# Homework index

<table>
<thead>
<tr>
<th></th>
<th>Task Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reading assignment (for next class)</td>
<td>83</td>
</tr>
<tr>
<td>2</td>
<td>Hanford security network design</td>
<td>93</td>
</tr>
<tr>
<td>3</td>
<td>Reading assignment (18 January)</td>
<td>169</td>
</tr>
<tr>
<td>4</td>
<td>Reading assignment</td>
<td>258</td>
</tr>
<tr>
<td>5</td>
<td>Bizarre scheduling idea</td>
<td>292</td>
</tr>
<tr>
<td>6</td>
<td>Reading assignment</td>
<td>293</td>
</tr>
<tr>
<td>7</td>
<td>Reading assignment</td>
<td>320</td>
</tr>
<tr>
<td>8</td>
<td>Lab six</td>
<td>323</td>
</tr>
</tbody>
</table>
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 queing (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 queueing 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 admissible 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 handfuls 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

Static  Dynamic
Taxonomy of real-time systems

Soft

Hard
Taxonomy of real-time systems

Periodic
- Single rate
- Multi-rate

Aperiodic
- Bounded arrival interval
- Unbounded arrival interval
Taxonomy of real-time systems
Taxonomy of real-time systems

- Static
- Dynamic
- Soft
- Hard
- Periodic
  - Single rate
  - Multi-rate
- Bounded arrival interval
- Aperiodic
  - Unbounded arrival interval
Taxonomy of real-time systems

- **Dynamic**
- **Static**
- **Soft**
- **Hard**
- **Single rate**
- **Multi-rate**
- **Periodic**
- **Unbounded arrival interval**
- **Bounded arrival interval**
- **Aperiodic**
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 \(\rightarrow\) Single-rate

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

- Multiple periods.
- Co-prime periods leads to analysis problems.
Taxonomy: Periodic → 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)

<table>
<thead>
<tr>
<th>Task</th>
<th>IBM PowerPC 405GP 266 MHz</th>
<th>IDT79RC32364 100 MHz</th>
<th>Imsys Cijp 40 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tooth</td>
<td>7.7E−6</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td>330E−9</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>FIR</td>
<td>4.1E−6</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Matrix</td>
<td>310E−3</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### 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

tree
reconvergent
undirected
directed
acyclic
cyclic
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)

Diagram:
- Nodes: NEG, DCT, FIL, FT
- Edges: 4 kb, 3 kb, 6 kb
- Delay:
  - Soft DL = 100 ms
  - Hard DL = 100 ms
  - Hard DL = 150 ms
  - Hard DL = 230 ms

Period = 200 ms
Depth-first search (DFS) – Pre-order for trees

\[ \Theta(|V| + |E|) \]
Depth-first search (DFS) – Pre-order for trees

\( \Theta(|V| + |E|) \)
Depth-first search (DFS) – Pre-order for trees

\[ \Theta(|V| + |E|) \]
Depth-first search (DFS) – Pre-order for trees

\[ O(|V| + |E|) \]
Depth-first search (DFS) – Pre-order for trees

\[ \Theta(|V| + |E|) \]
Depth-first search (DFS) – Pre-order for trees

\( \Theta(|V| + |E|) \)
Depth-first search (DFS) – Pre-order for trees

\( O(|V| + |E|) \)
Depth-first search (DFS) – Pre-order for trees

$\Theta(|V| + |E|)$
Breadth-first search (BFS) – Pre-order for trees

$O(|V|)$

A - B - C - D - E - F - G
Breadth-first search (BFS) – Pre-order for trees

\[ O(|V|) \]

A → B → C → D → E → F → G
Breadth-first search (BFS) – Pre-order for trees

$\theta(|V|)$
Breadth-first search (BFS) – Pre-order for trees

\( \Theta(|V|) \)
Breadth-first search (BFS) – Pre-order for trees

\[ \Theta(|V|) \]
Breadth-first search (BFS) – Pre-order for trees

$\Theta(|V|)$
Breadth-first search (BFS) – Pre-order for trees

$\mathcal{O}(|V|)$
Breadth-first search (BFS) – Pre-order for trees

\[ \Theta(|V|) \]
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)

\[ \Theta(|V| + |E|) \]
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)

$O(|V| + |E|)$
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)

\[ \Theta (|V| + |E|) \]
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)

$\Theta(|V| + |E|)$
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)

\[ O(|V| + |E|) \]
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)

\[ O(|V| + |E|) \]
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)

\[ \Theta(|V| + |E|) \]
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)

\[ \Theta(|V| + |E|) \]
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)

$\Theta (|V| + |E|)$
Definition: Deadline violation

- Period = 200 ms

- IOP
  - 4 kb
  - 3 kb
  - 6 kb

- DCT
  - 4 kb
  - 3 kb
  - Soft DL = 100 ms

- FIL
  - Hard DL = 150 ms
  - Hard DL = 230 ms

- FT

- NEG
  - Hard DL = 150 ms
  - Hard DL = 230 ms
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?
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 it’s taking you in a good direction
Summary

- Real-time systems taxonomy and overview
- Definitions
- Importance of problem formulation
Reading assignment (for next class)

- 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
Recall that sorting may be done in $\Theta(n \lg n)$ time.

$\text{DFS} \in \Theta(|V| + |E|)$, $\text{BFS} \in \Theta(|V|)$, $\text{Topological sort} \in \Theta(|V| + |E|)$. 

![Graph showing the growth of $n$, $n \lg n$, and $n^2$ functions as $n$ increases.](chart)
NP-completeness

There also exist exponential-time algorithms: $O\left(2^{\lg n}\right)$, $O\left(2^n\right)$, $O\left(3^n\right)$
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, NP-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 NP-complete in polynomial time, all can be solved in polynomial time

• Unproven conjecture: $\text{NP} \neq \text{P}$
NP-completeness

• What is 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 NP problems in polynomial time.

• Nobody has been able to prove either

\[ P \neq NP \]

or

\[ P = NP \]
NP-completeness

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

\[
P \subseteq \text{NP} \Rightarrow \text{NP-complete} \cap P = \emptyset
\]
Basic complexity classes

- **P** solvable in polynomial time by a computer (Turing Machine)
- **NP** solvable in polynomial time by a nondeterministic computer
- **NP-complete** converted to other **NP-complete** problems in polynomial time
Hard (NP-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 NP-complete?

• If not, solve it

• If so, then what?
How to deal with hard problems

• What should you do when you encounter an apparently hard problem?
  • Is it in NP-complete?
  • If not, solve it
  • If so, then what?

Despair.
How to deal with hard problems

• What should you do when you encounter an apparently hard problem?

• Is it in NP-complete?

• If not, solve it

• If so, then what?

Solve it!
How to deal with hard problems

• What should you do when you encounter an apparently hard problem?

• Is it in NP-complete?

• If not, solve it

• If so, then what?

  Resort to a suboptimal heuristic.
  Bad, but sometimes the only choice.
How to deal with hard problems

• What should you do when you encounter an apparently hard problem?

• Is it in NP-complete?

• If not, solve it

• If so, then what?

    Develop an approximation algorithm.
    Better.
How to deal with hard problems

• What should you do when you encounter an apparently hard problem?

• Is it in NP-complete?

• If not, solve it

• If so, then what?

Determine whether all encountered problem instances are constrained.

Wonderful when it works.
One example

Terminology

- Book’s terminology fine, others also exist
- Different groups → 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
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
- Hardware oriented
- Graph based
- Resource description
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
• Graph based
• Resource description
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
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
• Amenable to theorem proving
• Translation to RTL or C possible
• Commercialized by Esterel Technologies
Design representations

- Software oriented
- Hardware oriented
- Graph based
- Resource description
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
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)
Synchronous dataflow graph (SDFG)
Synchronous dataflow graph (SDFG)
Synchronous dataflow graph (SDFG)
Synchronous dataflow graph (SDFG)
Synchronous dataflow graph (SDFG)
Synchronous dataflow graph (SDFG)
Synchronous dataflow graph (SDFG)
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)
Finite state machine (FSM)
## Finite state machine (FSM)

<table>
<thead>
<tr>
<th>input</th>
<th>current</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

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

From A. Zimmermann’s token game demonstration.
Petrinet

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

From A. Zimmermann’s token game demonstration.
M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann’s token game demonstration.
Petri net

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

From A. Zimmermann’s token game demonstration.
Petri net

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

From A. Zimmermann’s token game demonstration.
Petri net

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

From A. Zimmermann’s token game demonstration.
M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann’s token game demonstration.
Petri net

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

From A. Zimmermann’s token game demonstration.
Petri net

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

From A. Zimmermann’s token game demonstration.
M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann’s token game demonstration.
M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann’s token game demonstration.
M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann’s token game demonstration.
M/D/3/2: Markov arrival, deterministic service delay,

From A. Zimmermann’s token game demonstration.
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)

  – Chapter 1
  – Chapter A5: Sequencing and scheduling

  – 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

Allows pipelining and pre/post-computation

In contrast with book, not difficult to use if conversion automated
Problem definition

- Given a set of tasks,
Problem definition

minimize completion time

- Given a set of tasks,
- a cost function,
Problem definition

minimize completion time

• Given a set of tasks,
• a cost function,
• and a set of resources,
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
Problem definition

- Given a set of tasks,
- a cost function,
- and a set of resources,
- decide the exact time each task will execute on each resource.
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
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 \( \mathbb{P} \)

• Multi-processor
  – There are multiple resources to which tasks may be scheduled

• Usually \( \text{NP-complete} \)
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 NP-complete
Independent tasks –

Precedence constraints

• Independent tasks: No previous execution sequence imposed
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

```
• Homogeneous tasks: All tasks are identical
```
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
One-shot – Periodic

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

- **3 copies**
  - Period: 20 ms
  - Deadline: 20 ms

- **2 copies**
  - Period: 30 ms
  - Deadline: 40 ms

System hyperperiod: 60 ms
Periodic graphs

3 copies
period = 20 ms
deadline = 20 ms

2 copies
period = 30 ms
deadline = 40 ms

system hyperperiod = 60 ms
Periodic graphs

3 copies

period = 20 ms
deadline = 20 ms

period = 30 ms
deadline = 40 ms

2 copies

system hyperperiod = 60 ms
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

A ready

B ready

B deadline

A deadline
Non-preemptive – Preemptive

- Non-preemptive: Tasks must run to completion

- Non-ideal preemptive: Task can be interrupted with cost

- Ideal preemptive: Task can be interrupted without cost

non-preempt.
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

---

• Non-preemptive: Tasks must run to completion
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

non-preempt.
Non-preemptive – Preemptive

- Non-preemptive: Tasks must run to completion
- Ideal preemptive: Tasks can be interrupted without cost
Non-preemptive – Preemptive

Non-preemptive: Tasks must run to completion

Ideal preemptive: Tasks can be interrupted without cost
Non-preemptive – Preemptive

- Non-preemptive: Tasks must run to completion
- Ideal preemptive: Tasks can be interrupted without cost
Non-preemptive – Preemptive

- Non-preemptive: Tasks must run to completion
- Ideal preemptive: Tasks can be interrupted without cost
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
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
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
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
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
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 **NP-complete** allocation-assignment and **NP-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-robin

• 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-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)

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


• 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?

• Wireless protocol?

• OS design?
Low-power sensor networks

- Power consumption central concern in design
- Processor?
  - RISC $\mu$-controllers common
- Wireless protocol?
- OS design?
Low-power sensor networks

• Power consumption central concern in design

• Processor?
  – RISC $\mu$-controllers common

• Wireless protocol?
  – Low data-rate, simple: Proprietary, Zigbee

• OS design?
Low-power sensor networks

• Power consumption central concern in design

• Processor?
  – RISC $\mu$-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?

• Language?
Low-power sensor networks

- Power consumption central concern in design
- Runtime environment?
  - Avoid unnecessary dynamism
- Language?
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-robin
- 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 P

• Mixed integer linear programming: One or more variables discrete
  – NP-complete

• Many good solvers exist

• Don’t rebuild the wheel
MILP scheduling

\( P \) the set of tasks

\( t_{\text{max}} \) maximum time

\( \text{start}(p,t) \) 1 if task \( p \) starts at time \( t \), 0 otherwise

\( D \) the set of execution delays

\( E \) the set of precedence constraints

\[ t_{\text{start}}(p) = \sum_{t=0}^{t_{\text{max}}} t \cdot \text{start}(p,t) \] the start time of \( p \)
MILP scheduling

Each task has a unique start time

\[ \forall p \in P, \sum_{t=0}^{t_{\text{max}}} \text{start}(p,t) = 1 \]

Each task must satisfy its precedence constraints and timing delays

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

Other constraints may exist

- Resource constraints
- Communication delay constraints
MILP scheduling

• Too slow for large instances of NP-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


- 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 \( t \)

\( \delta D_t \) 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**

\[
B = \sum_{b \in \text{pred}} \sum_{t \in F_b} D_t \cdot \delta D_t
\]

**successor force**

\[
C = \sum_{c \in \text{succ}} \sum_{t \in F_c} D_t \cdot \delta D_t
\]
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
Force directed scheduling
Force directed scheduling

probabilistic concurrency
Forcedirected scheduling

probabilistic concurrency
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


• 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
  \[ e'_k = e_k - \varepsilon \]

  \[ U' - U = \frac{e'_i}{p_i} + \frac{e'_k}{p_k} - \frac{e_i}{p_i} - \frac{e_k}{p_k} = \frac{\varepsilon}{p_i} - \frac{\varepsilon}{p_k} \]

  \[ \text{Note that } p_i < p_k \rightarrow U' > U \]
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 ≠ 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
- $\mu$C/OS-II
- Linux
- Static list scheduler
TinyOS

- Most behavior event-driven
- High rate → 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
\( \mu C/OS-II \)

- Similar to BD threads
- More flexible
- Bigger footprint
Old linux scheduler

- Single run queue
- $\mathcal{O}(n)$ scheduling operation
- Allows dynamic goodness function
\( \Theta(1) \) 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 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


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 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

Applications
- MPEG encoding
- Communication
- ABS
- etc.

RTOS services
- IPC
- Memory manager
- Basic IO
- Timer
- Task manager
- ISR

Hardware
- Processor
- Memory
- Timer
- Other hardware
- Network interface
- Database
- Message composer
- Organizer
- Micro-browser

Tasks
Related work and contributions

- **Instruction level power analysis**

- **System-level power simulation**
  Y. Li and J. Henkel, Design Automation Conf., 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
- \( \mu \text{C/OS-II RTOS used} \)
Single task network interface

Checksum computation and output

Procuring Ethernet controller has high energy cost
TCP example

Checksum computation and output

Straight-forward implementation

Multi-task implementation
RTOS power analysis used for process re-organization to reduce energy
21% reduction in energy consumption. Similar power consumption.
ABS example

Timer transition?

- Y: Sense speed and pedal conditions
  - Compute acceleration
    - Brake decision
      - Actuate brake
  - Sleep

- N: Sleep
ABS example timing

Timer
Brake pedal
ABS process
Wheel sensor
Brake action

Time
Straight-forward ABS implementation

- Timer transition?
- Sense speed and pedal conditions
- Compute acceleration
- Brake decision
- Actuate brake

Graph showing:
- Timer
- Brake pedal
- ABS process
- Wheel sensor
- Brake action

Time
Periodically triggered ABS

**Flowchart:**
- **Timer transition?**
  - Y: Sense speed and pedal conditions → Compute acceleration → Brake decision → Actuate brake
  - N: Sleep
Periodically triggered ABS timing

- Timer
- Brake pedal
- ABS process
- Wheel sensor
- Brake action

Time
Selectively triggered ABS

- Pedal pressed?
  - Yes: Sense speed and pedal conditions
  - No: Sleep

- Sleep
  - No: Timer transition?
    - No: Sleep
    - Yes: Actuate brake
  - Yes: Compute acceleration

- Compute acceleration
  - No: Brake decision
  - Yes: Actuate brake
Selectively triggered ABS timing

63% reduction in energy and power consumption
Power-optimized ABS example

- Pedal pressed?
  - Yes: Sense speed and pedal conditions
  - No: Sleep

Sleep
  - Timer transition?
    - Yes: Actuate brake
    - No: Sleep

Timer

Brake action

ABS process

Wheel sensor

Brake pedal

Time
Infrastructure

Application code
SPARClite compiler
OS code
External stimulus

SPARClite cache simulator
SPARClite ISS
Instruction-level energy model

Cache controller model
Bus interface unit model

Timer model
UART model
Models for other peripherals

Energy by call tree position for task A
OSSched() main()
OSSem()

Energy by call tree position for task B
Experimental results
Experimental results – time
Agent example

Key

- - - - - - Broadcast
- - - - - - Price advertisement
- - - - - - Sale

Agent 1

Agent 2

Agent 3

Agent 4

Agent 5

Agent 6

Money
Commodity 1
Commodity 2
Commodity 3
Commodity 4
Experimental results

(a) SleepSynchronizationTask control
(b) Semaphore
Application Floating-point Initialization Input/output Interrupt Mailbox Misc. Scheduling Energy (mJ) Time (ms)
Experimental results

![Energy Consumption Graph](chart.png)

**Agent**
- **Mail**: Various energy consumption levels across different tasks.
- **Tuned**: Similar to mail with slight variations.

**Ethernet**
- **Non-buf**: Energy consumption across different tasks.
- **Buf**: Energy consumption across different tasks.

Legend:
- **Application**
- **Floating-point**
- **Initialization**
- **Input/output**
- **Interrupt**
- **Mailbox**
- **Memory**
- **Misc.**
- **Scheduling**
- **Semaphore**
- **Sleep**
- **Synchronization**
- **Task control**
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Energy/invocation (μJ)</th>
<th>Energy (%)</th>
<th>Time (mS)</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>realstart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.41 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.02 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>init_lvecs</td>
<td>0.41</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>init_timer</td>
<td>1.31</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>lifeleled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do_main</td>
<td>887.44</td>
<td>0.28</td>
<td>2.18</td>
<td>1</td>
</tr>
<tr>
<td>save_data</td>
<td>1.56</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>init_data</td>
<td>1.31</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>init_bss</td>
<td>0.88</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>cache_on</td>
<td>2.72</td>
<td>0.00</td>
<td>0.01</td>
<td>1</td>
</tr>
<tr>
<td>startup</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.90 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.28 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSDisableInt</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>OSEnableInt</td>
<td>0.32</td>
<td>0.10</td>
<td>0.89</td>
<td>1000</td>
</tr>
<tr>
<td>Task1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155.18 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48.88 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>win_unf_trap</td>
<td>1.90</td>
<td>1.20</td>
<td>9.73</td>
<td>1999</td>
</tr>
<tr>
<td>_OSDisableInt</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>_OSEnableInt</td>
<td>0.32</td>
<td>0.10</td>
<td>0.89</td>
<td>1000</td>
</tr>
<tr>
<td>sparcSim_terminate</td>
<td>0.75</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>OSSemPend</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31.18 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.82 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>win_unf_trap</td>
<td>2.48</td>
<td>0.78</td>
<td>6.33</td>
<td>999</td>
</tr>
<tr>
<td>_OSDisableInt</td>
<td>0.29</td>
<td>0.18</td>
<td>1.59</td>
<td>1999</td>
</tr>
<tr>
<td>_OSEnableInt</td>
<td>0.29</td>
<td>0.18</td>
<td>1.59</td>
<td>1999</td>
</tr>
<tr>
<td>OSEventTaskWait</td>
<td>3.76</td>
<td>1.18</td>
<td>9.22</td>
<td>999</td>
</tr>
<tr>
<td>OSSched</td>
<td>19.07</td>
<td>6.00</td>
<td>47.97</td>
<td>999</td>
</tr>
<tr>
<td>OSSemPost</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.90 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.91 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_OSDisableInt</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>_OSEnableInt</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>OSTimeGet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.43 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.45 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_OSDisableInt</td>
<td>0.27</td>
<td>0.08</td>
<td>0.70</td>
<td>1000</td>
</tr>
<tr>
<td>_OSEnableInt</td>
<td>0.29</td>
<td>0.09</td>
<td>0.78</td>
<td>1000</td>
</tr>
<tr>
<td>CPUInit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.09 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.03 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSPInit</td>
<td>1.09</td>
<td>0.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>exceptionHandler</td>
<td>4.77</td>
<td>0.02</td>
<td>0.17</td>
<td>15</td>
</tr>
<tr>
<td>printf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112.90 mJ total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.56 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>win_unf_trap</td>
<td>2.05</td>
<td>0.65</td>
<td>5.06</td>
<td>1000</td>
</tr>
<tr>
<td>viprintf</td>
<td>108.89</td>
<td>34.30</td>
<td>258.53</td>
<td>1000</td>
</tr>
</tbody>
</table>
Energy per invocation for $\mu$C/OS-II services

<table>
<thead>
<tr>
<th>Service</th>
<th>Minimum energy ($\mu$J)</th>
<th>Maximum energy ($\mu$J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSEventTaskRdy</td>
<td>18.02</td>
<td>20.03</td>
</tr>
<tr>
<td>OSEventTaskWait</td>
<td>7.98</td>
<td>9.05</td>
</tr>
<tr>
<td>OSEventWaitListInit</td>
<td>20.43</td>
<td>21.16</td>
</tr>
<tr>
<td>OSInit</td>
<td>1727.70</td>
<td>1823.26</td>
</tr>
<tr>
<td>OSMboxCreate</td>
<td>27.51</td>
<td>28.82</td>
</tr>
<tr>
<td>OSMboxPend</td>
<td>7.07</td>
<td>82.91</td>
</tr>
<tr>
<td>OSMboxPost</td>
<td>5.82</td>
<td>84.55</td>
</tr>
<tr>
<td>OSMemCreate</td>
<td>19.40</td>
<td>19.75</td>
</tr>
<tr>
<td>OSMemGet</td>
<td>6.64</td>
<td>8.22</td>
</tr>
<tr>
<td>OSMemInit</td>
<td>27.41</td>
<td>27.47</td>
</tr>
<tr>
<td>OSMemPut</td>
<td>6.38</td>
<td>7.91</td>
</tr>
<tr>
<td>OSQInit</td>
<td>20.10</td>
<td>20.93</td>
</tr>
<tr>
<td>OSSched</td>
<td>6.96</td>
<td>52.34</td>
</tr>
<tr>
<td>OSSemCreate</td>
<td>27.87</td>
<td>29.04</td>
</tr>
<tr>
<td>OSSemPend</td>
<td>6.54</td>
<td>73.64</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>
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 $\mu$-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
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
Assignment

- Assignment of tasks to PEs
- Connection of communication resources to PEs
k, l, and n need not be scheduled
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

Solution A

Solution B

PE type

PE allocation

Task assignment

Cut
Cluster genetic operator constraints motivation

Solution A

Solution B

PE type

PE allocation

Task assignment

Cut

DCT

DIV

FIR

DCT

DIV

FIR

372
Cluster genetic operator constraints motivation

Solution A

PE type

DCT  DIV  FIR

Cut

Solution B

PE type

DCT  DIV  FIR

Cut

PE allocation

Task assignment
Cluster genetic operator constraints
Locality in solution representation

A, B, and C attributes each solve sub-problems
Locality in solution representation

Cut

A1 A2 A3 B1 B2 B3 C1 C2 C3 Soln. 1

A1 A2 A3 B1 B2 B3 C1 C2 C3 Soln. 2

A1 B1 C1 A2 B2 C2 A3 B3 C3 Soln. 1

A1 B1 C1 A2 B2 C2 A3 B3 C3 Soln. 2
Information trading

Don't swap

Swap

Random orientation

90°
A solution dominates another if all its costs are lower, i.e.,
\[ \text{dom}_{a,b} = \forall_{i=1}^{n} \text{cost}_{a,i} < \text{cost}_{b,i} \land a \neq b \]

A solution’s rank is the number of other solutions which do not dominate it, i.e.,
\[ \text{rank}_{s'} = \sum_{i=1}^{n} \text{not dom}_{s_i,s'} \]
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
\[ \sum_{i=1}^{n} \text{not dom}_{s_i,s'} \]
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

Cluster loop

- Task prioritization
- Communication assignment
- Schedule
- Change task assignment
- Change core allocation
- Initialization
- Clock selection

Architecture loop

- Link re-prioritization
- Bus structure
- Block placement
- Link prioritization
- Results
MOCSYN algorithm overview

Cluster loop
- Task prioritization
- Communication assignment
- Schedule
- Change task assignment
- Link re-prioritization
- Bus structure
- Block placement
- Link prioritization

Architecture loop
- Clock selection
- Initialization
- Change core allocation
- Results
Clock selection

• Cores have different maximum frequencies
• Globally synchronous system forces underclocking
• Multiple crystals too expensive
• Use linear interpolating clock synthesizers
  – Standard CMOS process
  – Each core runs near highest speed
  – Global clock frequency can be low to reduce power
• Optimal clock selection algorithm in pre-pass
MOCSYN algorithm overview

Cluster loop

- Clock selection
- Initialization
- Change core allocation
- Results

Architecture loop

- Task prioritization
- Communication assignment
- Schedule
- Change task assignment
- Link re-prioritization
- Bus structure
- Block placement
- Link prioritization
MOCSYN algorithm overview

Cluster loop

- Clock selection
- Initialization
- Change core allocation
- Results
- Task prioritization
- Communication assignment
- Schedule
- Change task assignment
- Link re-prioritization
- Bus structure
- Block placement
- Link prioritization

Architecture loop
MOCSYN algorithm overview
MOCSYN algorithm overview
Link prioritization

Priority = −2
Slack = 2 ms
Deadline = 20 ms

Slack = 2 ms
Priority = −2
Deadline = 20 ms
MOCSYN algorithm overview

Block placement to determine communication time, energy
Balanced binary tree of cores formed
Division takes into account:
  • Link priorities
  • Area of cores on each side of division
Floorplanning block placement
Floorplanning block placement
MOCSYN algorithm overview

Cluster loop

- Link re-prioritization
- Bus structure

Architecture loop

- Block placement
- Link prioritization
- Change task assignment
- Schedule
- Communication assignment
- Task prioritization

Initialization

- Change core allocation

Results

Clock selection

- Initialization
- Change core allocation

Bus topology generation: minimize contention under routability constraints
Bus formation

Use efficient red-black tree data structure for intersection tests
RMST bus length reduction

Total length = 5.6 mm

Total length = 2.1 mm

Merge
Bus formation

Highest density

Link pri = 7

Link pri = 5

Merge

Link pri = 12
MOCSYN algorithm overview

Cluster loop

- Change task assignment
- Schedule
- Communication assignment
- Task prioritization

Architecture loop

- Link prioritization
- Block placement
- Bus structure
- Link re-prioritization

Initialization

- Change core allocation

Results

Clock selection
Task prioritization

Deadline = 20 ms
Slack = 3 ms
Priority = -3
Scheduling

- Fast list scheduler
- Multi-rate
- Handles period $< \text{deadline}$ as well as period $\geq \text{deadline}$
- 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

Average proportion of maximum internal frequencies

External frequency (MHz)

8X frequency multiplication
No frequency multiplication
### MOCSYN feature comparisons experiments

<table>
<thead>
<tr>
<th>Example</th>
<th>MOCSYN price ($)</th>
<th>Worst-case commun. price ($)</th>
<th>Best-case commun. price ($)</th>
<th>Single bus price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>216</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>16</td>
<td>138</td>
<td>n.a.</td>
<td>n.a.</td>
<td>177</td>
</tr>
<tr>
<td>17</td>
<td>283</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>18</td>
<td>253</td>
<td>n.a.</td>
<td>n.a.</td>
<td>253</td>
</tr>
<tr>
<td>19</td>
<td>211</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Better</td>
<td></td>
<td>38</td>
<td>44</td>
<td>28</td>
</tr>
<tr>
<td>Worse</td>
<td></td>
<td>3</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Example</th>
<th>Price ($)</th>
<th>Average power (mW)</th>
<th>Soft DL viol. prop.</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>automotive-industrial</td>
<td>91</td>
<td>120</td>
<td>0.60</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>91</td>
<td>120</td>
<td>0.61</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>113</td>
<td>0.88</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>115</td>
<td>0.60</td>
<td>4.0</td>
</tr>
<tr>
<td>networking</td>
<td>61</td>
<td>72</td>
<td>0.94</td>
<td>38.4</td>
</tr>
<tr>
<td>telecomm</td>
<td>223</td>
<td>246</td>
<td>2.31</td>
<td>9.9</td>
</tr>
<tr>
<td></td>
<td>223</td>
<td>246</td>
<td>2.76</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>255</td>
<td>3.47</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td>236</td>
<td>247</td>
<td>2.29</td>
<td>9.9</td>
</tr>
<tr>
<td></td>
<td>236</td>
<td>249</td>
<td>2.60</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>242</td>
<td>221</td>
<td>2.67</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>242</td>
<td>230</td>
<td>2.44</td>
<td>25.9</td>
</tr>
<tr>
<td></td>
<td>242</td>
<td>237</td>
<td>1.72</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>272</td>
<td>226</td>
<td>2.22</td>
<td>192.1</td>
</tr>
<tr>
<td></td>
<td>272</td>
<td>226</td>
<td>2.34</td>
<td>9.4</td>
</tr>
<tr>
<td></td>
<td>353</td>
<td>258</td>
<td>1.23</td>
<td>4.0</td>
</tr>
<tr>
<td>consumer</td>
<td>134</td>
<td>281</td>
<td>1.40</td>
<td>34.1</td>
</tr>
<tr>
<td></td>
<td>134</td>
<td>281</td>
<td>1.50</td>
<td>21.6</td>
</tr>
<tr>
<td>office automation</td>
<td>64</td>
<td>370</td>
<td>0.23</td>
<td>36.8</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>55</td>
<td>0.00</td>
<td>7.2</td>
</tr>
</tbody>
</table>
### MOGAC run on Hou’s examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Yen’s System</th>
<th>MOGAC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Price ($)</td>
<td>Price ($)</td>
</tr>
<tr>
<td></td>
<td>CPU Time (s)</td>
<td>CPU Time (s)</td>
</tr>
<tr>
<td>Hou 1 &amp; 2 (unclustered)</td>
<td>170</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>10,205</td>
<td>2.8</td>
</tr>
<tr>
<td>Hou 3 &amp; 4 (unclustered)</td>
<td>210</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>11,550</td>
<td>1.6</td>
</tr>
<tr>
<td>Hou 1 &amp; 2 (clustered)</td>
<td>170</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>16.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Hou 3 &amp; 4 (clustered)</td>
<td>170</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>0.6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Example 〈Perform〉</th>
<th>Prakash &amp; Parker’s System</th>
<th>MOGAC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Price ($)</td>
<td>CPU Time (s)</td>
</tr>
<tr>
<td>Prakash &amp; Parker 1 〈4〉</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>Prakash &amp; Parker 1 〈7〉</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>Prakash &amp; Parker 2 〈8〉</td>
<td>7</td>
<td>4,511</td>
</tr>
<tr>
<td>Prakash &amp; Parker 2 〈15〉</td>
<td>5</td>
<td>385,012</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Example</th>
<th>Yen’s System</th>
<th>MOGAC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Price ($)</td>
<td>CPU Time (s)</td>
</tr>
<tr>
<td>Random 1</td>
<td>281</td>
<td>10,252</td>
</tr>
<tr>
<td>Random 2</td>
<td>637</td>
<td>21,979</td>
</tr>
</tbody>
</table>

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

![Graph showing the relationship between price and power. Points marked with prices and power values:]

- **Price = $153, Power = 254 mW**
- **Price = $158, Power = 157 mW**
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
MOCSYN Networking example

Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Price, power, and area only. Soft deadline violation omitted.
Problem complexity

Allocations:
\[ \text{max}_\text{PE}_\text{per}_\text{type}^{\text{max}_\text{PE}\text{types}} \times \text{max}_\text{link}_\text{per}_\text{type}^{\text{max}_\text{link}\text{types}} \]

Assignments:
\[ O \left( \text{PE}\_\text{count}^{\text{task}\_\text{count}} \right) \]

Link Connectivities:

- Consider each PE to be a node in a graph
- Each link is a group which can contain up to \( \text{max}_\text{contacts}_\text{per}_\text{link} \) nodes

\[ O \left( C(\text{PE}\_\text{count, max}_\text{contacts}_\text{per}_\text{link})^{\text{link}\_\text{count}} \right) \]
Take a simple system:

\[
\begin{align*}
\text{max}_\text{PE}_\text{per}_\text{type} &= \text{max}_\text{link}_\text{per}_\text{type} = 3 \\
\text{max}_\text{PE}_\text{types} &= \text{max}_\text{link}_\text{types} = 3 \\
\text{PE}_\text{count} &= \text{link}_\text{count} = 9 \\
\text{task}_\text{count} &= 10 \\
\text{max}_\text{contacts}_\text{per}_\text{link} &= 2
\end{align*}
\]

\[
\begin{align*}
\text{allocations} &= 3^3 \cdot 3^3 = 27 & \text{good} \\
\text{assignments} &= \Theta(9^{10}) = \Theta(3.49 \times 10^9) & \text{bad} \\
\text{connectivities} &= \Theta(C(9, 2)^9) = \Theta(1.02 \times 10^{14}) & \text{worse}
\end{align*}
\]

Number of architectures to evaluate:

\[
\Theta(27 \cdot 3.49 \times 10^9 \cdot 1.02 \times 10^{14}) = \Theta(9.57 \times 10^{24})
\]

...and this does not even take scheduling complexity or multi-core ICs into account
Counter-division only clock selection

Reference = 50 MHz
Quality = 0.707

Reference = 80 MHz
Quality = 0.867
Counter-division only clock selection

Reference = 100 MHz
Quality = 0.875

Reference = 150 MHz
Quality = 0.896
Bus formation inner kernel

$l$ is number of communicating core pairs

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

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

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

Evaluate the density, $new\_dens$, of congest $\mathcal{O}(l^3)$

Evaluate new maximum contention estimate, $cont\_est$ $\mathcal{O}(l^4)$

If $new\_dens$ decreased for any tentative merge:

Merge the pair with greatest $new\_dens$ decrease $\mathcal{O}(l^2)$

Break ties by selecting merge with least $cont\_est$ increase.