Joint Computation and Communication Analysis of Hard Real-Time Applications in Manycores

Anderson R. P. Domingues, Lucas Damo, Sergio Johann Filho, Fernando Gehm Moraes
School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
{anderson.domingues, lucas.damo}@edu.pucrs.brs, {sergio.filho, fernando.moraes}@pucrs.br

***Abstract*—Manycores appeared as a scalable computing infrastructure to support massively parallel applications. Due to the advancements of on-chip technology, manycores present an attractive trade-off between power consumption, area, and fabrication cost. Due to the inherent distributed computing model of manycores, predicting the behavior of modern, mission-critical applications becomes challenging. On the one hand, the community introduced real-time (RT) networks-on-chips (NoC) to support the RT analysis of communication in manycores. On the other hand, implementing non-functional requirements (NFRs) adds complexity to the design, making RT-NoCs prohibitively in resource-constrained scenarios. This work presents a computation and communication approach to RT analysis in manycores. Our approach suits systems without RT-NoCs, enabling manycores equipped with simple low-cost NoCs to run RT applications. We show a proof-of-concept where we can reduce the frequency of the baseline manycore to fit the needs of the target application. Results demonstrate that our approach guarantees RT without modifying the baseline hardware.**

***Index Terms*—Network-on-Chip, Real-Time, Manycores.**

I. Introduction

NoCs are attractive options for system interconnection in manycores due to their potential for massively parallel communication and scalability while inserting low overhead for area and energy consumption into the design. NoC routers provide the communication infrastructure in a typical NoC, allowing multiple data streams to traverse the system simultaneously [1]–[3]. Some NoC designs acknowledge the implementation of NFR, e.g., security, safety [4], [5], and real-time. The main concern of RT-NoCs is predictability — a particularly important property in control-theoretic, cyber-physical, and robotics domains due to the need for compliance with stringent safety constraints [6]–[8].

Designing hardware that accounts for multiple NFRs is challenging due to the implementation of conflicting NFRs. For instance, implementing a security feature may modify the timing of the system. Similarly, predictable hardware may add energy and area overheads, due to additional circuitry. The lack of specialized literature aggravates the problem, as the techniques often manage NFRs as isolated. Also, studies on multiple NFRs, e.g., power and real-time [9] propose techniques on non-publicly available platforms, making it unfeasible to replicate the techniques on industrial-grade projects.

Concerning RT, the analysis of applications on NoC-based, private memory, and multitasking manycores must consider both the individual processing elements (PEs) and the underlying NoC. The analysis of PEs regards the CPU and I/O peripherals, while RT-NoCs partake in the RT guarantees for communication. Recent studies on RT-NoCs [10]–[12] do not approach computation, missing mechanisms for synchronizing processing and communication operations. Considering commercial off-the-shelf (COTS) platforms, whose hardware cannot be changed, is it possible to run a given application and guarantee RT for tasks and traffic? Assuming platforms where we can change the nominal frequency, what is the minimum frequency to address the RT requirements of an application? How can we address RT without tampering with other NFRs?

This paper addresses RT analysis on NoC-based, multitasking, private memory manycores whose hardware cannot be changed, e.g., COTS platforms, targeting manycores equipped with conventional, non-RT NoCs. As these platforms cannot take advantage of specific design-time optimizations [13], [14], we propose an off-line, *pre-runtime* framework for computation and communication RT analysis that can:

1) determine if a given application meet its RT requirements, considering the architectural features and nominal operating frequency of the platform;
2) find the minimum frequency to address an RT application in platforms whose frequency can be changed.

At the computation scope, we combine a discrete-event simulation (DES) strategy with the critical-path analysis (CPA). The goal is to validate the CPU requirements for tasks at the PE level and the end-to-end application execution time at a system-wide level. We adopt a contention-free model for communication, assuming a pre-runtime optimization method. We assume the zero-load latency model of the NoC as a performance boundary. This assumption enables us to perform our analysis on *virtually any NoC*, adding RT support to systems without RT-NoCs. Our approach is *novel* because it supports COTS, is fully automated, and uses only open-source tools. We validate our approach in an open-hardware platform, favoring the reproducibility of our study.

II. Baseline Platform

Figure 1 presents the baseline platform, that uses a publicly available 2D-mesh NoC [15] (32-bit flit-width, credit-based, wormhole, XY routing, 8-flit input buffer depth). The manycore contains identical PEs, with a RISC-V (RV32E) processor, two scratchpad memories, a memory-mapped registers unit, a peripheral management module (counters and configurable interrupts), and a DMA-like, interleaving-based network interface (NI). An address translation unit (ADT) provides access to the different memory zones of the CPU (RAM, ROM, and peripherals). We monitor the manycore at the register-transfer level (RTL) through SystemVerilog interfaces, adding no performance overhead to the hardware at the simulation level.

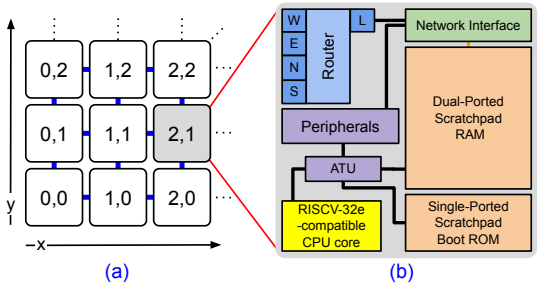

Fig. 1. The 2D-mesh interconnection of the manycore (left) and the magnification of a PEs (right). The NI is connected to the local port of the router, and the remainder ports (SNWE) connects to neighbor PEs.

Similarly to other Network Interfaces (NI) with direct memory access (DMA) features [16], [17], our NI allows the PE to send and receive data asynchronously without CPU intervention. Once a packet is entirely received, the NI interrupts the CPU. Sending a packet follows a similar strategy, where the network device driver configures the NI to copy the packet from the RAM into the local port of the router. At the kernel level, the network driver has a queue of packets, where the NI stores packets, dropping packets if the queue is full. Tasks can probe the driver for packets or perform a self-blocking operation until a packet is received. The driver also allows tasks in the same PE to communicate by moving packets between queues.

PEs run the UCX-OS [18] kernel, which we modify the task scheduler to implement the priority-based round-robin (PBRR) technique [19], accounting for task dependency. The kernel keeps track of the number of iterations of tasks as new packets arrive, prioritizing (i) tasks in the oldest iterations and (ii) tasks meeting their data dependencies in case of a tie.

III. A Framework for Pre-runtime RT Analysis

We propose a framework to identify whether a NoC-based manycore can run a real-time application without deadline violations. We implemented a couple of tools to automate our approach, which relies on the *scheduling* of system resources — computation and communication. Scheduling tasks on a single-core CPU is not novel, as some existing solutions date back to the 80s [20]. However, incorporating the I/O components of the system, managing interruptions, and addressing the needs of NoCs requires additional modeling.

Our framework concerns dataflow applications, where tasks have data dependency on sensors, the network, data offloading, or other tasks. In addition, tasks must employ a variation of the predictable execution (PREM) [21] and the logical execution time (LET) [22] models; we divide tasks in *receiving*, *processing*, and *sending* phases as the controlled execution of tasks is key for I/O access predictability. We expect applications to have real-time requirements on (i) the iteration time and (ii) the number of iterations per second. The former is the end-to-end application execution time, which is particularly important in sensor applications as it dictates the accuracy of sensors in time, i.e., how old the reading is. The latter is the application execution rate, which relates to the periodic execution of tasks.

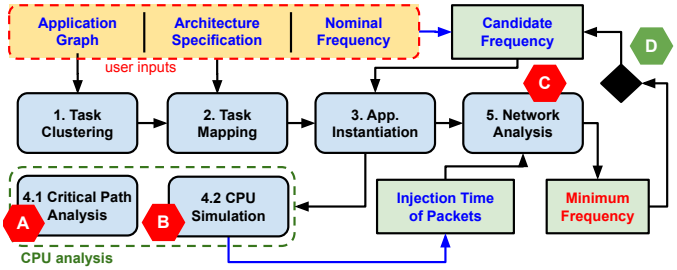

Fig. 2. A workflow depicting our approach. User inputs appear at the top. Activities appear as blue shapes. Outcomes appear as green shapes. Tags A, B, and C represent activities of the workflow that may fail due to the unavailability of system resources. Tag D represents the end of the workflow, where users can choose to abandon the workflow or further optimize the result.

The proposed framework (Figure 2) takes three inputs: (i) application graph, (ii) architecture specification, and (iii) nominal manycore frequency.

i) We described applications using a directed, potentially cyclic, graph $G = (E, V)$, where V is the set of vertices and E is the set of edges. Vertices represent tasks, to which we tag the corresponding worst-case execution time (WCET), in cycles. Edges represent flows of data, i.e., a set of packets with periodic behavior, given by a 3-tuple $f = \langle p, c, d \rangle$, corresponding to the period (p), capacity (c), and deadline of flows (d), in clock cycles.
ii) The architecture specification must describe a zero-load latency model of the network and the set of allowed operating frequencies of the manycore.
iii) The framework considers the nominal frequency as the start point for the frequency optimization.

At the first iteration, the framework computes whether the nominal frequency suits the application. The framework runs in interactive or automated modes. In interactive mode, users must decide to increase the frequency (if the platform allows), try a lower frequency, or abandon the process at each iteration. In automated mode, the framework runs until it reaches a given number of iterations or cannot optimize the frequency further, assuming a tolerance factor. The framework evaluates the application at each iteration, using a 5-step analysis: (i) task clustering, (ii) task mapping, (iii) application instantiation, (iv) CPU analysis, and (v) network analysis. CPU analysis splits in critical path analysis (CPA) and CPU simulation.

A. Task Set Clustering and Mapping

Task set clustering consists of splitting an application graph into multiple task sets to be assigned by the framework to a PE during the mapping step (Figure 2, step 1). The goal is to generate one cluster per PE. We created an algorithm to cluster graphs that performs $O(n)$ on the number of vertices of the application task graph. The algorithm iteratively removes edges from the graph, combining vertices until the resulting graph has $|V| \leq |PEs|$, based on one of two criteria. The MIN-COMM criterion reduces the overall network load, eliminating edges with the most communication load groups communication-intensive tasks in the same cluster. Oppositely, the MAX-PROC criterion balances the CPU usage between PEs by grouping tasks with the least CPU load. Once the algorithm computes the clusters, we can map them to the manycore using

a task mapping strategy (Figure 2, step 2). Note that task mapping is a well-developed topic in the literature [23], so discussing task mapping is out of the scope of this paper.

B. Application Instantiation and Framework Loop

As the framework explores different frequencies during its execution, it must adjust the application parameters to match the frequency at each iteration (Figure 2, step 3). The framework carries the WCET value of tasks along the multiple iterations, as the CPU architecture does not change. However, the framework must consider the periodicity of flows, adjusting the period and deadline of flows accordingly. Iterations occur as follows. At the first iteration, the framework takes the nominal frequency as the candidate frequency, employing no adjustments to the application. Based on the results of CPU analysis (Figure 2, step 4) and network analysis (Figure 2, step 5), the framework either increases or decreases the frequency. If the framework identifies violations of deadlines (tasks or flows), the candidate frequency will increase at the next iteration. Similarly, the framework reduces the frequency if it detects no deadline violations. Searching for the minimum feasible frequency performs $O(log\ n)$, i.e., the time complexity of the binary search, where n is the number of frequencies supported by the target platform (discrete).

C. CPU Analysis - Time Boundary and Utilization

The CPU analysis is a 2-step process to assert whether the manycore has the necessary resources to run an application from the computation perpective.

1) Critical Path Analysis (CPA): The framework uses the CPA method [24] to evaluate the end-to-end processing time of an application iteration (Figure 2, step 4.1). As PEs carry only subsets of the application task set, we must account for internal communication (task-to-task communication through memory spaces) and external communication (NoC). The CPA method proceeds as Djkistra's algorithm for computing the shortest path between two vertices [25], except that we multiply the weights of edges (communication load) by -1. Although we cannot apply topological sort to cyclic graphs, we use depth-first search (DFS) to find and remove cycles, adding the weight of a removed edge to its origin vertex. The result of this step is the number of cycles that the application takes to execute, i.e., the iteration time. The CPU must have enough cycles to run the application. Otherwise, the framework discards the candidate frequency and moves on to the next iteration.

2) Discrete-Event Simulation (DES) of PEs: The execution time of some kernel routines, e.g., dynamic memory allocation (*malloc*) and interruption handling, is hard to predict. Our framework uses a DES engine to simulate PEs while assuming a worst-case characterization of kernel operations (Figure 2, step 4.2). It simulates PEs individually to find whether they can run the assigned task cluster. The simulation considers the WCET of the task scheduler, I/O interrupts, and other minor system-specific programmable interrupts. If the framework detects that one of the PEs cannot run the assigned task set, the framework discards the candidate frequency and moves to the next iteration. Due to the employment of the PREM and LET models, the framework can estimate the time in which tasks inject packets into the network. Packets enter the network only at the *sending* phase, triggering an I/O event in the kernel. The framework uses these values during the network analysis step (Figure 2, step 5) to predict the behavior of packets.

D. Network Analysis

The network analysis occurs after the CPU analysis, considering that the framework kept the candidate frequency. In this step, the framework asserts whether the network flows can traverse the NoC without deadline violations. Our flow model reassembles the Job-Shop model [26], where we replace machines with links (channels) and jobs with flits (data units). The goal of the network analysis step is to find a schedule for packets. The network analysis has three steps:

1) Unwrap flows to packets $p = \langle mrt, size, ad \rangle$, having a minimum release time (mrt), data size ($size$), and absolute deadline (ad) each.
2) Discover the path of packets in the NoC to compute the occupancy of links (L), i.e., a relation $O : P \times L \times T$, matching packets to the links they occupy in the discrete time domain (cycles), where $\forall p \in P$.
3) Find a schedule where the following constraints hold:
   - (C1) $prt \geq mrt$, where ptr is the release time of packets in the found schedule;
   - (C2) $ad \geq prt + size/lw$, where lw is the link width;
   - (C3) packets cannot overlap (single channel constraint);
   - (C4) flits of a same packet must enter and leave the network at the same time (wormhole constraint).

Instead of generating a network schedule, which is NP-complete (similarly to job-shop), we use the prt values collected during the CPU analysis step, reducing the network analysis step to a decision problem. The framework indicates whether the schedule, considering C1-C4, is feasible or not. If the schedule is feasible, the framework *memorizes* the minimum frequency found so far. Then, the framework can either try to find a lower frequency or abandon the process.

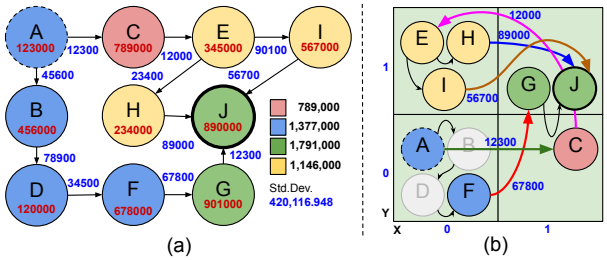

Fig. 3. The clustered SAE application (MAX-PROC), where squared shapes displays the CPU load of clusters, in cycles (a); and clusters mapped onto a 2×2 manycore (b). Communication in the same clusters occurs through software, while extra-cluster communication occurs through the NoC.

IV. Proof of Concept and Evaluation

We demonstrate our approach using a synthetic application whose minimum frequency to meet its RT requirements is known (500MHz). We address the application as "SAE" for the remainder of the paper. Figure 3 (a) shows the application graph of the SAE application for a 2×2 instance of our manycore. We feed the application graph of SAE to our clustering algorithm using the MAX-PROC criterion and $n = 4$. Then, we assigned the resulting clusters to PEs, as shown in

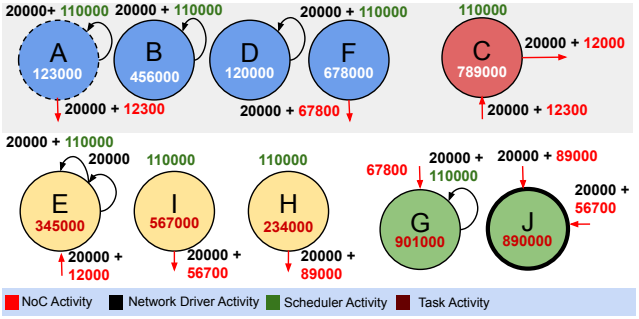

Fig. 4. End-to-end execution time (CPA method, cycles). Colors denote the different clusters. Red arrows indicate packets transferred through the NoC. Black arrows denote inter-cluster communication.

Figure 3(b). The nominal frequency of our platform is 2.5GHz, which is also given as input to our framework. By the last iteration, our framework found 500.4MHz as the minimum frequency to meet the RT constraints on the platform. Since we cannot display the results for all iterations, we couple to show the last step and demonstrate how the found frequency meets the requirement of the application (the last frequency).

The CPA method (Figure 4) estimated the end-to-end iteration execution time of SAE based on the application graph and the WCET of the kernel operations. Our scheduler has a WCET of 110 kcycles, and our network driver has a WCET of 20 kcycles. Tasks are scheduled once per iteration due to our PBRR implementation. The total iteration execution time, as pointed out by the CPA method, is 8.464.895 cycles. The actual RTL execution is 6.799.200 cycles (80%). Table I shows the actual cluster execution time (Execution, ModelSim simulator), the CPA estimation (Estimation), and the results from DES (Usage). The CPA method overestimates $\simeq 26\%$ of the actual CPU usage. However, the overestimation is *acceptable* ($< 30\%$) as the actual CPU usage stays below 70%; a CPU usage over 70% is often considered questionable [27].

TABLE I
COMPARISON OF EXECUTION TIME (CPU).

Cluster	Execution[1]	Estimation[2]	Difference	Diff. (%)[3]	Usage[4]
ABDF	1,439,000	1,819,300	+380,300	+26.428%	72.722%
C	801,300	963,300	+162.000	+20.217%	38.532%
EHI	2,174,500	2,667,895	+493,395	+22.690%	71.616%
GJ	2,384,400	3,014,400	+630.000	+26.421%	86.980%

[1]Execution time (RTL, cycles); [2]CPA estimation; [3]Execution time is 100%; [4]DES.

The framework captured the time that tasks finished during the DES simulation. As tasks implement the PREM and LET models, we assume that packets are injected at the same time they leave the CPU. Table II shows the characterization of flows, where ph is the phase time (the period of scheduling interrupts), pv is the *phase* of the task, and $\psi = (ph \times pv)$. A phase corresponds to the alignment of application execution to the scheduler tick. For instance, the SAE application cannot finish before the 4^{th} phase due to the critical path traverses 4 tasks in the application graph. This is a necessary step, as the DES simulation does not account for task dependency. Deadlines match the end of the phase of the receiving task.

In Figure 5, we present the RTL simulation of our manycore running the SAE application at the minimum frequency of 500.4MHz, obtained from our framework. The goal is to

TABLE II
CHARACTERIZATION OF FLOWS FOR THE SAE APPLICATION.

Flow	MRT[1]	S	D	Volume	Deadline
F_1	$285300 + \psi$	A	C	12300	$110000 + \psi$
F_2	$1819300 + \psi$	F	G	67800	$110000 + \psi$
F_3	$963300 + \psi$	C	E	12000	$110000 + \psi$
F_4	$1280700 + \psi$	I	J	56700	$1208800 + \psi$
F_5	$1733700 + \psi$	H	J	89000	$1208800 + \psi$

(S) source task, (D) destination task, [1]Minimum release time.

demonstrate that the application meets its RT requirements. The hyperperiod of SAE is 20ms, corresponding to 4 phases of 5ms each, meeting the expected iteration time and iterations per second requirements. SAE receives stimuli from outside the system once per 5ms, triggering 4 instances of the application per $20ms$ (Task A). SAE returns results to outside the system via Task J. The 3rd instance of the application (pink) has additional 1.457ms in its iteration time, related to the scheduling of Task I; the task executes two iterations at once due to two packets received in the phase time, delaying Task J. Nevertheless, Task J meets its deadlines for all instances of SAE, for an iteration time of 16,820ms for the 1st, 2nd and 3th instances, and 18.277ms for the 3rd instance.

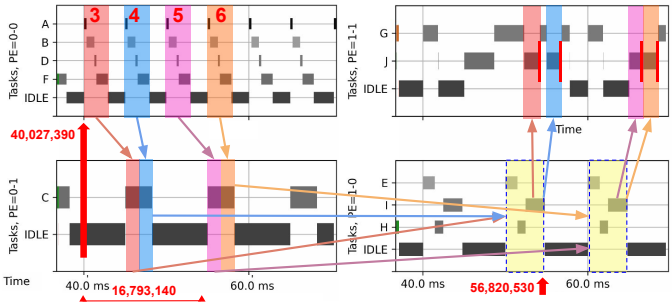

Fig. 5. Interaction of tasks of the SAE application during RTL simulation. Rectangles with the same color represent tasks in the same iteration (instances). Arrows indicate the direction of the communication. The yellow rectangle represents a phase overlapping.

V. DISCUSSION, CONCLUSION AND FUTURE WORKS

This paper proposes a framework for asserting the RT properties of applications running on NoC-based, multitasking, private-memory manycores. Our framework observes the computation and communication aspects of the application, adjusting the frequency of the target platform to the minimum while allowing the application to meet its RT requirements. We demonstrate our framework on the SAE application, running on our manycore. Our framework could reduce the manycore frequency without compromising the RT requirements of SAE.

Our framework enables the exploration of RT at the pre-runtime, alleviating the effects of stacked NFRs in the design. We avoid modeling design-specific features in our framework, e.g., buffer, allowing our framework to suit virtually any NoC that provides a zero-load latency model. Future works include (i) evaluating our approach while considering multiple NFRs, e.g. area and energy requirements; (ii) experiment with more complex applications [28]; (iii) make use of dynamic voltage-frequency scaling (DVFS) to improve energy efficiency without compromising real-time performance, and (iv) improve the automation of the approach, e.g., integration with ModelSim.

ACKNOWLEDGMENTS

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001; Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), 309605/2020-2 and 407829/2022-9; Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), 21/2551-0002047-4 and 23/2551-0002200-1.

REFERENCES

[1] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2002, pp. 117–124, https://doi.org/10.1109/ISVLSI.2002.1016885.

[2] L. Benini and G. De Micheli, "Networks on chip: a new paradigm for systems on chip design," in *Design, Automation Test in Europe Conference (DATE)*, 2002, pp. 418–419, https://doi.org/10.1109/DATE.2002.998307.

[3] A. Jantsch, R. Lauter, and A. Vitkowski, "Power analysis of link level and end-to-end data protection in networks on chip," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2005, pp. 1770–1773, https://doi.org/10.1109/ISCAS.2005.1464951.

[4] S. Charles and P. Mishra, "Securing Network-on-Chip Using Incremental Cryptography," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 168–175, https://doi.org/10.1109/ISVLSI49217.2020.00039.

[5] P. Guo, W. Hou, L. Guo, Z. Cao, and Z. Ning, "Potential Threats and Possible Countermeasures for Photonic Network-on-Chip," *IEEE Communications Magazine*, vol. 58, no. 9, pp. 48–53, 2020, https://doi.org/10.1109/MCOM.001.2000029.

[6] AUTOSAR, "Standards – AUTOSAR," 2021, online. Visited in February 2021. https://www.autosar.org/standards/.

[7] ISO, "ISO 26262:9 — Automative Safety Integrity Level (ASIL)," 2020, online. Visited in February 2021. https://www.iso.org/obp/ui/#iso:std:iso:26262:-9:ed-2:v1:en.

[8] VDA, "Verband der Automobilindustrie e.V. (VDA) – Automotive SPICE," 2024, online. Visited in march 2024. http://www.automotivespice.com/.

[9] X. Li, Z. Li, Y. Ju, X. Zhang, R. Wang, and W. Zhou, "Cop: A combinational optimization power budgeting method for manycore systems in dark silicon," *IEEE Transactions on Computers*, vol. 72, no. 5, pp. 1356–1370, 2023.

[10] S. Deniziak and R. Tomaszewski, "Codesign of energy and resource efficient contention-free Network-on Chip for real-time embedded systems," in *International Workshop on Network on Chip Architectures (NoCArc)*, 2018, pp. 1–6, https://doi.org/10.1109/NOCARC.2018.8541199.

[11] Y. Chen, E. Matus, S. Moriam, and G. P. Fettweis, "High performance dynamic resource allocation for guaranteed service in network-on-chips," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 503–516, 2020.

[12] T. Picornell, J. Flich, C. Hernández, and J. Duato, "DCFNoC: A Delayed Conflict-Free Time Division Multiplexing Network on Chip," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6, https://doi.org/10.1145/3316781.3317794.

[13] O. Peckham, "Esperanto Unveils ML Chip with Nearly 1,100 RISC-V Cores," 2020, https://www.hpcwire.com/2020/12/08/esperanto-unveils-ml-chip-with-nearly-1100-risc-v-cores.

[14] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Bass, "A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array," in *IEEE Symposium on VLSI Circuits (VLSIC)*, 2016, pp. 1–2, https://doi.org/10.1109/vlsic.2016.7573511.

[15] F. G. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration*, vol. 38, no. 1, pp. 69–93, 2004, https://doi.org/10.1016/j.vlsi.2004.03.003.

[16] M. Ruaro, L. L. Caimi, V. Fochi, and F. G. Moraes, "Memphis: a framework for heterogeneous many-core SoCs generation and validation," *Design Automation for Embedded Systems*, vol. 23, no. 4, pp. 103–122, 2019, https://doi.org/10.1007/s10617-019-09223-4.

[17] M. Ruaro, F. Lazzarotto, C. Marcon, and F. G. Moraes, "DMNI: A specialized network interface for NoC-based MPSoCs," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 1202–1205, https://doi.org/10.1109/ISCAS.2016.7527462.

[18] S. Johann, "UCX/OS - Microcontroller Executive / OS," 2024, online. Visited in January 2024. https://github.com/sjohann81/ucx-os.

[19] N. Nithya and S. Itapu, "Design Of Low Area Interconnect Architecture for CPU-GPU Network-On-Chips (NoCs)," in *IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, 2023, pp. 1–5, https://doi.org/10.1109/CONECCT57959.2023.10234778.

[20] D. Gusfield, "Bounds for naive multiple machine scheduling with release times and deadlines," *Journal of Algorithms*, vol. 5, no. 1, pp. 1–6, 1984. [Online]. Available: https://www.sciencedirect.com/science/article/pii/019667748490035X

[21] I. Senoussaoui, M. K. Benhaoua, H.-E. Zahaf, and G. Lipari, "Toward memory-centric scheduling for PREM task on multicore platforms, when processor assignments are specified," in *International Conference on Embedded & Distributed Systems (EDiS)*, 2022, pp. 11–15, https://doi.org/10.1109/EDiS57230.2022.9996534.

[22] K.-B. Gemlau, L. KÖHLER, R. Ernst, and S. Quinton, "System-level Logical Execution Time: Augmenting the Logical Execution Time Paradigm for Distributed Real-time Automotive Software," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, pp. 1–27, 2021, https://doi.org/10.1145/3381847.

[23] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *ACM/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–10, https://doi.org/10.1145/2463209.2488734.

[24] J. E. Kelley and M. R. Walker, "Critical-path planning and scheduling," in *Eastern Joint IRE-AIEE-ACM Computer Conference*, 1959, https://doi.org/10.1145/1460299.1460318.

[25] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959, http://eudml.org/doc/131436.

[26] D. S. J. Michael R. Garey, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1st ed. Freeman, 1979.

[27] P. A. Laplante and S. J. Ovaska, *Real-Time Operating Systems*, 4th ed. Wiley-IEEE Press, 2011.

[28] Z. Shi, A. Burns, and L. Indrusiak, "Schedulability Analysis for Real Time On-Chip Communication with Wormhole Switching," *International Journal of Embedded and Real-Time Communication Systems*, vol. 1, pp. 1–22, 2010, https://doi.org/10.4018/jertcs.2010040101.