ECE382N.23: Embedded System Design and Modeling

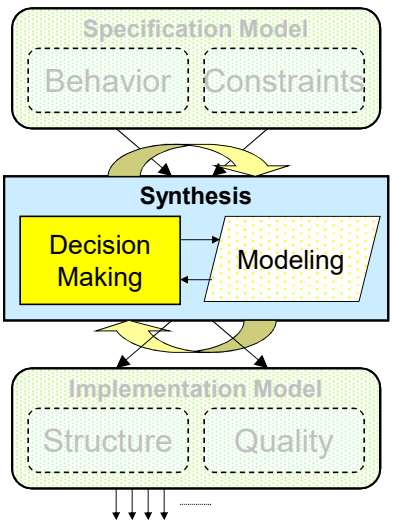
Lecture 8 – System-Level Mapping & Exploration

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

System-Level Synthesis

- **Decision making**
  - Allocation
  - Binding
  - Scheduling
  - ➢ Mapping
- ➢ **Design-time**
  - Static allocation & mapping
- ➢ **Run-time**
  - Dynamic scheduling & mapping
- **Modeling**
  - Create and evaluate architecture & cost/quality models



Lecture 8: Outline

- **Hardware/software co-design**
  - Separate partitioning & scheduling definitions
  - Traditional partitioning & scheduling algorithms
- **System-level design**
  - Multi-processor systems-on-chip (MPSoCs)
  - Combined partitioning & scheduling
- **Mapping problems**
  - Mapping formulations
  - Mapping approaches

Partitioning (Allocation + Binding)

- **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

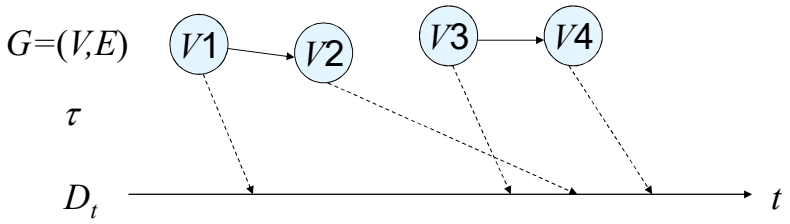
➢ **In system-level design:**

- 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

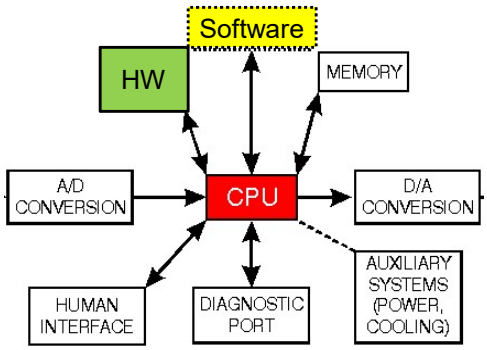
- **Assume that we are given a specification graph $G=(V,E)$**
- **A *schedule* τ of G is a mapping $V \rightarrow D_t$ of a set of tasks V to start times from domain D_t, such that none overlap**



- **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

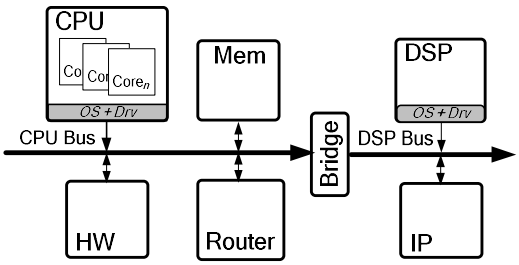
- **Constructive heuristics**
  - Hierarchical clustering
    - Minimize notion of communication cost between partitions
- **Iterative heuristics**
  - Kernighan-Lin (min-cut)
    - Minimize notion of communication cost between partitions
- **Meta-heuristics**
  - Simulated annealing
    - Generic optimization 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

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 …

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

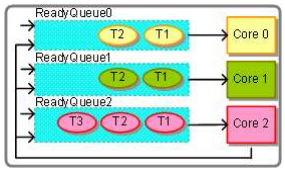
- **Scheduling**
  - Restricted multi-core scheduling
    - Periodic, independent tasks
    - Homogeneous processors/cores
    - ➢ Real-time extensions [G-EDF, P-Fair, …]
  - General multi-processor scheduling
    - General task graphs
    - Heterogeneous processors
    - ➢ Schedule & partitioning inter-dependent!

- ➢ **Integrated partitioning & scheduling**

Multi-Core Mapping

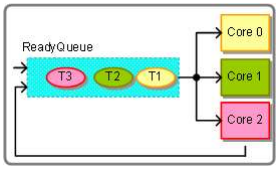
- **Partitioned scheduling**
  - Partition tasks to cores
  - Apply uni-processor scheduling on each core
  - Optional load-balancing

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)

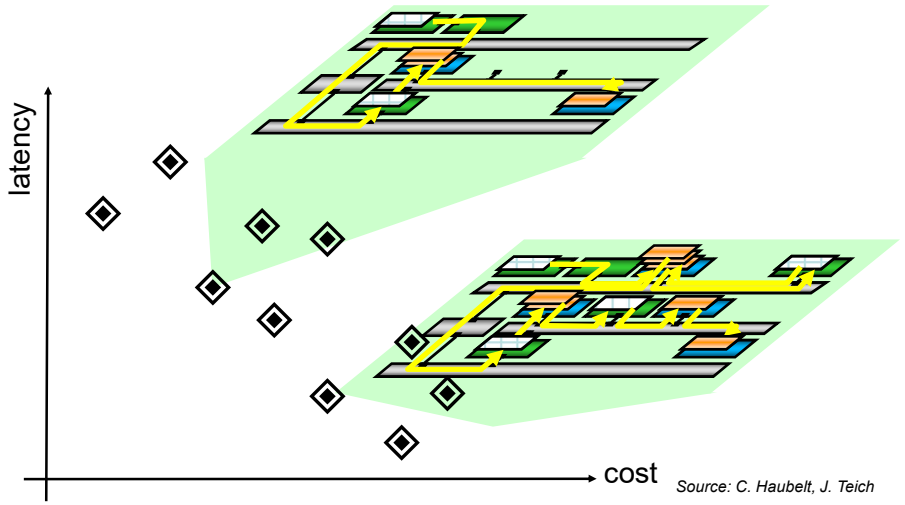
- **Models of computation**
  - Set of tasks (processes/actors) $\{ T_1, T_2, \ldots \}$
    - Independent
    - Task graph = data-flow/precedence graph (DFG/HSDF) = directed, acylic graph (DAG)
    - Generalized task models (KPN, SDF)
  - Timed models
    - Arrival/release times a_i (periods t_i), soft/hard 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

Design Space Exploration (DSE)

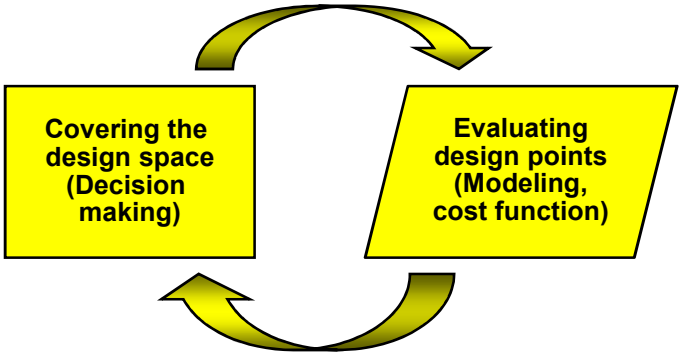
- **Static design-time decisions**
- **Single- vs. multi-objective optimization**
- **Heuristic search for Pareto-optimal design points**



Source: C. Haubelt, J. Teich

Optimization & Exploration

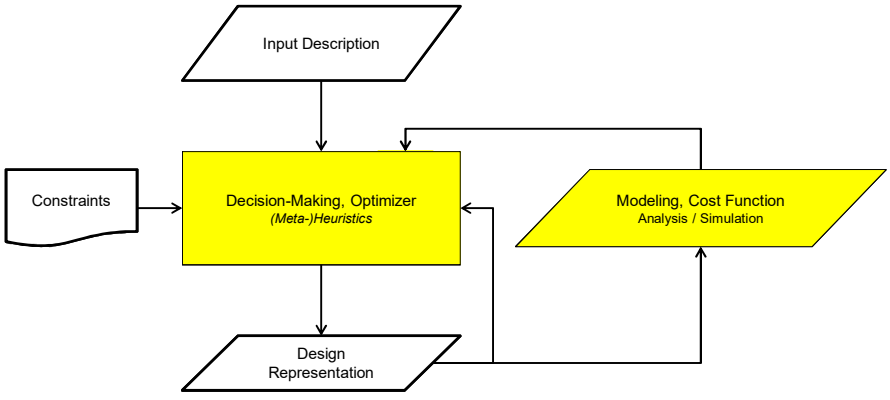
- **Design space exploration (DSE)**
  - Exact: exhaustive search
  - Incremental: local, stepwise exploration
  - Iterative: global search & exploration



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

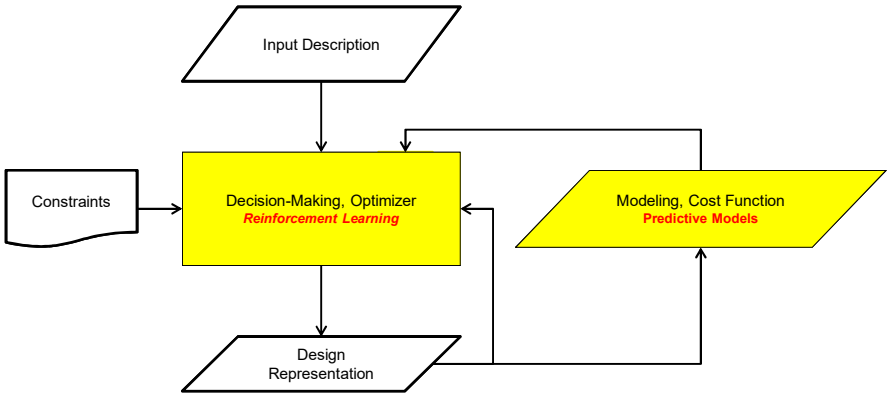
Traditional Design Flow

- **Manually derived heuristics & models**
  - Analytical & simulation-based models
  - Optimization & exploration (meta-)heuristics



Machine Learning (ML)-Based Design Flow

- **Apply advanced ML methods for modeling & optimization**
  - Predictive modeling
  - Reinforcement or other learning for optimization



Classic 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 DSE meta-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, …

ML-Based Mapping Approaches

- **Supervised learning**
  - Decision trees & variants
  - (Deep) neural networks
- **Reinforcement learning**
  - Q-learning
  - Multi-arm bandits (MAB)
  - Monte-carlo tree search (MCTS)
  - Deep reinforcement learning
- **Bayesian optimization**
  - Classic Bayesian methods
  - DSE-specific extensions
- **…**

➢ **Can be incremental/constructive or iterative**
➢ **Can be design-time (DSE) or run-time (OS)**