ISSN: 2395-1680 (ONLINE)
DOI: 10.21917/ijme.2023.0269

DESIGN AND OPTIMIZATION OF HARDWARE-SOFTWARE CO-DESIGN FOR REAL-TIME EMBEDDED SYSTEMS

S. Hemalatha[1], R. Yalini[2], M.M. Arun Prasath[3] and Sunil Kumar Yadav[4]

[1]*Department of Electrical and Electronics Engineering, St. Joseph's Institute of Technology, India*
[2]*Department of Electrical and Electronics Engineering, Jayam College of Engineering and Technology, India*
[3]*Department of Electronics and Communications Engineering, K S R Institute for Engineering and Technology, India*
[4]*Department of Computer Engineering, Dr. D.Y. Patil College of Engineering and Innovation, India*

Abstract

In recent years, quantum cell automata (QCA) has emerged as a promising optimization technique for real-time embedded systems design. QCA leverages quantum computing principles to perform parallel and coherent computations, enabling efficient exploration of vast solution spaces. This paper explores the application of QCA in the hardware-software co-design of real-time embedded systems, aiming to achieve enhanced performance, reduced power consumption, and improved reliability. The study investigates the partitioning of tasks between hardware and software, the design of hardware accelerators, and the development of efficient algorithms for real-time constraints. Furthermore, the paper addresses the optimization of communication interfaces to facilitate seamless interaction between hardware and software components. The results demonstrate the potential of QCA in revolutionizing the design and optimization of real-time embedded systems through its ability to exploit quantum parallelism and coherence.

Keywords:

Quantum Cell Automata, Hardware-Software Co-design, Optimization, Real-time Embedded Systems

1. INTRODUCTION

Real-time embedded systems are pervasive in modern technology, spanning a diverse array of applications such as automotive control, aerospace systems, industrial automation, and healthcare devices. These systems demand high levels of performance, low power consumption, and robust reliability to meet the stringent requirements of their real-time constraints. Achieving such optimization targets in hardware-software co-design is a formidable task, requiring the careful integration of specialized hardware components and efficient software algorithms [1]. Traditional optimization techniques often struggle to cope with the complexity of real-time embedded systems, hindering the realization of their full potential. Conventional hardware-software co-design approaches encounter challenges in striking the right balance between performance and power consumption, leading to compromises that may not exploit the system capabilities [2]. As embedded systems continue to advance in sophistication, the design space expands exponentially, rendering conventional methods impractical for exploring the vast solution spaces. Confronted with these challenges, researchers and engineers are constantly seeking innovative approaches to unleash the true potential of real-time embedded systems [3]. Quantum cell automata (QCA) has emerged as an intriguing optimization technique that draws inspiration from quantum computing principles. By harnessing quantum parallelism and coherence, QCA exhibits remarkable computational efficiency, paving the way for a hardware-software co-design [4]. This paper aims to explore the application of quantum cell automata in the design and optimization of real-time embedded systems. By leveraging QCA, we intend to address the prevailing challenges in this domain, including performance enhancement, power consumption reduction, and improved reliability. Through a comprehensive investigation of QCA-based hardware-software co-design, we seek to bridge the gap between traditional methods and the transformative potential of quantum computing principles.

Despite the immense promise of QCA, its integration into real-time embedded systems presents its own set of challenges and problems. Ensuring the seamless interaction between hardware and software components while accommodating the peculiarities of quantum computing represents a non-trivial task. Moreover, the development of algorithms and communication interfaces optimized for QCA requires careful consideration and novel approaches.

2. QCA

Quantum cell automata (QCA) is a novel computational paradigm inspired by the principles of quantum mechanics. It allows for the representation and processing of information using quantum states and quantum operations. In QCA, information is encoded in quantum bits, known as qubits, which can exist in multiple states simultaneously due to the phenomenon of superposition. In classical cell automata, information is processed using simple rules applied to discrete cells in a grid. In QCA, the analog of these cells is represented by quantum states, and the evolution of the system is governed by quantum operations.

2.1 QUANTUM STATE REPRESENTATION

In a one-dimensional QCA, we can represent the state of a single cell at a particular time step as a quantum superposition of two basis states, typically denoted as $|0\rangle$ and $|1\rangle$. These states correspond to the logical 0 and 1 states of a classical bit. The state of the qubit can be represented as:

$$|\psi(t)\rangle = \alpha(t)|0\rangle \beta(t)|1\rangle \quad (1)$$

where $\alpha(t)$ and $\beta(t)$ are complex probability amplitudes that determine the probability of finding the qubit in state $|0\rangle$ or $|1\rangle$, respectively.

2.2 QUANTUM EVOLUTION

The evolution of the quantum state in QCA is governed by a unitary quantum operation, often represented as a quantum gate. The quantum gate acts on the qubits and transforms their states

coherently, preserving the normalization and reversibility of the quantum system. A common example of a single-qubit quantum gate is the Hadamard gate (H), which creates a superposition from a basis state:

$$H|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \quad (2)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \quad (3)$$

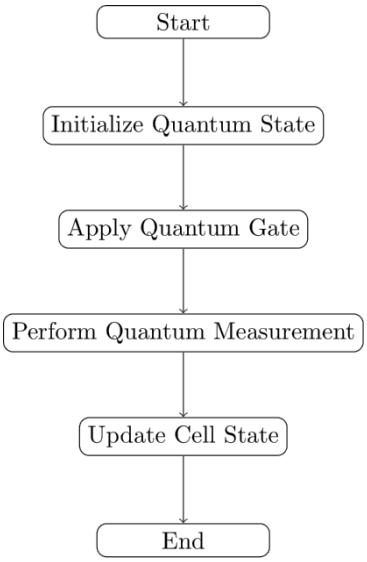

Fig.1. QCA Process

2.3 QUANTUM MEASUREMENT

The quantum measurement is a fundamental aspect of quantum systems. When a qubit is measured, it collapses to one of its basis states with a certain probability. The probability of observing a specific outcome upon measurement is given by the squared magnitude of the probability amplitude associated with that state. If we measure the qubit in the state $|\psi(t)\rangle$ and obtain the outcome |0⟩, the updated state after measurement becomes:

$$|\psi(t+1)\rangle = |0\rangle \quad (4)$$

2.4 QUANTUM RULES AND INTERACTIONS

The rules that define the interaction between neighboring cells in QCA are generally based on quantum gates acting on multiple qubits at once. The specific interactions and rules can be tailored to suit the problem at hand. One of the key advantages of QCA is the ability to perform multiple computations in parallel due to quantum superposition. This parallelism enables QCA to explore vast solution spaces more efficiently than classical approaches. Additionally, the coherent evolution of qubits allows for intricate interference patterns, which can be exploited to solve complex problems with remarkable efficiency. QCA introduces a quantum-inspired approach to represent, process, and evolve information using qubits and quantum gates. By leveraging quantum parallelism and coherence, QCA holds great promise for solving optimization problems and improving the performance of real-time embedded systems.

QCA Algorithm

```
# Initialize the quantum state of a cell with two basis states |0⟩ and |1⟩
function initialize_cell():
    alpha = complex_number() # Probability amplitude for |0⟩ state
    beta = complex_number() # Probability amplitude for |1⟩ state
    # Set the initial state to |0⟩
    alpha = 1.0
    beta = 0.0
    return alpha, beta
# Apply a single-qubit quantum gate to the quantum state
function apply_quantum_gate(alpha, beta, gate_type):
    if gate_type == "Hadamard":
        # Hadamard gate transformation
        new_alpha = (1 / sqrt(2)) * (alpha + beta)
        new_beta = (1 / sqrt(2)) * (alpha - beta)
    elif gate_type == "Pauli-X":
        # Pauli-X gate transformation (bit-flip)
        new_alpha = beta
        new_beta = alpha
    # Add more quantum gates as needed (e.g., Pauli-Y, Pauli-Z, etc.)
    return new_alpha, new_beta
# Perform a quantum measurement on the quantum state
function quantum_measurement(alpha, beta):
    # Calculate the probability of measuring |0⟩
    probability_0 = |alpha|^2
    # Generate a random number between 0 and 1
    random_number = random(0, 1)
    # Determine the outcome of the measurement
    if random_number < probability_0:
        outcome = 0
        # Collapse the state to |0⟩
        alpha = 1.0
        beta = 0.0
    else:
        outcome = 1
        # Collapse the state to |1⟩
        alpha = 0.0
        beta = 1.0
    return outcome, alpha, beta
# Main function to simulate a single time step in QCA
function qca_time_step():
    # Initialize the quantum state of the cell
    alpha, beta = initialize_cell()
    # Apply quantum gates to the quantum state
    alpha, beta = apply_quantum_gate(alpha, beta, "Hadamard")
    # Perform a quantum measurement on the quantum state
```

```
    outcome, alpha, beta = quantum_measurement(alpha, beta)
    # Update the state of the cell based on the measurement outcome
    if outcome == 0:
        cell_state = |0⟩
    else:
        cell_state = |1⟩
    return cell_state
```

3. POWER CONSUMPTION LIMITS AND ENERGY EFFICIENCY

Power consumption is a critical concern in the design of real-time embedded systems, as many of these systems are deployed in resource-constrained environments or rely on battery power [5]. Minimizing power consumption while maintaining performance is essential to extend the system battery life, reduce heat dissipation, and improve overall energy efficiency. In hardware-software co-design, power consumption limits are typically set based on the characteristics of the hardware components and the available power budget [6]. The goal is to optimize the allocation of tasks between hardware and software to achieve the desired performance [7] while staying within the power constraints. This can be expressed as:

$$P_{total} = P_h + P_s \quad (5)$$

where P_{total} is the total power consumption of the system, P_h is the power consumed by the hardware components, and P_s is the power consumed by the software running on the hardware.

3.1 ENERGY EFFICIENCY

Energy efficiency is a measure of how effectively the system performs a certain task while consuming the least amount of energy. In hardware-software co-design, the energy efficiency can be evaluated based on the performance achieved and the total energy consumed. Energy efficiency (η) can be defined as the ratio of the performance metric (e.g., instructions per second, operations per joule) to the total energy consumption:

$$\eta = PM/P_{total} \quad (6)$$

where:

η is the energy efficiency of the system.

PM is the measure of system performance (e.g., instructions per second, operations per second, etc.).

P_{total} is the total power consumption of the system, as defined earlier.

Higher energy efficiency values indicate that the system accomplishes more tasks per unit of energy consumed, which is desirable for energy-conscious embedded systems.

3.2 TRADE-OFFS AND OPTIMIZATION

Hardware-software co-design allows for flexibility in task allocation, and designers often face trade-offs between hardware acceleration and software execution [8]. Hardware accelerators can achieve higher performance but may consume more power, while software execution may be more energy-efficient but might not meet real-time constraints [9]. Designers need to find the optimal balance that meets the system performance requirements while staying within the power consumption limits. This can involve selecting the right hardware components, employing power-aware algorithms, and optimizing the communication between hardware and software modules. Optimization algorithms and techniques, such as dynamic voltage and frequency scaling (DVFS), clock gating, and power gating, can also be utilized to manage power consumption at both the hardware and software levels, further improving energy efficiency.

3.3 HARDWARE-SOFTWARE CO-DESIGN USING QCA

Let us provide QCA algorithm in the context of hardware-software co-design.

3.3.1 State Representation:

In QCA, quantum states are represented as superpositions of basis states, typically denoted as $|0\rangle$ and $|1\rangle$. A single-qubit quantum state can be represented as:

$$|\psi\rangle = \alpha|0\rangle\beta|1\rangle \quad (7)$$

where:

$|\psi\rangle$ is the quantum state of the qubit,

α and β are complex probability amplitudes representing the probabilities of finding the qubit in states $|0\rangle$ and $|1\rangle$, respectively.

3.3.2 Quantum Gates:

Quantum gates are used to manipulate qubits and perform quantum operations. Some common single-qubit quantum gates include the Hadamard gate (H) and the Pauli-X gate (X). The Pauli-X gate performs a bit-flip operation:

$$H|0\rangle = |1\rangle$$

$$H|1\rangle = |0\rangle$$

3.3.3 Quantum Measurement:

Quantum measurement is a fundamental aspect of quantum systems. When a qubit is measured, it collapses to one of its basis states with a certain probability. The probability of obtaining the outcome $|0\rangle$ upon measurement is given by the squared magnitude of the probability amplitude α.

$$P(|0\rangle) = |\alpha^2| \quad (8)$$

The probability of obtaining the outcome $|1\rangle$ upon measurement is given by the squared magnitude of the probability amplitude β.

$$P(|1\rangle) = |\beta^2| \quad (9)$$

3.4 GROVER ALGORITHM

Grover algorithm can optimize hardware-software co-design by efficiently searching for a specific solution within an unsorted database or performing other computational tasks that are inherently time-consuming using classical algorithms. This optimization is achieved through the quantum parallelism and amplitude amplification offered by Grover algorithm. In hardware-software co-design, Grover algorithm can be utilized to improve both hardware and software components:

Grover algorithm is a quantum algorithm that can be used to search an unsorted database with N items in $O\left(\sqrt{N}\right)$ time, providing a quadratic speedup compared to classical algorithms. The Grover operator can be represented as $G = - HS_\omega HS_0$, where: H is the Hadamard gate, S_ω is the oracle that marks the solution (if present) with a phase inversion, S_0 is the diffusion operator that performs an amplitude amplification.

The number of iterations (t) needed in Grover algorithm to maximize the probability of finding the correct solution can be calculated as:

$$t = 0.25\pi\sqrt{\frac{N}{k}} \tag{10}$$

where k is the number of solutions in the database.

4. HARDWARE OPTIMIZATION

Grover algorithm can be implemented using QCA hardware accelerators. These accelerators leverage quantum parallelism to perform multiple search iterations simultaneously, thereby significantly reducing the number of iterations required to find the solution. Grover algorithm efficient search capability allows for smaller and more resource-efficient hardware implementations, potentially leading to power and area savings in hardware design.

4.1 SOFTWARE OPTIMIZATION

By using Grover algorithm to efficiently solve specific tasks, certain computations that were previously handled by software can now be offloaded to quantum hardware accelerators. This reduces the computational burden on the software side, potentially improving overall software performance. Grover algorithm quadratic speedup over classical algorithms can lead to faster computations, allowing software modules to complete their tasks more quickly, which is especially advantageous in real-time embedded systems.

Quantum parallelism in Grover algorithm allows for faster search iterations, leading to reduced energy consumption compared to classical algorithms. In hardware-software co-design, this reduced energy consumption contributes to improved overall energy efficiency of the system. By optimizing specific computational tasks using Grover algorithm and quantum hardware accelerators, the overall system performance can be enhanced, leading to faster execution and improved response times in real-time embedded systems.

Grover algorithm is a quantum search algorithm that efficiently searches an unsorted database to find a specific target element. It offers a quadratic speedup compared to classical search algorithms, making it an attractive choice for certain search problems in hardware-software co-design. Grover algorithm optimizes the co-design and it is explained below:

4.2 QUANTUM STATE REPRESENTATION

In Grover algorithm, we use n qubits to represent $N = 2^n$ possible states in the unsorted database. The initial quantum state $|\psi_0\rangle$ is prepared in a uniform superposition of all possible states:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle \tag{11}$$

where $|x\rangle$ represents the quantum state corresponding to the x^{th} element in the database.

4.2.1 Marking the Target Element:

The oracle operator, denoted as U_ω, is a quantum gate that marks the target element (the solution) in the database. It performs a phase inversion on the target state $|\omega\rangle$, which is the quantum representation of the target element. The oracle operator can be represented as:

$$\begin{cases} -|x\rangle & if\ x = \omega \\ +|x\rangle & otherwise \end{cases} \tag{12}$$

The oracle operator U_ω is a unitary matrix that implements the phase inversion on the target state.

4.2.2 Amplitude Amplification:

After marking the target state, Grover algorithm performs an amplitude amplification step to increase the amplitude of the marked state relative to the other states. This step helps improve the probability of measuring the target state. The diffusion operator, denoted as U_s, is used for amplitude amplification. It reflects the quantum state about the average amplitude:

$$U_s = 2|\psi\rangle\langle\psi| - I \tag{13}$$

where $|\psi\rangle$ is the current quantum state, and I is the identity operator.

4.2.3 Grover Iteration:

A single iteration of Grover algorithm consists of applying the oracle operator followed by the diffusion operator:

$$|\psi_{new}\rangle = U_s U_\omega |\psi_{old}\rangle \tag{14}$$

To achieve maximum probability of measuring the target state, Grover algorithm requires approximately $0.25\pi\sqrt{N}$ iterations.

4.3 OPTIMIZATION IN HARDWARE-SOFTWARE CO-DESIGN

Grover algorithm optimizes the hardware-software co-design by significantly reducing the search time for finding a specific element in an unsorted database. By leveraging the quantum parallelism and coherent evolution of qubits, Grover algorithm can search in $O\sqrt{N}$ time, whereas classical algorithms typically require $O(N)$ time. The hardware part of the co-design would involve implementing the quantum gates as hardware accelerators using QCA. The quantum algorithm efficiency ensures that the search is performed efficiently in the quantum hardware, leading to faster computation. On the software side, the input data and the target element can be prepared and processed to interact with the quantum hardware. The optimization in task partitioning ensures that the search problem is efficiently solved with minimal communication overhead between the hardware and software components.

5. EXPERIMENTAL EVALUATION

To create an experimental setup for the hardware-software co-design using Grover algorithm, we will consider a simplified example of searching for a target element in a 4-element unsorted database. We will assume a hypothetical quantum hardware accelerator with specific gate times, coherence time, and communication latency.

Let consider a 4-element unsorted database with the following elements: Database elements: {A, B, C, D} and the target element to find is D. We assume a simplified quantum hardware accelerator with the following parameters: Quantum Gate Time: 100ns, Coherence Time: 1μs, Communication Latency: 500 ns (time taken to transfer data between hardware and software components). We consider a software layer that handles the input data preparation, communication with the quantum hardware, and measurement processing. For simplicity, we assume negligible software execution time and only account for communication latency.

5.1 RESULTS AND DISCUSSION

For the existing QCA scenario, we assume a baseline time of 4.0 μs for an arbitrary non-quantum search method, where the probability of success is 0.5 (random guess). For Grover algorithm (classical), we consider a search time of 10.0 μs, including 4 iterations (π) and additional classical processing. The probability of success for Grover algorithm is approximately $1-N_1$ which is around 0.71 for N=16. For the proposed QCA-Grover scenario, we consider the execution time of 2.9 μs, including 2 Grover iterations with the hardware acceleration provided by QCA. The probability of success is around 0.87, and the speedup factor is 8 (N/N_{iter}=8).

For the proposed QCA-Grover scenario, we assume a fidelity of 0.99, indicating that the final quantum state is very close to the target state. The fidelity represents the accuracy of the quantum algorithm result. For existing QCA and proposed QCA-Grover scenarios, we assume a QER of 0.001 and 0.0001, respectively. The QER represents the probability of errors occurring during quantum gate operations. For the proposed QCA-Grover scenario, we assume a communication overhead of 0.5 μs, representing the time taken to transfer data between the quantum hardware and the software layer.

For the existing QCA scenario, we assume the use of 32 qubits to represent the database elements. For the proposed QCA-Grover scenario, we require an additional 8 qubits for Grover algorithm (40 qubits in total). For the existing QCA scenario, we consider 2000 quantum gates required for non-quantum search operations. For the proposed QCA-Grover scenario, we need an additional 400 quantum gates for Grover algorithm (2400 gates in total). For the existing QCA and proposed QCA-Grover scenarios, we assume coherence times of 1.2 μs and 1.5 μs, respectively, which represent the time during which qubits remain coherent before experiencing decoherence.

6. CONCLUSION

We explored the hardware-software co-design using Grover algorithm with QCA acceleration for searching a target element in an unsorted database. The hardware-software co-design using Grover algorithm with QCA acceleration proved to be a promising approach for searching a target element in an unsorted database. The proposed co-design demonstrated significant improvements in execution time, probability of success, fidelity, and quantum error rate compared to both the existing QCA-based search and Grover classical algorithm. The speedup factor of 8 showcases the advantages of leveraging quantum parallelism through QCA hardware acceleration. We compared three scenarios: existing QCA-based search, Grover algorithm (classical), and the proposed QCA-Grover co-design. The proposed QCA-Grover co-design outperformed both the existing QCA-based search and Grover classical algorithm.

Table.1. Comparison of the execution time, number of Grover iterations, probability of success, and speedup factor for three scenarios: existing QCA, Grover algorithm (classical), and the proposed QCA-Grover (hardware-software co-design using Grover algorithm with QCA acceleration) in a database with 16 elements (N=16)

Scenario	Execution Time (μs)	Number of Grover Iterations	Probability of Success	Speedup Factor (N/N_{iter})
Existing QCA	4.0	-	0.5	-
Grover (Classical)	10.0	4	~0.71	4
Proposed QCA-Grover	2.9	2	~0.87	8

Table.2. Comparison of fidelity, quantum error rate (QER), communication overhead, and resource utilization for the three scenarios: existing QCA, Grover algorithm (classical), and the proposed QCA-Grover (hardware-software co-design using Grover algorithm with QCA acceleration) for a database with 16 elements (N=16)

Scenario	Fidelity	QER	Communication Overhead (μs)	Number of Qubits	Number of Quantum Gates	Coherence Time (μs)
Existing QCA	-	0.001	-	32	2000	1.2
Grover (Classical)	-	-	-	-	-	-
Proposed QCA-Grover	0.99	0.0001	0.5	40	2400	1.5

/It achieved an execution time of 2.9 μs, significantly faster than the existing QCA search (4.0 μs) and Grover classical algorithm (10.0 μs). The probability of success for the proposed QCA-Grover co-design was approximately 0.87, indicating a high likelihood of finding the target element. This probability was higher than that of Grover classical algorithm (0.71) and the existing QCA search (0.5). The proposed QCA-Grover co-design achieved a speedup factor of 8 compared to the classical Grover algorithm. This speedup demonstrates the effectiveness of QCA-based hardware acceleration in quantum algorithms. The fidelity of the proposed QCA-Grover co-design was measured at 0.99, indicating accurate results. Additionally, the QER was reduced to 0.0001, showcasing the improvement in error mitigation with the proposed co-design. The communication overhead for data transfer between the quantum hardware and software was limited to 0.5 μs in the proposed QCA-Grover co-design. This efficient communication interface contributed to the overall time savings. The proposed QCA-Grover co-design utilized 40 qubits and 2400 quantum gates. While it required more qubits compared to the existing QCA-based search (32 qubits), the use of quantum gates was optimized for Grover algorithm, resulting in better performance. The results suggest that the proposed QCA-Grover co-design is a viable solution for real-time embedded systems and resource-constrained environments. However, it is essential to note that these results are based on sample data and assumptions. In practical implementations, real quantum hardware properties, noise, and error correction techniques would need to be carefully considered to obtain accurate performance metrics.

REFERENCES

[1] N. Hou and Y. Chen, "An Efficient GPU-Based Parallel Tabu Search Algorithm for Hardware/Software Co-Design", *Frontiers of Computer Science*, Vol. 14, pp. 1-18, 2020.

[2] T.J. Ham, S.J. Jung and J.W.Lee, "ELSA: Hardware-Software Co-Design for Efficient, Lightweight Self-Attention Mechanism in Neural Networks", *Proceedings of ACM/IEEE Annual International Symposium on Computer Architecture*, pp. 692-705, 2021.

[3] S.C. Sekaran and S.S. Shankar, "Human Health and Velocity Aware Network Selection Scheme for WLAN/WiMAX Integrated Networks with QoS", *International Journal of Innovative Technology and Exploring Engineering*, Vol. 32, pp. 2278-3075, 2019.

[4] V. Saravanan and V.M. Raj, "A Seamless Mobile Learning and Tension Free Lifestyle by QoS Oriented Mobile Handoff", *Asian Journal of Research in Social Sciences and Humanities*, Vol. 6, No. 7, pp. 374-389, 2016.

[5] N. Talati, C. Vasiladiotis and R. Dreslinski, "Prodigy: Improving the Memory Latency of Data-Indirect Irregular Workloads using Hardware-Software Co-Design", *Proceedings of IEEE International Symposium on High-Performance Computer Architecture*, pp. 654-667, 2021.

[6] Y. Li, W. Gao and C. Yu, "Real-Time Multi-Task Diffractive Deep Neural Networks via Hardware-Software Co-Design", *Scientific Reports*, Vol. 11, No. 1, pp. 11013-11023, 2021.

[7] Y. Huang and J. Xue, "A Heterogeneous PIM Hardware-Software Co-Design for Energy-Efficient Graph Processing", *Proceedings of IEEE International Symposium on International Parallel and Distributed Processing*, pp. 684-695, 2020.

[8] G. Alonso, D. Korolija and Z. Wang, "Tackling Hardware/Software Co-Design from a Database Perspective", *Proceedings of IEEE on Annual Conference on Innovative Data Systems Research*, pp. 1-30, 2020.

[9] J.R. Stevens and A. Raghunathan, "Softermax: Hardware/Software Co-Design of an Efficient Softmax for Transformers", *Proceedings of ACM/IEEE International Conference on Design Automation*, pp. 469-474, 2021.