How to prove the existence of an optimal frequency assignment algorithm for CPU and memory accesses in absence of an appropriate energy model

Myriam Mabrouki*, Liliana Cucu-Grosjean*, Stéphan Plassart†
*Kopernic, Inria, France, Email: firstname.lastname@inria.fr
† University of Savoie Mont-Blanc, France, Email: firstname.lastname@univ-smb.fr

***Abstract*—While targeting a reduced energy consumption for the embedded real-time systems community, different techniques exist and, in this paper, we are interested in Dynamic Voltage and Frequency Scaling. Quan and Hu propose an algorithm ensuring an energy-optimal CPU frequency assignment for executing independent programs on a single-core processor under real-time constraints. Since its optimality is proposed for CPU variable frequencies, we propose to add variable frequency assignment for memory accesses to the problem solved by Quan and Hu. Based on numerical evaluation of energy consumption of TACLeBench programs, we study the impact of both CPU and memory accesses frequencies on the energy consumption but no existing energy model considers these two factors. Moving back to the existence of an optimal algorithm to assign both frequencies, one may propose to find the appropriate pair of CPU-memory accesses frequencies, but an appropriate energy model is required to compare two different pairs.**

I. Motivation, our problem and existing results

Since reducing energy consumption is a major concern for the embedded real-time systems community, techniques like Dynamic Voltage and Frequency Scaling (DVFS) [1] and Dynamic Power Management (DPM) [2] have been developed. In particular, DVFS consists of reducing processor voltage and frequency to reduce power consumption while meeting timing constraints. Nonetheless, in the literature this technique mainly considers the processor frequency (or the CPU frequency) and not the memory frequency. In this paper, we study the impact of memory accesses on energy consumption and discuss the existence of an optimal frequency assignment algorithm for CPU and memory accesses.

Our problem More precisely, we consider a *real-time system* composed of a set τ of n independent, synchronous, implicit deadline, periodic tasks $\{\tau_1, \tau_2, \cdots, \tau_n\}$ scheduled by a preemptive fixed-priority scheduling algorithm on a single core processor.

Each task τ_i is defined by its worst-case execution time (WCET) C_i, and its period T_i equal to its deadline. An execution or instance of a task τ_i is called job. The execution time of the jth instance, i.e. the jth job, of task τ_i is denoted c_j. The execution time of a job is not necessarily equal to the WCET of the task. Since we deal with periodic tasks, a task releases a job at any instant $k * T_i$, where $k \in \mathbb{N}$.

Within the paper, unless stated otherwise, we consider a *fixed-priority algorithm*. Without any loss of generality, the priority assignment is given: the tasks are ordered in the decreased order of their priorities, i.e., τ_i has a higher priority than τ_j if $i < j$. Tasks are executed on a processor with one core and one memory storage, each of which has an associated execution frequency, i.e., new operations are performed at each new cycle. The processor frequency is denoted as f_p and the memory frequency is denoted as f_m. Moreover, f_p and f_m vary by having discrete values respectively within the interval $[0, f_p^{max}]$ and $[0, f_m^{max}]$ with f_p^{max} maximum processor frequency and f_m^{max} maximum memory frequency. Since an execution time of a task τ_i depends on CPU and memory frequencies, we denote it as $c_i^{f_p,f_m}$.

In this paper, we consider that the hardware features of the processor are such that we cannot modify the frequency during the execution of a program to match realistic hardware components. Thus, within our model of real-time system we consider that both frequencies, memory and processor are fixed during the schedule.

Existing work To the best of our knowledge, there is no result providing a solution of finding an optimal frequency assignment for both CPU and memory accesses to minimize energy consumption for the real-time systems considered in this paper. If one considers only the CPU frequency, then Hong et al. [3] propose a heuristic scheduler, while Quan and Hu provide a heuristic [4] and then a speed schedule for a fixed-priority real-time system, based on the offline algorithm of Yao et al. [1], that leads to a minimal energy consumption [5]. Moreover, these results consider that the CPU frequency is modifiable during the execution of tasks. Interestingly, Kim et al. [6] show that the lowest processor and memory frequencies do not always imply the lowest energy consumption.

Organization of the paper: We provide a detailed description of our experimental environment, then we propose a discussion on obtained energy consumption measures as well as the identification of future work.

II. Description of experimental environment

We consider a real-time system composed of tasks or programs of the benchmark TACLeBench [7] executed on a Raspberry Pi 3 model B+ [8]. This micro-controller has a 64-bit ARM Cortex-A53 quad-core processor, whose frequency should vary from 600 MHz to 1400 MHz. The memory is an *SDRAM*. The default minimum and maximum values set for the memory frequency are respectively 400 MHz and

500 MHz [9]. The operating system used is *Raspberry Pi OS*, a Linux operating system based on *Debian*.

In order to monitor the energy consumption, we use the power analyser *Otii Ace Pro* [10] and its associated software, *Otii 3 Desktop App* [11].

There are three steps in our experiments. The *first one* consists of creating several test programs with different quantities of memory accesses. The *second step* involves building a real-time system to study the impact of CPU and memory access frequency on the energy-consumption and the *third step* consists of measuring the energy consumption of our test programs and our system.

Building test programs We build four test programs. The first two programs are test programs that are used to control the impact of memory accesses, and the last two are actual applications on a realistic benchmark, TACLeBench [7].

The first program, called $test_mem$, involves allocating a two dimensional array and doing write accesses on each cell. These accesses are made in a way to maximise cache misses and, thus, memory accesses. Indeed, if at the iteration k we access the cell at the position (i, j), with i representing the row number and j the column number, then we access the cell at the position $(i+1, j)$ at the iteration $k+1$. The second program, called $test_cpu$, consists of looping as many times as there are cells in the array of the previous test. In this way, we have two control tests against which to compare energy consumption results.

The last two consist of executing one application of the benchmark TACLeBench [7] in a loop, however they differ with their sensitivity to the CPU frequency. The third program is *statemate*, executed in a loop of 100 iterations. It is chosen among all the TACLeBench tasks, because its execution time is less sensitive to the CPU frequency. Thus, this program is performing more memory requests (see Section III.B in [12]). The fourth program is *ammunition*. For this task, the execution time is more sensitive to the CPU frequency and should have less memory accesses [12]. Contrary to *statemate*, it is executed in a loop of 10 iterations due to its larger execution time.

All these tests are set to the highest priority with the `set_priority` function. Moreover, we transmit a message to the power analyser through the Universal asynchronous receiver transmitter (UART) protocol [13] at the beginning and at the end of each test. These messages are used to get precise timestamps of the beginning and the end of each measure.

Description of the real-time system Last section consider specific programs in isolation, however our real-time system is more complex, with several tasks. Therefore, we build a program called *global_system* composed of three tasks of TACLeBench [7] described in Table I : *statemate*, *ndes*, and *cjpeg_wrbmp*. These tasks are chosen since their execution times are less sensitive to the processors speed and, therefore, their number of memory accesses is higher than other tasks (see Section III.B in [12]).

The experimentation is done in two steps: First, the execution time of these three tasks are measured at the maximum

TABLE I: Task parameters of the systems

Task	i	$c_i^{f_p=1400, f_m=500}$	T_i
statemate	1	1.60 ms	9.86 ms
ndes	2	1.61 ms	9.86 ms
cjpeg_wrbmp	3	1.72 ms	9.86 ms

CPU and memory frequencies. Every application is executed on a single core with the `sched_setaffinity` function. To ensure our measures are as reliable as possible each program is set to the highest priority in the user-space with the `set_priority` function. Moreover, we disable, via *raspi-config* and *config.txt* the desktop mode, the camera module, Bluetooth, Wi-Fi and interfaces options (Serial Peripheral Interface (SPI) [14], Inter-Integrated Circuit (I2C) [15], Wire [16], Virtual network computing VNC [17], and Remote General Purpose Input/Output (GPIO)). Each application is executed 500 times. The execution time is measured with the `perf` command. Averages, standard deviations and 99% confidence intervals are then computed [18]. The execution times are the upper limit of the confidence intervals.

Second, we implement a scheduler [18]. The scheduler executes 1000 times a loop corresponding to the hyper-period of the real-time system. At the end of a loop, we check whether the tasks have been completed on time. If a deadline is missed, we end the program, thus the result is not considered in our experimentation. At the beginning and the end of the scheduler, we transmit a message to the power analyser with the UART protocol as in the test programs. Applying Quan and Hu's speed schedule to our system gives us a single interval with an associated speed which is the initial speed, thus the initial frequency, divided by two.

Energy consumption measures We power supply the power analyser *Otii Ace Pro* [10] with an adjustable power adapter. We connect the power analyser to *5V* and *Ground* pins of the micro-controller Raspberry Pi 3B+ through banana to jumper wires and to a computer through a USB wire. We also connect *RX* and *Digital Ground* pins of the power analyser respectively to *TX* and *Ground* pins of the micro-controller to capture UART logs. We use the software *Otii 3 Desktop App* [11] to switch on the power analyser which switches on the Raspberry Pi 3B+.

Since the micro-controller has a quad-core processor, we disable three of them to get a single core by setting `maxcpus=1` in */boot/firmware/cmdline.txt*.

In order to see how energy consumption evolves and to figure out a pattern, we measure the energy consumption under different frequency assignments for CPU and memory. We call a *configuration* a pair of CPU frequency and memory frequency (f_p, f_m). For each configuration, we measure our programs 100 times.

The configurations (f_p, f_m) we choose for our test programs *test_cpu, test_mem, statemate* and *ammunition* are the ones with f_p varying from 200 MHz to 1400 MHz in steps of 200 and f_m varying from 200 MHz to 500 MHz in steps of 100. The ranges chosen for the memory frequency are similar to the ones used in [12].

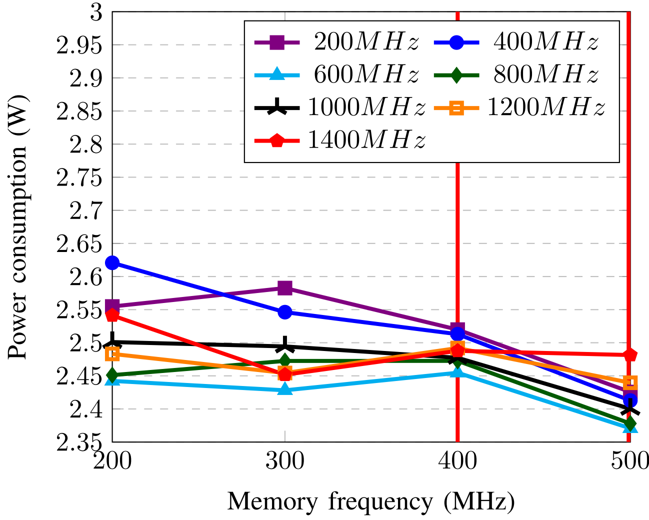

Fig. 1: Power consumption according to memory frequency at different fixed CPU frequency for $test_cpu$

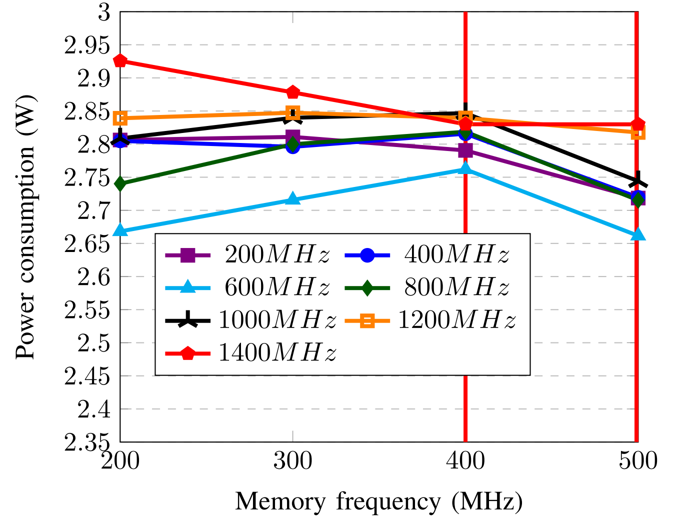

Fig. 2: Power consumption according to memory frequency at different fixed CPU frequency for $test_mem$

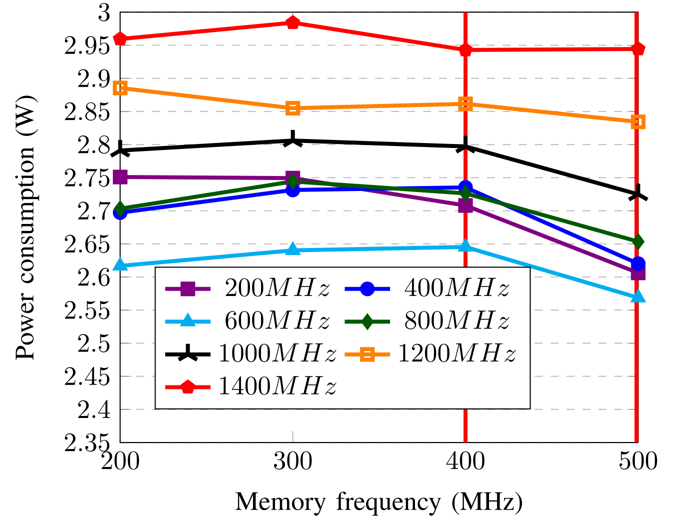

Fig. 3: Power consumption according to memory frequency at different fixed CPU frequency for $ammunition$

For the *global_system* program, the CPU frequency is set up to 700 MHz according to Quan and Hu's algorithm output. We measure the energy consumption under different memory frequency assignment; $f_m \in \{200, 300, 400, 500, 600\}$.

CPU frequency is set with `arm_freq` in */boot/firmware/config.txt*. Since we do not want the frequency to lower during the execution, we set `arm_freq_min` in */boot/firmware/config.txt* with the same frequency. In the same way, memory frequency is set with `sdram_freq` and `sdram_freq_min` in */boot/firmware/config.txt*.

Although, lowering the CPU frequency under 600 MHz does not results in power savings, we keep configurations where the CPU frequency is below 600 MHz to observe power consumption patterns, especially when the memory frequency is equal to or higher than the CPU frequency. The same goes for the memory frequency, even if it is not explicitly mentioned that memory frequencies below 400 MHz are not supported. In addition, we go above the indicated maximum frequency of the memory for the *global_system* program since we can overclock the SDRAM in order to have an additional power consumption value.

III. Discussion on Energy Consumption Results and Future Work

The results obtained are in two CSV files [18] for reproducibility purpose. One is about power consumption results and the other contains the UART log. A power consumption is given every 20 microseconds. We compute the average power consumption of each measures and then we compute the average power consumption between all measures [18]. To do that, we consider power consumptions whose timestamps are included between the timestamps "end" and "begin" given in the UART log file. Timestamps in the UART log file are given in milliseconds. It is not the same precision as the timestamps in the power consumption file but is sufficient for our measures since the power consumption does not vary significantly during one millisecond. In addition, standard deviations, minimum and maximum are also computed [18].

When we look at the results of our test programs (Figures 1–4), we first see that the program *test_mem* is more power consuming than *test_cpu* as expected since *test_mem* does the same as *test_cpu* but with more memory accesses. Moreover, *ammunition* is more power consuming than *statemate*. The program *ammunition* has less memory accesses so the variations of the power consumption is mainly due to CPU frequencies. Conversely, with *statemate*, the memory frequency seems to have more impact than the CPU frequency on the power consumption. It is consistent with the characteristics of this task which has more memory requests.

The Rasberry Pi3 model B+ has a supported memory

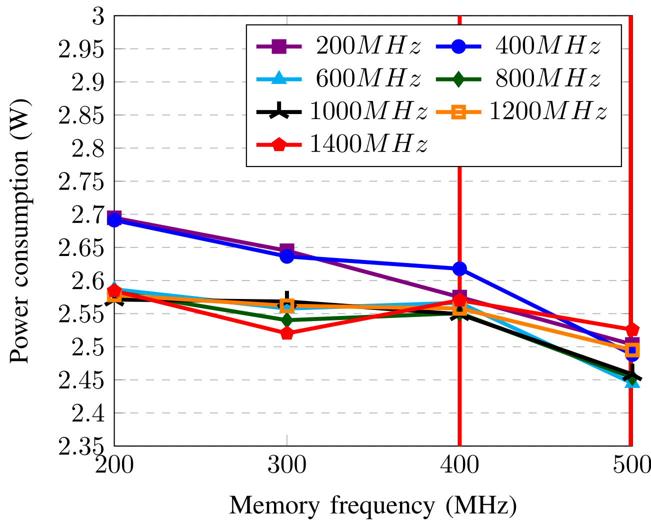

Fig. 4: Power consumption according to memory frequency at different fixed CPU frequency for *statemate*

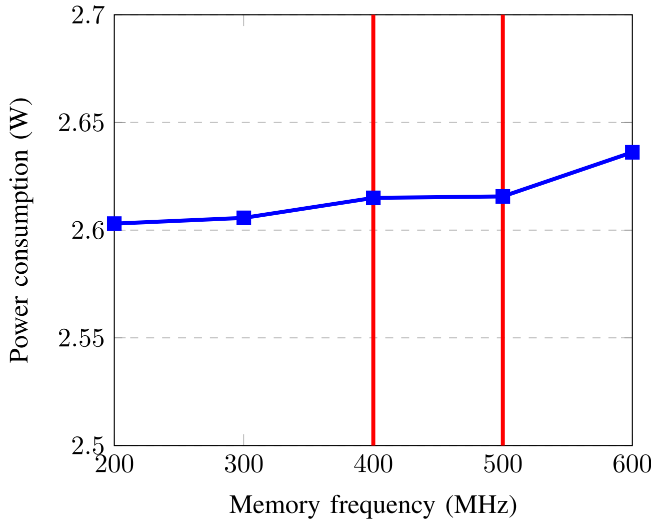

Fig. 5: Power consumption according to memory frequency at a CPU frequency of 700 MHz for *global_system*

frequency range of 400MHz to 500MHz [9] (these limits are represented by the vertical red lines in Figures 1–5). On this specific interval, one can observe that no matter the CPU frequency, the power consumption obtained at higher memory frequency, i.e., f_m = 500 MHz, is equivalent or lower than the power consumption obtained at lower memory frequency, i.e., f_m = 400 MHz.

When we consider lower CPU frequencies, the lowest power consumption is almost every time associated with the highest memory frequency. We observe that this is never the lowest memory frequency, nor the one closest to the CPU frequency that leads to the minimal energy consumption. Most of the time, the patterns of the power consumption in Figures 1–4 are either decreasing curves, or curves that increase and then decrease. Based on these results, there is a single memory frequency (this frequency differs according to the programs and the CPU frequency) that leads to the highest power consumption which means that, starting from this frequency, we can either increase or decrease the memory frequency to lower the power consumption. Given that lowering the memory frequency increases the execution time [12], it is better to increase the memory frequency to lower the total energy consumption. However, sometimes the curves of power consumption decrease, increase and then decrease again. Moreover, the power consumption decreases with the CPU frequency until the minimum default value which is f_p = 600 MHz as expected since lowering the CPU does not results in power savings [9].

As for the power consumption of the considered real-time system, we observe that the behaviour of the power consumption differs from the one of the test programs. Indeed, we can see in Figure 5 the power consumption is increasing with the memory frequency. If we link this result to the analysis done in the previous paragraph, it could mean we should increase the memory frequency to get the memory frequency leading to the highest power consumption. Notwithstanding, the power consumption is increasing very slowly. The lowest power consumption in this case is obtained when the memory frequency is set to 200 MHz and equals 2.60 W. The highest power consumption is obtained when the memory frequency is set to 600 MHz and its value is 2.64 W. Although both power consumptions are very close, we can get a lower power consumption. Therefore, there is a gain of 1.5% in power consumption when exploiting the impact of memory frequency.

Yet, in terms of gain in power consumption the highest we get are with *statemate* at f_p = 400 MHz with a reduce rate of 7.4% or with *test_cpu* at f_p = 400 MHz with a reduce rate of 8%. Indeed, with *statemate* we get a power consumption of 2.69 W at f_m = 200 MHz and 2.49 W at f_m = 500 MHz. For *test_cpu*, the power consumption equals 2.62 W at f_m = 200 MHz and then decreases to 2.41 W at f_m = 500 MHz. However, *test_cpu* is not supposed to have a lot of memory accesses.

Nevertheless, the measures show huge variations of power consumption, in spite of the pattern of the power consumption for each program. Indeed, the power consumption varies from 1.8 W to 4 W. Even when the CPU is idle, the power consumption varies around the same values.

Our discussion indicates that an energy model considering both CPU and memory accesses frequency is not straightforward. Moving back to the existence of an optimal algorithm to assign both frequencies, one may propose an exhaustive search algorithm by considering all possible pairs of CPU-memory accesses frequencies in order to identify the one with the lowest energy consumption. Nevertheless, an appropriate energy model is required to compare two different pairs within this exhaustive search algorithm. In conclusion, we identify as future work the proposition of an appropriate energy model as mandatory condition towards the proposition of an optimal algorithm.

REFERENCES

[1] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 374–382. IEEE Computer Society, 1995.

[2] Yann-Hang Lee, Krishna P. Reddy, and C. Mani Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS 2003), 2-4 July 2003, Porto, Portugal, Proceedings*, pages 105–112. IEEE Computer Society, 2003.

[3] Inki Hong, Gang Qu, M. Potkonjak, and M.B. Srivastavas. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 178–187, 1998.

[4] Gang Quan and Xiaobo Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 828–833. ACM, 2001.

[5] Gang Quan and Xiaobo Hu. Minimum energy fixed-priority scheduling for variable voltage processor. In *DATE*, pages 782–787. IEEE Computer Society, 2002.

[6] Young-Jin Kim and Jihong Kim. Exploration of memory-aware dynamic voltage scheduling for soft real-time applications. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005), 17-19 August 2005, Hong Kong, China*, pages 177–180. IEEE Computer Society, 2005.

[7] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. TACLeBench: A benchmark collection to support worst-case execution time research. In Martin Schoeberl, editor, *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:10, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[8] Raspberry pi 3 model b+. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#raspberry-pi-3-model-b, 2024.

[9] config.txt. https://www.raspberrypi.com/documentation/computers/config_txt.html, 2024.

[10] Otii ace pro. https://docs.qoitech.com/user-manual/otii/hardware/otii-ace-pro#datasheet, 2024.

[11] Otii 3 desktop app. https://docs.qoitech.com/user-manual/otii/software, 2024.

[12] Roberto Medina and Liliana Cucu-Grosjean. Work-in-progress: Probabilistic system-wide DVFS for real-time embedded systems. In *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*, pages 508–511. IEEE, 2019.

[13] He Chun-zhi, Xia Yin-shui, and Wang Lun-yao. A universal asynchronous receiver transmitter design. In *2011 International Conference on Electronics, Communications and Control (ICECC)*, pages 691–694, 2011.

[14] Cliff Wootton. *Serial Peripheral Interface (SPI)*, pages 335–349. Apress, Berkeley, CA, 2016.

[15] Jayant Mankar, Chaitali Darode, Komal Trivedi, Madhura Kanoje, and Prachi Shahare. Review of i2c protocol. *International Journal of Research in Advent Technology*, 2(1), 2014.

[16] Sashavalli Maniyar. 1-wire® communication with pic® microcontroller. In *Application note AN1199*. Microchip Technology Inc, 2008.

[17] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[18] https://github.com/Energy-Memory/Energy-Memory-Acesses, 2024.