Energy-aware memory allocation in real-time systems

Loïc Thomas

LAAS CNRS - VERTICS

l_thomas@insa-toulouse.fr

November 1, 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 with different memory configurations on cortex M4 microcontrolers.

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 with different memory configurations on cortex M4 microcontrolers.
2. Measure the impact of each memory allocation.

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 with different memory configurations on cortex M4 microcontrolers.
2. Measure the impact of each memory allocation.
3. Create a model to optimize the energy consumption.

Goal of the study

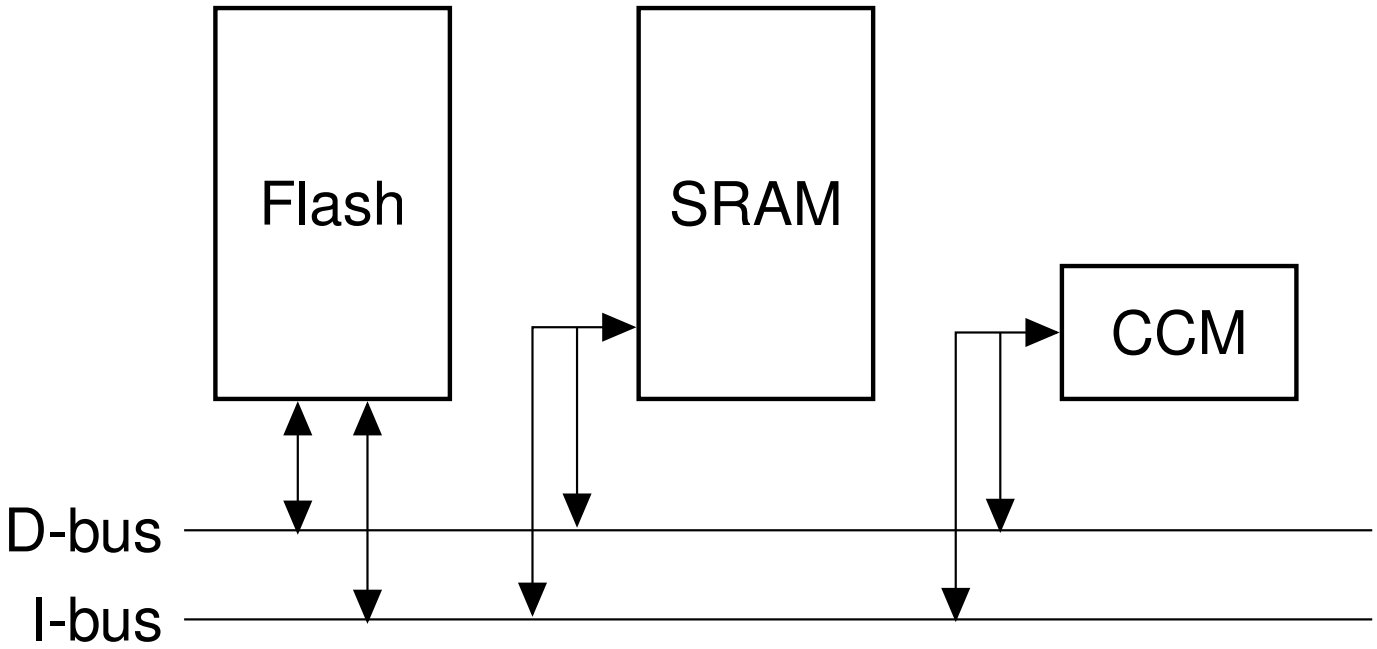
Find a new solution for energy-aware memory allocation in real-time systems on low power microcontroller.

1. Get energy consumption data with different memory configurations on cortex M4 microcontrolers.
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.

Table of Contents

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

CCM SRAM



The core coupled memory is a special area of memory. It is connected to both instruction and data buses. It allows a zero 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.
  $\Rightarrow$ FLASH execution is blocked by wait states.
- The CCM-SRAM can transfer instructions at CPU clock rate.
  $\Rightarrow$ CCM-SRAM execution time is proportionnal to frequency.

Previous work

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

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

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

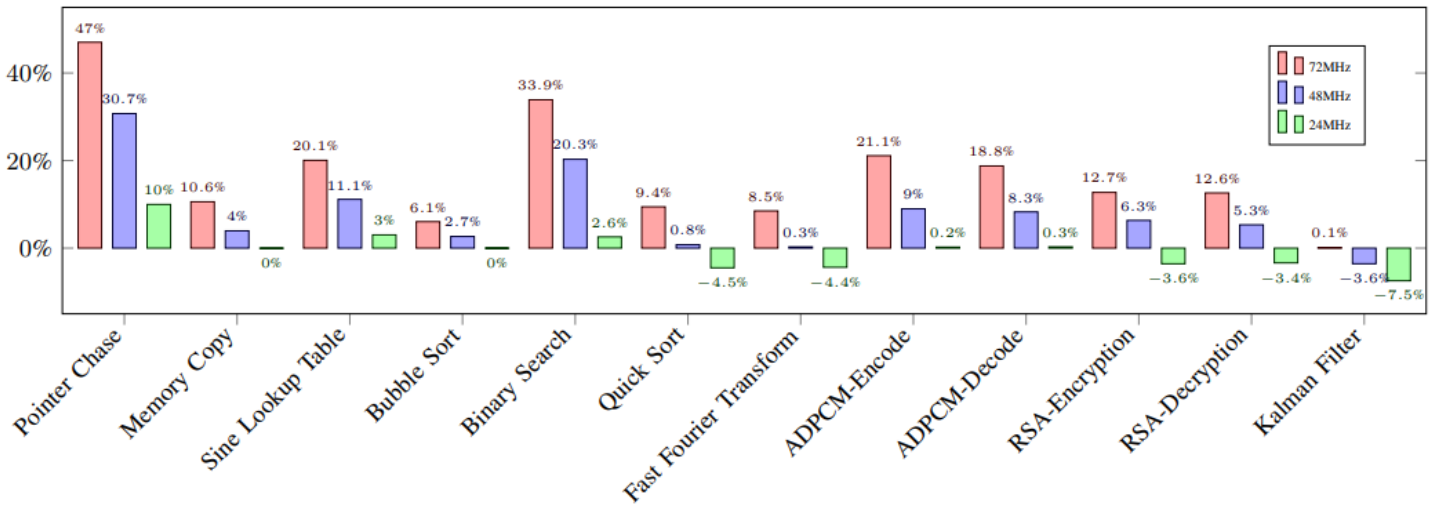


Figure: Results of the previous work (Runtime decreases when code run from CCM-SRAM)

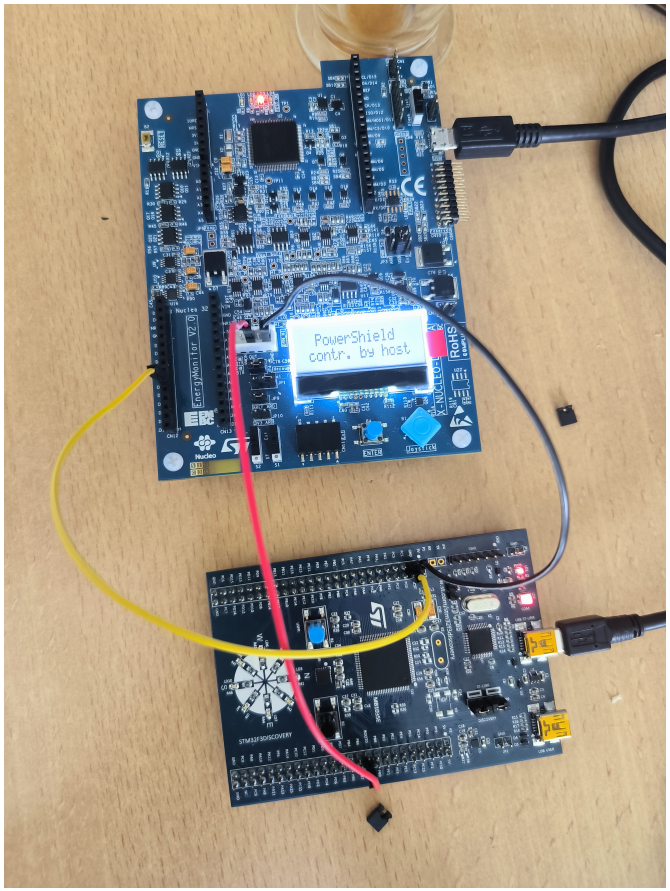
Table of Contents

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

NUCLEO LPM01A

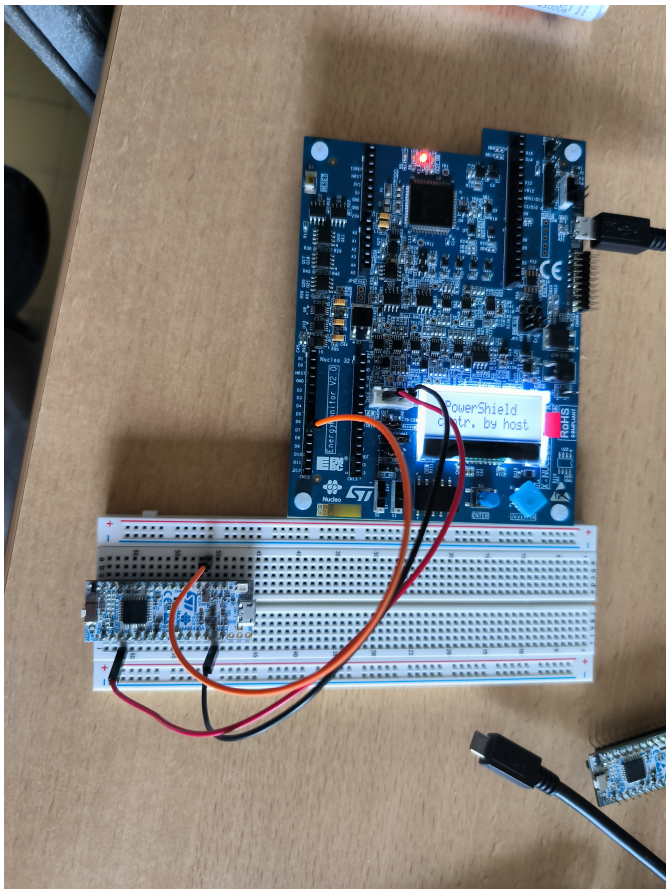
- Consumption averaging (from 1 nA up to 200 mA)
- Measure acquisition at 3.2 Msamples/s max
- Transfer measures with USB

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

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. Microcontrolers features
3. Current measurement
4. STM32F performances
5. STM32G performances
6. Optimization model
7. Future work

First measures for current consumption

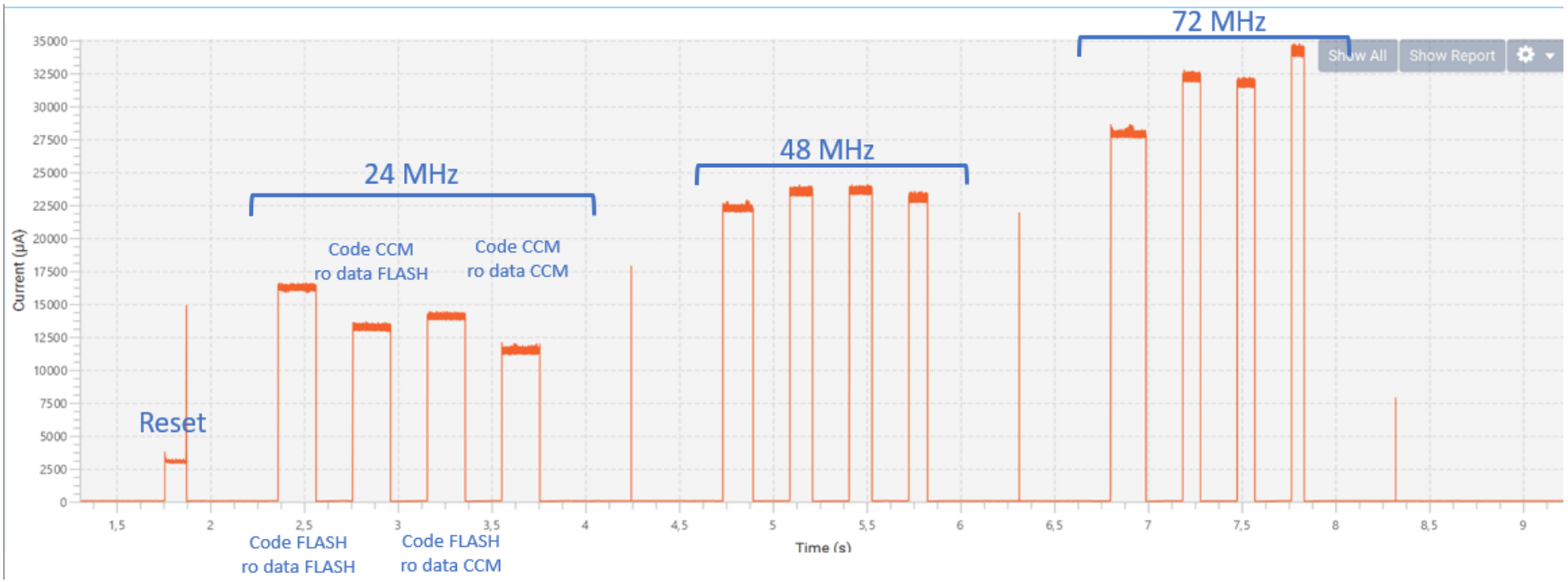


Figure: Intensity consumption graph for differents pointer chase executions

Final measures for current consumption

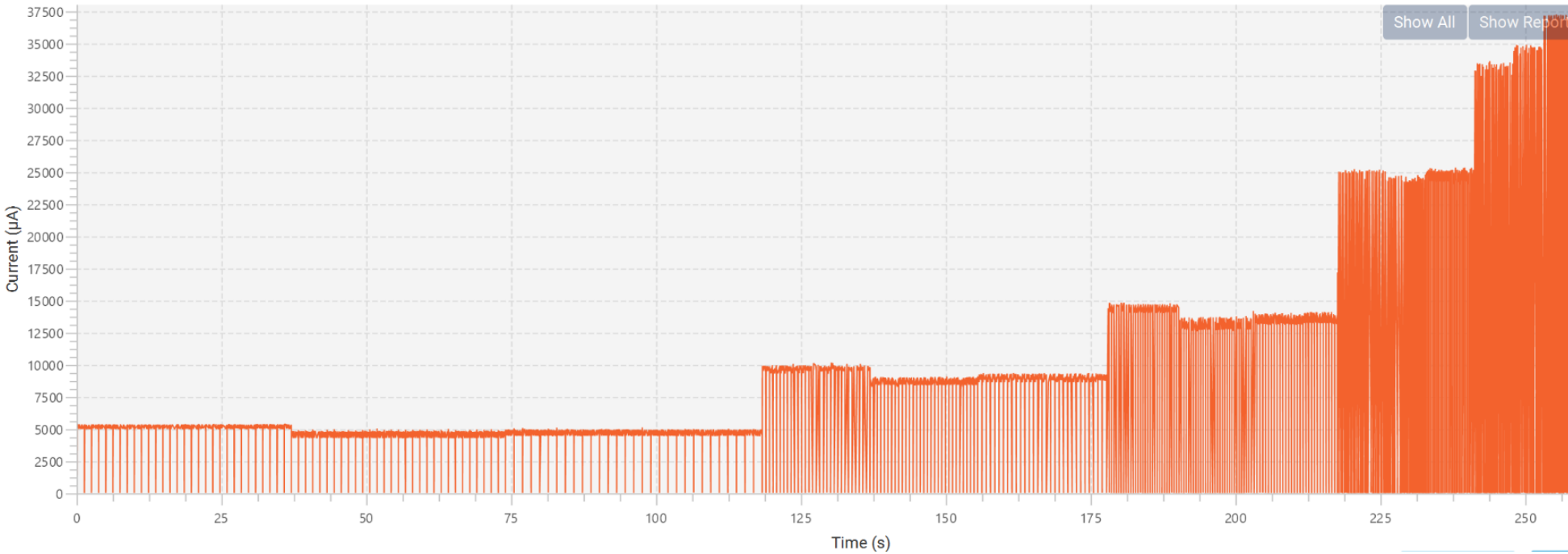
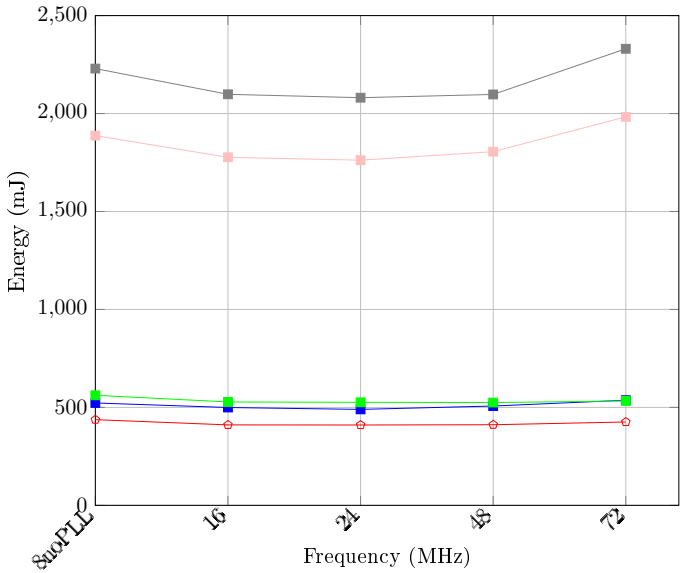


Figure: Intensity consumption graph with 30 execution per memory configuration

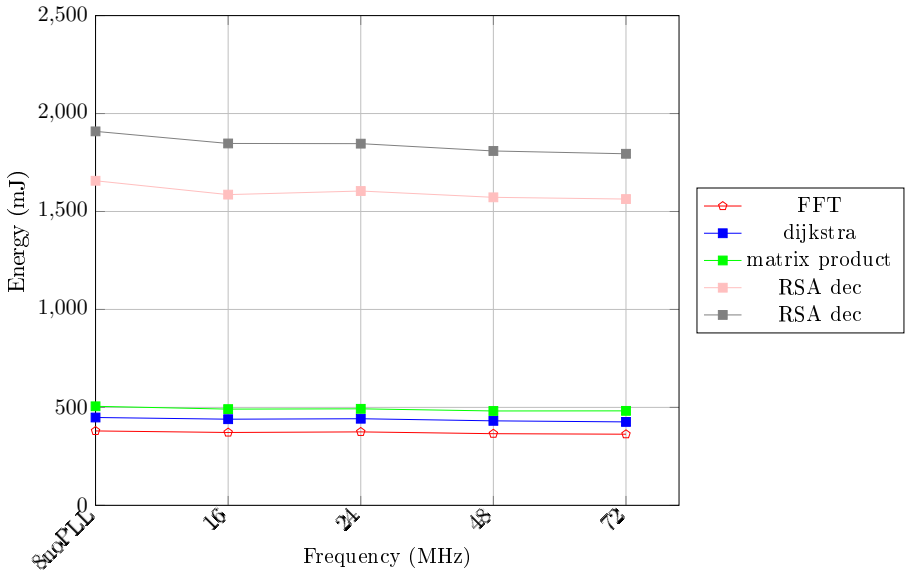
Table of Contents

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

Absolute energy (mJ) when code is in FLASH and CCM



(a) Energy (mJ) when code in FLASH



(b) Energy when code in CCM

- *Instructions in FLASH : frequency* $\nearrow \Rightarrow$ *energy* $\nearrow$
- *Instructions in CCM : frequency* $\nearrow \Rightarrow$ *energy* $\rightarrow$

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

Table of Contents

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

STM32G features

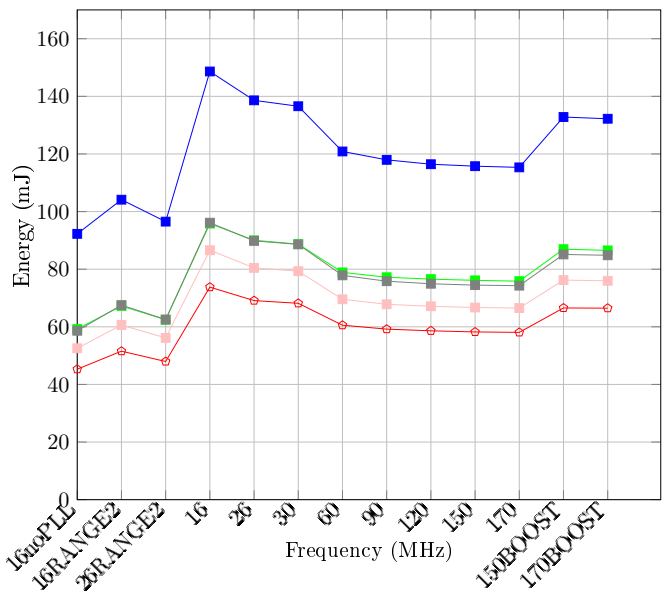
Different execution modes

- 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.

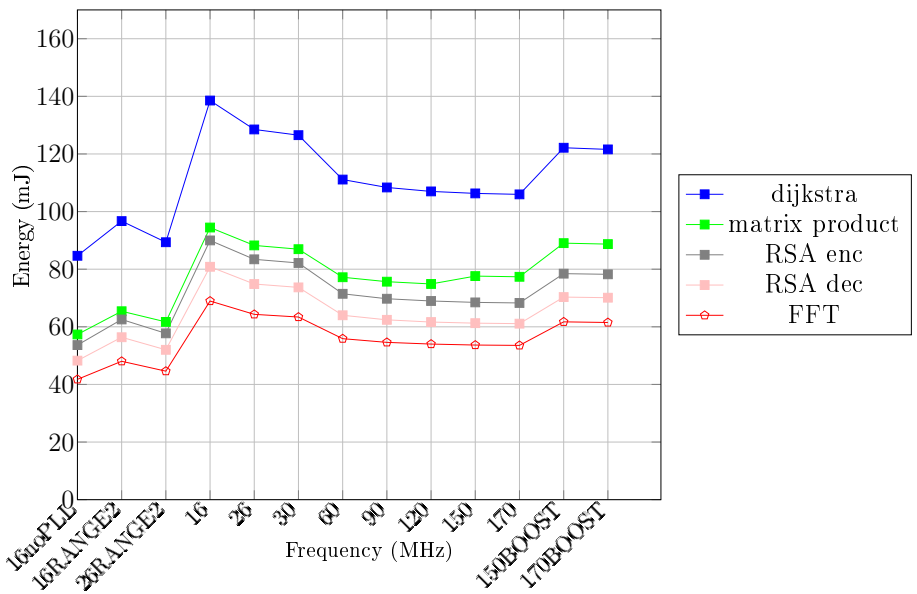
FLASH instruction cache

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

DVFS and CCM impact on STM32G



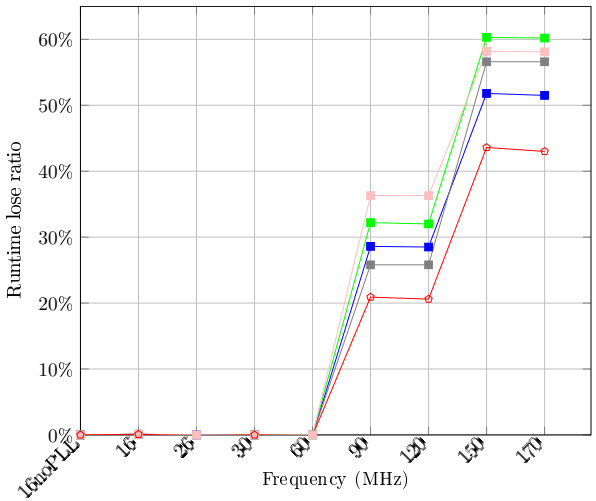
(a) Energy when code in FLASH with Cache



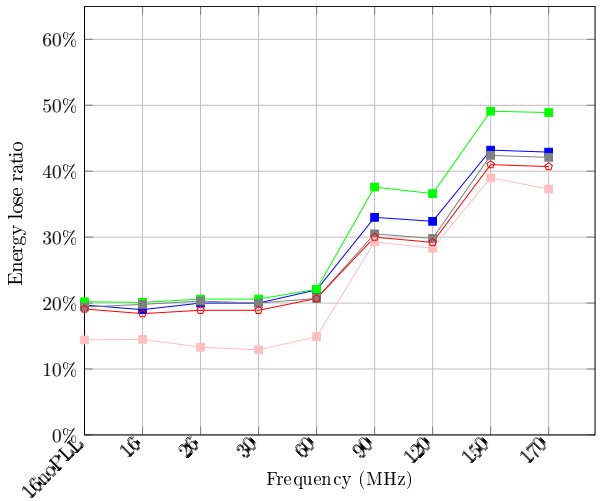
(b) Energy when code in CCM

- Low power mode lowers the energy consumption, but used only in low frequencies.
- BOOST mode consumes more energy to go faster at high frequencies.

Cache impact



(a) Runtime loss when cache disabled



(b) Energy loss when cache 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

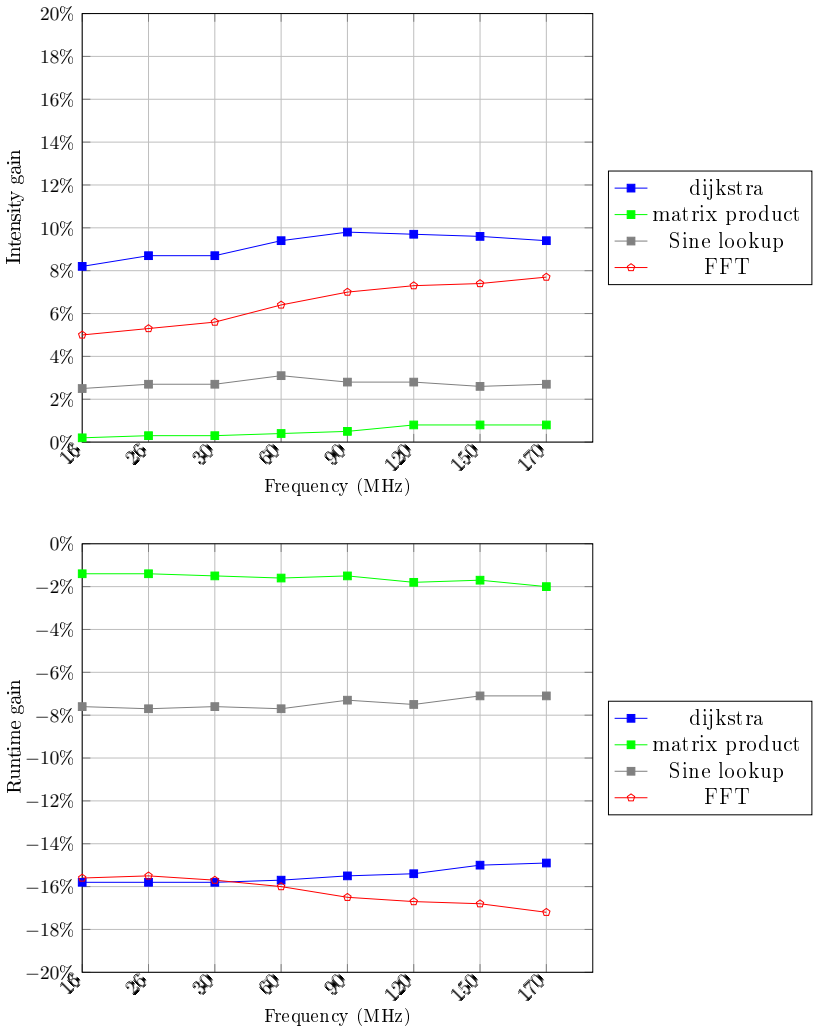


Figure: *Intensity* and *Runtime* gain when we move data from RAM1 to RAM2 (code FLASH)

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

It will be better to put data in SRAM1.

Pre-Fetch in STM32G

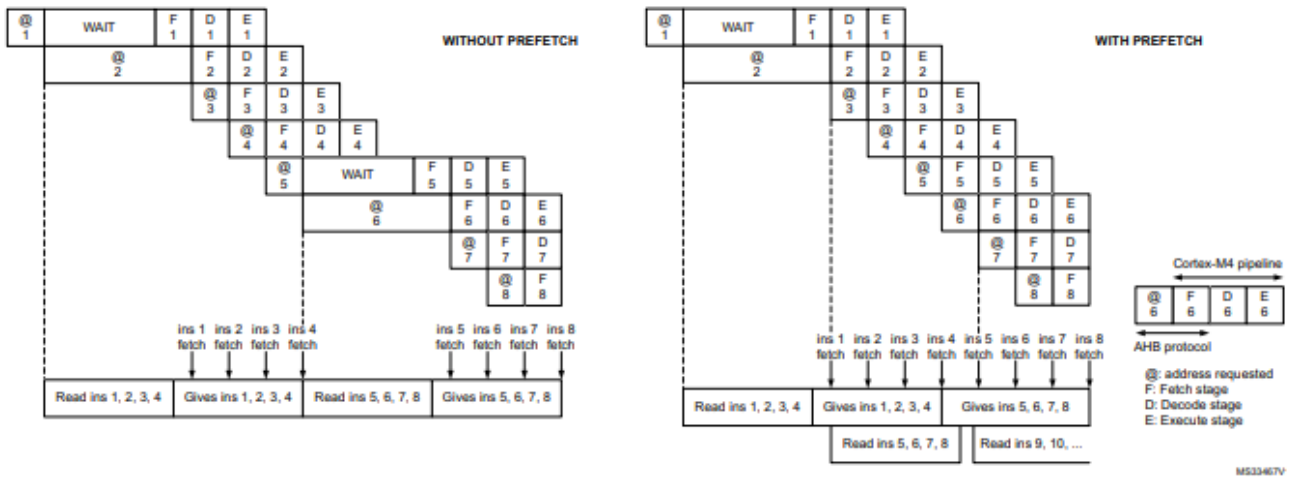
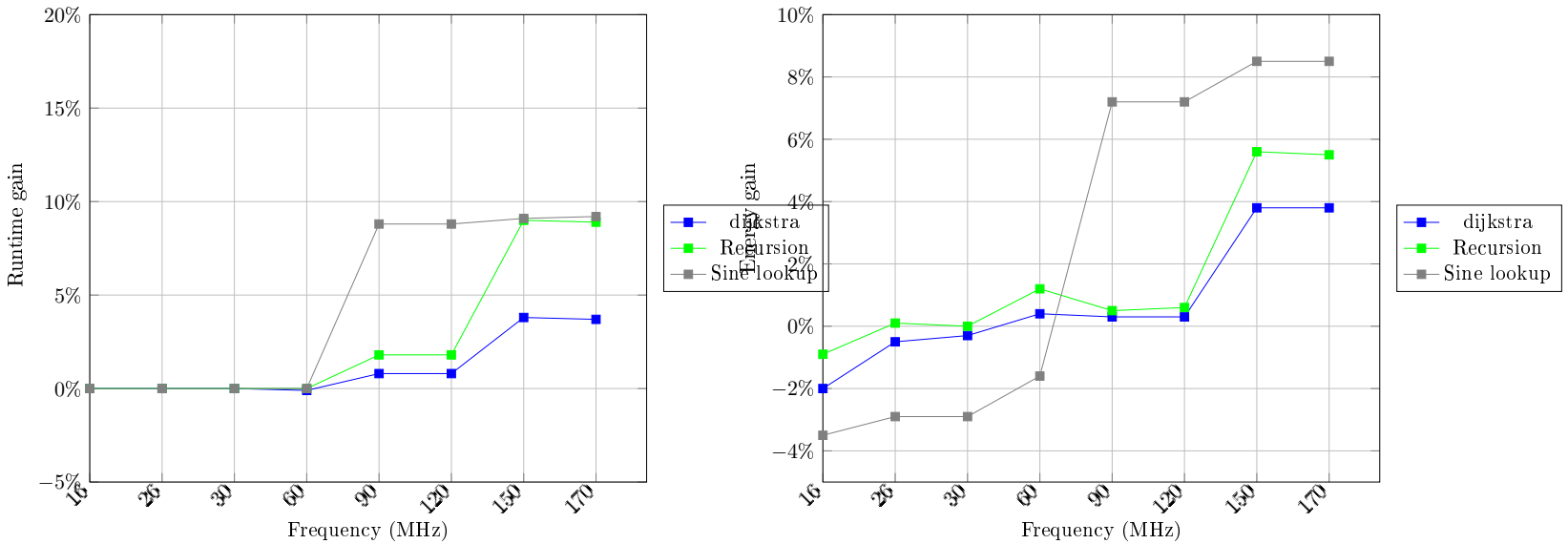


Figure: Sequential 16-bit instructions execution (64-bit read data width)

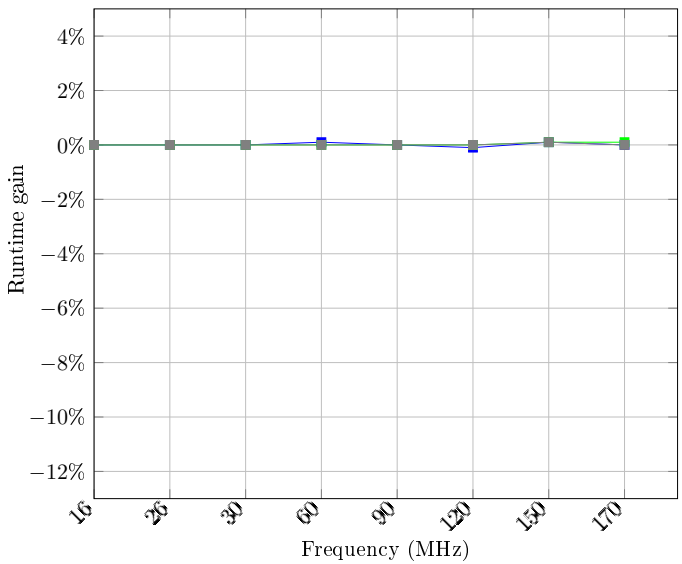
Pre-fetch on the instruction bus can be used to read the next sequential instruction line from the Flash memory while the current instruction line is being requested by the CPU.

Pre-Fetch impact when cache is disabled

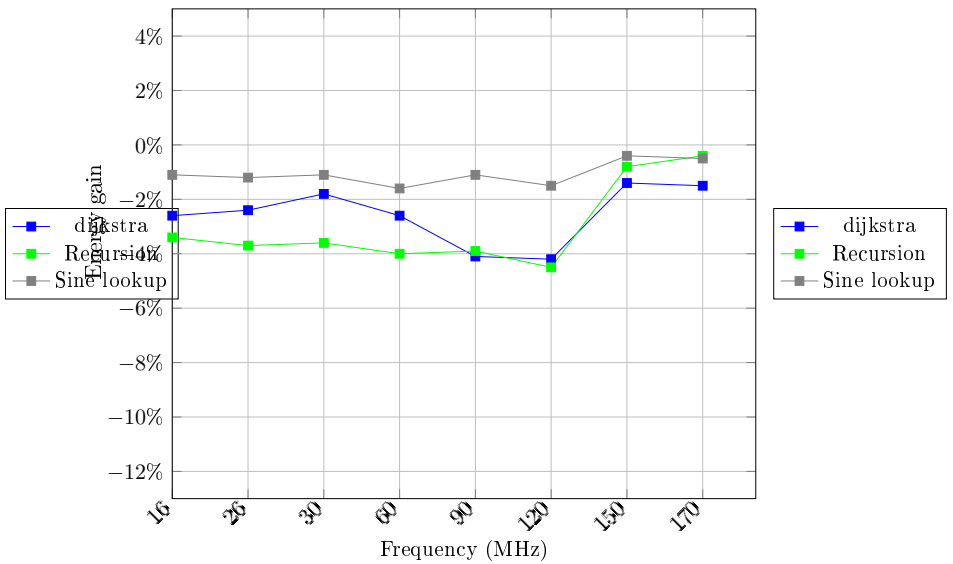


(a) Runtime gain if pre-fetch enabled (Cache OFF) (b) Energy gain if pre-fetch enabled (Cache OFF)

Pre-Fetch impact when cache is enabled



(a) Runtime gain if pre-fetch enabled (Cache ON)



(b) Energy gain if pre-fetch enabled (Cache ON)

Table of Contents

1. Context and introduction
2. Microcontrolers 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 under EDF.
- On standard configuration the utilization is 1 $\Rightarrow$ always a solution

Model memory constraints

Constraints 3, 4 and 5

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

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

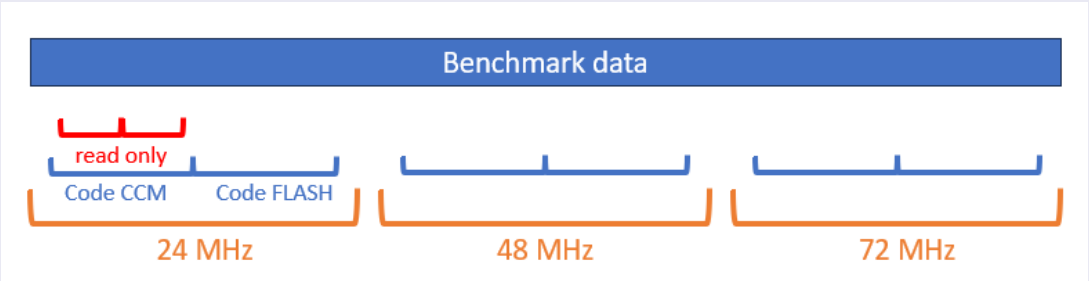
$$\sum_{i=1}^{n} \left(\sum_{c \in \mathcal{C}}^{d=R} \left(x_i^c \cdot m_i^D \right) + \sum_{c \in \mathcal{C}}^{ro=R} \left(x_i^c \cdot m_i^{Ro} \right) \right) \leq M^R$$

- 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)

Model implementation

Step 1

Redeem all the previous data and create arrays with all the configs.



Step 2

- Create a random taskset based on the benchmarks.
- The taskset is schedulable with the default configuration.

Step3

GUROBI OPTIMIZATION

We use the solver Gurobi on our model to find the best configuration.

Energy results on STM32F

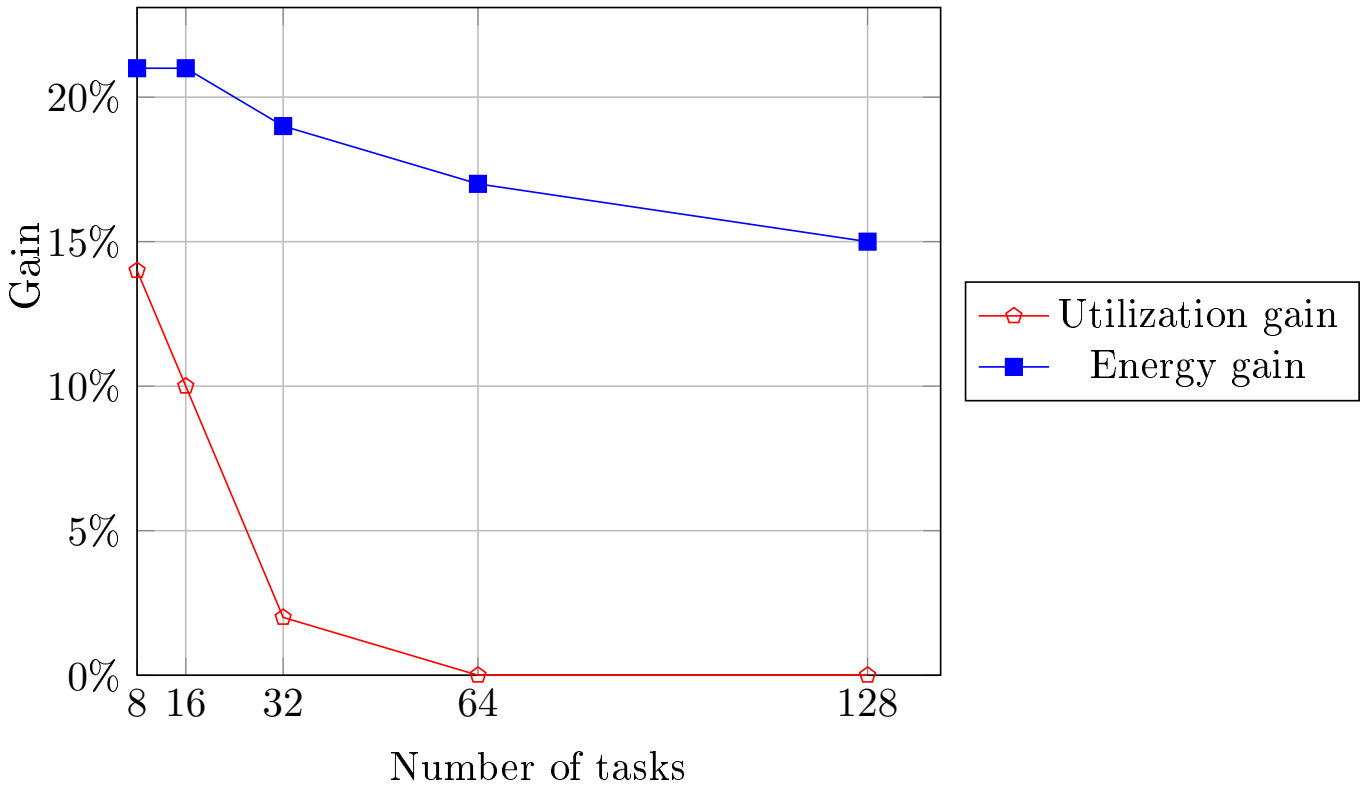


Figure: Utilization and energy gains with the model on STM32F

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

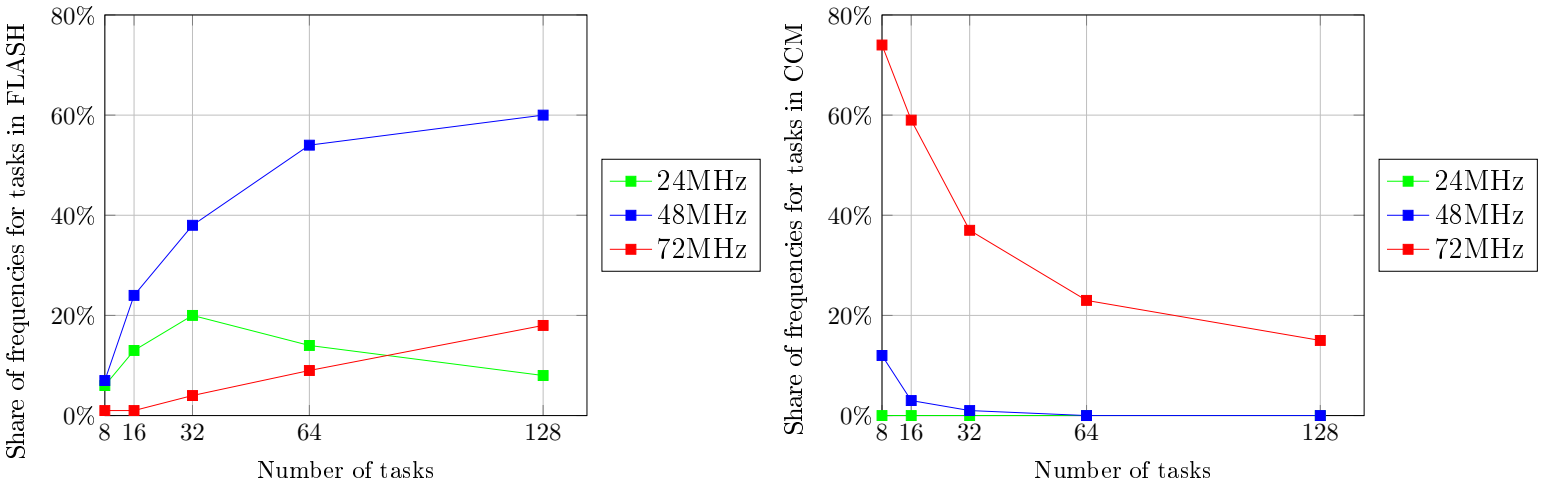
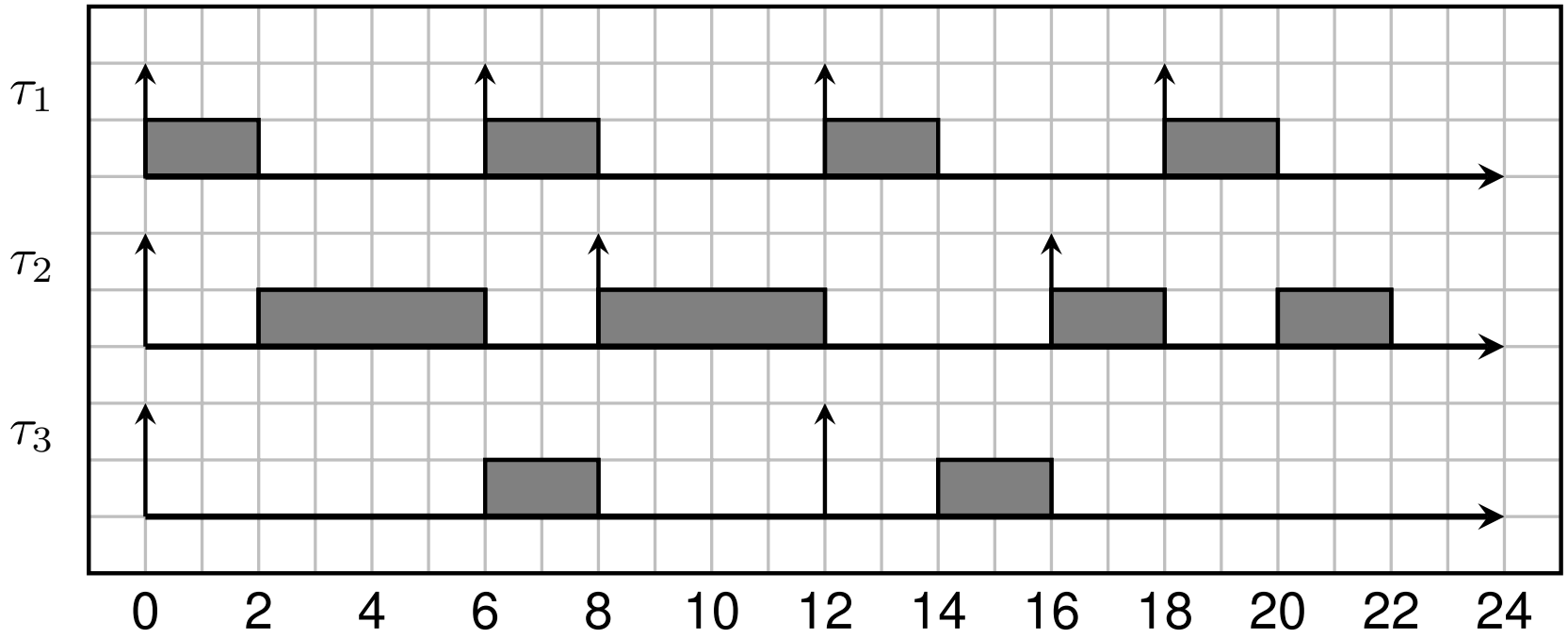


Figure: Frequency choices made by the model for tasks in FLASH and CCM

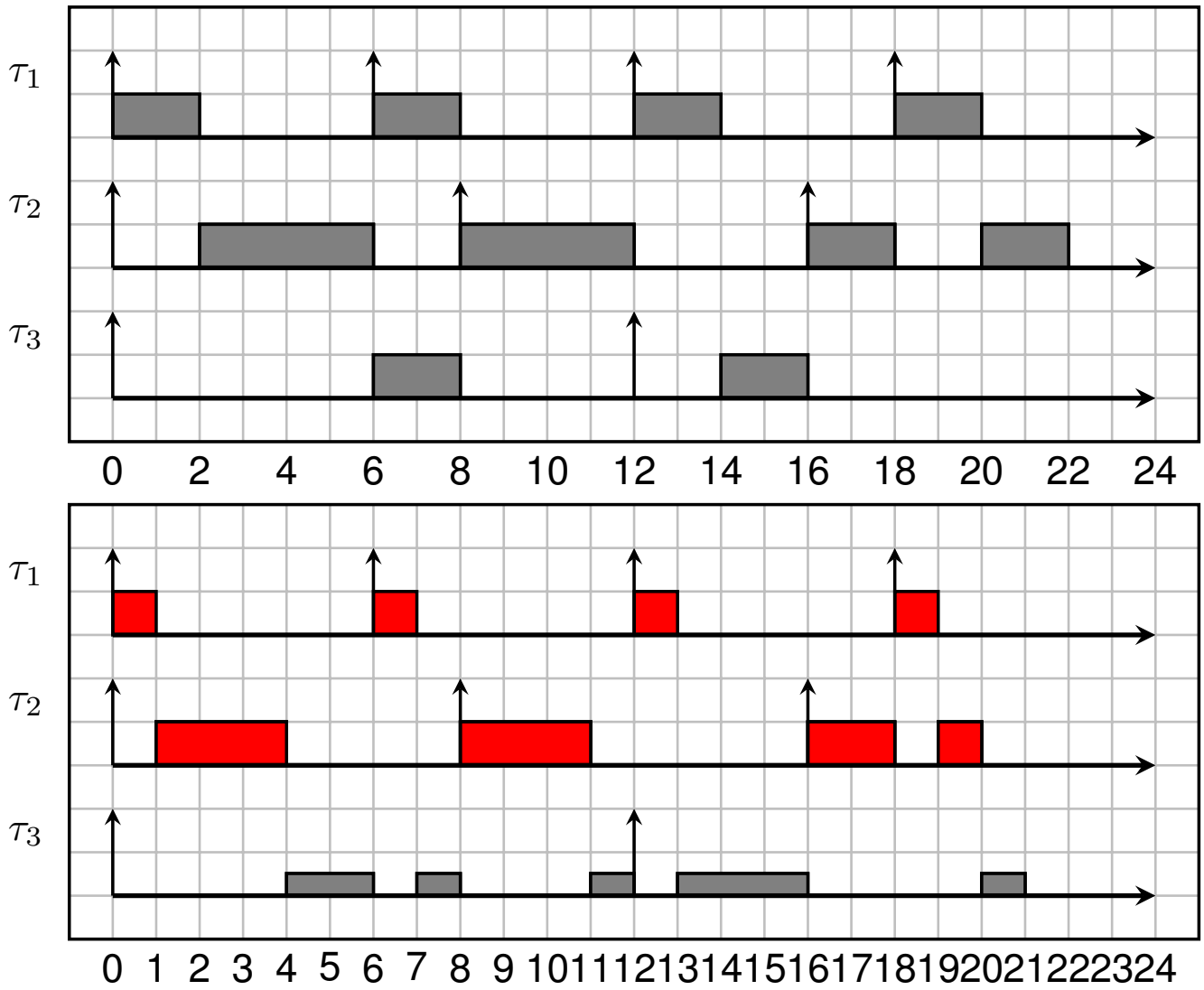
nb tasks $\nearrow \Rightarrow$ proportion in CCM $\searrow$
Low number of tasks $\Rightarrow$ tasks in FLASH at low or mid frequency.
High number of tasks $\Rightarrow$ tasks in FLASH at 72 MHz
$\Rightarrow$ if system fully-loaded $\rightarrow$ default config (72MHz and FLASH).

Normal EDF example

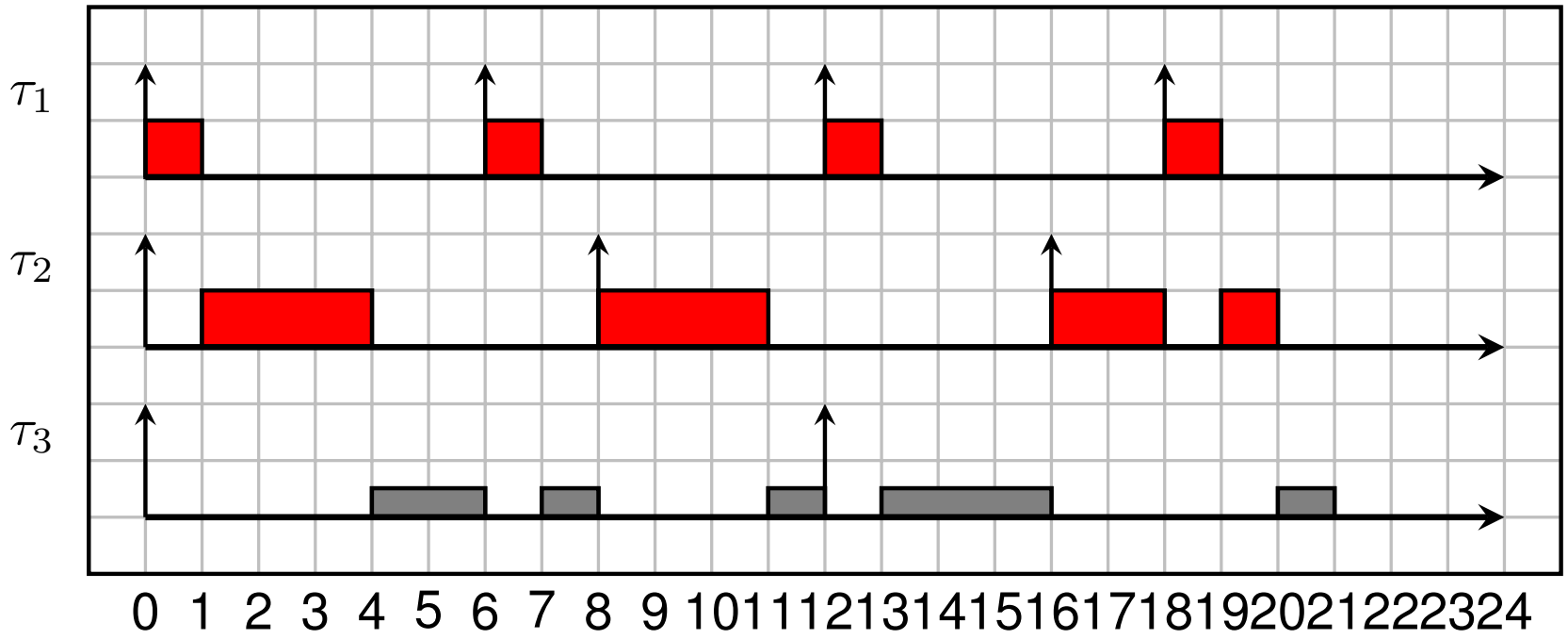


We have 3 tasks in the standard configuration.
Instructions are in FLASH and the CPU runs them at 72 MHz.

Comparison if we use the model



Application of the previous algorithm



We move τ_1 and τ_2 in the CCM
$\Rightarrow$ runtime is reduced and consume less energy.

We use the utilization gain of the CCM to run τ_3 at 48 MHz
$\Rightarrow$ runtime is bigger, but, we consume less energy.

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

Model results on STM32G

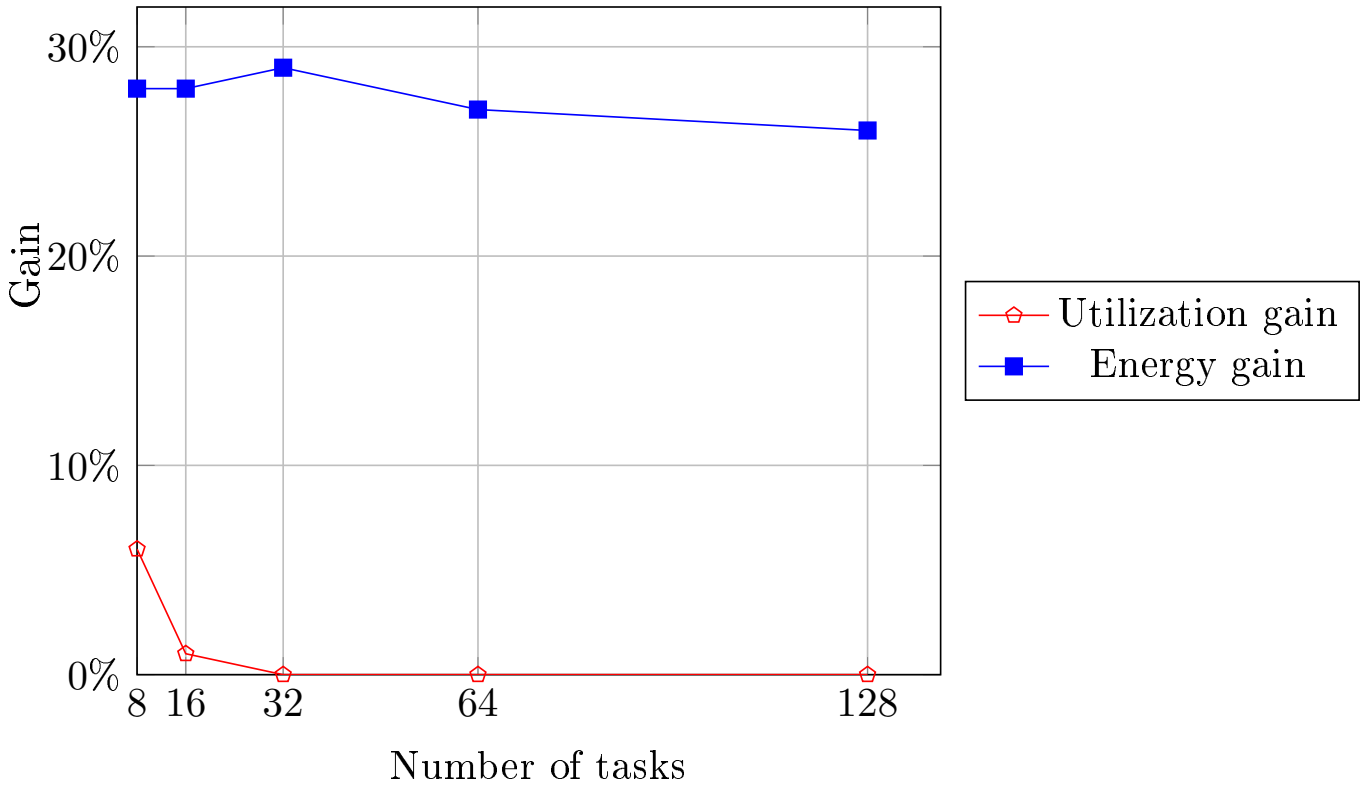


Figure: Utilization and energy gains with the model on STM32G

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

Frequency results on STM32G

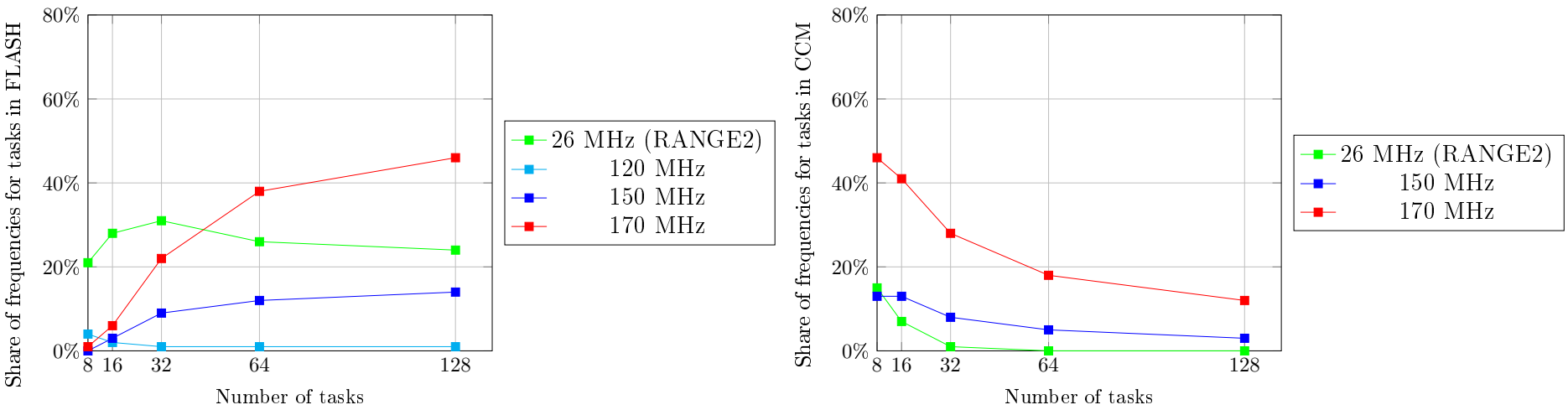


Figure: Frequency choices made by the model for tasks in FLASH and CCM for STM32G

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. Microcontrolers 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 execution time before the deadline, then we lower the frequency.

Bambagini, Mario and Marinoni, Mauro and Aydin, Hakan and Buttazzo, Giorgio

Energy-aware scheduling for real-time systems: A survey.*ACM TECS 2016*

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

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

The End

Questions? Comments?