Kernel and Application Partitioning for EDF Schedule Feasibility

Andrew Morton
University of Waterloo
Canada

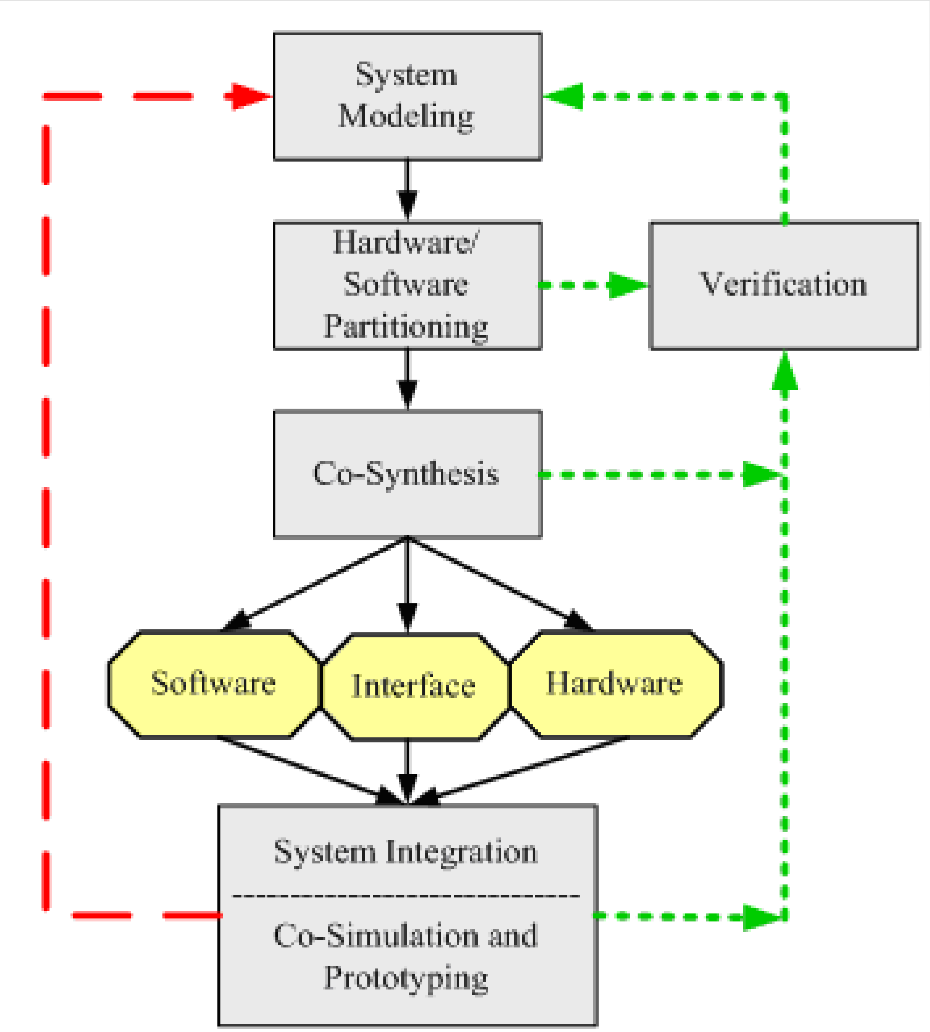
Outline

1) Introduction and motivation
2) Review of kernel partitioning
3) EDF summary
4) Partitioning model
5) Results
6) Summary

Introduction

- Hardware/Software Codesign
  - Specification
  - Partitioning
  - Synthesis
  - Verification

Sample Codesign Flow



- Taken from Micaela Serra at UVic
  - http://webhome.cs.uvic.ca/~mserra/HScodesign.html
-
- Most codesign flows do scheduling after partitioning
- Few consider concurrent real-time systems

Partitioning the Kernel

- Previous work
  - δ framework
    - User selected components of Atalanta kernel can be moved to hardware
    - Demonstrated increased speed in database-type application from 20-40%
  - Spring OS
    - Moved all scheduling into the SSCoP (Spring Scheduling CoProcessor)
    - Demonstrated 4x – 6x speedup in scheduling

Why the Kernel?

- The kernel executes more often than any/all tasks
  - It is invoked every time a task releases, blocks, unblocks or terminates
- High execution frequency:
  - Small reductions in execution time can lead to significant gains in schedule feasability

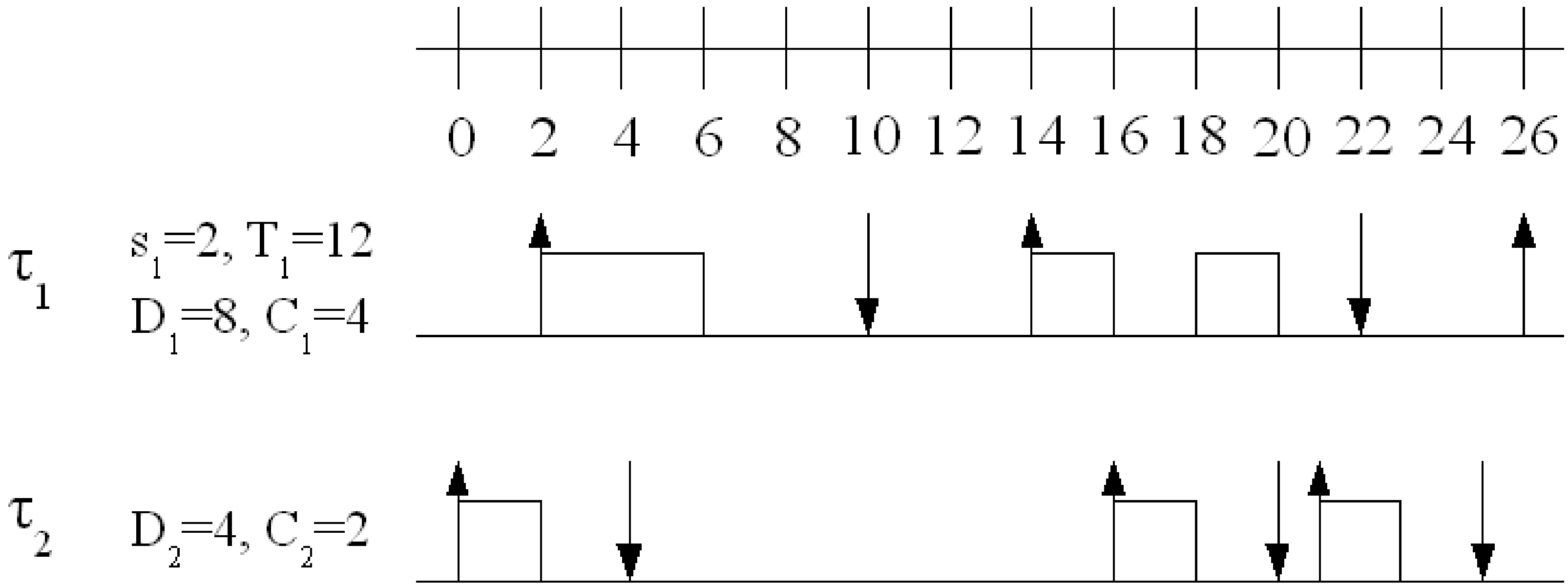
Earliest Deadline First (EDF)

- Definition of EDF
  - of all ready tasks, the task with the earliest deadline is executed first
  - if another task arrives with earlier deadline, it preempts the current task

Why EDF?

- Earliest Deadline First
  - EDF is optimal – will only miss a deadline if no other policy could make it
  - It can achieve 100% processor utilization
    - (compared to 70% limit for Rate Monotonic)
  - It's a “natural” way to specify deadlines in embedded system
  - Had studied it's theory and implementation during PhD

Scheduling Notation



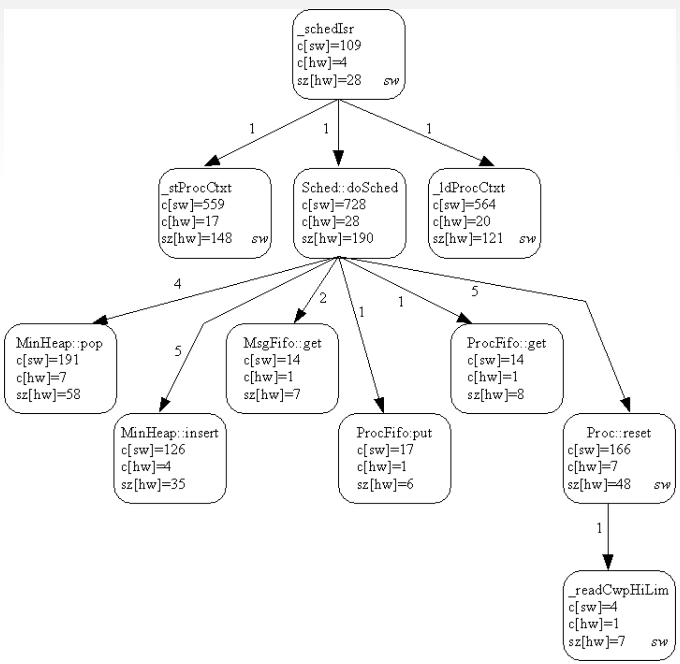
- s_i start time, T_i period, C_i worst-case execution time, D_i deadline

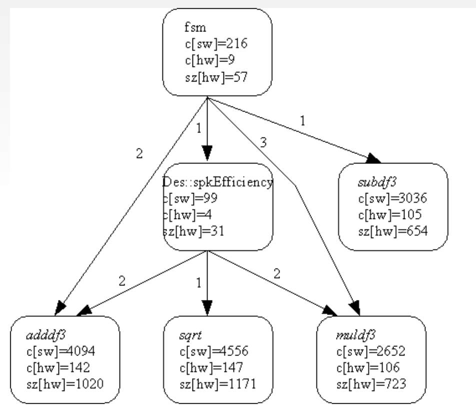
Partitioner Input

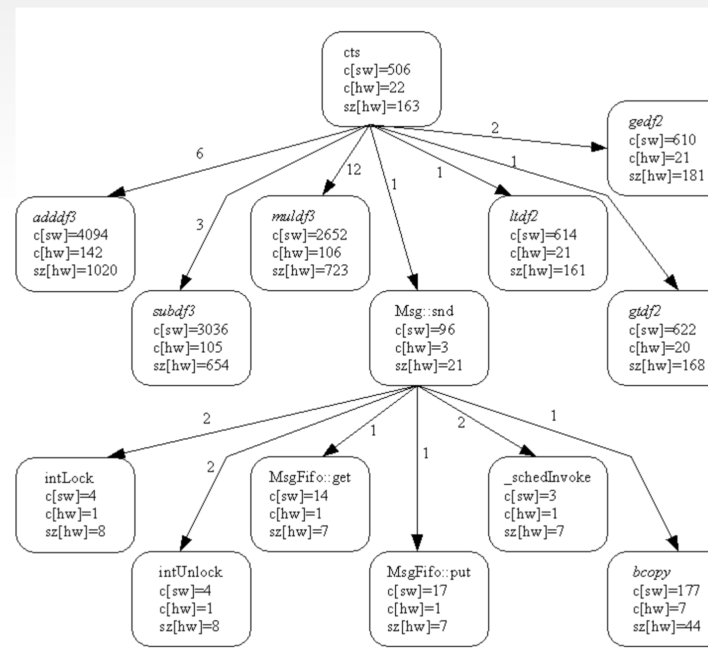
- SLIF Graphs
  - (system level intermediate format)
  - A call graph
  - Nodes represent functions
    - Labelled with hw size and hw/sw execution time
  - Directed edges represent invocations
    - Labelled with invocation frequency
  - One per task
- Also task period T_i and deadline D_i

Partitioner Input

kernel | $\tau_1 : T_1, D_1$ | $\tau_2 : T_2, D_2$

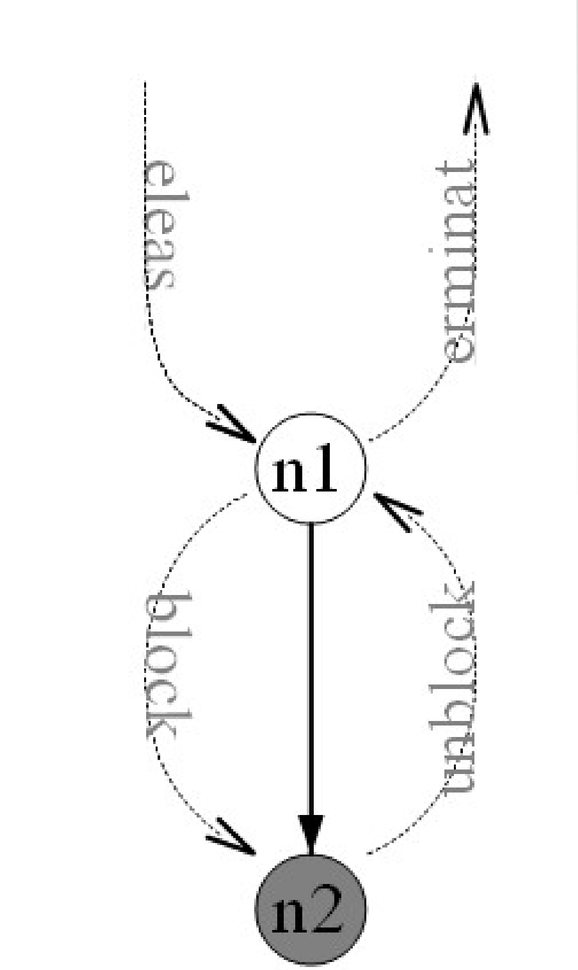






Assumptions

- Assign nodes to hw/sw
    - Task or kernel nodes
- Each cut task edge adds 2 kernel invocations
- Each task also requires 2 kernel invocations for release/terminate



Principle of Operation

1) Assign every node to hw/sw (kernel and application nodes)
2) Objective: minimize processor utilization
3) Check schedule feasibility
4) Add constraints for violated deadlines and repeat

Testing

- Used Embedded Systems Synthesis Suite (E3S) application benchmarks
  - Automotive, consumer, networking, office automation and telecommunications
  - Used MPC555 data
- Kernel: 11 nodes
  - 5 bound to software (e.g. context switch)
  - 6 eligible for hw/sw
- Application nodes
  - All eligible for hw/sw

Results
Nodes Assigned to Hardware

	Kernel			Task		
App	Eligible	Assigned	Fraction	Eligible	Assigned	Fraction
Automotive	6	5	83.3%	24	20	83.3%
Consumer	6	5	83.3%	12	0	0%
Networking	6	5	83.3%	13	8	54.5%
Office	6	4	66.7%	5	0	0%
Telecom	6	1	16.7%	30	30	100%

- Of 26 tasks, 21 were not partitioned (i.e. all hardware or all software)

Summary

- Contributions
  - Unified model for partitioning and scheduling of real-time systems
  - Demonstrated a preference to assign kernel functions to hardware