Introduction to Real-Time Systems

ECE 397-1

Northwestern University

Department of Computer Science
Department of Electrical and Computer Engineering

Teachers:	Robert Dick	Peter Dinda
Office:	L477 Tech	338, 1890 Maple Ave.
Email:	dickrp@ece.northwestern.edu	pdinda@cs.northwestern.edu
Phone:	467–2298	467-7859
Webpage:	http://www.ece.northwestern.edu/EXTERNAL/realtime	

Homework index

1 Reading assignment (for next class) 56
2 Hanford security network design 66
3 Reading assignment (18 January) 108
4 Reading assignment 151
5 Bizarre scheduling idea 176
6 Reading assignment 177
7 Reading assignment 204
8 Lab six 207

Topics list: Real-time networking

- Chapter 11, Tenet Paper, K&R chapter 7
- Workload models – describing burstiness
  - Leaky Bucket
  - Ferarri
  - Why we can't just do "average bandwidth"
- How does a queue deal with burstiness? What are the consequences for latency
- Weighted fair queuing (WFQ)

Topics list: Real-time networking

- How to combine WFQ and Leaky Bucket to estimate the queuing delay at a node and thus to do admission control for it.
- End-to-end admission control and reservations
- Why it is difficult to make per-flow real-time behavior scale
- RTP - why should we care if there is no guarantee
- RSVP
- Diffserve versus Intserve
- Overlay networks

Media networking

- K&R Chapter 7
- What buffering does to latency and why/when we might want to use it anyway
- Workloads of media (ie, self-similarity issue) and how buffering can be of less help than expected.
- Why is the workload so complex? Scene dynamics and compression
- RT queuing theory (read the Lehokzy paper)

Distributed real-time systems

- Ramamritham, Bestavros, Schmidt, Quorum
- Scaling behavior - job sizes, deadlines, and transmission times scale as the system scales
- Initial placement versus migration
- Scheduling all of the workload versus just a part of it
- Having full control over local schedulers versus not.

Distributed real-time systems

- Structures of RT systems
  - single node (master) with global admission control, multiple backend servers
  - peer nodes with local admission control
  - scaling versus being able to admit all admittable tasks
  - bidding versus focused addressing
  - work stealing

Distributed real-time systems

- Parallel jobs
  - fork-join task graphs and their implications
  - Cluster scheduling
  - space sharing versus gang scheduling versus synchronized periodic real-time schedules

Real-time adaptive systems

- Dinda, Noble, Mitzenmacher
- Power-of-two-choices
- Workload prediction
  - Predicting job sizes and arrivals
  - Predicting queue depth
- Scheduler modeling

Real-time adaptive systems

- Adaptation mechanisms
  - job placement and migration
  - job selection (which function to call)
  - quality modulation
  - network path selection

Real-time adaptive systems

- Application goals / QoS
  - minimize response time, maximize throughput
  - deadlines
  - QoS parameters (frame rate, frame latency, etc)
  - utility functions
- Control problem
- Event-driven simulators

Lecture packet one

- Taxonomy of real-time systems
- Graph definitions
- Graph algorithms
- Timing constraints
- Cost functions
- Jagged edges in real-time problem categorization
- Allocation, assignment, and scheduling
- Real-Time Operating systems
- Distributed systems
- Formal problem definitions: Optimization

Lecture packet two

- Example optimization problem
- Crash course in computational complexity (why?)
- Design representations: SW-oriented, HW-oriented, graph-based
- Introduction to NesC

Lecture packets three and four

- Processors
- Communication resources
- Graph extensions
- Taxonomy of scheduling problems
- Example real scheduling problems
- Scheduling methods
- Scheduling examples

Lecture packet five *

- Rate monotonic scheduling
- Critical instants and utilization bounds
- Threads and processes
- Example scheduler implementations

Lecture packets six and seven *

- Recent work in RTOS performance/power analysis
- Recent solution to off-line hard real-time allocation/assignment/scheduling problem
- Implicit vs. explicit representation of time in formal methods

Goals for lecture

- Handle a few administrative details
- Form lab groups
- Broad overview of real-time systems
- Definitions that will come in handy later
- Example of real-time sensor network

Administrative tasks

- Backgrounds
- Question rule
- Office hours

Backgrounds

- Lab teams had best be balanced (low-level vs. high-level experience)
- Name
- Which are you better at?
  - Low-level ANSI-C/assembly experience
  - High-level object-oriented programming experience
- What's your major?

Question rule

- If something in lecture doesn't make sense, please ask
- You're paying a huge amount of money for this
- Letting something important from lecture slip by for want of a question is like burning handfulls of money

Core course goal

By the end of this course, we want you to
learn how to build real-time systems
and build a useful real-time sensor network.

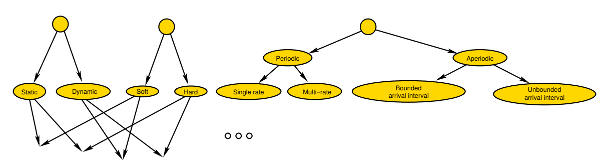
Office hours

- When shall I schedule my office hours?

Today's topics

- Taxonomy of real-time systems
- Optimization and costs
- Definitions
- Optimization formulation
- Overview of primary areas of study within real-time systems

Taxonomy of real-time systems



Taxonomy: Static

- Task arrival times can be predicted.
- Static (compile-time) analysis possible.
- Allows good resource usage (low processor idle time proportions).
- Sometimes designers shoehorn dynamic problems into static formulations allowing a good solution to the wrong problem.

Taxonomy: Dynamic

- Task arrival times unpredictable.
- Static (compile-time) analysis possible only for simple cases.
- Even then, the portion of required processor utilization efficiency goes to 0.693.
- In many real systems, this is very difficult to apply in reality (more on this later).
- Use the right tools but don't over-simplify, e.g.,

  We assume, without loss of generality, that all tasks are independent.

  If you do this people will make jokes about you.

Taxonomy: Soft real-time

- More slack in implementation
- Timing may be suboptimal without being incorrect
- Problem formulation can be much more complicated than hard real-time
- Two common (and one uncommon) methods of dealing with non-trivial soft real-time system requirements
  - Set somewhat loose hard timing constraints
  - Informal design and testing
  - Formulate as optimization problem

Taxonomy: Hard real-time

- Difficult problem. Some timing constraints inflexible.
- Simplifies problem formulation.

Taxonomy: Periodic

- Each task (or group of tasks) executes repeatedly with a particular period.
- Allows some nice static analysis techniques to be used.
- Matches characteristics of many real problems...
- ... and has little or no relationship with many others that designers try to pretend are periodic.

Taxonomy: Periodic $\longrightarrow$ Single-rate

- One period in the system.
- Simple.
- Inflexible.
- This is how a *lot* of wireless sensor networks are implemented.

Taxonomy: Periodic $\longrightarrow$ Multirate

- Multiple periods.
- Can use notion of circular time to simplify static (compile-time) schedule analysis E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations Research*, pp. 699–719, July 1966.
- Co-prime periods leads to analysis problems.

Taxonomy: Periodic $\longrightarrow$ Other

- It is possible to have tasks with deadlines less than, equal to, or greater than their periods.
- Results in multi-phase, circular-time schedules with multiple concurrent task instances.
  - If you ever need to deal with one of these, see me (take my code). This class of scheduler is nasty to code.

Taxonomy: Aperiodic

- Also called sporadic, asynchronous, or reactive
- Implies dynamic
- Bounded arrival time interval permits resource reservation
- Unbounded arrival time interval impossible to deal with for any resource-constrained system

Definitions

- Task
- Processor
- Graph representations
- Deadline violation
- Cost functions

Definitions: Task

- Some operation that needs to be carried out
- Atomic completion: A task is all done or it isn't
- Non-atomic execution: A task may be interrupted and resumed

Definitions: Processor

- Processors execute tasks
- Distributed systems
  - Contain multiple processors
  - Inter-processor communication has impact on system performance
  - Communication is challenging to analyze
- One processor type: Homogeneous system
- Multiple processor types: Heterogeneous system

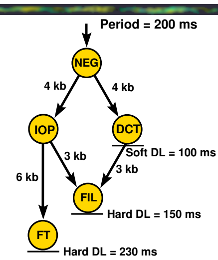
Task/processor relationship

WC exec time (s)

	IBM PowerPC 405GP 266 MHz	IDT79RC32364 100 MHz	Imsys Cjip 40 MHz
Tooth	7.7E−6	···	
Road	330E−9	···	
FIR	4.1E−6	···	
Matrix	310E−3	···	

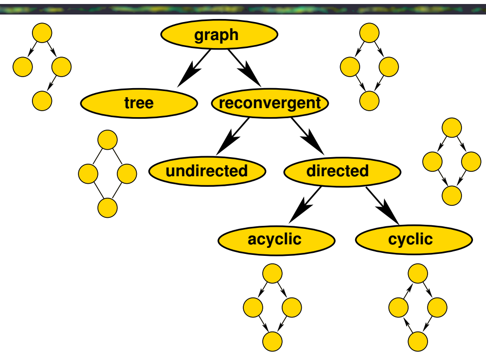
Relationship between tasks, processors, and costs
E.g., power consumption or worst-case execution time

Graph definitions



- Set of vertices (V)– usually operations
- Set of edges (E)– directed or undirected relationships on vertex pairs

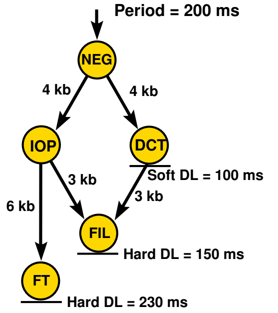
Example graph classifications



Some graph uses

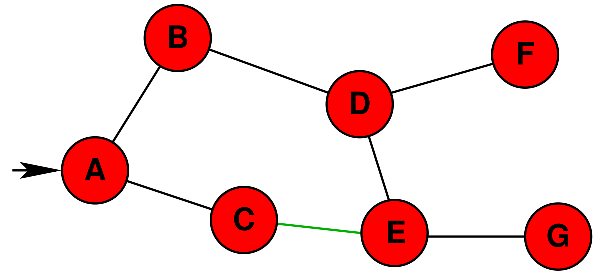
- Problem representations
- Timing constraint specification
- Resource binding
- And many more...

A few basic graph algorithms



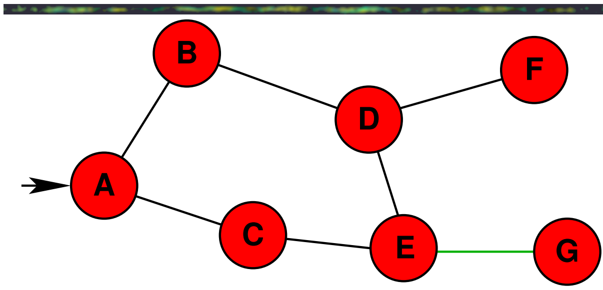
- Depth-first search (DFS)
- Breadth-first search (BFS)
- Topological sort
- Minimal spanning tree (MST)

Depth-first search (DFS) – Pre-order for trees



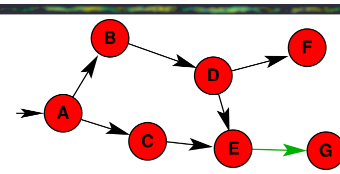
$\mathcal{O}(|V|+|E|)$

Breadth-first search (BFS) – Pre-order for trees



$\mathcal{O}(|V|)$

Topological sort

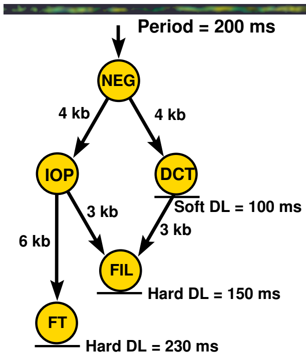


Static timing analysis of data-dependent real-time systems

- Earliest finish time (EFT)
- Earliest start time (EST)
- Latest finish time (LFT)
- Latest start time (LST)

$\mathcal{O}(|V|+|E|)$

Definition: Deadline violation



Cost functions

- Mapping of real-time system design problem solution instance to cost value
- I.e., allows price, or hard deadline violation, of a particular multi-processor implementation to be determined

Back to real-time problem taxonomy: Jagged edges

- Some things dramatically complicate real-time scheduling
- These are horrific, especially when combined
  - Data dependencies
  - Unpredictability
  - Distributed systems
- These are irksome
  - Heterogeneous processors
  - Preemption

Central areas of real-time study

- Allocation, assignment and **scheduling**
- Operating systems and **scheduling**
- Distributed systems and **scheduling**
- **Scheduling is at the core or real-time systems study**

Allocation, assignment and scheduling

How does one best

- Analyze problem instance specifications
  - E.g., worst-case task execution time
- Select (and build) hardware components
- Select and produce software
- Decide which processor will be used for each task
- Determine the time(s) at which all tasks will execute

Allocation, assignment and scheduling

- In order to efficiently and (when possible) optimally minimize
  - Price, power consumption, soft deadline violations
- Under hard timing constraints
- Providing guarantees whenever possible
- For all the different classes of real-time problem classes

This is what I did for a Ph.D.

Operating systems and scheduling

How does one best design operating systems to

- Support sufficient detail in workload specification to allow good control, e.g., over scheduling, without increasing design error rate
- Design operating system schedulers to support real-time constraints?
- Support predictable costs for task and OS service execution

Distributed systems and scheduling

How does one best dynamically control

- The assignment of tasks to processing nodes...
- ... and their schedules

for systems in which computation nodes may be separated by vast distances such that

- Task deadline violations are bounded (when possible)...
- ... and minimized when no bounds are possible

This is part of what Professor Dinda did for a Ph.D.

The value of formality: Optimization and costs

- The design of a real-time system is fundamentally a cost optimization problem
- Minimize costs under constraints while meeting functionality requirements
  - Slight abuse of notation here, functionality requirements are actually just constraints
- Why view problem in this manner?
- Without having a concrete definition of the problem
  - How is one to know if an answer is correct?
  - More subtly, how is one to know if an answer is optimal?

Optimization

Thinking of a design problem in terms of optimization gives design team members objective criterion by which to evaluate the impact of a design change on quality.

- Still need to do a lot of hacking
- Know whether its taking you in a good direction

Summary

- Real-time systems taxonomy and overview
- Definitions
- Importance of problem formulation

Reading assignment (for next class)

- J. W. S. Liu, *Real-Time Systems.* Prentice-Hall, Englewood Cliffs, NJ, 2000
- Chapter 2
- Start on Chapter 3

Goals for lecture

- Justify treating real-time design problem as optimization problem
- Example problem to illustrate specification and design
- Tractable algorithm design (NP-completeness in a nutshell)
- Detail on design representations
- Sensor network motivations
- NesC overview

The value of formality: Optimization and costs

- The design of a real-time system is fundamentally a cost optimization problem
- Minimize costs under constraints while meeting functionality requirements
  - Slight abuse of notation here, functionality requirements are actually just constraints
- Why view problem in this manner?
- Without having a concrete definition of the problem
  - How is one to know if an answer is correct?
  - More subtly, how is one to know if an answer is optimal?

Optimization

Thinking of a design problem in terms of optimization gives design team members objective criterion by which to evaluate the impact of a design change on quality.

- Still need to do a lot of hacking
- Know whether its taking you in a good direction

Simple example

- Ensure that a wireless data display 300 m away from a temperature sensor always displays the correct temperature with a lag of, at most, 100 ms.
- Wireless broadcasts reach 100 m with high probability and 200 m with very low probability.
- There are two, evenly distributed, rebroadcast nodes between the sensor and the data display.
- Functional requirements?
- Constraints?
- Costs?

Example problem

- Richland, Washington's Hanford Reservation plutonium finishing facility
- July 1988 facility's last reactor, Reactor N, put into cold standby due the nation's surplus of plutonium
- Was used for processing weapons-grade fissile material

Example problem

- Currently holds 11.0 metric tons of plutonium-239 and 0.6 metric tons of uranium-235
  - The two fissile materials most commonly used in nuclear weapons
- Even without refining, a small quantity of either would convert conventional explosives into weapons capable of causing long-term damage far beyond their blast radii
- Ongoing provisions for security required

Example problem

- Build perimeter security network
- Functional requirements?
- Constraints?
- Costs?

Example tasks

- Sense audio
- Compress it
- Determine whether it is unusual
- Sense, compress, and stream video
- Analyze information from region to determine most promising messages to forward, given network contention

Example constraints

- Data rate
- Dependencies between tasks
- Price
- Lifetime of battery-powered devices
- Etc.

Hanford security network design

- By 18 January, working with your lab partner, provide
  - A paragraph formalizing the real-time system design goals
  - A paragraph giving an overview of the design you propose
- Keep it within a page. We want you thinking about this and learning but you should focus on the lab assignment.
- Have questions? Do research. The Hanford Reservation is real.
  - Post to the newsgroup if you get stuck.

Lab one

- Subversion working for everybody?
- Access to mailing list?
- Anybody stuck on development?

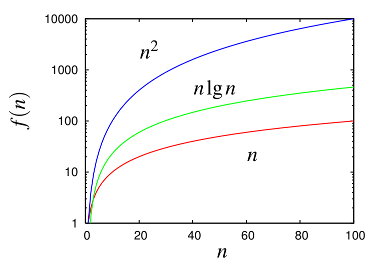
NP-completeness

- Scheduling is central to real-time systems design and research
- Tractable algorithm design is central to scheduling
- Many (but not all) interesting and useful scheduling problems are NP-complete
- We need to understand what this means, at least at a high level

NP-completeness

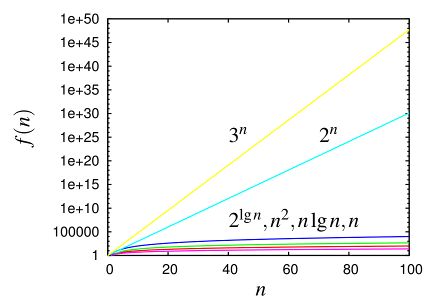
Recall that sorting may be done in $\mathcal{O}(n \lg n)$ time

DFS $\in \mathcal{O}(|V|+|E|)$, BFS $\in \mathcal{O}(|V|)$, Topological sort $\in \mathcal{O}(|V|+|E|)$



NP-completeness

There also exist exponential-time algorithms: $\mathcal{O}\left(2^{\lg n}\right)$, $\mathcal{O}(2^n)$, $\mathcal{O}(3^n)$



NP-completeness

For $t(n) = 2^n$ seconds

$t(1) = 2$ seconds

$t(10) = 17$ minutes

$t(20) = 12$ days

$t(50) = 35,702,052$ years

$t(100) = 40,196,936,841,331,500,000,000$ years

NP-completeness

- There is a class of problems, $\mathbf{NP\text{-}complete}$, for which nobody has found polynomial time solutions
- It is possible to convert between these problems in polynomial time
- Thus, if it is possible to solve any problem in $\mathbf{NP\text{-}complete}$ in polynomial time, all can be solved in polynomial time
- Unproven conjecture: $\mathbf{NP} \neq \mathbf{P}$

NP-completeness

- What is $\mathbf{NP}$? Nondeterministic polynomial time.
- A computer that can simultaneously follow multiple paths in a solution space exploration tree is nondeterministic. Such a computer can solve $\mathbf{NP}$ problems in polynomial time.
- Nobody has been able to prove either

$$\mathbf{P} \neq \mathbf{NP}$$

or

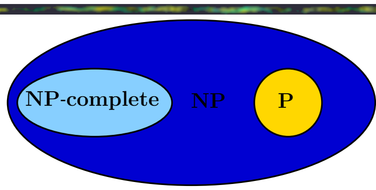
$$\mathbf{P} = \mathbf{NP}$$

NP-completeness

If we define $\mathbf{NP\text{-}complete}$ to be a set of problems in $\mathbf{NP}$ for which any problem's instance may be converted to an instance of another problem in $\mathbf{NP\text{-}complete}$ in polynomial time, then

$$\mathbf{P} \subsetneq \mathbf{NP} \Rightarrow \mathbf{NP\text{-}complete} \cap \mathbf{P} = \varnothing$$

Basic complexity classes



- $\mathbf{P}$ solvable in polynomial time by a computer (Turing Machine)
- $\mathbf{NP}$ solvable in polynomial time by a nondeterministic computer
- $\mathbf{NP\text{-}complete}$ converted to other $\mathbf{NP\text{-}complete}$ problems in polynomial time

Hard ($\mathbf{NP\text{-}complete}$) scheduling problems

- Uniprocessor scheduling with hard deadlines and release times
- Uniprocessor scheduling to minimize tardy tasks
- Multiprocessor scheduling
  - Easy if all tasks are identical
- Multiprocessor precedence constrained scheduling
- Multiprocessor preemptive scheduling
- etc.

How to deal with hard problems

- What should you do when you encounter an apparently hard problem?
- Is it in $\mathbf{NP\text{-}complete}$?
- If not, solve it
- If so, then what?

Determine whether all encountered problem instances are constrained.
Wonderful when it works.

One example

O. Coudert, "Exact coloring of real-life graphs is easy," *Design Automation*, pp. 121–126, June 1997.

Terminology

- Book's terminology fine, others also exist
- Different groups $\longrightarrow$ different terminology
- Not confusing, terse definitions provided
- Book on jobs, tasks: Jobs discrete, tasks groups of related jobs
- Other sources: Tasks discrete, hierarchical

Additional terminology

- Or vs. And data dependencies
- Conditionals
  - Doesn't help hard real-time unless perfect path correlation
  - Can help soft real-time

Terminology

- Scheduling, allocation, and assignment
- Scheduling central but not only thing
- Book treats scheduling as combination of scheduling and assignment
- More fine-grained definitions exist

Substantial quirks

1. Every processor is assigned to at most one job at any time
   - O.K.
2. Every job is assigned at most one processor at any time
   - Broken
3. No job scheduled before its release time
   - O.K., but the whole notion of absolute release times is broken for some useful classes of real-time systems.
4. Etc.

Design representations

- Introduction
- Software oriented
- Hardware oriented
- Graph based
- Resource description

Specification language requirements

- Specify constraints on design
- Indicate system-level building blocks
- To allow flexibility in compilation/synthesis, must be abstract
  - Specify implementation details only when necessary (e.g., HW/SW)
  - Concentrate on requirements, not implementation
  - Make few assumptions about platform

Design representations

- Introduction
- Software oriented
  - ANSI-C
  - SystemC
  - Other SW language-based, e.g., Ada
- Hardware oriented
- Graph based
- Resource description

ANSI-C advantages

- Huge code base
- Many experienced programmers
- Efficient means of SW implementation
- Good compilers for many SW processors

ANSI-C disadvantages

- Little implementation flexibility
  - Strongly SW oriented
  - Makes many assumptions about platform
- Little (volatile)/no built-in support for synchronization
  - Especially fine-scale HW synchronization
- Doesn't directly support specification of timing constraints

SystemC

Advantages

- Support from big players
  - Synopsys, Cadence, ARM, Red Hat, Ericsson, Fujitsu, Infineon Technologies AG, Sony Corp., STMicroelectronics, and Texas Instruments
- Familiar for SW engineers

Disadvantages

- Extension of SW language
  - Not designed for HW from the start
- Compiler available for limited number of SW processors
  - New

Other SW language-based

- Numerous competitors
- Numerous languages
  - ANSI-C, C++, and Java are most popular starting points
- In the end, few can survive
- SystemC has broad support

Design representations

- Software oriented
- Hardware oriented
  - VHDL
  - Verilog
  - Esterel
- Graph based
- Resource description

VHDL

Advantages

- Supports abstract data types
- System-level modeling supported
- Better support for test harness design

Disadvantages

- Requires extensions to easily operate at the gate-level
- Difficult to learn
- Slow to code

Verilog

Advantages

- Easy to learn
- Easy for small designs

Disadvantages

- Not designed to handle large designs
- Not designed for system-level

Verilog vs. VHDL

- March 1995, Synopsys Users Group meeting
- Create a gate netlist for the fastest fully synchronous loadable 9-bit increment-by-3 decrement-by-5 up/down counter that generated even parity, carry and borrow
- 5 / 9 Verilog users completed
- 0 / 5 VHDL users competed

Does this mean that Verilog is better?

Maybe, but maybe it only means that Verilog is easier to use for simple designs.

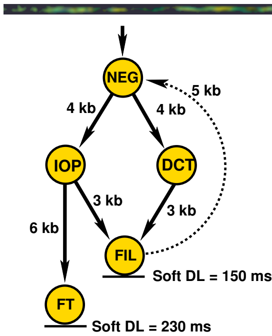
Esterel

- Easily allows synchronization among parallel tasks
- Works at a high level of abstraction
  - Doesn't require explicit enumeration of all states and transitions
- Recently extended for specifying datapaths and flexible clocking schemes
- Amenible to theorem proving
- Translation to RTL or C possible
- Commercialized by Esterel Technologies

Design representations

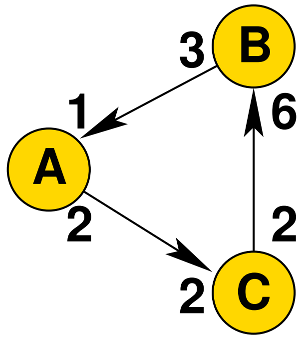
- Software oriented
- Hardware oriented
- Graph based
  - Dataflow graph (DFG)
  - Synchronous dataflow graph (SDFG)
  - Control flow graph (CFG)
  - Control dataflow graph (CDFG)
  - Finite state machine (FSM)
  - Petri net
  - Periodic vs. aperiodic
  - Real-time vs. best effort
  - Discrete vs. continuous timing
  - Example from research
- Resource description

Dataflow graph (DFG)

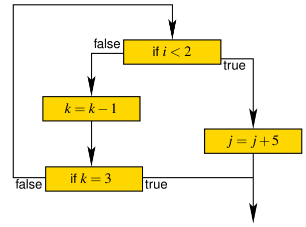


- Nodes are tasks
- Edges are data dependencies
- Edges have communication quantities
- Used for digital signal processing (DSP)
- Often acyclic when real-time
- Can be cyclic when best-effort

Synchronous dataflow graph (SDFG)

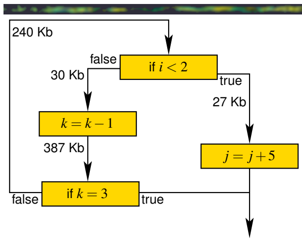


Control flow graph (CFG)



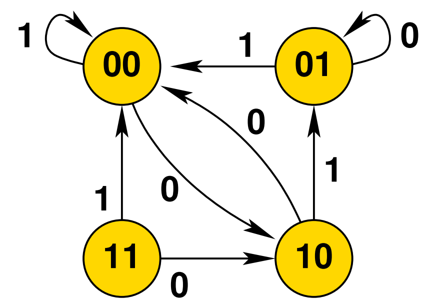
- Nodes are tasks
- Supports conditionals, loops
- No communication quantities
- SW background
- Often cyclic

Control dataflow graph (CDFG)



- Supports conditionals, loops
- Supports communication quantities
- Used by some high-level synthesis algorithms

Finite state machine (FSM)



Finite state machine (FSM)

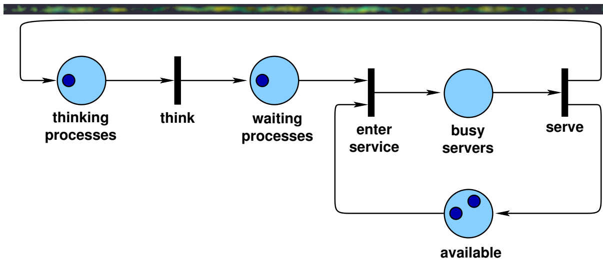
	input	
	0	1
00	10	00
01	01	00
10	00	01
11	10	00
current	next	

- Normally used at lower levels
- Difficult to represent independent behavior
  - State explosion
- No built-in representation for data flow
  - Extensions have been proposed
- Extensions represent SW, e.g., co-design finite state machines (CFSMs)

Petri net

- Graph composed of places, transitions, and arcs
- Tokens are produced and consumed
- Useful model for asynchronous and stochastic processes
- Places can have priorities
- Not well-suited for representing dataflow systems
- Timing analysis quite difficult
- Large flat graphs difficult to understand
- Real-time use: Specification and formal timing verification

Petri net



M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann's token game demonstration.

NesC

- View as a ANSI C with additional layer
- Specify interfaces between components
- Centers on *commands* and *events*
- Commands
  - Provided by interface, do things
  - Non-blocking, split-phase (response from events)
  - Call down
  - E.g., transmit data

NesC

Events

- Provided by interface
- Used to signal command completion
- Interrupt tasks
- Require concurrency control (*atomic* blocks)

NesC

- Tasks: Interrupted only by events, no normal preemption
- Asynchronous code: can be reached by interrupt handlers
- Synchronous code: can be reached only from tasks
- Not the only option

Summary

- Justify treating real-time design problem as optimization problem
- Example problem to illustrate specification and design
- Tractable algorithm design (NP-completeness in a nutshell)
- Detail on design representations
- Sensor network motivations
- NesC overview

Reading assignment (18 January)

- M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company, NY, 1979.
  - Chapter 1
  - Chapter A5: Sequencing and scheduling
- J. W. S. Liu, *Real-Time Systems*. Prentice-Hall, Englewood Cliffs, NJ, 2000.
  - Chapter 3
  - Chapter 4

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

Processing resource description

- Often table-based
- Price, area
- For each task
  - Execution time
  - Power consumption
  - Preemption cost
  - etc.
- etc.

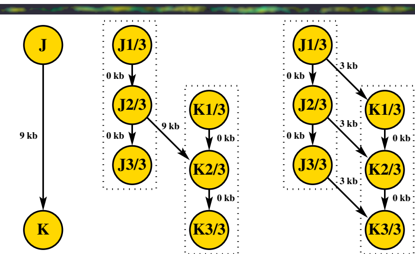
Similar characterization for communication resources

Wise to use process-based

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!

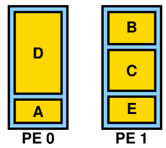
Graph extensions



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?

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

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

Unconstrained – Constrained resources

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

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

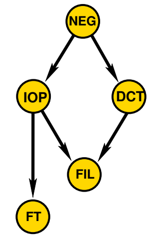
Homogeneous – Heterogeneous processors

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

Free – Expensive communication

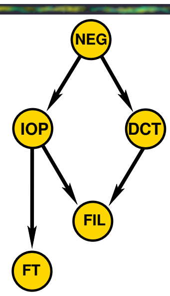
- Free communication
  - Data transmission between resources has no time cost
- Expensive communication
  - Data transmission takes time
  - Increases problem complexity
  - Generation of schedules for communication resources necessary
  - Usually $\mathbf{NP}$-complete

Independent tasks – Precedence constraints



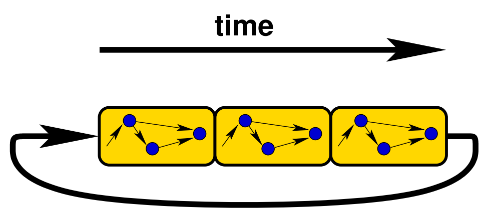
- Independent tasks: No previous execution sequence imposed
- Precedence constraints: Weak order on task execution order

Homogeneous – Heterogeneous tasks



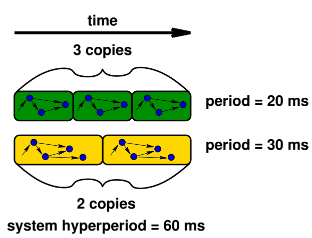
- Homogeneous tasks: All tasks are identical
- Heterogeneous tasks: Tasks differ

One-shot – Periodic



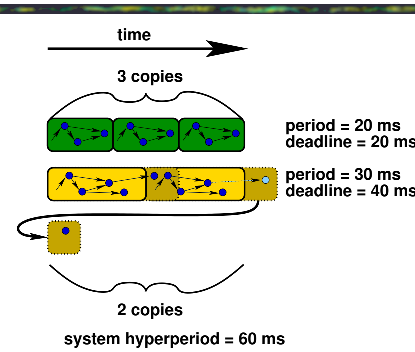
- One-shot: Assume that the task set executes once
- Periodic: Ensure that the task set can repeatedly execute at some period

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



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

Periodic vs. aperiodic

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 to periodic

Can design periodic specifications that meet requirements posed by aperiodic/sporadic specifications

- Some resources will be wasted

Example:

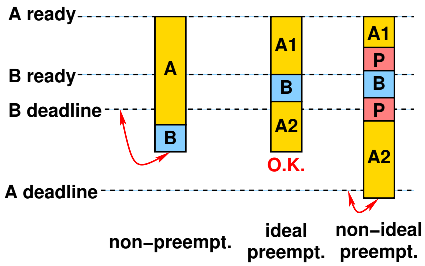
- At most one aperiodic task can arrive every 50 ms
- It must complete execution within 100 ms of its arrival time

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

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

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

Hardware-software co-synthesis scheduling

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

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?

Scheduling methods

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

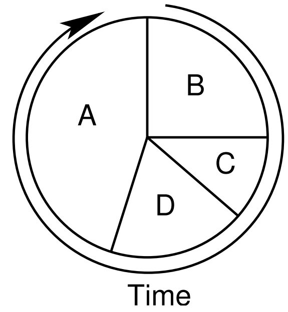
Clock-driven scheduling

Clock-driven: Pre-schedule, repeat schedule

Music box:

- Periodic
- Multi-rate
- Heterogeneous
- Off-line
- Clock-driven

Weighted round robbin



Weighted round-robbin: 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)

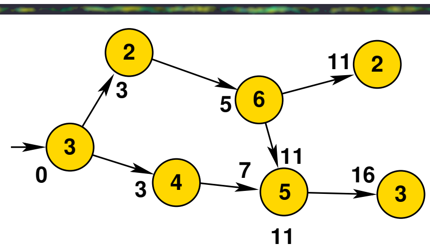
List scheduling

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

Prioritization

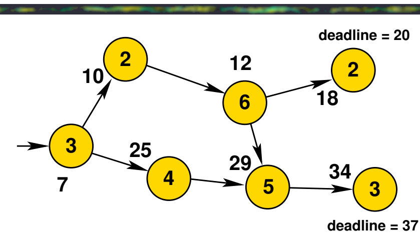
- As soon as possible (ASAP)
- As late as possible (ALAP)
- Slack-based
- Dynamic slack-based
- Multiple considerations

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 mononotic 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

- Skim and refer to K. Ramamritham and J. Stankovic, “Scheduling algorithms and operating systems support for real-time systems,” *Proc. IEEE*, vol. 82, pp. 55–67, Jan. 1994
- Skim and refer to Y.-K. Kwok and I. Ahmad, “Static scheduling algorithms for allocating directed task graphs to multiprocessors,” *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999
- J. W. S. Liu, *Real-Time Systems.* Prentice-Hall, Englewood Cliffs, NJ, 2000
- Finish Chapter 5, read Chapter 6 by Thursday

Goals for lecture

- Sensor networks
- Finish overview of scheduling algorithms
- Mixing off-line and on-line
- Design a scheduling algorithm: DCP
  - Will initially focus on static scheduling
- Useful properties of some off-line schedulers

Lab two?

- Everybody able to finish?
- Any problems to warn classmates about?
- 18 motes should be arriving tomorrow
  - No equipment sign-out required for next motes lab
- Linux vs. Windows development environments

Sensor networks

- Gather information over wide region
- Frequently no infrastructure
- Battery-powered, wireless common
- Battery lifespan of central concern

Low-power sensor networks

- Power consumption central concern in design
- Processor?
  - RISC μ-controllers common
- Wireless protocol?
  - Low data-rate, simple: Proprietary, Zigbee
- OS design?
  - Static, eliminate context switches, compile-time analysis

Low-power sensor networks

- Power consumption central concern in design
- Runtime environment?
  - Avoid unnecessary dynamism
- Language?
  - Compile-time analysis of everything practical

Multi-rate tricks

- Contract deadline
  - Usually safe
- Contract period
  - Sometimes safe
- Consequences?

Scheduling methods

- Clock
- Weighted round-robbin
- List scheduling
- Priority
  - EDF, LST
  - Slack
  - Multiple costs

Scheduling methods

- MILP
- Force-directed
- Frame-based
- PSGA

Linear programming

- Minimize a linear equation subject to linear constraints
  - In $\mathbf{P}$
- Mixed integer linear programming: One or more variables discrete
  - $\mathbf{NP\text{-}complete}$
- Many good solvers exist
- Don't rebuild the wheel

MILP scheduling

P the set of tasks

t_{max} maximum time

$start(p,t)$ 1 if task p starts at time t, 0 otherwise

D the set of execution delays

E the set of precedence constraints

$t_{start}(p) = \sum_{t=0}^{t_{max}} t \cdot start(p,t)$ the start time of p

MILP scheduling

Each task has a unique start time

$$\forall_{p \in P}, \sum_{t=0}^{t_{max}} start(p,t) = 1$$

Each task must satisfy its precedence constraints and timing delays

$$\forall \{p_i, p_j\} \in E, \sum_{t=0}^{t_{max}} t_{start}(p_i) \geq t_{start}(p_j) + d_j$$

Other constraints may exist

- Resource constraints
- Communication delay constraints

MILP scheduling

- Too slow for large instances of $\mathbf{NP\text{-}complete}$ scheduling problems
- Numerous optimization algorithms may be used for scheduling
- List scheduling is one popular solution
- Integrated solution to allocation/assignment/scheduling problem possible
- Performance problems exist for this technique

Force directed scheduling

- P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, pp. 661–679, June 1989
- Calculate EST and LST of each node
- Determine the force on each vertex at each time-step
- Force: Increase in probabilistic concurrency
  - Self force
  - Predecessor force
  - Successor force

Self force

F_i all slots in time frame for i

F_i' all slots in new time frame for i

D_t probability density (sum) for slot t

δD_t change in density (sum) for slot t resulting from scheduling

self force

$$A = \sum_{t \in F_a} D_t \cdot \delta D_t$$

Predecessor and successor forces

pred all predecessors of node under consideration

succ all successors of node under consideration

predecessor force

$$B = \sum_{b \in \textbf{pred}} \sum_{t \in F_b} D_t \cdot \delta D_t$$

successor force

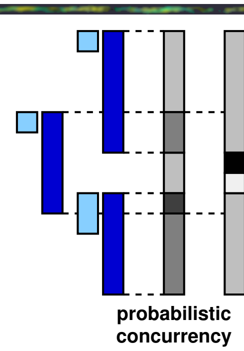
$$C = \sum_{c \in \textbf{succ}} \sum_{t \in F_c} D_t \cdot \delta D_t$$

Intuition

total force: $A + B + C$

- Schedule operation and time slot with minimal total force
  - Then recompute forces and schedule the next operation
- Attempt to balance concurrency during scheduling

Force directed scheduling



Force directed scheduling

- Limitations?
- What classes of problems may this be used on?

Implementation: Frame-based scheduling

- Break schedule into (usually fixed) frames
- Large enough to hold a long job
  - Avoid preemption
- Evenly divide hyperperiod
- Scheduler makes changes at frame start
- Network flow formulation for frame-based scheduling
- Could this be used for on-line scheduling?

Problem space genetic algorithm

- Let's finish off-line scheduling algorithm examples on a bizarre example
- Use conventional scheduling algorithm
- Transform problem instance
- Solve
- Validate
- Evolve transformations

Examples: Mixing on-line and off-line

- Book mixes off-line and on-line with little warning
- Be careful, actually different problem domains
- However, can be used together
- Superloop (cyclic executive) with non-critical tasks
- Slack stealing
- Processor-based partitioning

Problem: Vehicle routing

- Low-price, slow, ARM-based system
- Long-term shortest path computation
- Greedy path calculation algorithm available, non-preemptable
- Don't make the user wait
  - Short-term next turn calculation
- 200 ms timer available

Examples: Mixing on-line and off-line

- Slack stealing
- Processor-based partitioning

Scheduling summary

- Scheduling is a huge area
- This lecture only introduced the problem and potential solutions
- Some scheduling problems are easy
- Most useful scheduling problems are hard
  - Committing to decisions makes problems hard: Lookahead required
  - Interdependence between tasks and processors makes problems hard
  - On-line scheduling next Tuesday

Bizarre scheduling idea

- Scheduling and validity checking algorithms considered so far operate in time domain
- This is a somewhat strange idea
- Think about it and tell/email me if you have any thoughts on it
- Could one very quickly generate a high-quality real-time off-line multi-rate periodic schedule by operating in the frequency domain?
- If not, why not?
- What if the deadlines were soft?

Reading assignment

- J. W. S. Liu, *Real-Time Systems*. Prentice-Hall, Englewood Cliffs, NJ, 2000
- Read Chapter 7

Goals for lecture

- Lab four
- Example scheduling algorithm design problem
  - Will initially focus on static scheduling
- Real-time operating systems
- Comparison of on-line and off-line scheduling code

Lab four

- Talk with Promi SD101
- Sample sound at 3 kHz
- Multihop

Example problem: Static scheduling

- What is an FPGA?
- Why should real-time systems designers care about them?
- Multiprocessor static scheduling
- No preemption
- No overhead for subsequent execution of tasks of same type
- High cost to change task type
- Scheduling algorithm?

Problem: Uniprocessor independent task scheduling

- Problem
  - Independent tasks
  - Each has a period = hard deadline
  - Zero-cost preemption
- How to solve?

Rate monotonic scheduling

Main idea

- 1973, Liu and Layland derived optimal scheduling algorithm(s) for this problem
- Schedule the job with the smallest period (period = deadline) first
- Analyzed worst-case behavior on any task set of size n
- Found utilization bound: $U(n) = n \cdot (2^{1/n} - 1)$
- 0.828 at $n = 2$
- As $n \to \infty$, $U(n) \to \log 2 = 0.693$
- Result: For any problem instance, if a valid schedule is possible, the processor need never spend more than 71% of its time idle

Optimality and utilization for limited case

- Simply periodic: All task periods are integer multiples of all lesser task periods
- In this case, RMS/DMS optimal with utilization 1
- However, this case rare in practice
- Remains feasible, with decreased utilization bound, for in-phase tasks with arbitrary periods

Rate monotonic scheduling

- Constrained problem definition
- Over-allocation often results
- However, in practice utilization of 85%–90% common
  - Lose guarantee
- If phases known, can prove by generating instance

Critical instants

Main idea:

A job's critical instant a time at which all possible concurrent higher-priority jobs are also simultaneously released

Useful because it implies latest finish time

Proof sketch for RMS utilization bound

- Consider case in which no period exceeds twice the shortest period
- Find a pathological case
  - Utilization of 1 for some duration
  - Any decrease in period/deadline of longest-period task will cause deadline violations
  - Any increase in execution time will cause deadline violations

RMS worst-case utilization

- In-phase
- $\forall_{k \text{ s.t. } 1 \leq k \leq n-1} : e_k = p_{k+1} - p_k$
- $e_n = p_n - 2 \cdot \sum_{k=1}^{n-1} e_k$

Proof sketch for RMS utilization bound

- See if there is a way to increase utilization while meeting all deadlines
- Increase execution time of high-priority task
  - $e'_i = p_{i+1} - p_i + \varepsilon = e_i + \varepsilon$
- Must compensate by decreasing another execution time
- This always results in decreased utilization
  - $e'_k = e_k - \varepsilon$
  - $U' - U = \frac{e'_i}{p_i} + \frac{e'_k}{p_k} - \frac{e_i}{p_i} - \frac{e_k}{p_k} = \frac{\varepsilon}{p_i} - \frac{\varepsilon}{p_k}$
  - Note that $p_i < p_k \rightarrow U' > U$

Proof sketch for RMS utilization bound

- Same true if execution time of high-priority task reduced
- $e''_i = p_{i+1} - p_i - \varepsilon$
- In this case, must increase other e or leave idle for $2 \cdot \varepsilon$
- $e''_k = e_k + 2\varepsilon$
- $U'' - U = \frac{2\varepsilon}{p_k} - \frac{\varepsilon}{p_i}$
- Again, $p_k < 2 \rightarrow U'' > U$
- Sum over execution time/period ratios

Proof sketch for RMS utilization bound

- Get utilization as a function of adjacent task ratios
- Substitute execution times into $\sum_{k=1}^{n} \frac{e_k}{p_k}$
- Find minimum
- Extend to cases in which $p_n > 2 \cdot p_k$

Notes on RMS

- Other abbreviations exist (RMA)
- DMS better than or equal RMA when deadline $\neq$ period
- Why not use slack-based?
- What happens if resources are under-allocated and a deadline is missed?

Essential features of RTOSs

- Provides real-time scheduling algorithms or primatives
- Bounded execution time for OS services
  - Usually implies preemptive kernel
  - E.g., linux can spend milliseconds handling interrupts, especially disk access

Threads

- Threads vs. processes: Shared vs. unshared resources
- OS impact: Windows vs. Linux
- Hardware impact: MMU

Threads vs. processes

- Threads: Low context switch overhead
- Threads: Sometimes the only real option, depending on hardware
- Processes: Safer, when hardware provides support
- Processes: Can have better performance when IPC limited

Software implementation of schedulers

- TinyOS
- Light-weight threading executive
- μC/OS-II
- Linux
- Static list scheduler

TinyOS

- Most behavior event-driven
- High rate $\longrightarrow$ Livelock
- Research schedulers exist

BD threads

- Brian Dean: Microcontroller hacker
- Simple priority-based thread scheduling executive
- Tiny footprint (fine for AVR)
- Low overhead
- No MMU requirements

μC/OS-II

- Similar to BD threads
- More flexible
- Bigger footprint

Old linux scheduler

- Single run queue
- $\mathcal{O}(n)$ scheduling operation
- Allows dynamic goodness function

$\mathcal{O}(1)$ scheduler in Linux 2.6

- Written by Ingo Molnar
- Splits run queue into two queues prioritized by goodness
- Requires static goodness function
  - No reliance on running process
- Compatible with preemptible kernel

Real-time linux

- Run linux as process under real-time executive
- Complicated programming model
- RTAI (Real-Time Application Interface) attempts to simplify
  - Colleagues still have problems at $> 18\,\mathrm{kHz}$ control period

Real-time operating systems

- Embedded vs. real-time
- Dynamic memory allocation
- Schedulers: General-purpose vs. real-time
- Timers and clocks: Relationship with HW

Summary

- Static scheduling
- Example of utilization bound proof
- Introduction to real-time operating systems

Reading assignment

- Read Chapter 12 in J. W. S. Liu, *Real-Time Systems*. Prentice-Hall, Englewood Cliffs, NJ, 2000
- Read K. Ghosh, B. Mukherjee, and K. Schwan, “A survey of real-time operating systems,” tech. rep., College of Computing, Georgia Institute of Technology, Feb. 1994

Goals for lecture

- Lab four?
- Lab six
- Simulation of real-time operating systems
- Impact of modern architectural features

Lab four

- Please email or hand in the write-up for lab assignment four
- Problems? See me.
  - Will need everything from lab four working for lab six

Lab six

- Develop priority-based cooperative scheduler for TinyOS that keeps track of the percentage of idle time.
- Develop a tree routing algorithm for the sensor network.
- Send noise, light, and temperature data to a PPC, via the network root.
- Have motes respond to *send audio samples* and *buzz* commands.
- Play back or display this data on PPCs to verify the that the system functions.

Outline

- Introduction
- Role of real-time OS in embedded system
- Related work and contributions
- Examples of energy optimization
- Simulation infrastructure
- Results
- Conclusions

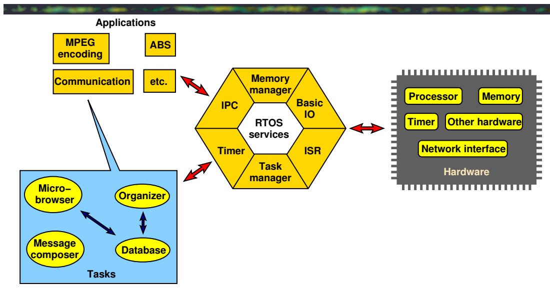
Introduction

- Real-Time Operating Systems are often used in embedded systems.
- They simplify use of hardware, ease management of multiple tasks, and adhere to real-time constraints.
- Power is important in many embedded systems with RTOSs.
- RTOSs can consume significant amount of power.
- They are re-used in many embedded systems.
- They impact power consumed by application software.
- RTOS power effects influence system-level design.

Introduction

- Real Time Operating Systems important part of embedded systems
  – Abstraction of HW
  – Resource management
  – Meet real-time constraints
- Used in several low-power embedded systems
- Need for RTOS power analysis
  – Significant power consumption
  – Impacts application software power
  – Re-used across several applications

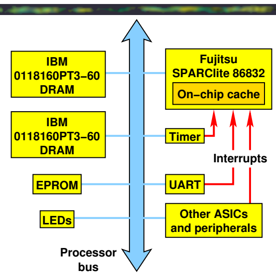
Role of RTOS in embedded system



Related work and contributions

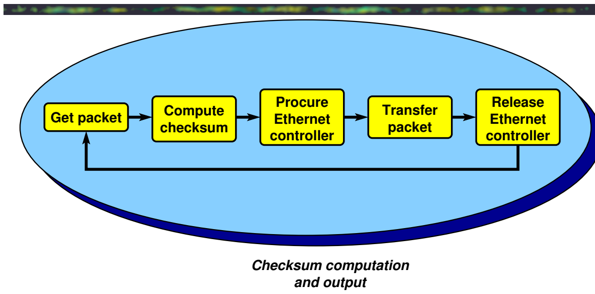
- **Instruction level power analysis**
  V. Tiwari, S. Malik, A. Wolfe, and T.C. Lee, Int. Conf. VLSI Design, 1996
- **System-level power simulation**
  Y. Li and J. Henkel, Design Automation Conf., 1998
- **MicroC/OS-II**: J.J. Labrosse, R & D Books, Lawrence, KS, 1998
- **Our work**
  – First step towards detailed power analysis of RTOS
  – Applications: low-power RTOS, energy-efficient software architecture, incorporate RTOS effects in system design

Simulated embedded system



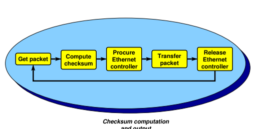
- Easy to add new devices
- Cycle-accurate model
- Fujitsu board support library used in model
- μC/OS-II RTOS used

Single task network interface

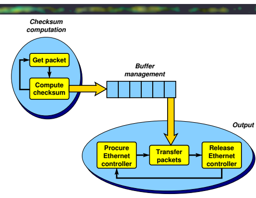


Procuring Ethernet controller has high energy cost

TCP example

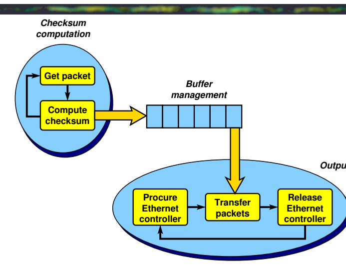


Straight-forward implementation



Multi-task implementation

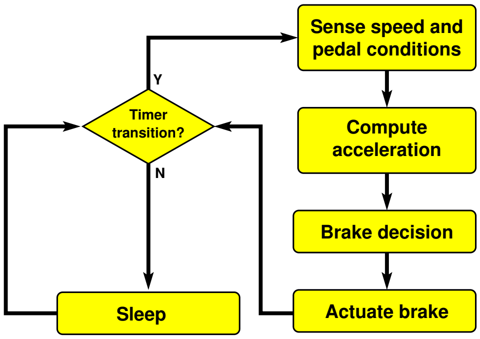
Multi-tasking network interface



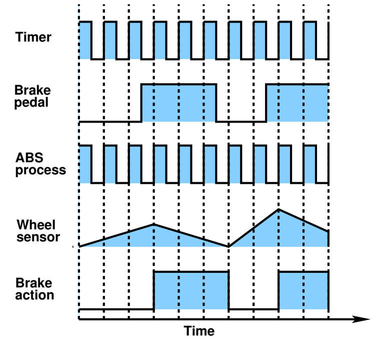
RTOS power analysis used for process re-organization to reduce energy

21% reduction in energy consumption. Similar power consumption.

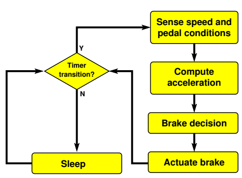
ABS example

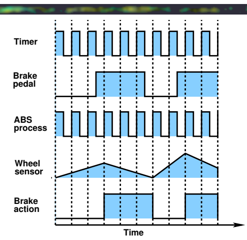


ABS example timing

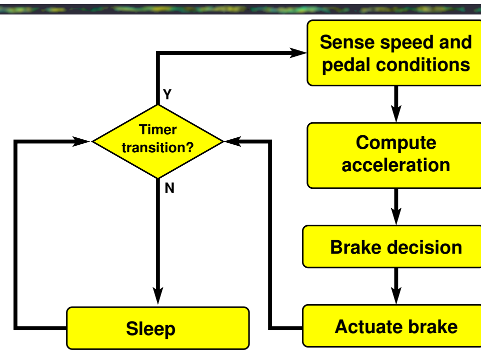


Straight-forward ABS implementation

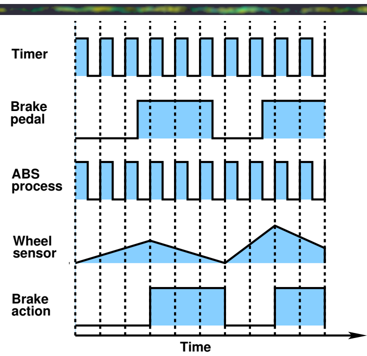




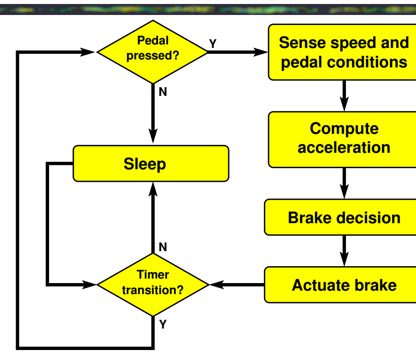
Periodically triggered ABS



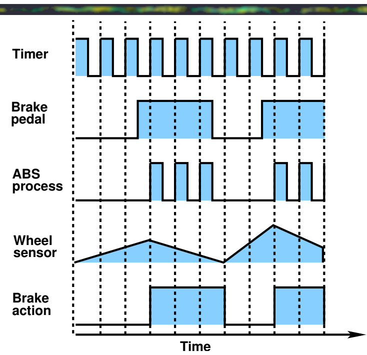
Periodically triggered ABS timing



Selectively triggered ABS

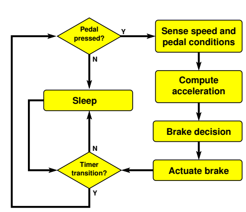


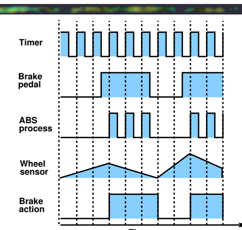
Selectively triggered ABS timing



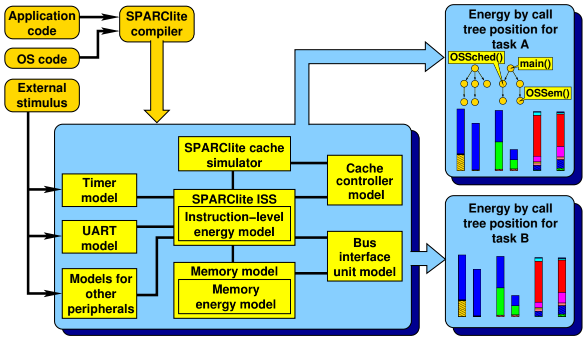
63% reduction in energy and power consumption

Power-optimized ABS example

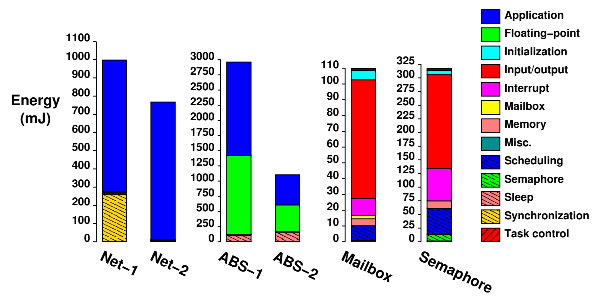




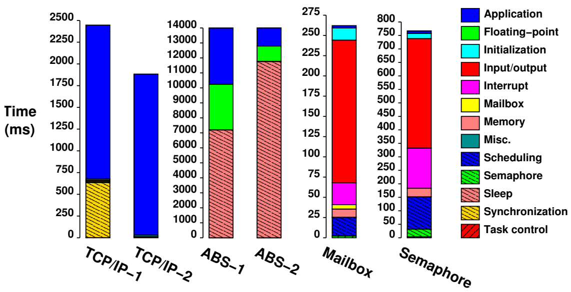
Infrastructure



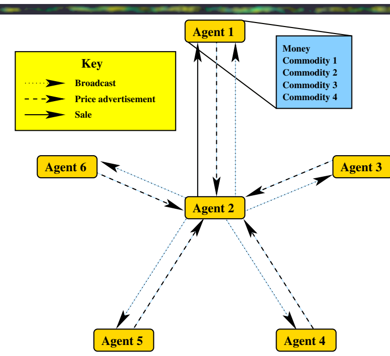
Experimental results



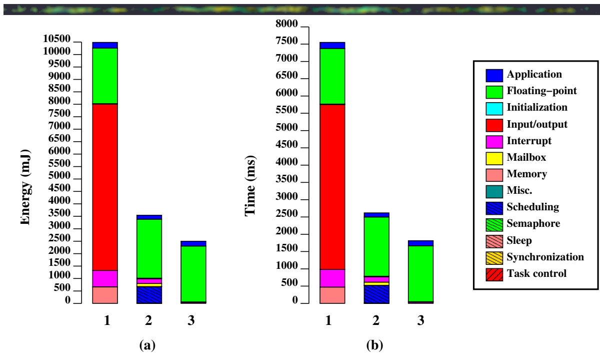
Experimental results – time



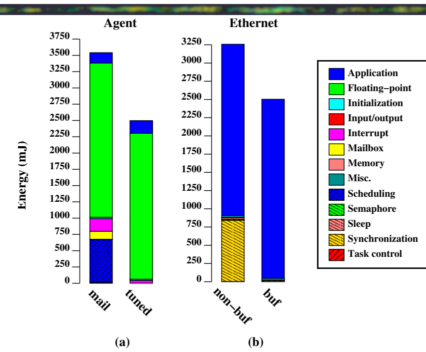
Agent example



Experimental results



Experimental results



Optimization effects

TCP example:

- 20.5% energy reduction
- 0.2% power reduction
- RTOS directly accounted for 1% of system energy

ABS example:

- 63% energy reduction
- 63% power reduction
- RTOS directly accounted for 50% of system energy

Mailbox example: RTOS directly accounted for 99% of system energy

Semaphore example: RTOS directly accounted for 98.7% of system energy

Partial semaphore hierarchical results

		Function	Energy/invocation (uJ)	Energy (%)	Time (mS)	Calls
realstart 6.41 mJ total 2.02 %	init_tvecs		0.41	0.00	0.00	1
	init_timer 5.51 mJ total 1.74 %	liteled	1.31	0.00	0.00	1
	startup 0.90 mJ total 0.28 %	do_main	887.44	0.28	2.18	1
		save_data	1.56	0.00	0.00	1
		init_data	1.31	0.00	0.00	1
		init_bss	0.88	0.00	0.00	1
		cache_on	2.72	0.00	0.01	1
Task1 155.18 mJ total 48.88 %	win_unf_trap		1.90	1.20	9.73	1999
	_OSDisableInt		0.29	0.09	0.78	1000
	_OSEnableInt		0.32	0.10	0.89	1000
	sparcsim_terminate		0.75	0.00	0.00	1
	OSSemPend 31.18 mJ total 9.82 %	win_unf_trap	2.48	0.78	6.33	999
		_OSDisableInt	0.29	0.18	1.59	1999
		_OSEnableInt	0.29	0.18	1.59	1999
		OSEventTaskWait	3.76	1.18	9.22	999
		OSSched	19.07	6.00	47.97	999
	OSSemPost 2.90 mJ total 0.91 %	_OSDisableInt	0.29	0.09	0.78	1000
		_OSEnableInt	0.29	0.09	0.78	1000
	OSTimeGet 1.43 mJ total 0.45 %	_OSDisableInt	0.27	0.08	0.70	1000
		_OSEnableInt	0.29	0.09	0.78	1000
	CPUInit 0.09 mJ total 0.03 %	BSPInit	1.09	0.00	0.00	1
		exceptionHandler	4.77	0.02	0.17	15
	printf 112.90 mJ total 35.56 %	win_unf_trap	2.05	0.65	5.06	1000
		vfprintf	108.89	34.30	258.53	1000

Energy per invocation for μC/OS-II services

Service	Minimum energy (μJ)	Maximum energy (μJ)
`OSEventTaskRdy`	18.02	20.03
`OSEventTaskWait`	7.98	9.05
`OSEventWaitListInit`	20.43	21.16
`OSInit`	1727.70	1823.26
`OSMboxCreate`	27.51	28.82
`OSMboxPend`	7.07	82.91
`OSMboxPost`	5.82	84.55
`OSMemCreate`	19.40	19.75
`OSMemGet`	6.64	8.22
`OSMemInit`	27.41	27.47
`OSMemPut`	6.38	7.91
`OSQInit`	20.10	20.93
`OSSched`	6.96	52.34
`OSSemCreate`	27.87	29.04
`OSSemPend`	6.54	73.64
etc.	etc.	etc.

Conclusions

- RTOS can significantly impact power
- RTOS power analysis can improve application software design
- Applications
  - Low-power RTOS design
  - Energy-efficient software architecture
  - Consider RTOS effects during system design

Impact of modern architectural features

- Memory hierarchy
- Bus protocols ISA vs. PCI
- Pipelining
- Superscalar execution
- SIMD
- VLIW

Summary

- Labs
- Simulation of real-time operating systems
- Impact of modern architectural features

Goals for lecture

- Explain details of a real-time design problem
- Give some background on development of area
- Synthesis solution
- Current commercial status

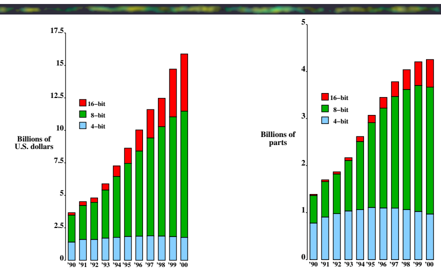
Distributed real-time: Part one

- Distributed needn't mean among cities or offices – Same IC?
- Process scaling trends
- Cross-layer design now necessary

Embedded system / SOC synthesis motivation

- Wireless: effects of the communication medium important
- Hard real-time: deadlines must not be violated
- Reliable: anti-lock brake controllers shouldn't crash
- Rapidly implemented: IP use, simultaneous HW-SW development
- High-performance: massively parallel, using ASICs
- SOC market from $1.1 billion in 1996 to $14 billion in 2000 (Dataquest), to $43 billion in 2009 (Global Information, Inc.)

Global μ-controller sales



Source: Embedded Processor and Microcontroller Primer and FAQ by Russ Hersch

Low-power motivation

- Embedded systems frequently battery-powered, portable
- High heat dissipation results in
  - Expensive, bulky packaging
  - Limited performance
- High-level trade-offs between
  - Power
  - Speed
  - Price
  - Area

Past embedded system synthesis work

- **Early 1990s**: Optimal MILP co-synthesis of small systems [Prakash & Parker], [Bender], [Schwiegershausen & Pirsch]
- **Mid 1990s**: One CPU-One ASIC [Ernst, Henkel & Benner], [Gupta & De Micheli] [Barros, Rosenstiel, & Xiong], [D'Ambrosio & Hu]
- **Late 1990s – present**: Co-synthesis of heterogeneous distributed embedded systems [Kuchcinski], [Quan, Hu, & Greenwood], [Wolf]

Past low-power work

- **Mid 1990s**: VLSI power minimization design surveys [Pedram], [Devadas & Malik]
- **Mid – late 1990s**: High-level power analysis and optimization [Raghunathan, Jha, & Dey], [Chandrakasan & Brodersen]
- **Late 1990s**: Embedded processor energy estimation [Li & Henkel], [Sinha & Chandrakasan]
- **Late 1990s – present**: Low-power hardware-software co-synthesis [Dave, Lakshminarayana, & Jha], [Kirrovski & Potkonjak]

Overview of system synthesis projects

- **TGFF**: Generates parametric task graphs and resource databases
- **MOGAC**: Multi-chip distributed systems
- **CORDS**: Dynamically reconfigurable
- **COWLS**: Multi-chip distributed, wireless, client-server
- **MOCSYN**: System-on-a-chip composed of hard cores, area optimized

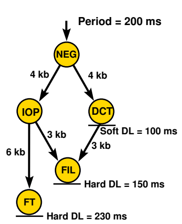
Overview of system synthesis projects

- Synthesize embedded systems
  - heterogeneous processors and communication resources
  - multi-rate
  - hard real-time
- Optimize
  - price
  - power consumption
  - response time

Overview of system synthesis projects

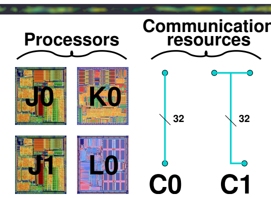
- **TGFF**: Generates parametric task graphs and resource databases
- **MOGAC**: Multi-chip distributed systems
- **CORDS**: Dynamically reconfigurable
- **COWLS**: Multi-chip distributed, wireless, client-server
- **MOCSYN**: System-on-a-chip composed of hard cores, area optimized

Definitions



- Specify
  - task types
  - data dependencies
  - hard and soft task deadlines
  - periods
- Analyze performance of each task on each resource
- Allocate resources
- Assign each task to a resource
- Schedule the tasks on each resource

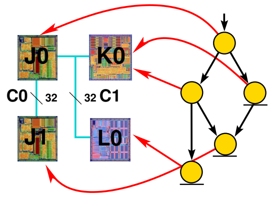
Allocation



Number and types of:

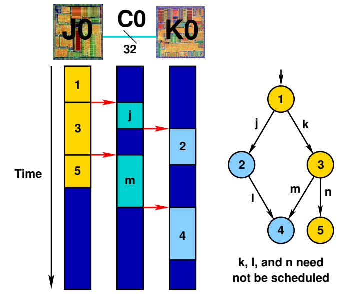
- PEs or cores
- Commun. resources

Assignment



- Assignment of tasks to PEs
- Connection of communication resources to PEs

Schedule



Costs

Soft constraints:

- price
- power
- area
- response time

Hard constraints:

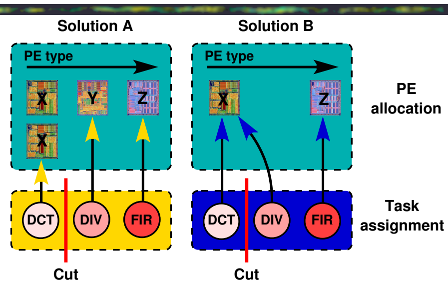
- deadline violations
- PE overload
- unschedulable tasks
- unschedulable transmissions

Solutions which violate hard constraints not shown to designer – pruned out.

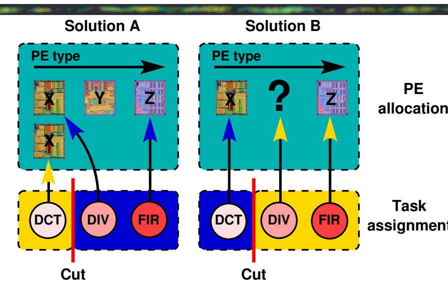
Genetic algorithms

- Multiple solutions
- Local randomized changes to solutions
- Solutions share information with each other
- Can escape sub-optimal local minima
- Scalable

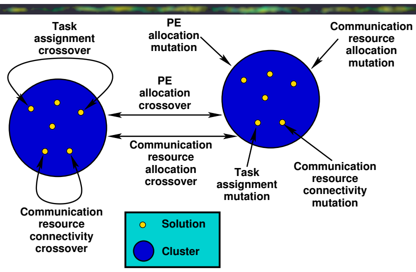
Cluster genetic operator constraints motivation



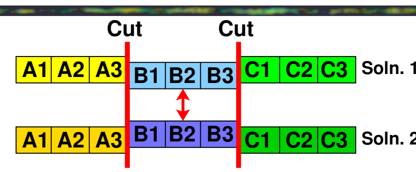
Cluster genetic operator constraints motivation



Cluster genetic operator constraints

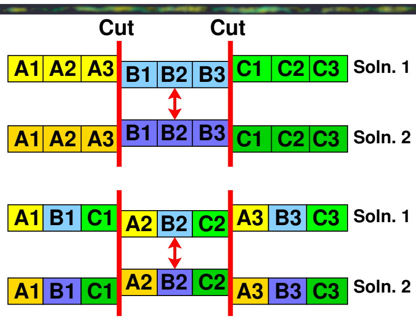


Locality in solution representation

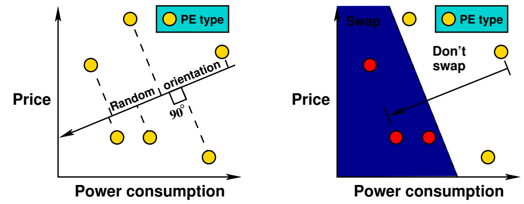


A, B, and C attributes each solve sub-problems

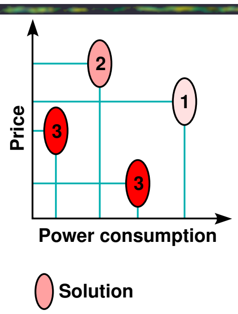
Locality in solution representation



Information trading



Ranking



A solution dominates another if all its costs are lower, i.e.,

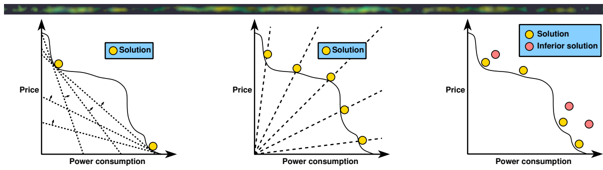
$\mathrm{dom}_{a,b} =$

$\forall_{i=1}^{n} cost_{a,i} < cost_{b,i} \wedge a \neq b$

A solution's rank is the number of other solutions which do not dominate it, i.e.,

$\mathrm{rank}_{s'} = \sum_{i=1}^{n} \text{not } \mathrm{dom}_{s_i,s'}$

Multiobjective optimization



Linear cost functions $\sum_{i=1}^{n} wt_i \cdot cost_i$

Non-linear cost functions $\max_{i=1}^{n} wt_i \cdot cost_i$

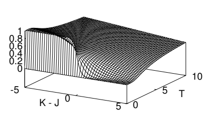
Pareto-rank cost function $\sum_{i=1}^{n} \text{not } \mathrm{dom}_{s_i,s'}$

Reproduction

Solution are selected for reproduction by conducting Boltzmann trials between parents and children.

Given a global temperature T, a solution with rank J beats a solution with rank K with probability:

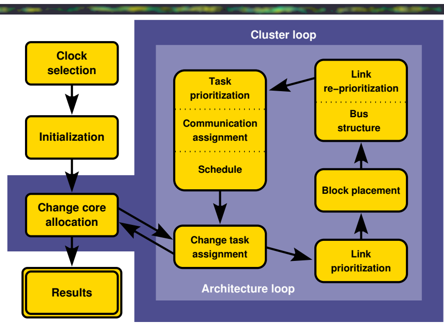
$$\frac{1}{1+e^{(K-J)/T}}$$



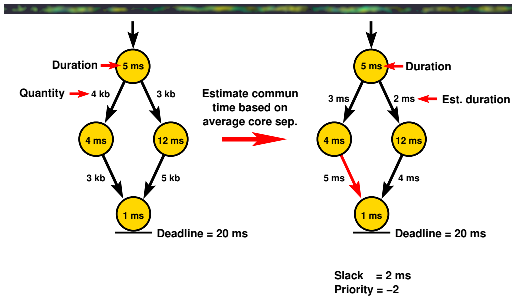
MOCSYN related work

- Floorplanning block placement – Fiduccia and Mattheyses, 1982 – Stockmeyer, 1983
- Parallel recombinative simulated annealing – Mahfoud and Goldberg, 1995
- Linear interpolating clock synthesizers – Bazes, Ashuri, and Knoll, 1996
- Interconnect performance estimation models – Cong & Pan, 2001

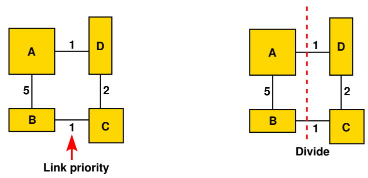
MOCSYN algorithm overview



Link prioritization



Floorplanning block placement

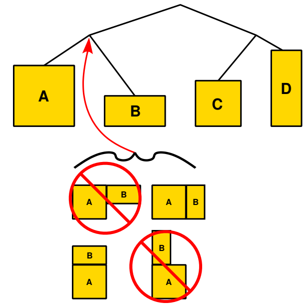


Balanced binary tree of cores formed

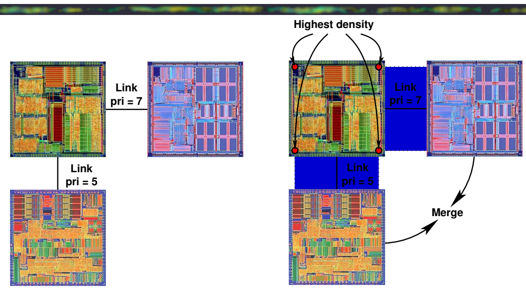
Division takes into account:

- Link priorities
- Area of cores on each side of division

Floorplanning block placement

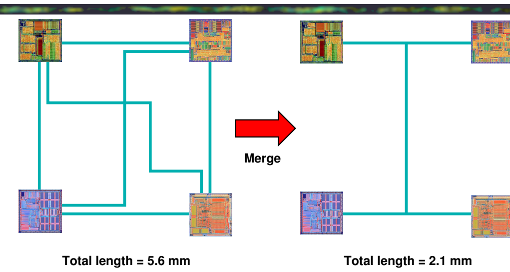


Bus formation

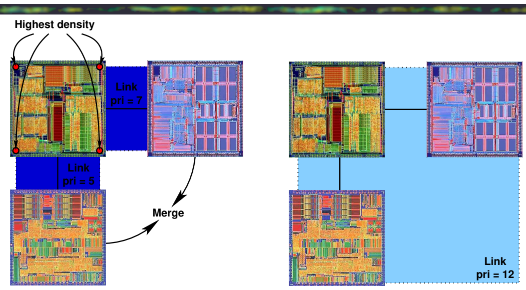


Use efficient red-black tree data structure for intersection tests

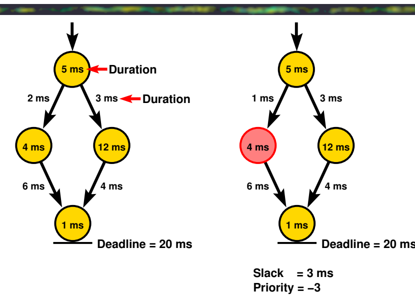
RMST bus length reduction



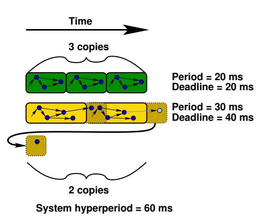
Bus formation



Task prioritization



Scheduling

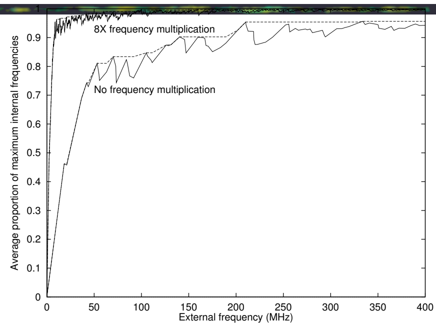


- Fast list scheduler
- Multi-rate
- Handles period $<$ deadline as well as period $\geq$ deadline
- Uses alternative prioritization methods: slack, EST, LFT
- Other features depend on target

Cost calculation

- Price
- Average power consumption
- Area
- PE overload
- Hard deadline violation
- Soft deadline violation
- etc.

Clock selection quality



MOCSYN feature comparisons experiments

Example	MOCSYN price ($)	Worst-case commun. price ($)	Best-case commun. price ($)	Single bus price ($)
...	...	...	...	...
15	216	n.a.	n.a.	n.a.
16	138	n.a.	n.a.	177
17	283	n.a.	n.a.	n.a.
18	253	n.a.	n.a.	253
19	211	n.a.	n.a.	n.a.
...	...	...	...	...
Better		38	44	28
Worse		3	1	9

17 processors, 34 core types, five task graphs, 10 tasks each, 21 task types from networking and telecomm examples.

MOCSYN multiobjective experiments

Example	Price ($)	Average power (mW)	Soft DL viol. prop.	Area (mm^2)
automotive-industrial	91	120	0.60	3.0
	91	120	0.61	2.0
	110	113	0.88	4.0
	110	115	0.60	4.0
networking	61	72	0.94	38.4
telecomm	223	246	2.31	9.9
	223	246	2.76	6.0
	233	255	3.47	4.5
	236	247	2.29	9.9
	236	249	2.60	8.0
	242	221	2.67	3.0
	242	230	2.44	25.9
	242	237	1.72	6.0
	272	226	2.22	192.1
	272	226	2.34	9.4
	353	258	1.23	4.0
consumer	134	281	1.40	34.1
	134	281	1.50	21.6
office automation	64	370	0.23	36.8
	66	55	0.00	7.2

MOGAC run on Hou's examples

Example	Yen's System		MOGAC		
	Price ($)	CPU Time (s)	Price ($)	CPU Time (s)	Tuned CPU Time (s)
Hou 1 & 2 (unclustered)	170	10,205	170	5.7	2.8
Hou 3 & 4 (unclustered)	210	11,550	170	8.0	1.6
Hou 1 & 2 (clustered)	170	16.0	170	5.1	0.7
Hou 3 & 4 (clustered)	170	3.3	170	2.2	0.6

Robust to increase in problem complexity.

2 task graphs each example, 3 PE types

Unclustered: 10 tasks per task graph Clustered: approx. 4 tasks per task graph

MOGAC run on Prakash & Parker's examples

Example ⟨Perform⟩	Prakash & Parker's System		MOGAC		
	Price ($)	CPU Time (s)	Price ($)	CPU Time (s)	Tuned CPU Time (s)
Prakash & Parker 1 ⟨4⟩	7	28	7	3.3	0.2
Prakash & Parker 1 ⟨7⟩	5	37	5	2.1	0.1
Prakash & Parker 2 ⟨8⟩	7	4,511	7	2.1	0.2
Prakash & Parker 2 ⟨15⟩	5	385,012	5	2.3	0.1

Quickly gets optimal when getting optimal is tractable.

3 PE types, Example 1 has 4 tasks, Example 2 has 9 tasks

MOGAC run Yen's large random examples

Example	Yen's System		MOGAC		
	Price ($)	CPU Time (s)	Price ($)	CPU Time (s)	Tuned CPU Time (s)
Random 1	281	10,252	75	6.4	0.2
Random 2	637	21,979	81	7.8	0.2

Handles large problem specifications.

No communication links: communication costs = 0

Random 1: 6 task graphs, approx. 20 tasks each, 8 PE types

Random 2: 8 task graphs, approx. 20 tasks each, 12 PE types

MOCSYN contributions, conclusions

First core-based system-on-chip synthesis algorithm

- Novel problem formulation
- Multiobjective (price, power, area, response time, etc.)
- New clocking solution
- New bus topology generation algorithm

Important for system-on-chip synthesis to do

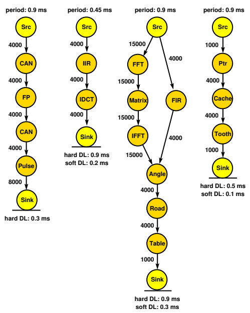
- Clock selection
- Block placement
- Generalized bus topology generation

Research contributions

- **TGFF**: Used by a number of researchers in published work
- **MOGAC**: Real-time distributed embedded system synthesis
  - First true multiobjective (price, power, etc.) system synthesis
  - Solution quality $\geq$ past work, often in orders of magnitude less time
- **CORDS**: First reconfigurable systems synthesis, schedule reordering
- **COWLS**: First wireless client-server systems synthesis, task migration

EEMBC-based embedded benchmarks

Automotive-Industrial



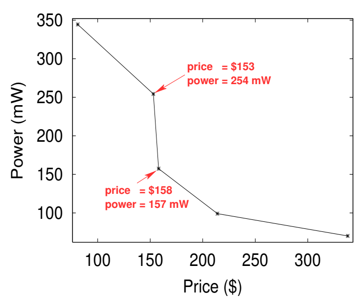
Processors

- AMD ElanSC520 133 MHz
- AMD K6-2 450 MHz
- AMD K6-2E 400MHz/ACR
- AMD K6-2E+ 500MHz/ACR
- AMD K6-IIIE+ 550MHz/ACR
- Analog Devices 21065L 60 MHz
- IBM PowerPC 405GP 266 MHz
- IBM PowerPC 750CX 500 MHz
- IDT32334 100 MHz
- IDT79RC32364 100 MHz
- IDT79RC32V334 150 MHz
- IDT79RC64575 250 MHz
- Imsys Cjip 40 MHz
- Motorola MPC555 40 MHz
- NEC VR5432 167 MHz
- ST20C2 50 MHz
- TI TMS320C6203 300MHz

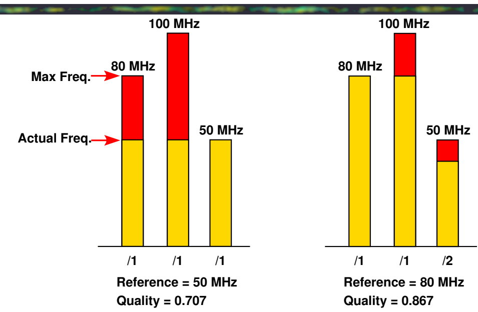
Recently started and future work

- Market-based energy allocation in low-power wireless mobile networks
  - paper under review
- Evolutionary algorithms for multi-dimensional optimization
  - future work
- Task and processor characterization
  - EEMBC-based resource database completed will publicly release
- Tightly coupling low-level, high-level design automation algorithms
  - recently started work in this area

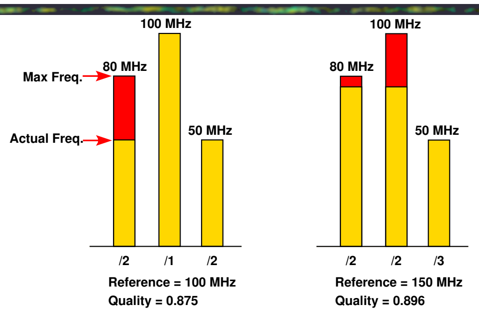
MOGAC run on Yen's second large random example



Counter-division only clock selection



Counter-division only clock selection



Bus formation inner kernel

l is number of communicating core pairs

For each bus, i, intersecting with highest density point: $\mathcal{O}\left(l^2\right)$

  For each bus, j: $\mathcal{O}\left(l^3\right)$

    Tentatively merge i and j $\mathcal{O}\left(l^4\right)$

    Evaluate the density, *new_dens*, of *congest* $\mathcal{O}\left(l^3\right)$

    Evaluate new maximum contention estimate, *cont_est* $\mathcal{O}\left(l^4\right)$

If *new_dens* decreased for any tentative merge:

  Merge the pair with greatest *new_dens* decrease $\mathcal{O}\left(l^2\right)$

  Break ties by selecting merge with least *cont_est* increase.