Data and Metadata. 2025; 4:835
doi: 10.56294/dm2025835

ORIGINAL

Heterogeneous Computing for High-Complexity Problems: A Performance Study with DPC++ on HPC Platforms

Computación heterogénea para problemas de alta complejidad: un estudio de rendimiento con DPC++ en plataformas HPC

Katari Tituaña[1], MacArthur Ortega-Bustamante[1], Pedro Granda[1], Marco Pusdá-Chulde[1]

[1]Universidad Técnica del Norte, Facultad de Ingeniería en Ciencias Aplicadas. Ibarra, Ecuador.

Cite as: Tituaña K, Ortega-Bustamante M, Granda P, Pusdá-Chulde M. Heterogeneous Computing for High-Complexity Problems: A Performance Study with DPC++ on HPC Platforms. Data and Metadata. 2025; 4:835. https://doi.org/10.56294/dm2025835

Submitted: 13-07-2025 **Revised:** 10-09-2025 **Accepted:** 03-12-2025 **Published:** 04-12-2025

Editor: Dr. Adrián Alejandro Vitón Castillo

Corresponding Author: Katari Tituaña

ABSTRACT

The All-Pairs Shortest Paths (APSP) algorithm is crucial for a wide range of computing applications that necessitate the efficient calculation of minimum distances between all pairs of nodes in a graph. High-performance computing (HPC) has made significant progress in response to the demand to solve complex computing problems that process large amounts of data. The cubic complexity of the Floyd-Warshall algorithm makes its execution a computational challenge with large graphs, limiting its applicability on conventional platforms. This research addresses this problem by evaluating optimized and parallelized implementations in HPC architectures. The study aims to evaluate the performance of the APSP algorithm by parallel implementation using Intel DPC++ on heterogeneous high-performance architectures, including CPU, GPU, and FPGA. The Intel OneAPI platform was used for the implementation and execution of the algorithm on Intel Xeon Gold 6128 CPU-3.40 GHz processors, Intel Data Center GPU Max 1100, and FPGA Emulation Device processors. The solution was evaluated considering quantitative and statistical metrics, obtaining a significant improvement in the performance of the algorithm for large graphs, with 700x speedup and 70 % efficiency in GPU architectures with 65536 nodes and 1024 parallelism subblocks. This study confirms the feasibility of implementing efficient and portable HPC solutions using DPC++ for massive data processing in heterogeneous architectures, offering a scalable and effective solution for modern computational challenges.

Keywords: Heterogeneous Programming; All Pairs Shortest Paths; HPC; Intel oneAPI; DPC++; Parallelism.

RESUMEN

El algoritmo All Pairs Shortest Paths (APSP) es fundamental para una amplia variedad de aplicaciones informáticas que requieren la evaluación eficiente de distancias mínimas entre todos los pares de nodos en un grafo. La computación de alto rendimiento (HPC) ha progresado significativamente en respuesta a la demanda para solucionar problemas de computación complejos que procesan grandes cantidades de datos. La complejidad cúbica del algoritmo Floyd-Warshall convierte su ejecución en un desafío computacional con grafos de gran tamaño, limitando su aplicabilidad en plataformas convencionales. Esta investigación aborda dicho problema mediante la evaluación de una implementación optimizada y paralelizada en arquitecturas HPC. EL estudio tiene como objetivo evaluar el rendimiento del algoritmo APSP mediante la implementación paralela utilizando Intel DPC++ sobre arquitecturas heterogéneas de alto rendimiento, incluyendo CPU, GPU y FPGA. Se utilizó la plataforma de Intel OneAPI para la implementación y ejecución del algoritmo en procesadores Intel Xeon Gold 6128 CPU-3.40 GHz, Intel Data Center GPU Max 1100 y FPGA Emulation Device. La solución fue evaluada considerando métricas cuantitativas y estadísticas obteniendo una mejora significativa en el rendimiento del algoritmo para grafos de gran tamaño, con speedup de 700x y eficiencia del 70 % en arquitecturas GPU con grafos de 65536 nodos y 1024 subbloques de paralelismo. Este estudio confirma la viabilidad de implementar soluciones HPC eficientes y portables utilizando DPC++ para el procesamiento masivo de datos en arquitecturas heterogéneas, ofreciendo una solución escalable y eficaz para los desafíos computacionales modernos.

Keywords: Programación Heterogénea; All Pairs Shortest Paths; HPC; Intel oneAPI; DPC++; Paralelismo.

INTRODUCTION

Enterprise computing systems require processing large volumes of data, which exceeds the capabilities of dedicated computer architectures for personal use due to the high computational cost of the algorithms employed. The lack of optimization and the use of conventional hardware without specialized accelerators, such as GPUs or FPGAs, limit its efficiency.(1,2,3,4) Processing large amounts of data can result in high runtimes or even the collapse of the system itself, as occurs in climate prediction models or complex mathematical algorithms.(5,6) Despite advances in High Performance Computing (HPC) and heterogeneous programming, its implementation remains limited, preventing the effective utilization of available resources.(7)

Advances in hardware development require adopting new curricula in careers focused on computer science and software engineering, as there is a knowledge gap in parallel computing topics.(6) Technologies such as heterogeneous architecture, parallel programming, and hardware accelerators (GPUs and FPGAs) have transformed the way computationally intensive algorithms are designed and implemented.(8) Heterogeneous programming has established itself as a technological strategy for the execution of high-performance applications on different electronic devices that support architectures for parallel processing.(9) Conventional programming languages have certain limitations to efficiently exploit the computing resources of systems that incorporate multiple processors, such as the GPU or FPGA.(10,11,12)

The Intel Developer Cloud (IDC) is a platform that provides free access to a variety of Intel architectures, enabling workloads to run in rendering, computer vision, machine learning, deep learning, and edge computing environments. IDC is an infrastructure that incorporates a set of pre-installed and optimized frameworks, tools, and libraries, such as oneAPI - DPC++, for efficient development and greater portability of source code in environments of heterogeneous Architectures.(13) DPC++, created by Intel, is an implementation of SYCL that incorporates an additional abstraction layer, which facilitates programming in heterogeneous architectures by providing a unified model for execution algorithms in different types of accelerators (CPU, GPU, FPGA).(14) DPC++ portability allows the same code to run on CPUs, GPUs, and other accelerator architectures, making it easier to explore efficient and scalable solutions.

The APSP algorithm allows you to calculate the minimum distances between pairs of nodes in a graph. APSP is used for applications in various areas of computing, such as social networking,(15) bioinformatics,(16) path analysis,(17,18) recommendation systems.(19) However, its $O(n^3)$ complexity according to the Big O notation, especially in the Floyd-Warshall version,(13,20) making it an ideal candidate for optimization using HPC techniques.(21) HPC accelerates, scales, and optimizes the development of AI models for large-scale research and applications that would be impossible with conventional resources.(22,23) In fact, convolutional neural networks are widely used in fields such as precision agriculture,(24,25,26,27) security,(28) livestock,(29) medicine,(30,31) education.(32)

In this article, the APSP algorithm was implemented in DPC++ sequentially and in parallel with Intel's HPC platform. In the parallelized version, performance was evaluated in three accelerators (CPU, GPU, and FPGA) to take advantage of the heterogeneity offered by the DPC++ language. A comparison of the algorithms was carried out using metrics to evaluate the behavior when increasing the size of the dataset.

Related Jobs

In the field of programming in HPC environments and heterogeneous programming, different studies have been carried out, such as the work.(33) The problem of optimizing parallel algorithms in heterogeneous systems was analyzed. The main problems that arise when adapting a sequential algorithm to a parallel one are discussed, and heuristics are proposed to address the detected problems.

The work (34) investigates the drawbacks that arise when developing software for HPC in each scientific field. Considering the complexity of developing such programs, it is useful to identify appropriate programming languages to solve these problems. To this end, a systematic mapping study (SMS) of the characteristics of several programming languages focused on HPC with intensive use in data was carried out. In addition, the result of the mapping study was compared with the results of a questionnaire-based survey conducted on 57 HPC experts. Studies revealed that the desired features of HPC programming languages are portability, performance, and ease of use. It was detected that programming languages for this purpose have a steep learning curve, which makes their adoption difficult.

The study [35] evaluates the performance of the Particle Swarm Optimization (PSO) algorithm and introduces a parallel version of the algorithm (PPSO) into the multi-core processing kernel to reduce determination. To facilitate the transfer of information between the particles in the shared area, and the exchange of information through random switching. The proposed algorithm utilizes a multi-core CPU technique to enhance its efficiency through parallelization, thereby increasing the application range of PSOs.

The work [36] implemented a crystallization algorithm using a separation technique of significant computational complexity, which is used due to its high efficiency and ease of operation. The acceleration of the Non-Dominant Genetic Ordering Algorithm II (NSGA-II) using NUMBA CUDA for the multi-objective optimization of ortho-aminobenzoic acid crystallization was studied. By leveraging parallel processing on the GPU, significant reductions in compute time are achieved, especially for large populations and complex optimization problems. The GPU-accelerated NSGA-II algorithm demonstrates a significant performance increase by reducing time by 122x without any reduction in solution quality.

The study [37], conducted an analysis on parallel programming in HPC systems, providing an overview of the main parallel programming models and paradigms. In addition to defining the fundamental concepts, the paper examines the implementation and performance of OpenMP, demonstrating its effectiveness in optimizing computational processes in high-performance environments. To evaluate OpenMP parallelism, the matrix multiplication scenario was used. The algorithm was tested on a system with a 1.10GHz dual-core Intel Core N4000 CPU and 4GB of RAM. It was determined that time decreases as the number of threads increases.

Based on the literature reviewed, the implementation of algorithms with parallelism is essential to take full advantage of the computing capacity of modern systems, allowing large and complex problems to be solved more quickly and efficiently. Parallelism significantly improves the performance, efficiency, and scalability of algorithms, especially in contexts where the volume of data or processing demand is high.

METHOD

This section describes the resources and methodological approach used to evaluate the performance of heterogeneous programming in the optimization of high-computational cost algorithms. The analysis and development of the algorithm were carried out in 5 phases: i) documentary research, ii) experimental environment, iii) algorithm implementation, iv) dataset, and v) evaluation metrics.

Documentary research

The documentary review was conducted using scientific articles published in reputable databases, including Scopus, IEEE, and Google Scholar. The topics reviewed included heterogeneous programming, high-computational-cost algorithms, HPC, parallel processing, and heterogeneous accelerators. Additionally, Intel documentation was utilized for the study of the heterogeneous programming language DPC++.

Experimental environment

The Intel DevCloud platform is fully integrated with Intel oneAPI and developer tools, providing support for DPC++ and enabling heterogeneous programming. The runtime used for algorithm evaluation is equipped with Intel Xeon processors that support acceleration via GPU cards and FPGAs. The Intel DevCloud platform operates under the Ubuntu 18.04.3 LTS operating system, providing a stable and optimized environment for running intensive workloads.

The CPU used for testing was Intel Xeon Gold 6128 CPU - 3.40GHz, 6 cores/12 threads. The GPU on which the tests were carried out was Intel Data Center GPU Max 1100, 56 ray tracing units, 448 vector units. The FPGA unit used was the Intel FPGA Emulation Device.

Algorithm implementation

To determine the algorithmic complexity, the Big O notation was used, which allows us to understand the behavior of the algorithm as the amount of input data increases. The Floyd-Warshall algorithm, used to solve the APSP problem, has a computational complexity of $O(n^3)$. This indicates that the algorithm execution time grows cubically with respect to the number of nodes, which represents a high computational cost, especially in large graphs.[38]

The Floyd-Warshall algorithm is based on a dynamic programming approach to calculate the minimum distances between all pairs of nodes in a graph, regardless of whether it is directed or undirected. The algorithm works as follows:

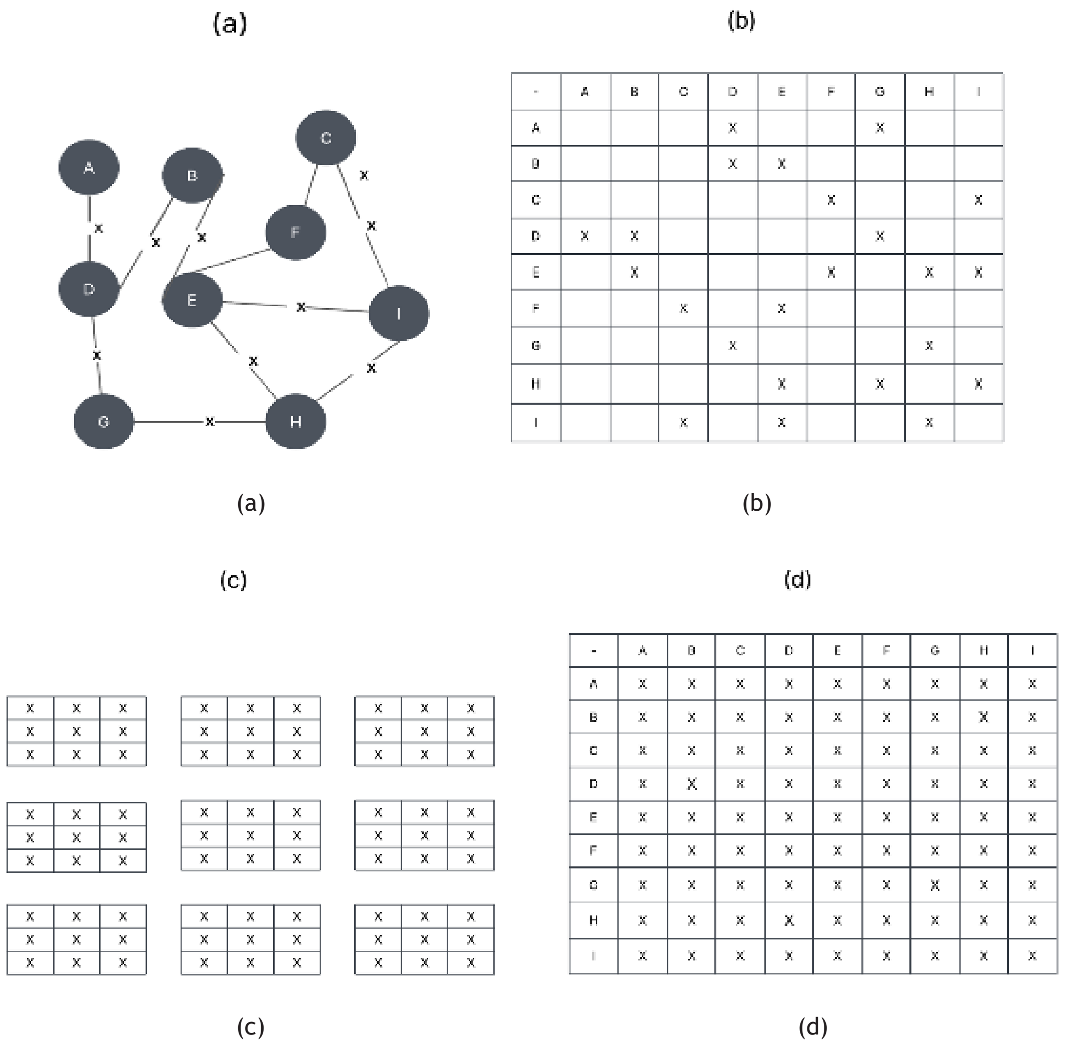
- The result matrix is initialized as the matrix of the input graph.
- All nodes are considered one by one; The shortest path between each pair of nodes is updated, including the node selected as the intermediate node.
- Choosing a k node as an intermediate node implies that nodes {0,1,...,k-1} have already been selected as intermediate nodes.

- Assuming that the element d[i][j] in the array denotes the shortest distance from a source node i to a destination node j, for each pair of nodes (i, j), one of the following conditions applies:
- If k is not an intermediate node on the shortest path from i to j, then d[i][j] remains unchanged.
- Otherwise, d[i][j] is updated to (d[i][k] + d[k][j]), as long as d[i][j] > (d[i][k] + d[k][j]).

Block-based parallelism

DPC++ facilitates the implementation of the APSP algorithm by using parallelization blocks (parallel_for) to divide the adjacency matrix into concurrently actionable subregions. In the CPU architecture, blocks are allocated by row in the array and processed using multiple threads with implicit core allocation, thereby maximizing core utilization. In the GPU, multidimensional blocks are applied where each work-item can represent a pair (i,j), allowing a massive and balanced parallelism in distance processing. For FPGAs, loop unrolling and kernel pipelining techniques are applied, combined with the use of local buffers, which allows the construction of logical blocks that process data flows continuously with high efficiency. This strategy enhances the performance, scalability, and portability of the algorithm in heterogeneous environments and is crucial for achieving consistent results across the various platforms evaluated.

In the implementation of the APSP algorithm in parallel blocks, each thread is responsible for processing a cell within a block. For each block, the corresponding function is executed the same number of times as the number of cells it contains. Figure 1 details the process of the APSP algorithm with parallelism. Each invocation performs iterations equivalent to the number of blocks in a single dimension of the array (c) using a parallel_for loop. At the end of each iteration, all threads working simultaneously within the same block are synchronized, thus ensuring consistency and accuracy of the results (d).



-	A	B	C	D	E	F	G	H	I
A				X			X		
B				X	X				
C						X			X
D	X	X					X		
E		X				X		X	X
F			X		X				
G				X				X	
H					X		X		X
I			X		X			X	

-	A	B	C	D	E	F	G	H	I
A	X	X	X	X	X	X	X	X	X
B	X	X	X	X	X	X	X	X	X
C	X	X	X	X	X	X	X	X	X
D	X	X	X	X	X	X	X	X	X
E	X	X	X	X	X	X	X	X	X
F	X	X	X	X	X	X	X	X	X
G	X	X	X	X	X	X	X	X	X
H	X	X	X	X	X	X	X	X	X
I	X	X	X	X	X	X	X	X	X

Figure 1. Stages of the APSP algorithm in parallel blocks: (a) original graph, (b) graph adjacency matrix, (c) calculation of the shortest path between all pairs of nodes in parallel blocks, and (d) regrouping the adjacency matrix of the graph

Datasets

The graphs used to evaluate the algorithms were generated in incremental sizes as an exponential function of base 2, where an adjacency matrix is created to store a weighted graph. The size of the array is nodes x nodes. For the parallelized algorithm, blocks in base 2 were applied in a way that increases the no. of nodes in the graph. The size of the generated graphs and the number of blocks are shown in table 1.

Table 1. Size and subblocks of graphs used

Graph	Nro. Nodes	Subblocks Parallel
Graph 1	128	8
Graph 2	256	16
Graph 3	512	32
Graph 4	1024	64
Graph 5	2048	128
Graph 6	4096	256
Graph 7	8192	512
Graph 8	16384	1024

Quantitative metrics

To carry out the evaluation and analysis of the results, specific criteria were established to ensure an objective measurement of the quality and performance of the algorithms. In the case of parallel algorithms, performance is quantified by means of consolidated metrics in the literature, including execution time, SpeedUp, and efficiency. Similarly, scalability is recognized as a determining factor, since it allows characterizing the behavior of the algorithm based on both the size of the input data and the particularities of the hardware architecture used in its implementation.(23)

In each experimentation scenario, the experiment was executed 10 times for each graph to obtain an average execution time. The evaluations were carried out on the three architectures with the maximum configurations and processing capacities mentioned in the experimental environment section.

Runtime: execution time is a metric that represents the total duration required by an algorithm to complete an instruction. This measure is based on recording the time elapsed from the beginning to the end of the algorithm's execution process. Because of the variations that may occur in code execution, even under the same test conditions. The start time (Ti) is obtained by capturing the current system time when the algorithm execution starts, and the end time (Tf) is obtained by capturing the current system time after the algorithm execution has finished. Sequential time is calculated from the difference between the end time and the start time, equation 1.

- Ti = Starting Time.
- Tf = Tiempo final.

$$Tsec = Tf - Ti \qquad (1)$$

Speedup: also referred to as acceleration, is a widely used metric to evaluate the improvement in an algorithm's performance when implemented in parallel compared to its sequential version. To determine the efficiency of parallelization, it is necessary to measure the execution times of both implementations and calculate the relationship between them, which allows quantifying the degree of acceleration obtained.(39) The speedup calculation is presented in equation 2.

- Tsec = Sequential Program Execution Time.
- Tpar = Parallel Program Execution Time.

$$Sp = \frac{Tsec}{Tpar} \qquad (2)$$

SpeedUp is interpreted based on 1; the higher the result compared to 1, the better the algorithm's performance in parallel.(39) If the result is equal to 1, it means that there is no performance improvement when parallelizing the algorithm; the execution time is the same in both versions, sequential and parallel. If the result is less than 1, it means that the parallel version is less efficient than the sequential version and therefore loses performance.

Efficiency: measures the use of parallelism in relation to the number of resources (subblocks).(39)

- Sp = Algorithm Acceleration.
- P = Number of subblocks.

$$Ef = \frac{Sp}{P} \quad (3)$$

Statistical Metrics

To reduce the effect of uncontrolled variations in system load, each experiment was repeated 10 times, recording the individual average execution times. The statistical metrics are detailed in table 2, including the mean (equation 4), standard deviation (equation 5), and the 95 % confidence interval (equation 6). These values were used to determine the stability and reliability of the performance observed in each architecture.

Table 2. Statistical metrics to determine stability and reliability

Media	Standard deviation	95 % confidence interval
$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$	$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$	$IC_{95\%} = \bar{x} \pm t_{n-1,0.025} \cdot \frac{8}{\sqrt{n}}$ tn−1,0,025 is the t-student value for 95 % confidence
(4)	(5)	(6)

RESULTS AND DISCUSSION

Quantitative metrics

The results evaluated (table 3) show that increasing the number of subblocks does not necessarily improve the performance of the APSP algorithm in CPU and FPGA architectures. Although increasing the number of subblocks could suggest greater parallelization, in practice, it generates an overload of coordination and communication, which explains the significant increase in execution times, especially in the CPU and FPGA, from 512 subblocks. On the contrary, the GPU maintains lower times in the same scenario, confirming its ability to efficiently handle highly parallel tasks.

In figure 2, the runtime in seconds is presented to solve the APSP algorithm of different increasing sizes, using parallel implementations with DPC++ in CPU, GPU, and FPGA architectures.

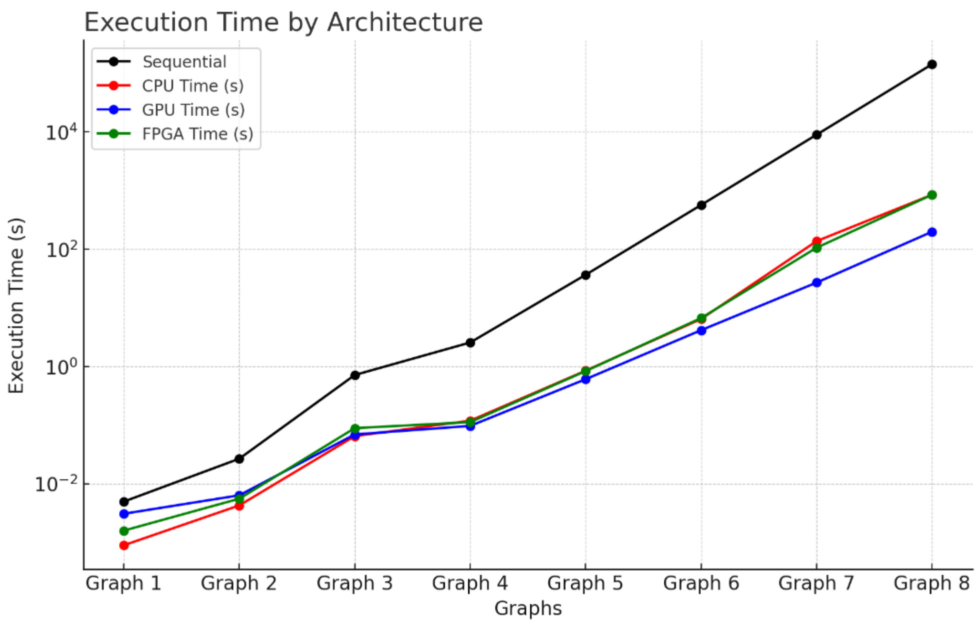


Figure 2. Runtime behavior in heterogeneous architectures

In small graphs (1 to 3), CPU execution times are lower due to the low load of nodes and parallelism subblocks. Starting with graph 4, GPU times consistently outperform CPU and FPGA architectures, with an incremental gap as the graph size increases. The FPGA has intermediate performance, taking less time than a CPU but longer than a GPU in most cases, except for very small graphs.

Table 3. Speedup and efficiency obtained based on average execution time times

Graph	Nodes	Sub Blocks(P)	Time(s)				Speedup (Sp)			Efficiency (Ef)		
			Sequential	CPU	GPU	FPGA	CPU	GPU	FPGA	CPU	GPU	FPGA
1	512	8	0,0050	0,0009	0,0031	0,0016	5,5933	1,6239	3,1463	0,6992	0,2030	0,3933
2	1024	16	0,0269	0,0053	0,0064	0,0056	5,0664	4,1956	4,7950	0,3167	0,2622	0,2997
3	2048	32	0,7189	0,0253	0,0698	0,0193	28,4138	10,2990	37,2472	0,8879	0,3218	0,0354
4	4096	64	2,5752	0,1197	0,0975	0,1131	21,5137	26,4123	22,7692	0,3362	0,4127	0,3558
5	8192	128	36,7015	0,8516	0,6097	0,8385	43,0971	60,1960	43,7704	0,3367	0,4703	0,3420
6	16 384	256	569,0211	6,4823	4,1755	6,7233	87,7807	136,2762	84,6342	0,3429	0,5323	0,3306
7	32 768	512	8966,9674	137,4724	27,0217	106,1309	65,2274	331,8432	84,4897	0,1274	0,6481	0,1650
8	65 536	1024	142 580,2703	848,9897	198,9133	849,8751	167,9411	716,7961	167,7661	0,1640	0,7000	0,1638

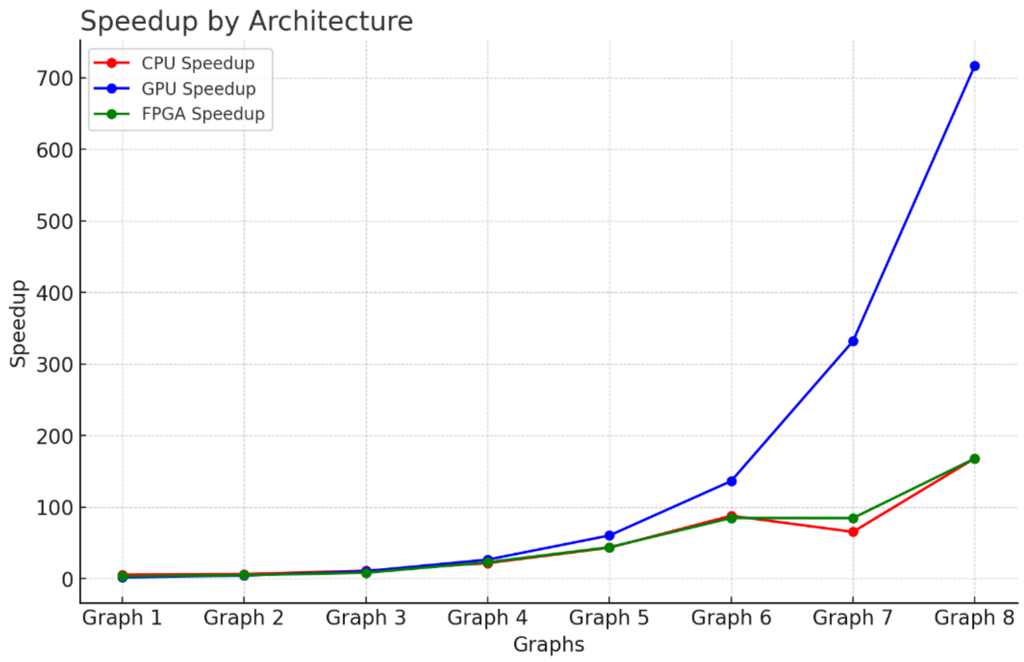


Figure 3. Speedup behavior in heterogeneous architectures

For the graph with the largest number of nodes (graph 8), the execution time is almost 4 times shorter than that of CPU and FPGA, confirming the capacity to process large amounts of data. Figure 3 shows the speedup achieved by each architecture, i.e., the number of times faster the algorithm is with parallelism with respect to the sequential one.

GPU speedup grows almost exponentially with graph size, peaking at 716,8× in graph 8. The CPU reaches a speed of 167,9×, but with a more linear trend. The FPGA, on the other hand, exhibits more irregular behavior, achieving good values in medium-sized graphs but stagnating in the largest ones (for example, in graph 8, it only achieves 167,8×, similar to the CPU). This behavior suggests that the GPU execution model scales better, while CPUs and FPGAs experience bottlenecks when increasing the workload.

Figure 4 shows efficiency, defined as the speedup divided by the number of subblocks used in the algorithm. It measures how much of the available theoretical parallelism is being exploited.

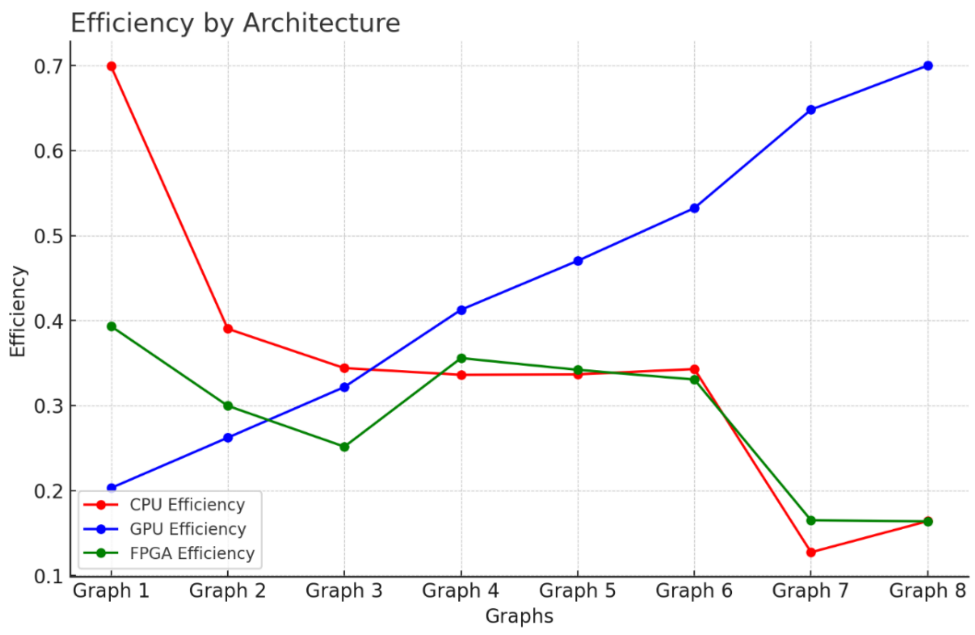


Figure 4. Graph of efficiency in heterogeneous architectures

The GPU shows the highest and most consistent efficiency, reaching 70 % efficiency in graph 8, which indicates optimal resource utilization and effective parallelism. CPU and FPGA have significantly lower values, decreasing on larger graphs due to scalability issues, synchronization overhead, or underutilization of available parallelism. CPU and FPGA efficiency reach only 16 % in graph 8, indicating that the increase in subblocks is insufficient to offset the performance loss under large loads.

Statistical Metrics

Table 4 presents the average execution times in seconds along with their 95 % confidence interval, estimated from 10 executions for each Architecture.

Table 4. Average execution times in seconds with 95 % confidence interval

Graph	Nodes	Sub-blocked	TCPU (s) ± IC95	TGPU (s) ± IC95	TFPGA (s) ± IC95
Graph 1	512	8	0,0009 ± 0,0001	0,0031 ± 0,0003	0,0016 ± 0,0002
Graph 2	1024	16	0,0043 ± 0,0005	0,0064 ± 0,0006	0,0056 ± 0,0006
Graph 3	2048	32	0,0653 ± 0,0074	0,0698 ± 0,0079	0,0893 ± 0,0101
Graph 4	4096	64	0,1197 ± 0,0136	0,0975 ± 0,0110	0,1131 ± 0,0128
Graph 5	8192	128	0,8516 ± 0,0970	0,6097 ± 0,0695	0,8385 ± 0,0955
Graph 6	16 384	256	6,4823 ± 0,7388	4,1755 ± 0,4758	6,7233 ± 0,7661
Graph 7	32 768	512	137,4724 ±15,6703	27,0217 ± 3,0817	106,1309 ±12,0936
Graph 8	65 536	1024	848,9897 ± 96,8467	198,9133 ± 22,6903	849,8751 ± 96,9571

The values in figure 5 reflect that the evaluations for the three architectures are statistically stable, especially in GPUs. The fluctuations observed in FPGAs and CPUs for large graphs suggest structural or administrative bottlenecks (back-end processes, controllers, etc.).

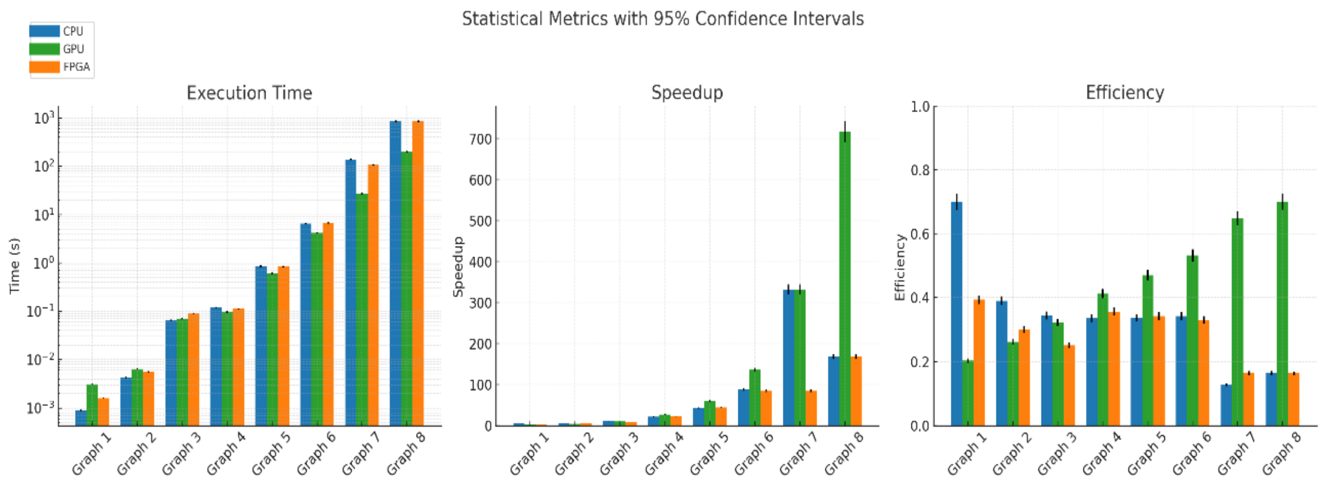


Figure 5. Summary of the evaluated marketing metrics

The comparative analysis of the present work showed that DPC++ offers a portable and efficient alternative to classic solutions based on parallelism and HPC, with the advantage of unifying programming for different architectures. The value of a maximum speedup of 716,79x in graph 8, with an efficiency of 70 %, indicates an optimal use of massive parallelism. This is due to its ability to run thousands of concurrent threads, especially beneficial in computationally intensive tasks such as APSP. In contrast, CPUs have fewer physical cores, which limits their scalability. The FPGA, while having good metrics on medium graphs, does not maintain the same performance on very large graphs, probably due to memory bandwidth constraints or bottlenecks in host-device communication.

The study,(11) which utilizes CPU-FPGA systems based on recursive variants of the Kleene and Floyd-Warshall algorithms, achieves performance comparable to that of the GPU, while consuming lower power and utilizing fewer hardware resources. It outperforms CPU-only solutions by more than 137x on large graphs and delivers 13 % more performance per watt than GPU deployments. The study (12) uses algorithms that partition the graph and process subgraphs in parallel (using OpenMP for CPU and CUDA for GPU), obtaining accelerations of up to 8,3x over traditional Dijkstra, outperforming other parallel algorithms such as n-Dijkstra, ParAPSP, and SuperFW. In contrast, the present work demonstrates that the execution time is almost 4 times shorter than that of CPUs and FPGAs with large graphs.

Limitations include that FPGA testing was performed in an emulation environment, which introduces a latency penalty that would not be present in a physical FPGA configured for high performance. Additionally, the use of fixed subblocks (without dynamic adaptation) can lead to underutilization of CPU resources or premature GPU saturation for small graphs. Another aspect to consider is that the observed performance does not include optimizations at the shared memory level or kernel fusion strategies in DPC++, which could improve real efficiency. Another limitation of the Intel Developer Cloud platform is that all the available hardware is owned by Intel, which makes it challenging to test with third-party hardware.

CONCLUSIONS

The use of programming languages designed for heterogeneous systems, such as DPC++, significantly optimizes development time by eliminating the need to learn multiple languages for different architectures. This provides greater flexibility, allowing programmers to efficiently use available computational resources according to their performance needs.

The results obtained in the performance tests confirm that the implementation of APSP in DPC++ is effective for execution on modern HPC platforms with heterogeneous architectures. Performance testing with the APSP algorithm, where a lower average execution time was obtained with the three architectures, compared to the sequential implementation (figure 2). Speedup's analysis shows that the performance of the algorithms is highly dependent on the architecture used and the size of the graphs.

The combination of portability, multi-level parallelization (subblocks), and support for Intel devices enables the development of scalable and sustainable solutions. The use of DPC++ (SYCL) made it possible to maintain a common code base for the three architectures, which demonstrates the viability of this model as a portable alternative to CUDA or OpenCL, especially in cloud environments with Intel devices. As future work, the use of other programming languages, integration with real FPGA architectures, and the evaluation of the algorithm in hybrid CPU/GPU networks of different manufacturers is proposed.

BIBLIOGRAPHIC REFERENCES

1. Huang S, Wu K, Chalamalasetti SR, El Hajj I, Xu C, Faraboschi P, et al. A Python-based High-Level Programming Flow for CPU-FPGA Heterogeneous Systems: (Invited Paper). Proceedings of PEHC 2021: Workshop on Programming Environments for Heterogeneous Computing, Held in conjunction with SC 2021: The International Conference for High Performance Computing, Networking, Storage and Analysis. 2021:20-6. https://doi.org/10.1109/PEHC54839.2021.00008.

2. Ramírez Patiño LM, Guerrero Hernández LE. Arquitecturas Híbridas o Heterogéneas Paralelo entre NVIDIA CUDA y Parallel Studio XE para Intel® Xeon PhiTM. 2017.

3. Gizopoulos D, Papadimitriou G, Chatzidimitriou A, Reddi VJ, Salami B, Unsal OS, et al. Modern Hardware Margins: CPUs, GPUs, FPGAs Recent System-Level Studies. 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS). 2019:129-34. https://doi.org/10.1109/IOLTS.2019.8854386.

4. Madiajagan M, Raj SS. Parallel Computing, Graphics Processing Unit (GPU) and New Hardware for Deep Learning in Computational Intelligence Research. Deep Learning and Parallel Computing Environment for Bioengineering Systems. 2019:1-15. https://doi.org/10.1016/B978-0-12-816718-2.00008-7.

5. Barney B, Frederick D, Livermore C. Introduction to Parallel Computing Tutorial. https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial.

6. Soto RT. Programación paralela sobre arquitecturas heterogéneas. 2016. https://repositorio.unal.edu.co/handle/unal/57830.

7. Tituaña K. Optimización del procesamiento en paralelo utilizando programación heterogénea para mejorar el rendimiento de algoritmos de alto coste computacional. 2024. https://repositorio.utn.edu.ec/handle/123456789/16350.

8. Malagon E, Rojas A. Analysis and simulation of graphs applied to learning with parallel programming in HPC. 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies, CHILECON 2017 - Proceedings. 2017;2017-January:1-7. https://doi.org/10.1109/CHILECON.2017.8229646.

9. Kuzmiakova A. Concurrent, Parallel and Distributed Computing. Arcler Press; 2023.

10. Reinders J, Ashbaugh B, Brodman J, Kinsner M, Pennycook J, Tian X. Data Parallel C++. Apress; 2021. https://doi.org/10.1007/978-1-4842-5574-2.

11. Chirila M, DrAlberto P, Ting H-Y, Veidenbaum A, Nicolau A. A Heterogeneous Solution to the All-pairs Shortest Path Problem using FPGAs. 2022 23rd International Symposium on Quality Electronic Design (ISQED). 2022:108-13. https://doi.org/10.1109/ISQED54688.2022.9806279.

12. Alghamdi MH, He L, Ren S, Maray M. Efficient Parallel Processing of All-Pairs Shortest Paths on Multicore and GPU Systems. IEEE Trans Consum Electron. 2024;70:2896-908. https://doi.org/10.1109/TCE.2023.3327328.

13. Intel Corporation. Intel DevCloud for oneAPI. https://devcloud.intel.com/oneapi/get_started/.

14. Intel Corporation. Intel oneAPI DPC++/C++ Compiler. https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compiler.html.

15. Li X, Sun L, Ling M, Peng Y. A survey of graph neural network based recommendation in social networks. Neurocomputing. 2023;549:126441. https://doi.org/10.1016/j.neucom.2023.126441.

16. Ju W, Fang Z, Gu Y, Liu Z, Long Q, Qiao Z, et al. A Comprehensive Survey on Deep Graph Representation Learning. Neural Netw. 2024;173:106207. https://doi.org/10.1016/j.neunet.2024.106207.

17. Jiao L, Chen J, Liu F, Yang S, You C, Liu X, et al. Graph Representation Learning Meets Computer Vision: A Survey. IEEE Trans Artif Intell. 2023;4:2-22. https://doi.org/10.1109/TAI.2022.3194869.

18. Ji H, Wang X, Shi C, Wang B, Yu P. Heterogeneous Graph Propagation Network. IEEE Trans Knowl Data Eng. 2021;1-1. https://doi.org/10.1109/TKDE.2021.3079239.

19. Li C, Xia L, Ren X, Ye Y, Xu Y, Huang C. Graph Transformer for Recommendation. In: Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY, USA: ACM; 2023. p. 1680-9. https://doi.org/10.1145/3539618.3591723.

20. Intel Corporation. Find the Shortest Path with a Floyd Warshall Algorithm SYCL* Implementation on GPU. https://www.intel.com/content/www/us/en/developer/articles/technical/shortest-path-sycl-based-floyd-warshall-on-gpu.html.

21. Sao P, Lu H, Kannan R, Thakkar V, Vuduc R, Potok T. Scalable All-pairs Shortest Paths for Huge Graphs on Multi-GPU Clusters. In: Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing. New York, NY, USA: ACM; 2021. p. 121-31. https://doi.org/10.1145/3431379.3460651.

22. Huerta EA, Khan A, Davis E, Bushell C, Gropp WD, Katz DS, et al. Convergence of artificial intelligence and high performance computing on NSF-supported cyberinfrastructure. J Big Data. 2020;7:88. https://doi.org/10.1186/s40537-020-00361-2.

23. Naiouf M, De Giusti AE, De Giusti LC, Chichizola F, Sanz VM, Pousa A, et al. Algoritmos paralelos y evaluación de rendimiento en plataformas de HPC. XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021, Chilecito, La Rioja). 2021:674-9.

24. Pusdá-Chulde MR, Salazar-Fierro FA, Sandoval-Pillajo L, Herrera-Granda EP, García-Santillán ID, De Giusti A. Image Analysis Based on Heterogeneous Architectures for Precision Agriculture: A Systematic Literature Review. 2020:51-70. https://doi.org/10.1007/978-3-030-33614-1_4.

25. Vinueza K, Sandoval-Pillajo L, Giret-Boggino A, Trejo-España D, Pusdá-Chulde M, García-Santillán I. Automatic weed quantification in potato crops based on a modified convolutional neural network using drone images. Data and Metadata. 2025;4:194. https://doi.org/10.56294/dm2025194.

26. Sandoval-Pillajo L, García-Santillán I, Pusdá-Chulde M, Giret A. Weed detection based on deep learning from UAV imagery: A review. Smart Agricultural Technology. 2025;12:101147. https://doi.org/10.1016/j.atech.2025.101147.

27. Moreria R, Pusdá-Chulde M, Granda P, García-Santillán I. Early Detection of Missing Plants in Maize Crops Through UAV Imaging. 2024. https://doi.org/10.1007/978-3-031-70760-5_40.

28. Chacua B, Garcia I, Rosero P, Suarez L, Ramirez I, Simbana Z, Pusda M. People Identification through Facial Recognition using Deep Learning. 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI). 2019:1-6. https://doi.org/10.1109/LA-CCI47412.2019.9037043.

29. Montenegro S, Pusdá-Chulde M, Caranqui-Sánchez V, Herrera-Tapia J, Ortega-Bustamante C, García-Santillán I. Android Mobile Application for Cattle Body Condition Score Using Convolutional Neural Networks. 2023. https://doi.org/10.1007/978-3-031-32213-6_7.

30. Ulloa F, Sandoval-Pillajo L, Landeta-López P, Granda-Peñafiel N, Pusdá-Chulde M, García-Santillán I. Identification of Diabetic Retinopathy from Retinography Images Using a Convolutional Neural Network. 2025. https://doi.org/10.1007/978-3-031-75702-0_10.

31. Salazar-Fierro F, Cumbal C, Trejo-España D, León-Fernández C, Pusdá-Chulde M, García-Santillán I. Detection of Scoliosis in X-Ray Images Using a Convolutional Neural Network. 2025. https://doi.org/10.1007/978-3-031-75702-0_13.

32. Guaichico E, Pusdá-Chulde M, Ortega-Bustamante M, Granda P, García-Santillán I. Mobile app for real-time academic attendance registration based on MobileFaceNet Convolutional neural network. Data and Metadata. 2025;4:193. https://doi.org/10.56294/dm2025193.

33. Voloshko A, Ivutin A, Novikov AS. Heuristics for Program Code Optimization in Heterogeneous Systems.

2021 31st International Conference Radioelektronika (RADIOELEKTRONIKA). 2021:1-6. https://doi.org/10.1109/RADIOELEKTRONIKA52220.2021.9420213.

34. Amaral V, Norberto B, Goulão M, Aldinucci M, Benkner S, Bracciali A, et al. Programming languages for data-Intensive HPC applications: A systematic mapping study. Parallel Comput. 2020;91:102584. https://doi.org/10.1016/j.parco.2019.102584.

35. Abdullah EA, Ahmed Saleh I, Al Saif OI. Performance Evaluation of Parallel Particle Swarm Optimization for Multicore Environment. ICOASE 2018 - International Conference on Advanced Science and Engineering. 2018:81-6. https://doi.org/10.1109/ICOASE.2018.8548816.

36. Rangavajhala A, Tadepalli A, Mitra K. Accelerating the Multi-Objective Optimization of Crystallization Process using High Fidelity Population Balance Model Under NUMBA CUDA Environment. 2024 Tenth Indian Control Conference (ICC). 2024:428-33. https://doi.org/10.1109/ICC64753.2024.10883754.

37. Alrawais A. Parallel Programming Models and Paradigms: OpenMP Analysis. Proceedings - 5th International Conference on Computing Methodologies and Communication, ICCMC 2021. 2021:1022-9. https://doi.org/10.1109/ICCMC51019.2021.9418401.

38. Moreno K. Complejidad de un algoritmo (notación Big-O). https://guias.makeitreal.camp/docs/algoritmos/complejidad#complejidad-espacial.

39. Rossainz López M. Programación Concurrente y Paralela. 2020.

FINANCING
The authors did not receive financing for the development of this research.

CONFLICT OF INTEREST
The authors declare that there is no conflict of interest.

AUTHORSHIP CONTRIBUTION
Conceptualization: Katari Tituaña, MacArthur Ortega-Bustamante, Pedro Granda, Marco Pusdá-Chulde.
Data curation: Katari Tituaña, MacArthur Ortega-Bustamante, Pedro Granda, Marco Pusdá-Chulde.
Formal analysis: Katari Tituaña, MacArthur Ortega-Bustamante, Pedro Granda, Marco Pusdá-Chulde.
Research: Katari Tituaña, MacArthur Ortega-Bustamante, Pedro Granda, Marco Pusdá-Chulde.
Methodology: Katari Tituaña, MacArthur Ortega-Bustamante, Pedro Granda, Marco Pusdá-Chulde.
Project management: Katari Tituaña, Marco Pusdá-Chulde.
Resources: Katari Tituaña, MacArthur Ortega-Bustamante, Marco Pusdá-Chulde.
Software: Katari Tituaña, MacArthur Ortega-Bustamante.
Supervision: MacArthur Ortega-Bustamante, Marco Pusdá-Chulde.
Validation: Katari Tituaña, Marco Pusdá-Chulde.
Display: Katari Tituaña, MacArthur Ortega-Bustamante, Pedro Granda, Marco Pusdá-Chulde.
Drafting - original draft: Katari Tituaña, MacArthur Ortega-Bustamante, Marco Pusdá-Chulde.
Writing - proofreading and editing: Katari Tituaña, MacArthur Ortega-Bustamante, Pedro Granda, Marco Pusdá-Chulde.