Goals for lecture

- Resource representations
- Graph extensions for pre/post-computation and streaming/pipelining
- Scheduling problem categories
- Overview of scheduling algorithms
  - Will initially focus on static scheduling
- Sensor networks

Communication resource description

- Can use bus-bridge based models for distributed systems
  - Some protocols make static analysis difficult
- Wireless models
- System-level design, especially for a single chip, depends on wire delays!

Graph extensions

a) conventional

b) pre- and post-computation

c) streaming

Allows pipelining and pre/post-computation
In contrast with book, not difficult to use if conversion automated

Problem definition

minimize completion time

Given a set of tasks,
a cost function,
and a set of resources,
decide the exact time each task will execute on each resource

Types of scheduling problems

- Discrete time – Continuous time
- Hard deadline – Soft deadline
- Unconstrained resources – Constrained resources
- Uni-processor – Multi-processor
- Homogeneous processors – Heterogeneous processors
- Free communication – Expensive communication
- Independent tasks – Precedence constraints
- Homogeneous tasks – Heterogeneous tasks
- One-shot – Periodic
- Single rate – Multirate
- Non-preemptive – Preemptive
- Off-line – On-line
Discrete vs. continuous timing

System-level: Continuous
- Operations are not small integer multiples of the clock cycle

High-level: Discrete
- Operations are small integer multiples of the clock cycle

Implications:
- System-level scheduling is more complicated...
- ...however, high-level also very difficult.
- Can we solve this by quantizing time? Why or why not?

Real-time – Best effort

- Why make decisions about system implementation statically?
  - Allows easy timing analysis, hard real-time guarantees
- If a system doesn’t have hard real-time deadlines, resources can be more efficiently used by making late, dynamic decisions
- Can combine real-time and best-effort portions within the same specification
  - Reserve time slots
  - Take advantage of slack when tasks complete sooner than their worst-case finish times

Uni-processor – Multi-processor

- Uni-processor
  - All tasks execute on the same resource
  - This can still be somewhat challenging
  - However, sometimes in P
- Multi-processor
  - There are multiple resources to which tasks may be scheduled
  - Usually NP-complete

Free – Expensive communication

- Free communication
  - Data transmission between resources has no time cost
- Expensive communication
  - Data transmission takes time
  - Increases problem complexity
  - Generation of schedules for communication resources necessary
  - Usually NP-complete

Hard deadline – Soft deadline

Tasks may have hard or soft deadlines
- Hard deadline
  - Task must finish by given time or schedule invalid
- Soft deadline
  - If task finishes after given time, schedule cost increased

Unconstrained – Constrained resources

- Unconstrained resources
  - Additional resources may be used at will
- Constrained resources
  - Limited number of devices may be used to execute tasks

Homogeneous – Heterogeneous processors

- Homogeneous processors
  - All processors are the same type
- Heterogeneous processors
  - There are different types of processors
  - Usually NP-complete

Independent tasks – Precedence constraints

- Independent tasks: No previous execution sequence imposed
- Precedence constraints: Weak order on task execution order
Homogeneous – Heterogeneous tasks

- Homogeneous tasks: All tasks are identical
- Heterogeneous tasks: Tasks differ

One-shot – Periodic

- One-shot: Assume that the task set executes once
- Periodic: Ensure that the task set can repeatedly execute at some period

Single rate – Multirate

- Single rate: All tasks have the same period
- Multirate: Different tasks have different periods
  - Complicates scheduling
  - Can copy out to the least common multiple of the periods (hyperperiod)

Periodic graphs

- Periodic applications
  - Power electronics
  - Transportation applications
    - Engine controllers
    - Brake controllers
  - Many multimedia applications
    - Video frame rate
    - Audio sample rate
  - Many digital signal processing (DSP) applications
However, devices which react to unpredictable external stimuli have aperiodic behavior
Many applications contain periodic and aperiodic components

Aperiodic/sporadic graphs

- No precise periods imposed on task execution
- Useful for representing reactive systems
- Difficult to guarantee hard deadlines in such systems
  - Possible if minimum inter-arrival time known

Aperiodic to periodic

Can design periodic specifications that meet requirements posed by aperiodic/sporadic specifications
- Some resources will be wasted

Example:
- At most one aperiodic task can arrive every 50 ms
- It must complete execution within 100 ms of its arrival time

Periodic vs. aperiodic

Aperiodic to periodic

- Can easily build a periodic representation with a deadline and period of 50 ms
  - Problem, requires a 50 ms execution time when 100 ms should be sufficient
- Can use overlapping graphs to allow an increase in execution time
  - Parallelism required

The main problem with representing aperiodic problems with periodic representations is that the tradeoff between deadline and period must be made at design/synthesis time
Non-preemptive – Preemptive

- Non-preemptive: Tasks must run to completion
- Ideal preemptive: Tasks can be interrupted without cost
- Non-ideal preemptive: Tasks can be interrupted with cost

Hardware-software co-synthesis scheduling

- Automatic allocation, assignment, and scheduling of system-level specification to hardware and software
- Scheduling problem is hard
  - Hard and soft deadlines
  - Constrained resources, but resources unknown (cost functions)
  - Multi-processor
  - Strongly heterogeneous processors and tasks
    - No linear relationship between the execution times of a tasks on processors

Behavioral synthesis scheduling

- Difficult real-world scheduling problem
  - Not multirate
  - Discrete notion of time
  - Generally less heterogeneity among resources and tasks
- What scheduling algorithms should be used for these problems?

Clock-driven scheduling

- Clock-driven: Pre-schedule, repeat schedule
  - Periodic
  - Multi-rate
  - Heterogeneous
  - Off-line
  - Clock-driven

Off-line – On-line

- Off-line
  - Schedule generated before system execution
  - Stored, e.g., in dispatch table, for later use
  - Allows strong design/synthesis/compile-time guarantees to be made
  - Not well-suited to strongly reactive systems
- On-line
  - Scheduling decisions made during the execution of the system
  - More difficult to analyze than off-line
    - Making hard deadline guarantees requires high idle time
    - No known guarantee for some problem types
  - Well-suited to reactive systems

Hardware-software co-synthesis scheduling

- Expensive communication
  - Complicated set of communication resources
- Precedence constraints
- Periodic
- Multirate
- Strong interaction between NP-complete allocation-assignment and NP-complete scheduling problems
- Will revisit problem later in course if time permits

Scheduling methods

- Clock
- Weighted round-robin
- List scheduling
- Priority
  - EDF, LST
  - Slack
  - RMS
  - Multiple costs
- MILP
- Force-directed

Weighted round robbin

- Weighted round-robin: Time-sliced with variable time slots
List scheduling

- Pseudo-code:
  - Keep a list of ready jobs
  - Order by priority metric
  - Schedule
  - Repeat
- Simple to implement
- Can be made very fast
- Difficult to beat quality

Priority-driven

- Impose linear order based on priority metric
- Possible metrics
  - Earliest start time (EST)
  - Latest start time
    - Danger! LST also stands for least slack time.
  - Shortest execution time first (SETF)
  - Longest execution time first (LETF)
  - Slack (LFT - EFT)

List scheduling

- Assigns priorities to nodes
- Sequentially schedules them in order of priority
- Usually very fast
- Can be high-quality
- Prioritization metric is important

Prioritization

- As soon as possible (ASAP)
- As late as possible (ALAP)
- Slack-based
- Dynamic slack-based
- Multiple considerations

As soon as possible (ASAP)

- From root, topological sort on the precedence graph
- Propagate execution times, taking the max at reconverging paths
- Schedule in order of increasing earliest start time (EST)

As late as possible (ALAP)

- From deadlines, topological sort on the precedence graph
- Propagate execution times, taking the min at reconverging paths
- Consider precedence-constraint satisfied tasks
  - Schedule in order of increasing latest start time (LST)

Slack-based

- Compute EFT, LFT
- For all tasks, find the difference, LFT – EFT
- This is the slack
- Schedule precedence-constraint satisfied tasks in order of increasing slack
- Can recompute slack each step, expensive but higher-quality result
  - Dynamic critical path scheduling

Multiple considerations

- Nothing prevents multiple prioritization methods from being used
- Try one method, if it fails to produce an acceptable schedule, reschedule with another method
Effective release times

• Ignore the book on this
  – Considers simplified, uniprocessor, case
• Use EFT, LFT computation
• Example?

EDF, LST optimality

• EDF optimal if zero-cost preemption, uniprocessor assumed
  – Why?
  – What happens when preemption has cost?
• Same is true for slack-based list scheduling in absence of preemption cost

Breaking EDF, LST optimality

• Non-zero preemption cost
• Multiprocessor
• Why?

Rate monotonic scheduling (RMS)

• Single processor
• Independent tasks
• Differing arrival periods
• Schedule in order of increasing periods
• No fixed-priority schedule will do better than RMS
• Guaranteed valid for loading $\leq \ln 2 = 0.69$
• For loading $> \ln 2$ and $< 1$, correctness unknown
• Usually works up to a loading of 0.88
• More detail in later lectures

Reading assignment

• Finish Chapter 5, read Chapter 6 by Thursday