Functional Reactive Programming and Response Time Analysis for Developing Embedded/Real-Time and Cyber-Physical Systems

Albert M. K. Cheng

- **Outline**
  - **Embedded Real-Time Systems**
  - **Functional Reactive Systems (FRS)**
  - **Cyber-Physical Systems (CPS)**
  - **Haskell and Functional Reactive Programming (FRP)**
  - **Priority-based FRP (P-FRP)**
  - **Response time analysis**
  - **Power-aware scheduling**

* Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

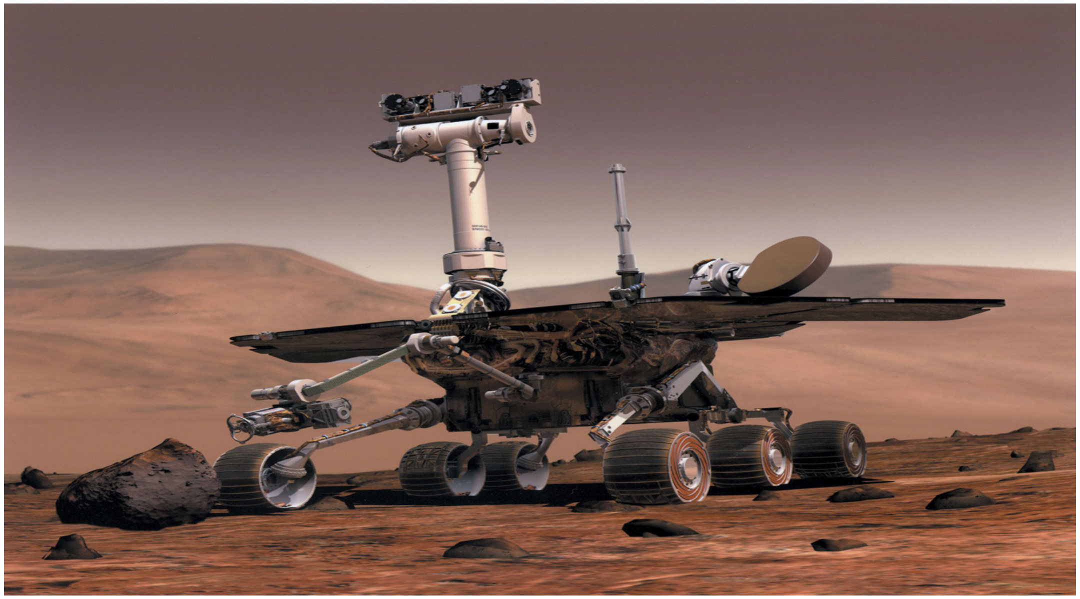
Real-Time Systems Group

- **Director**
  Prof. Albert M. K. Cheng
- **PhD students**
  Yong Woon Ahn, Yu Li, Xingliang Zou, Behnaz Sanati, Sergio Chacon, Zeinab Kazemi, Chaitanya Belwal (just graduated)
- **MS students**
  Daxiao Liu, Yuanfeng Wen (just graduated), Fang Liu (just graduated)
- **Undergraduate students (NSF-REU)**
  Mozahid Haque, Kaleb Christoffersen, Dylan Thompson (just completed), James Hyatt (just completed)
- **Visiting scholars**
  Yu Jiang, Heilongjiang University, Harbin, China; Qiang Zhou (arriving in November 2013), Beihang University, Beijing, China

Yu Li (Best Junior PhD Student Awardee and Friends of NSM Graduate Fellow) and Prof. Albert Cheng visit the NSF-sponsored Arecibo Observatory (world's largest and most sensitive radiotelescope) in Arecibo, Puerto Rico, after their presentation at the flagship RTSS 2012.

Real-time systems research group at Yuanfeng Wen's graduation party in May 2013.

Real-Time Systems Theory

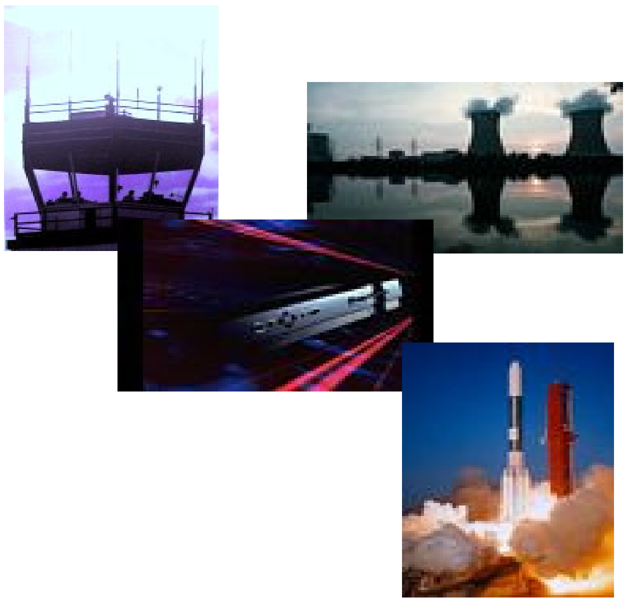


Pathfinder mission to Mars: best known Priority Inversion problem.

Failure to turn on priority Inheritance (PI) - Most PI schemes complicate and slow down the locking code, and often are used to compensate for poor application designs.

http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/mars_pathfinder.html

Real-Time Systems Theory

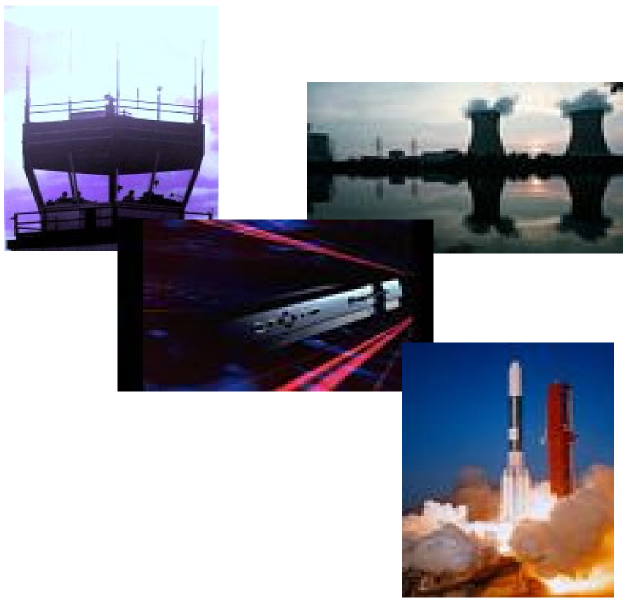


- **The more components a real-time system has, the more difficult it is to build and maintain.**
  - In such systems, **preemptive scheduling** may not be suitable, since it is likely to create runtime overheads which can result in worst-case task execution times of up to **40%** greater than fully **non-preemptive** execution.
    - Yao G., Buttazzo G., Bertogna M., "Feasibility analysis under fixed priority scheduling with limited preemptions," Real-Time Systems, Volume 47 Issue 3, pages: 198-223, May 2011.

Real-Time Systems Theory

- However, **preemptive** scheduling allows for more feasible schedules than **non-preemptive** scheduling.
- **Non-preemptive** scheduling automatically prevents unbounded priority inversion, which avoids the need for a concurrency control protocol, leading to a less complex scheduling model.
- However, fully **non-preemptive** scheduling is too inflexible for some real-time applications, and has the added disadvantage of potentially introducing large blocking times that would make it impossible to guarantee the schedulability of the task set.

Real-Time Systems Theory



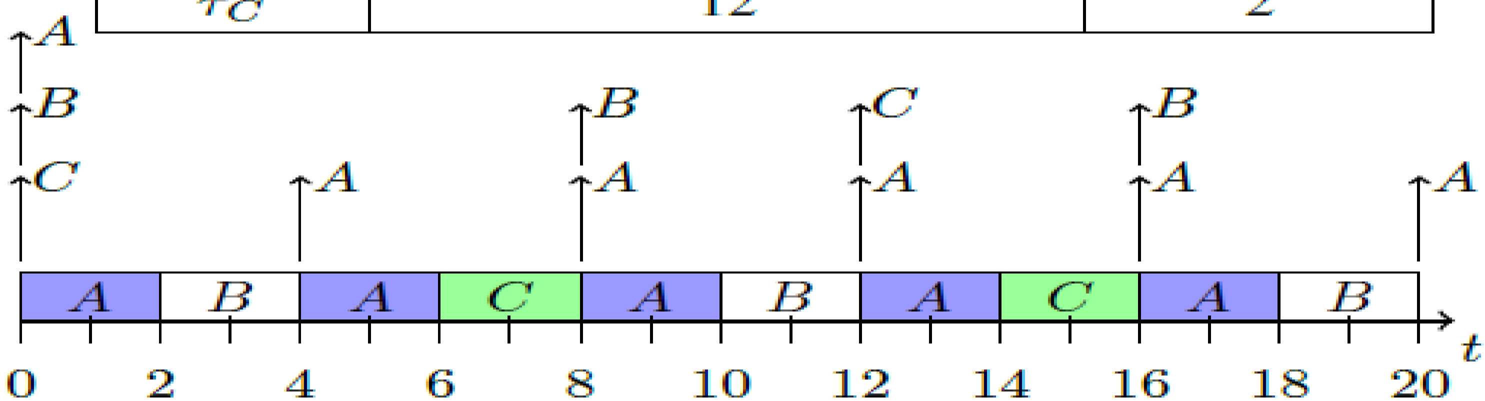
- Simplify the design and scheduling
  - Avoid priority inheritance
  - Use functional programming
  - Use abort-and-restart
  - Use harmonic task sets
    - However, harmonic tasks sets may be too restrictive for some situations. For example, one sensor needs to be serviced every 9 seconds and another (because of its design / physical characteristics) 10 seconds.

Real-Time Systems Theory

- Example (1) - Harmonic task sets
  - Can achieve 100% CPU utilization
  - Can avoid preemption and context switches costs

V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese, "**Polynomial-Time Exact Schedulability Tests for Harmonic Real-Time Tasks," RTSS 2013.**

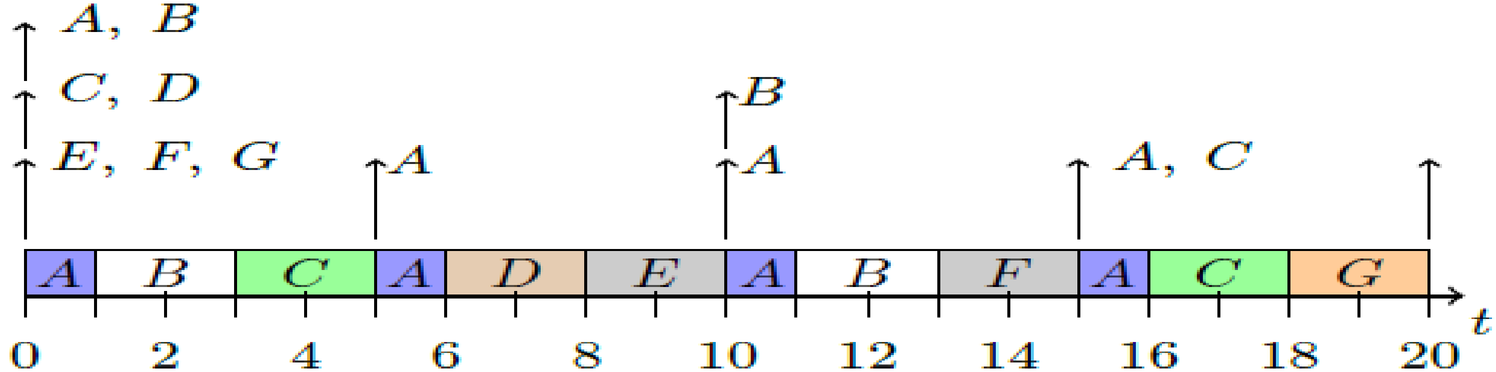
Task ID	Period=Deadline (T = D)	WCET (C)
τ_A	4	2
τ_B	8	2
τ_C	12	2



Real-Time Systems Theory

- Example (2) - Harmonic task sets

Task ID	Period=Deadline (T = D)	WCET (C)
τ_A	5	1
τ_B	10	2
τ_C	15	2
τ_D	20	2
τ_E	25	2
τ_F	30	2
τ_G	35	2

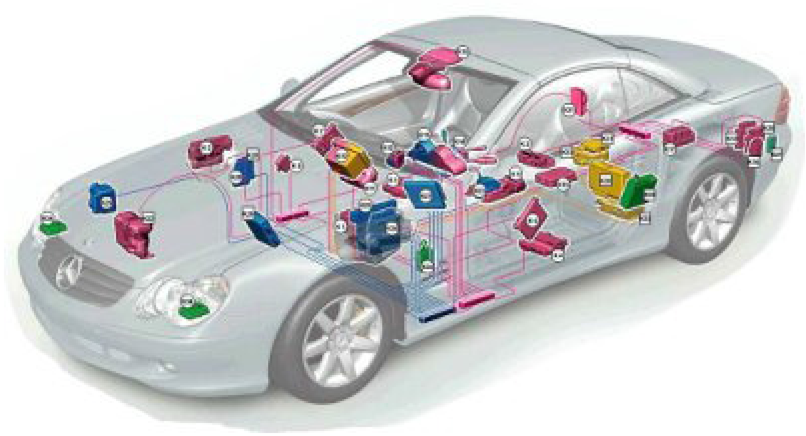


Embedded Real-Time Systems

- An embedded system is a computer system designed for specific control functions within a larger system

  (**A** is embedded into **B** for control)

- Often with such systems there are constraints such as deadlines, memory, power, size, etc.



Embedded Real-Time Systems

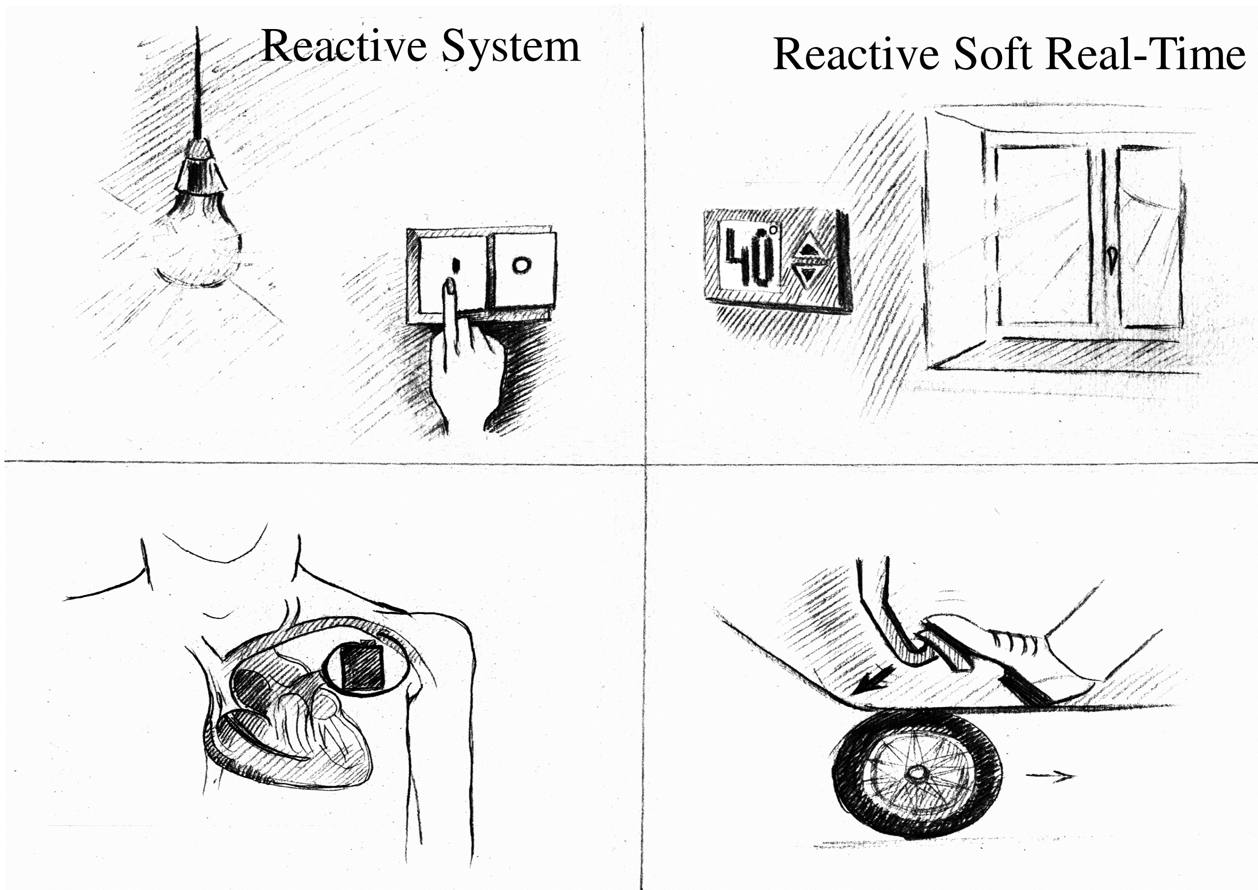
- **Real-time systems (RTS)** are reactive systems that are required to respond to an environment in a bounded amount of time.
- **Functional reactive systems (FRS)**
- **Cyber-physical systems (CPS)**
  - Challenges
    - Complexity
    - Reliability
      - Fault-tolerant design
      - Meeting deadlines (Response Time Analysis (RTA))
    - Security/Privacy

Functional Reactive Systems (FRS)

Systems that react to the environment being monitored and controlled in a timely fashion using functional (reactive) programming are known as Functional Reactive Systems (FRS).

These systems can range from small devices (which are not a CPS) to distributed and complex components (similar to a CPS).

Functional Reactive Systems (FRS)



Reactive Hard Real-Time System

Reactive Hard Real-Time System

Cyber-Physical Systems (CPS)

- Systematic integration of computation/information processing and physical processes and devices.
- Communication and sensing are components of CPS

Cyber-Physical Systems (CPS)

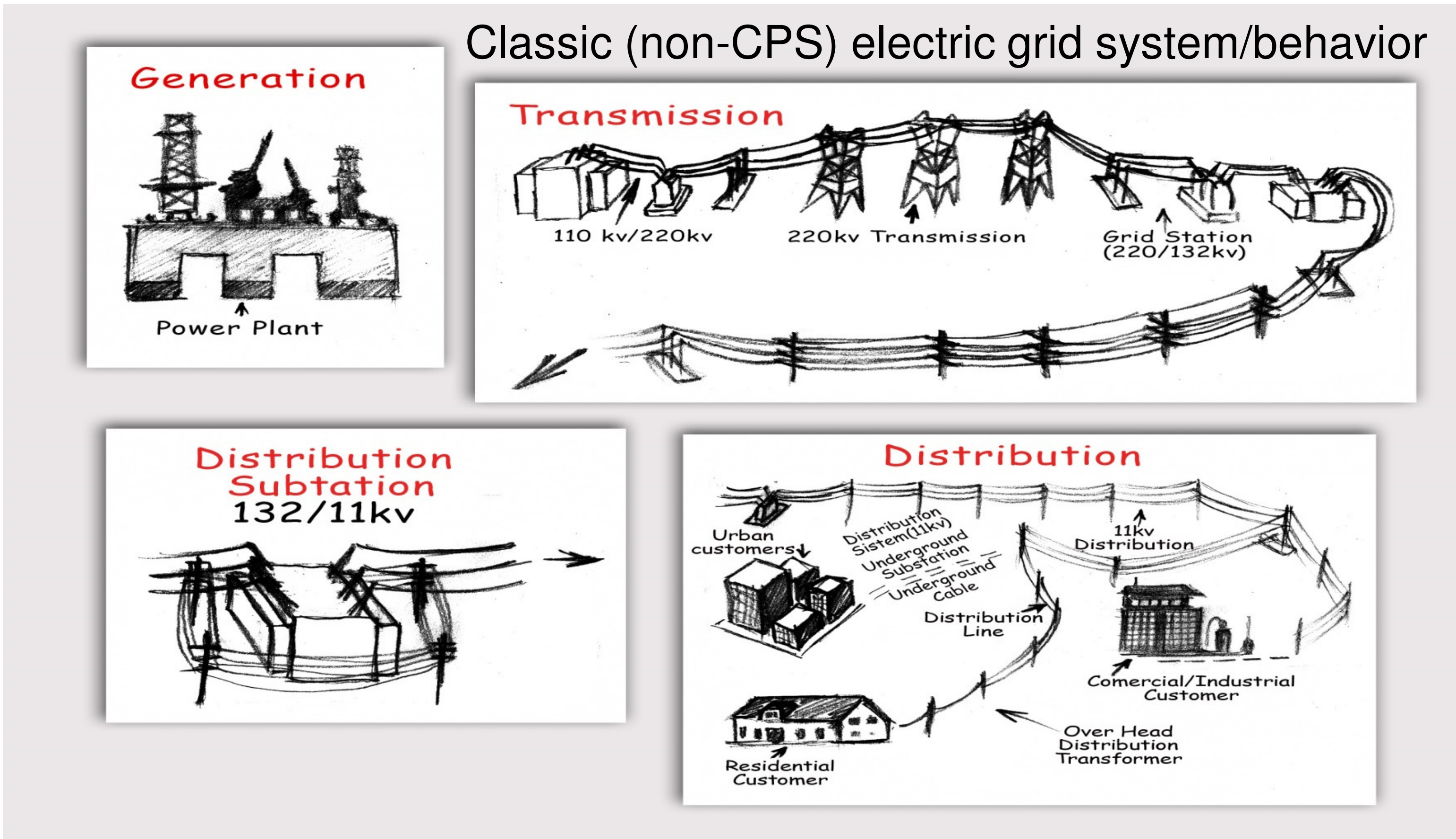
The current set of tools available for analysis cannot handle the complexity of CPS and thus are unable to predict system behavior with high degree of accuracy.

The consequences of these shortcomings:

Consider the electric power grid -- Massive failures leading to blackouts can be triggered by minor events.

CS@UH

Cyber-Physical Systems (CPS)



Classic (non-CPS) electric grid system/behavior

CS@UH

Cyber-Physical Systems (CPS)

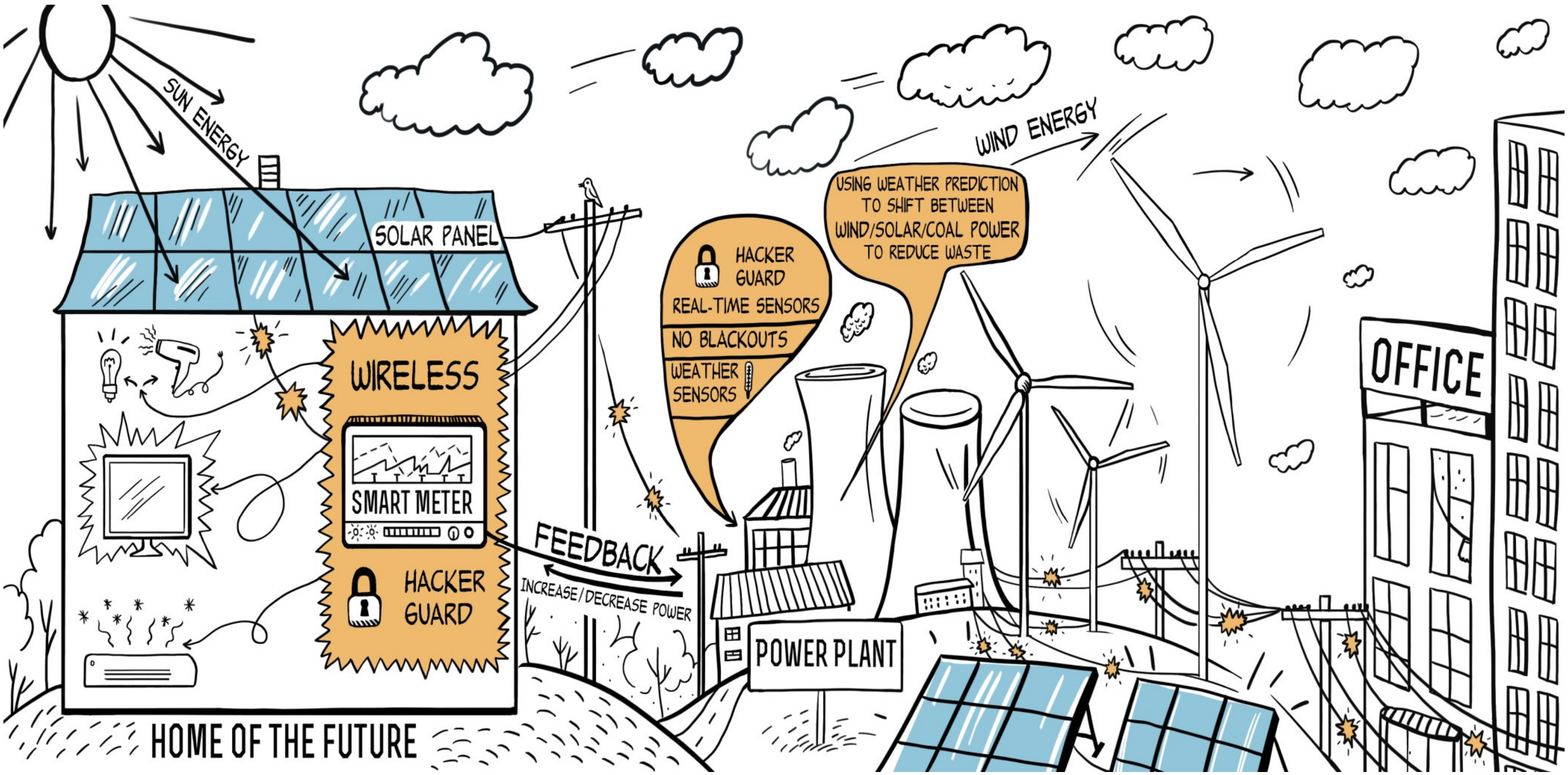
- In a CPS, wireless/wired smart meters measuring real-time electricity usage and historical data (state) feedback (communication) to the generation station to better manage and distribute electricity.

- Current and predicted weather condition data can also further inform the decision-making in where to distribute electricity (very hot or very cold weather increase electricity demand).

- There is also a need to guard against intrusion into the system.

- Advocate formal verification to ensure satisfaction of safety properties.

Cyber-Physical Systems (CPS)

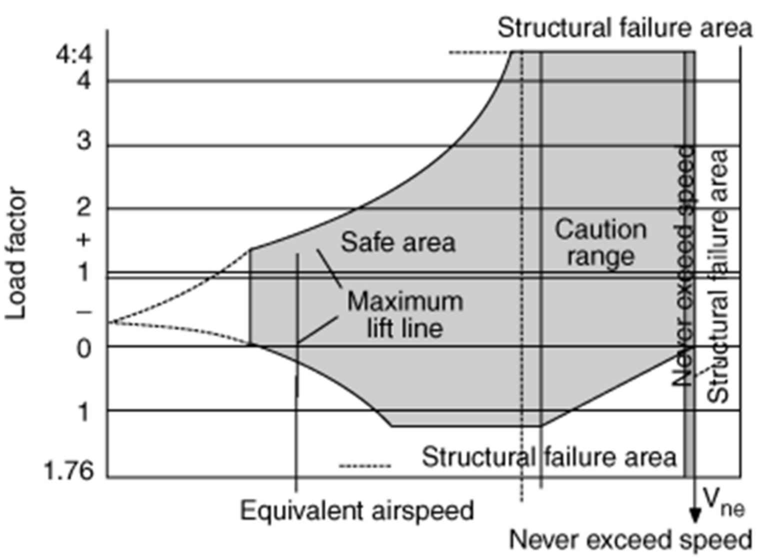


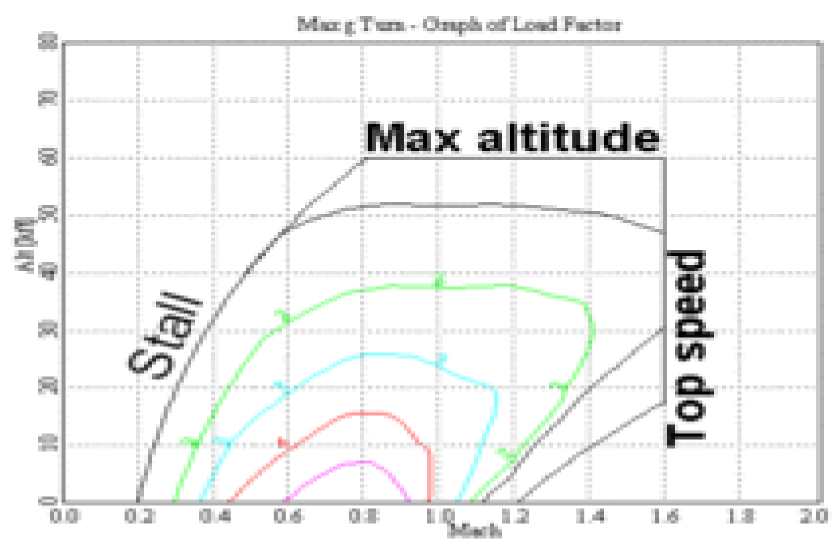
Cyber-Physical Systems (CPS)

Example 2: Imagine an airplane that refuses to crash. While preventing all possible causes of a crash is not possible, a well-designed flight control system can prevent certain causes. The systems that do this are good examples of cyber-physical systems.

Cyber-Physical Systems (CPS)

For example, some airplanes use a technique called flight envelope protection to prevent a plane from going outside its safe operating range, and prevent a pilot from causing a stall.

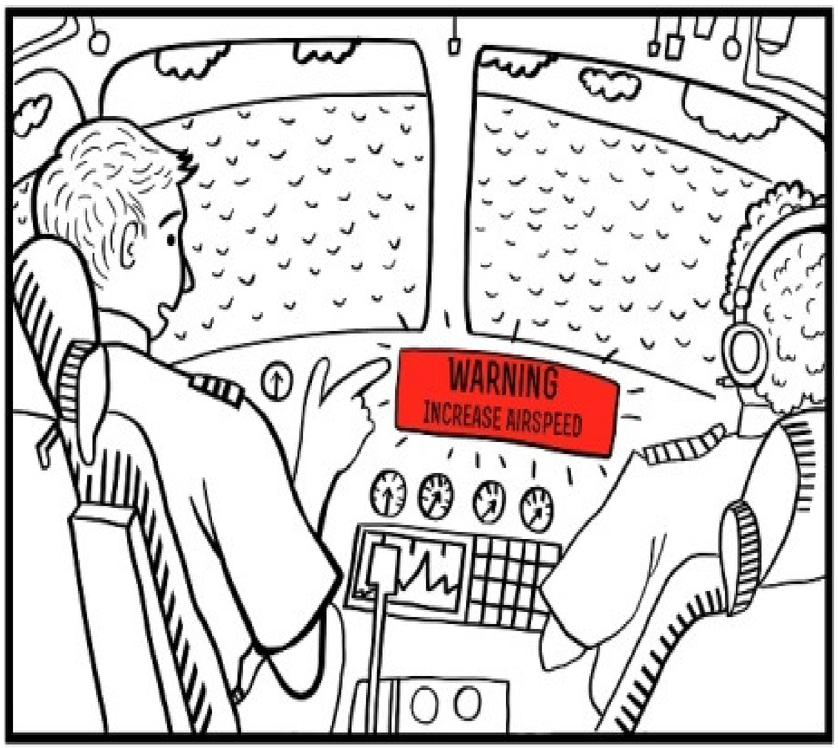


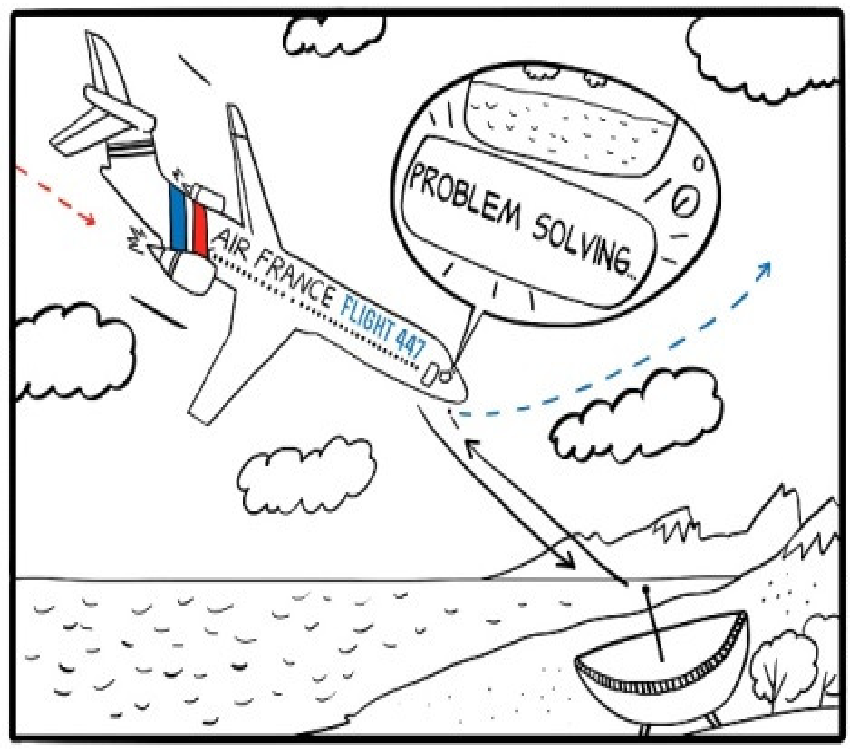


Cyber-Physical Systems (CPS)

- **The embedded control system can over-ride erroneous operation that would lead to an accident.**

http://en.wikipedia.org/wiki/Air_France_Flight_447

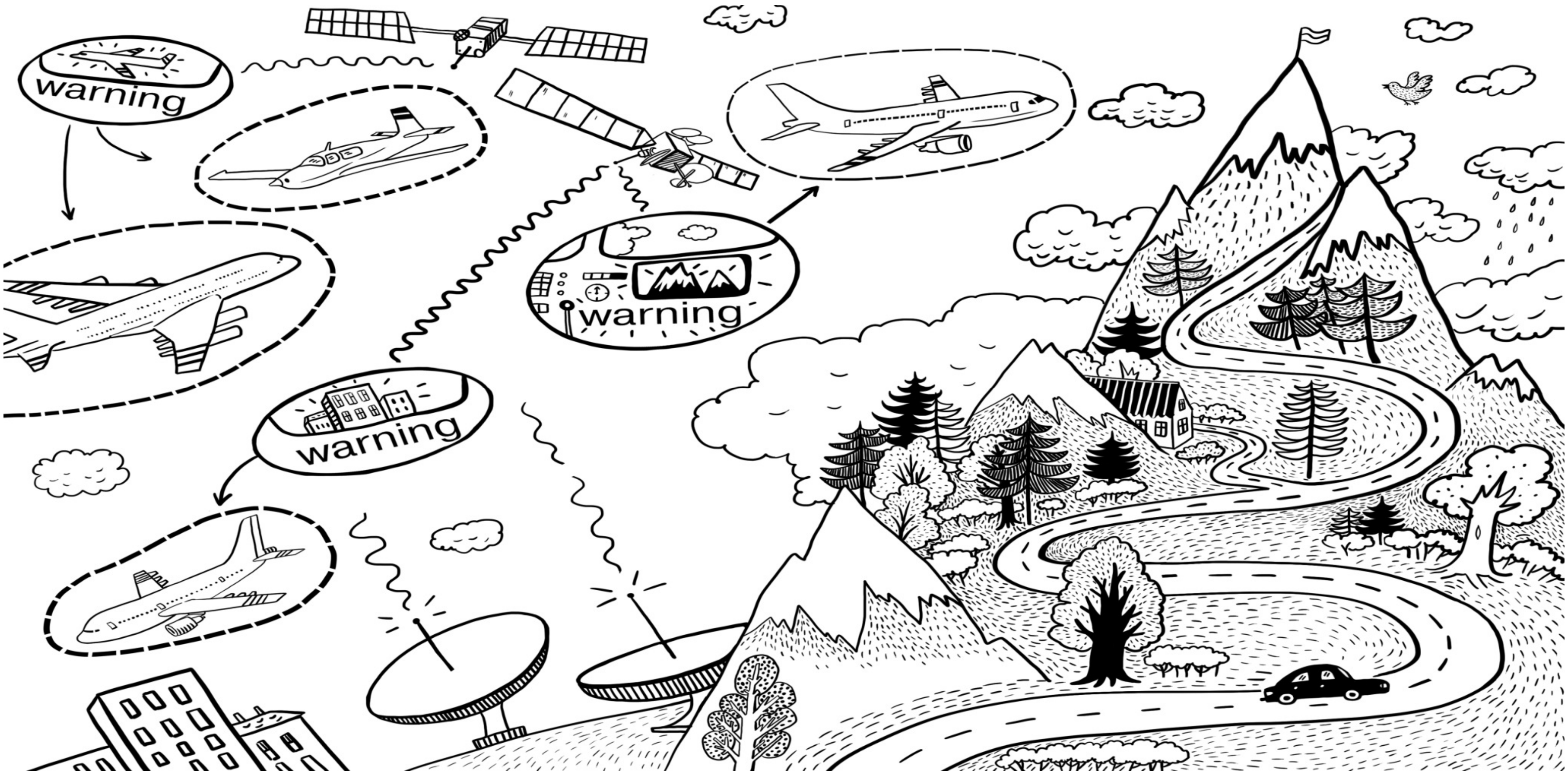




Cyber-Physical Systems (CPS)

- The concept of flight envelope protection could be extended to prevent other causes of crashes. For example, the soft walls system proposed by Prof. Edward Lee, if implemented, would track the location of the aircraft on which it is installed and prevent it from flying into obstacles such as mountains.
  - E. A. Lee, "Soft Walls - Modifying Flight Control Systems to Limit the Flight Space of Commercial Aircraft," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M01/31, 2001.

Cyber-Physical Systems (CPS)



CS@UH

Cyber-Physical Systems (CPS)

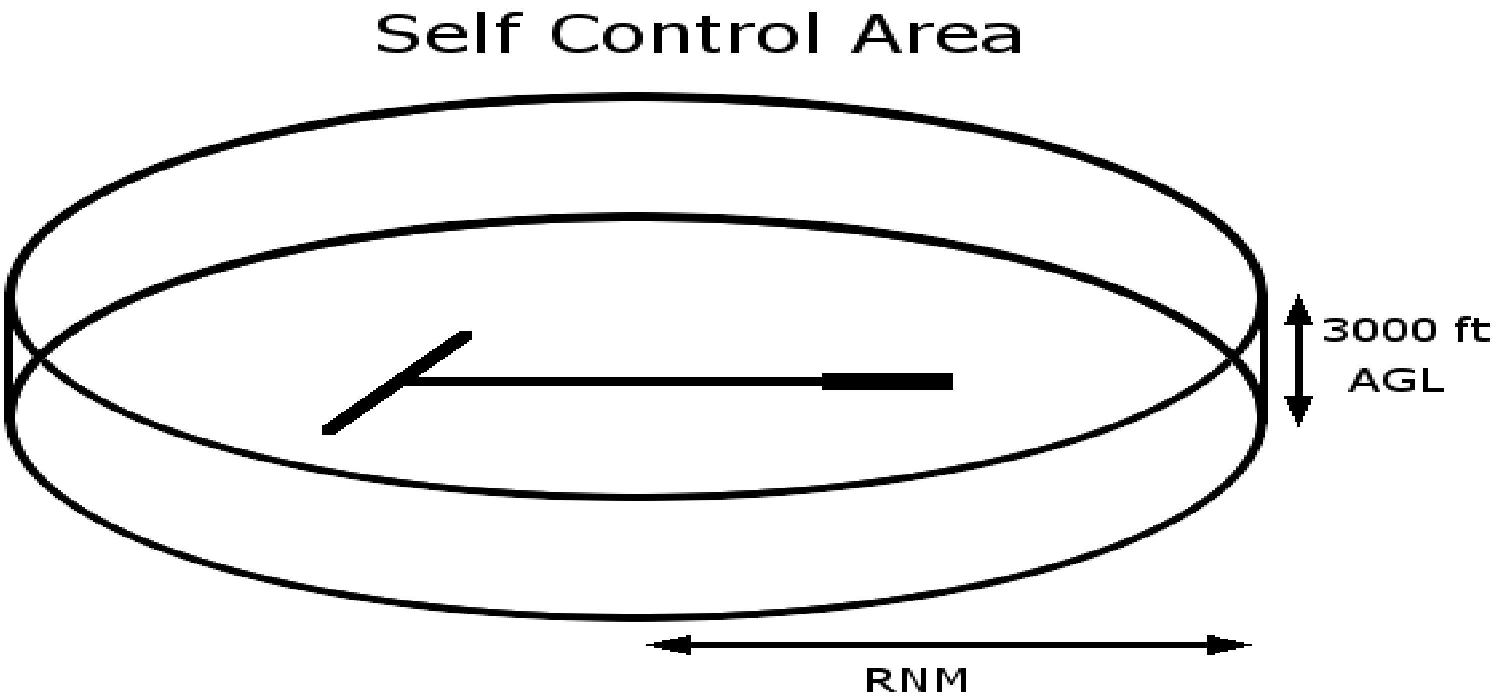
- One of the key goals in our research is to develop the core tools that can be used to facilitate the analysis, design and engineering of highly-complex systems.

- With such tools, we can ensure that these systems are reliable, predictable, efficient, secure and resilient to multiple points of failure, and hence that their operation and safety can be depended upon with a high degree of confidence.

- We advocate formal verification to ensure safety of CPS's, but their complexity requires further research in verification tools.

Cyber-Physical Systems (CPS)

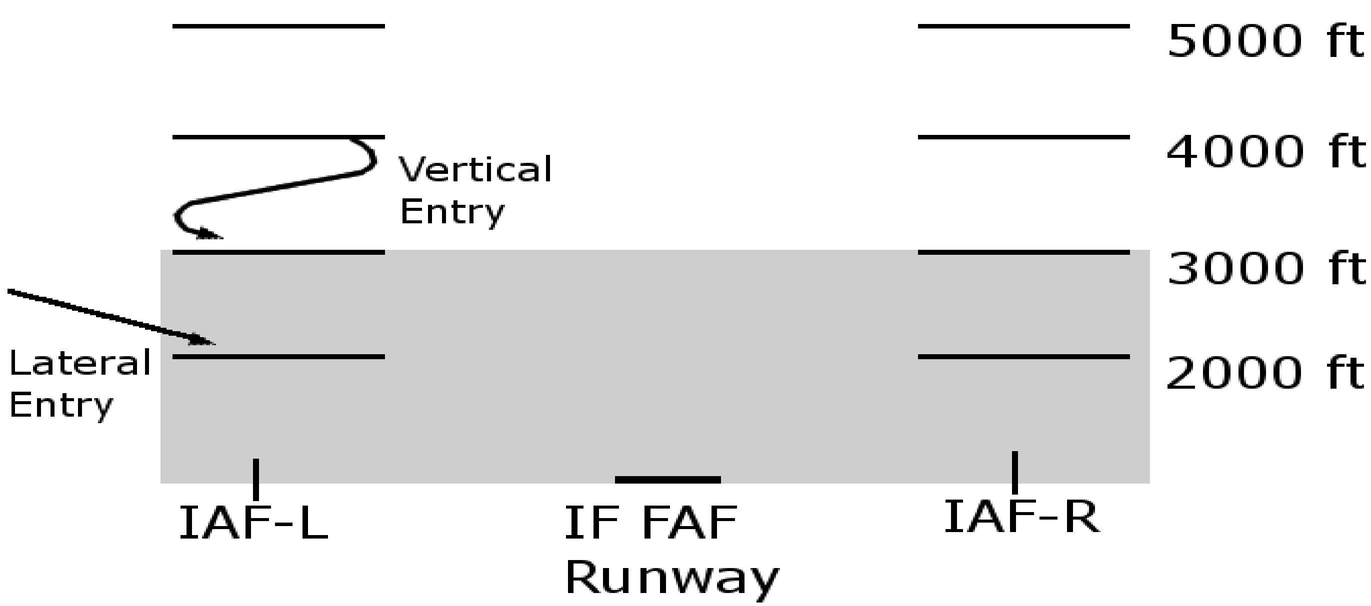
Small Aircraft Transportation System (SATS)



Self Control Area Airspace Volume

Cyber-Physical Systems (CPS)

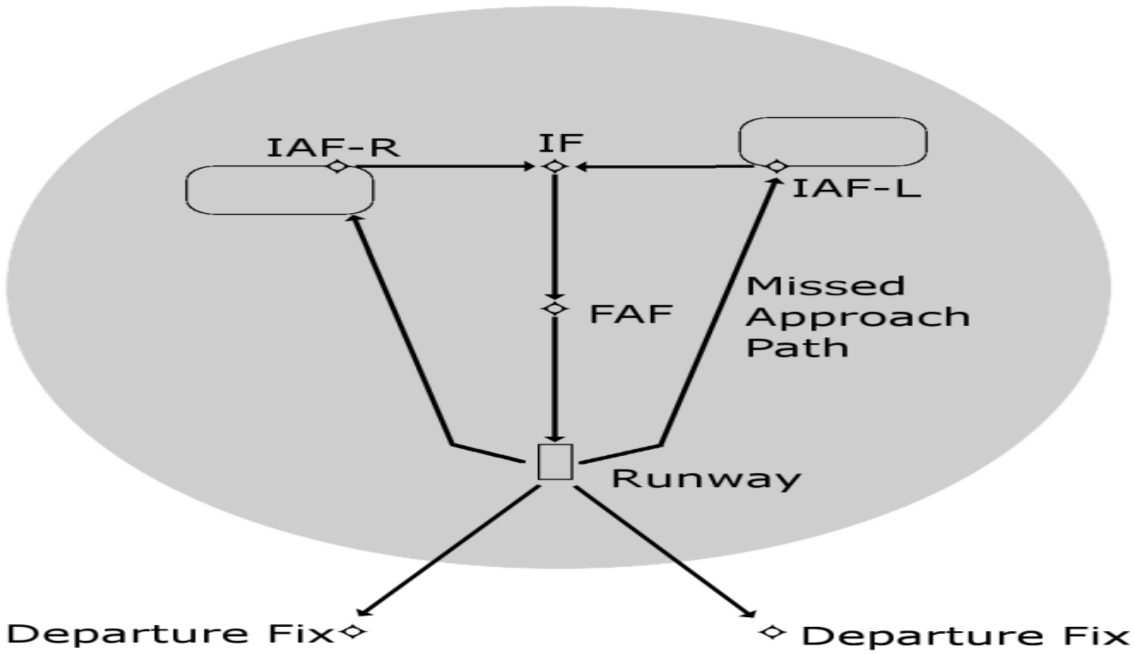
Small Aircraft Transportation System (SATS)



Side View of the SCA

Cyber-Physical Systems (CPS)

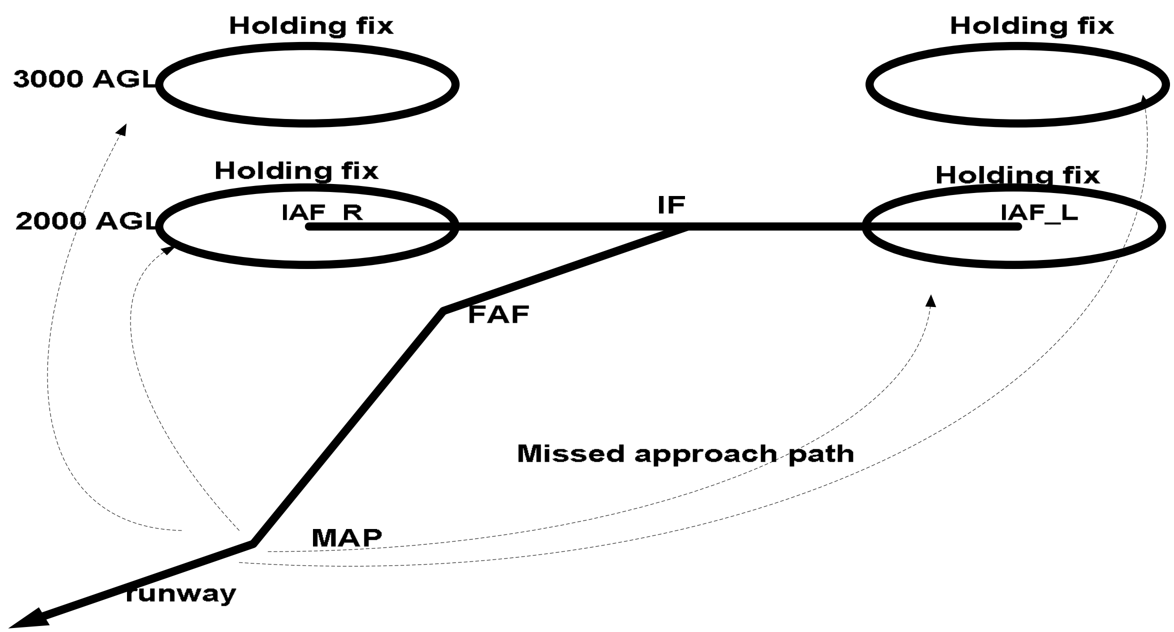
Small Aircraft Transportation System (SATS)



Top View of SCA

Cyber-Physical Systems (CPS)

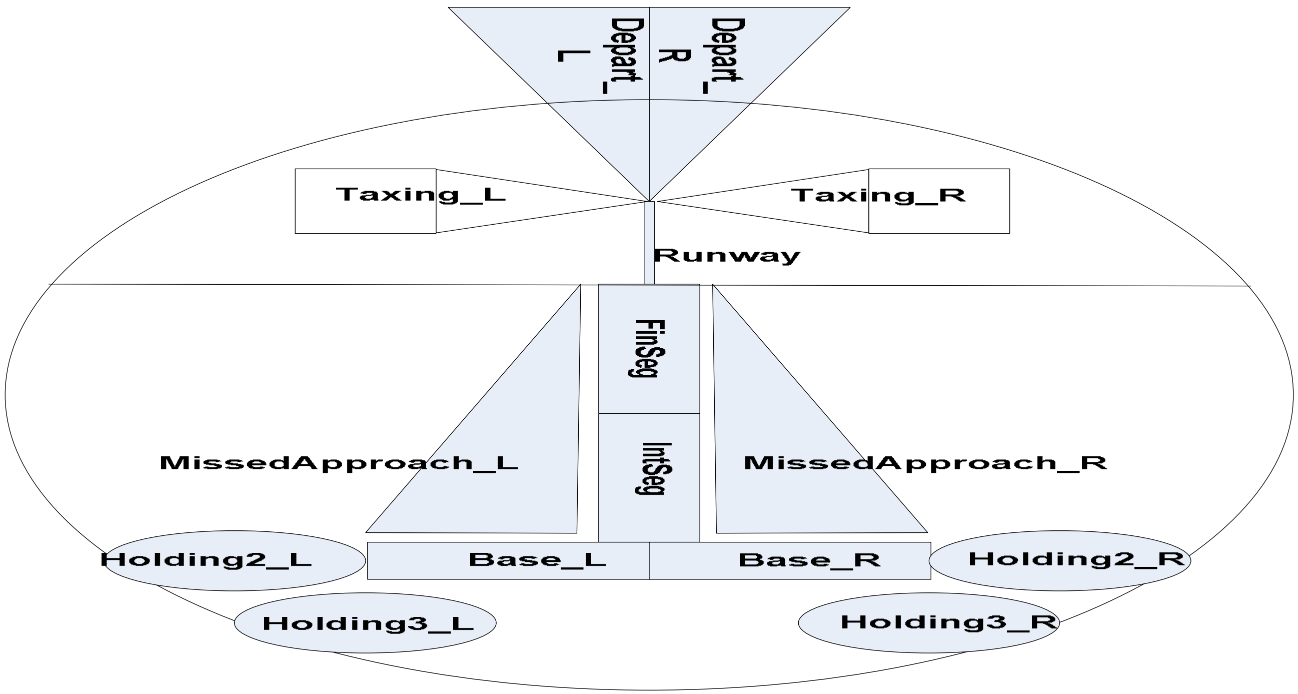
Small Aircraft Transportation System (SATS)



3-D View of the SCA

Cyber-Physical Systems (CPS)

Small Aircraft Transportation System (SATS)



Logical Zones of RTL Model

Cyber-Physical Systems (CPS)

Small Aircraft Transportation System (SATS)

$$\forall i @(\uparrow T_1\_ReqSeqNum, i) < @(T_1\_Approach\_IAF2\_L, i) - 5 \wedge$$
$$@(\downarrow T_1\_ReqSeqNum, i) < @(T_1\_Approach\_IAF2\_L, i) \wedge$$
$$@(\uparrow T_1\_ReqSeqNum, i) < @(\uparrow T_2\_ReqSeqNum, i) \wedge$$
$$@(\uparrow T_2\_ReqSeqNum, i) < @(T_2\_Approach\_IAF2\_R, i) - 5 \wedge$$
$$@(\downarrow T_2\_ReqSeqNum, i) < (T_2\_Approach\_IAF2\_R, i) \wedge$$
$$@(\uparrow T_2\_ReqSeqNum, i) < @(\uparrow T_3\_ReqSeqNum, i) \wedge$$
$$@(\uparrow T_3\_ReqSeqNum, i) < @(T_3\_Approach\_IAF3\_L, i) - 5 \wedge$$
$$@(\downarrow T_3\_ReqSeqNum, i) < @(T_3\_Approach\_IAF3\_L, i) \wedge$$
$$@(\uparrow T_3\_ReqSeqNum, i) < @(\uparrow T_4\_ReqSeqNum, i) \wedge$$
$$@(\uparrow T_4\_ReqSeqNum, i) < @(T_4\_Approach\_IAF3\_R, i) - 5 \wedge$$
$$@(\downarrow T_4\_ReqSeqNum, i) < (T_4\_Approach\_IAF3\_R, i)$$

- We will introduce RTL-based formal verification later in the tutorial.

Functional Reactive Programming

- Priority-based Functional Reactive Programming (P-FRP)
- P-FRP provides real-time guarantees using static priority assignment
- Higher-priority tasks preempt lower-priority ones; preempted tasks are aborted
- Multi-version commit model of execution
- Atomic execution – “all or nothing” proposition
- Execution different from ‘standard’ models

Other Examples of Functional Programming (FP) Languages:

- Haskell
- Atom - Domain Specific Language in Haskell
- Erlang - Developed at Ericsson for programming telecommunication equipment
- Esterel - Designed for reactive programming
- F# - Developed by Microsoft; available as a commercial platform

The Haskell Functional Programming Language

- The GHC compiler/interpreter uses a round-robin scheduler for Haskell threads
- No thread priorities yet for forkIO threads in GHC

A simple ABS example in Haskell:

```
import Control.Concurrent  -- For threading facilities: "forkIO", "newEmptyMVar", "takeMVar", and "putMVar".
import Control.Monad (forever) -- For "forever"

-- Type aliases help keep track of what values we are talking about.

type WheelSpeed    = Double  -- A "double" floating point value
type AverageSpeed = Double

-- | The ABS can either forcibly release, focibly engage, or stay neutral for each wheel.
-- The deriving clause creates the obvious Show instance for this ADT.

data BrakeSignal = Release | Engage | Neutral
   deriving Show
```

The Haskell Programming Language

```haskell
-- | Compute the average speed by dividing the sum of the list of speeds by the length.
-- fromIntegral is there to convert the result of length (Int) into a Double
-- Note, this will traverse the list twice, ineffcient for vehicles with millions of wheels.
averageSpeed :: [WheelSpeed] -> AverageSpeed
averageSpeed speeds = sum speeds / (fromIntegral $ length speeds)

-- | This algorithm may be much more complicated, but the basic idea is present.
-- Given the average speed and a particular wheel speed, check to see if we are
-- within 5 (mph, kph, m/s, whatever) of the average. If we are below the minimum
-- send the release signal to the brakes. If we are within 5, remain neutral, otherwise
-- send a signal to engage the brakes.
ecuHelper :: AverageSpeed -> WheelSpeed -> BrakeSignal
ecuHelper average speed | speed < min = Release
                        | speed < max = Neutral
                        | otherwise = Engage
  where
    min = average - 5
    max = average + 5
```

The Haskell Programming Language

```
-- A list of wheel speeds are averaged and the speed of each wheel compared to it
-- and converted into an ABS signal.

ecu :: [WheelSpeed] -> [BrakeSignal]
ecu speeds = let avgS = averageSpeed speeds
             in map (ecuHelper avgS) speeds
-- The Main Function:
-- The first thread does all printing whenever information becomes available to the ABS.
-- The second thread waits for sensor data, sends it to the ECU and stores the result in the ABS
-- The main thread waits for someone to type in a list of numbers and sends it to the "sensors".

main = do
   -- Print initial instructions and an example.
   print "Enter wheel speeds: [45,46,45,47]"
   -- Create sensors represented as a list of WheelSpeeds, i.e., Doubles.
   sensors <- newEmptyMVar
   -- Create an ABS represented as as list of BrakeSignals.
   abs <- newEmptyMVar
```

The Haskell Programming Language

```
-- This thread handles all printing to the console.
  forkIO $ forever $ do
        putStr "Enter a Speed: "
        absOutput <- takeMVar abs          -- Read ABS status
        print absOutput                    -- Print ABS status.

  -- This thread is the ABS. It reads the sensors, then processes the data and updates the ABS.
  forkIO $ forever $ do
        sensorData <- takeMVar sensors                -- Read sensors.
        let brakeCommands = ecu sensorData            -- Calculate brake response
        putMVar abs brakeCommands                     -- Update ABS status.

  -- The main thread simply waits for users to enter data which is then written to the sensors.
  forever $ do
        input <- getLine                              -- User enters a line of text
        let wheelSpeedData = read input               -- Text is read as [WheelSpeed]
        putMVar sensors wheelSpeedData                -- wheelSpeedData is written to sensors
```

The Haskell Programming Language

To run program from the command prompt
In GHCi, type

*Main> **main**
"Enter wheel speeds: [45,46,45,47]"
Enter a Speed: [**45,45,45,55**]
[Neutral,Neutral,Neutral,**Engage**]

Enter a Speed: [45,45,45,45]
[**Neutral,Neutral,Neutral,Neutral**]

Enter a Speed: [45,45,**55**,45]
[Neutral,Neutral,**Engage**,Neutral]

The idea is that you keep entering new sensor data.
The system calculates the new ABS signals to send to the vehicle.
The session should look like this:

C# and F#

Code Example in C#

```
static void ExampleOfCorrectClosure()
{
  Task[] tasks = new Task[4];
  for (int i = 0; i < 4; i++)
  {
    var tmp = i;
    tasks[i]=Task.Factory.StartNew(() =>
                Console.WriteLine(tmp));
  }
  Task.WaitAll(tasks);
}
```

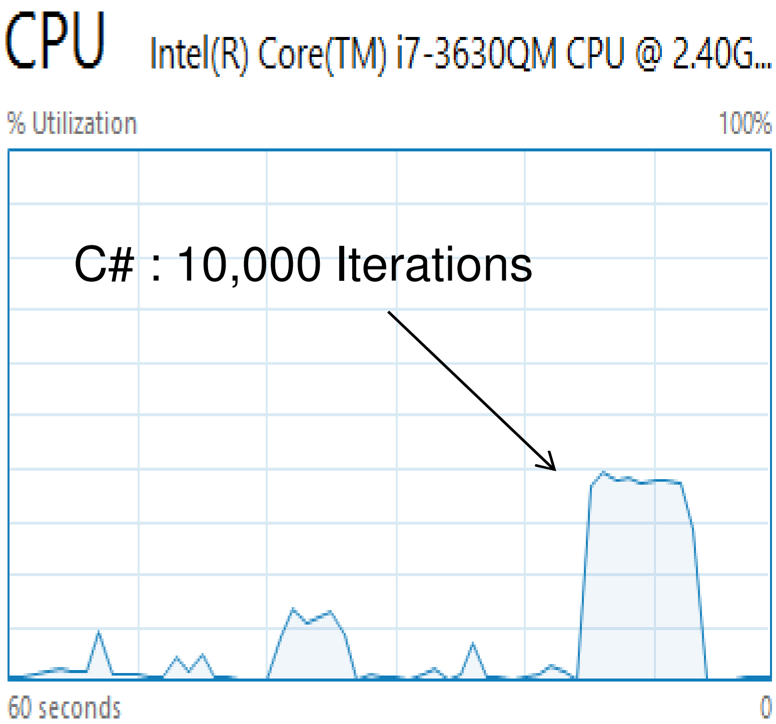
Code Example in F#

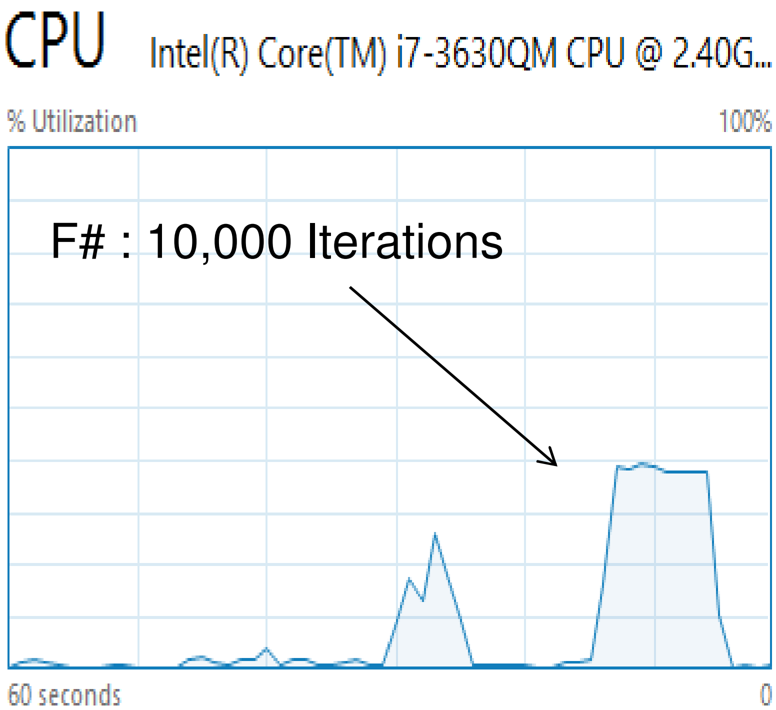
```
let main() =

  let tasks = Array.zeroCreate 4
  for i in 0 .. 3 do
    tasks.[i] <- Task.Factory.StartNew
           (fun () -> Console.WriteLine(i))
  Task.WaitAll(tasks)

do
  Task.Factory.StartNew(main).Wait()
```

Execution Time (ns.)

C #	3905515
F #	3880312

C# and F#


CPU
Intel(R) Core(TM) i7-3630QM CPU @ 2.40G...
% Utilization
100%
C# : 10,000 Iterations
60 seconds
0


CPU
Intel(R) Core(TM) i7-3630QM CPU @ 2.40G...
% Utilization
100%
F# : 10,000 Iterations
60 seconds
0

Functional Reactive Programming (FRP)

- Functional reactive programming (FRP) is a style of functional programming where programs are inherently stateful, but automatically react to changes in state.
- FRP allows intuitive specification and formal verification of safety-critical behaviors, thus reducing the number of defects during the design phase, and the stateless nature of execution avoids the need for complex programming involving synchronization primitives.
- Therefore, the program remains an algebraic description of system state, with the task of keeping the stated (unidirectional) relationships in sync left to the *language*.

Functional Reactive Programming (FRP)

- FRP is essentially (though rarely acknowledged as such) an extension to the old idea of dataflow programming.

- A key difference is that FRP supports higher-order functions, and modern FRP systems are generally well-integrated into broader languages.

- The original (modern) FRP work was built in the context of Haskell, though major FRP systems have also been built atop many other languages.

Functional Reactive Programming (FRP)

CS@UH

- Type-safe programming language
- Discrete and Continuous aspects
- Transactional model prevents priority inversion
- Synchronization primitives not required
- Ideal for parallel execution

Basic Abstractions

- FRP divides inputs into two basic classes:
  - Behaviors or signals: Functions of time.
  - Events: Temporal sequences of discrete values.
- An FRP language must include a means of altering or replacing a program based on event occurrences - this is the basis of FRP's reactivity.
- These abstractions may be reified in an FRP language or may form the basis of other abstractions, but they must be present.

Functional Reactive Programming (FRP)

CS@UH

- Classic FRP
  - Fran (Functional Reactive Animation: Bouncing Balls)
  - Reactive
  - Reactive-banana
  - Elm
- Signal-Function FRP
  - Fruit
  - RT-FRP
  - Yampa   (Animations and Games: Space Invaders)
  - Netwire

Functional Reactive Programming (FRP)

Examples in FRAN

- Values, called behaviors, that vary over time

- As an example the following expression evaluates to an animation (i.e., an image behavior) containing a circle over a square. At time t, the circle has size *sin t*, and the square has size *cos t*.

bigger (sine time) circle

bigger (cos time) square

Functional Reactive Programming (FRP)

Examples in FRAN

- Events
- Like behaviors, events may refer to happenings in the real world (e.g., mouse button presses). For example the event describing the first left-button press after time t0 is simply *lbp t0*;
- One describing time squared being equal to 5 is just:

predicate (time^2 == 5) t0

Functional Reactive Programming (FRP)

Examples in FRAN

- Many behaviors are expressed in terms of reactions to events. But even these reactive behaviors have declarative semantics in terms of temporal composition.
- For example, a color-valued behavior that changes from **red** to **green** with each button press can be described by the following simple recurrence:

```
colorCycle  t0 =
 red      'untilB' lbp t0 *=> \t1 ->
  green  'untilb' lbp t1  *=> \t2 ->
  colorCycle t2
```

Functional Reactive Programming (FRP)

Examples in FRAN and RT-FRP

- At the moment of an event occurrence, it is good to take a **snapshot** of a behavior's value. For example, the behavior:

b1 untilB (lbp t0 snapshot (sin time)) => X(e,y), b2

- Grabs the sine of the time at which the left button is pressed, binds it to *y*, and continues with behavior *b2*, which depends on *y*.

Functional Reactive Programming (FRP)

Examples in FRAN and RT-FRP

- The following computes the difference between the current time and the time at the previous execution step:

```
let snapshot t0 <- time in
        let snapshot t1 <- delay 0 time in ext (t0 -t1)
```

Functional Reactive Programming (FRP)

CS@UH

Netwire

Netwire is a library for functional reactive programming

This language lets you express **reactive systems**, which means systems that change over time.

It shares the basic concept with **Yampa**

The Haskell **Cabal** is a system for building and packaging

Haskell libraries and programs.

Functional Reactive Programming (FRP)

CS@UH

A simple ABS example with Netwire

Everything above "main" is identical to the Haskell program example (except imports)

```
            import Control.Wire
-- Print initial instructions and an example.
   print "Enter wheel speeds: [45,46,45,47]"
      let loop w' session' = do                        --This is the main loop
      (mx, w, session) <- stepSession w' session' ()    --Step forward in session
      case mx of                                       --Check for success
         Left ex -> putStrLn ("Error: " ++ show ex)    --If failure, print why
         Right x -> putStrLn ("Success: " ++ show x)   --If success, print result
      loop w session                                   --Loop again
   loop ( absControl . sensors ) clockSession          --Create the connection between
                                       -               --the sensors and absControl and
                                                       --create a clock session
```

Functional Reactive Programming (FRP)

-- This WIRE waits for input which is checked for validity then written to the sensors.
-- This wire also handles printing to the console.

```
sensors :: Wire String IO () [WheelSpeed]
sensors = mkStateM [ ] (\_ (_,s) -> do
   putStr "Enter a Speed: "                          -- Print prompt
   r <- getLine                                      -- User enters a line of text
   let r2 = case maybeRead r                         -- Text is read as [WheelSpeed]
            of Just x -> (Right x,x)
               Nothing -> (Left "Bad Input",s)       -- If read fails, inhibit the wire.
   return r2)                                        -- wheelSpeedData is sent to sensors
```

Functional Reactive Programming (FRP)

```
-- This WIRE is the ABS.
-- It reads the sensors, then processes the data and updates the ABS.
absControl :: Wire String IO [WheelSpeed] [BrakeSignal]
absControl = mkPure (\_ sensorData -> (Right (ecu sensorData),absControl))

-- Helper function for read.
maybeRead :: Read a => String -> Maybe a
maybeRead s = case reads s of
   [(x, "")] -> Just x
   _         -> Nothing
```

Functional Reactive Programming (FRP)

- **Weaknesses**
  - FRP is still relatively new and the design space is still being explored.

- **Strengths**
  - FRP makes writing reactive programs easier to reason about and to avoid common errors

  - It is easier to expand and create new behaviors. Once the program becomes more complex, **forkIO** and multiple threads might start interfering with each other, or there would be odd interleaving, blocking, or other bad concurrency behavior.

FRP is still Haskell. It is just a different style.

Functional Reactive Programming (FRP)

- Using FRP makes the controllers (the computational components of CPS) more amenable to analysis and verification.
- We can treat components (programed in FRP) as mathematical functions, which can be composed and synthesized to form a much larger, complex system.
- More resistant to faults since there are no intermediate states. They can be connected and composed more easily.
- With procedural programs, there are more uncertainties, for example, intermediate states if faults/interruptions occur that need to be specified/modeled, making developing a CPS with guaranteed safety and response much more complex and potentially intractable.
- In the electric grid example, different generating stations have control components which analyze real-time data from smart meters, weather data, and industrial plants' energy usage to determine optimal or near-optimal generation and distribution of electricity.

Priority-based FRP (P-FRP)

- P-FRP aims to improve the programming of reactive real-time systems.
  - Supports assignment of different priorities to events
  - Benefits of using P-FRP over the imperative styles
    - P-FRP allows the programmer to intuitively describe safety-critical behaviors of the system, thus lowering the chance of introducing bugs in the design phase.
    - Its stateless nature of execution does not require the use of synchronization primitives like mutexes and semaphores, thus reducing the complexity in programming.

Priority-based FRP (P-FRP)

- **To preserve data consistency, shared resources must be accessed in mutual exclusion:**

Priority-based FRP (P-FRP)

- **However, mutual exclusion introduces extra delays:**

Priority-based FRP (P-FRP)

Example: The Car Controller

* C = worst case execution time
* T = (sampling) period = D (deadline)

- Speed Measurement: C=4ms, T=20ms, D=20ms
- ABS control: C=10ms,T=40ms, D=40ms
- Fuel injection: C=40ms,T=80ms, D=80ms
- Other software with soft deadlines, audio, air condition, etc.

Try any method to schedule the tasks

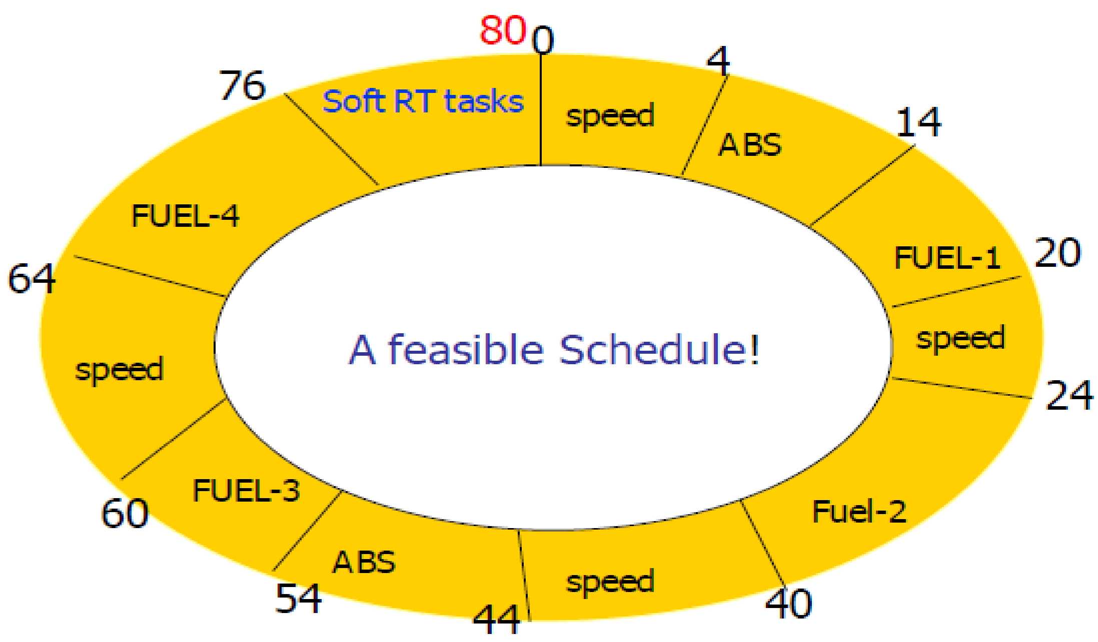
Priority-based FRP (P-FRP)

Static cyclic scheduling: + and –

- Deterministic: predictable (+)
- Easy to implement (+)
- Inflexible (-)
  - Difficult to modify, e.g., adding another task
  - Difficult to handle external events
- The table can be huge (-)
  - Huge memory-usage
  - Difficult to construct the time table

Priority-based FRP (P-FRP)

The Car Controller (Time table constructed with EDF)



Can use the Stack Resource Policy (SRP) or the Priority Ceiling Protocol (PCP) for concurrency control.

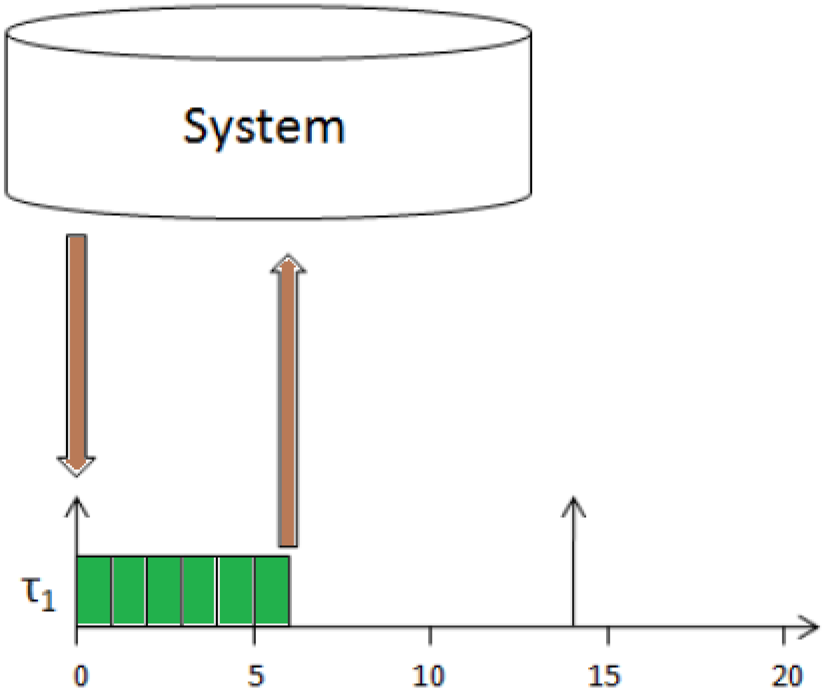
- Inheritance algorithms are complicated and difficult to program correctly.

Priority-based FRP (P-FRP)

- **In P-FRP, the scheduling model is called** Abort-and-Restart **(ANR)**
  - **Copy and restore operations**
    - **To allow for correct restarting of handlers, compilation is extended to generate statements that store variables modified in an event handler into fresh *temporary* (or *scratch*) variables in the beginning of the handler while interrupts are turned off, and to restore variables from the temporary variables at the end of the handler while interrupts are turned off.**

Priority-based FRP (P-FRP)

τ_1 starts at 0 and it copies a set of data from the system. After six ticks, its work is done and then it restore the updated data into the system.



Copy-and-Restore Operation

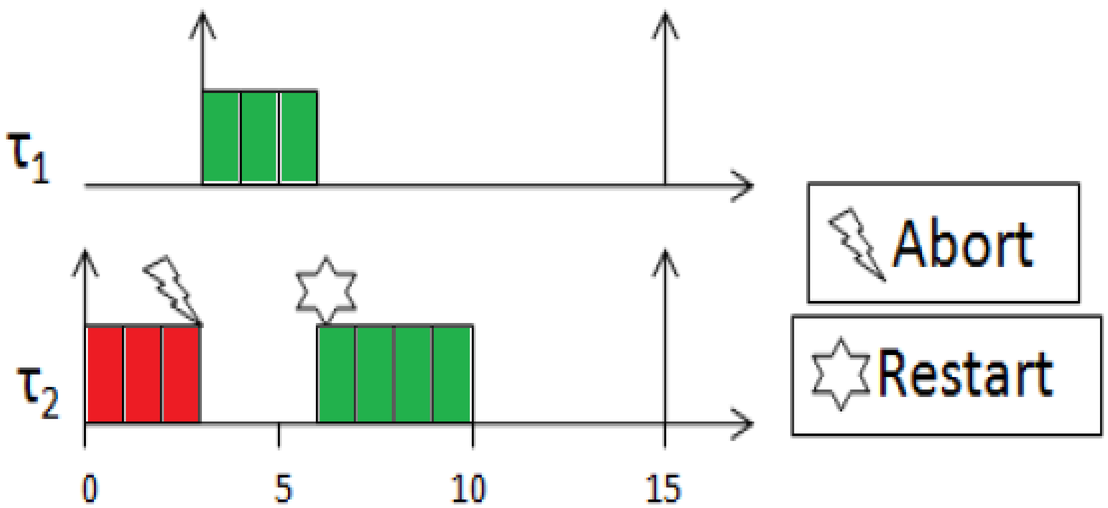
Priority-based FRP (P-FRP)

- **The Abort-and-Restart (ANR) Scheduling Model**
  - The idea of the ANR model is that a lower-priority task is aborted when a higher priority task arrives into the system. Once the higher-priority task is done, the lower priority task restarts as new.

Priority-based FRP (P-FRP)

Task	Period	WCET
τ_1	12	3
τ_2	15	4

(τ_1 has the highest priority)



Priority-based FRP (P-FRP)

- **Advantages of Abort-and-Restart (ANR)**
  - A simpler programming model
  - Tasks execute atomically so no task is blocked by another task
    - The priority inversion problem is removed
    - No overheads caused by priority inheritance
    - Closer adherence to priority scheduling

Priority-based FRP (P-FRP)

```
-- Use Software Transactional Memory (STM) for shared and Private data
type EventAction = (TVar Integer,TVar Integer) -> STM (Integer,Integer)

type EventQueue = TChan (Integer,Event)                -- The queue for the events.

data Event = MkEvent {  eventPr        :: Int          -- Priority
                       ,executionTime  :: Int          -- Execution time
                       ,eventAction    :: EventAction  -- Workload function
                       ,eventName      :: String       -- Name of the event
                       ,releaseTime    :: Integer      -- Creation time
                       ,shortName      :: String}      -- Short nickname
```

Priority-based FRP (P-FRP)

```
event1 = MkEvent {eventPr = 4, executionTime = 1, eventAction = e1,
                  eventName = "Event1", releaseTime = 0, shortName = "e1: "}

event2 = MkEvent {eventPr = 5, executionTime = 2, eventAction = e2,
                  eventName = "Event2", releaseTime = 0, shortName = "e2: " }

event3 = MkEvent {eventPr = 7, executionTime = 2, eventAction = e3,
                  eventName = "Event3", releaseTime = 0, shortName = "e3: " }
```

Priority-based FRP (P-FRP)

```
-- Create and launch three events

ee1 <- forkIO $ eventEmitter launcherMessages event1
ee2 <- forkIO $ eventEmitter launcherMessages event2
ee3 <- forkIO $ eventEmitter launcherMessages event3
```

Priority-based FRP (P-FRP)

```
do -- schedular: new event has arrived

  case currThreadInfo of

    Nothing ->
        do -- no thread is running => launch the event
          newThreadId <- runEvent launcherMessages eventStates newEv
          launcherLoop stats' $ Just (newEv, newThreadId, tm) -- go wait for the next event
```

Priority-based FRP (P-FRP)

```
Just (currEv, threadId, startTime) -> -- some thread is already running
  let
    currCV   = getCV currEv tm (Just startTime)
    currExpl = explainCV currEv tm (Just startTime)
  in
    if (newCV >= currCV) -- test if newly arrived thread has a lower priority
    then atomically (writeTChan eventQueue (newCV,newEv)) -- enqueue the new event
              >> launcherLoop stats'{queued = queued stats' + 1} currThreadInfo

    else do -- otherwise, abort the running thread

          throwTo threadId $ StringException  "" -- stop the running thread

          -- because state modification is run in the STM monad, there cannot be a corruption
          -- of the state. In other words, either the thread fully succeeds, or fully fails
          -- This makes the handling of exceptions and resuming the process much easier.

          newThreadId <- runEvent launcherMessages eventStates newEv -- launch the new event
```

Priority-based FRP (P-FRP)

```
EventEnded ev -> -- current event completed with newState

    do -- get next event

       -- queue is not empty => launch/restart the event fetched from the queue
      (newCV, newEv) <- atomically $ readTChan eventQueue
      -- Fetching from queue
      newThreadId <- runEvent launcherMessages eventStates newEv -- launch the worker thread
```

Priority-based FRP (P-FRP)

Limited Work on Scheduling and Schedulability Analysis

- While there is an extensive understanding of the theory and proof-carrying capability of functional programs and their reactive versions, relatively little work is available on the scheduling of primitives in the corresponding imperative code.
- Also, performance studies of the computational platforms on which these functional programs execute are mostly absent.

Priority-based FRP (P-FRP)

- The worst-case response time of a task is the length of the longest interval from a release of that task till its completion.
- With ANR, interference from higher-priority tasks induces both an interference cost and an abort cost on the response time of the preempted lower-priority task.
- Current focus is on response time analysis with abstract memory and I/O access times. Next challenges include accounting for precise memory and I/O access times.

Priority-based FRP (P-FRP)

- Response time analysis is an **exact** schedulability test to calculate the worst-case response time of a task which includes the time of interference from other higher priority tasks and blocking from lower priority tasks.

- RTA is **not** exact unless **blocking** is exact - which it is not. If the worst-case response time of a task is longer than its deadline (***D***), it means the task will not meet its deadline. The opposite situation is that if the worst-case response time of the task is less than or equal to its deadline, the task will meet its deadline.

- The analysis can be applied for ***D = T*** (task's period), ***D < T***, or ***D > T***.

Priority-based FRP (P-FRP)

Response time Analysis for ANR

- For the highest-priority task, its worst response time will be equal to its own computation time, that is $R = C$.

- If task j has the highest arrival rate, then the execution time of a task i cannot exceed $T_j - C_j$ or task i will suffer **interference** (I) and **aborts** (α). So for a general task i :

$$R_i = C_i + I_i + \alpha_i$$

Priority-based FRP (P-FRP)

Interference Cost

- If the execution time of some task ***i*** exceeds $T_j - C_j$, then task ***i*** will **never** be able to complete execution.

- A simple expression for obtaining this Interference Cost is using the ceiling function:

$$I_i = \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

Priority-based FRP (P-FRP)

Maximum Interference

- Each task of higher-priority is interfering with task ***i**, and so:*

$$I_i = \left\lceil \frac{R_i}{T_{i+1}} \right\rceil \cdot C_{i+1} + \left\lceil \frac{R_i}{T_{i+2}} \right\rceil \cdot C_{i+2} + \ldots + \left\lceil \frac{R_i}{T_n} \right\rceil \cdot C_n$$

- This gives us the following equation:

$$I_i = \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

Priority-based FRP (P-FRP)

Maximum Abort Costs

- Each higher-priority task is interfering with task ***i***, so the maximum Abort Costs are as follows*:*

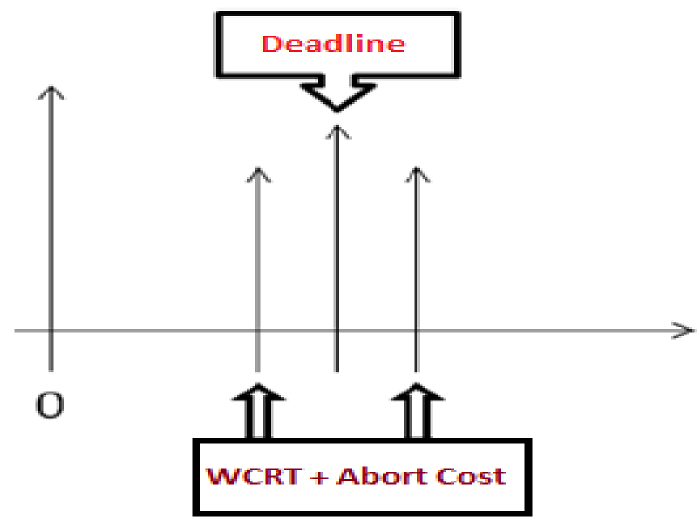
$$\alpha_i = \sum_{j=i+1}^{N} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \max_{k=i}^{j-1} C_k$$

- C_k is the maximum execution time between *i* and the highest-priority task.

Priority-based FRP (P-FRP)

Maximum Abort Costs

- The maximum abort cost equation is sensible and simple but overly pessimistic. Therefore, the test is said to be **sufficient** but not **necessary**.



Priority-based FRP (P-FRP)

- Abort-and-Restart with a limit on the number of aborts

$$R_i \leq C_i + I_i + B_i + \alpha_i$$

$$I_i = \sum_{j=i+1}^{n} min(MaxNumberAborts, \left\lceil \frac{R_i}{T_j} \right\rceil) \cdot C_j$$

$$B_i = \max_{j \in LowerPriority(i))} C_j \quad (Blocking\ Costs)$$

$$\alpha_i = MaxNumberAborts \cdot \sum_{j=i}^{n-1} C_j$$

Priority-based FRP (P-FRP) Example

Antilock braking system in a car is a simple example of an embedded hard real-time system with real-time constraints.

The ABS is expected to release a vehicle's brakes, preventing dangerous wheel locking, in a predictably short time frame.

ABS uses a kind of an Abort-and-Restart Scheme.

Kaleb R. Christoffersen and Albert M. K. Cheng, ``Model-Based Design: Anti-lock Brake System with Priority-Based Functional Reactive Programming,'' submitted to RTSS WIP 2013.

Priority-based FRP (P-FRP)

Anti-Lock Brake Types

ABS uses different schemes depending on the type of brakes in use.

- **Four-channel, four-sensor ABS** (the best scheme) - there is a speed sensor on all 4 wheels and a separate valve for all four wheels. With this setup, the controller monitors each wheel individually to make sure it is achieving maximum braking force.

- **Three-channel, three-sensor ABS -** this scheme found often on pickup trucks. It has a speed sensor and a valve for each of the front wheels, with one valve and one sensor for both rear wheels.

- **One-channel, one-sensor ABS -** this system found also often on pickup trucks with rear-wheel ABS. It has one valve, which controls both rear wheels, and one speed sensor.

Priority-based FRP (P-FRP)

Example: ABS Controller

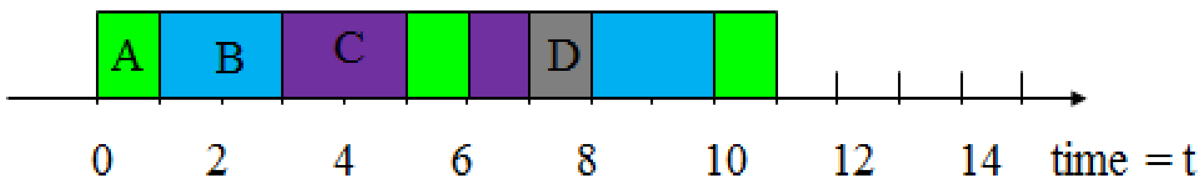
- Activities of an ABS control system
  1. C = worst case execution time
  2. T = (sampling) period = D (deadline)

- (A) Car speed measurement: C= 1 ms, T= 5 ms
- (B) Wheel speed measurement: C= 2 ms,T=8 ms
- (C) Analysis and computation task : C= 3 ms,T=20 ms
- (D) Brakes (Abort (release) /Retry (pressure)) : C= 1 ms,T=25 ms

Priority-based FRP (P-FRP)

Example: ABS Controller - with RM scheduling

- The shortest repeating cycle / LCM = 200 ms



A's response time = 1 (Same as its own Computation Time)

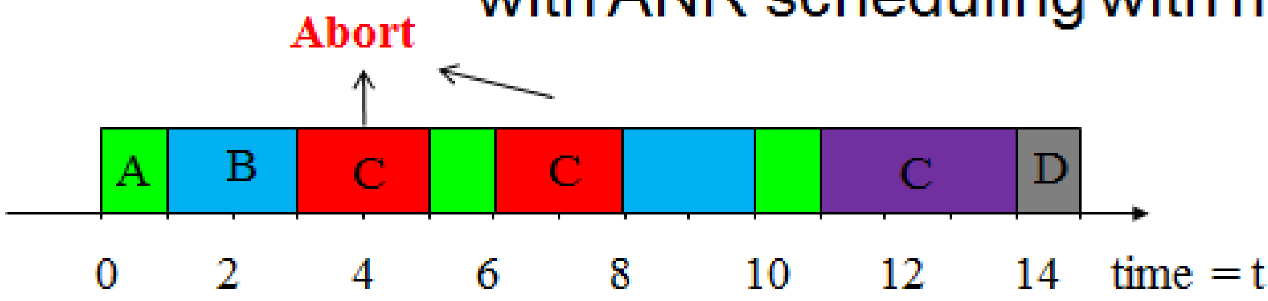
B's = 2 + 1 (time to execute task A) = 3

C's = 3 + 1 (A's) + 2 (B's) = 6

Priority-based FRP (P-FRP)

Example: ABS Controller –

with ANR scheduling with max 2 aborts.



A's response time = 1 (Same as its Computation Time)

B's = 2 + 1 (time to execute task A) = 3

C's = 3 + 1 (A's) + 2 (B's) = 14 < Deadline

Priority-based FRP (P-FRP)

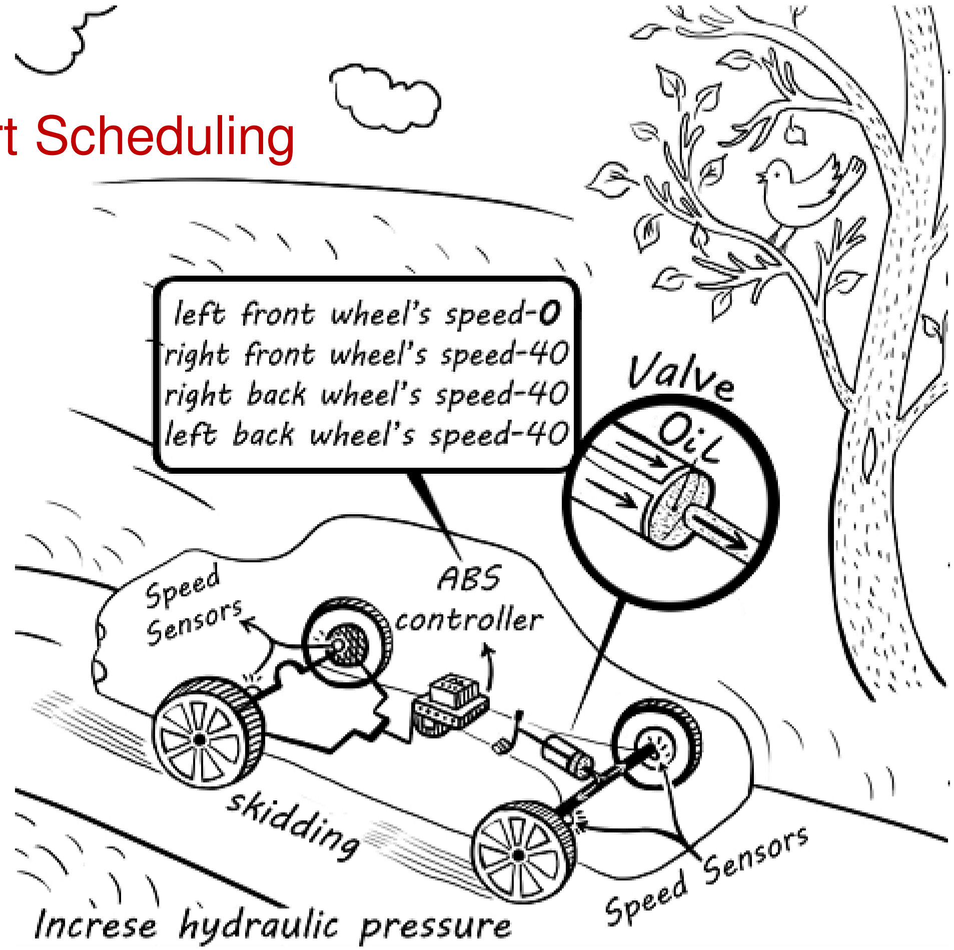
Typically ABS includes

- Electronic control unit (ECU)
- Wheel speed sensors
- At least two hydraulic valves within the brake hydraulics

- The ECU constantly monitors the rotational speed of each wheel; if it detects a wheel rotating significantly slower than the others, a condition indicative of impending wheel lock, it actuates the valves to reduce hydraulic pressure to the brake at the affected wheel, thus reducing the braking force on that wheel; the wheel then turns faster.

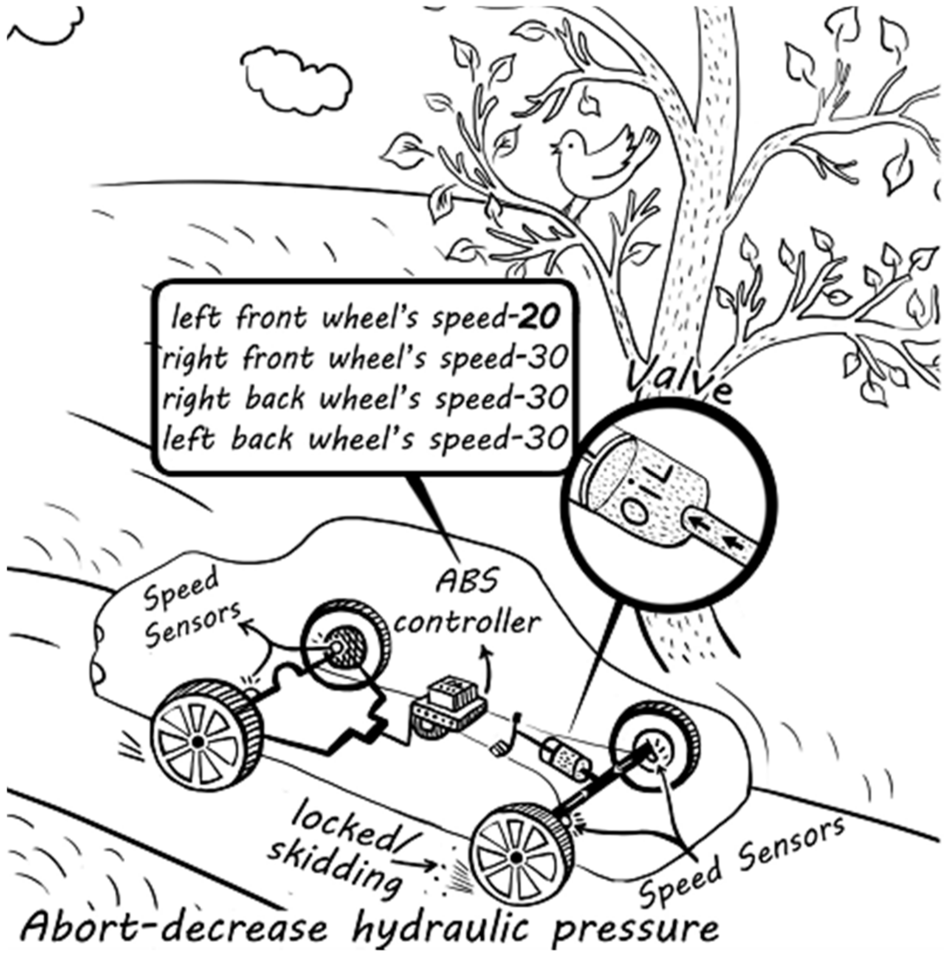
Priority-based FRP (P-FRP)

Abort-and-Restart Scheduling

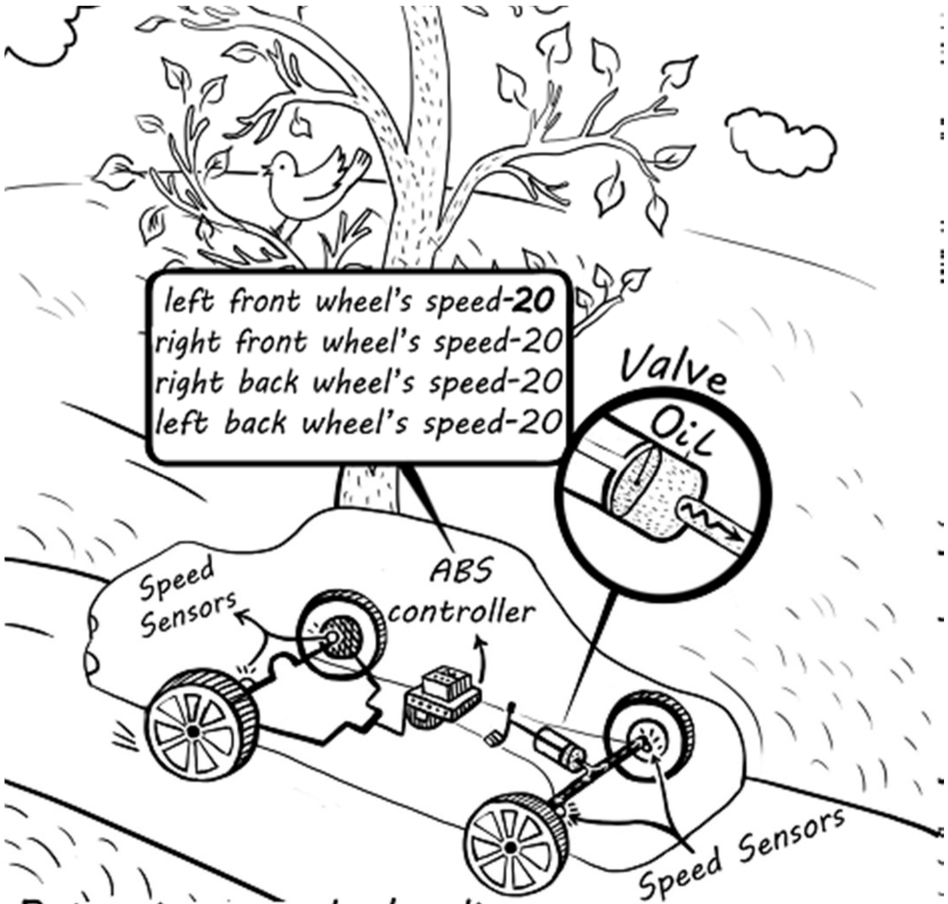


CS@UH

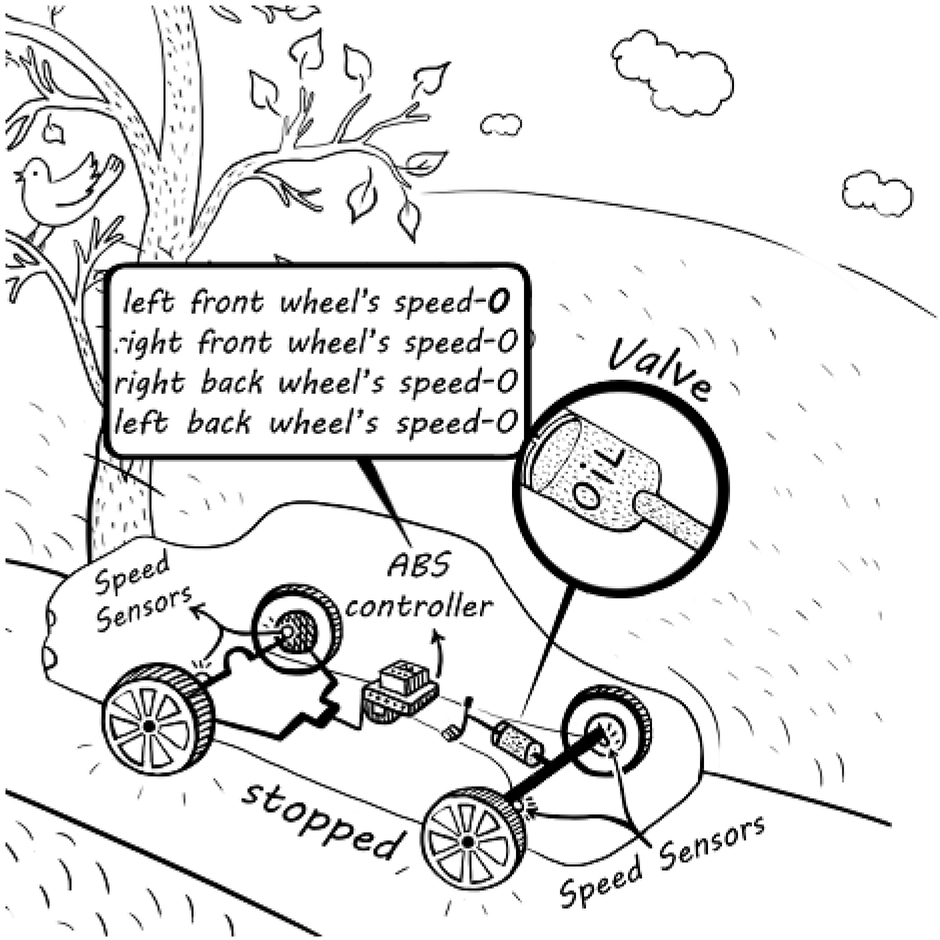
Priority-based FRP (P-FRP)



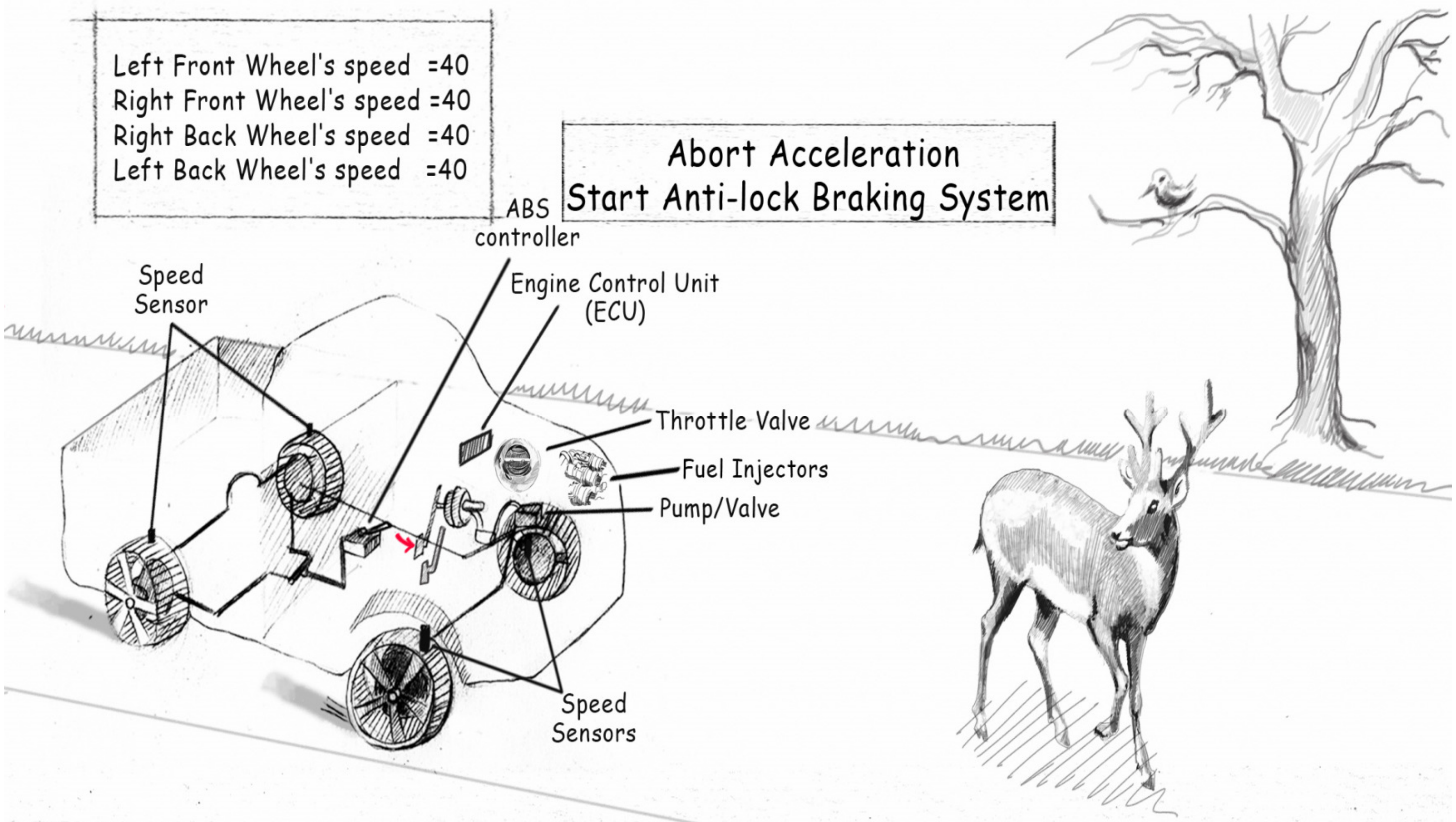
Priority-based FRP (P-FRP)



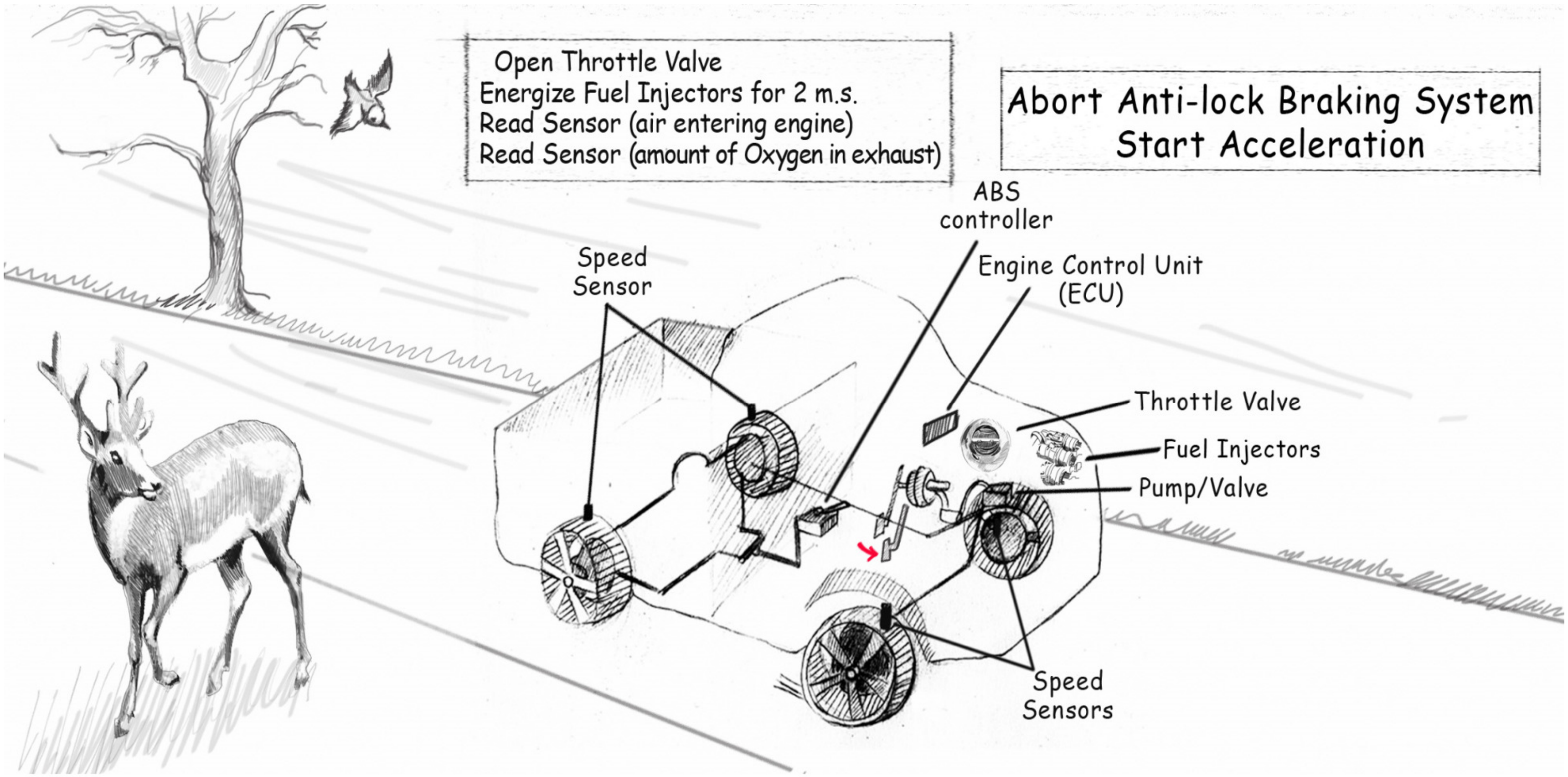
Priority-based FRP (P-FRP)



Priority-based FRP (P-FRP)



Priority-based FRP (P-FRP)



Highlights of Research Results

1. **Real-time Systems, CPS, FRS, P-FRP background**
2. **<u>Actual Response Time</u>**
3. Worst-case Response Time (WCRT) through Exhaustive Enumeration
4. Approximating WCRT in polynomial time
5. Feasibility Interval
6. Optimal Priority Assignments
7. Utilization Bounds
8. Partitioned Scheduling in Multi-processor Systems
9. Dynamic Voltage and Frequency Scaling
10. Response Time through Timed Automata
11. Response Time through Time Petri Nets

Priority-based FRP (P-FRP)

Algorithm 1 P-FRP Exact Schedulability Test Algorithm

Input: $\Gamma_n, [0, LCM)$

Output: True/False, (schedulable or not)

$\sigma_n([0, LCM)) \leftarrow \{[0, LCM)\}$

for $\tau_i = n \rightarrow 2$ **do**

  $\sigma_{i-1}([0, LCM)) \leftarrow \lambda(\sigma_i([0, LCM)), \Gamma_n)$

  if $|\sigma_{i-1}([0, LCM))| = 0$ **then**

    return false

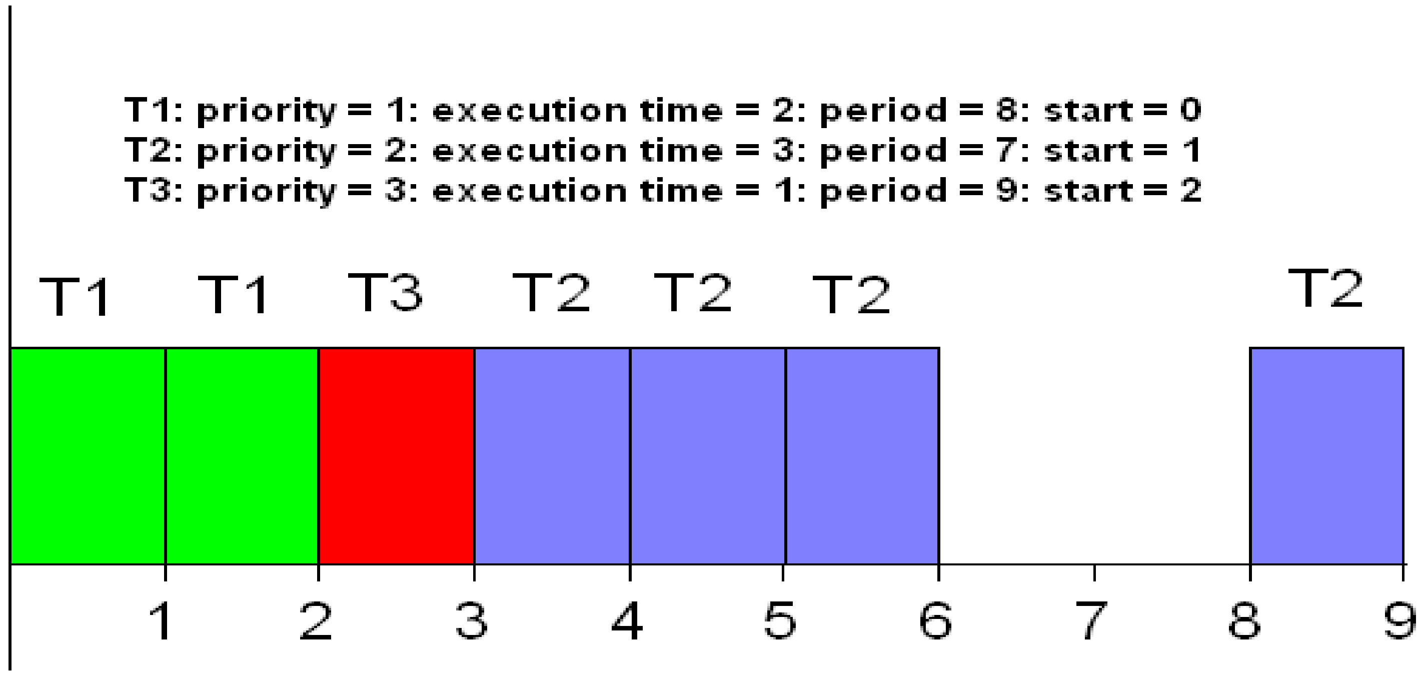
  end if

end for

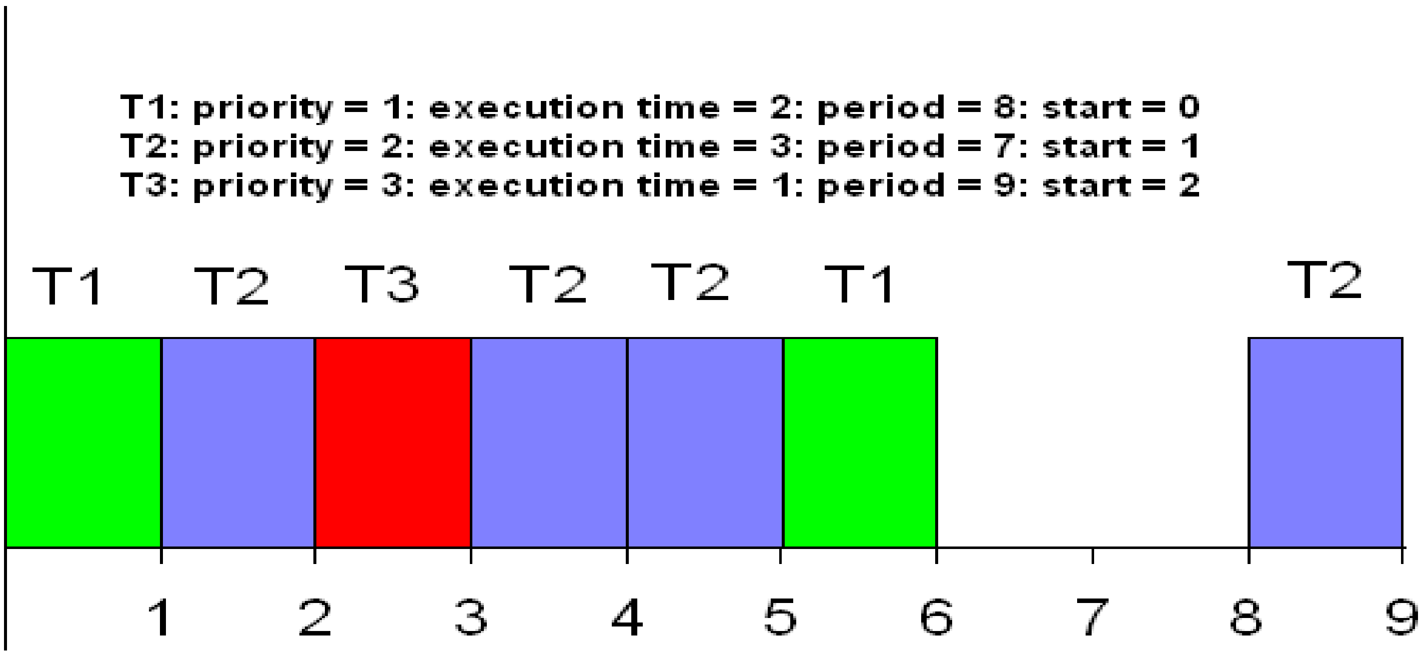
return $\leftarrow \mu(\sigma_1([0, LCM)), C_1)$

- On-line Schedulability Test returns the gap (the amount of execution time available) for the next lower-priority task.
- Precise (tight) timing characterization of the embedded controller software execution leads to faster physical system response compared with one designed without accurate controller timing analysis (and thus requires more tolerance of execution time variations).

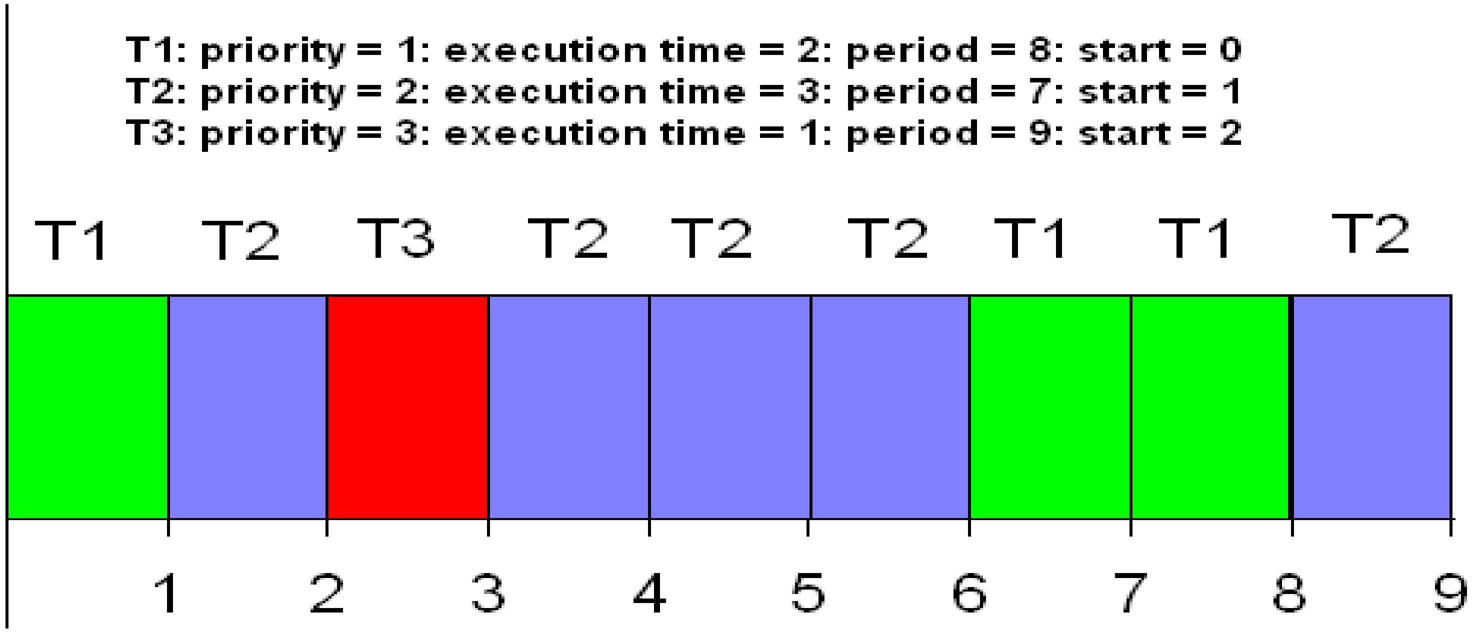
Non-Preemptive Execution



Preemptive Execution



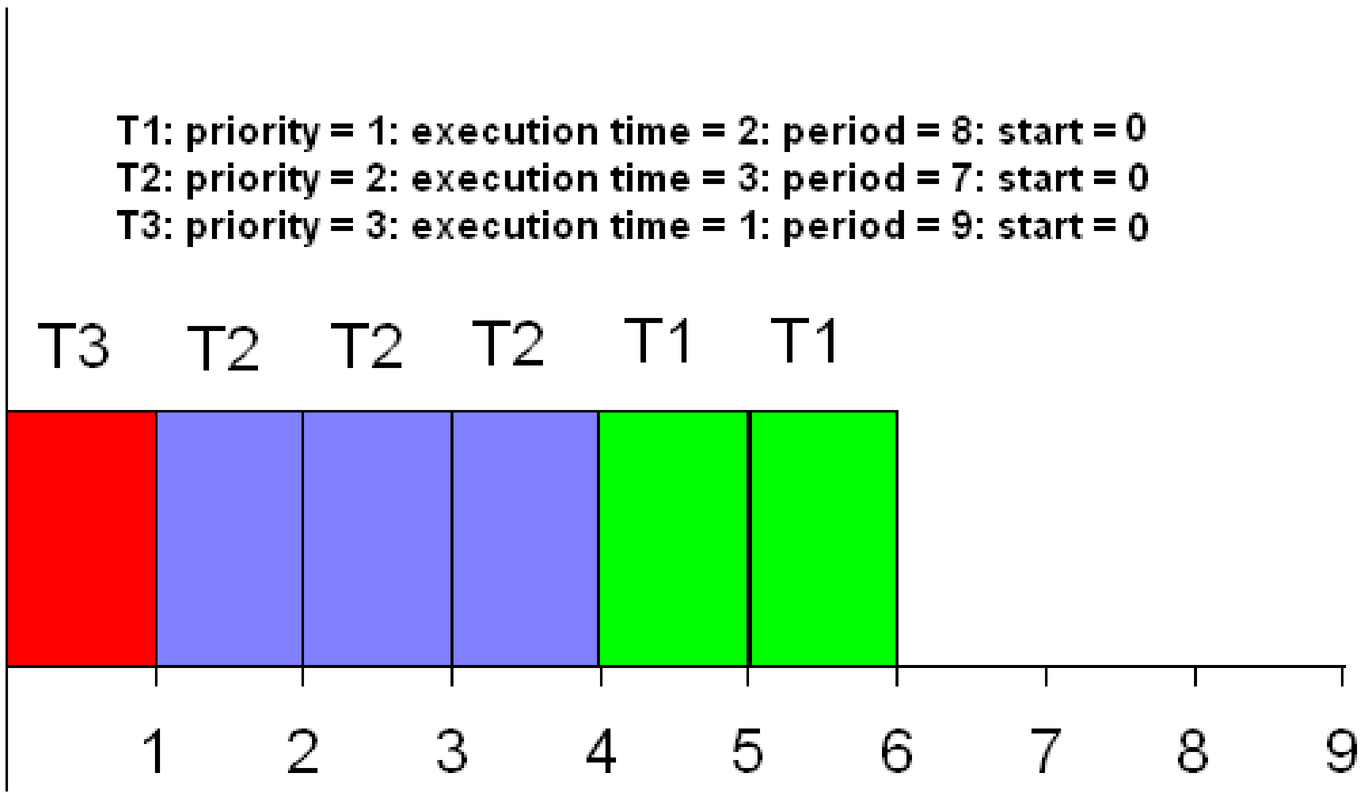
P-FRP Execution



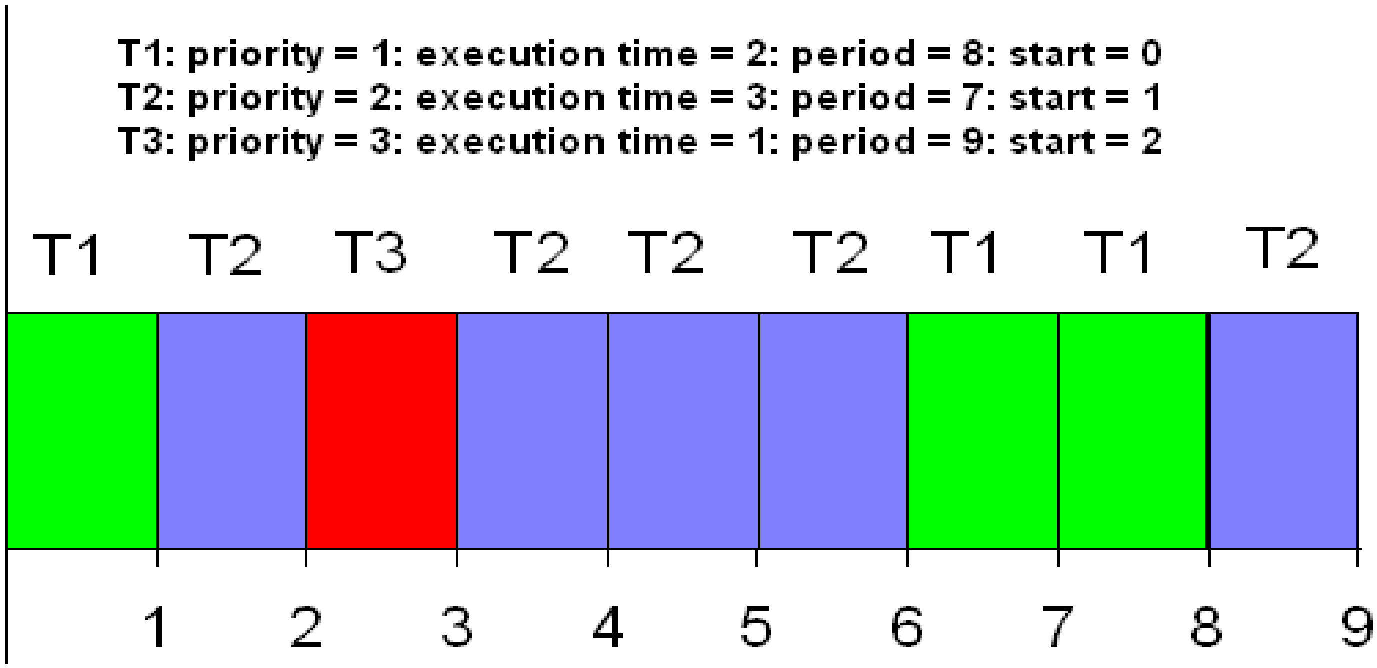
P-FRP Challenges

- Ascertaining temporal properties is difficult
  – Execution time is dynamic in nature
  – Information known *a priori* cannot be used
  – No notion of **Critical Instant**
- Existing methods for preemptive / non-preemptive execution cannot be applied
- New methods are required for Response Time Analysis and Schedulability

Critical Instant - Synchronous



Critical Instant - Asynchronous



Definitions

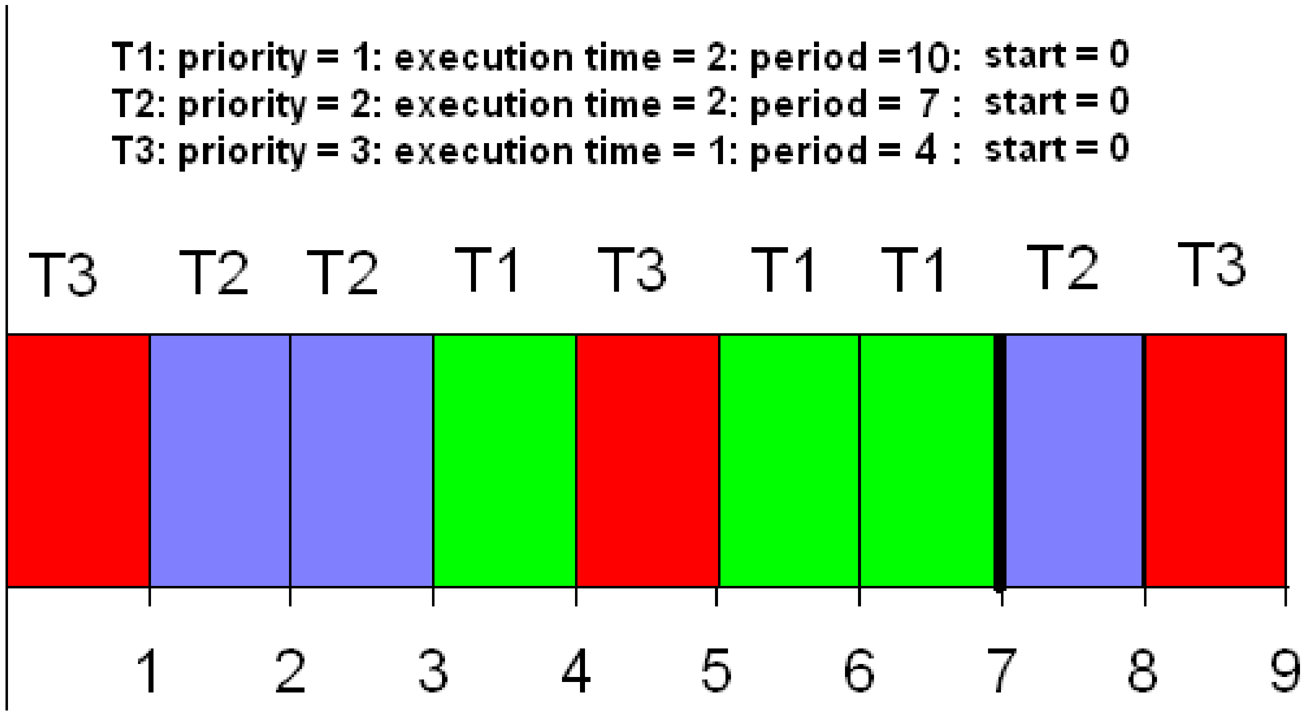
- **Interference cost -** In the preemptive model of execution, if a higher priority τ_i interferes with the execution of a lower priority task τ_j, then τ_i will preempt τ_j. The response time of τ_j will be delayed by time taken to process τ_i, which is P_i. This is referred to as the interference cost
- **Abort Cost -** In the P-FRP execution model, preempted tasks are also aborted. The amount of time spent in aborted processing is called the abort cost

Contributions

- This work deals with finding **actual** response time in P-FRP
- Actual response time is not an approximate value
- Actual response time is found for *a priori* known release scenario
- Method for finding actual response time is required for determining worst-case response time …

… as well as developing exact schedulability tests, analyzing multi-processor schedulability etc.

Existing Approach: Audsley et al



Existing Approach: Audsley et al

CS@UH

$$\text{Iteration } 1: \quad 2 + \left\lceil \frac{0}{4} \right\rceil \cdot 1 + \left\lceil \frac{0}{7} \right\rceil \cdot 2 = 2$$

$$\text{Iteration } 2: \quad 2 + \left\lceil \frac{2}{4} \right\rceil \cdot 1 + \left\lceil \frac{2}{7} \right\rceil \cdot 2 = 5$$

$$\text{Iteration } 3: \quad 2 + \left\lceil \frac{5}{4} \right\rceil \cdot 1 + \left\lceil \frac{5}{7} \right\rceil \cdot 2 = 6$$

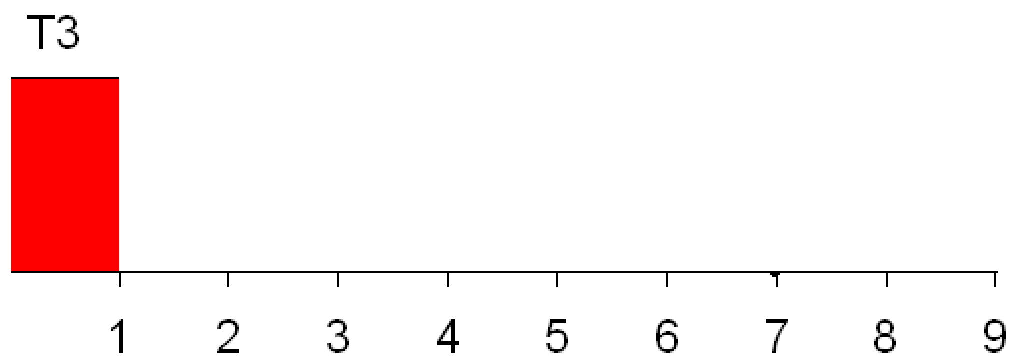
$$\text{Iteration } 4: \quad 2 + \left\lceil \frac{6}{4} \right\rceil \cdot 1 + \left\lceil \frac{6}{7} \right\rceil \cdot 2 = 6$$

Existing Approach: Ras & Cheng

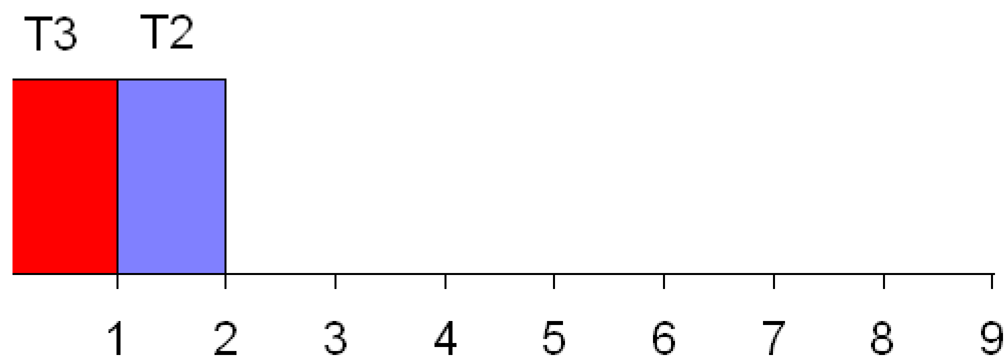
- Extension of Audsley's Method
- Abort cost is added on response time
- Abort cost from each higher priority task is accounted for
- Computed response time is not exact, but an upper bound on WCRT
- Solution does not converge for several cases

Simulation

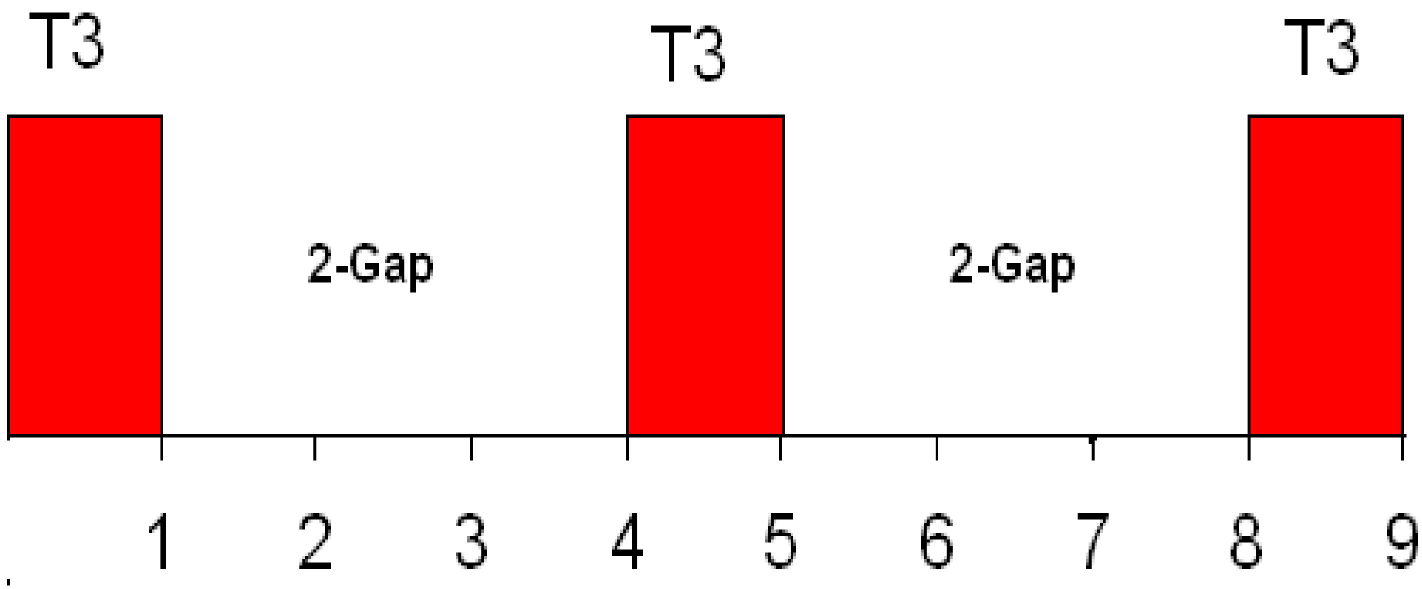
Iteration 1



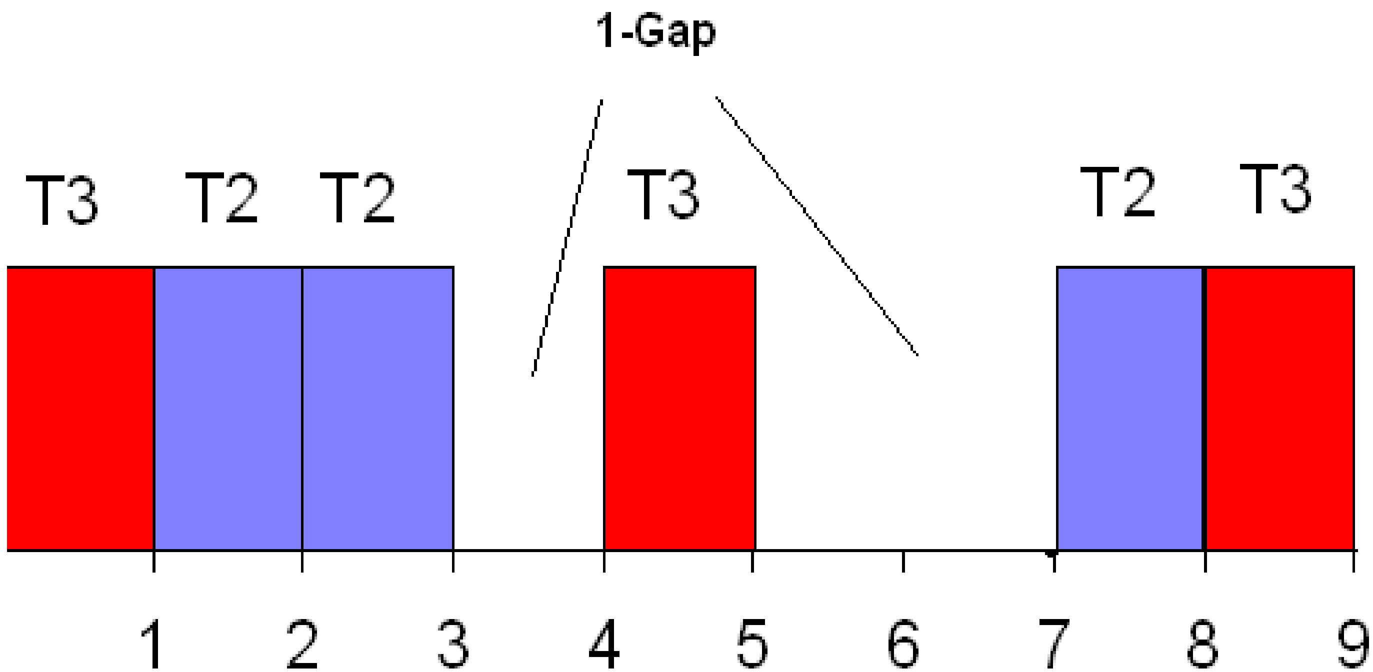
Iteration 2



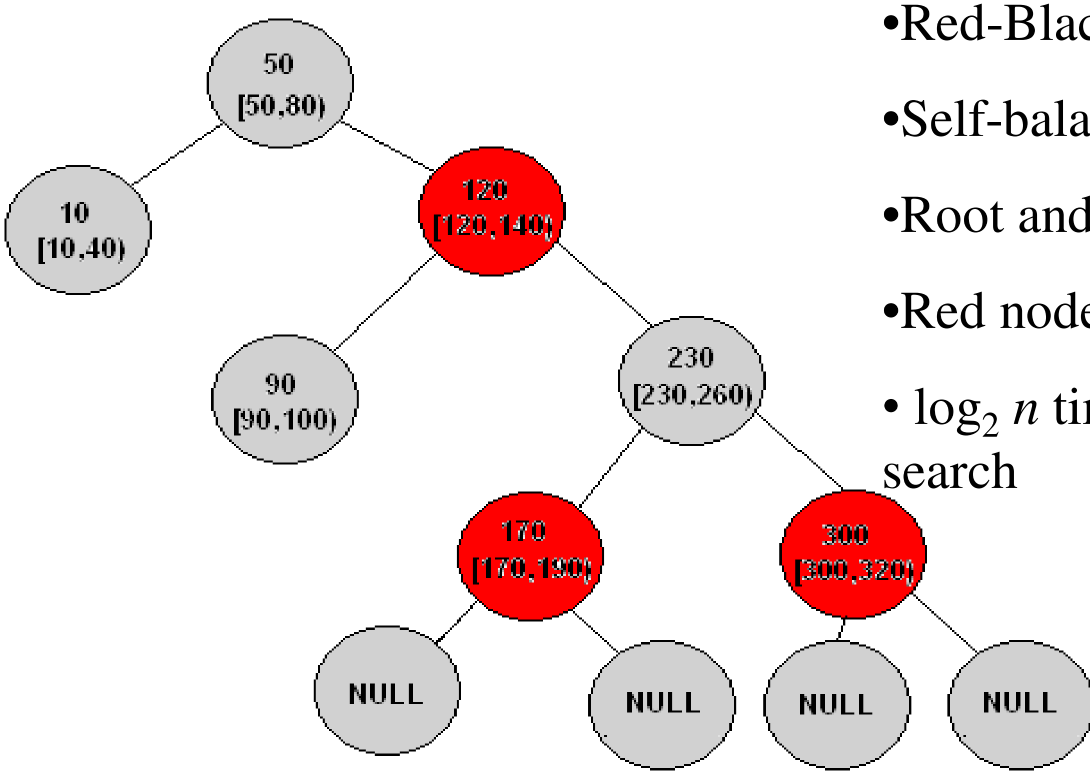
Gap Enumeration



Gap Enumeration



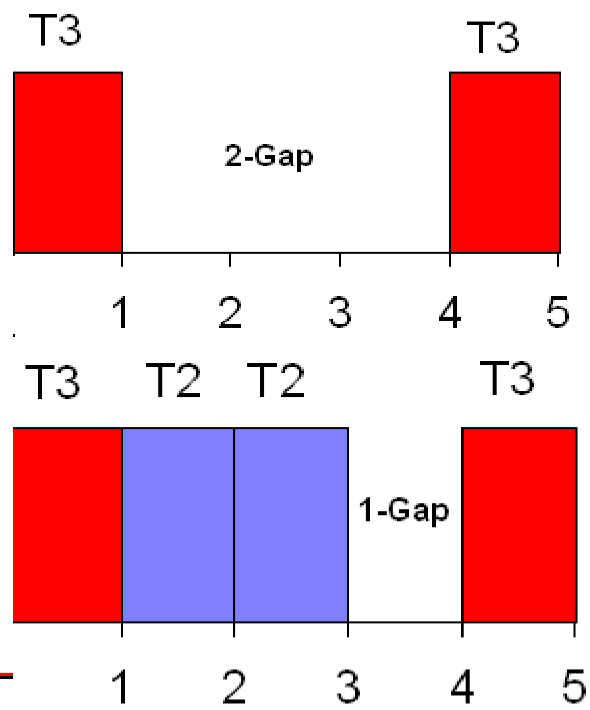
Gap Enumeration – Storage



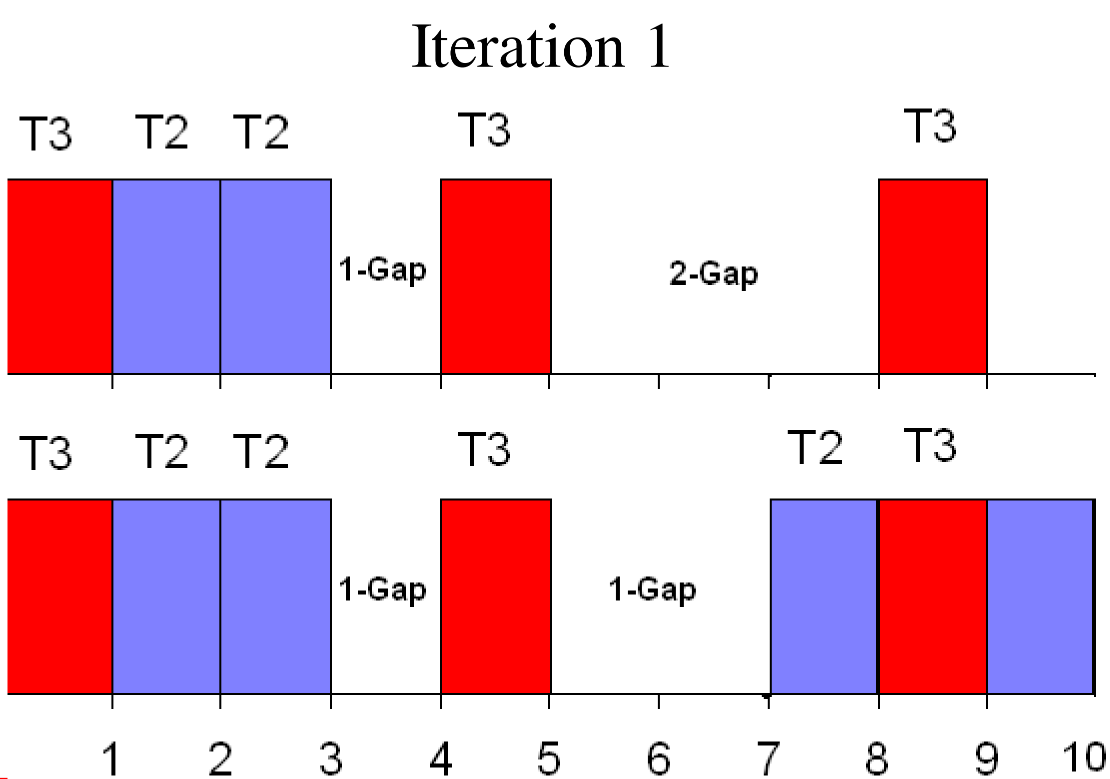
- Red-Black Tree
- Self-balancing binary search tree
- Root and leaf nodes are black
- Red node has black children
- $\log_2 n$ time for insertion, delete and search

Gap Enumeration – Dynamic Size

Iteration 1

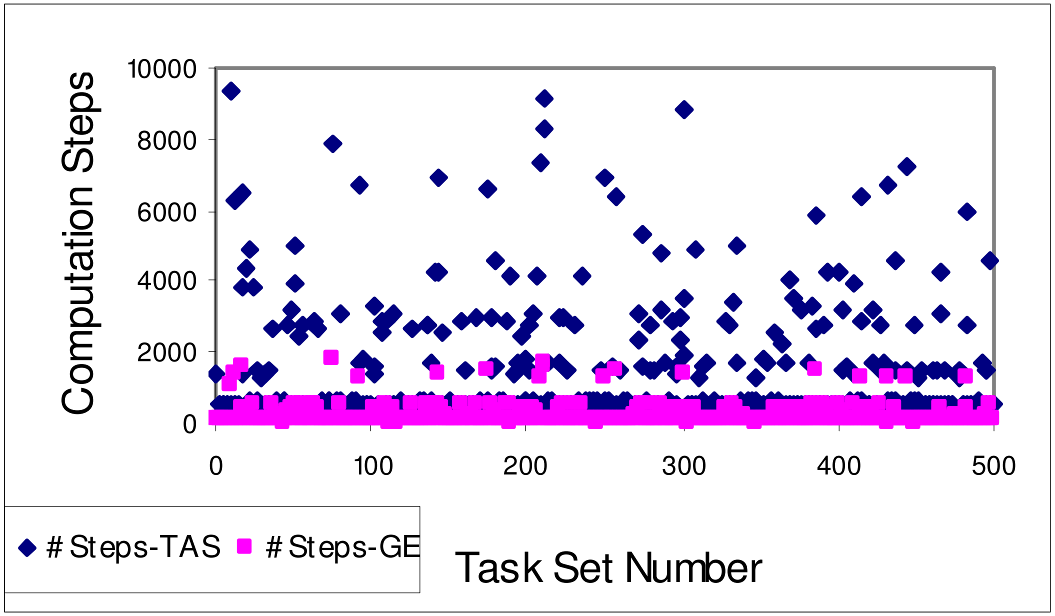


Gap Enumeration – Dynamic Size



Experimental Analysis

7 Tasks



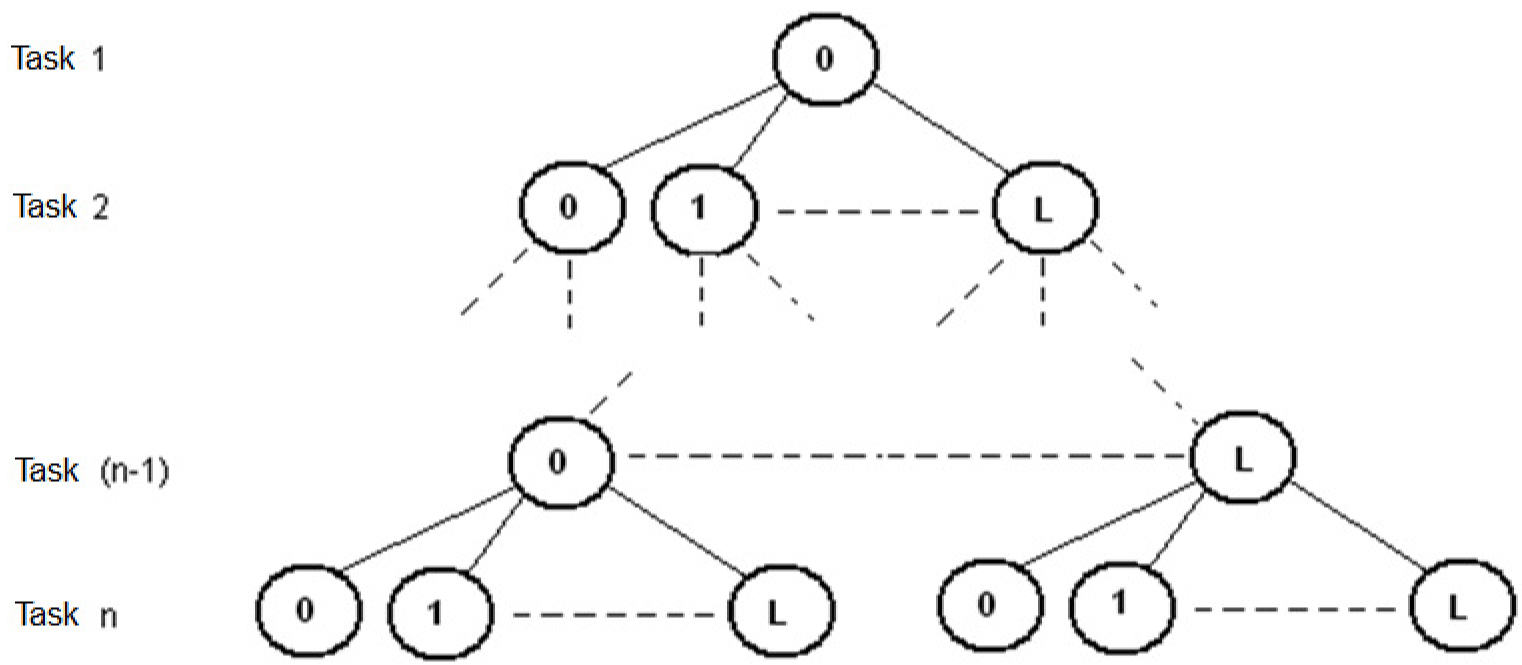
Remarks

- New method for response time computation
- Can compute response time under any given release scenario
- Chaitanya Belwal and Albert M. K. Cheng, “Determining Actual Response Time in P-FRP”, 13th International Symposium on Practical Aspects of Declarative Languages (PADL), Austin, Texas, USA January 24-25, 2011

Contents

1. **P-FRP and Real-time Systems background**
2. **Actual Response Time**
3. **Worst-case Response Time (WCRT) through Exhaustive Enumeration**
4. Approximating WCRT in polynomial time
5. Feasibility Interval
6. Optimal Priority Assignments
7. Utilization Bounds
8. Partitioned Scheduling in Multi-processor Systems
9. Dynamic Voltage and Frequency Scaling
10. Response Time through Timed Automata
11. Response Time through Time Petri Nets

Determining Exact WCRT



Determining Exact WCRT

- For a task set of size n, the total number of enumerations whose response time has to be evaluated is:

  $(D_j - t + t{+}1)^{n-j} = (D_j{+}1)^{|HP|}$ where |HP| is the number of higher priority tasks
- Number of enumerations and hence the computational cost, is dependent on the deadline of τ_j as well as the size of the task set
- Prior works in P-FRP only deal with computing approximate values of response time

Contributions

- We present techniques for determining the lower and upper bound on release offset of higher priority tasks for computation of exact WCRT in P-FRP
- This reduces the number of enumerated release scenarios by a considerable amount
- Highlight schedulability characteristics
- Present algorithm to computer release offset upper bound

Determining WCRT

- **Theorem**. Let Γ_n be a *n* task set: $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$. The release offsets of tasks $\tau_{j+1} \ldots \tau_n$ which lead to the worst-case response time of τ_j, are guaranteed to be more than or equal to the worst-case abort costs that can be induced on τ_j
- Theorem establishes a **lower bound** on release offset (lower bound = worst-case abort costs that can be induced on τ_j)
- Lower bound = Processing time of τ_j - 1

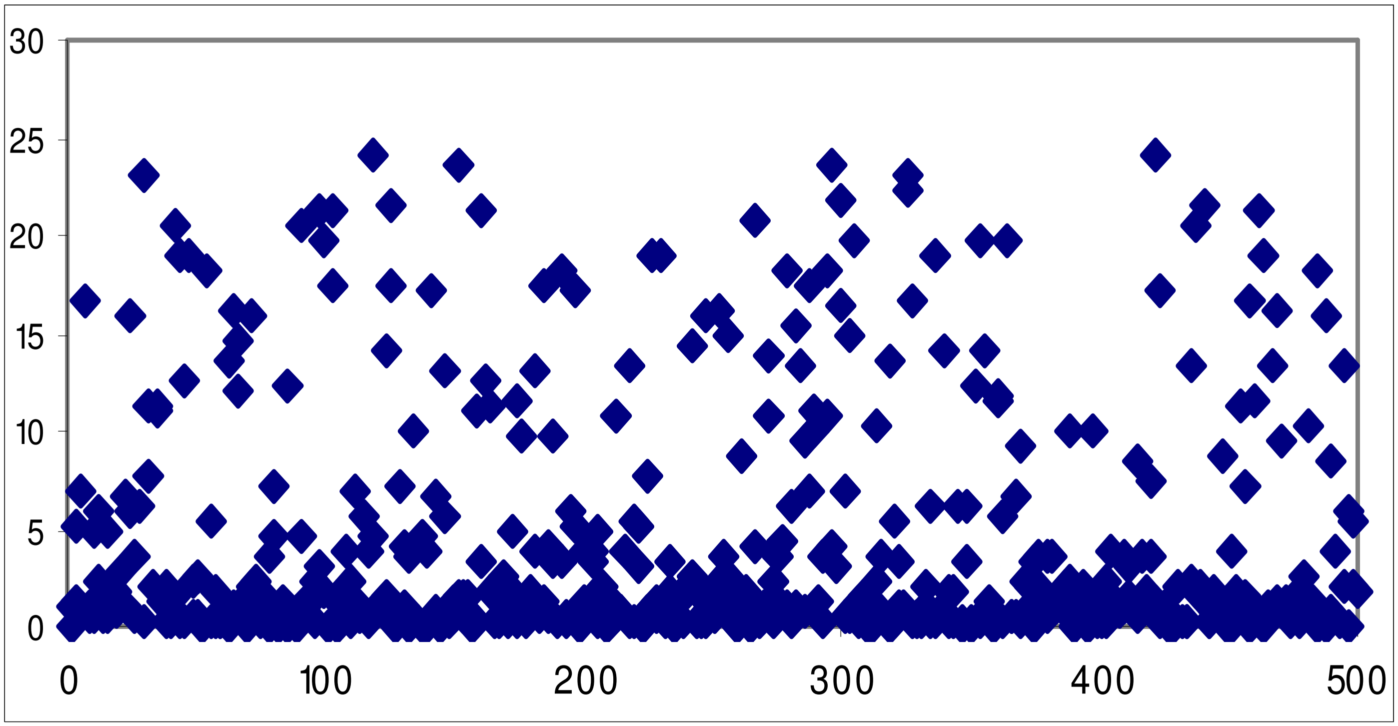
Determining WCRT

- **Theorem.** For a *n-task set* $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$, the release offset values of tasks $\tau_{j+1} \ldots \tau_n$, which lead to the worst-case response time of τ_j, have an upper bound
- Theorem proves that there is an **upper bound** on release offset of higher priority tasks

Release Offset Upper Bound

- Intuitive way to compute the release offset upper bound is to release the highest priority task first
- Followed by other tasks in priority order
- Release tasks at intervals such as to induce maximum abort cost on the lower priority task τ_j
- Does not lead to WCRT
- The 2nd or 3rd job of a higher priority tasks can further delay the response time
- Algorithm is used to compute Upper Bound

Results – 5 Tasks



% of enumerations required in offset bound relative to the number of enumerations computed in the deadline

Remarks

- Till now all release offset scenarios in the period $[0, T_j)$ have to be evaluated to determine WCRT of τ_j
- Our approach requires evaluation between the release offset bounds and is more efficient
- Chaitanya Belwal, Albert M. K. Cheng and Walid Taha, “Release Offset Bounds for Response Time Analysis of P-FRP”, 8th IEEE International Conference on Embedded Software and Systems (ICESS), Changsha, China, Nov. 16-18, 2011

Contents

Approximate WCRT in Polynomial Time

- As shown, Audsley's method cannot be used to determine response time in P-FRP
- Ras and Cheng's method computes approximate value of WCRT…
- …However this method does not converge for several task sets
- Guaranteed method for approximating WCRT in P-FRP is required

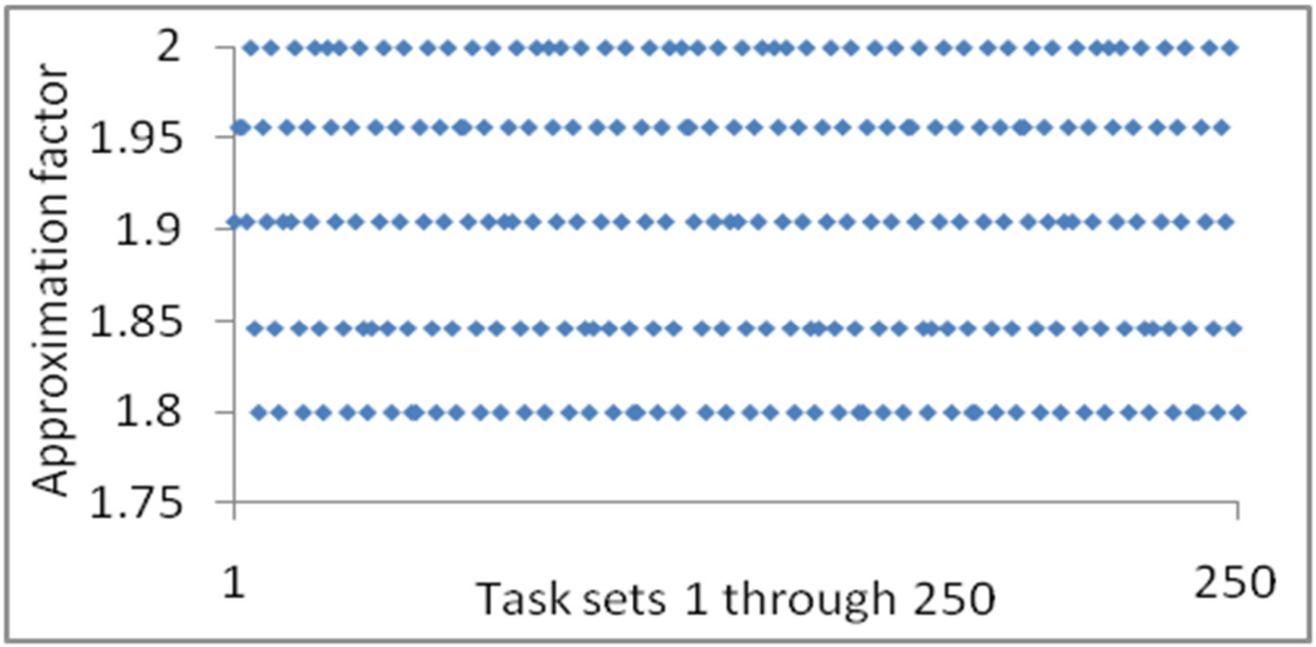
Contributions

- Derive an algorithm to compute approximate values of WCRT in P-FRP
- This algorithm is guaranteed to converge to a result
- Approximation factors evaluated through experimental task sets

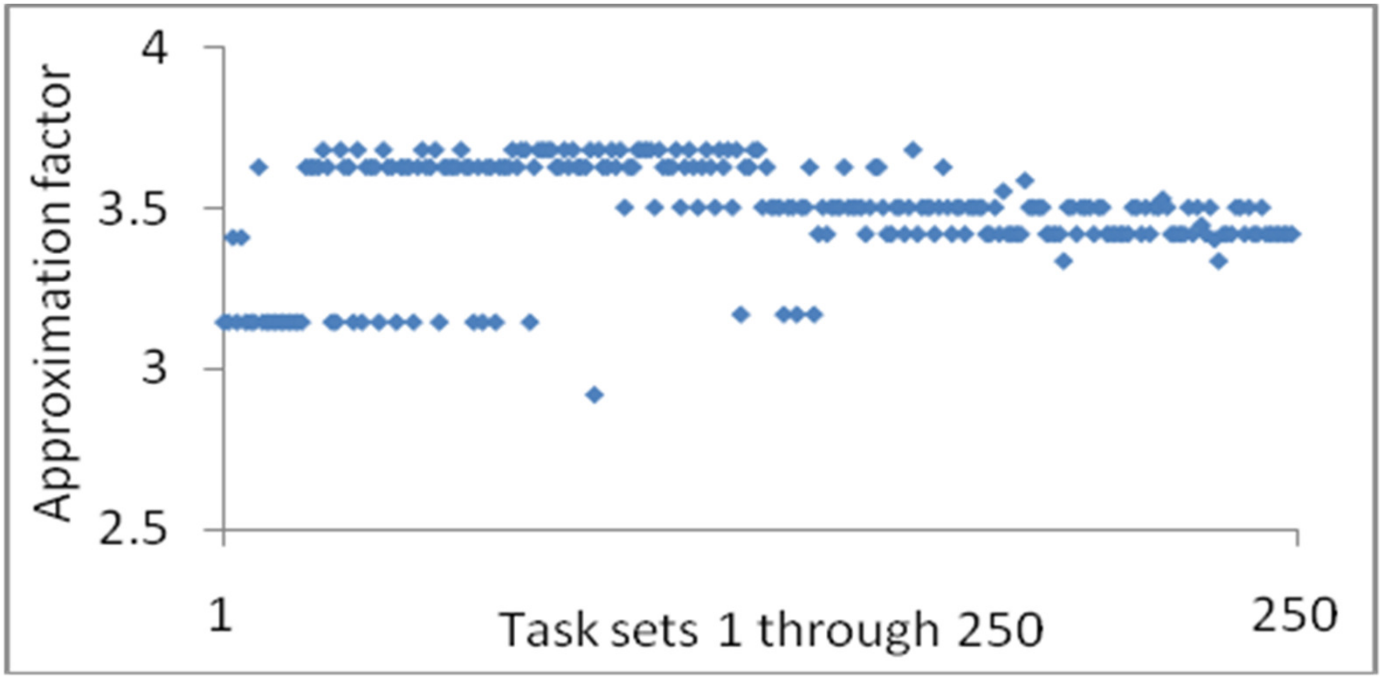
Algorithm Outline

- Set lower bound of WCRT equal to the value computed by Audsley's algorithm
- Use the lower bound as a base value and add interference and abort costs
- Run an iterative loop based on number of higher priority tasks
- Add costs for prior tasks in every iteration
- Iterative loop is guaranteed to complete

Results – 3 Tasks / Low Utilization

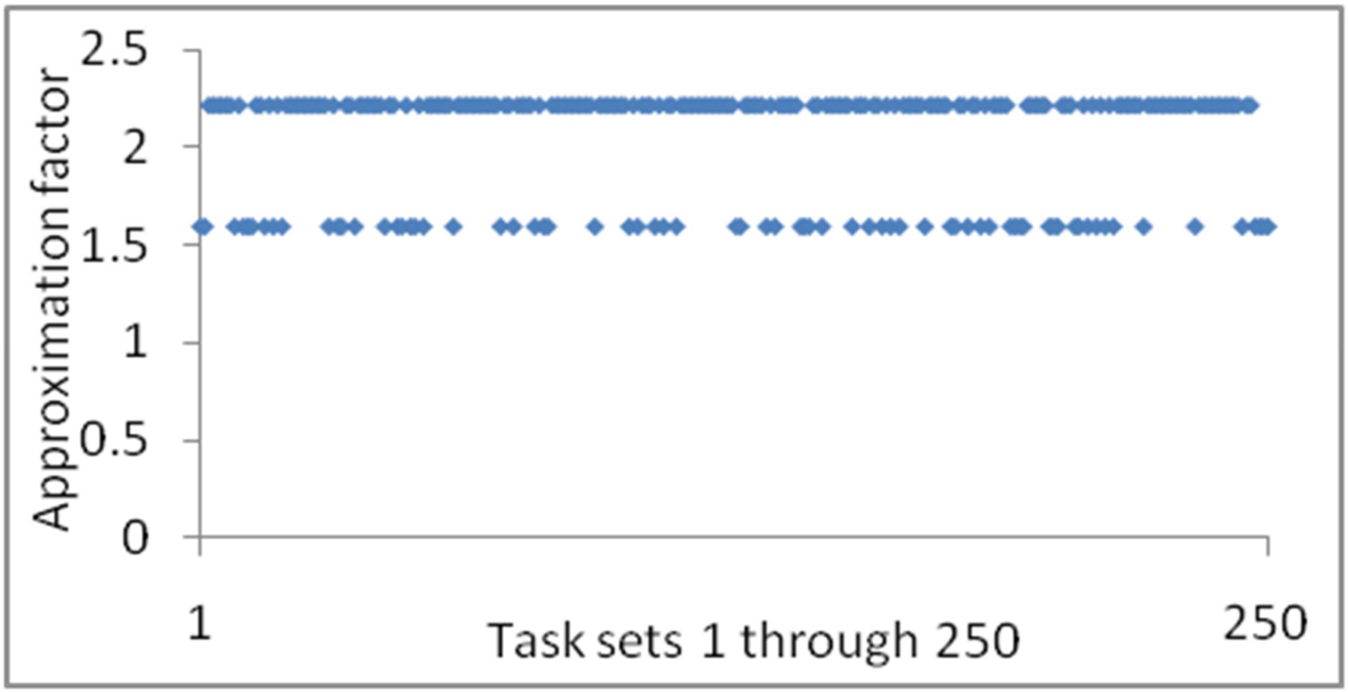


Results – 5 Tasks / Low Utilization

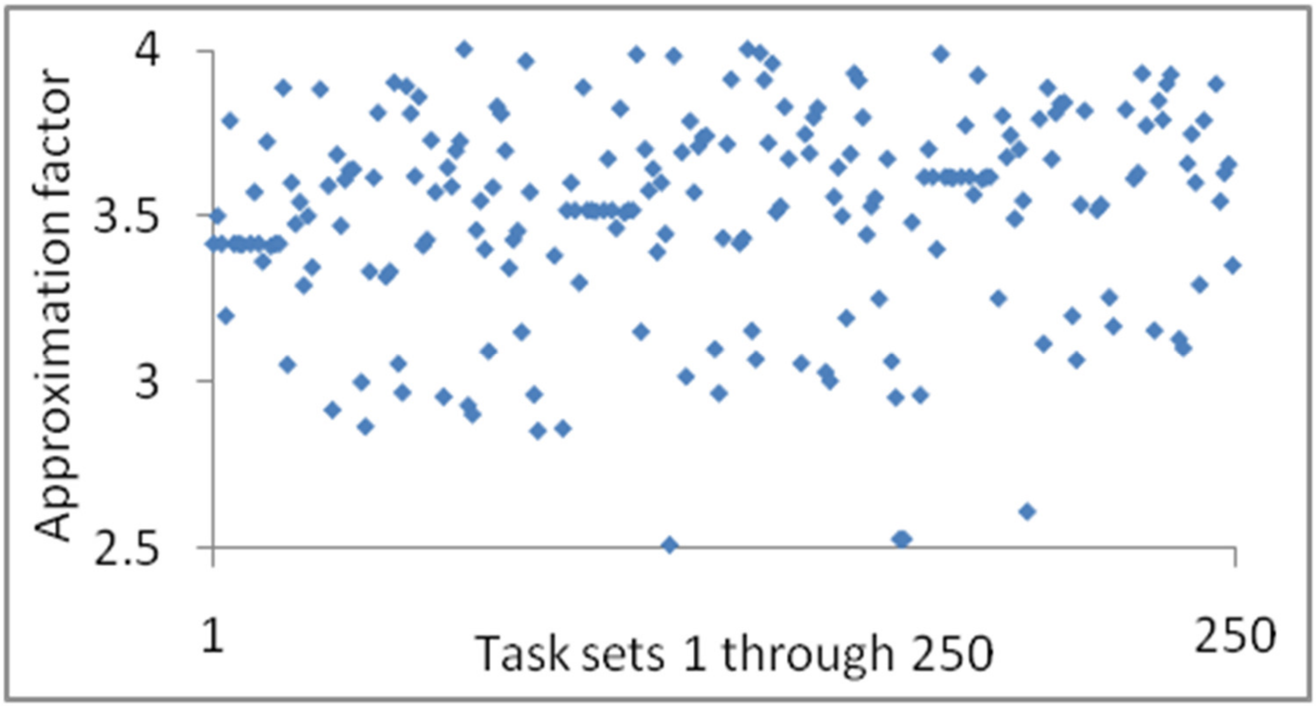


CS@UH

Results – 3 Tasks / High Utilization



Results – 5 Tasks / High Utilization



Remarks

- High approximation factor for larger task sets due to larger pessimism in abort costs
- Reducing pessimism while maintaining correctness is challenging
- C. Belwal, A. M. K. Cheng, W. Taha, and A. Zhu, “Time Analysis of the Priority based FRP System”, IEEE-CS Real-Time and Embedded Technology and Applications Symposium WIP Session, St. Louis, MO, April 22-24, 2008

Contents

Feasibility Interval

- Real-time System tasks can run infinitely often
- No tasks should have a deadline miss as long as system is running (*hard* real-time)
- Ascertaining schedulability for an infinite period is not possible
- Finite time is used to analyze schedulability
- Termed *feasibility interval* in real-time studies

Feasibility Interval for Preemptive Execution

- In their seminal paper, Liu and Layland have shown that the *WCRT* occurs when tasks are released *synchronously* (at the same time)
- The feasibility interval in a synchronous release is [0, *L*), where L is the *least common multiple* of all task periods
- Schedulability in [0,*L*) guarantees schedulability since worst-case schedulability is also analyzed

Contributions

- Formally present execution characteristics of tasks in a P-FRP system with 2 tasks
- Formally present execution characteristics of tasks in a P-FRP system with > 2 tasks
- Derive the feasibility interval of P-FRP

Processing Pattern

- Two time intervals of equal lengths $[t_1, t_1+a)$ and $[t_2, t_2+a)$ are said to have the same **processing pattern**, if for every value of relative time t: $0 \leq t < a$, the task that is processed at relative time t in $[t_1, t_1+a)$ (absolute time $t_1 + t$), is also processed at relative time t in $[t_2, t_2+a)$ (absolute time $t_2 + t$)

Feasibility Interval in P-FRP

- In P-FRP preempted tasks are aborted
- Leads to different execution semantics
- Unknown if feasibility interval of the preemptive model can be applied to this execution model
- Fresh approach required to determine the feasibility interval

Feasibility Interval

- **Theorem.** For $\Gamma_n=\{\tau_1, \tau_2, \ldots, \tau_n\}$ and $R_{max} = max\{\Phi_i\}$, the feasibility interval of Γ_n is $[t, t+L)$, where $t \geq R_{max}$
- **Corollary**. The earliest feasibility interval of Γ_n is $[R_{max}, R_{max}+L)$
- **Corollary**. If all tasks in Γ_n are synchronously released, then the earliest feasibility interval is $[0, L)$

Remarks

- Formally derived the feasibility interval in P-FRP
- Can be extended to consider non-periodic tasks
- Chaitanya Belwal and Albert M. K. Cheng, “Feasibility Interval for the Transactional Event Handlers of P-FRP”, 8th IEEE International Conference on Embedded Software and Systems (ICESS), Changsha, China, Nov. 16-18, 2011.

Contents

1. **P-FRP and Real-time Systems background**
2. **Actual Response Time**
3. **Worst-case Response Time (WCRT) through Exhaustive Enumeration**
4. **Approximating WCRT in polynomial time**
5. **Feasibility Interval**
6. **<u>Optimal Priority Assignments</u>**
7. Utilization Bounds
8. Partitioned Scheduling in Multi-processor Systems
9. Dynamic Voltage and Frequency Scaling
10. Response Time through Timed Automata
11. Response Time through Time Petri Nets

CS@UH

Optimal Priority Assignments

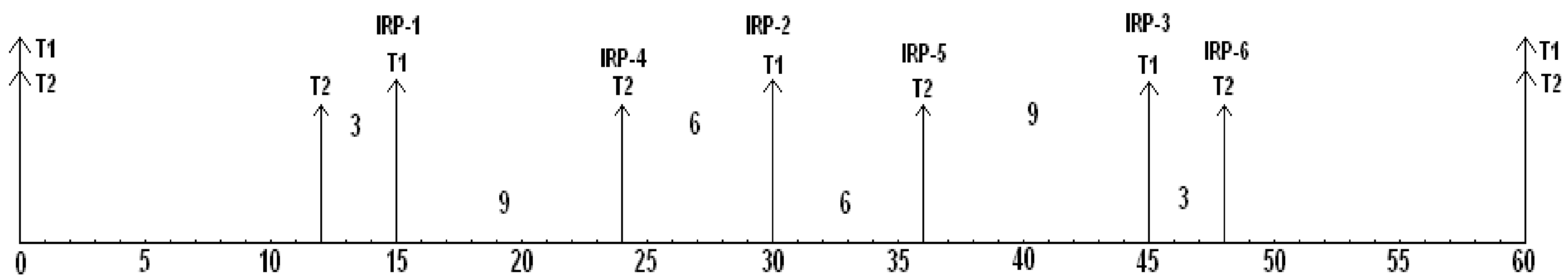
- Rate-Monotonic (RM) priority assignment is optimal in the preemptive model (Liu and Layland)
- RM is not optimal in P-FRP ...
- … can be easily proven with an example
- Unknown if an optimal priority assignment can even exist for this execution model

Contributions

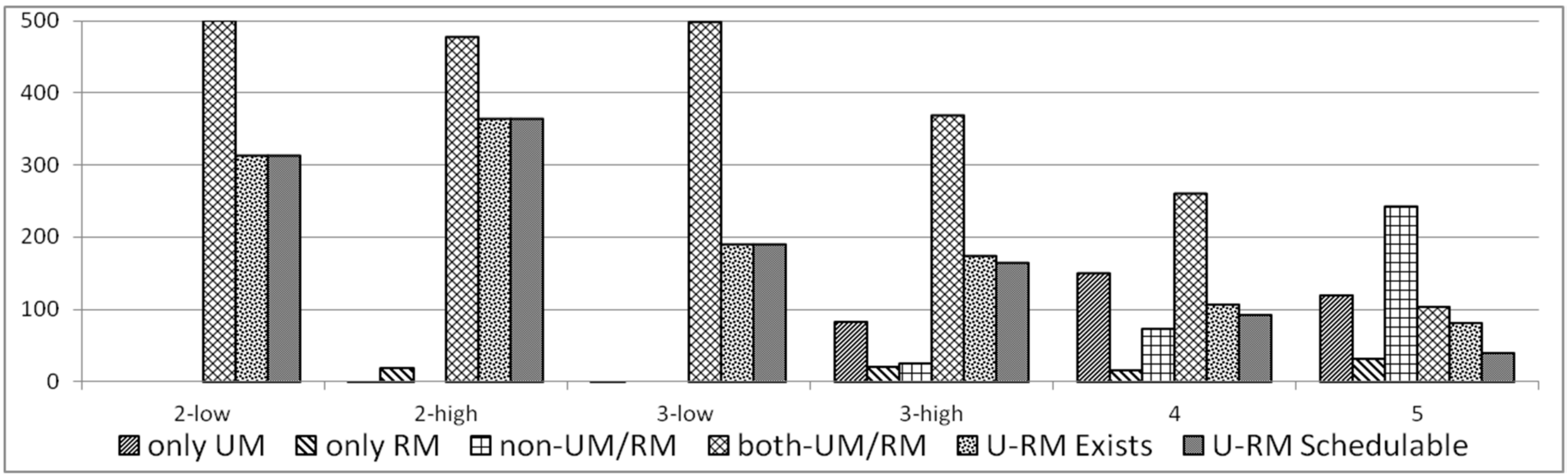
- Analyze schedulability characteristics of P-FRP tasks
- Several Theorems are proved
- Study priority assignment for 2 tasks and formally prove that U-RM (Utilization and Rate Monotonic) priority assignment is optimal
- Prove that no single priority assignment can be optimal for more than 2 tasks
- Experimentally evaluate results

Intermediate Release Points (IRPs)

Task	*pr*	*P*	*T*	*U*
τ_1	1	7	15	0.46
τ_2	2	3	12	0.25



Results



500
400
300
200
100
0
2-low
2-high
3-low
3-high
4
5
only UM
only RM
non-UM/RM
both-UM/RM
U-RM Exists
U-RM Schedulable

Remarks

- U-RM is the optimal priority assignment in 2-task sets
- For more than 2 tasks no single priority assignment can be optimal
- Several large tasks sets are still U-RM schedulable
- Chaitanya Belwal and Albert M. K. Cheng. “On Priority Assignment in P-FRP”, Proc. IEEE-CS Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Stockholm, Sweden, April 13-16, 2010

Contents

Utilization-based Sufficient Tests

- Liu and Layland's (LL) utilization bound is widely used as a sufficient schedulability test

$$U \leq n \cdot (2^{1/n} - 1)$$

n = number of tasks,

U = sum of utilization ratios of all tasks

- For 2 tasks $U \leq 0.83$, for 3 tasks $U \leq 0.78$ etc, for task set to be guaranteed schedulable

Utilization-based Sufficient Tests

- Liu and Layland's bound is derived by considering worst-case release scenario
- Worst-case release scenario is also assumed in derivations of other schedulability tests (e.g. Bini and Baruah's)
- Worst-case scenario is derived using critical instant
- In P-FRP, the worst case release scenario is not the synchronous release of tasks

Contributions

- Derive a worst-case release scenario with 2 P-FRP tasks
- Use this worst-case scenario to derive sufficient utilization bounds for P-FRP tasks sets with 2 tasks
- Prove that worst-case scenario for 2 and *n* (*n* > 2) tasks is different
- Present a pessimistic condition with *n* tasks
- Use the pessimistic condition to derive utilization bound for *n* tasks
- Experimental Analysis

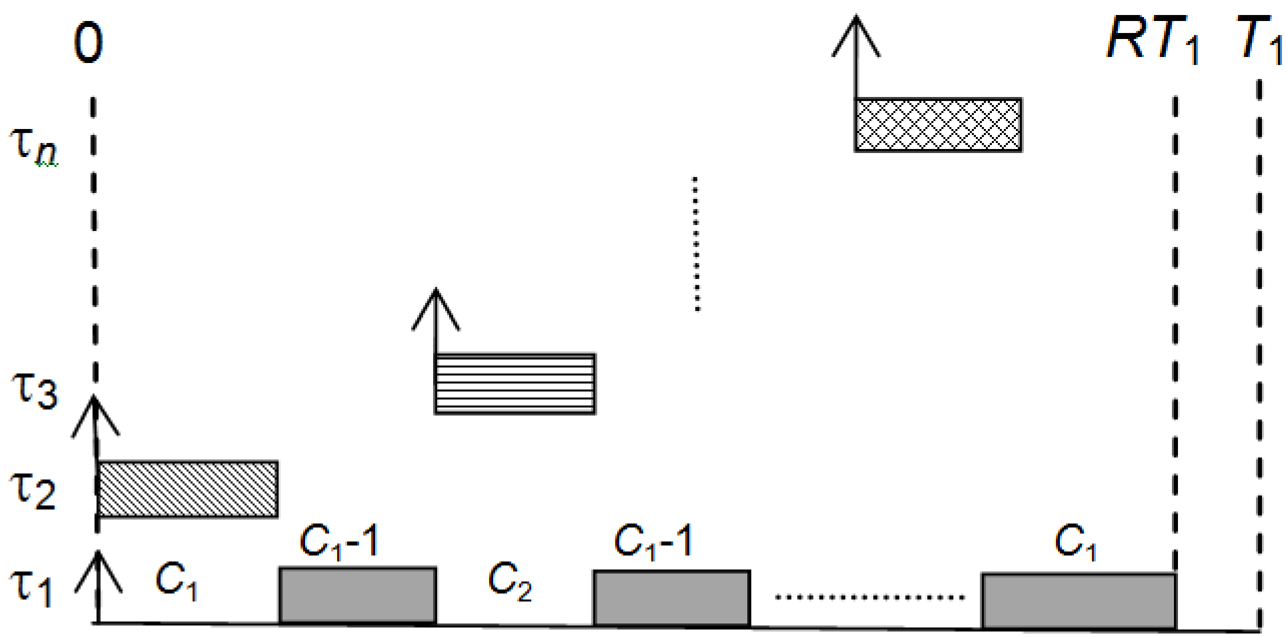
Utilization Bound for 2 Tasks

- **Theorem.** A task set with 2 tasks $\{\tau_1, \tau_2\}$ where $T_2 \leq 2 \cdot T_1$ is guaranteed to be schedulable when the total utilization factor U of this task set is less than or equal to 0.5. Or, the sufficient utilization bound of the task when $T_2 \leq 2 \cdot T_1$ is: $U \leq 0.5$.
- When $T_2 > 2 \cdot T_1$ then tasks with $U \rightarrow 0$ can also be schedulable, and a sufficient bound does not exist

Utilization Bound with *n* Tasks

- Approach used for 2 tasks cannot be directly applied
- Worst-case release scenario can be different for unique task sets
- Identify a low utilization task set
- Derive bound under full utilization for this task set

Utilization Bound for *n* Tasks

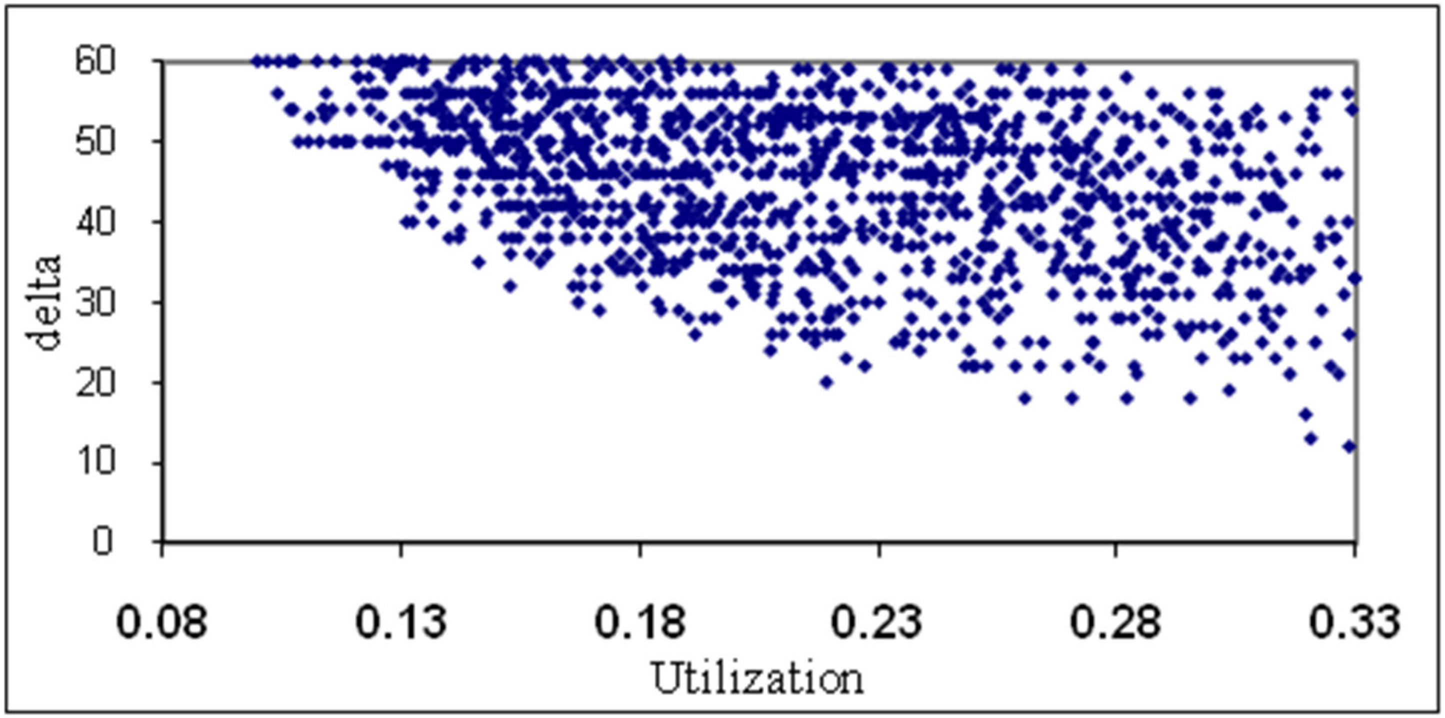


Worst-case release scenario for a pessimistic task set

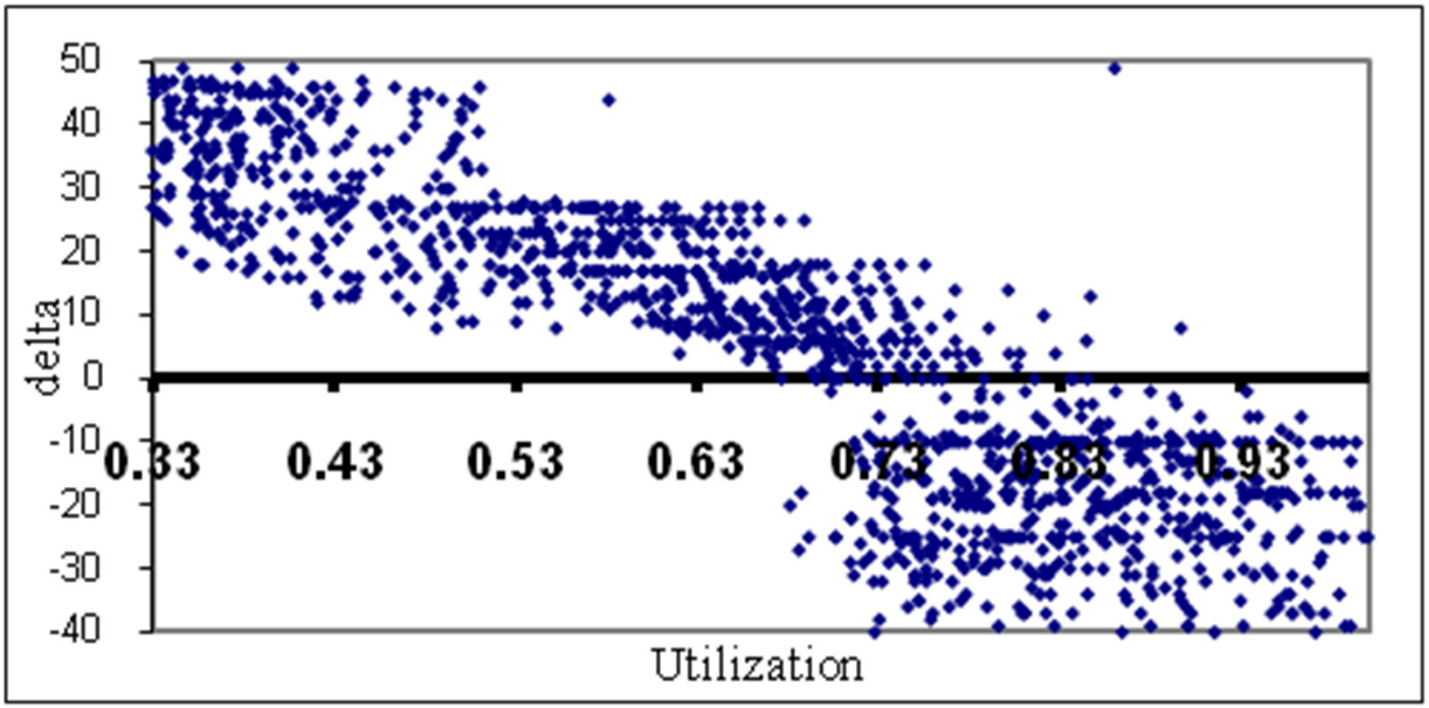
Utilization Bound with n Tasks

Theorem. A task set having n tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ such that $n \cdot T_1 \geq T_i$, $i = 2, \ldots, n$, is guaranteed to be schedulable when the total utilization factor U of this task set is less than or equal to $1/n$. Or, the sufficient utilization bound of Γ_n when $n \cdot T_1 \geq T_i$ is $U \leq 1/n$.

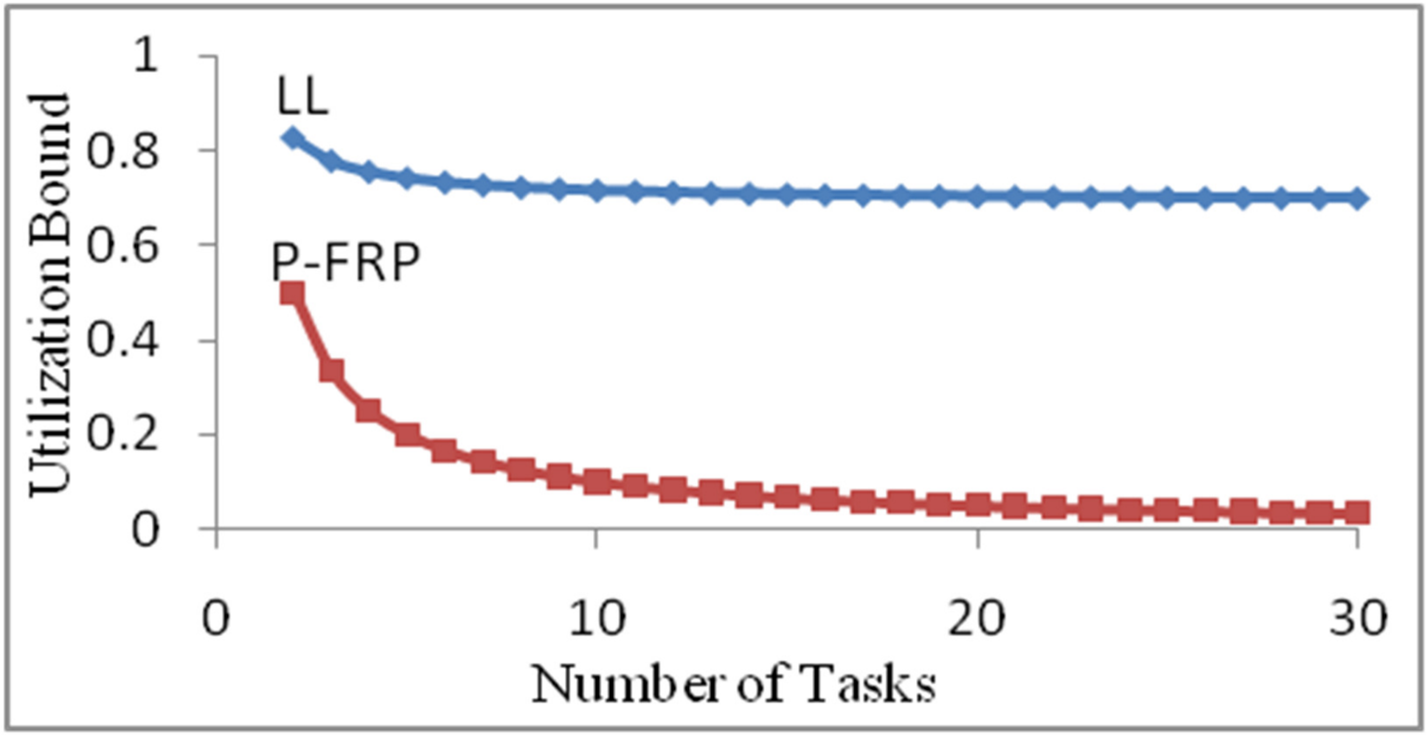
Results 3 Tasks - Schedulability



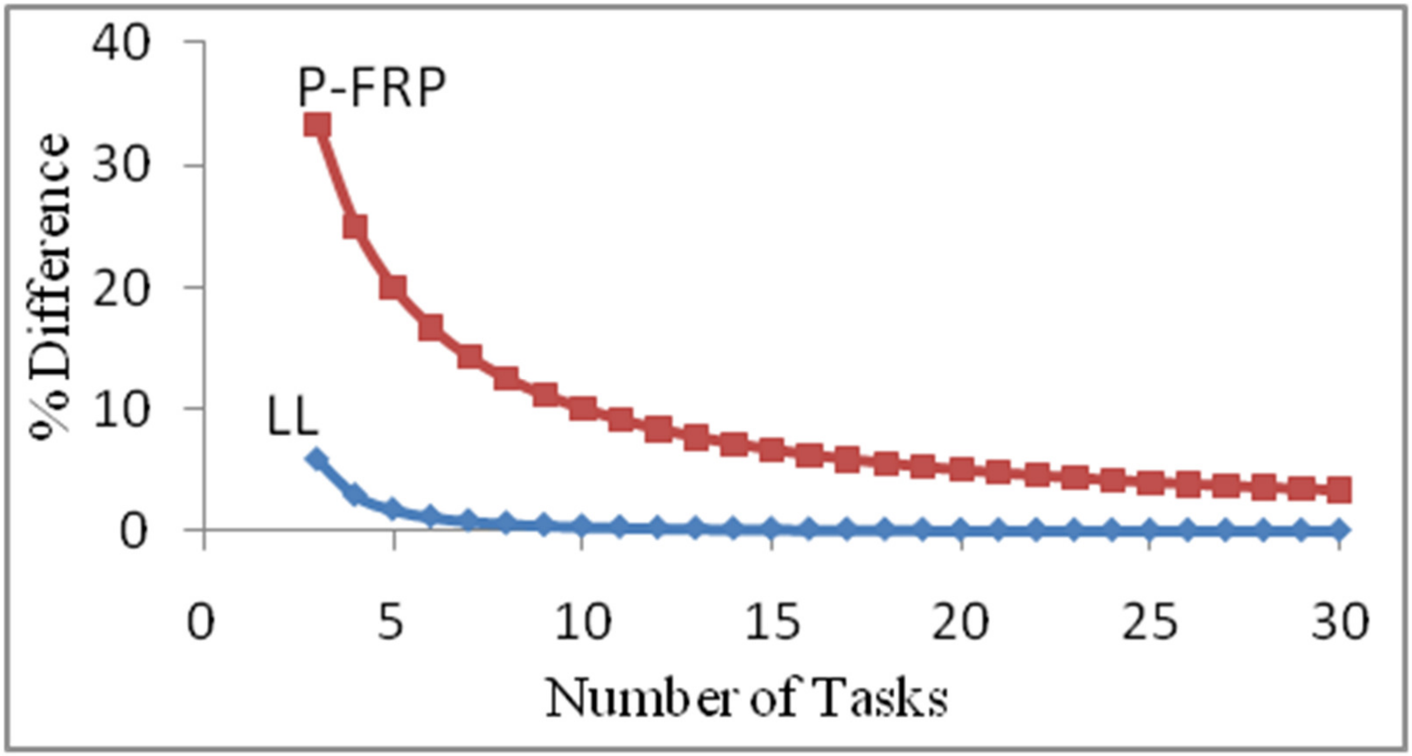
Results 3 Tasks - Unschedulability



Comparisons with LL Bound



Comparisons with LL Bound



Remarks

- Determined sufficient utilization condition for P-FRP task sets
- Chaitanya Belwal and Albert M. K. Cheng, “A Utilization based Sufficient Condition for P-FRP”, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC), Melbourne, Australia, Oct 24-26, 2011

Contents

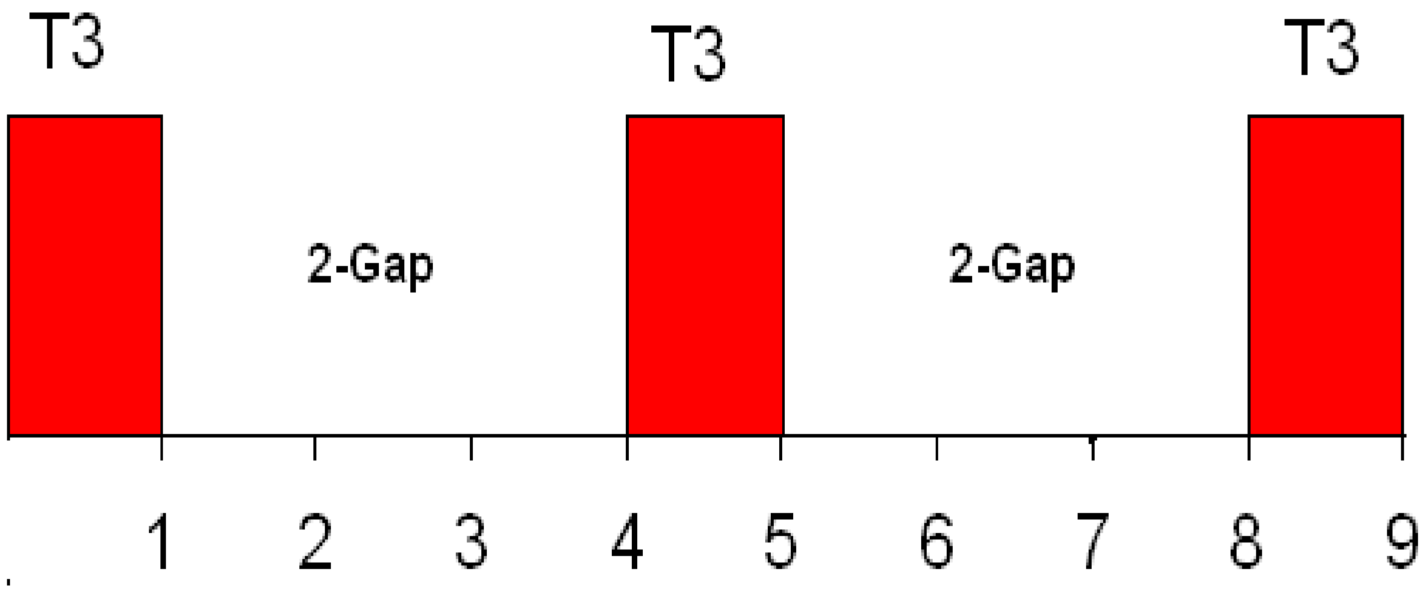
Static vs. Global Partitioning

- Partitioning refers to assignment of tasks that will execute in a processor
- In static partitioning, task assignment to processors is done offline
- Task assignment cannot be changed while system is running
- Global partitioning is dynamic, and tasks can move between processors while system is running
- No partitioning scheme is ideal

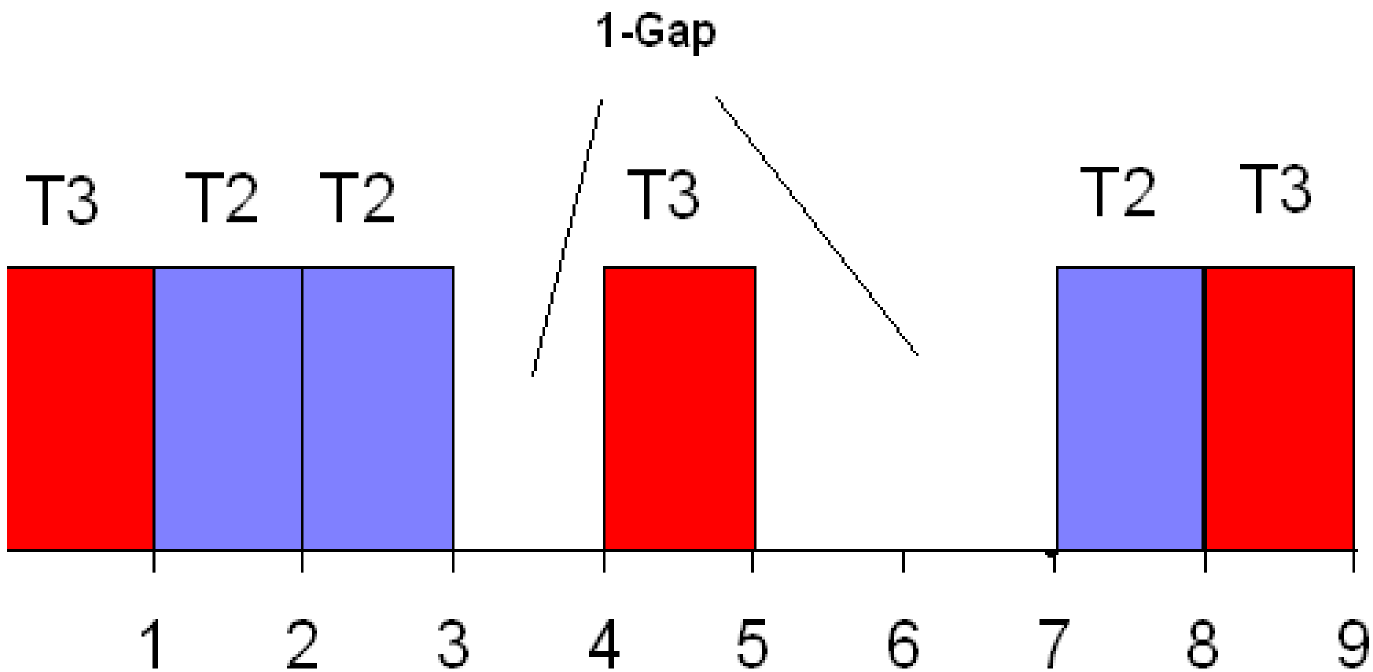
Contributions

- Study static partitioning of P-FRP in multi-processor systems
- Develop an exact schedulability test for P-FRP tasks
- Three schemes applying first-fit algorithm on different sorting criterion
- Partitioning schemes analyzed in rigorous experimental analysis by comparing it with an optimal scheme
- Valid for synchronous release of tasks

Exact Schedulability Test



Exact Schedulability Test



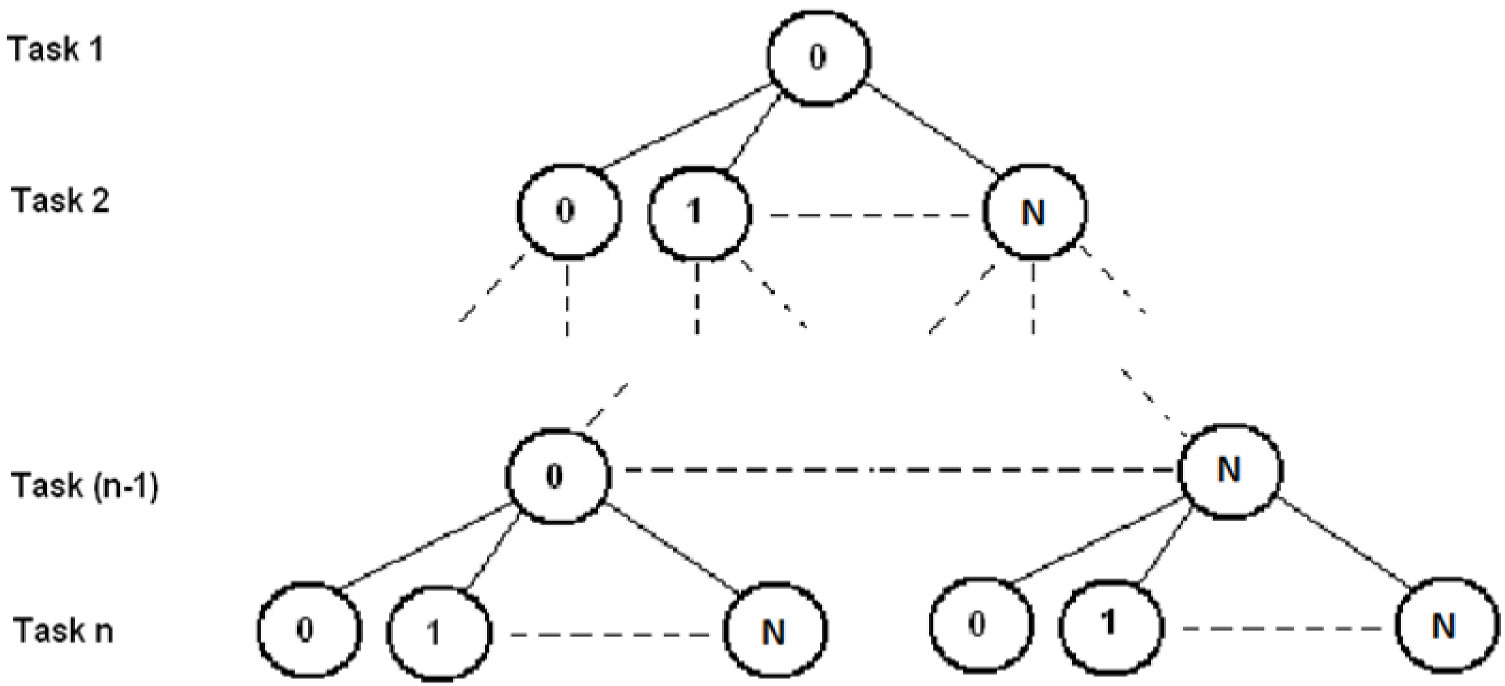
Bin-Packing

- Classical NP-hard problem in Computer Science
- Object with different sizes are packed in finite number of bins
- Has previously been used in static partitioning of tasks in SMP platforms
- Tasks are sorted using some criterion
- First-fit, last-fit heuristics widely used

Bin-Packing with Schedulability Test

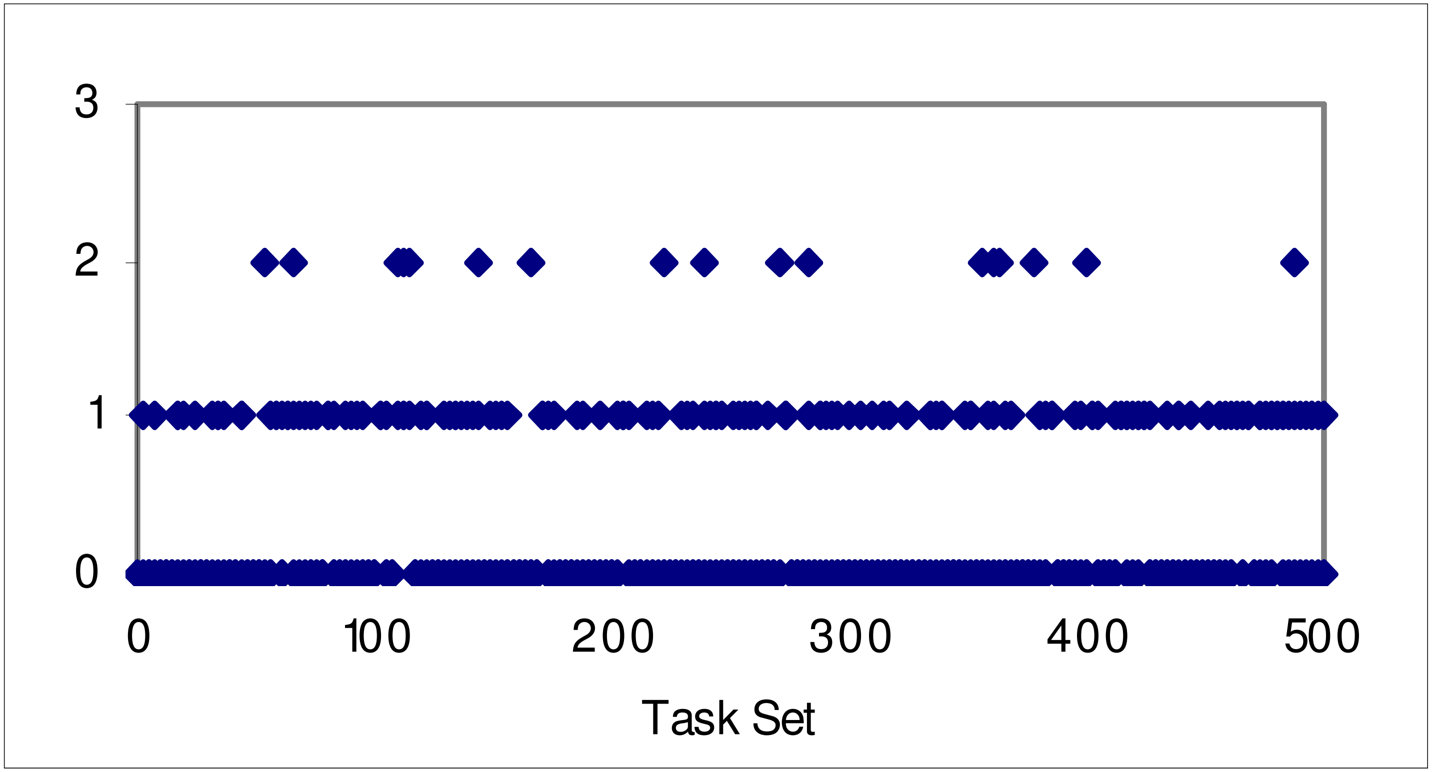
- Tasks are sorted based on a defined criterion
- First-fit scheme is used
- P-FRP exact schedulability test is used to identify if processor (bin) is 'full'
- Tasks are assigned to the next processor until it is 'full' and so on
- After last task in sorted order is assigned to a processor, partitioning is complete

Optimal Partitioning – Brute Force

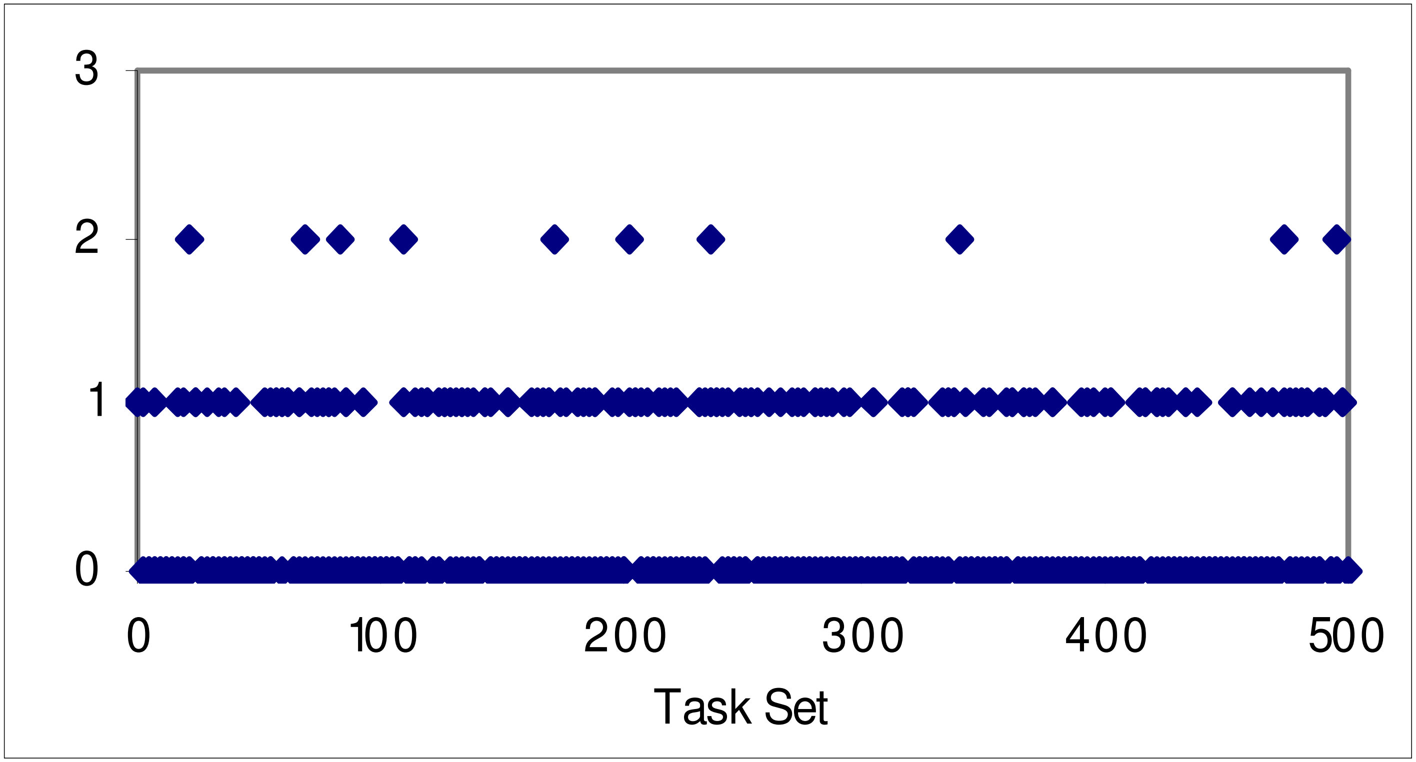


Combinatorial B-tree for enumeration all partitions in 'N' processors

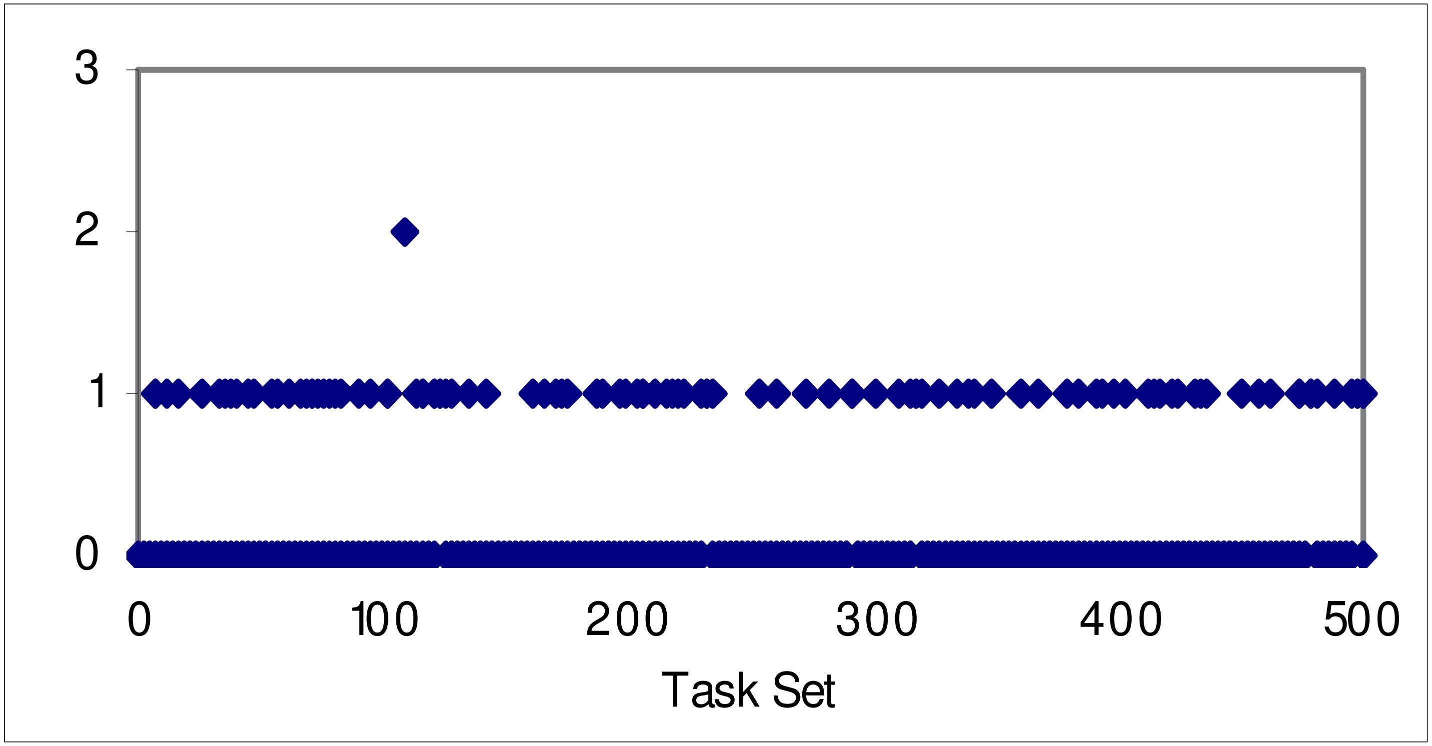
Difference with Optimal: FFDR



Difference with Optimal: FFDU



Difference with Optimal: FFDP



Remarks

- Applied first-fit partitioning using a new exact schedulability test for P-FRP
- Three sorting criterion used with first-fit algorithms
- Chaitanya Belwal and Albert M. K. Cheng, “Partitioned Scheduling of P-FRP in Symmetric Homogeneous Multiprocessors”, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC), Melbourne, Australia, Oct 24-26, 2011

Contents

Dynamic Voltage and Frequency Scaling

- Energy function of CMOS

$$E = C \cdot V^2 \cdot f$$

- Operating the CPU at a lower voltage consumes less energy
- Lowering the voltage decreases the number of CPU clock cycles available per unit time
- Goal is to save energy as well meet real-time guarantees
- Applying DVFS in P-FRP is different from preemptive execution

Contributions

- Derived Schedulability conditions for Static DVFS
- Presented algorithm for Progressive Voltage Scale (PVS)
- Presented the Voltage Scaling Points (VSP) algorithm
- Experimental evaluations and comparison between each approach

Variable Voltage Scheduling with P-FRP

In static-mode DVFS, the task set operates on a single scaled CPU voltage, which we set before the start of task execution. The voltage is kept constant as long as the task set is unchanged. Before setting the scaling voltage, it is necessary to determine if the task set can be scheduled under the scaled voltage.

Algorithm 2 Static-Mode DVFS Exact Schedulability Test

Input: Γ_n, α

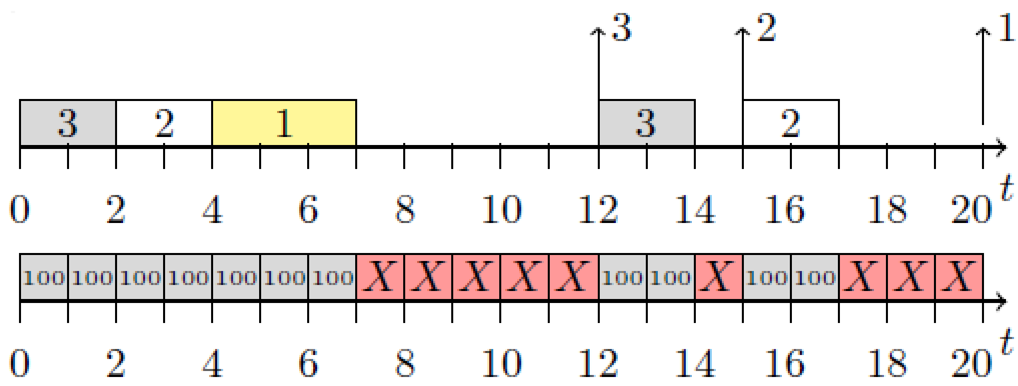
Output: True/False, (schedulable or not)

for each $\tau_i \in \Gamma_n$ **do**
    $C_i = \lceil \frac{c_i}{\alpha} \rceil$
    if $\Omega(\Gamma_n, [0, LCM))$ is true **then**
        return true
    else
        return false
    end if
end for

Variable Voltage Scheduling with P-FRP

Example : Normal execution of this task set without any voltage scaling.

Task	C	T
τ_1	3	20
τ_2	2	15
τ_3	2	12

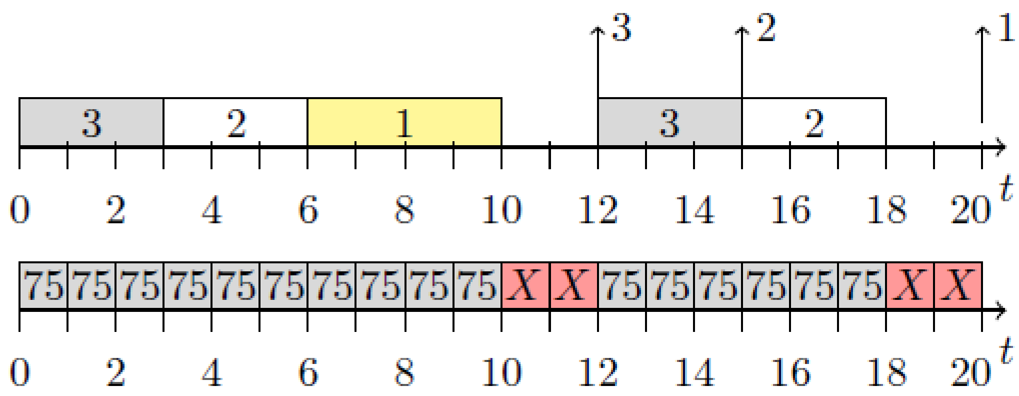


- **Figure 1.** Task execution and voltage graph showing the execution of Γ_3 in normal mode without voltage change. The numbers 1, 2 and 3 represent tasks τ_1, τ_2 and τ_3 respectively. In normal mode the system is assumed to run at a voltage scaling factor of 1 (100%). Power consumed for X is P_{idle}.

Variable Voltage Scheduling with P-FRP

Example : Static Voltage Scaling (The total power that is consumed in the feasibility interval in normal execution is 29.06, while with static voltage scaling, it is 23.25.

Task	C	T
τ_1	3	20
τ_2	2	15
τ_3	2	12

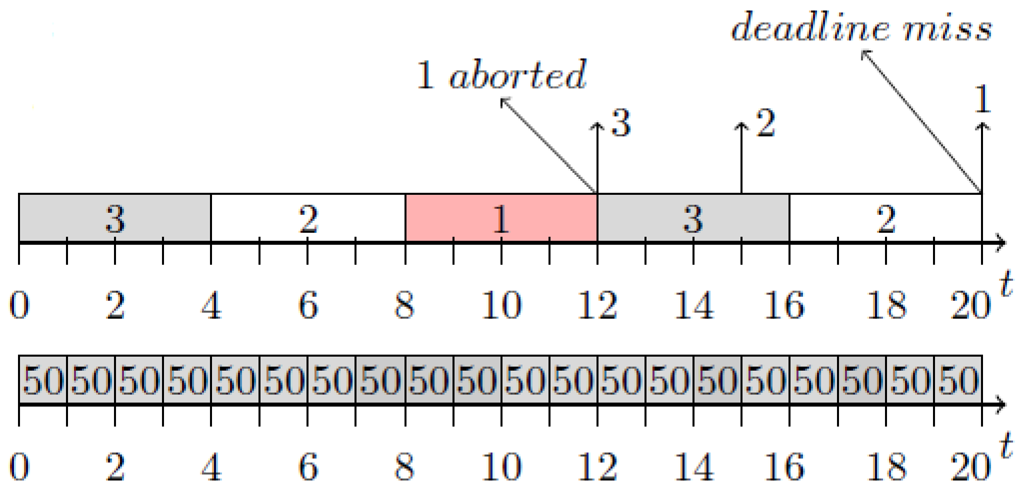


- **Figure 2.** Execution of Γ_3 with static voltage scaling using a scaling factor of 75%.

Variable Voltage Scheduling with P-FRP

Example : Static Voltage Scaling (If the voltage is scaled to 50%, the first job of 1 will have a deadline miss at time 20).

Task	C	T
τ_1	3	20
τ_2	2	15
τ_3	2	12

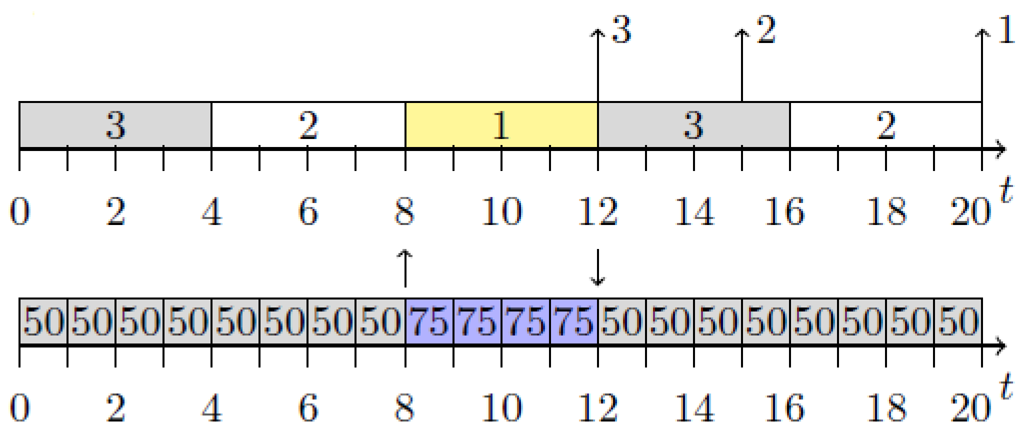


- **Figure 3.** Task τ_1 has a deadline miss at time 20 with scaling factor 50%.

Variable Voltage Scheduling with P-FRP

Example : Progressive Voltage Change (During the execution of tasks 2 and 3, the voltage can be scaled down to a factor of 0.5, while during execution of 1 the voltage can be scaled down to a factor of 0.75 . The total power consumed in the feasibility interval is 17.06.

Task	C	T
τ_1	3	20
τ_2	2	15
τ_3	2	12

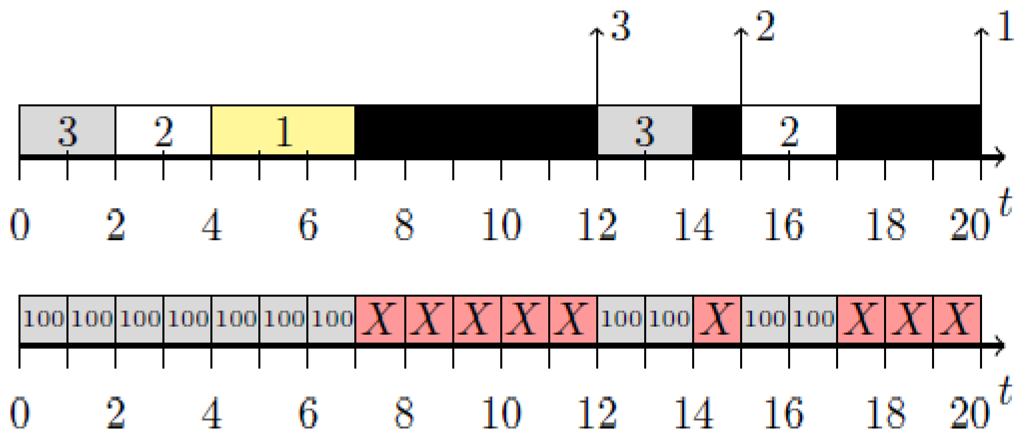


- **Figure 4.** Task execution with progressive voltage scaling.

Variable Voltage Scheduling with P-FRP

Example : The figure below shows the level-1 idle periods. The black areas identify those idle periods present in the feasibility interval.

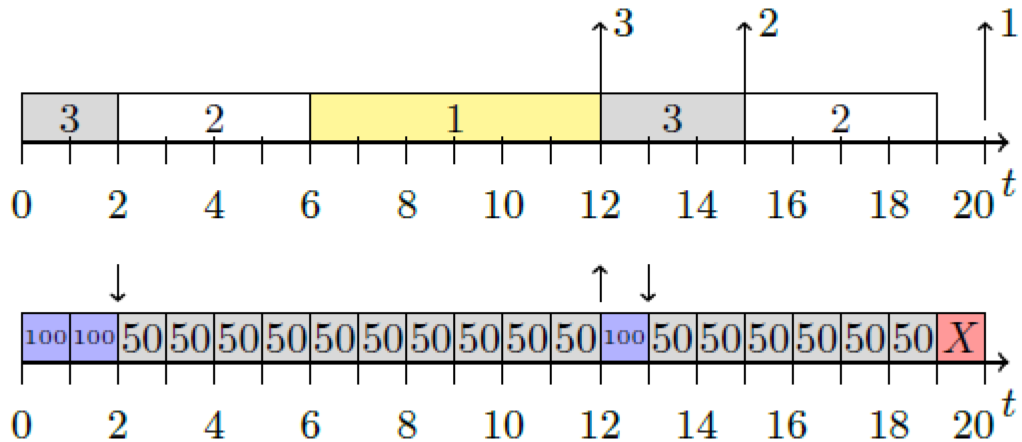
Task	C	T
τ_1	3	20
τ_2	2	15
τ_3	2	12



Variable Voltage Scheduling with P-FRP

Example : Voltage Scaling Points (The total power that is consumed is 17.25)

Task	C	T
τ_1	3	20
τ_2	2	15
τ_3	2	12



- **Figure 6.** Task execution with Voltage Scaling Points. The up/down arrows in the voltage graph show the voltage scale up and scale down points respectively.

Variable Voltage Scheduling with P-FRP

Experiments:

Tested from 100 to 500 task sets with different configurations.

Utilization factors for these tasks were in the range [0:22 to 0:65] and execution times and arrival periods were selected from the ranges [3 to 70] respectively.

Variable Voltage Scheduling with P-FRP

Experiments: For the static voltage scaling, 15% to 25% savings was achieved for maximum task sets.

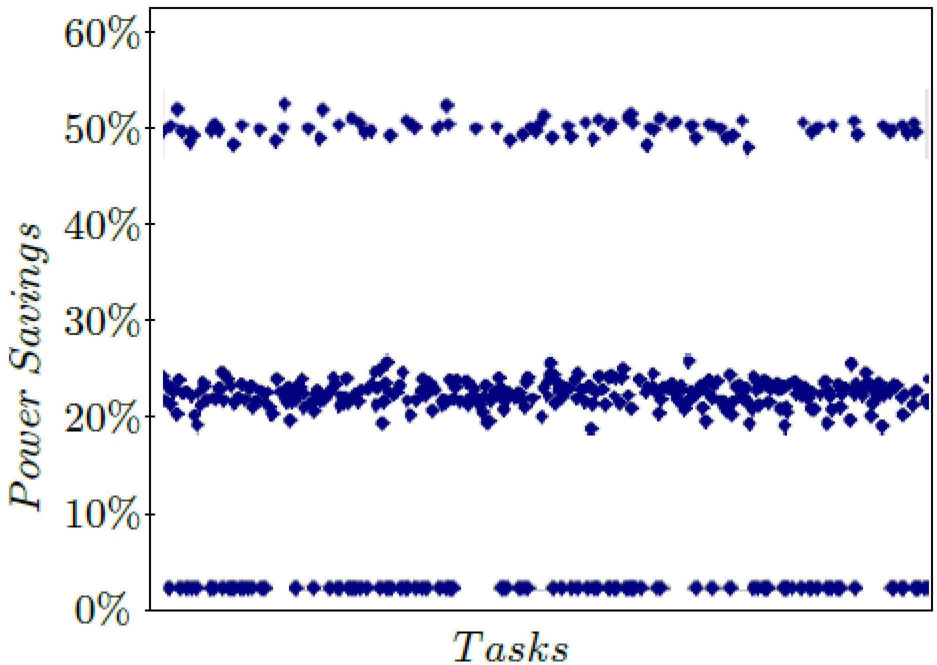


Figure 7. % of Power Savings: Static Voltage Scaling

Variable Voltage Scheduling with P-FRP

<u>Experiments</u>: PVS produced a more distributed range, with voltage savings for 500 task sets in the range of 0-52%.

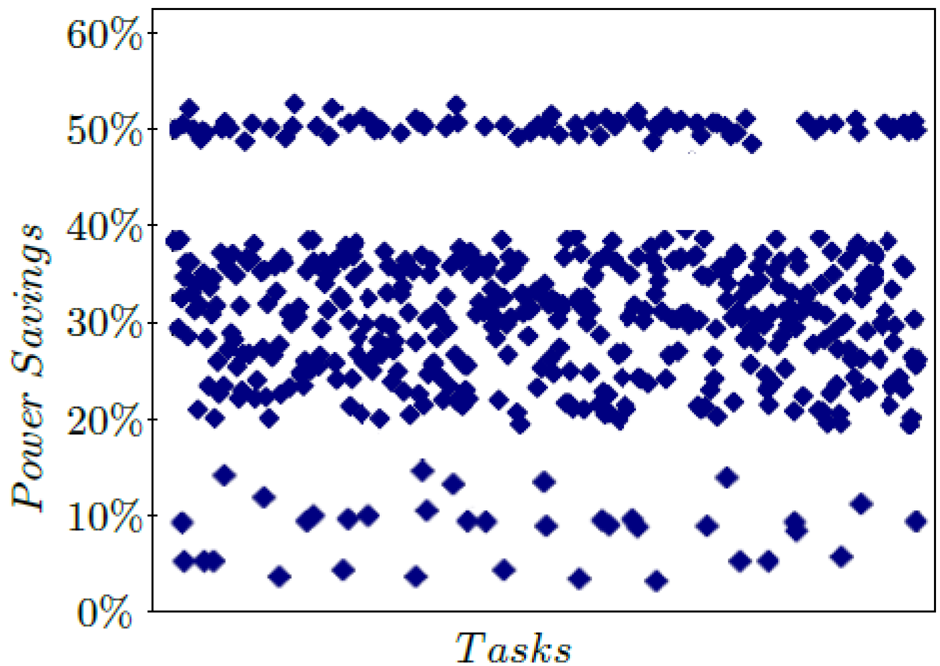


Figure 8. % of Power Savings: Progressive Voltage Scaling

Variable Voltage Scheduling with P-FRP

Experiments: VSP produced voltage savings in the range 26-52%.

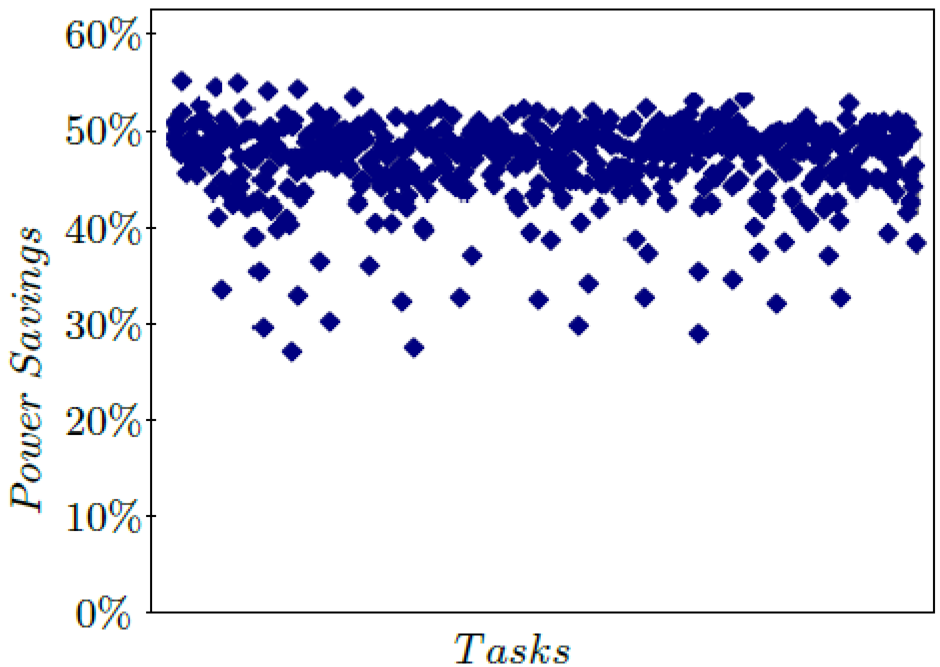


Figure 9. % of Power Savings: Voltage Scaling Points

![CS@UH]

Variable Voltage Scheduling with P-FRP

Experiments:

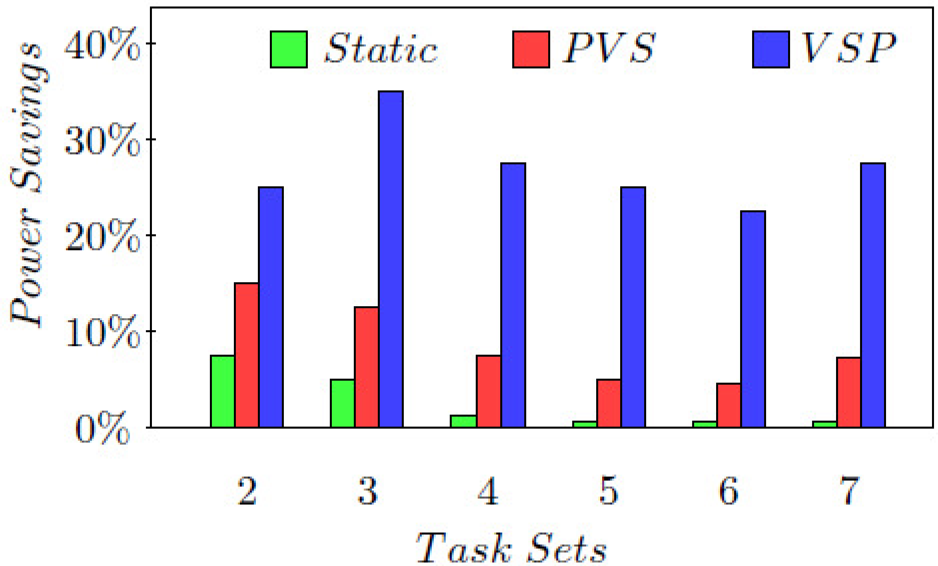


Figure 10. Percentage of Power Savings (Y-axis) with All DVS Algorithms with Different Task Set Sizes (X-axis) and Utilizations in the Range (0.51,1)

Variable Voltage Scheduling with P-FRP

Experiments:

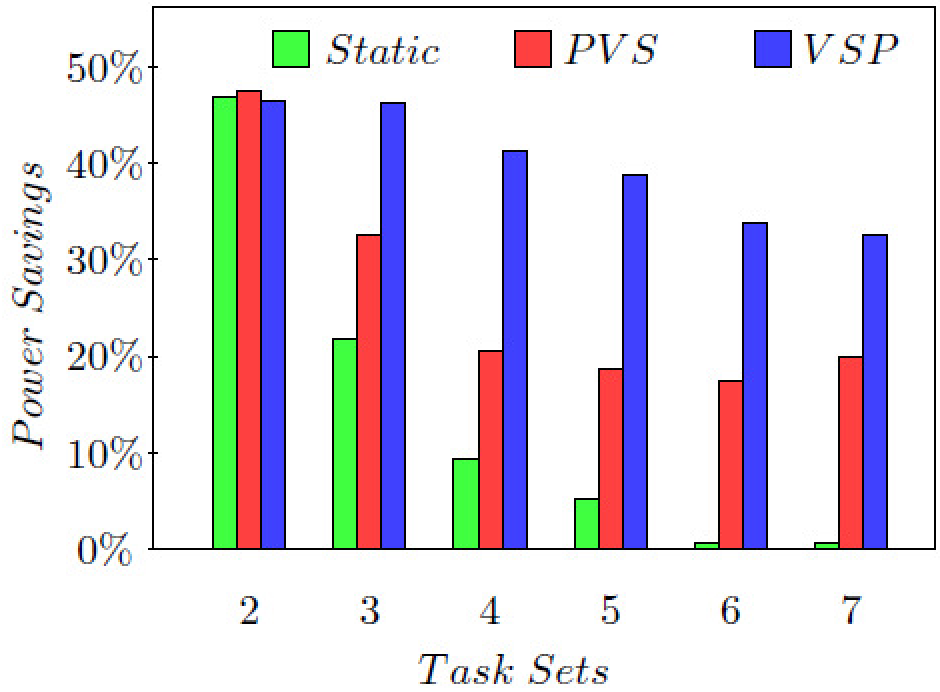


Figure 11. Percentage of Power Savings (Y-axis) with All DVS Algorithms with Different Task Set Sizes (X-axis) and Utilizations in the Range (0.1,0.5)

Remarks

- DVFS can lead to significant energy savings
- VSP gives the best results
- Did not consider leakage current
- Chaitanya Belwal and Albert M. K. Cheng, “Optimizing Energy Use in P-FRP through Dynamic Voltage Scaling”, 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Chicago, IL, USA, part of the Cyber-Physical Systems Week (CPS Week), April 11-14, 2011

Contents

Contributions

- Developed Timed Automata (TA) models for schedulability analysis of P-FRP
- Prove that TA models offers an efficient alternative for schedulability analysis of P-FRP
- Use a publicly available tool for TA modeling
- Validate correctness through experimental task sets

Timed Automata

- Developed by Alur and Dill in 1994
- Extends finite state automata by using clocks
- Extended Timed Automata (ETA): states represent the execution of tasks (Fersman et al)
- ETA - standard for representing schedulability models using Timed Automata
- Used in this work

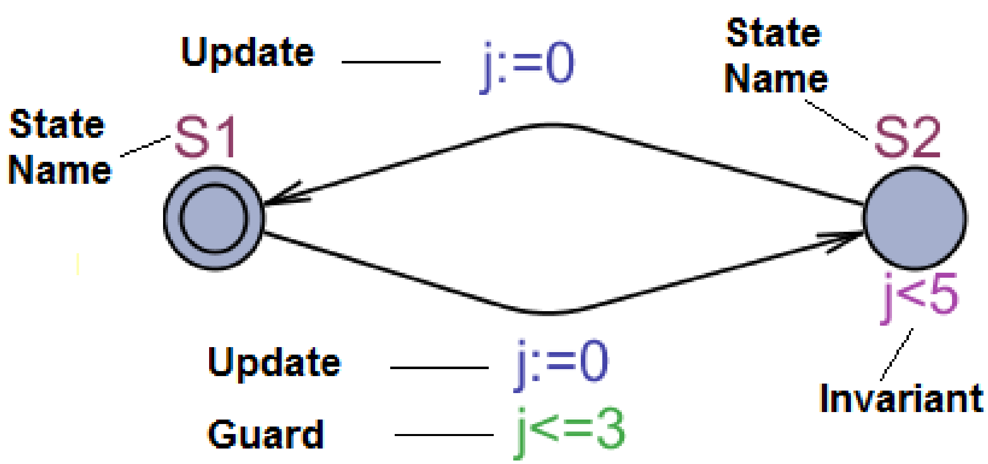
UPPAAL

- Developed at Aalborg and Uppsala Universities
- GUI-based tool
- Allows the description and evaluation of a Timed Automata (TA) model
- Several automata can run in parallel
- Allows **user variables** and **synchronization channels**

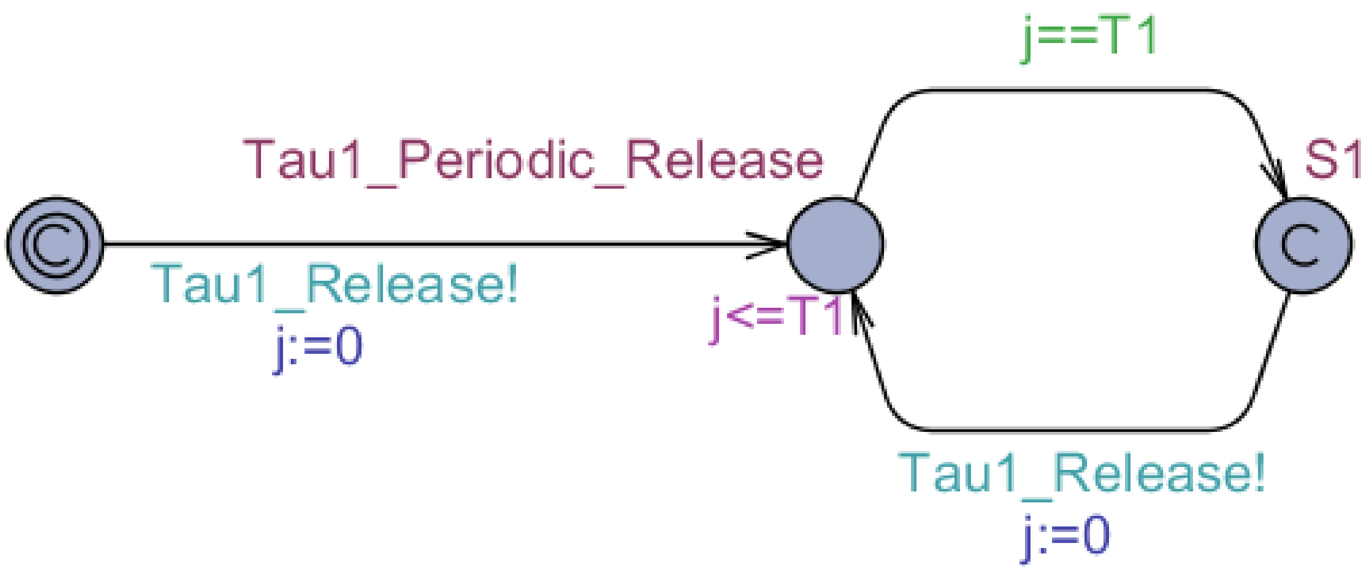
UPPAAL

- Transitions between locations are protected by clock guards
- **Invariants** - clock constraints in locations
- User declared variables can change value
- All TA encodings in our work have been tested in UPPAAL
- More details: http://www.uppaal.org

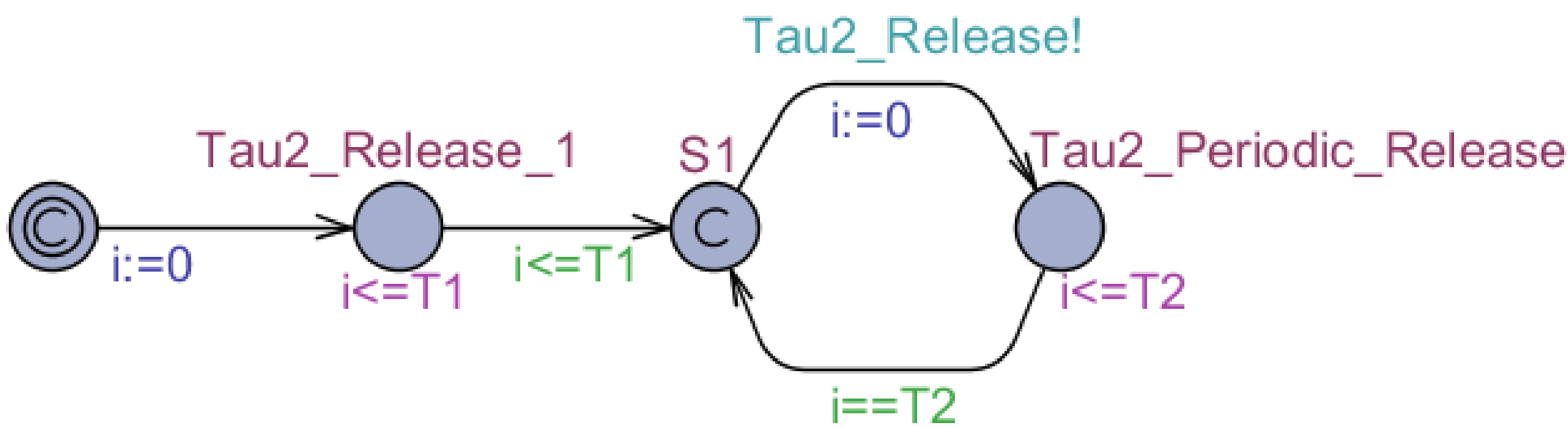
UPPAAL – Simple Automata



Task Release Automaton – Lowest Priority Task



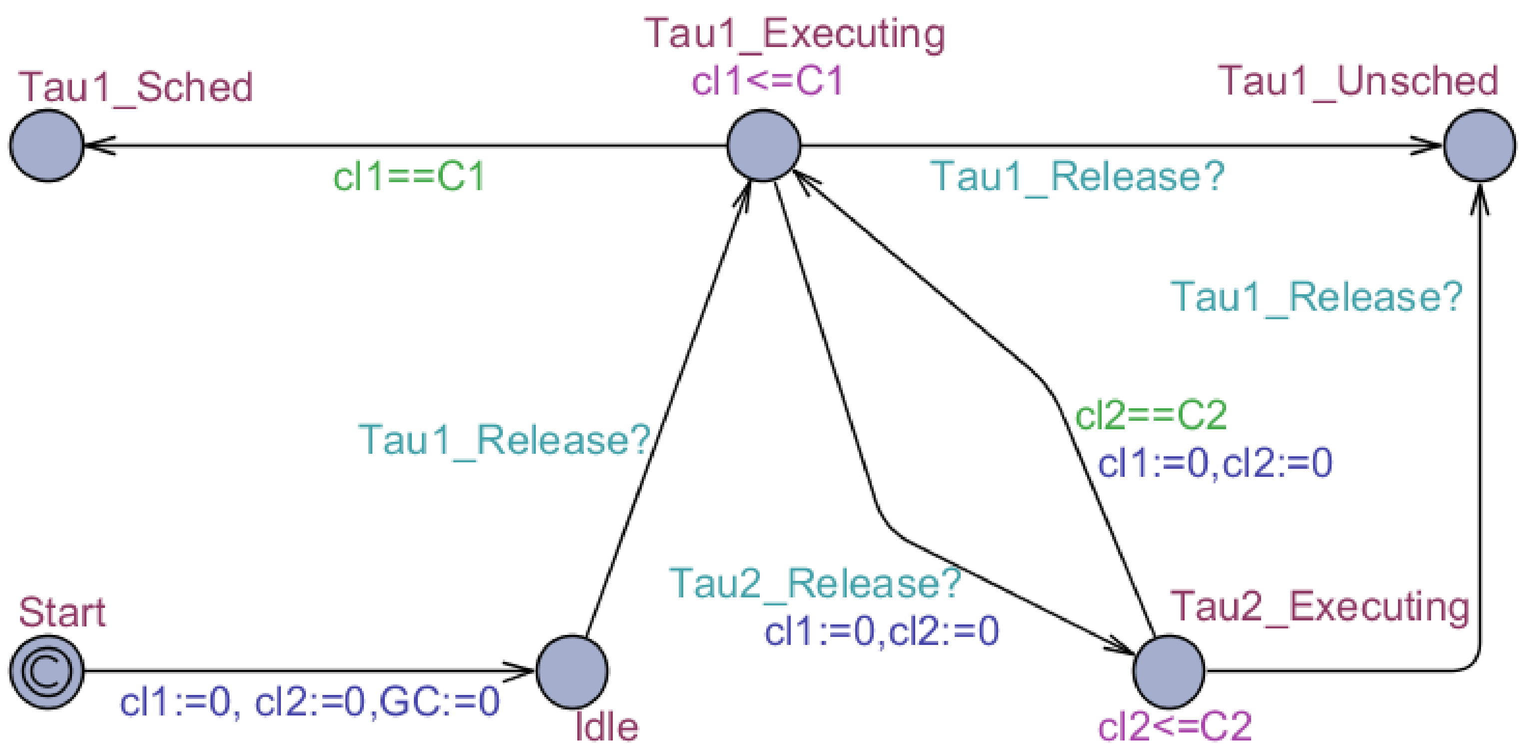
Task Release Automaton – Other tasks



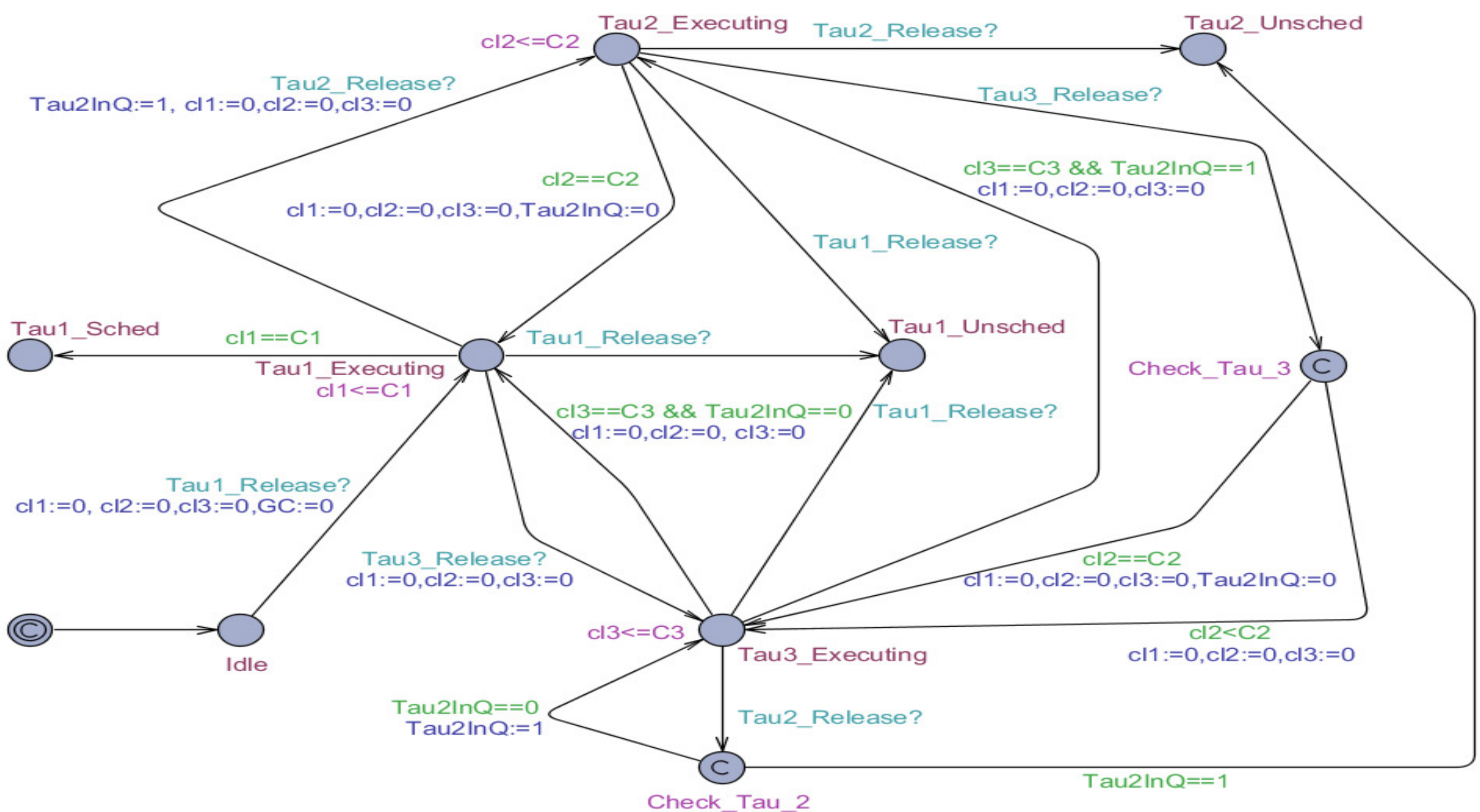
Generic Variables

- GC
- cl*i*
- T*i*
- C*i*
- Tau*j*InQ

Scheduler Automaton - 2 Tasks



Scheduler Automaton - 3 Tasks



Schedulability Analysis

- Schedulability analysis is same as determining the reachability of state 'Tau*i*_Unsched'
- Achieved by the following Computation Tree Logic (CTL) query:

  E<> Scheduler.Tau*i*_Unsched
- Should return *false* for task *i* to be schedulable

Schedulability Analysis

- Determine the schedulability of *n*-task set
- Query needs to be executed for every lower priority task
- Example for the 2 task automaton following CTL should return *false:*

    E<> Scheduler.Tau1_Unsched

- For the 3-task automaton the following queries should return *false*:

    E<> Scheduler.Tau1_Unsched

    E<> Scheduler.Tau2_Unsched

Remarks

- Schedulability analysis in P-FRP is difficult
- Current techniques scales exponentially with task size
- We have derived an alternate approach using TA and validated it
- Chaitanya Belwal and Albert M. K. Cheng, “Schedulability Analysis of Transactions in Software Transactional Memory using Timed Automata”, 8th IEEE International Conference on Embedded Software and Systems (ICESS), Changsha, China, Nov. 16-18, 2011

Contents

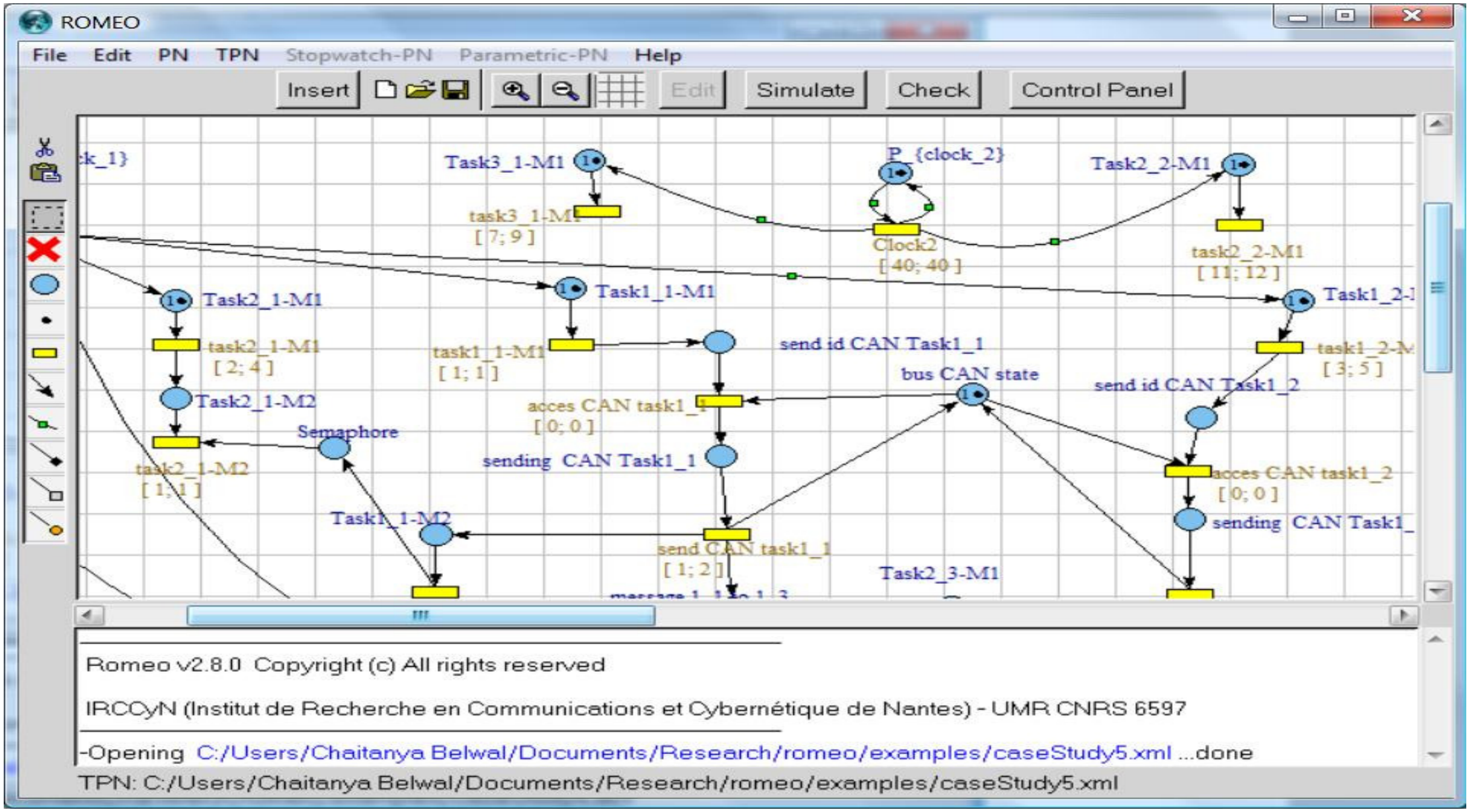
Contributions

- Developed Time Petri Net (TPN) models for schedulability analysis of P-FRP
- Prove that TPN models offers an efficient alternative for schedulability analysis of P-FRP
- Prove that conversion to corresponding TA models is not required
- Use a publicly available tool for TPN modeling
- Validate correctness through experimental task sets

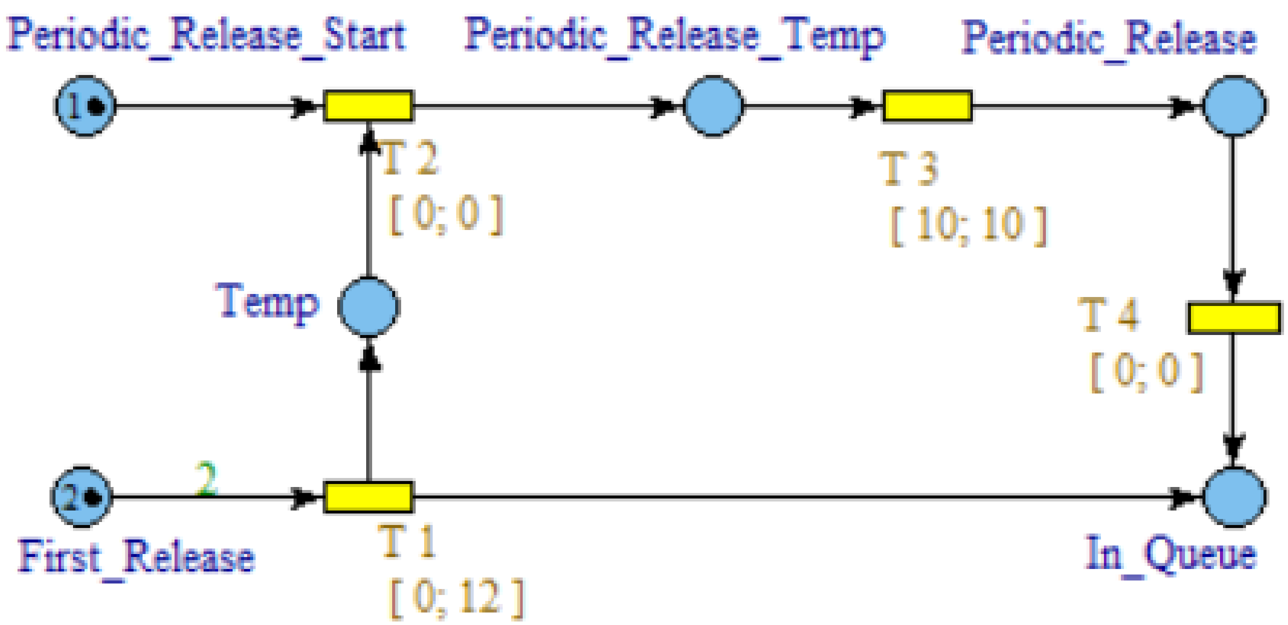
Time Petri Nets

- A Time Petri Net (TPN) is a tuple *(P, T,* B, *F,* M_O, *SI)* where:
- $P=\{p_1,p_2,p_3,\ldots,p_n\}$ is a finite non-empty set of places; T= $\{t_1,t_2,t_3,\ldots,t_n\}$ is a finite nonempty, set of transitions
- B: $P \times T \rightarrow N$ is the **backward incidence** function; where ***N*** is the set of non-negative integers; F: $T \times P \rightarrow N$ is the **forward incidence** function
- M_O is the **initial marking** *(P, T, B, F* and M_O together define a Petri net)
- *SI* is a mapping called **static interval**, $\forall t \in T, SI(t) = [SEFT(t), SLFT(t)]$, where $SEFT(t)$ is the **static earliest firing time** and $SLFT(t)$ the **static latest firing time**

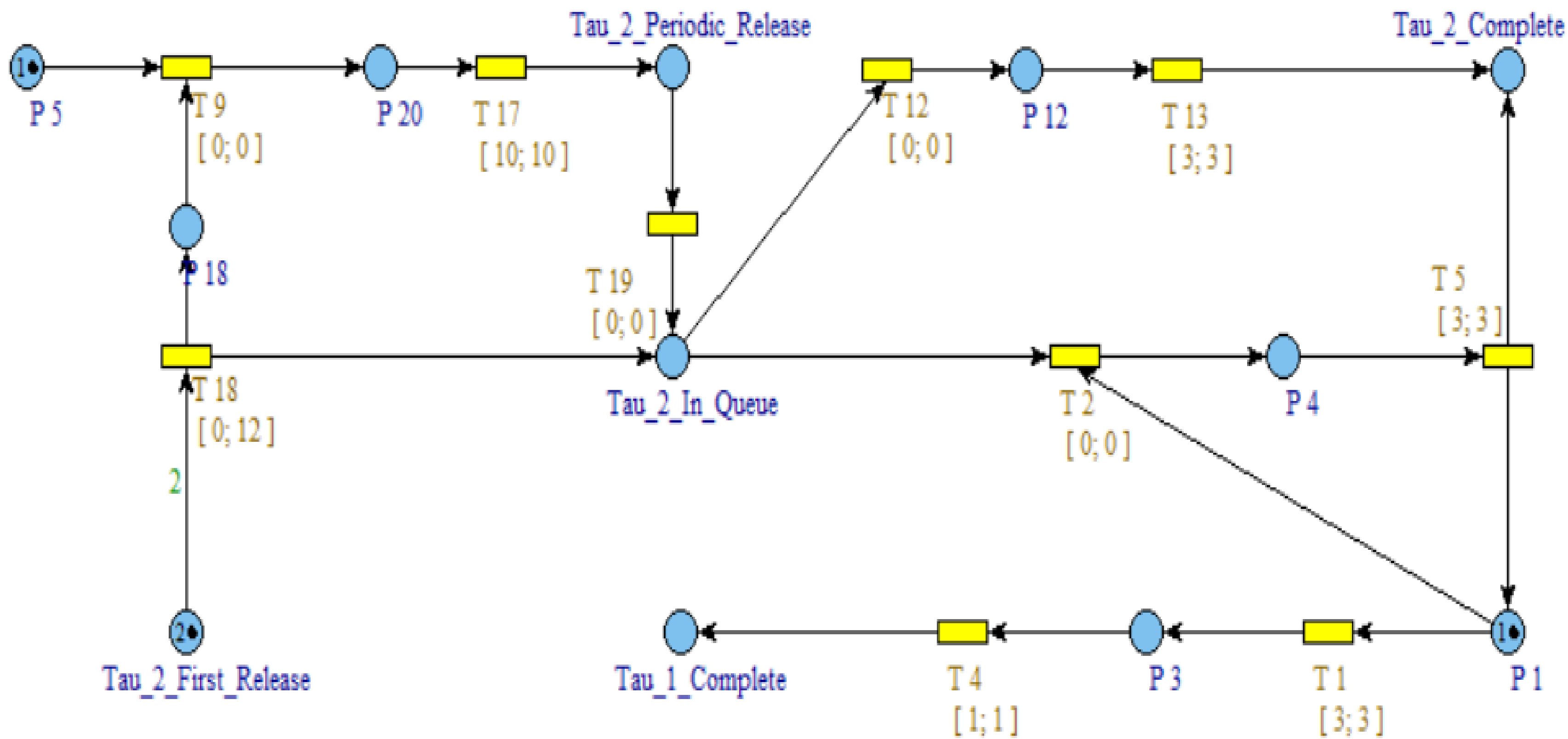
ROMEO – Tool for TPN



TPN – Periodic Task Release

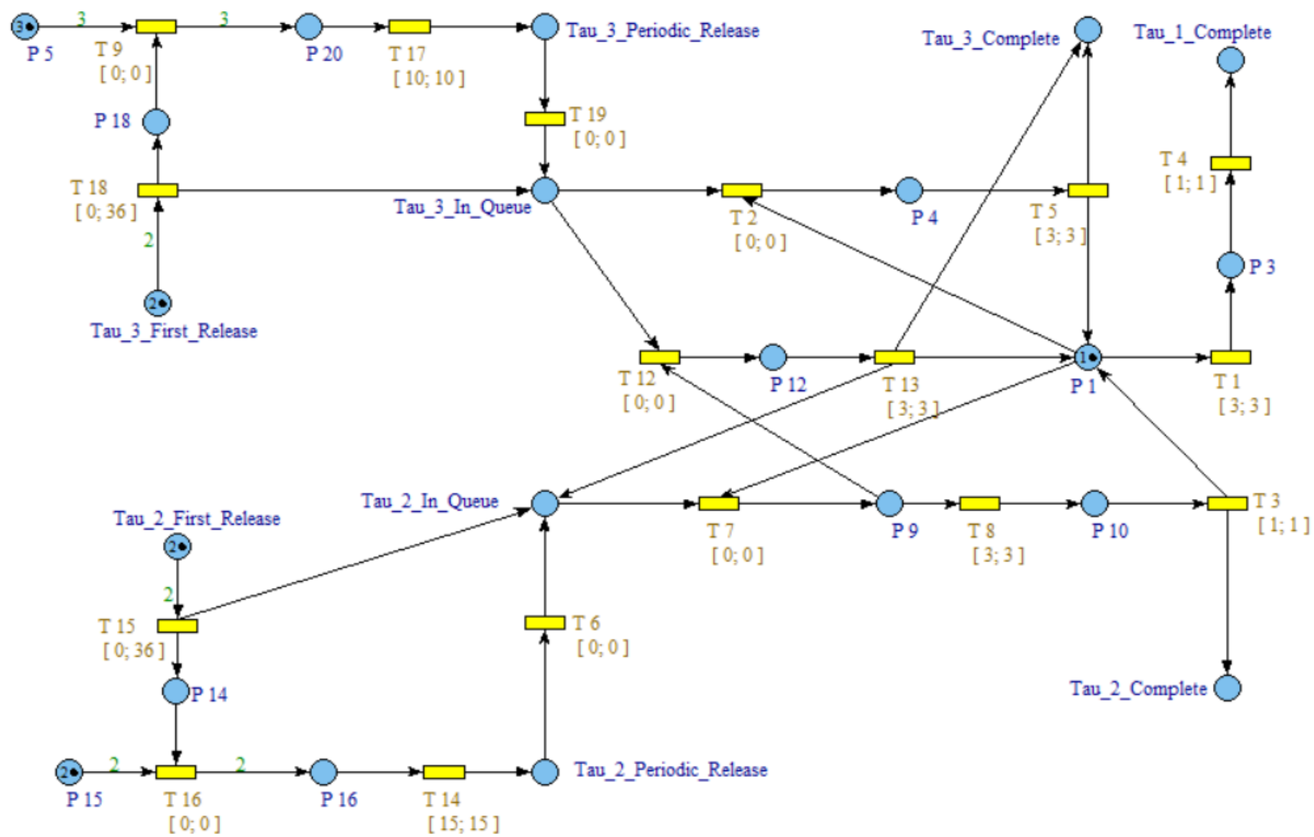


TPN – 2 Tasks



P 5
T 9
[0; 0]
P 20
T 17
[10; 10]
Tau_2_Periodic_Release
T 12
[0; 0]
P 12
T 13
[3; 3]
Tau_2_Complete
P 18
T 19
[0; 0]
T 5
[3; 3]
T 18
[0; 12]
Tau_2_In_Queue
T 2
[0; 0]
P 4
2
Tau_2_First_Release
Tau_1_Complete
T 4
[1; 1]
P 3
T 1
[3; 3]
P 1

TPN – 3 Tasks



P 5
3
T 9
[0; 0]
3
P 20
T 17
[10; 10]
Tau_3_Periodic_Release
Tau_3_Complete
Tau_1_Complete
P 18
T 19
[0; 0]
T 4
[1; 1]
T 18
[0; 36]
Tau_3_In_Queue
T 2
[0; 0]
P 4
T 5
[3; 3]
2
P 3
Tau_3_First_Release
T 12
[0; 0]
P 12
T 13
[3; 3]
P 1
T 1
[3; 3]
Tau_2_In_Queue
Tau_2_First_Release
T 7
[0; 0]
P 9
T 8
[3; 3]
P 10
T 3
[1; 1]
2
T 15
[0; 36]
T 6
[0; 0]
P 14
Tau_2_Complete
P 15
2
T 16
[0; 0]
2
P 16
T 14
[15; 15]
Tau_2_Periodic_Release

Schedulability Analysis

- TPN is converted to corresponding state space
- Uses Timed CTL queries
- *EF[37,37](M(22)=0)*
- At time 37 is it possible for place at index 22 (i.e. ‘Tau_1_Complete’) to have no tokens ?
- No token => release scenario of higher priority tasks exists in which τ_1 misses it deadline
- For τ_1 to be schedulable, the query should be *false*

Remarks

- TPN offers an efficient alternative to schedulability analysis
- TPNs for large models can be complicated
- Chaitanya Belwal and Albert M. K. Cheng, "Schedulability Analysis of P-FRP using Time Petri Nets", 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) WiP Session, Toyama, Japan, August 28-31, 2011

Future Work

- Modify techniques to consider variables times for copy and restore operations
- Develop pruning techniques to reduce the number of release scenarios in determining exact WCRT
- Improve the polynomial time method for greater accuracy (lower the approximation factors)
- Develop an algorithm for finding the specific optimal priority assignment for any *n*-task set
- Develop global partitioning algorithms for P-FRP tasks in multi-processor platforms

Future Work

- Experimentally evaluate multi-processor partitioning schemes in hardware
- Implement DVFS algorithms in the Real-Energy platform
- Modify DVFS algorithms to consider leakage current
- Modify TA and TPN models for easier scalability
- Formally prove if exact WCRT can be determined/or not determined in polynomial time
- Extend this work to STM and lock-free execution as well as general scheduling theory (job-shop)

Evaluation

- Does precise timing characterization of the embedded controller software execution lead to faster physical system response compared with one designed without accurate controller timing analysis (and thus requires more tolerance of execution time variations)?
- How does the time to develop new control components with accurate response time analysis tools compare to doing the same with older methods?
- Automotive application: Do the new scheduling/execution such as AWR lead to safer physical system behaviors such as shorter stopping distance for ABS-equipped cars?
- Do optimizations to the runtime controller software such as reducing event-handler preemptions and better priority assignments result in faster controller response as measured by developed analytical methods, simulation, and actual physical system testing?

Evaluation

- Does the inclusion of power-aware and power-saving measures maintain the satisfaction of timing and space/memory constraints imposed on the embedded controller and controlled physical system behaviors? What is the amount of energy savings in the physical system and embedded controller achieved with these approaches compared with systems without them?
- Does the resulting approach make it easier and safer to make minor modifications to components of the control systems?
- Does this framework and toolset facilitate the design of the controller and its timing/safety verification? Is the time from design to actual implementation shortened and the development cost lowered?

Concluding Remarks

- Our goal: Enhance the safety and performance of a physical system controlled by an embedded controller consisting of single or networked control components with functional reactive programming (FRP).
- FRP allows intuitive specification and formal verification of safety-critical behaviors, thus reducing the number of defects injected during the design phase, and the stateless nature of execution avoids the need for complex programming involving synchronization primitives.
- Accurate response time analysis tools (accounting for CPU execution, memory access, I/O, and sensor processing times), novel scheduling techniques, and new power-conserving methods are needed.
- Research impact: Facilitate the design and update of the embedded controller (or network of controllers) as well as its (their) timing and safety verification.

Thank you!

Comments?
Questions?

References (1)

- Andrei S., Mozahid H., and Cheng A.M.K., "Optimizing the Linear Real-Time Logic Verifier," 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Philadelphia, PA, April 8, 2013.
- Andrei S. and Cheng A.M.K., "Decomposition-based Verification of Linear Real-Time Systems Specifications," 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), Washington, D.C., USA (Co-located with IEEE RTSS 2009), December 1, 2009.
- Andrei S. and Cheng A.M.K., "Efficient Verification and Optimization of Real-Time Logic Specified Systems," IEEE Transactions on Computers, vol. 58, no. 12, pp. 1640-1653, December 2009.
- Andrei S., Chin W., Lupa M., Cheng A.M.K., "Automatic Debugging of Real-Time Systems Based on Incremental Satisfiability Counting," IEEE Transactions on Computers, Vol. 55, No. 7, pp. 830-843, July 2006. Selected as this issue's featured article.
- Andrei S., Mozahid H., and Cheng A.M.K., Optimizing the Linear Real-Time Logic Verifier," 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Philadelphia, PA, April 8, 2013.

References (2)

- Andrei S., Radulescu V., McNicholl T., Cheng A.M.K., "Toward an optimal power-aware scheduling technique," 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, September 26-29, 2012.
- Belwal C. and Cheng A.M.K., "Chaitanya Belwal, Albert M. K. Cheng, and Bo Liu, `` Feasibility Interval for the Transactional Event Handlers of P-FRP," Special Issue on UbiSafe Computing and Communications, Elsevier's Journal of Computer and System Sciences, 2012.
- Belwal C. and Cheng A.M.K., "Determining Actual Response Time in P-FRP," Proc. Thirteenth International Symposium on Practical Aspects of Declarative Languages (PADL), Austin, Texas, USA, pages: 250-264, January 24-25, 2011.
- Belwal C. and Cheng A.M.K., "Determining Actual Response Time in P-FRP using Idle-Period Game Board," Proc. 14th IEEE International Symposium on Object, Component, and Service-Oriented Real-time Distributed Computing (ISORC), Newport Beach, CA, USA, pages: 136-143, March 28-31, 2011.

References (3)

- Belwal C. and Cheng A.M.K., "Response Time Bounds for Event Handlers in the Priority-based Functional Reactive Programming (P-FRP) Paradigm," ACM Research in Applied Computation Symposium (RACS), San Antonio, Texas, USA, October 23-26, 2012.
- Cheng A.M.K, Niktab H., and Walston M., "Timing Analysis of Small Aircraft Transportation System (SATS)," International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Seoul, Korea, August 2012.
- Ras J. and Cheng A.M.K., "Response Time Analysis of the Abort-and-Restart Model under Symmetric Multiprocessing." The 7th IEEE International Conferences on Embedded Software and Systems (ICESS-10)," pages: 1954-1961, 2010.
- Ras J. and Cheng A.M.K., "Response Time Analysis for the Abort-and-Restart Event Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm," Embedded and Real-Time Computing Systems and Applications (RTCSA-09), pages: 305-314, 2009.

References (4)

- Ras J. and Cheng A.M.K., "An Evaluation of the Dynamic and Static Multiprocessor Priority Ceiling Protocol and the Multiprocessor Stack Resource Policy in an SMP System," Proc. IEEE-CS Real-Time and Embedded Technology and Applications Symposium (RTAS), San Francisco, California, April 2009.
- Wen Y., Liu Z., Shi W., Jiang Y., Yang F., Kohar A., Cheng A.M.K., "Support for Power Efficient Mobile Video Playback on Simultaneous Hybrid Display,“ 10th IEEE Symposium on Embedded Systems for Real-Time Multimedia Tampere, Finland, October 11-12, 2012.
- Wen Y., Belwal C., Cheng A.M.K., "Response Time Bounds for Event Handlers in the Priority based Functional Reactive Programming (P-FRP) Paradigm," ACM Research in Applied Computation Symposium (RACS), San Antonio, Texas, 2012.
- Wen Y., Belwal C., Cheng A.M.K., "Time Petri Nets for Schedulability Analysis of the Transactional Event Handlers of P-FRP," ACM Research in Applied Computation Symposium (RACS), San Antonio, Texas, USA, October 23-26, 2012.
- Li Y. and Cheng A.M.K., "Static Approximation Algorithms for Regularity-based Resource Partitioning," 33rd Real-Time Systems Symposium (RTSS), San Juan, Puerto Rico, USA, December 4-7, 2012.

References (5)

- Chaitanya Belwal, Yuanfeng Wen and Albert M. K. Cheng, ``Utilization Bounds of P-FRP Tasks,'' to appear in International Journal of Embedded Systems, 2013.
- Yong woon Ahn, Albert M. K. Cheng, Jinsuk Baek, Minho Jo, and Hsiao-Hwa Chen, ``An Auto-Scaling Mechanism for Virtual Resources to Support Mobile, Pervasive, Real-Time, Healthcare Applications in Cloud Computing,'' IEEE Network, Sept. 2013.
- Chaitanya Belwal, Albert M. K. Cheng, and Bo Liu, `` Feasibility Interval for the Transactional Event Handlers of P-FRP,'' Special Issue on UbiSafe Computing and Communications, Elsevier's Journal of Computer and System Sciences, Volume 79, Issue 5, pages 530-541, August 2013.
- Yuanfeng Wen, Chaitanya Belwal, and Albert M. K. Cheng, ``Towards Optimal Priority Assignments for the Transactional Event Handlers of P-FRP,'' ACM International Conference on Reliable And Convergent Systems (RACS), Montreal, QC, Canada, October 1-4, 2013.
- Chaitanya Belwal, Albert M. K. Cheng, J. Ras, and Yuanfeng Wen, ``Variable Voltage Scheduling with the Priority-based Functional Reactive Programming Language,'' ACM International Conference on Reliable And Convergent Systems (RACS), Montreal, QC, Canada, October 1-4, 2013.