

# Inter-Core Communication Performance Evaluation of a Multicore Microcontroller for Edge Computing Applications

Tamás Kovácsházy

*Department of Measurement and Information Systems  
Budapest University of  
Technology and Economics (BME-MIT)  
Budapest, Hungary  
khazy@mit.bme.hu*

Gergely Kovács

*Department of Measurement and Information Systems  
Budapest University of  
Technology and Economics (BME-MIT)  
Budapest, Hungary  
kovacsgergely@edu.bme.hu*

**Abstract**—Multicore microcontrollers are available on the market to provide the required real-time EDGE computing resources present in modern distributed embedded systems. These devices contain multiple processor cores interconnected with device-specific communication solutions to utilize the computing resources of the cores for applications. Ideally, the computing resources of the individual cores add together and can be fully utilized by the embedded software running on these devices. However, due to the inefficiencies of inter-core communication, practical applications can expect drastically lower available throughput, and the actual throughput depends on the selected inter-core communication software and its configuration substantially. It is paramount to have some information about this performance loss before application development. Unfortunately, there exists no common benchmark for the performance evaluation of these inter-core communication solutions nor quality performance data is available. In the paper, we introduce our initial performance evaluation of the STM32H745 two-core MCU using the ST-provided inter-core communication framework for the FreeRTOS operating system. As the memory architecture of the STM32H745 is quite complex, we investigate how memory allocation and caching influence the performance. The performance is evaluated as a function of individual data transfer sizes. We measure the delay and data rate of communication.

**Index Terms**—Inter-Core Communication, Multicore Microcontroller, Edge Computing, Artificial Intelligence, Performance Evaluation, Message Passing

## I. INTRODUCTION

High-performance computing tasks, such as Artificial Intelligence (AI) and classic array processing (sound, video, radar, etc.), are executed in the cloud computing environment in modern distributed embedded systems (EDGE). However, the cloud cannot provide execution time guarantees required in real-time systems, partially due to the unacceptable delay and jitter of communication between the EDGE and the cloud, and partially due to the task scheduling inefficiencies of the

cloud. Therefore, it is unavoidable to execute computationally complex tasks on EDGE devices, such as microcontrollers (MCU). To accommodate these new requirements MCUs have started to change lately, and some new types of devices have appeared on the market. Earlier MCUs were designed to be low-power, relatively low-speed solutions utilizing a single, low-clock-speed processor core, but new ones introduce substantial changes. First, the clock speed of MCUs has been increasing continuously reaching GHz speeds today (at the time of writing the paper, the highest speed MCU known by the authors is the TI AM2x with a maximum of 800 MHz core clock speed). Second, multicore MCUs appear on the market integrating multiple identical and/or different processor cores on the chip with diverse interconnect, memory, and peripheral architectures.

## A. Multicore Microcontrollers and their Application

Multi-core general-purpose CPU architectures, such as the x86 and ARM Cortex-A, have been on the market for more than 15 years, and their software challenges (scheduling, inter-core communication, etc.) are considered solved [1] mainly in the scientific community. Contrary to this, the mainstream application of multi-core MCUs is new both in the industry and in scientific research. For example, one of the first multi-core MCUs on the market was the Infineon TriCore architecture in the Aurix MCU line in 2011 [2], scientific paper addressing the multi-core capabilities only started to appear in the last year [3], [4] based on our literature study. Our industrial practical experience shows also very slow acceptance of the multi-core features for anything other than implementing hardware-level safety functions, like master-checker pair CPUs and intelligent watchdogs, primarily due to a lack of software support for more diverse use cases.

Currently, most mainstream MCU manufacturers offer multi-core MCUs. Some STM32H7x devices have an additional Cortex-M4 processor augmenting the standard Cortex-M7, in this paper, the STM32H745 is investigated with this architecture. The TI AM2x line of MCUs provides a maximum

This work has received partial funding at BME-MIT from the European Union's Horizon 2020 research and innovation program under Grant Agreement No 872614 - SMART4ALL: Selfsustained Cross Border Customized Cyberphysical System Experiments for Capacity Building among European Stakeholders.

of four identical ARM Cortex-R processors and a Cortex-M4, with additional specialized execution units for real-time execution. Some ESP32 wireless MCUs from Espressif Systems have two MCU cores and an additional Low-Power MCU. NXP has the i.MX RT Crossover family of devices with mixed Cortex-M7/M4 and Cortex-M33/M7 processors.

These devices are mainly targeted towards high-performance distributed measurement and control applications with safety and real-time requirements. The automotive, aerospace, railways, energy, and similar industrial control fields experiment with them currently, primarily to efficiently implement computationally intensive EDGE algorithms, such as AI.

## II. SOFTWARE SUPPORT FOR INTER-CORE COMMUNICATION

Having hardware on the market is insufficient today, chip manufacturers must provide a complete software ecosystem for their MCUs for market success. The software ecosystems for MCUs include the integrated development environment (IDE), compilers, and also software components and example applications demonstrating and utilizing all hardware features. Therefore, inter-core communication software frameworks must be also provided for the new multicore MCUs in the development system letting developers build applications efficiently and on time. These inter-core communication frameworks are built on top of embedded operating systems (OS), such as FreeRTOS. Using an embedded OS is a typical approach in EDGE MCU software as the complexity of the software is substantial; for example, a large number of parallel tasks must be executed including ones related to network communication, data acquisition, data processing and control, etc., which is easier to implement or already implemented (such as the network communication stack) in embedded OSs as services.

The software interface for inter-core communication mimics the the software interfaces provided by the embedded OSs for inter-process communication (also called message passing), such as message queues or mailboxes. These approaches are known by developers and already used effectively, and only their internal semantics must be changed when adapted to inter-core communication.

The operation of the inter-core communication involves the following steps assuming we have a one-to-one communication scheme (there is a single sender and a single receiver):

- 1) The inter-core communication infrastructure is initialized. As most multicore MCUs use some kind of shared memory between cores for communication it involves allocation of the shared-memory, and buffers must be created on it for messages. In addition, inter-core interrupts are also initialized to let the receiving core know about the new information.
- 2) A sufficiently sized free memory buffer must requested before inter-core communication from the allocated shared memory by the sender.

- 3) When there is information to be sent to some another core, the information is written to the memory buffer by the sender core.
- 4) The sender core interrupts the receiving cores and continues processing.
- 5) The receiving core is informed by the interrupt, and the interrupt handler invokes the message handler that dispatches it to the relevant code processing the message on that core.
- 6) The information is copied to the memory area of the receiving core, and the memory buffer is returned to the free pool of buffers.

This process takes time. The sender core needs to set up operation and copy information in memory, and the speed of this operation depends on the speed of the core and memory. Then the interrupt is issued, and later, with some latency, the interrupt handler of the receiving node is invoked. Then the receiving core needs to set up and copy information to its memory area, which also depends on core and memory speed. The throughput (in bytes/s) and latency of this operation can be measured and computed assuming that both cores have read access to a common timer for time and latency measurements. Evaluating the performance for various clock speeds, shared-memory allocations, and if present, cache configurations for different message sizes a performance model of the MCU can be built, which makes possible the design and implementation of high-performance multi-core applications on the MCU with predictable time domain behaviour.

To reduce latency and CPU utilization, some experimental inter-core communication solutions use zero-copy operation [5], or Direct Memory Access (DMA, also called Data Movement Architecture), for example [6]. In this paper we do not consider evaluating these advanced solutions as most of the developers will use the default solution offered by the development system; therefore, they are interested in the performance of the standard solution. On the other hand, our performance evaluation approach can be used for these zero-copy or DMA-based solutions also, and comparable results can be collected.

## III. THE STM32H7X MULTI-CORE ARCHITECTURE

The STM32H7x family is a high-end, high-performance, network-enabled MCU utilizing the ARM Cortex-M7 processor core with a maximum clock speed of 480 MHz or 520 MHz depending on the actual part. The Cortex-M7 core has 16 kB ICACHE and 16 kB DCACHE to feed the processor with data at these high clock speeds. The ICACHE and DCACHE can be independently enabled or disabled; however, cache coherency must be guaranteed by configuring the Memory Protection Unit (MPU) to disable caching for sensitive memory areas, for example, accessed by DMA. Some members of the family are extended by a second ARM Cortex-M4 with a 240 MHz maximum clock speed, such as the STM32H745. We evaluated this MCU on a NUCLEO-H745ZI-Q development board.

The internal architecture of the dual-core version is depicted in Fig. 1, which shows the configurable power domains,

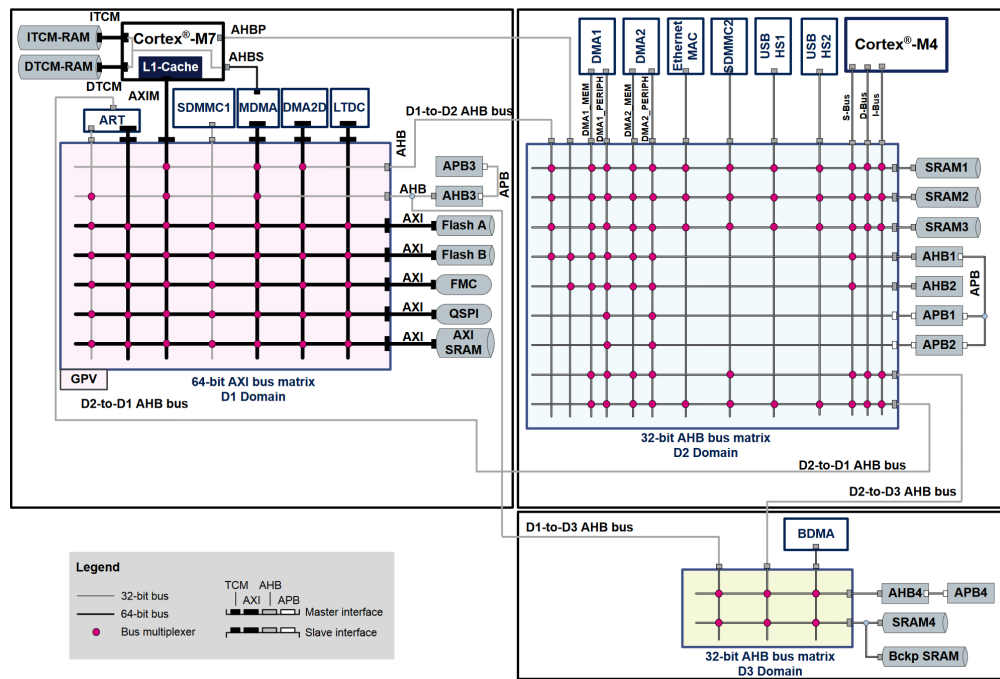


Fig. 1: STM32H7 dual-core system architecture [7]

memory assignment, and interconnect of the STM32H745. The detailed explanation of the internal functions can be found in [7]. Any SRAM memory available in the D1 (Cortex-M7), D2 (Cortex-M4), and D3 (auxiliary) power domains can be used by the processor core as shared memory for inter-core communication. Therefore, not only the CPU clock speeds, direction and data size of communication, and cache configuration, but also shared memory allocation influences inter-core communication performance for the dual-core STM32H7x MCUs.

#### IV. PERFORMANCE EVALUATION METHODOLOGY

Based on the above-mentioned facts, the factors influencing inter-core communication performance are the following:

- 1) Cortex-M7 (max. 480 MHz, setting 480, 240, 120, and 60 MHz) and Cortex-M4 (max. 240 MHz, settings 240, 120, and 60 MHz) clock speeds,
- 2) Cortex-M7 ICACHE and DCACHE configuration (enabled/disabled),
- 3) Location of the shared memory (D1, D2, or D3),
- 4) Direction of communication (Cortex-M7 to Cortex-M4, or Cortex-M4 to Cortex-M7),
- 5) Size of data to be sent to the other core in Bytes.

Furthermore, performance depends also on compiler configuration, from which the -O0 (non-optimized, v10\_O3 identifier), -O3 (maximum safe optimization, v8\_O3), -Os (O2, with some modification for size, - v12\_Osize identifier), and -Ofast (maximum optimization, v13\_Ofast identifier) are tried. There are other optimization options, for example, ones with debug code, these are captured by the vX part of the compiler

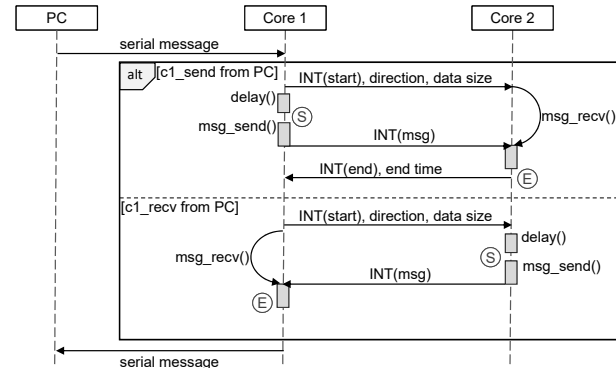


Fig. 2: Sequence diagram of the measurement software

configuration identifier; however, only measurements from the above-mentioned ones are presented here.

For all of these factors, the relevant values must be determined (e.g., clock frequencies applied during test, etc.), and after that individual performance tests must be executed for all combinations of values. The sequence diagram of an individual measurement is shown in Fig. 2.

The individual measurements are configured and execution is started from a PC over the serial port provided by the NUCLEO-H745ZI-Q development board. All individual measurements are repeated 1024 times, which number was determined by initial measurements with significantly bigger repetition times (65536) for achieving a 95% confidence interval. Time measurement is implemented by a hardware timer accessible by both cores, as one core starts the time measurement (beginning of send), and the other stops the time

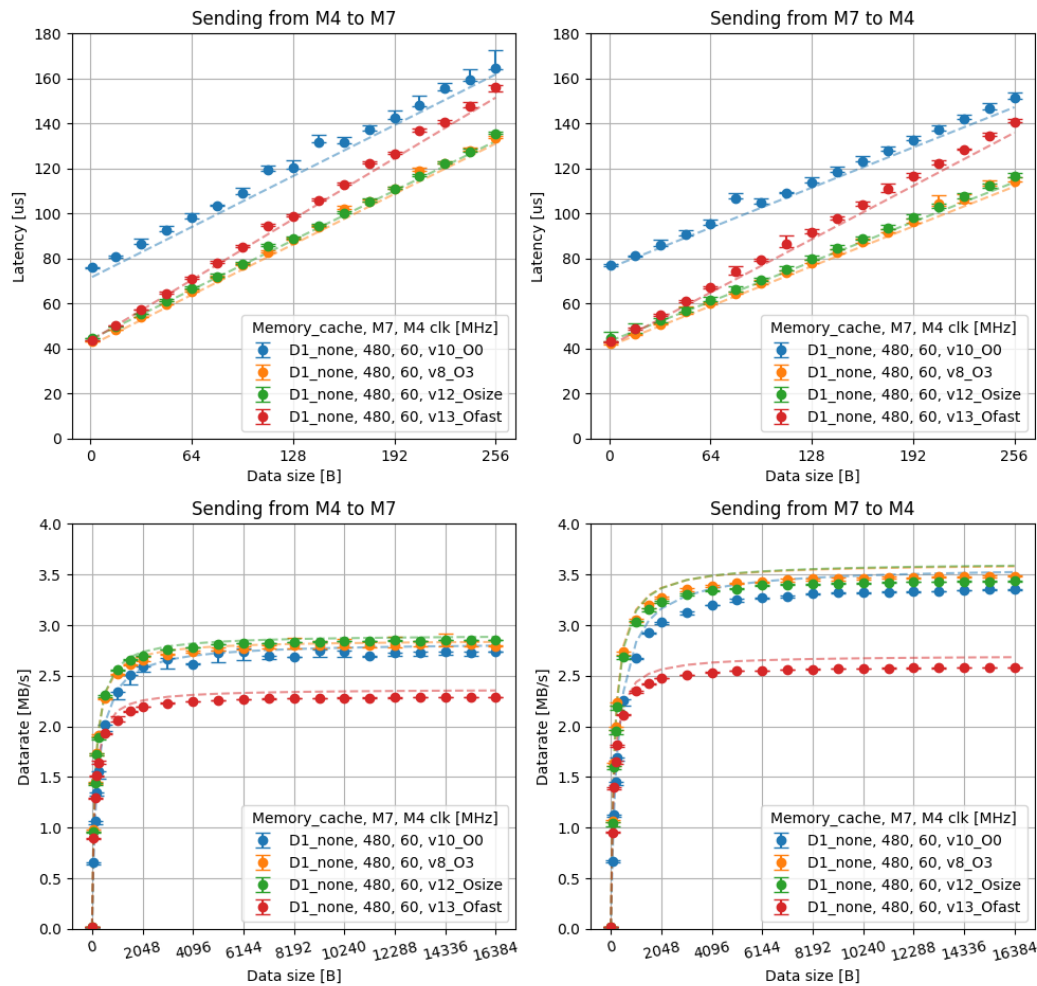


Fig. 3: Performance results as the function of compiler optimization settings

measurement (end of receive), so for delay measurement a common timer is needed. The timer is initialized by one core, the other core just reads the timer value based on a properly set timer initialization structure (for which the initialization function is never called).

Measurement results are also collected, processed, and visualized by the same PC. The measurement program is written in Python. A linear model is fitted on the measurement data using the Least-Squares method.

## V. PERFORMANCE EVALUATION RESULTS

The larger number of factors influencing performance makes it practically impossible to present the results in a concise form. Therefore, here we present only the results for compiler optimization, the clock speed selection, shared memory location, and cache setting.

Fig. 3. shows the performance changes caused by compiler optimization settings for specific clock speeds (Cortex-M7 480 MHz, Cortex-M6 60 MHz), for D1 memory with no caching (D1\_none). The results tell us that the -O3 or the -Os optimization settings provide the lowest latency independently

of the data size and communication direction, while -O0 is the worst. Data rate (throughput) does not depend on the optimization except the -Ofast setting, which produces the worst data rate contradicting its purpose (fast operation). All other settings produce similar results, so setting -O3 or -Os is advised.

Fig. 4. presents the performance data as the function of clock speeds and shared memory location with all caches disabled. Both latency and data rate are determined by the clock speed of the Cortex-M4 processors primarily in both directions, as the speed of the Cortex-M7 speed is substantially higher having a small contribution to the overall performance. The location of the shared memory determines performance similarly, the memory of the slow Cortex-M4 core (D3) is better to be used as shared memory. Likely because the slow Cortex-M4 core can access the D1 memory of the Cortex-M7 processor very slowly over the system interconnect, while it can handle its local memory (D3) somewhat efficiently. However, it is interesting to note that the D2 memory, located in a power domain other than any cores provides the best result (though marginally better than D3) overall. Selecting

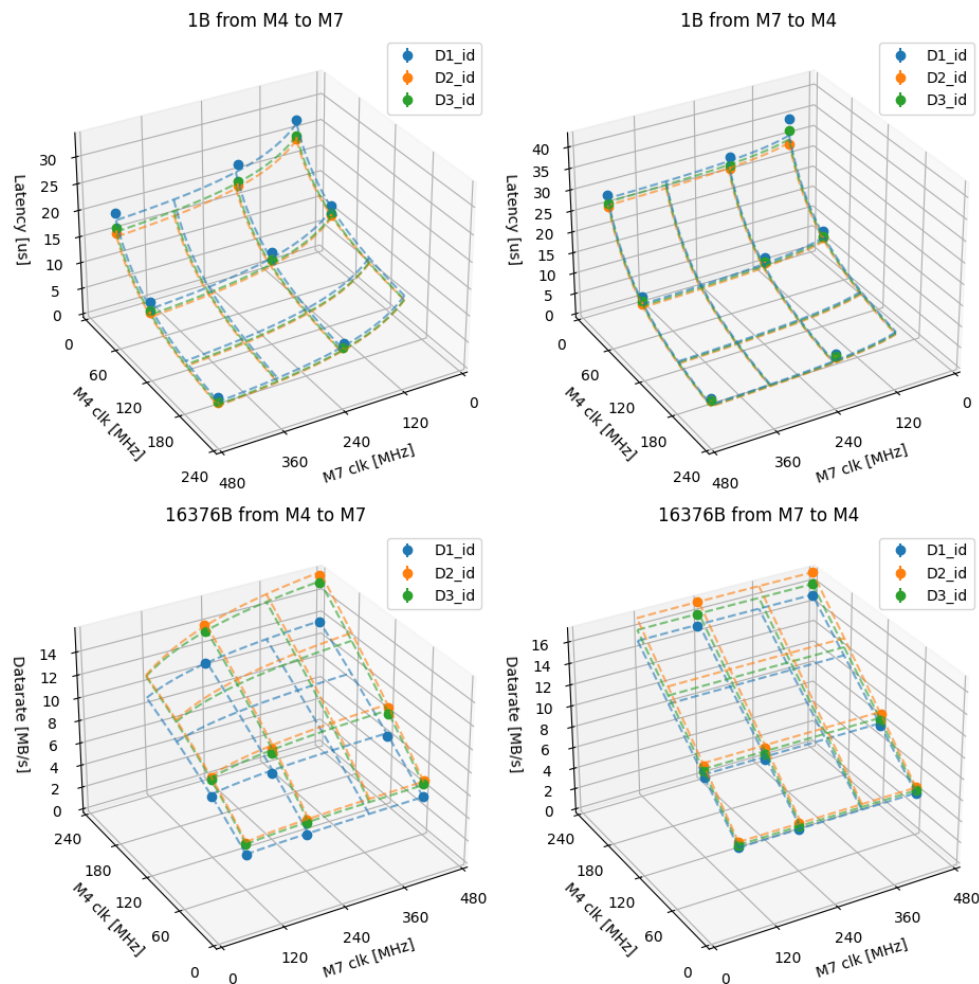


Fig. 4: Performance results as the function of clocks and shared memory locations. Actual measurement points depicted by circles and lines are predictions of the linear performance model fitted on the data.

this memory is also reasonable if D1 and/or D3 are powered down temporarily to reduce power consumption as D2 can be made always available with minimal power consumption, and inter-core interrupts can wake up the sleeping core after writing data into D2 reducing power consumption even further.

Fig. 4. proves also that the Least-Squares linear model can predict performance well, as the lines fit the actual measurement points with less than 5% relative error, which is close to the variability of the measurements.

Fig. 5. introduces the cache dependence of performance for a specific set of factors (both CPU cores run at 240 MHz, -O3 optimization, shared memory located in D1. Here we must note that caching of the shared memory is disabled by configuring the MPU to eliminate cache coherency problems, so the performance differences are due to other memory usage, not the use of the shared memory location. Configuring the data and/or the instruction cache on the Cortex-M4 improves latency, and the best results can be achieved by enabling both. Unfortunately, if the Cortex-M4 core sends to the Cortex-M7 data rate improves only marginally by enabling the caches. On

the other hand, the Cortex-M7 to Cortex-M4 direction gains some data rate improvement from the data cache, while the instruction cache helps very little.

## VI. CONCLUSIONS AND FUTURE WORK

Overall, modest optimization settings for the compiler (-O3 and -Os), setting high clock speed on the Cortex-M4 processor, the use of data and instruction caching on the Cortex-M7 processor, and the use of D2 shared memory are found to be offering the best performance on the STM32H745 dual-core MCU. Out of these findings, the optimum of using D2 memory for the inter-core communication shared memory is unexpected. As nearly all applications of these multi-core MCUs are driven by high-performance EDGE compute high clock speeds, enabled caches, and execution speed centric compiler optimizations are likely, only selecting shared memory location must be made based on performance data.

The developed measurement methodology and the linear performance model are found to be applicable, and its prototype implementation is started on the TI AM2424 quad-core

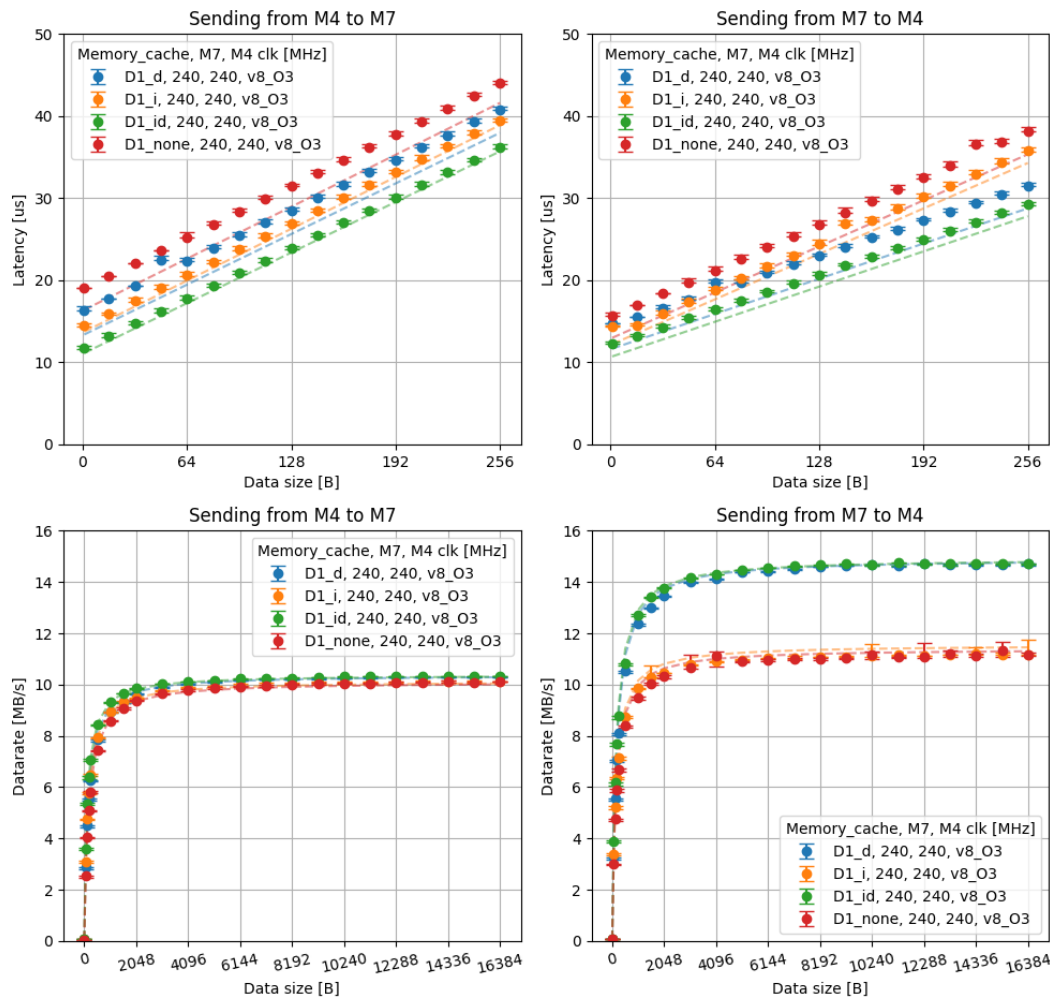


Fig. 5: Performance results as the function of cache configuration

MCU based on the inter-core communication framework available in its development system. It uses the rpmsg framework developed for inter-core communication on heterogeneous systems on chips and evaluated by us in [8] for the Beaglebone platform under Linux. Experimenting with other MCUs and comparing the results to the presented STM32H745 results can help also MCU platform selection.

The evaluated inter-core communication solution uses heavy memory copy operations on both the sender and receiver side, which simplifies programming but can drastically increase overhead. Similar problems were present in network protocol stacks, where the zero-copy approach or using DMA to move data between memory locations in the system were found to reduce processor resource utilization significantly, so their implementation may improve performance. If such solutions appear in the market, our performance evaluation methodology may be used to prove their advantages.

## REFERENCES

- [1] S. Siddha, V. Pallipadi, and A. Mallick, "Process scheduling challenges in the era of multi-core processors." *Intel Technology Journal*, vol. 11, no. 4, 2007.
- [2] Infineon introduces microcontroller multicore architecture for automotive applications. [Online]. Available: <https://www.infineon.com/cms/en/about-infineon/press-market-news/2011/INFATV201110-003.html>
- [3] I. Senoussaoui, "Processor and memory co-scheduling of embedded real-time applications on multicore platforms," Ph.D. dissertation, Université de Lille; Université d'Oran 1 Ahmed Ben Bella, 2023.
- [4] N. Singh, K. Renganathan, C. Rebeiro, J. Jose, and R. Mader, "Kryptonite: Worst-case program interference estimation on multi-core embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1–23, 2023.
- [5] F. O'Carroll, H. Tezuka, A. Hori, and Y. Ishikawa, "The design and implementation of zero copy mpi using commodity hardware with a high performance network," in *Proceedings of the 12th international conference on Supercomputing*, 1998, pp. 243–250.
- [6] P. Pazzaglia, D. Casini, A. Biondi, and M. D. Natale, "Optimizing inter-core communications under the LET paradigm using DMA engines," vol. 72, no. 1, 2023, pp. 127–139.
- [7] STMicroelectronics. (2022) An5557: Stm32h745/755 and stm32h747/757 lines dual-core architecture. [Online]. Available: [https://www.st.com/resource/en/application\\_note/an5557-stm32h745755-and-stm32h747757-lines-dualcore-architecture-stmicroelectronics.pdf](https://www.st.com/resource/en/application_note/an5557-stm32h745755-and-stm32h747757-lines-dualcore-architecture-stmicroelectronics.pdf)
- [8] T. Kovácskásy and G. Fekete, "Application experiment with the standard linux services for asymmetric multiprocessing on heterogeneous system on a chips," in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2022, pp. 1–6.