

# Energy aware memory allocation in real-time systems

Loïc Thomas

LAAS CNRS

*[l\\_thomas@insa-toulouse.fr](mailto:l_thomas@insa-toulouse.fr)*

October 20, 2023

# Goal of the study

Find a new solution for energy-aware memory allocation in real-time systems on low power microcontroller.

- ① Get energy consumption data on multiple benchmark with different memory configurations (CCM-SRAM, FLASH, SRAM) on cortex M4 microcontrollers.
- ② Measure the impact of each memory allocation.
- ③ Create a model to optimize the energy consumption.
- ④ Find scheduling algorithms which can work for this case.

# Outline of talk

- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# Table of Contents

- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# Context

## Previous work



Zhishen Zhang, Yuwen Shen, Binqi Sun, Tomasz Kloda, and Marco Caccamo

Memory allocation for low-power real-time embedded microcontroller. *IEEE ETFA 2022*

- Tasks' memory allocation can reduce runtime in real-time context.

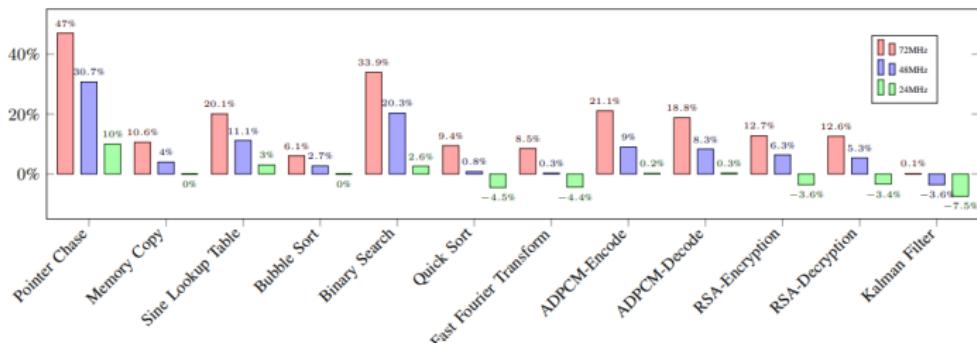
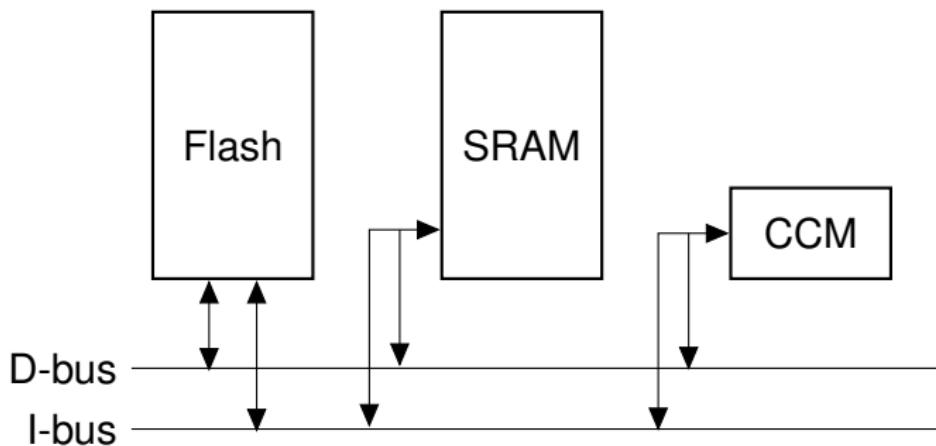


Figure: Results of the previous work (Runtime decrease when code run from CCM-SRAM)

# CCM SRAM



The core coupled memory is a special area of memory. It is connected to both instruction and data buses. It allows a no wait state access to memory.

# Wait states

- FLASH memory can be clocked up to 24MHz.
- When the CPU is faster than memory, it has to wait the memory to be ready.  
⇒ FLASH execution is interrupted by wait states.
- The CCM-SRAM can transfer instructions at CPU clock rate.  
⇒ CCM-SRAM execution time is proportionnal to frequency.

# Table of Contents

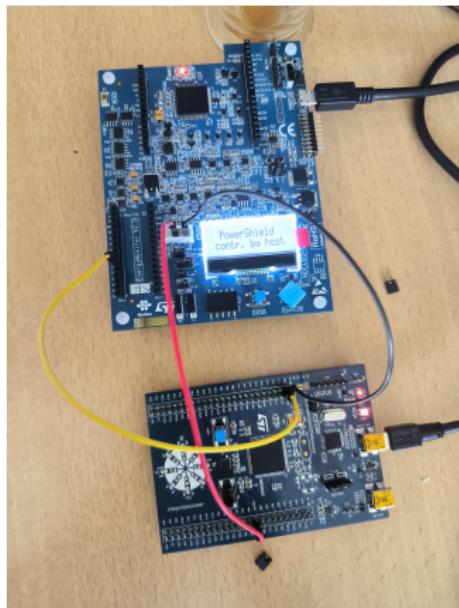
- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# NUCLEO LPM01A



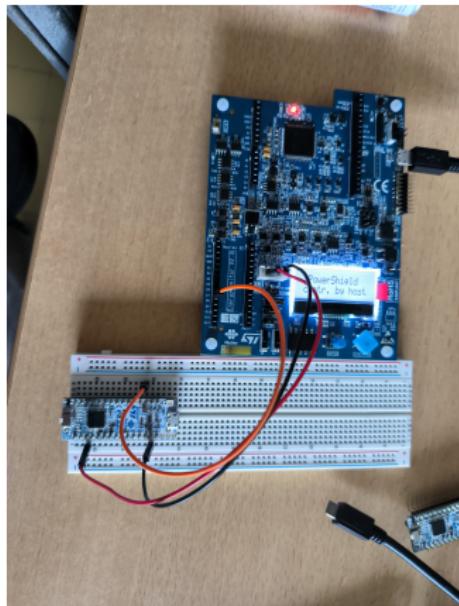
- Consumption averaging (from 1 nA up to 200 mA)
- Measure acquisition at 3.2 Msamples/s max
- Transfer measures with USB
- Results presented as 2 columns .csv file (time/intensity)

# STM32F303



- Frequency up to 72 MHz
- 8 kB CCM-SRAM
- 16 kB SRAM
- 512 kB FLASH
- FLASH speed up to 24 MHz
- No dynamic voltage frequency scaling

# STM32G43KB



- Frequency up to 170 MHz
- 10 kB CCM-SRAM
- 22 kB SRAM
- 128 kB FLASH
- Instruction and data cache for flash  
(32 and 8 lines of 4 x 64 bits)
- Pre-Fetch feature
- Dynamic voltage scaling (3 modes)
- Two different SRAM

# STM32F possibilities

Contrary to the STM32G, the STM32F do not have Dynamic Voltage Frequency Scaling (DVFS).

The options are more limited, the results that we will have are the consequences of the memory allocation.

## Memory configurations

- Instructions in FLASH, SRAM and CCM
- Input data in SRAM
- Read only data in FLASH, SRAM and CCM

# Table of Contents

- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# First measures for current consumption

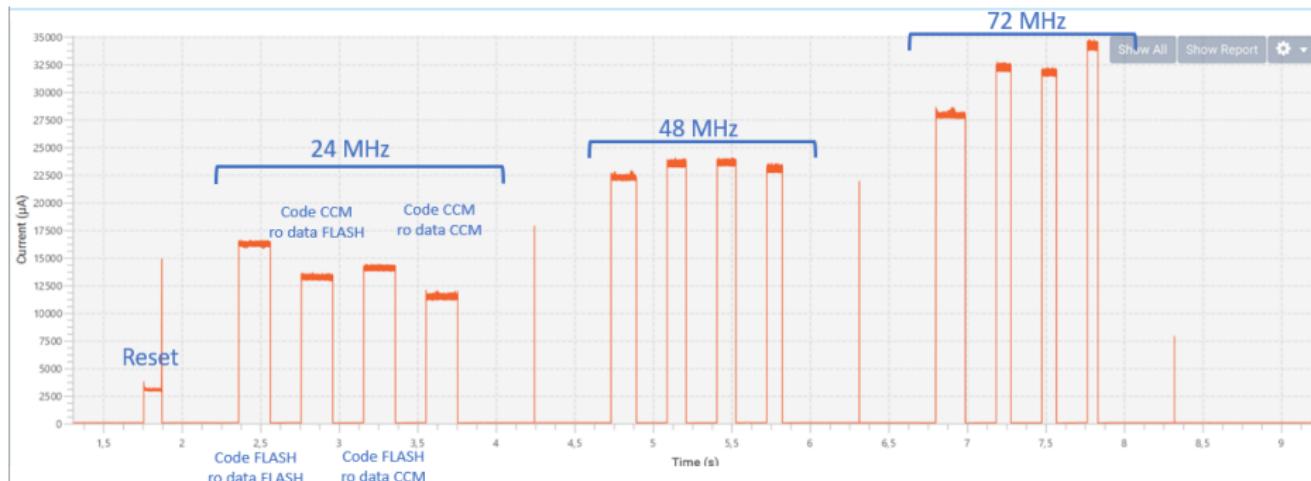


Figure: Intensity consumption graph for different pointer chase executions

# Final measures for current consumption

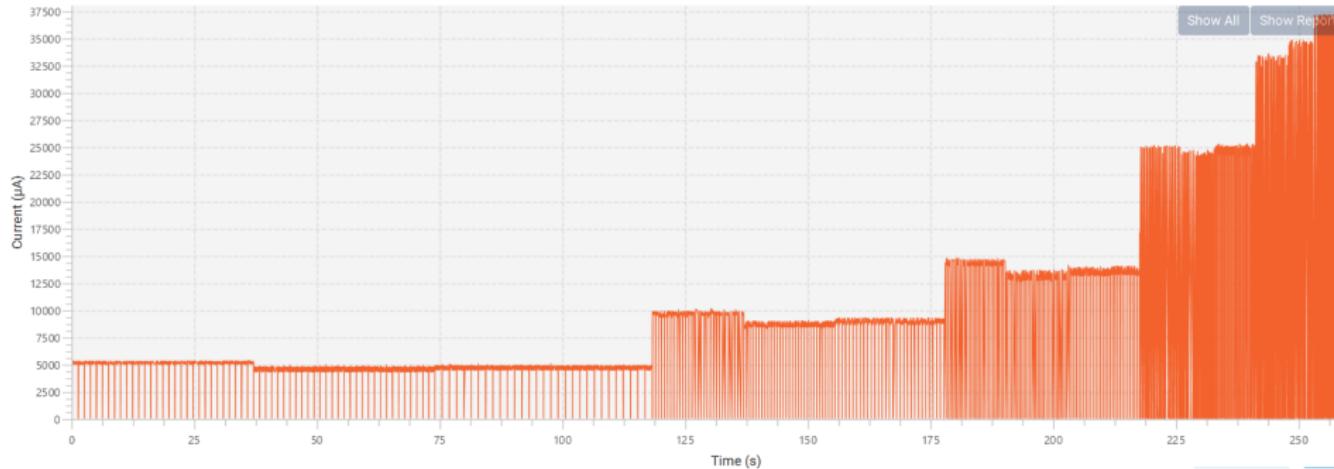


Figure: Intensity consumption graph with 30 execution per memory configuration

# Table of Contents

- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# Absolute energy when code is in FLASH and CCM

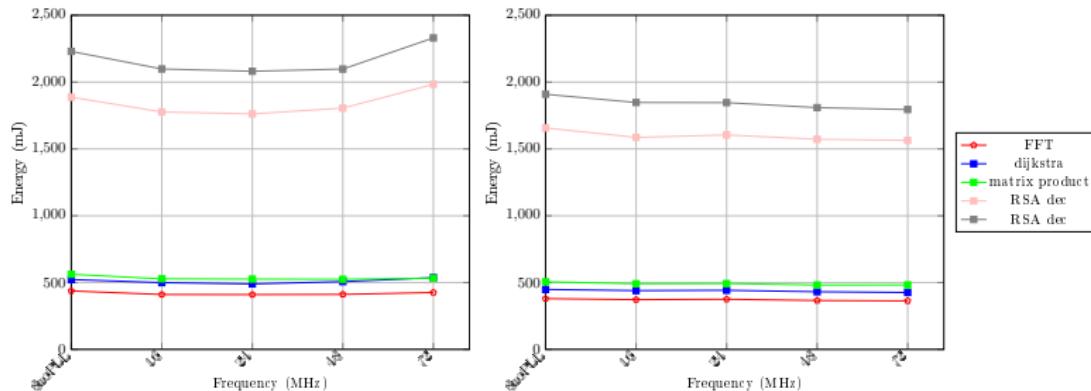


Figure: Energy (mJ) when instructions are in FLASH and CCM (STM32F)

- *Instructions in FLASH : frequency ↑ ⇒ energy ↑*
- *Instructions in CCM : frequency ↑ ⇒ energy →*

To spare energy, tasks should run at low frequency in FLASH and high frequency in CCM.

# Table of Contents

- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# STM32G features

## Different execution modes

- Range 1 mode (normal) : normal execution from 8 to 150 MHz.
- Range 1 boost mode : frequency up to 170 MHz, less wait states but higher voltage.
- Range 2 low power mode : frequency between 8 and 32 MHz, better consumption between these frequencies.

## FLASH instruction cache

Allows zero wait states when instructions are in FLASH → same results as the CCM (runtime).

DVFS

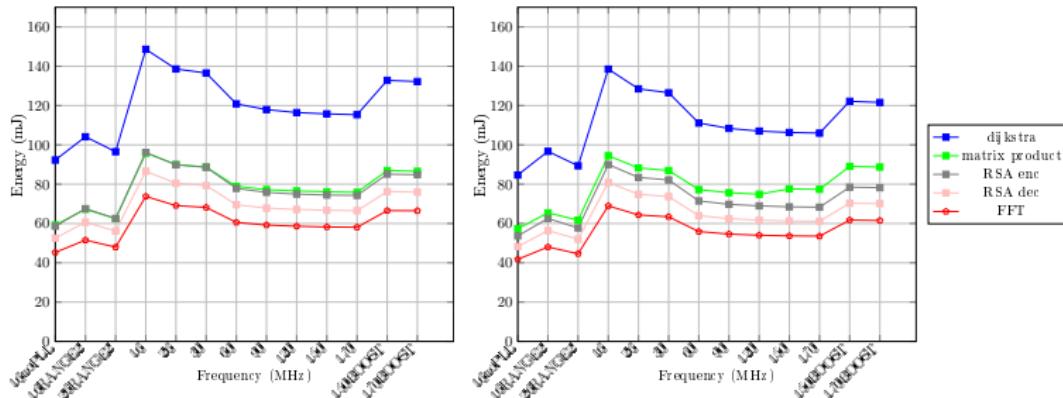


Figure: Energy (mJ) when instructions are in FLASH and CCM (STM32G)

- Range 2 mode can significantly lower the energy consumption, but it can be used only in low frequencies.
  - BOOST mode consume more energy to go faster in high frequencies. If the time gain is high enough it can be a good alternative to let other task run slower.

# Cache impact

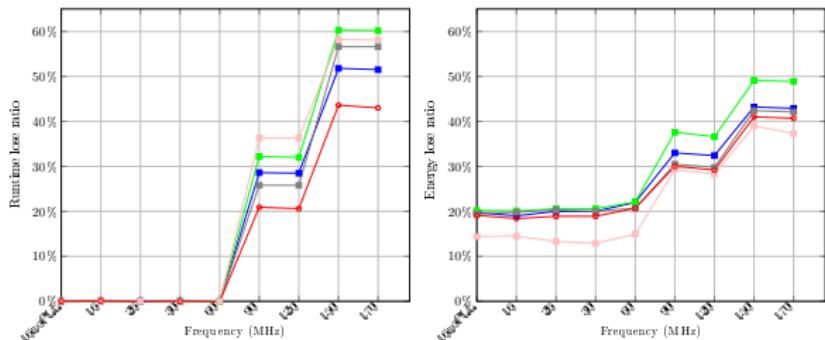
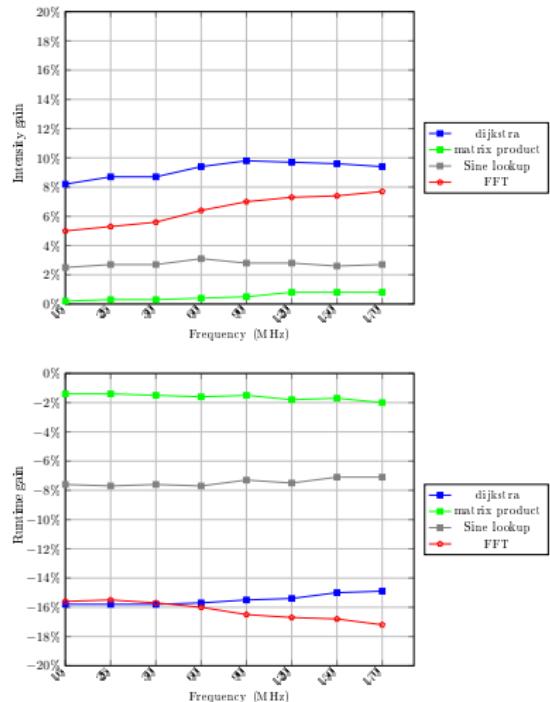


Figure: Runtime and Energy loss when cache is disabled

- Different wait states levels
- In high frequencies the wait states reduce the global intensity
- With no wait state, there are the same runtime performances

# SRAM 1 and 2



- The SRAM2 allows a better power consumption.
- This intensity gain is not enough to compensate the runtime loss.

It will be better to put data in SRAM1.

Figure: Intensity and Runtime gain when we move data from RAM1 to RAM2 (code FLASH)

# Pre-Fetch in STM32G

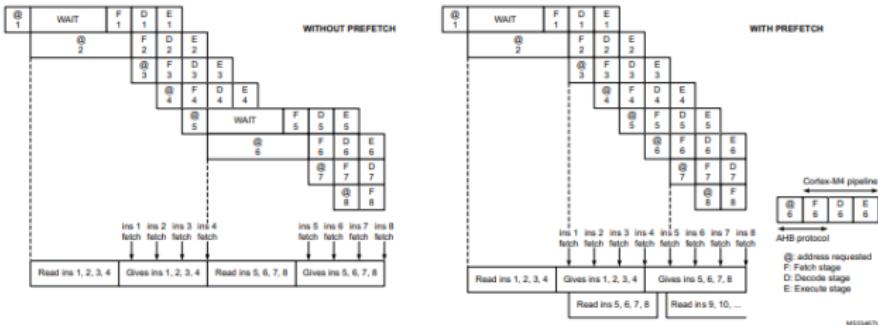


Figure: Sequential 16-bit instructions execution (64-bit read data width)

Pre-fetch on the instruction bus can be used to read the next sequential instruction line from the Flash memory while the current instruction line is being requested by the CPU.

# Pre-Fetch impact when cache is disabled

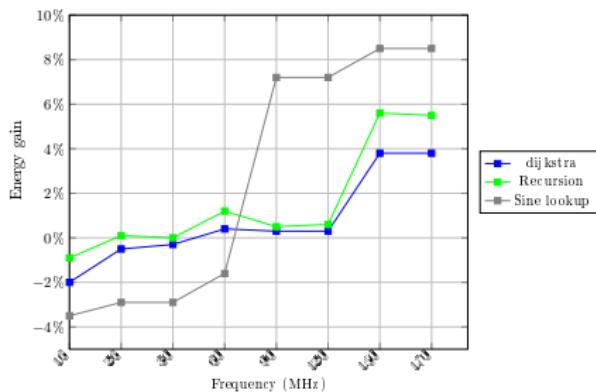
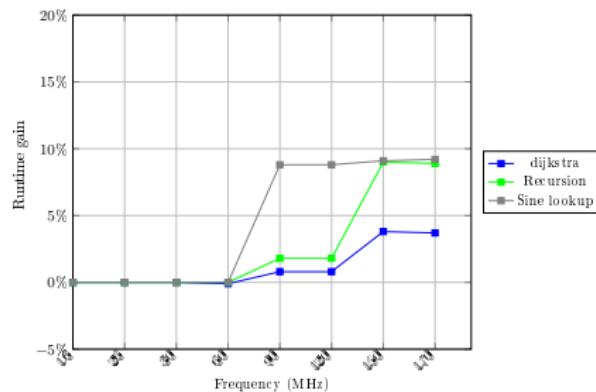


Figure: Runtime and energy gains when pre-fetch is enabled

# Pre-Fetch impact when cache is enabled

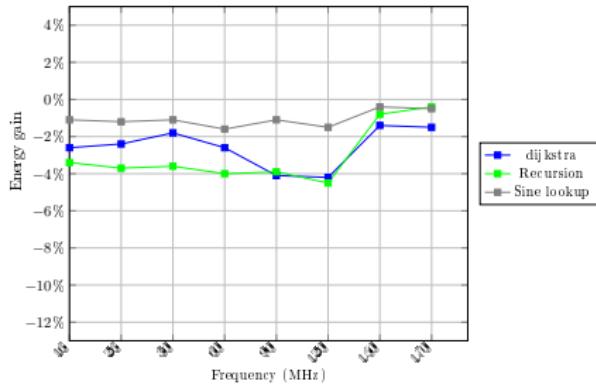
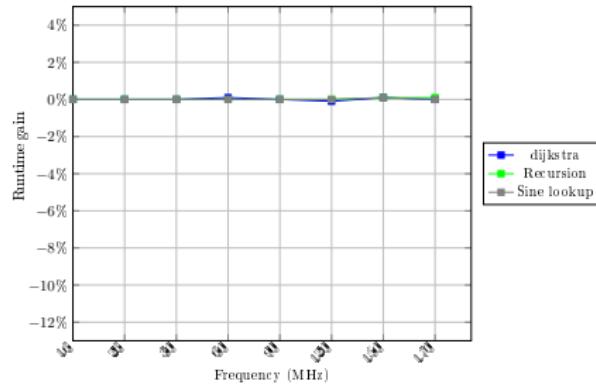


Figure: Runtime and energy loss when pre-fetch is enabled

# Table of Contents

- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# Observations conclusion

## Comparaison with FLASH

Moving instructions in CCM SRAM reduces runtime and energy consumption. It is almost always better to run code from CCM.

## Energy over frequency

- Energy is constant when code runs from CCM at every frequency. There are only advantages to run from CCM at maximum frequency.
- When instructions are in FLASH we can gain energy if the frequency is lower

# Offline algorithm

Choose the best tasks to move in CCM and which one we lower the frequency

$$\min \sum_{i=1}^n \left( \sum_{c \in \mathcal{C}} \frac{E_i^c}{T_i} \cdot x_i^c \right)$$

## Constraint 1

$$\sum_{c \in \mathcal{C}} x_i^c = 1 \quad \forall \tau_i \in \Gamma$$

- $\mathcal{C}$  Represent the set of all possible configuration (memory and frequency)
- Only one configuration can be selected per task.

## Constraint 2

$$\sum_{i=1}^n \sum_{c \in \mathcal{C}} U_i^c \cdot x_i^c \leq 1$$

- The taskset has to remain schedulable over EDF.
- On standard configuration the utilization is 1  $\Rightarrow$  always a solution

# Model memory constraints

## Constraints 3, 4 and 5

$$\sum_{i=1}^n \left( \sum_{c \in C}^{p=P} (x_i^c \cdot m_i^P) + \sum_{c \in C}^{ro=R} (x_i^c \cdot m_i^{Ro}) \right) \leq M^C$$

$$\sum_{i=1}^n \left( \sum_{c \in C}^{p=F} (x_i^c \cdot m_i^P) + \sum_{c \in C}^{ro=F} (x_i^c \cdot m_i^{Ro}) \right) \leq M^F$$

$$\sum_{i=1}^n \left( \sum_{c \in C}^{d=R} (x_i^c \cdot m_i^D) + \sum_{c \in C}^{ro=R} (x_i^c \cdot m_i^{Ro}) \right) \leq M^R$$

- Constraint on CCM, FLASH and SRAM space
- The sums on c cover only the case which concern the good memory  
(ex :  $d = R$  means that input data is in SRAM)

# Energy results on STM32F

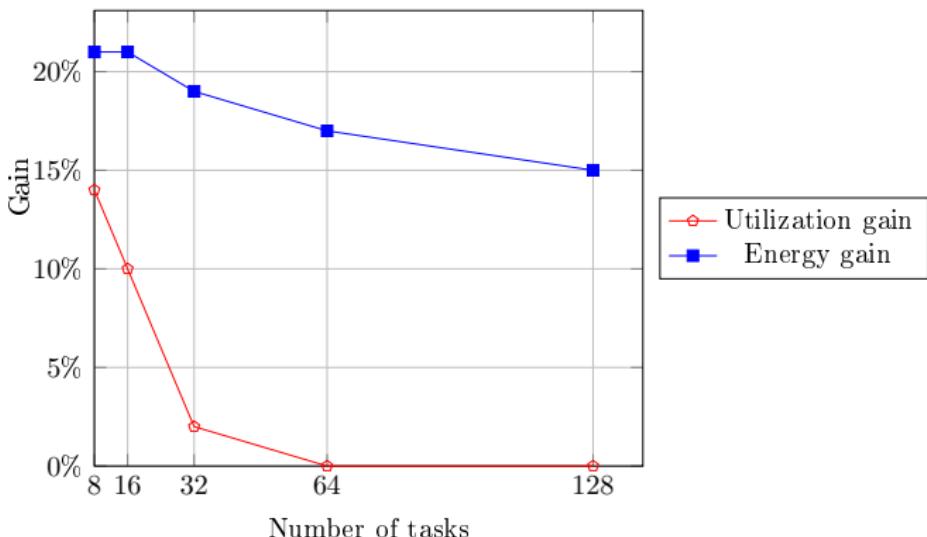


Figure: Utilization and energy gains with the model on STM32F

Comparing to the default configuration we can consume 20% less energy. With a low number of tasks we can even gain utilization.

# Frequency results on STM32F

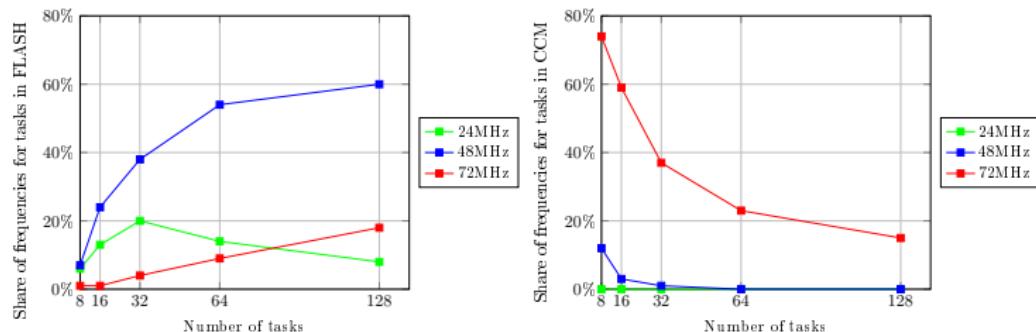
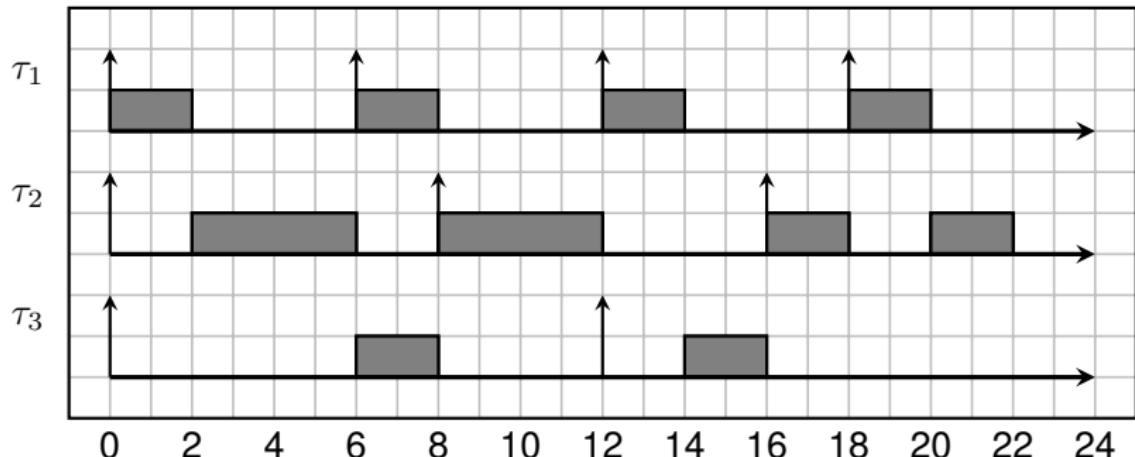


Figure: Frequency choices made by the model for tasks in FLASH and CCM

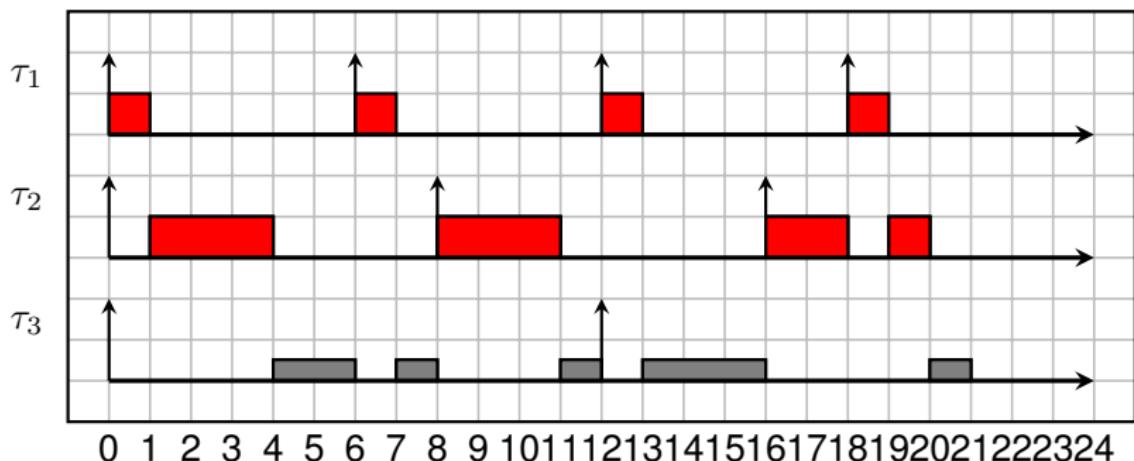
The proportion of tasks in CCM decrease over the number of tasks. With less tasks, the tasks in FLASH run at low or mid frequency. When the number of task begin to be high more tasks in FLASH are executed at 72MHz  
⇒ when the system is crowded ( $U \rightarrow 1$ ) it tends to the default config.

# Normal EDF example



We have 3 tasks in the standard configuration.  
Instructions are in FLASH and the CPU runs them at 72 MHz.

# Application of the previous algorithm



We move  $\tau_1$  and  $\tau_2$  in the CCM

$\Rightarrow$  runtime is reduced and consume less energy.

We use the utilization gain of the CCM to run  $\tau_3$  at 48 MHz

$\Rightarrow$  runtime is bigger, but, we consume less energy.

The utilization of the system is still 1 and we spare energy.

# Model results on STM32G

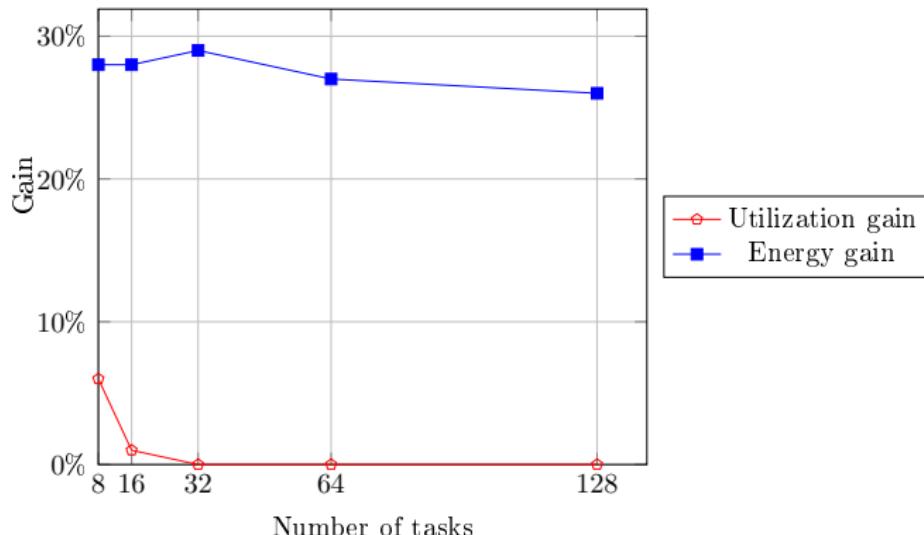


Figure: Utilization and energy gains with the model on STM32G

The default configuration use the cache so it is hard to gain utilization.

We ave a better energy gain with the STM32G, with the DVFS it may be easier to consume less.

# Frequency results on STM32G

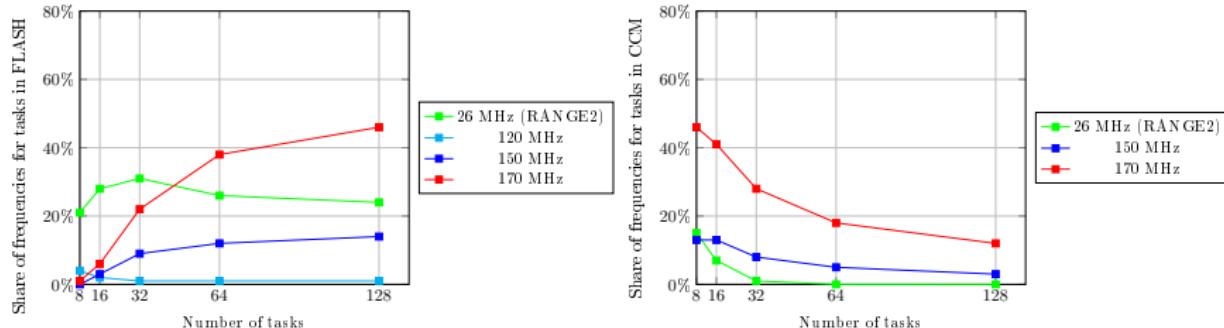


Figure: Frequency choices made by the model for tasks in FLASH and CCM for STM32G

RANGE2 mode is even used with the CCM. However the boost mode is not used at all (the runtime gain is too low compared to the waste of energy).

# Table of Contents

- ① Context and introduction
- ② Microcontrollers features
- ③ Current measurement
- ④ STM32F performances
- ⑤ STM32G performances
- ⑥ Optimization model
- ⑦ Future work

# Online algorithms

## Dynamic frequency and SLACK algorithm

If we can still complete the task with the worst case time execution before the deadline. Then we lower the frequency.

## Dynamic CCM SRAM allocation

- Divide the CCM into slots
- Before the execution we copy tasks' instructions in one CCM slot
- the worst case execution time is equal to the CCM execution time plus memory copy

⇒ Changes in the scheduling algorithm: mutex, blocking, non-preemptive tasks ...

# The End

Questions? Comments?