YASMIN: Yet Another Scheduling MIddleware for exploratioN

TEAMPLAY

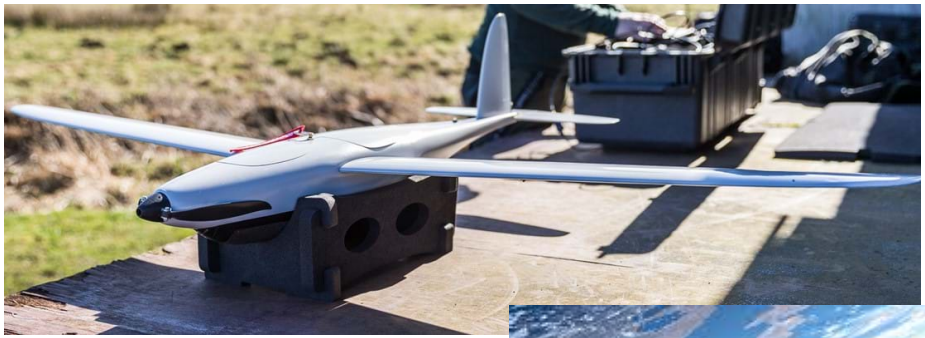
Time, Energy and security Analysis for Multi/Many-core heterogeneous PLAtforms

Benjamin Rouxel*
Sebastian Altmeyer†
Clemens Grelck*

University of Amsterdam*, Augsburg University†

Middleware, December 2021

Embedded Real-Time Systems



Embedded

- Application specific
- Dedicated systems

Real-Time

- Need to enforce the timing

Enforcing timing

Application

as tasks

- Periodic/Sporadic
- DAG/SDF
- Simple/Multi-version tasks

Enforcing timing

Application as tasks

- Periodic/Sporadic
- DAG/SDF
- Simple/Multi-version tasks

Schedulability analysis

- He et al, 2021
- Davis et al, 2009

Off-line scheduler

- Rouxel et al, 2019
- Roeder et al, 2020

Enforcing timing

Application as tasks

- Periodic/Sporadic
- DAG/SDF
- Simple/Multi-version tasks

Schedulability analysis

- He et al, 2021
- Davis et al, 2009

Off-line scheduler

- Rouxel et al, 2019
- Roeder et al, 2020

Schedule deployment

- OS-based
- Molisson et al, 2013

Two Scheduling Schemes

Goal

Deciding on which core and in which order tasks will run

Off-line

- Static scheduling & mapping
- Create a *gantt*-chart:
- Need on-line dispatcher

On-line

Two Scheduling Schemes

Goal

Deciding on which core and in which order tasks will run

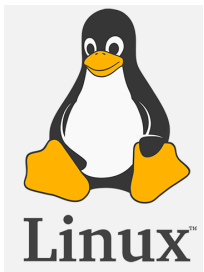
Off-line

- Static scheduling & mapping
- Create a *gantt*-chart:
- Need on-line dispatcher

On-line

- Dynamic scheduling & mapping
- Priority assignment
- Mapping strategy

How to deploy a schedule?



LITMUS^RT
Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Embedded With RTEMS
www.rtems.org

freeRTOS

ERIKA 3

Deploying schedules on Linux

The offer

- PREEMPT_RT patchset
- Schedulable element:
  - POSIX Thread
- Scheduling policies:
  - FIFO, RoundRobin, Deadline
- Periodicity:
  - Alarm, Busy waiting

- No task dependency
- No multi-version

- Very limited
- No off-line schedule support
- Limited #priorities

- Limited #alarms

- Switching between policies
- Adding new policies is hard

Brief overview of other solutions

OS schedulers

- Lower range of supported platforms
- Lack of heterogeneity support
- Limited task models
- Limited scheduling strategies
- More or Less easy to implement your own policy

Middleware

- Code not available/outdated
- Limited task models
- Limited to one or two strategies
- More or Less easy to implement your own policy

Principles

- Middleware library
- User-space scheduling
- Shielded processors
- Separate scheduler thread
- Unified API (switch at compile time)
- Schedule units: components/tasks

Off-line schedulers

- Follow a pre-computed schedule
- On-line dispatcher

On-line schedulers

- Scheduling decided at run-time
- Mapping can be done at run-time
- Priority ordered queue

Benefits of a Middleware

- Customisation: implement SOA scheduling algorithms
- Exploration: best fit scheduling algorithm
- Flexibility: task model support
- Adaptability: environmental constraints
- Maintainability: easy system upgrade
- Portability: easy system switch
- Control: timing control

Overview of YASMIN

- Written in C
- Target POSIX system
- Compiled & linked to end-user application
- Requires a configuration Header file
- No dynamic allocation
- Bounded loops

```
#include "constants.h"
#define PERIODIC_TASK_SIZE 1 /*there 1 periodic task: the fork
   (cont.)task*/
#define NONPERIODIC_TASK_SIZE 3  /*all other tasks are
   (cont.)activated depending on the presence of input data*/
#define CHANNEL_SIZE 4  /*there are 4 FIFO channels connecting
   (cont.)tasks*/
#define VERSION_MAX_SIZE 2  /*At most 2 versions are used*/
#define VERSION_SELECTION ENERGY /*adapt the structure and code
   (cont.) to select versions of task based on remaining energy.
   (cont.)*/
#define HWACCEL_SIZE 1  /*One hardware accelerator is used*/
#define MAPPING_SCHEME  GLOBAL  /*the example uses a global on-
   (cont.)line scheduler*/
#define PRIORITY_ASSIGNMENT EDF  /*priority are given using EDF
   (cont.)*/
#define THREADS_SIZE 2  /*2 worker threads will be used*/
```

Features

- Global/Partitioned mapping
- Off-line/On-line scheduling
- Off-line/On-line priority assignment
- Pre-emption
- Components migration
- Computing unit heterogeneity
- Automatic version selection
- ...

Task models

- Periodic/Sporadic tasks
- DAG tasks
- After transformation:
  Synchronous Dataflow graphs, Fork-join graphs, ...
- Multi-version tasks

Evaluations

Comparisons

- Scheduling overhead against Mollison and Anderson, RTAS'2013
- Task activation latency against Linux & Litmus

Application

- TeamPlay drone use-case from SkyWatch

In a nutshell

- User library
- Pre-emptive G-EDF scheduling policy
- Spawns threads, tasks are scheduled within threads
- No separate scheduler thread
- Test-and-set synchronisation of the global queue
- Rely on alarm
- Target Architecture: x86
- No heterogeneity
- Only periodic task model

Overhead vs Mollison and Anderson

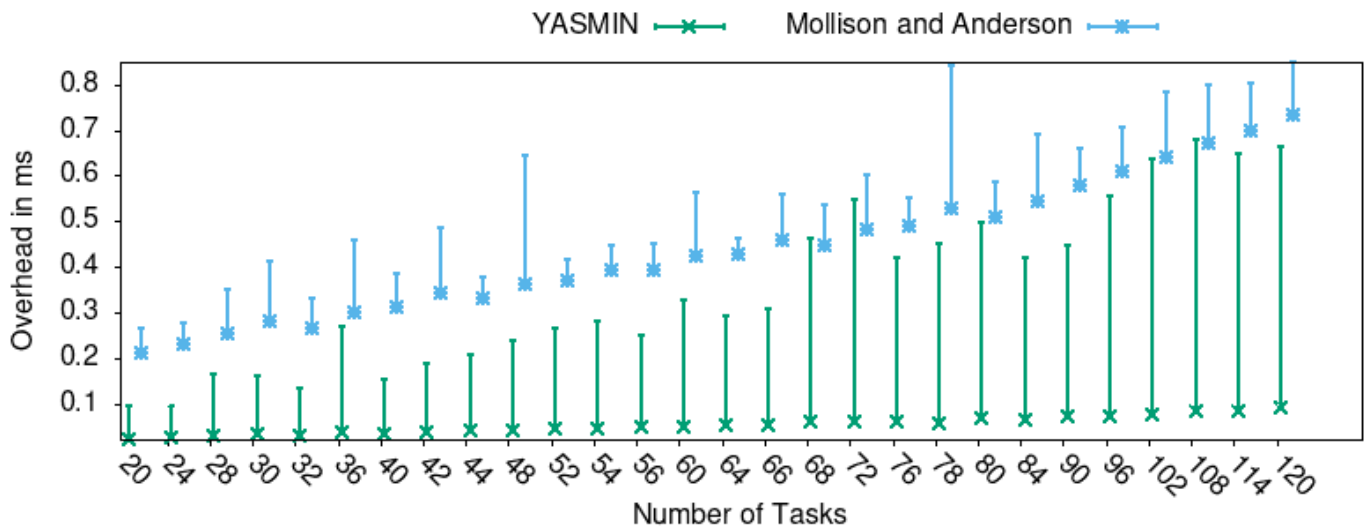
Experimental setup

- Platform
  - Odroid-XU4
  - ARM big.LITTLE 2x4 cores
  - Mali GPU
- Synthetic workload
  - #tasks: [20;120]
  - big cores: 2,3
    (+1 for our scheduler thread)
  - Utilisation: [0.2;2]
  - #tasksets: 5 per configuration

Overhead vs Mollison and Anderson

Experimental setup

- Platform
  - Odroid-XU4
  - ARM big.LITTLE 2x4 cores
  - Mali GPU
- Synthetic workload
  - #tasks: [20;120]
  - big cores: 2,3
    (+1 for our scheduler thread)
  - Utilisation: [0.2;2]
  - #tasksets: 5 per configuration



In a nutshell

- Measure accurately & precisely measure latency
- Sporadic tasks
- Available in the RT-apps collection by the Linux/PREEMPT_RT patch team
- Adapted to SCHED_DEADLINE, Litmus^RT, and Yasmin

Latency with *cyclictest*

Experimental Setup

- Spawns 6 threads, concurrently woken up 10000 times every 10ms
- Save 1 thread for our scheduler and 1 thread for the OS
- Generate interfering load: Stress-NG
  - Configured to spawn 8 threads that mess up with: cache, computation, timer events, scheduler.

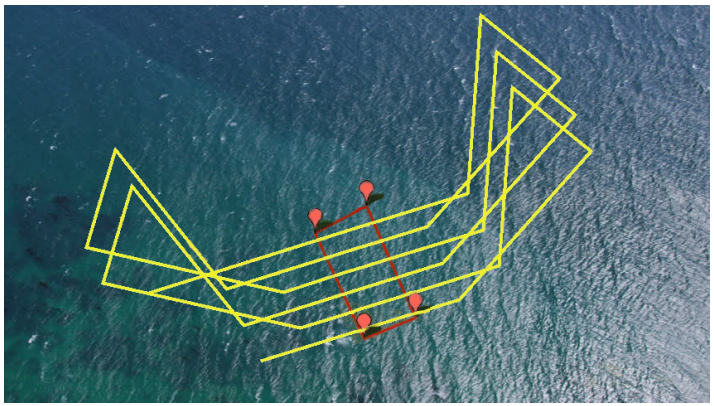
Latency with *cyclictest*

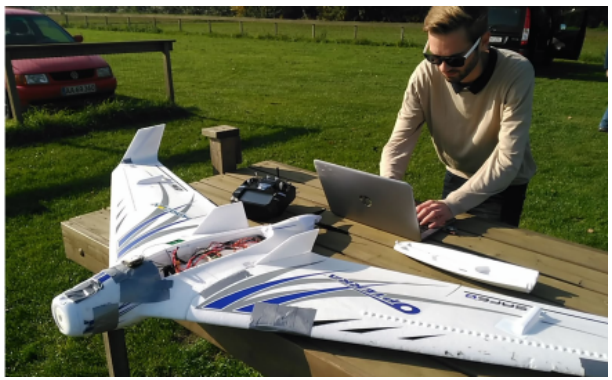
Experimental Setup

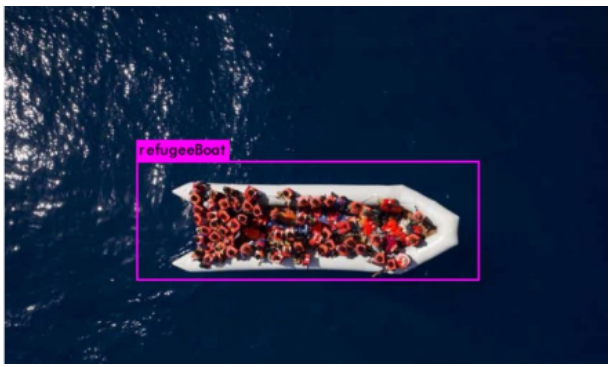
- Spawns 6 threads, concurrently woken up 10000 times every 10ms
- Save 1 thread for our scheduler and 1 thread for the OS
- Generate interfering load: Stress-NG
  - Configured to spawn 8 threads that mess up with: cache, computation, timer events, scheduler.

OS	*Cyclictest* version	Latency in μs $< min, max, avg >$
Linux +PREEMPT_RT 4.14.134-rt63	Yasmin	90, 1481, 500
	RTapps	176, 1550, 463
Litmus^RT 4.9.30-litmus	Yasmin	67, 318, 170
	RTapps	33, 222, 74
	litmus+GSN-EDF	35, 247, 84
	litmus+P-RES	988, 1206, 1027

Industrial Use-case

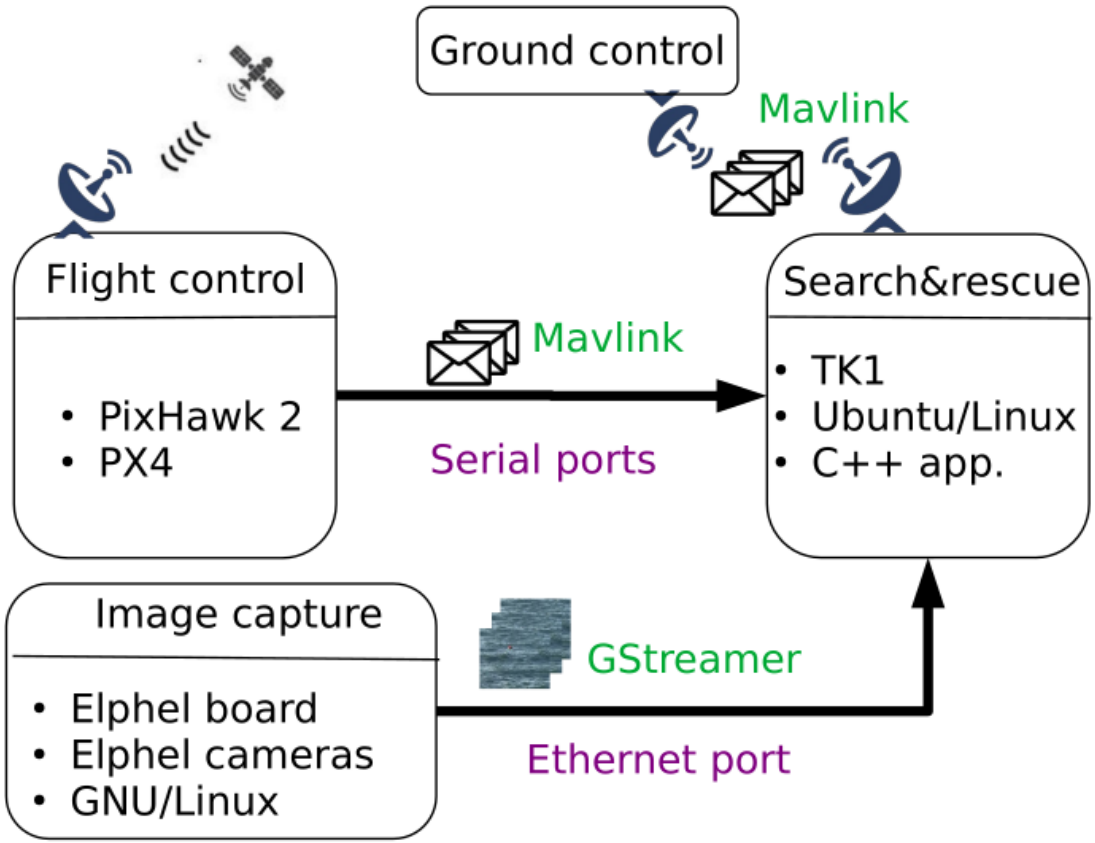




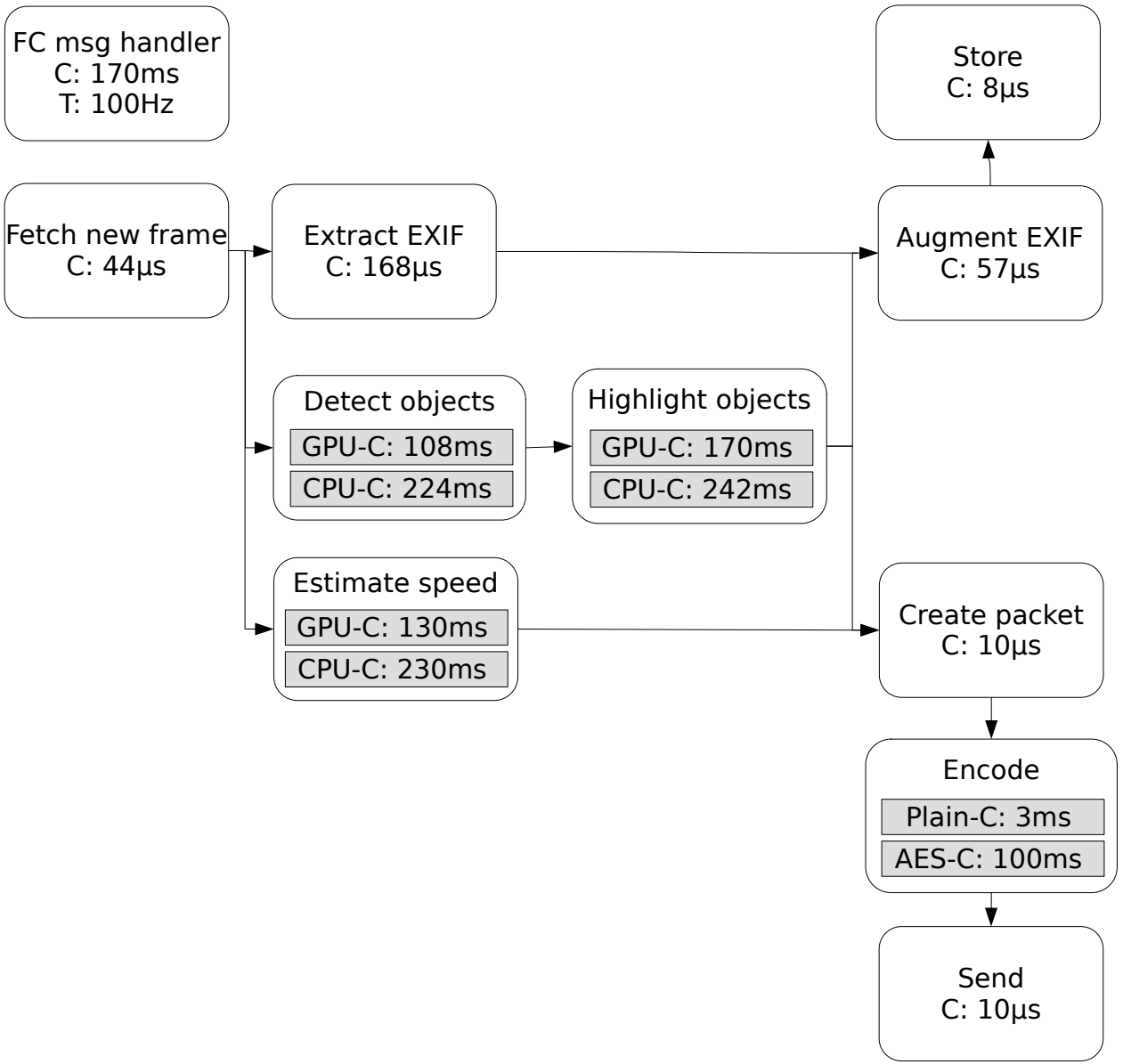


- Fixed-wings drone
- Fly above the sea
- Detect life boats
- Contact rescue teams

System overview



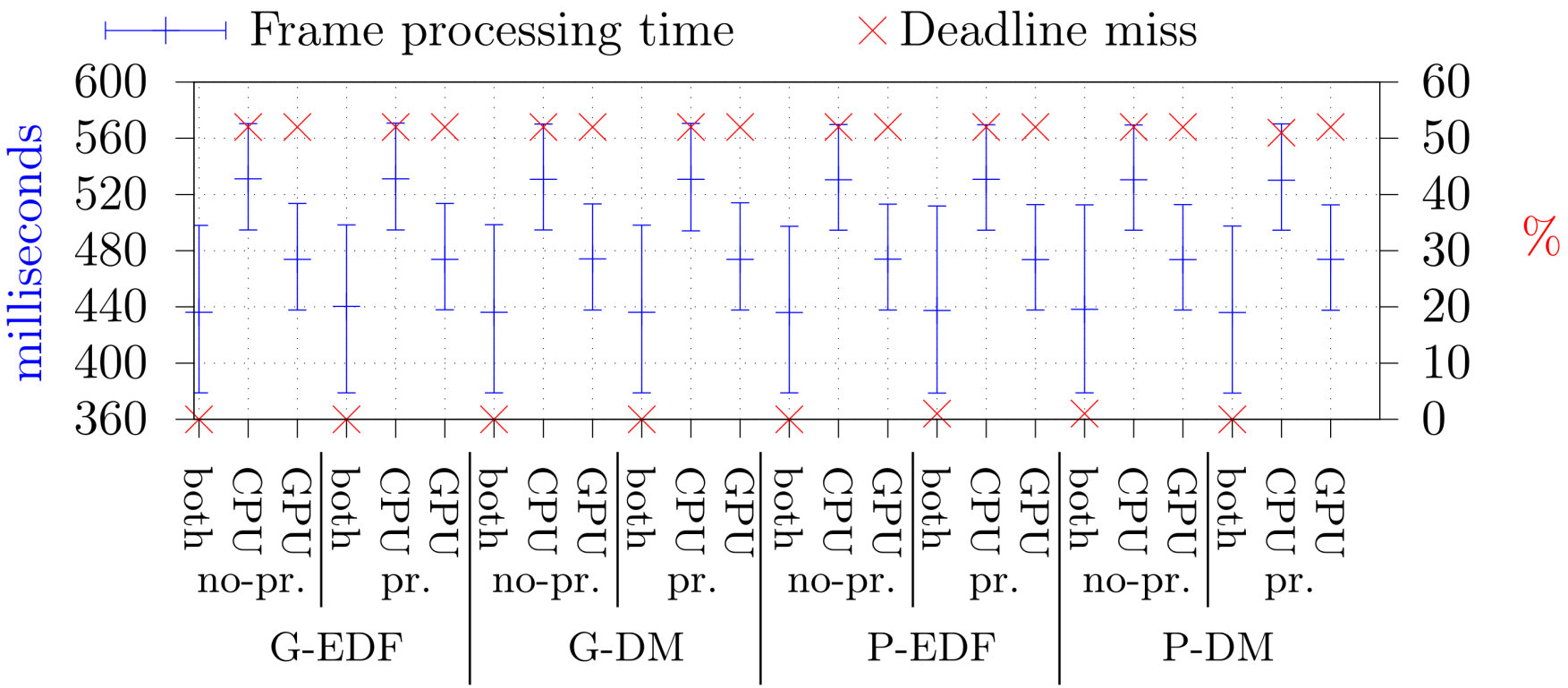
Drone tasks


FC msg handler
C: 170ms
T: 100Hz
Store
C: 8µs
Fetch new frame
C: 44µs
Extract EXIF
C: 168µs
Augment EXIF
C: 57µs
Detect objects
GPU-C: 108ms
CPU-C: 224ms
Highlight objects
GPU-C: 170ms
CPU-C: 242ms
Estimate speed
GPU-C: 130ms
CPU-C: 230ms
Create packet
C: 10µs
Encode
Plain-C: 3ms
AES-C: 100ms
Send
C: 10µs

Scheduling exploration

Experimental setup

- Handcrafted mission
- About 1000 processed frames



Conclusion & Future work

Summary

- Yasmin: a middleware to deploy schedule on behalf of the kernel
- Implemented as a library to link with the end-user application
- Better overhead than SOA
- Small overhead & latency

https://bitbucket.org/uva-sne/coordinationruntime

Conclusion & Future work

Summary

- Yasmin: a middleware to deploy schedule on behalf of the kernel
- Implemented as a library to link with the end-user application
- Better overhead than SOA
- Small overhead & latency

Future work

- Improve aperiodic task management
- Support for asynchronous accesses to accelerators
- Allow multi-rate task graphs

https://bitbucket.org/uva-sne/coordinationruntime