ECE382M.20: System-on-Chip (SoC) Design

Lecture 6 – HW/SW Partitioning & Scheduling

Sources:
Prof. Margarida Jacome, UT Austin
Prof. Lothar Thiele, ETH Zürich
Prof. Peter Marwedel, TU Dortmund

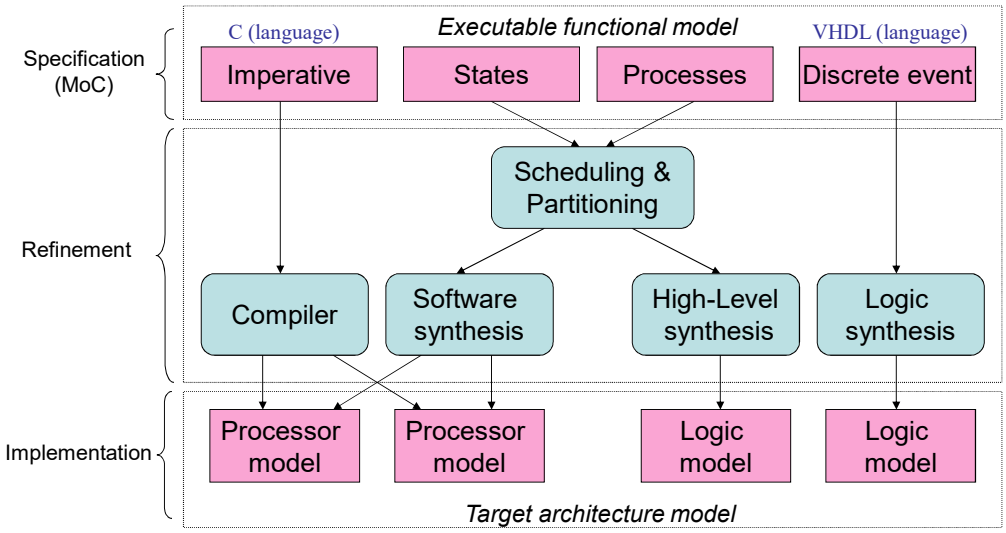
Andreas Gerstlauer
Electrical and Computer Engineering
The University of Texas at Austin
gerstl@ece.utexas.edu

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

Lecture 6: Outline

- **HW/SW co-design tasks**
  - System mapping
- **Partitioning**
  - Constructive heuristics
  - Iterative heuristics
- **Scheduling**
  - Uni-processor scheduling
  - Task graph scheduling
- **System-level design**
  - MPSoC synthesis

Design Synthesis



Synthesis Tasks

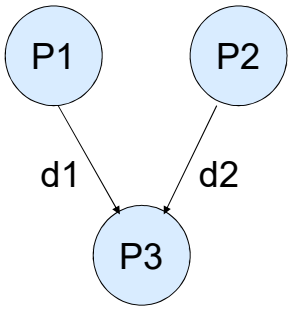
- **Mapping**
  - Allocate resources (hardware/software processors)
  - Bind computations to resources
  - Schedule operations in time

  ➢ Partitioning = (allocation +) binding
  ➢ Mapping = binding + scheduling

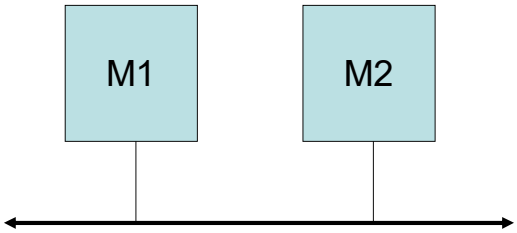
- **Allocation, scheduling and binding interact, but separating them helps**
  - Alternatively allocate, bind, then schedule

© 2023 A. Gerstlauer

Mapping Example



Task graph

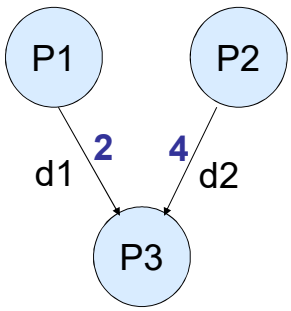


Hardware platform

Example Cost Model

- **Process execution times**

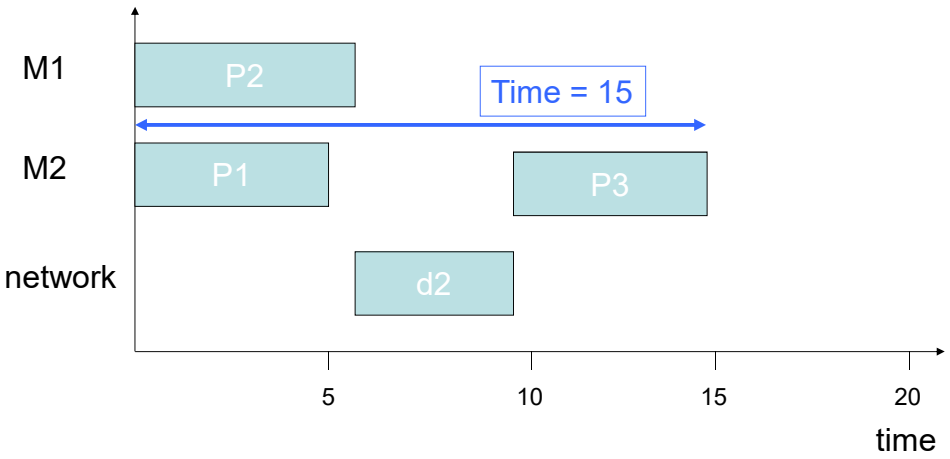
	M1	M2
P1	5	5
P2	5	6
P3	--	5



- **Communication cost**
  - Assume communication within PE is free
  - Cost of communication from P1 to P3 is d1 = 2
  - Cost of P2 to P3 communication is d2 = 4

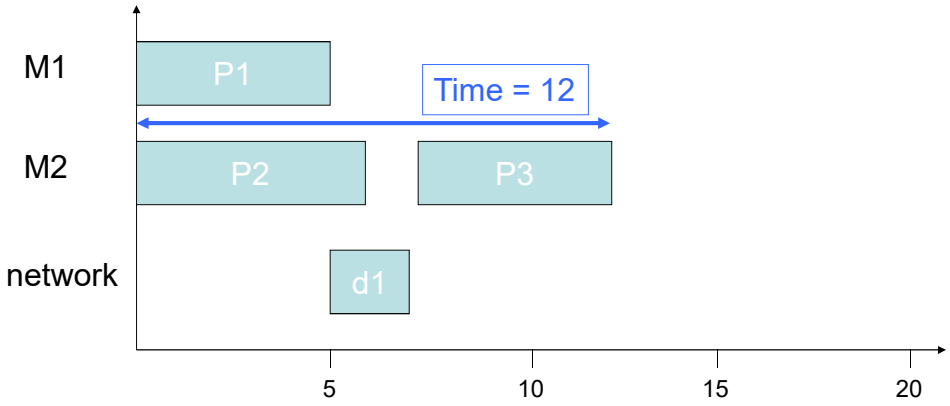
First Design

- **Allocate P2 -> M1; P1, P3 -> M2.**



Second Design

- **Allocate P1 -> M1; P2, P3 -> M2:**

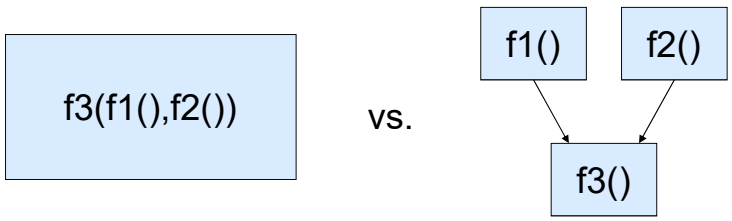


Lecture 6: Outline

- ✓ HW/SW co-design tasks
  - ✓ System mapping
- **Partitioning**
  - Constructive heuristics
  - Iterative heuristics
- Scheduling
  - Uni-processor scheduling
  - Task graph scheduling
- System-level design
  - MPSoC synthesis

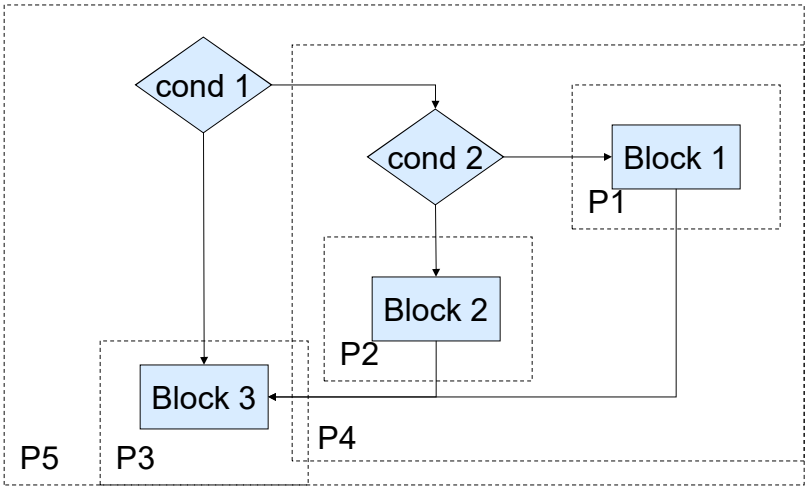
Decomposition

- **Divide functional specification into modules**
  - Map units onto PEs
  - Units may become processes
- **Determine proper level of parallelism**



Decomposition Example

- **Divide program into Control-Data Flow Graph (CDFG)**
- **Hierarchically decompose CDFG to identify partitions**



Partitioning

- **Assign tasks (objects) to processors (partitions) such that all objects are assigned to unique partitions**
  - Minimize communication cost (graph partitioning)
  - Minimize partition count (bin packing)
  - Partition size, partition count, etc. constraints

➢ **Exact methods**
  - Exhaustive enumeration, integer linear programming (ILP)

➢ **Constructive heuristics**
  - Random mapping, hierarchical clustering

➢ **Iterative heuristics**
  - Hill climbing, Kernighan-Lin, simulated annealing

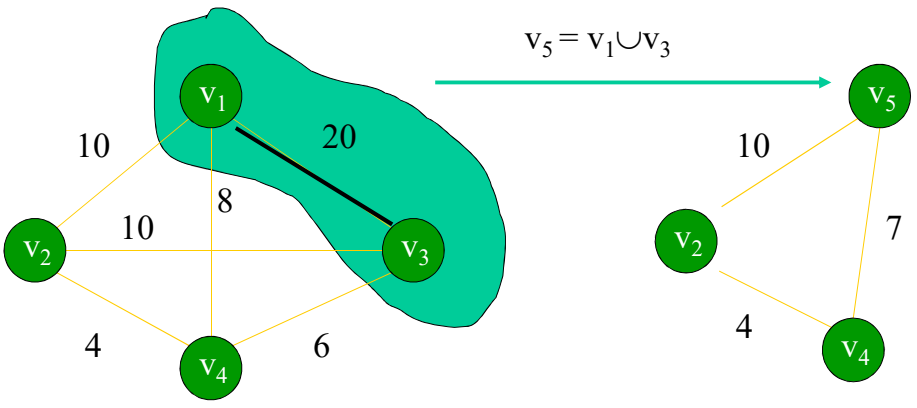
Constructive Methods

- **Construct solution one by one**
  - Visit every object once
  - Can generate a starting partition for iterative methods
  - Shows the difficulty of finding proper closeness functions

➢ **Random mapping**
  - Each object is assigned to a block randomly

➢ **Hierarchical clustering**
  - Stepwise grouping of objects
  - Closeness function determines how desirable it is to group two objects

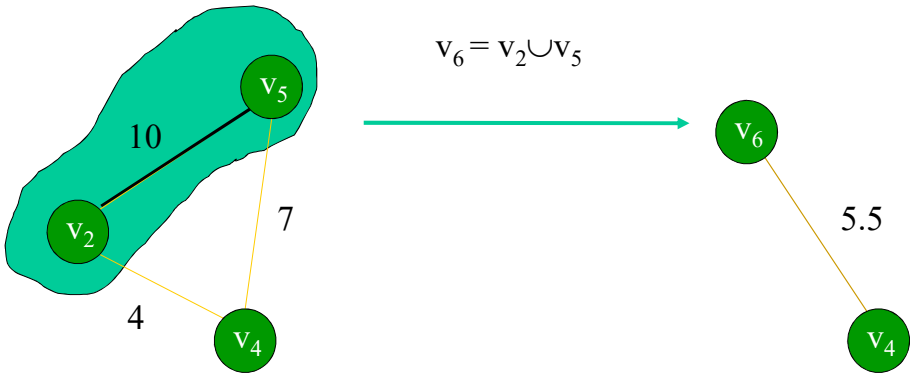
Hierarchical Clustering - Example (1)



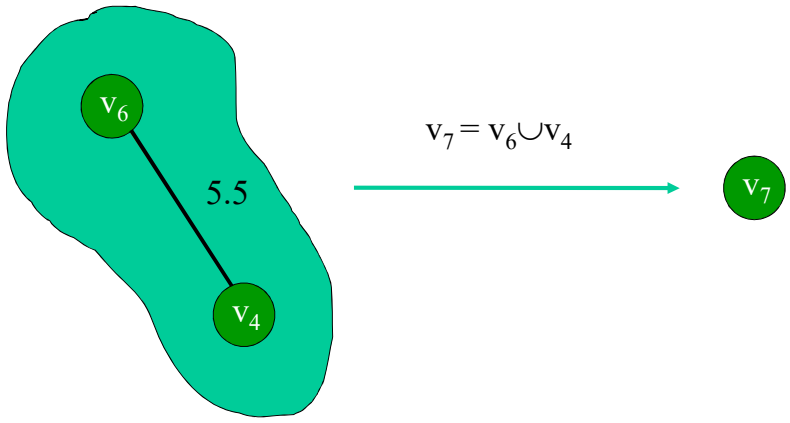
Closeness function: arithmetic mean of weights

© 2023 A. Gerstlauer

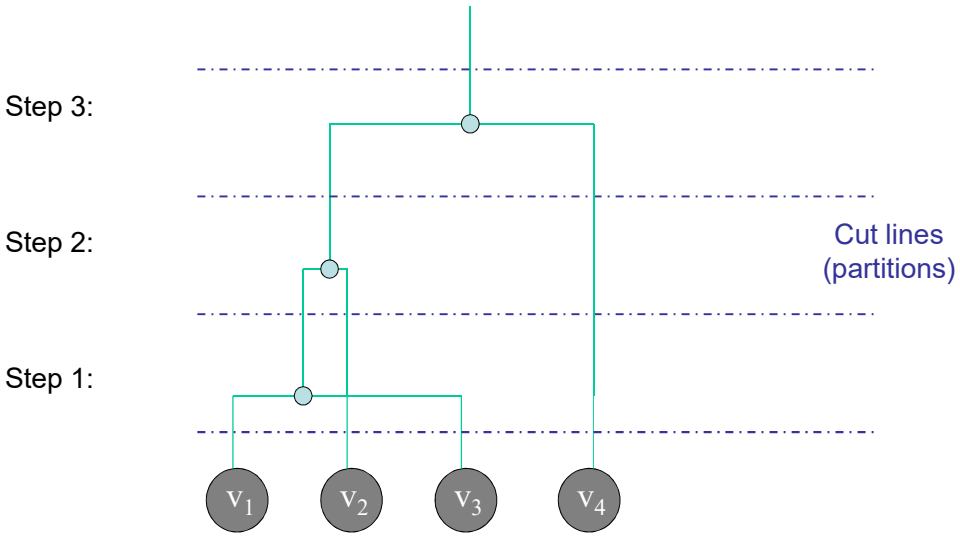
Hierarchical Clustering - Example (2)



Hierarchical Clustering - Example (3)



Hierarchical Clustering - Example (4)



Iterative Methods

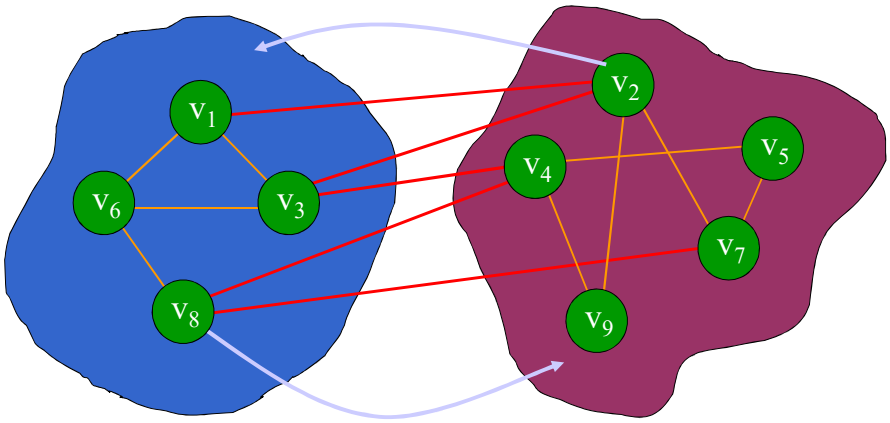
- **Principles**
  - Start with some initial solution
  - Search neighborhood (similar solutions), select candidate and make local change based on fitness/cost function
  - End on stopping criterion

➢ **Simple iterative improvement or "hill climbing"**
  - Select candidate with best improvement in cost
  - Stop when no candidate with lower cost is found

➢ **Kernighan-Lin**
  - More exhaustive search to escape local optima

Iterative Improvement (Hill Climbing)

- **Simple greedy heuristic**
  - Until there is no improvement in cost: re-group a pair of objects which leads to the largest gain in cost



Example: Cost = number of edges crossing the partitions
Before re-group: 5 ; after re-group: 4 ; gain = 1

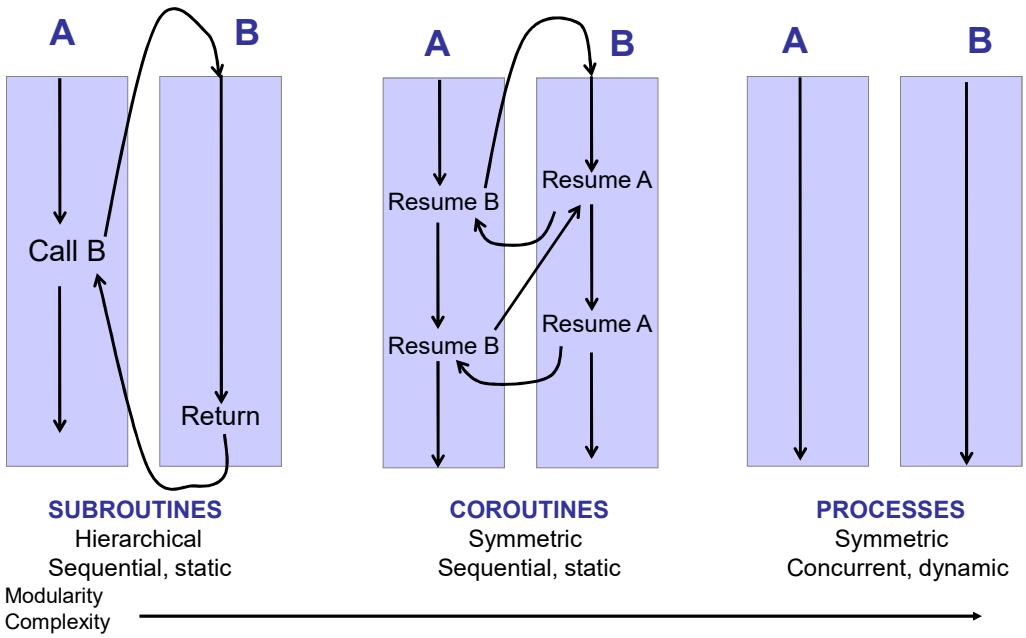
Kernighan-Lin

- **Problem**
  - Simple greedy heuristic can get stuck in a local minimum
- **Kernighan-Lin algorithm**
  - As long as a better partition is found
    - From all possible pairs of objects, virtually re-group the “best” (lowest cost of the resulting partition)
    - From the remaining not yet touched objects, virtually re-group the “best” pair, etc.,
    - Continue until all objects have been re-grouped
    - From these *n/2* partitions take the one with smallest cost and actually perform the corresponding re-group operations. $O(n^2 \log n)$ complexity
  - Still can get stuck in local minimum
    - Among sequences of moves
- ➢ **More complex strategies**
  - Randomize search, e.g. simulated annealing

Lecture 6: Outline

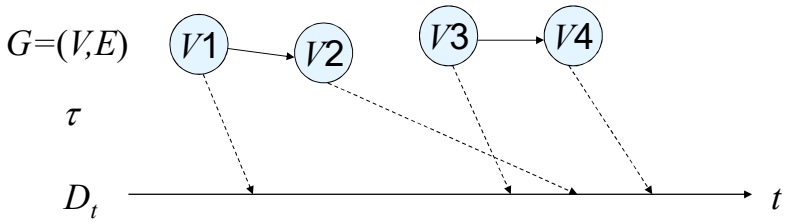
- ✓ HW/SW co-design tasks
  - ✓ System mapping
- ✓ Partitioning
  - ✓ Constructive heuristics
  - ✓ Iterative heuristics
- **Scheduling**
  - Uni-processor scheduling
  - Task graph scheduling
- System-level design
  - MPSoC synthesis

Multiplexing HW/SW Modules



Scheduling

- **A schedule is a mapping of a set of tasks to start times such that none overlap**
  - Optimize throughput, latency (schedule length/makespan)
  - Dependency, real-time (deadline) constraints



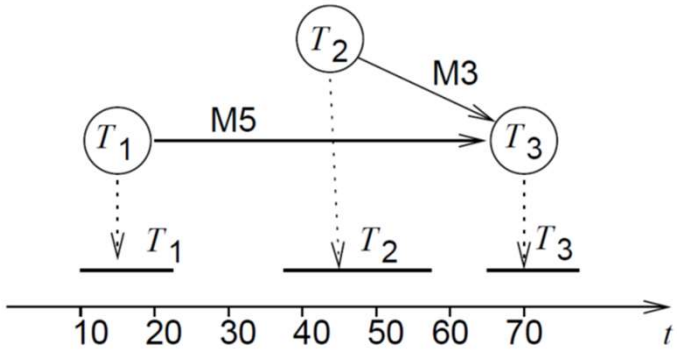
- **Static scheduling**
  - Fixed start times or fixed order (semi-static)
- **Dynamic scheduling**
  - Under control of an operating/runtime system

Uni-Processor Scheduling

- **Trivial without deadlines**
  - Makespan is constant, throughput = 1 / makespan
- **Aperiodic, independent task set**
  - Earliest Due Date (EDD) to minimize max. lateness
- **Periodic, independent task set**
  - Maximize CPU utilization while meeting deadlines
  - Rate Monotonic Scheduling (RMS) optimal w/ fixed priority
  - Earliest Deadline First (EDF) optimal w/ dynamic priority
- **Dependent task graph**
  - Latest Deadline First (LDF) to minimize max. lateness
  - Modified EDF* for dynamic schedule

Dependent Tasks

- **Task graph**

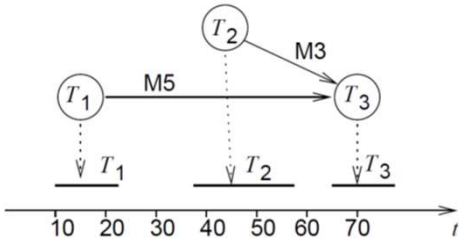


➢ **Periodic or aperiodic**
  - All tasks must have same period

Simultaneous Arrival Times

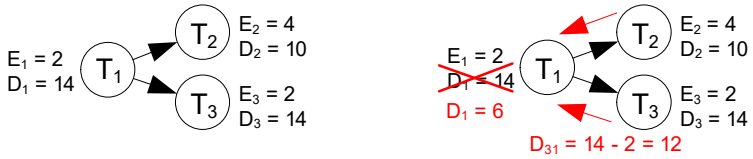
- **Latest Deadline First (LDF)**
  - Process task graph from sinks to sources
  - Among tasks without successors, insert the ones with the latest deadline into the schedule
  - At runtime, process in generated static reverse order

➢ **Optimal for uni-processor**
  - Non-preemptive
  - If no local deadlines, just topological sort



Different Arrival Times

- **Modified Earliest Deadline First (EDF*)**
  - Process graph from sinks to sources
  - Propagate deadlines adjusted for execution times
  - Under global time basis (adjusted for arrival times)
  - At each node, deadline = min(original,propagated)
  - Run from source to sinks in deadline order



- **Optimal for uni-processor**
  - Preemptive

Scheduling Considerations

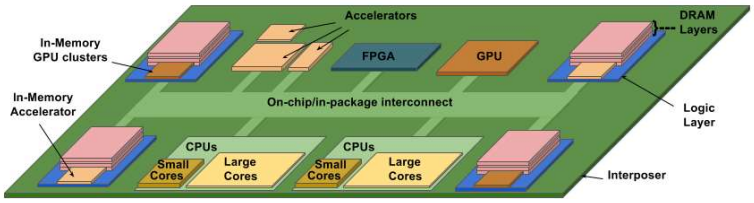
- **Special extensions**
  - HW accelerators (special task arrival/dependency models)
  - Task execution times (worst-case + dynamic scheduling)
  - Context switch overheads (fold into task execution times)
  - Interrupt overhead (treat as high-priority tasks)

- **What if your set of processes is unschedulable?**
  - Change deadlines in requirements
  - Reduce execution times of processes
  - Get a faster CPU
  - Change the HW/SW partitioning!

Lecture 6: Outline

- ✓ HW/SW co-design tasks
  - ✓ System mapping
- ✓ Partitioning
  - ✓ Constructive heuristics
  - ✓ Iterative heuristics
- ✓ Scheduling
  - Uni-processor scheduling
  - Task graph scheduling
- **System-level design**
  - MPSoC synthesis

Multi-Processor SoC (MPSoC)



- **Heterogeneous, accelerator-rich SoC architectures**
  - Heterogeneous programmable processors
    - Heterogenous CPUs, DSPs, GPUs
  - Large number of accelerators
    - ML/AI, audio, video, …
  - Custom memories
    - In-package DRAM, in-/near-memory computing
  - Heterogeneous interconnect
    - Networks-on-chip (NoCs), local bus hierarchies, interposers
  - I/O

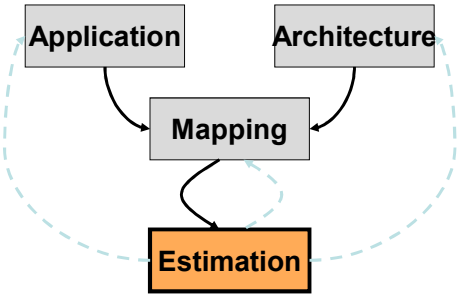
Multi-Processor Mapping

- **Partitioning & scheduling strongly interact**
  - Exploit parallelism
  - Optimize and trade off throughput and latency
  - NP-complete in general
- **Multi-processor SoC (MPSoC) mapping heuristics**
  - For simple cases, partition first and schedule separately
  - In general, solve partitioning & scheduling jointly

➢ **Heuristics from high-level synthesis (see later lectures)**
  ➢ Static scheduling
➢ **MPSoC mapping heuristics**
  ➢ Dynamic platform effects

MPSoC Synthesis

- **Design space exploration (DSE)**
  - Parallel application models & multi-processor architectures
  - Multi-objective, Pareto optimality
  ➢ Traditional HW/SW co-design approaches not sufficient
- **Iterative heuristics**
  - Determine mapping
    - Partitioning & scheduling
  - Evaluate solutions
    - Virtual platforms
- **DSE approaches**
  - Simulated annealing
  - Evolutionary algorithms
  - Reinforcement learning



➢ **ECE382N.23: Embedded System Design & Modeling**

Source: Lothar Thiele, ETH Zürich