/cores …

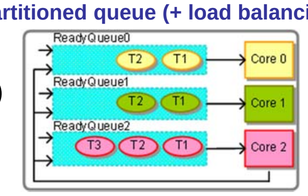
Multi-Processor Mapping

- **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**
  - Multi-core scheduling (SMP)
    - Tasks can migrate (frequency? overhead? cache pollution?)
    - ➢ Real-time extensions
      - Exact global P-fair scheduling for indep. periodic task sets
      - Partitioned/global EDF heuristics for indep./dep. task sets
  - ➢ True multi-processor scheduling (AMP)
    - General (dependent/aperiodic) tasks with or without migration (NP-hard)
    - ➢ Integrated partitioning & scheduling

Global queue (+ affinity)



Partitioned queue (+ load balancing)



Multi-Processor Mapping Formulations (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 Formulations (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 DAG/DFG 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, …

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)

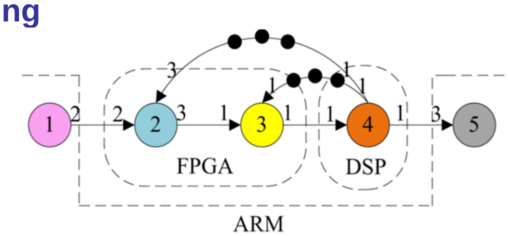
Source: L. Thiele

Integer Linear Program for Scheduling

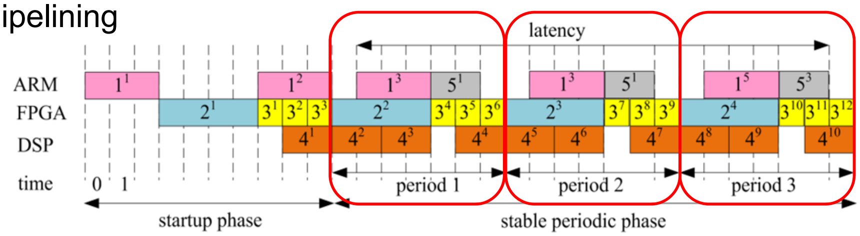
- **Task graph model**
  - Time window: $0 \le l \le T$
  - Execution time $e_{i,k}$ of task t_i on processor p_k
  - Cost $c_{i,k}$, if task t_i is in processor p_k
- **Decision variables**
  - $s_{i,l} \in \{0,1\}$: task t_i starts at time l
  - $x_{i,k} \in \{0,1\}$: task t_i in processor p_k
- **Constraints**
  - Single task execution: $\sum_l s_{i,l} = 1, \quad 1 \le i \le n$
  - Unique mapping of tasks to processors: $\sum_k x_{i,k} = 1, \quad 1 \le i \le n$
  - Non-overlapping execution on each processor: $\sum_i \sum_{\tau=l-e_{i,k}+1}^{l} x_{i,k} \cdot s_{i,\tau}, \quad 1 \le k \le m,\ 0 \le l \le T$
  - Task dependencies $t_i \rightarrow t_j$: $\sum_l l \cdot s_{j,l} \ \ge\ \sum_l l \cdot s_{i,l} + \sum_k x_{i,k} \cdot e_{i,k}$
- **Objective**
  - Weighted cost & latency: $w_1(\sum_i \sum_k c_{i,k} \cdot x_{i,k}) + w_2(\sum_l l \cdot s_{n,l} + \sum_k x_{n,k} \cdot e_{n,k})$

SDF Mapping

- **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)

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

Partitioning & Scheduling ILP (1)

- **Multi-objective cost function**
  - Minimize: $w_1 \cdot Throughput + w_2 \cdot Latency + w_3 \cdot Cost$
- **Decision variables**
  - Actor to processor binding
  - Actor start times
- **Constraints**
  - Execution precedence according to SDF semantics
  - Unique actor mapping
  - Processor-dependent actor execution times
  - Sequential execution on each processor
  - Stable periodic phase

➢ **Optimize partition and schedule simultaneously**

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

Partitioning & Scheduling ILP (2)

- **ILP formulation of multi-processor SDF mapping**
  - Inputs
    - Time window: $0 \le t \le T$
    - Repetition vector: number of executions r_i for actor i
    - Production and consumption rates on edge $i1\text{->}i2$: $c_{i1,i2}$, $p_{i1,i2}$
    - Initial tokens on edge $i1\text{->}i2$: $o_{i1,i2}$
    - Execution time of actor i on processor j: $d_{i,j}$
    - Cost of processor j: pc_j
  - Decision variables
    - $A_{i,j} \in \{0,1\}$: Actor i mapped to processor j
    - $S_i(t), E_i(t)$: Number of started/ended executions of actor i till time t
    - $start(t)$: Indicator for start of periodic phase
  - Helper variables
    - $W_i(t) = \sum_{\tau=0}^{t} (S_i(\tau) - E_i(\tau))$: number of executions of i at time t
    - $F_i(t)$: step function indicating first start of i in stable phase

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

Partitioning & Scheduling ILP (3)

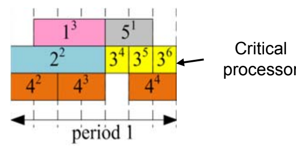
- **ILP formulation of multi-processor SDF mapping (cont'd)**
  - Constraints
    - Unique actor mapping: $\sum_j A_{i,j} = 1$
    - Actor execution time: $S_i(t) = \sum_j A_{i,j} E_i(t + d_{i,j})$
    - Token balance equations: $c_{i1,i2} S_{i2}(t) \le p_{i1,i2} E_{i1}(t) + o_{i1,i2}$
    - Sequential (non-overlapping) execution: $\sum_i A_{i,j}(S_i(t) - E_i(t)) \le 1$
    - Periodicity of schedule: $W_i(T) - \sum_t W_i(t) start(t) = r_i \sum_j A_{i,j} d_{i,j}$
  - Objectives
    - $Period = T - \sum_t t \cdot start(t)$
    - $Cost = \sum_j Alloc_j \cdot pc_j$
    - $Latency = \underbrace{\sum_t (F_1(t) - F_I(t)) + \sum_j A_{I,j} d_{I,j}}_{\text{Time interval between source's 1st start and sink's 1st end in the periodic phase}} + \underbrace{(S_1(T) - S_I(T))}_{\text{Difference in iteration numbers}} \cdot Period$

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

SDF Mapping Optimizations

- **Integer Linear Programming (ILP) formulation**
  - Optimal, but single solution only and exponential

➢ **Heuristics**
  - Maximum throughput partition
    - For fixed partition, the best throughput is determined by the critical processor
    - Best throughput achievable if acyclic SDF or enough initial tokens



  - Two-stage ILP optimization process

Stage I: Partitioning
Maximize throughput and minimize cost

→

Stage II: Scheduling
Minimize latency under throughput constraint

  ➢ Throughput and cost are prioritized over latency

➢ **Integrate communication model**
  ➢ J. Lin, A. Gerstlauer, B. Evans, “Communication-aware Heterogeneous Multiprocessor Mapping for Real-time Streaming Systems,” JSP’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: pi ← Idle;
Ready ← Initial tasks (no dependencies);
while (!empty(Ready)) {
   forall pi: status(pi) == Idle {
      t = first(Ready, pi);  // by priority
      pi ← (t, l, l + exec_time(t));
   }
   l = min(l + 1, finish_time(pi));
   forall pi: finish_time(pi) == l {
      Ready ← successors(current(pi));
      pi ← 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 model (KPN, DDF, …)
  - Iteratively decrease temperature
    - In each temperature step, perform random moves until equilibrium
    - Sometimes (with a certain probability) increases in cost are accepted.

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);
}
```

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

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 8: Outline

- ✓ **Automated decision making**
  - ✓ Problem formulation
  - ✓ Optimization approaches
- ✓ **Partitioning & scheduling**
  - ✓ Traditional hardware/software co-design
  - ✓ System-level design
- **Design space exploration**
  - Multi-objective optimization
  - Exploration algorithms

Multi-Objective Exploration

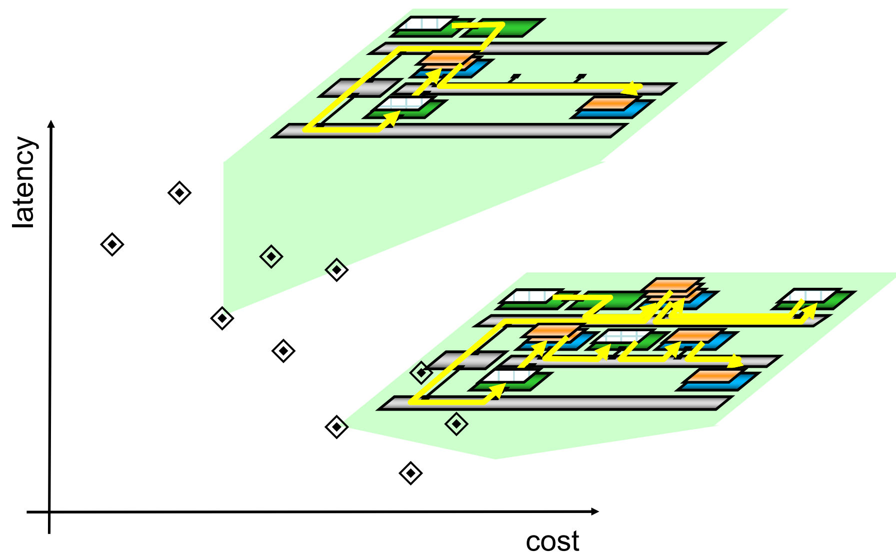
- **Multi-objective optimization (MOO)**
  - In general, several solutions (implementations) exist with different properties, e.g., area and power consumption, throughput, etc.
  - Implementations are often optimized with respect to many (conflicting) objectives
  - Finding best implementations is task of multi-objective optimization

➢ **Exact, constructive & iterative methods are prohibitive**
  - ➢ Large design space, multiple objectives, dynamic behavior

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

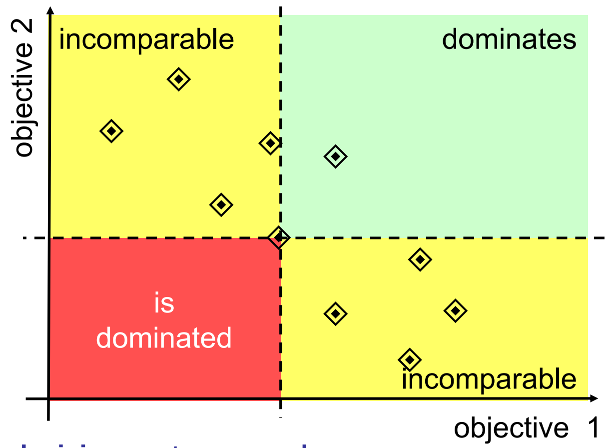
Source: C. Haubelt, J. Teich

Objective Space



Source: C. Haubelt, J. Teich

Pareto Dominance

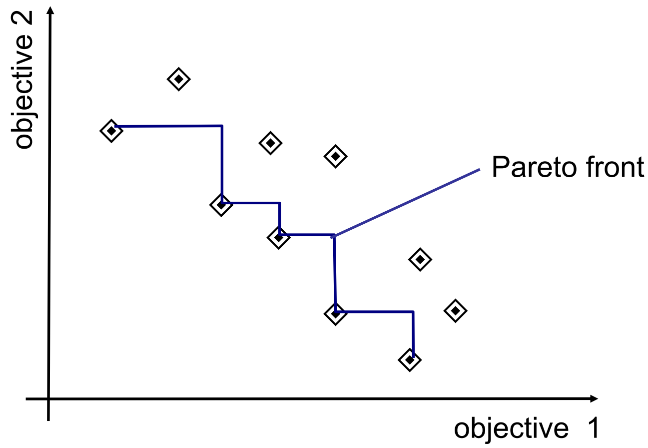


- **Given: two decision vectors x_1 and x_2**
  - $x_1 \succ\succ x_2$ (strongly dominates) if $\forall i$: $f_i(x_1)<f_i(x_2)$
  - $x_1 \succ x_2$ (dominates) if $\forall i$: $f_i(x_1)\leq f_i(x_2) \wedge \exists j$: $f_j(x_1)<f_j(x_2)$
  - $x_1 \sim x_2$ (indifferent) if $\forall i$: $f_i(x_1)=f_i(x_2)$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

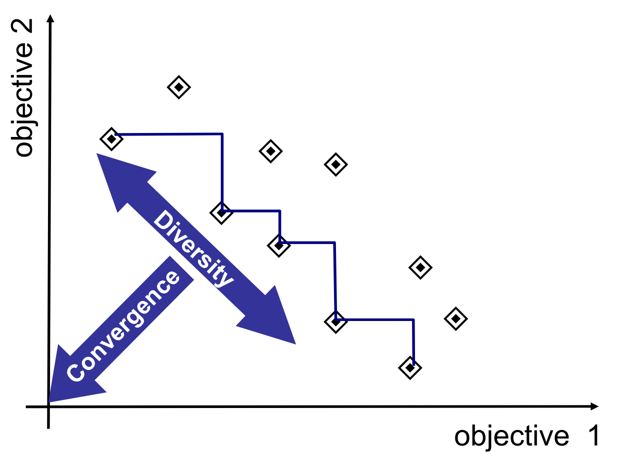
- Set of all solutions X
- A decision vector $x \in X$ is said to be *Pareto-optimal* if $\nexists y \in X$: $y \succ x$



Source: C. Haubelt, J. Teich

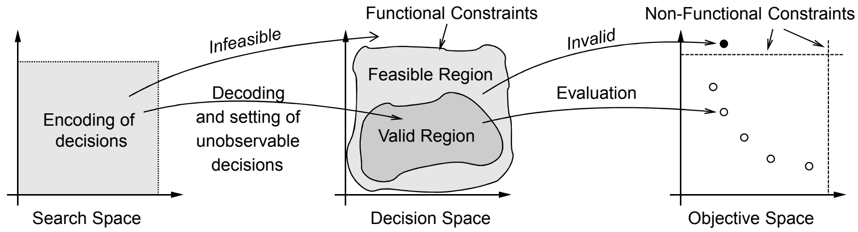
Optimization Goals

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



Source: C. Haubelt, J. Teich

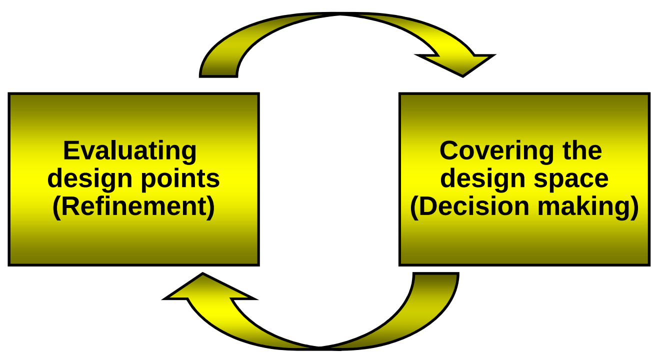
Design Space Exploration (DSE)



- **Search space vs. decision space vs. design space**
  - Encoding of decisions defines search space
    - Focus on observable decisions, hardcode unobservable ones
      - » No observable effect on design quality, e.g. address mappings
  - Functional & architecture constraints define decision space
    - Quickly prune & reject infeasible decisions
      - » Smart encoding, avoid during construction, attach large quality penalty
  - Quality constraints restrict objective space
    - Invalid solutions outside of valid quality range

Design Space Exploration (DSE)

- **Design Space Exploration is an iterative process**
  - How can a single design point be evaluated?
    - Most DSE approaches rely on simulation-based cost models
  - How can the design space be covered during the exploration process?



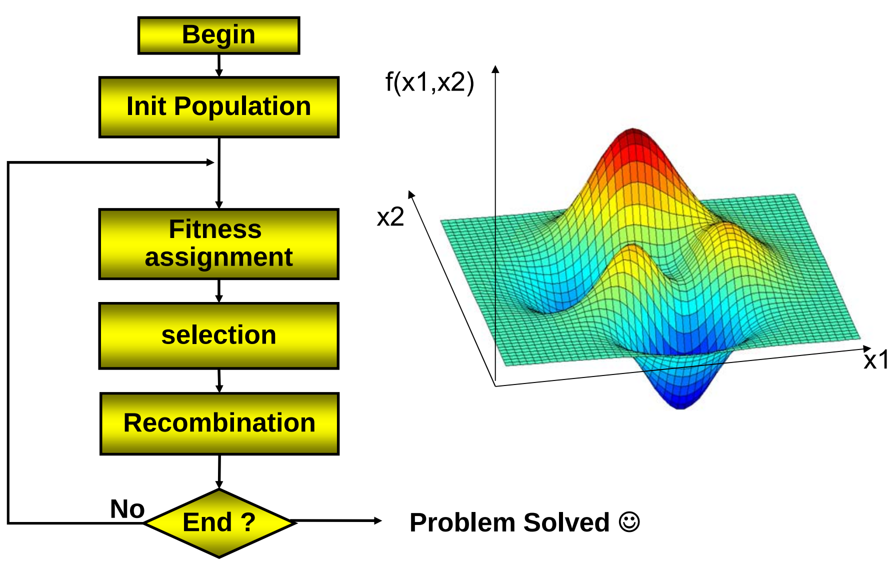
Source: C. Haubelt, J. Teich, Univ. of Erlangen-Nuremberg

Design Space Exploration (DSE)

- **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

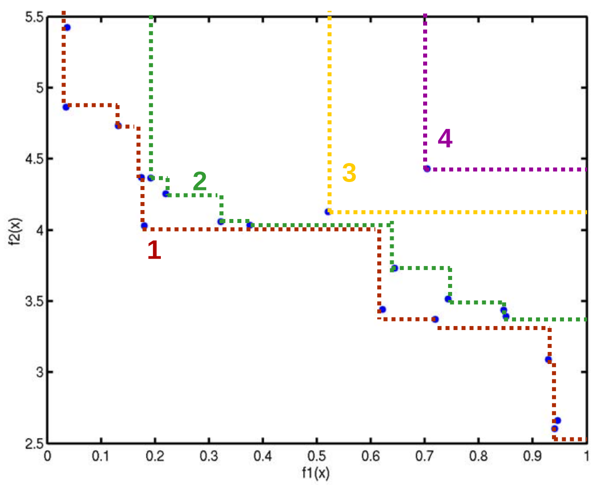
Multi-Objective Evolutionary Algorithm



Source: C. Haubelt, J. Teich

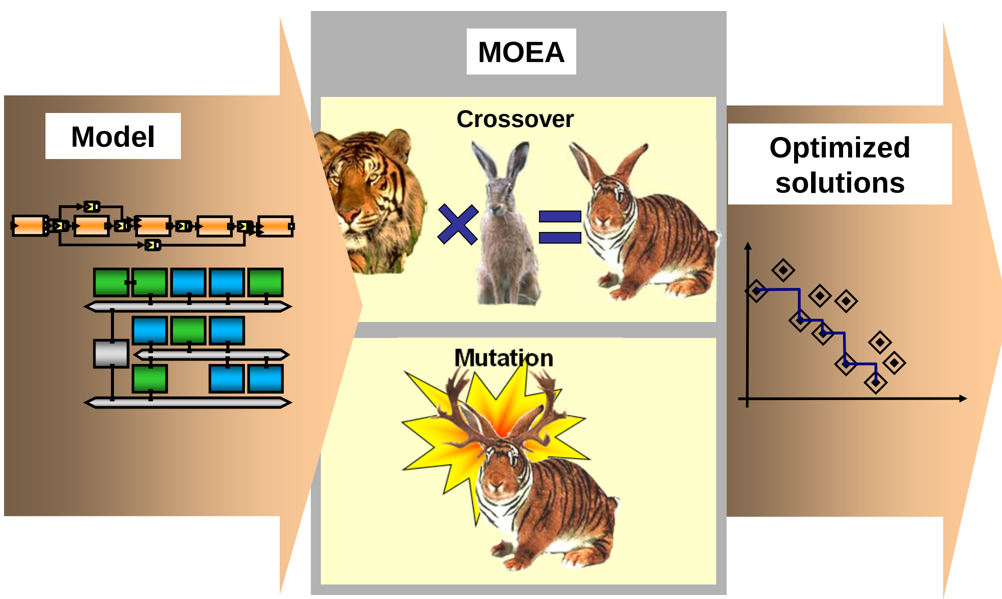
Fitness Selection

- **Pareto ranking**



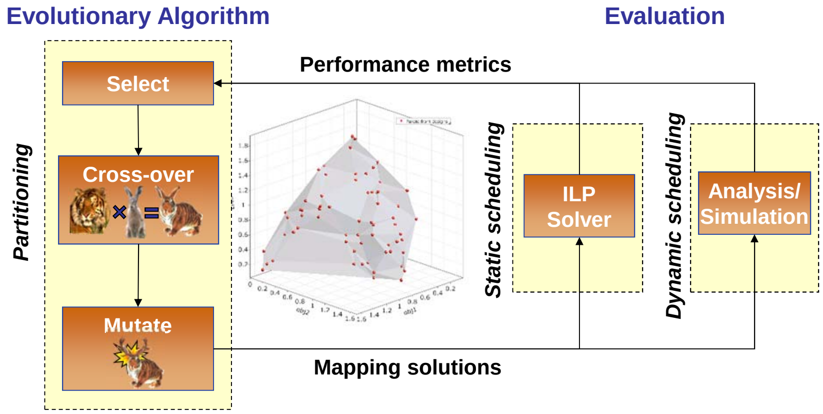
Source: C. Haubelt, J. Teich

Recombination



Source: C. Haubelt, J. Teich

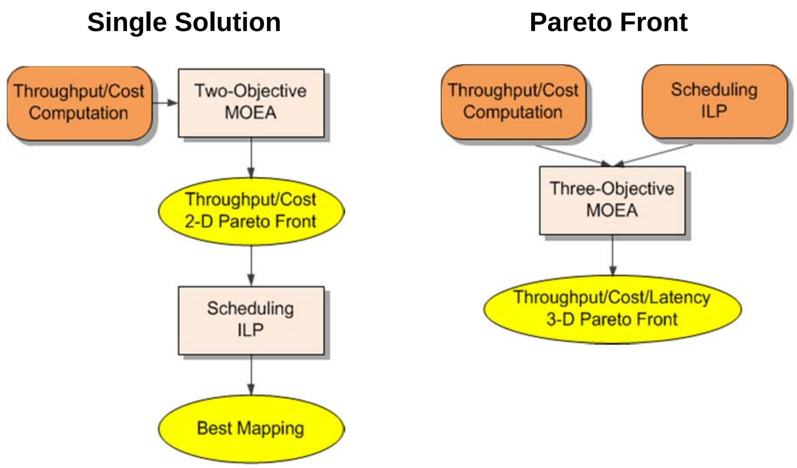
Hierarchical Optimization



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

SDF Mapping Heuristics

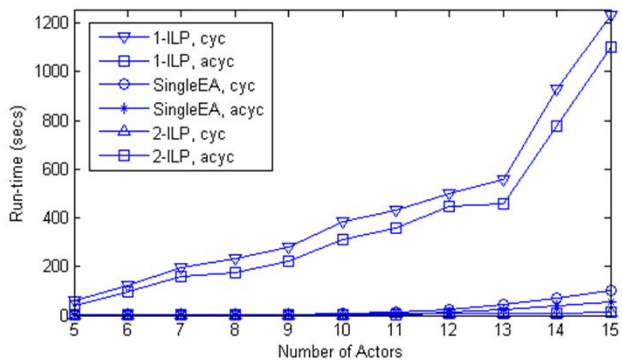
- **MOEA with Scheduling ILP**



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

SDF Mapping Results (1)

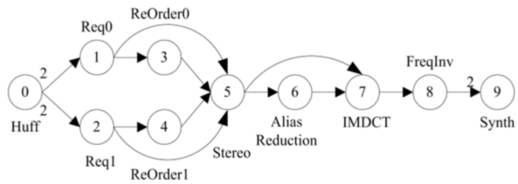
- **Run-time comparison**
  - Artificial cyclic/acyclic SDF graphs mapped to 3 processors



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

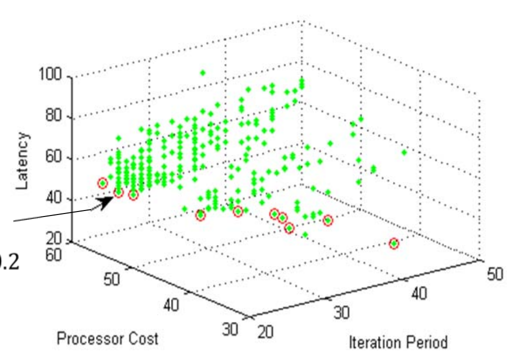
SDF Mapping Results (2)

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



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

Solution of global ILP with $\lambda_1 = 0.8$ and $\lambda_2 = 0.2$



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

Lecture 8: Summary

- **Multi-Processor Mapping**
  - Formalization as a basis for automation
    - – Partitioning (allocation, binding) & scheduling
    - – General optimization problems
  - Classical HW/SW co-design approaches
    - – Single processor + co-processors
    - – Real-time scheduling theory
  - Multi-processor mapping heuristics
    - – ILPs, list scheduling, simulated annealing
  - Design space exploration (DSE)
    - – Multi-objective optimization (MOO)
    - – Set-based iterative methods: MOEAs