Energy aware memory allocation in real-time systems

CCM-SRAM allocation to reduce consumption

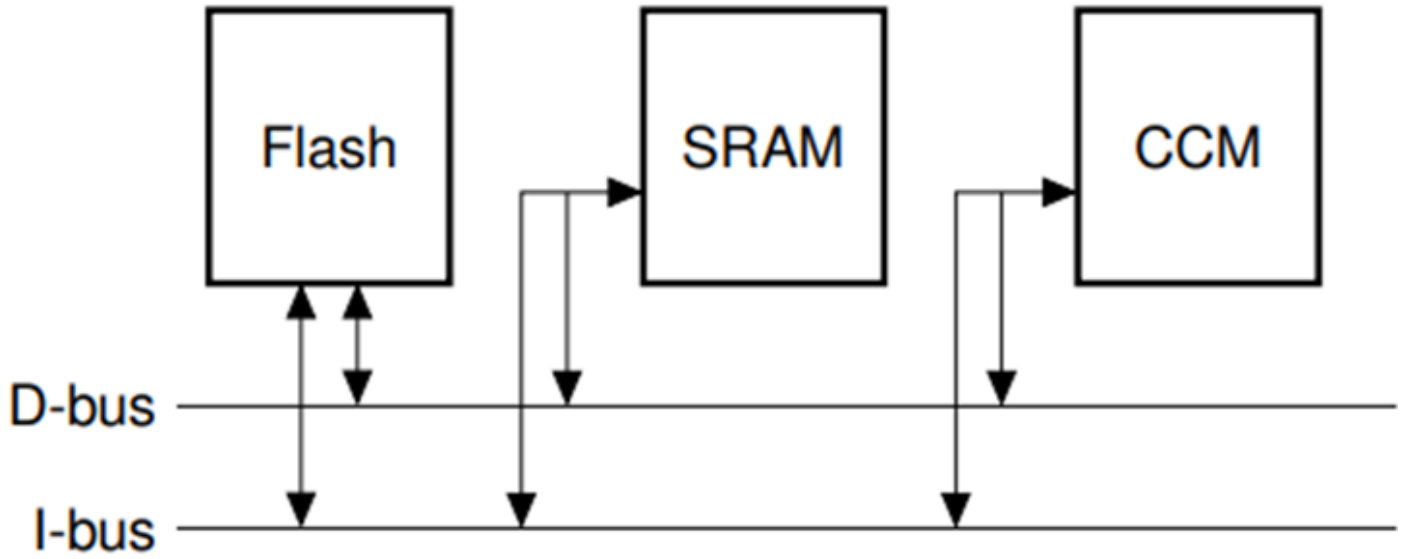
Loïc Thomas

LAAS CNRS

lthomas@laas.fr

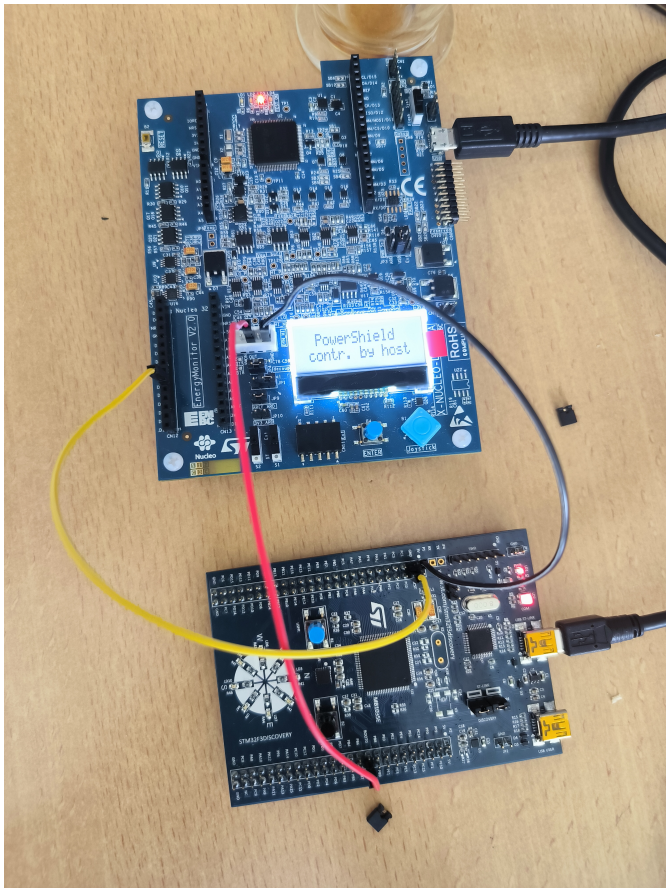
October 2, 2023

CCM SRAM



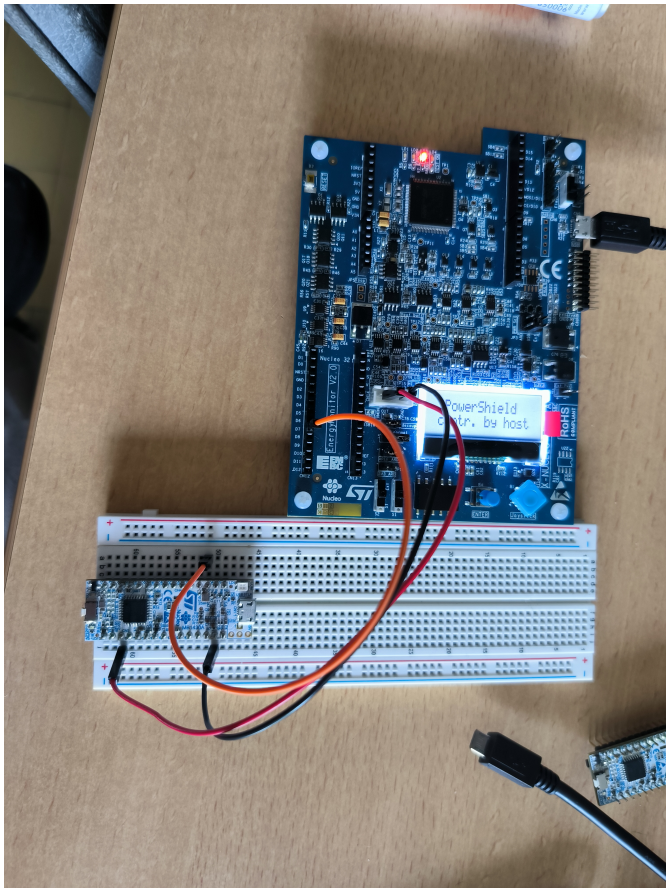
The core coupled memory is a special area of memory. It is connected to both instruction and data buses. It allows a no wait state access to memory.

STM32F303



- Frequency up to 72 MHz
- 8 kB CCM SRAM
- FLASH speed up to 24 MHz

STM32G43KB



- Frequency up to 170 MHz
- 10 kB CCM SRAM
- Instruction and data cache for flash
- Pre-Fetch feature
- Dynamic voltage scaling 3*modes*
- Two different SRAM

Goal of the study

Find a new solution for energy-aware memory allocation in real-time systems.

1. Get energy consumption data on multiple benchmark with different memory configurations.
2. Measure the impact of each memory allocation.
3. Create a model to optimize the energy consumption.
4. Find realtime algorithms which can work for this case.

First measures for current consumption

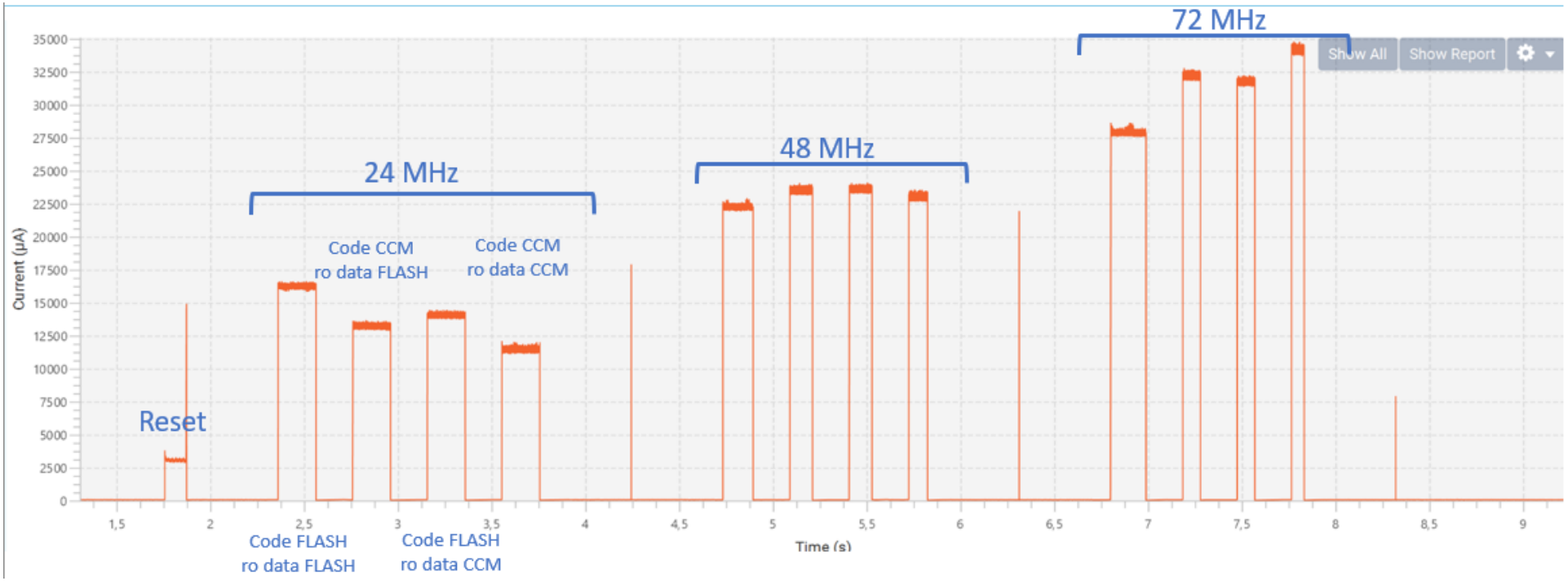


Figure: Intensity consumption graph for differents pointer chase executions

Final measures for current consumption

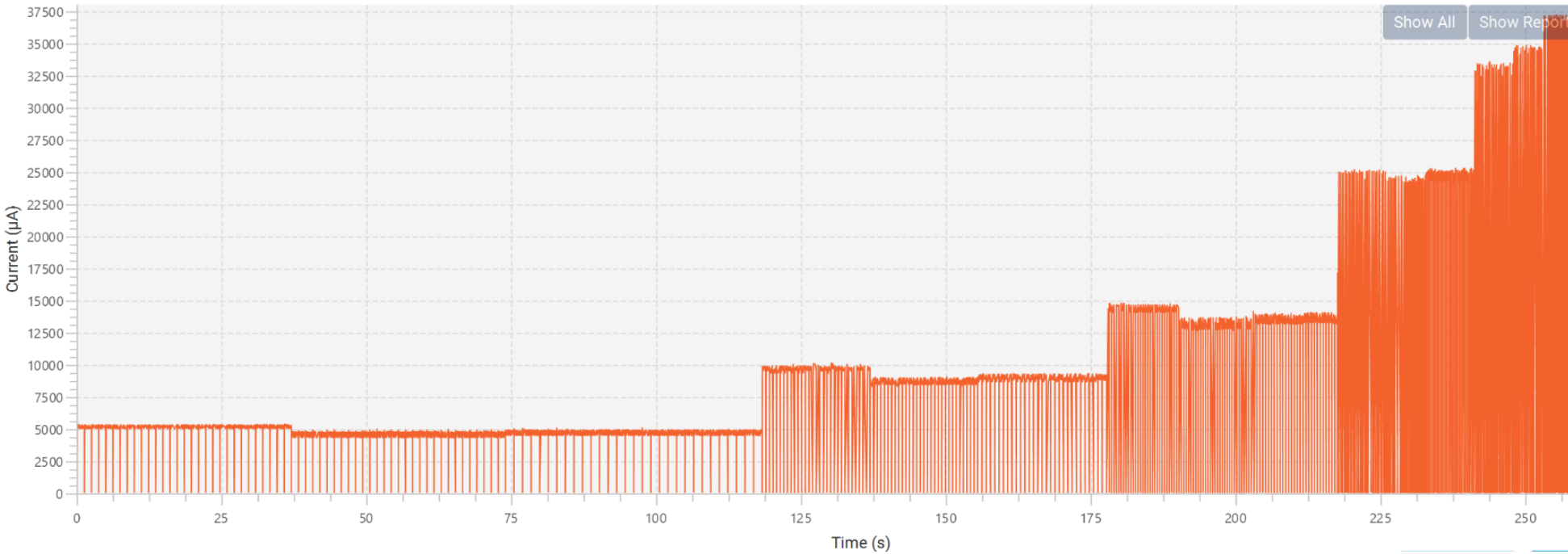
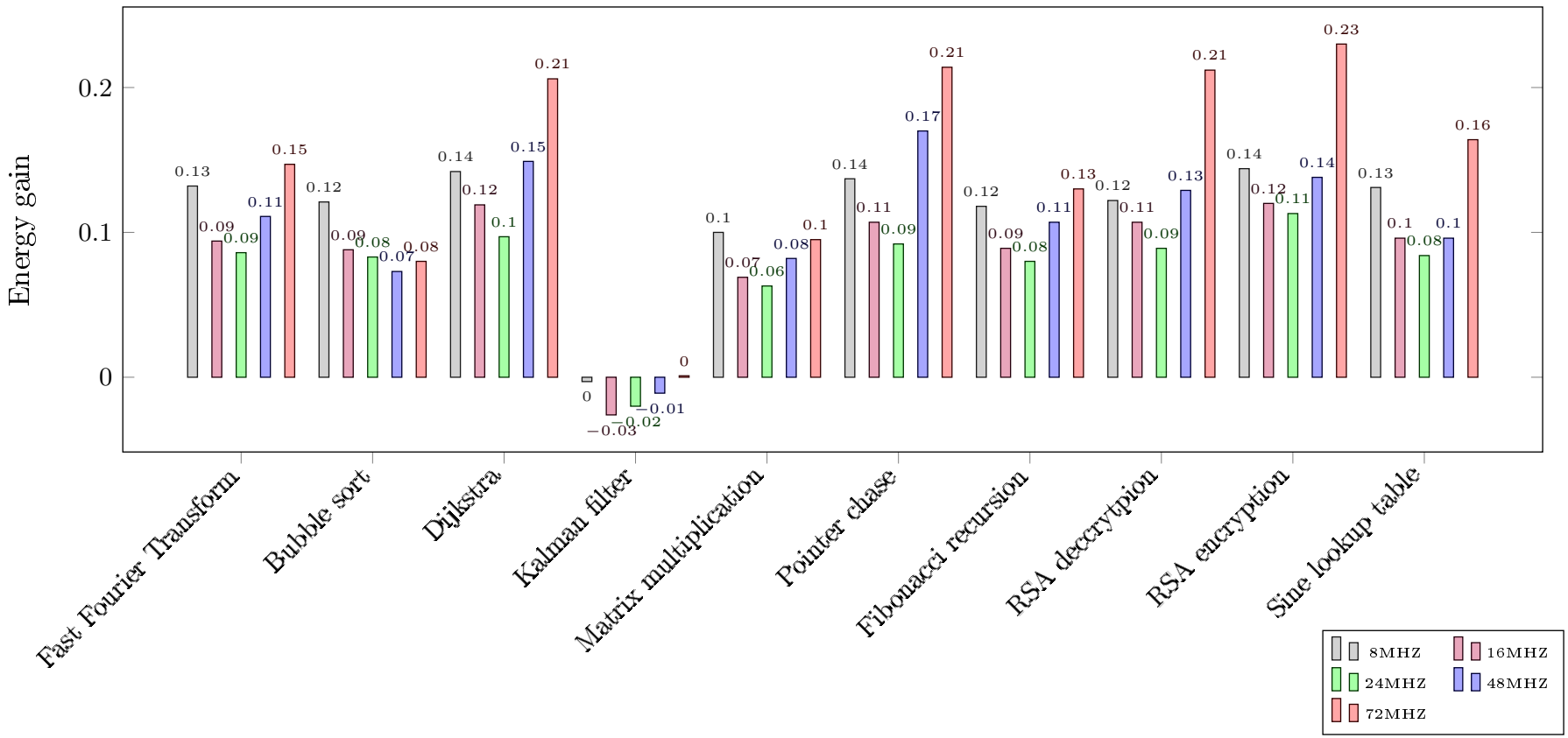


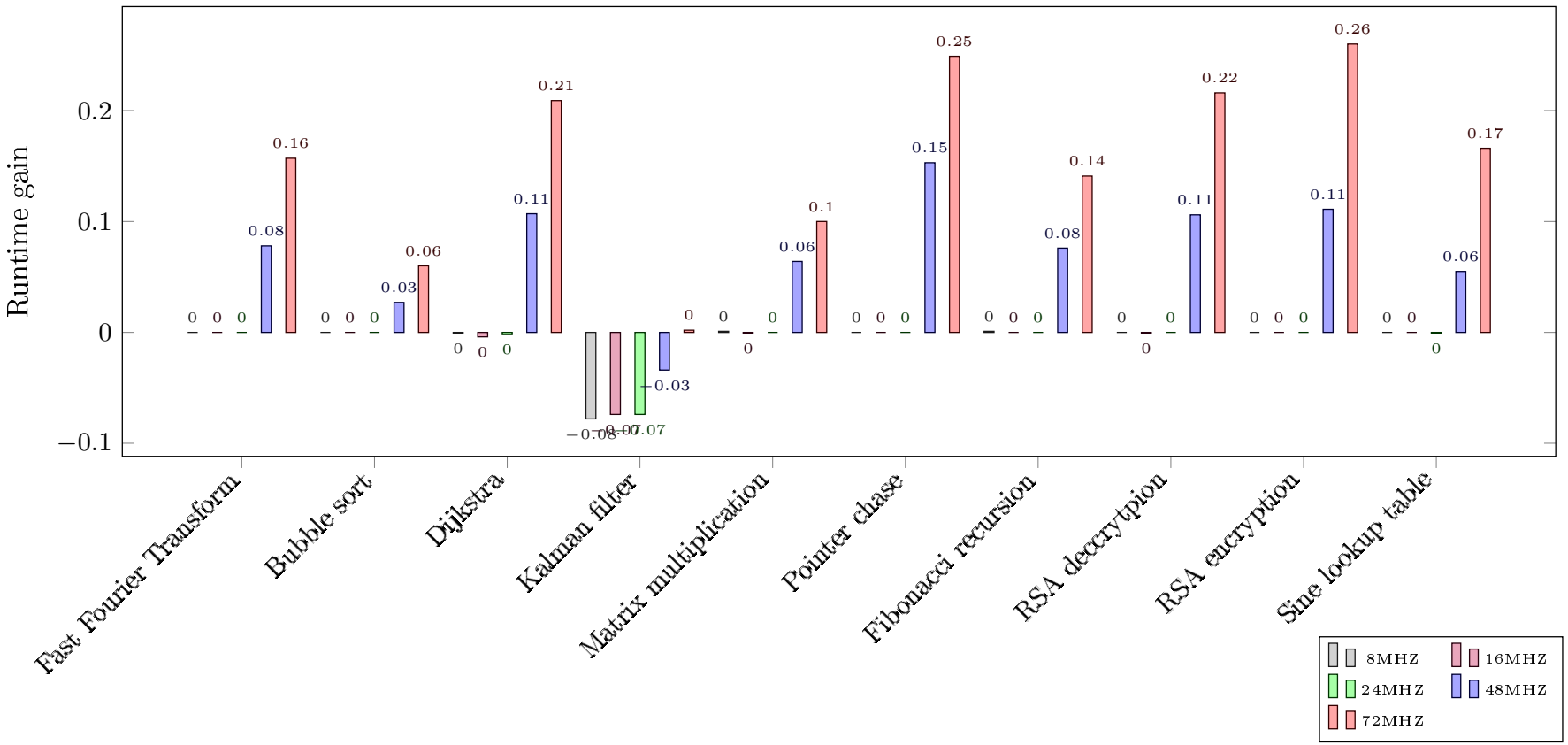
Figure: Intensity consumption graph with 30 execution per memory configuration

Energy decrease when instruction are moved into CCM



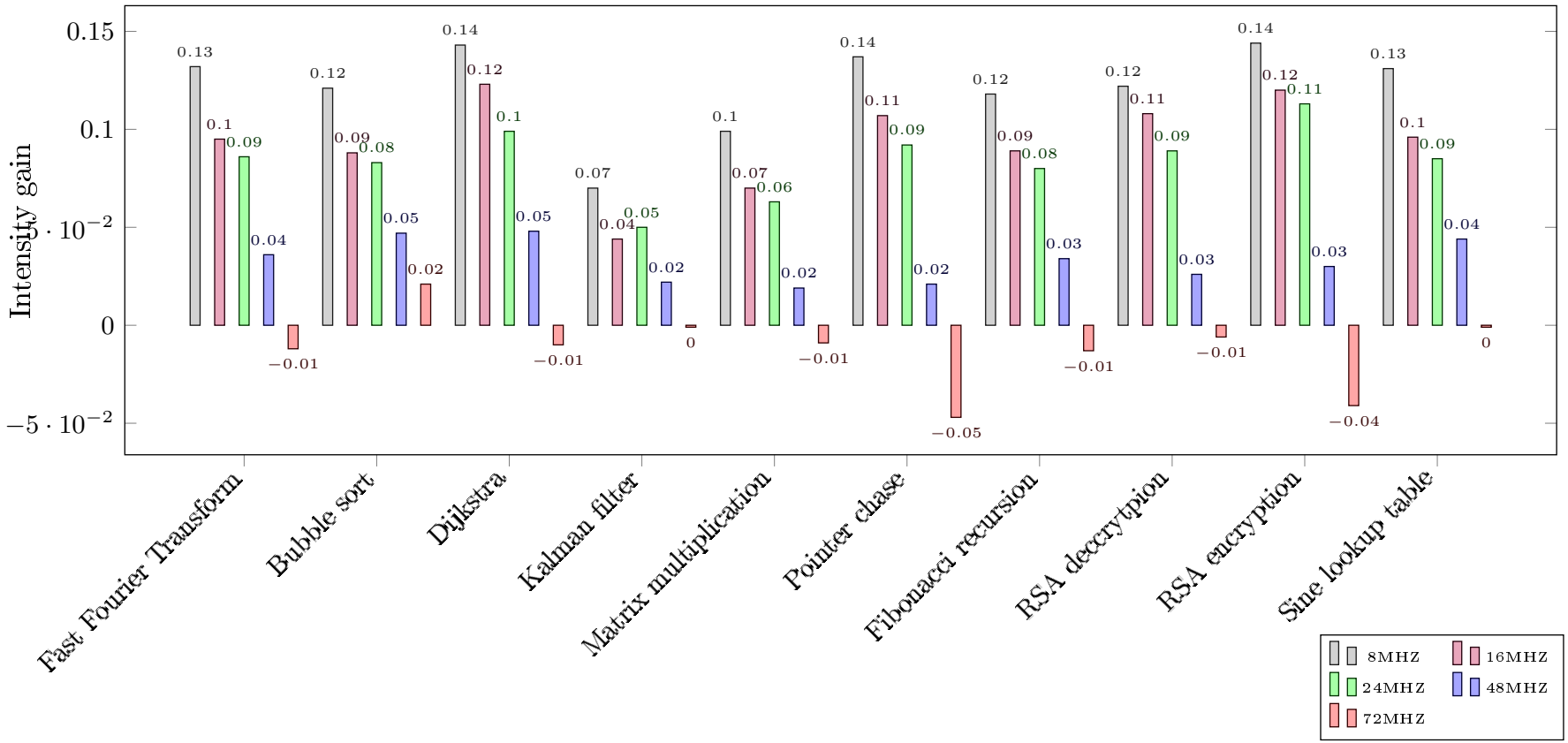
Moving instruction into CCM SRAM can overall improve energy consumption

Runtime decrease when instruction are moved into CCM



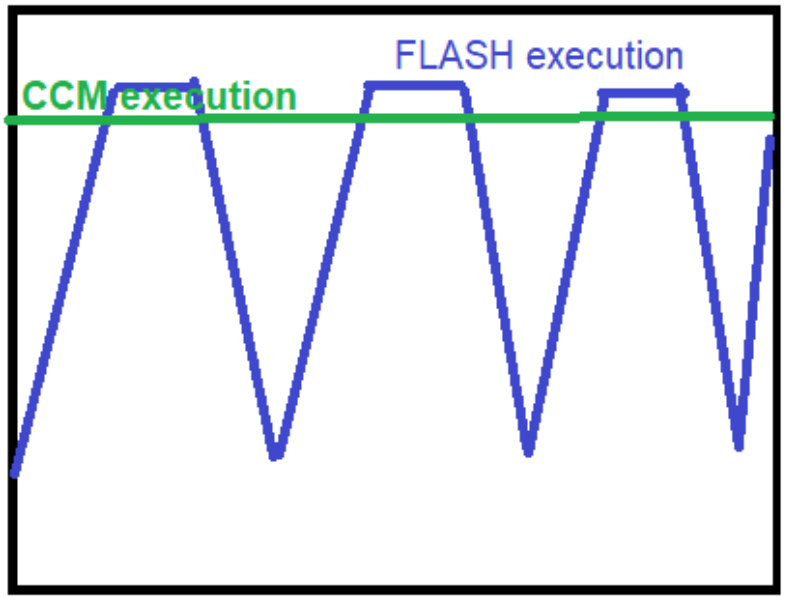
In high frequencies the runtime is reduced when code is moved into CCM SRAM.

Intensiy decrease when instruction are moved into CCM

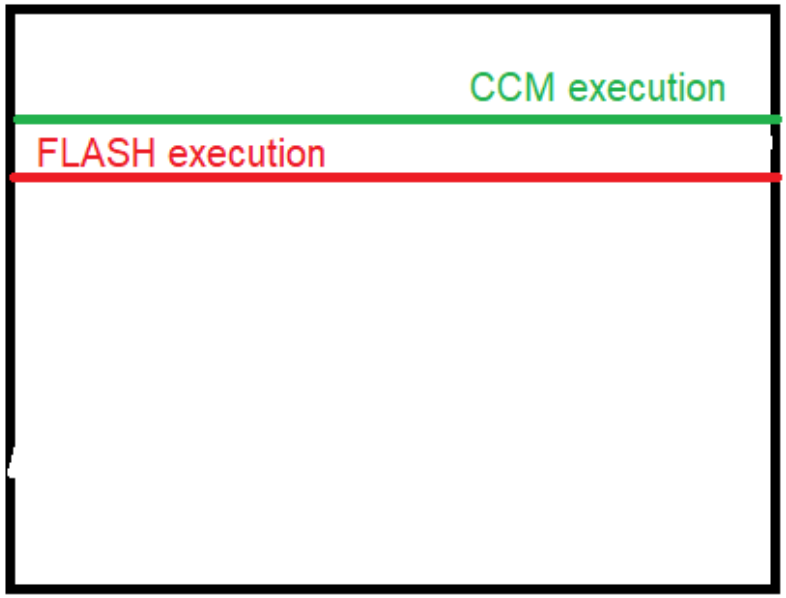


In high frequencies, intensity is lowered when the code is running from flash.

Wait states impact on intensity



What we could observe if we had a sampling frequency of 48 MHz



What we really see

Wait states lower the average intensity of an execution. This effect

Absolute energy when code is in FLASH and CCM

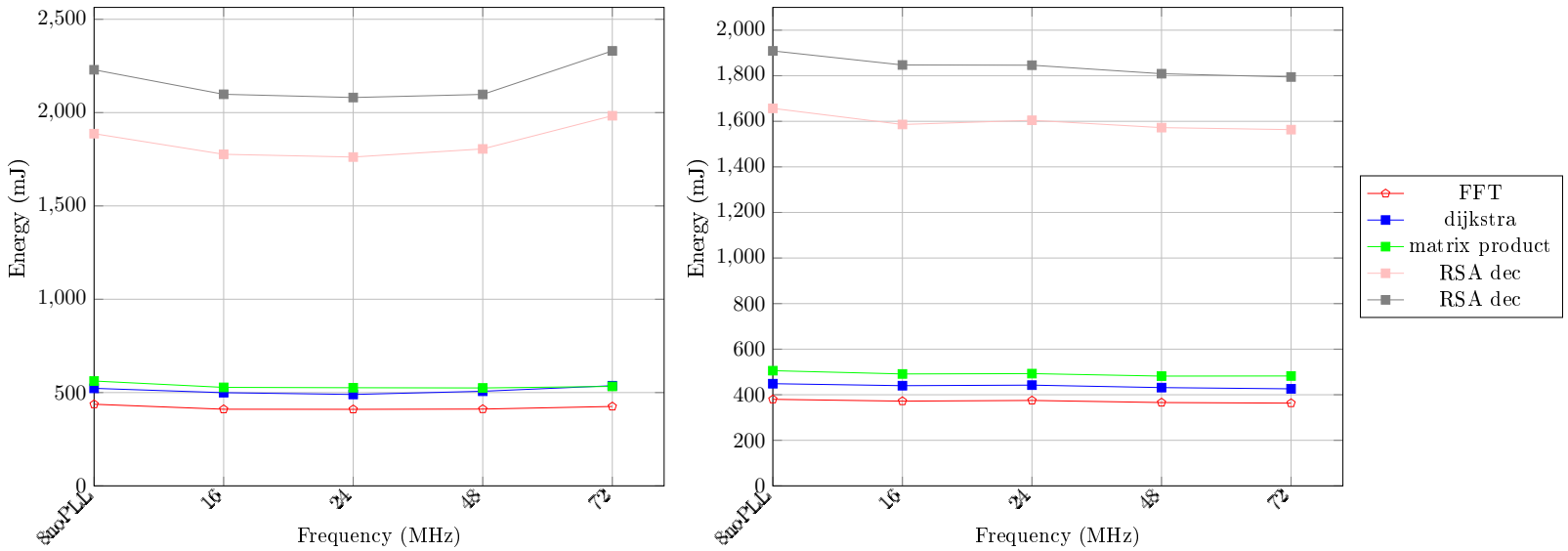


Figure: Energy (mJ) when instructions are in FLASH and CCM (STM32F)

- When frequency increase, the energy consumed when instructions are in FLASH memory can increase.
- When instructions are in the CCM, the energy in high frequencies is almost the same as in low frequencies. It can even be lower.

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

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 in these frequencies.

FLASH instruction cache

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

DVFS

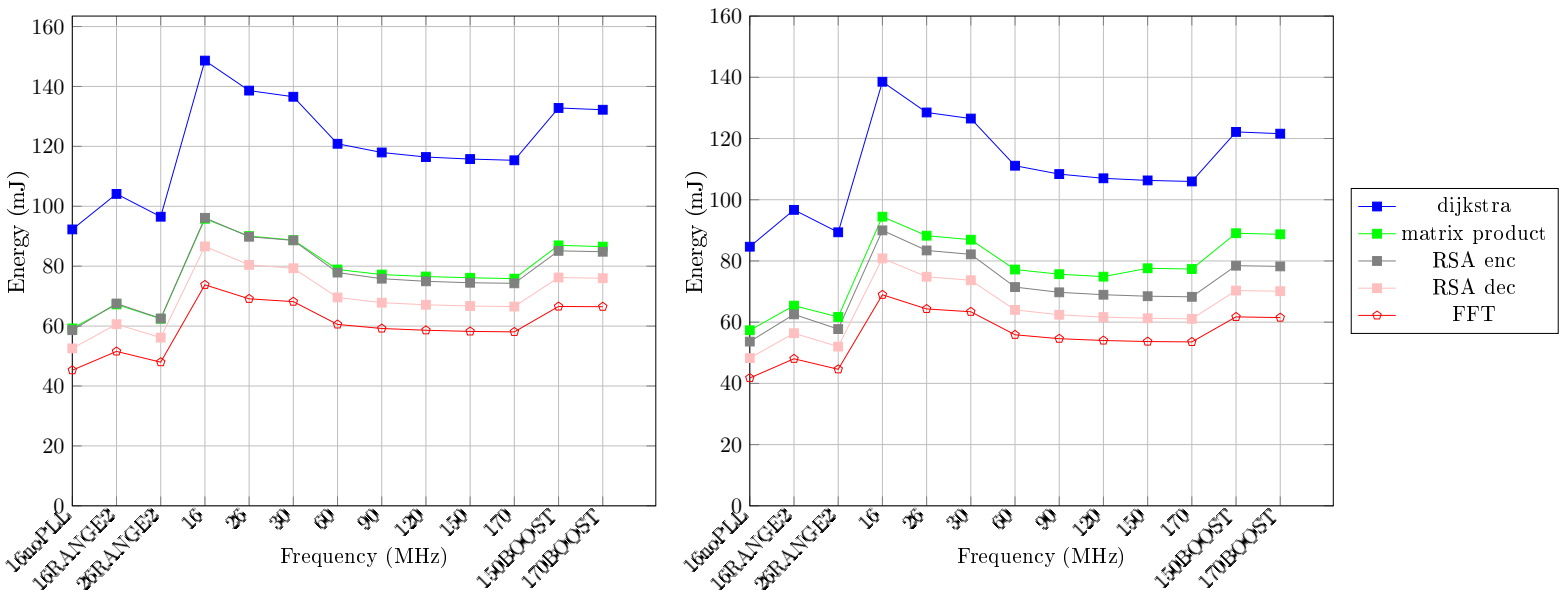


Figure: Energy (mJ) when instructions are in FLASH and CCM (STM32G)

- 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

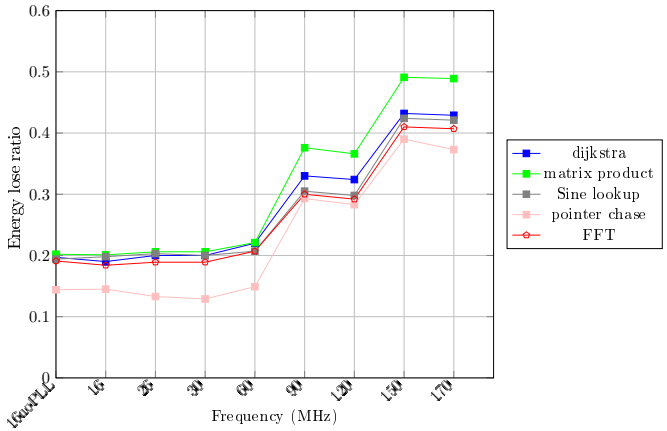


Figure: Energy loss when cache is disabled

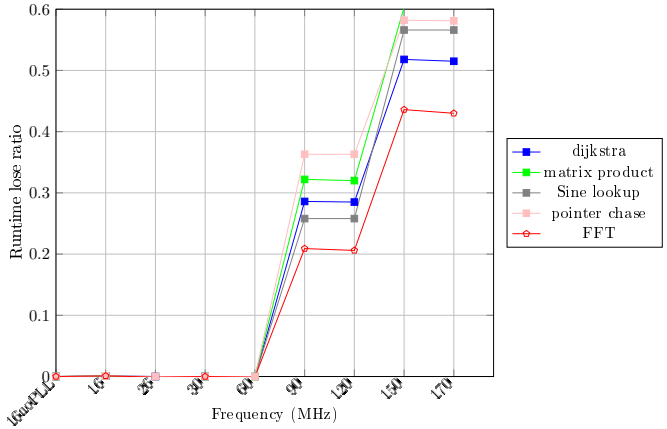


Figure: Runtime loss when cache is disabled

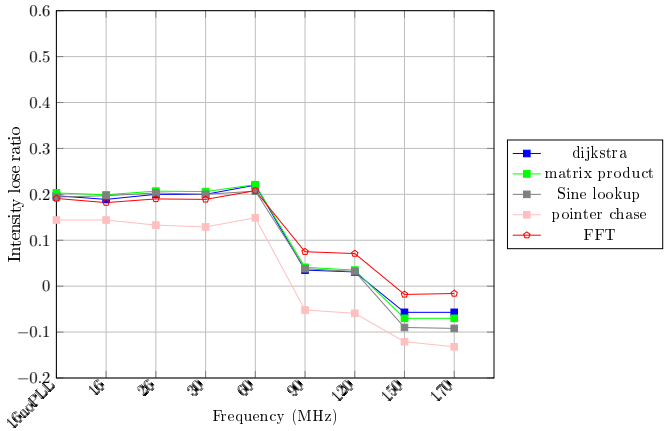


Figure: Intensity loss when cache is disabled

- We can see the different wait states levels
- In high frequencies the wait states can even reduce the global intensity
- With no wait states, there are the same runtime performances

SRAM 1 and 2

Pists

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

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

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

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

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

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

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

Energy results on STM32F

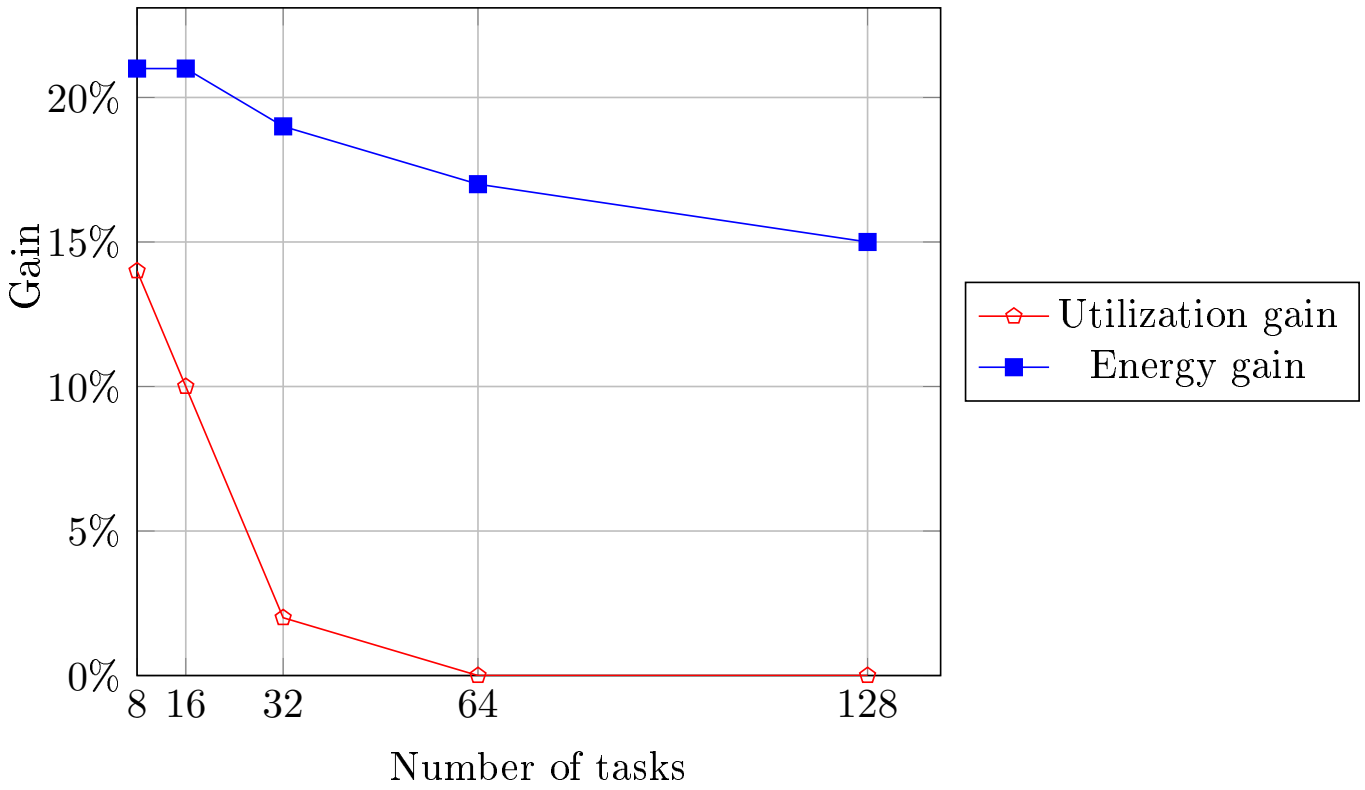
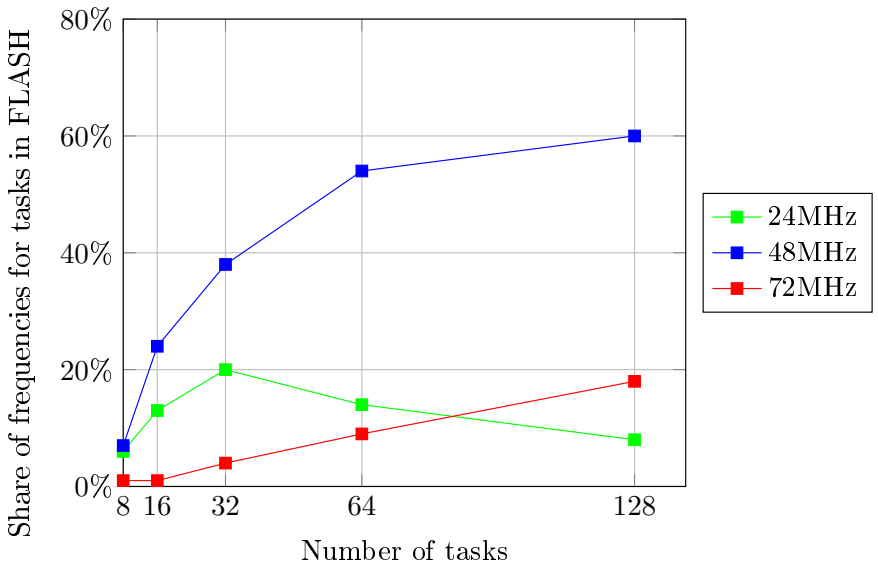
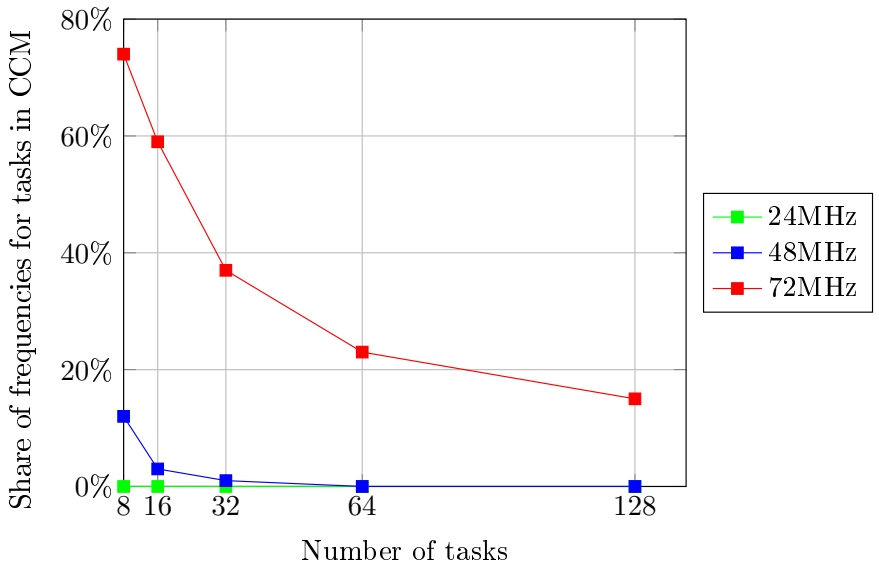


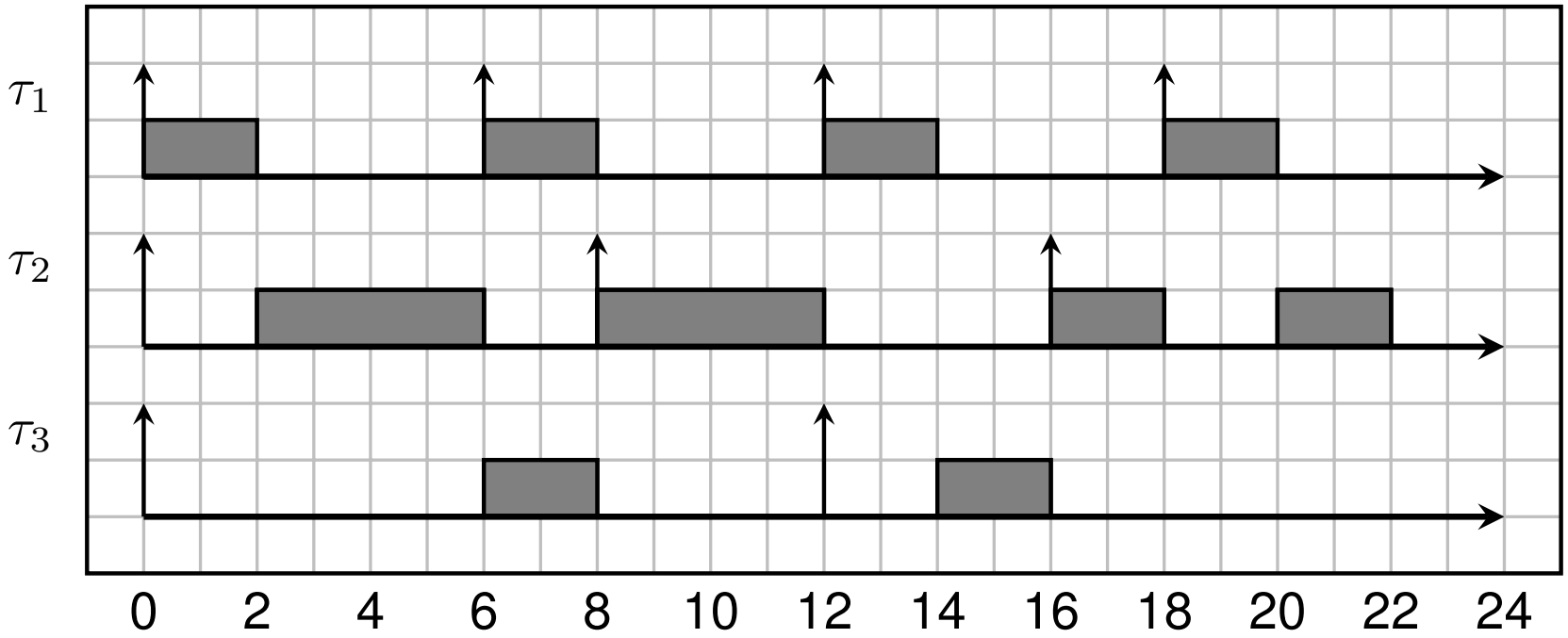
Figure: Utilization and energy gains with the model on STM32F

Frequency results on STM32F

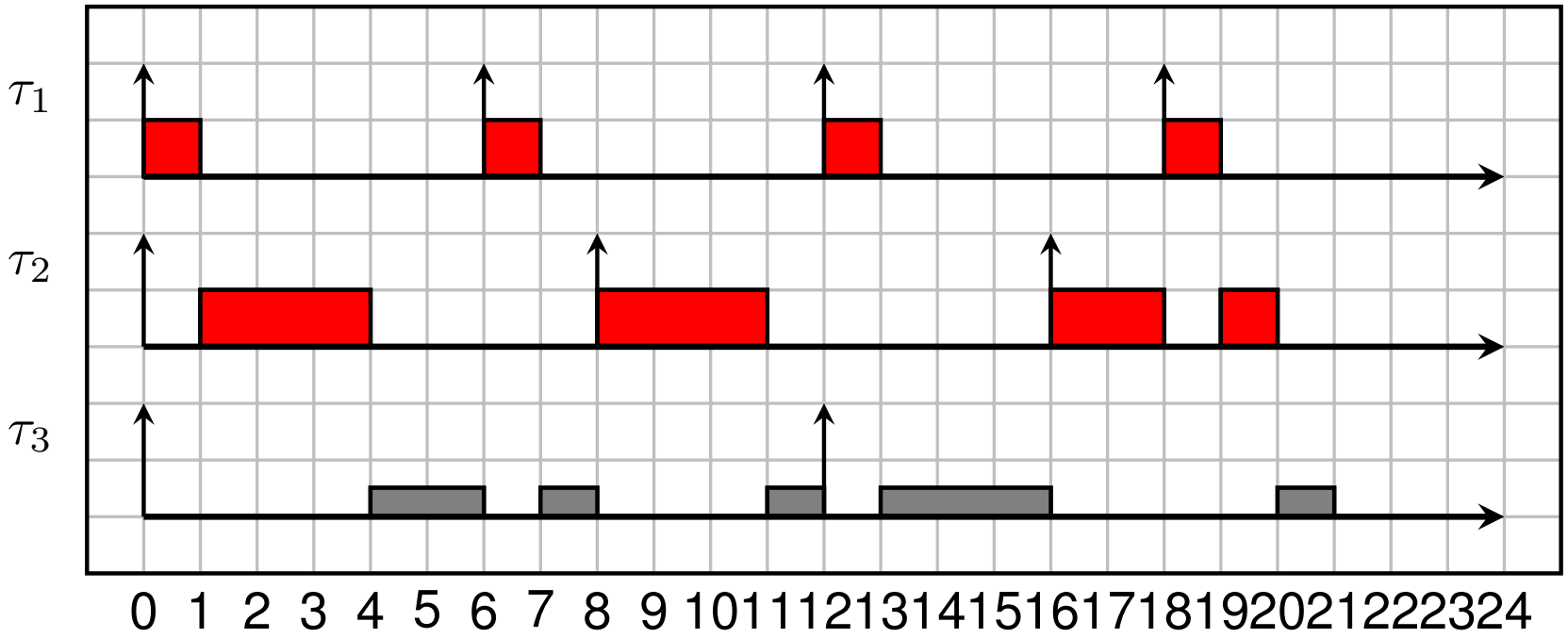




Normal EDF example



Application of the previous algorithm



Online algorithms

Dynamic frequency scaling for FLASH

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 executing copy task instructions in the CCM slot
- the worst case execution time is equal to the CCM execution time with the associated memcopy

$\Rightarrow$ Changes in the scheduling algorithm: mutex, blocking, non-preemptive tasks . . .

The End

Questions? Comments?