computers

MDPI

Article

Co-Design of Multicore Hardware and Multithreaded Software for Thread Performance Assessment on an FPGA

George K. Adam

CSLab Computer Systems Laboratory, Department of Digital Systems, University of Thessaly, 41500 Larisa, Greece; gadam@uth.gr; Tel.: +30-2410-684-596

Abstract: Multicore and multithreaded architectures increase the performance of computing systems. The increase in cores and threads, however, raises further issues in the efficiency achieved in terms of speedup and parallelization, particularly for the real-time requirements of Internet of things (IoT)-embedded applications. This research investigates the efficiency of a 32-core field-programmable gate array (FPGA) architecture, with memory management unit (MMU) and real-time operating system (OS) support, to exploit the thread level parallelism (TLP) of tasks running in parallel as threads on multiple cores. The research outcomes confirm the feasibility of the proposed approach in the efficient execution of recursive sorting algorithms, as well as their evaluation in terms of speedup and parallelization. The results reveal that parallel implementation of the prevalent merge sort and quicksort algorithms on this platform is more efficient. The increase in the speedup is proportional to the core scaling, reaching a maximum of 53% for the configuration with the highest number of cores and threads. However, the maximum magnitude of the parallelization (66%) was found to be bounded to a low number of two cores and four threads. A further increase in the number of cores and threads did not add to the improvement of the parallelism.

Keywords: multicore; multithreading; performance evaluation; real-time systems

Citation: Adam, G.K. Co-Design of Multicore Hardware and Multithreaded Software for Thread Performance Assessment on an FPGA. *Computers* **2022**, *11*, 76. https://doi.org/10.3390/computers11050076

Academic Editor: Paolo Bellavista

Received: 17 April 2022
Accepted: 6 May 2022
Published: 9 May 2022

1. Introduction

The continuous evolution of electronic fabrication technologies, which allows SoCs (system-on-chip circuits) to integrate more and more powerful processing and communication architectures in a single device, has led to the mass production of consumer electronic devices providing many functionalities, particularly for applications in Internet of things (IoT), ubiquitous, cloud, and edge computing. The architecture of such systems can be considered to consist of one or more programmable elements (multicore processors, microcontrollers, FPGAs) acting as the master control units and interacting with several peripherals to perform a set of real-time tasks. Today, embedded applications quite often demand more critical control, often requiring simultaneous execution of multiple tasks in a real-time environment. A real-time OS (RTOS) provides support for such requirements, since time-critical data processing is performed in high-priority tasks with deterministic execution times.

With the advent of multicore architectures, the research interest in applications running multiple threads in parallel has increased [1,2]. Multicore and multithreaded architectures increase system performance by providing high-level parallelism on multithreaded applications. Despite their considerable impact, their efficient utilization is still under research, due to the continuous increase in the number of CPU cores. As Amdahl's law states [3], this multicore technology has efficiency limits [4]. However, further improvements could be made in the organization of the multithreaded software applications executed in multicore systems.

FPGAs have taken advantage of this multicore technological evolution. Most of the FPGA vendors have concentrated their efforts on providing devices with reconfigurable soft-core processors, combined with hard-core multicore processors, on homogeneous or

heterogeneous multicore architectures, with symmetric multiprocessing (SMP) or asymmetric multiprocessing (AMP). Reconfigurable devices such as FPGAs are widely used as design platforms for experimental research in investigating the performance and efficiency of multicore systems [5]. FPGAs are capable of working with diverse systems and applications that require parallel processing of large amount of data in real time. These platforms provide operating systems with real-time support for time-constrained applications. One of the most adopted solutions for real-time support in Linux OS is the PREEMPT_RT patch [6]. Currently, the PREEMPT-RT patch is actually mostly mainlined in the Linux kernel versions since v2.6.11. Real-time embedded devices often require multiple applications to be executed in parallel and/or concurrently, in order to satisfy strict timing requirements [7,8]. However, the performance gain is not always evident.

Flexibility is one of the main advantages of architectures based on soft processors, enabling the development of custom solutions to meet design requirements. In addition, their scalability enables the addition of new resources and the possibility of replicating existing system components, e.g., implementing more than one processors/cores in the same FPGA chip. These advantages were exploited during the design of the proposed multicore FPGA architecture. The objective of this research work was to investigate the efficiency of a multicore FPGA-based architecture with real-time OS support, to exploit the TLP of real-time tasks running in parallel on multiple cores. Toward this direction, the main contributions of this research are summarized as follows:

1. The design of a 32-core (NIOS II/f soft cores) reconfigurable architecture with embedded real-time Linux support (kernel version 4.9 patched with PREEMPT_RT), used for the assessment of multithreaded applications running recursively sorting algorithms in parallel.
2. An approach for the performance evaluation of a soft multithreaded multicore architecture conducted in real time on an FPGA, based on recursive generation and execution of the popular sorting algorithms merge sort and quicksort.
3. A prototype of the proposed architecture implemented on the commercially available Altera DE2-115 device featuring a Cyclone IV FPGA chip.

The research investigates the performance efficiency and timing of parallel execution using multiple threads. The POSIX (Portable Operating System Interface) thread programming interface in C was used to enable and control thread and data parallelism [9]. The experiments included a number of core and thread combinations under different workloads (merge sort and quick sort algorithms) and datasets (arrays of 32-bit integers). Both sorting algorithms utilize data partitioning in dividing data in subsets and recursively sorting, which makes them appropriate for parallelization. Performance metrics included execution time, speedup, and parallelization efficiency.

The remainder of the paper is organized as follows: first, related work is discussed in Section 2. Section 3 describes the methodology followed in the design and evaluation of the proposed multicore FPGA architecture. Section 4 presents the performance evaluation metrics used to assess the proposed design. Section 5 presents and discusses the experimental results obtained from running the multithreaded application software in the experimental FPGA platform. Lastly, Section 6 concludes the paper and proposes future work.

2. Related Work

In order for multicore architectures to cope with continuous demands for real-time computational requirements, parallelism is one of the techniques most exploited, i.e., the ability of a system to execute several tasks as threads (multithreading) running concurrently on each core and in parallel on multiple cores. In multicore architectures, memory resources and peripherals are usually shared among the processors (shared memory systems). Such architectures have many performance benefits, including increased parallelization efficiency and execution speedup. However, increasing the number of cores and threads also presents some challenges, such as process scheduling and thread parallelization issues. The hardware-based multithreading capabilities on FPGA platforms targeting multithreaded

workloads are under further research. One of the main research interests lays in the parallelization efficiency and speedup that can be achieved by multithreaded applications, in relation to the number of threads and soft cores utilized.

Certain research approaches are based on reconfigurable hardware such as FPGAs, which can be dynamically adapted to specific multicore configurations [10,11]. In particular, several multicore architectures with Nios II soft processors have been proposed for design and testing [12–16]. In the work of Muttillo et al. [17], the authors presented an interesting design methodology for soft-core platforms on FPGA with SMP Linux, based on the LEON3 32-bit synthesizable soft-processor [18], but without real-time OS support.

Real-time operating systems have also been employed to provide support to system applications running tasks with timing constraints, particularly in embedded systems [19,20]. However, very few studies have employed real-time operating systems in reconfigurable hardware to provide real-time support to multithreaded applications. As an Aspencore study indicated [21], most hardware makes use of Embedded Linux and open-source real-time operating systems, such as µClinux [22], uC/OS-III [23], or FreeRTOS [24], which target microcontrollers without MMU support [25–27]. The work of Fradi et al. [28] presented such an approach of using µClinux, designed for processors without MMU, in the design of a system on programmable chip (SoPC), based on the Nios II processor, to facilitate the implementation of an image processing application. Other custom microarchitectures and hardware implementations for real-time operating systems/schedulers such as those provided by Renesas [29] or nMPRA and nHSE microarchitectures [30] could have been used to improve timing metrics.

The performance of multicore FPGA systems and multithreaded applications has been analyzed using many different approaches [31–33]. Although the techniques and tools used depend on the aspects of performance that are targeted for assessment, the most commonly used are benchmarks for efficiency metrics [34–36]. The work of Baklouti and Abid [37] presented the design of a parallel multicore system based on the Nios II soft processor, the performance of which was tested in terms of speedup and efficiency. Another study by Azarian and Cardoso [38] presented an FPGA-based multicore architecture for pipelined task execution, evaluated with benchmarks. However, there are currently almost no parallel thread applications available for real-time multicore systems, qualifying as benchmarks.

Despite the considerable research efforts in the design and performance assessment of multicore architectures and multithreaded applications, there has not been much work on using an FPGA platform with MMU and real-time OS support. Our research has some common elements with the above studies; however, in contrast, the performance of multithreaded applications running recursively in parallel on a 32-core FPGA architecture with MMU and real-time Linux OS support was investigated. Performance evaluation was based upon recursively generated multithreaded sorting tasks, running in parallel in the 32-core FPGA architecture.

3. Design Methodology

The co-design of the proposed system encompassed hardware and software parts. The hardware part was a reconfigurable 32-core architecture synthesized into an FPGA configuration, and the software part comprised a multithreaded application software.

3.1. Proposed System Architecture

The proposed multicore 32-core FPGA architecture was based upon a design for the Cyclone III series under the Intel license agreement [39]. This design was restructured and reconfigured to work with a Cyclone IV FPGA chip used in this research. The result reconfirms the fact that soft processors can usually easily be migrated to new families of devices. Reconfigurable architectures, due to their flexibility, allow to prototype complete systems, from small logic circuits to complex architectures involving multiple processors, buses, and many other devices. In addition, such an architecture has the ability to provide massive concurrency with a comparatively low energy consumption.

The Quartus software and Qsys tool were used to redesign and configure this Cyclone III-based architecture to work with the Cyclone IV EP4CE115F29 FPGA chip. The final system integrated a real-time Linux OS that supports MMU. This system architecture consisted of 32 NIOS II/f soft cores with MMU enabled, on-chip memory, flash, and SDRAM memory. The Altera's Nios II soft processor is a general-purpose RISC soft processor with 32-bit instruction words and datapaths. The Nios II/f (fast) variant is a single issue in-order execution processor, optimized for performance.

The FPGA's soft cores were implemented using the distributed logic resources and the specialized hardware blocks of the FPGA fabric, as well as its interconnection resources. One of the soft cores (core 0), set as the master core, ran Linux with real-time support and was responsible for booting and handling the multithreaded application's task allocations to the remaining cores configured as slaves. The master core used a separate memory containing both instructions and data. It was also responsible for communication with a host computer and synchronization of tasks running on each slave core. A JTAG UART was used to download the application program and data from the host computer into the corresponding on-chip memory.

The development platform used to implement this architecture was Altera's DE2-115 development board from Terasic Inc., Hsinchu, Taiwan [40]. This board includes a Cyclone IV FPGA chip and several other components, which support various types of applications. The Cyclone IV FPGA has a sufficient capacity of 114,480 logic elements (LEs) and 4 MBits of on-chip memory. The implementation of the 32-core configuration consumed about 30% of the FPGA's available logic elements. The architecture also supports multiprocessing and has several software tools that assist in the development of the multithreaded applications.

A schematic representation of the 32-core system architecture is shown in Figure 1.

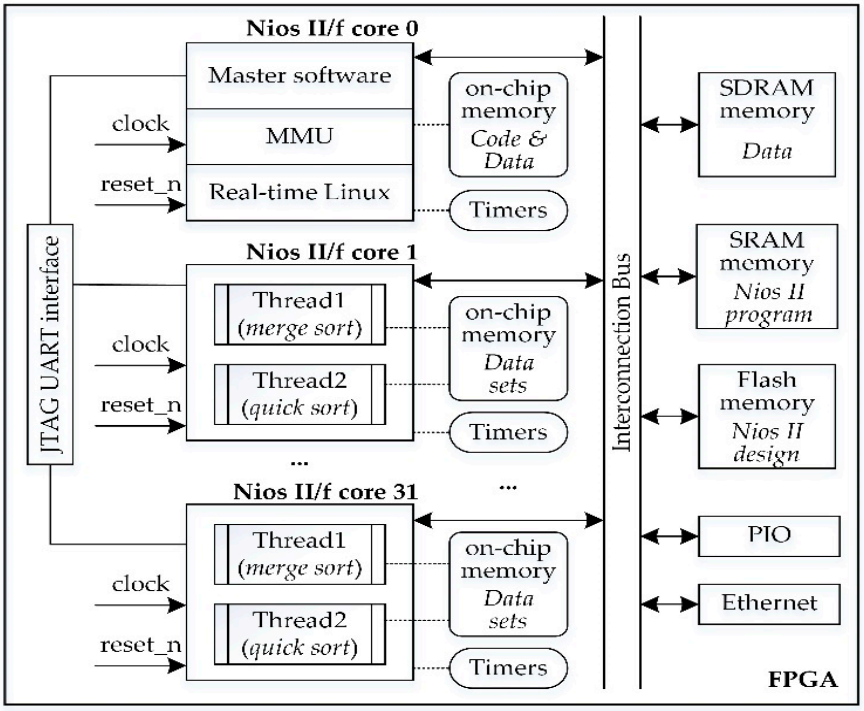

Figure 1. The 32-core system architecture.

3.2. Software Infrastructure

An instance of Linux kernel (4.9.178-rt131 with real-time support -patched with PREEMPT_RT-), intended for microcontrollers with MMUs, ran on the master NIOS II soft core. The Nios II Linux kernel was integrated in this design to provide support for real-time multithreaded applications running in this Nios II soft-core architecture. The supported kernel was based on Altera's open-source Linux provided on GitHub [41]. This

provides an adequate environment to deploy timely real-time application tasks, requiring a deterministic response (mostly interactive applications), for instance, in performance critical applications in industrial Internet of things (IIoT) systems.

The development of the multithread control software applied for testing performance metrics was based upon the use of the POSIX user level threads library in C. PThreads is an execution model that exists independently from a language and a parallel execution model. POSIX Threads is an API defined by the standard IEEE POSIX.1c. In the proposed shared memory 32-core FPGA architecture, PThreads was used to implement parallelism.

The FPGA's soft cores were designated to run the multithreaded application's two sorting tasks as two separate threads, concurrently on each core, and in parallel in multiple-core configurations. Each core had its own part of on-chip memory. Input data for each thread were placed into the same on-chip memory section. All cores shared components such as SDRAM memory and other peripherals in a hierarchical mode with the master core. Resource sharing is one of the most important advantages of shared memory architectures. The fact that all cores share a common space reduces the need for modifications in data or control structures. After execution is completed, each core writes its own data (performance measurements results) to the shared memory.

The hardware and software co-design flow and system infrastructure are shown in Figure 2.

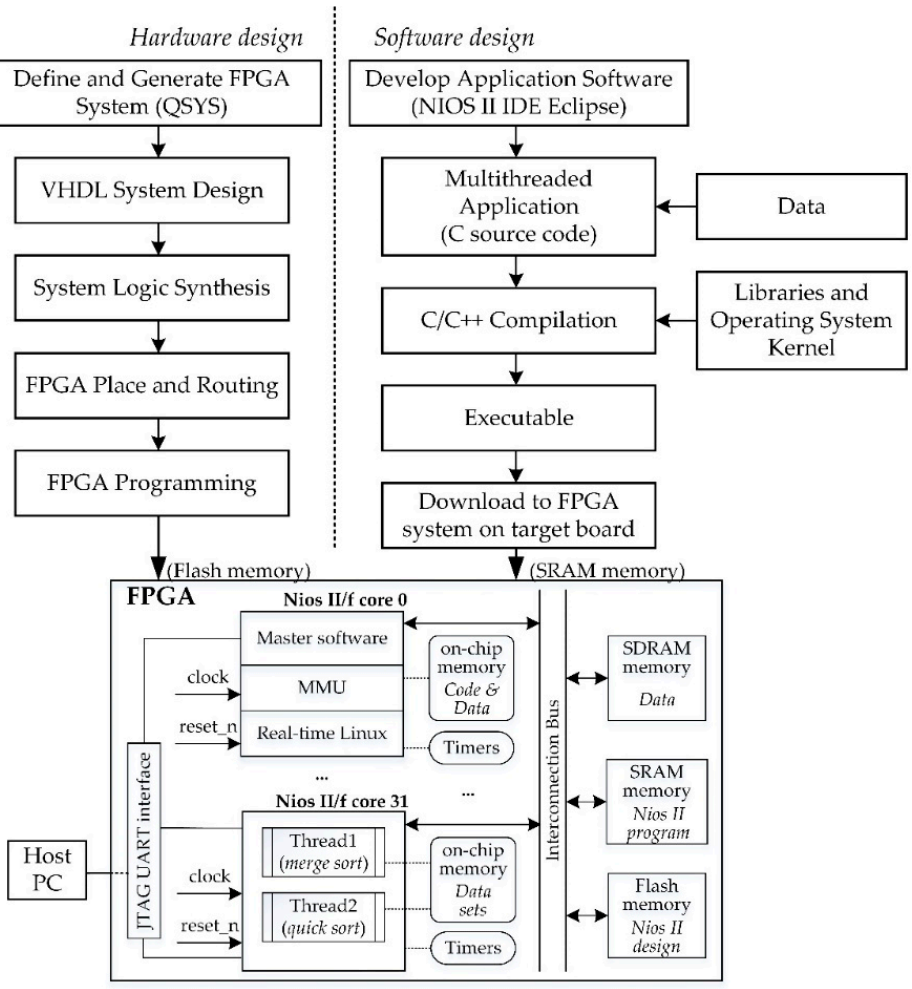

Figure 2. Hardware and software co-design flow and system infrastructure.

Such an infrastructure was oriented to get the most possible advantage of parallelism to maximize processing performance. However, a limiting factor was related to the dynamic distribution of the task data. This could affect the ability of the system to provide a predictable timing response, which is a requirement in real-time applications. For this purpose, the application in control allocated data for each task in its own on-chip memory, as a read-only sharing resource. Each task retrieved data in the order in which the processing was required, independently from each other, without affecting the other task's execution.

Figure 3 gives a partial view of the configuration of the Nios II-based system.

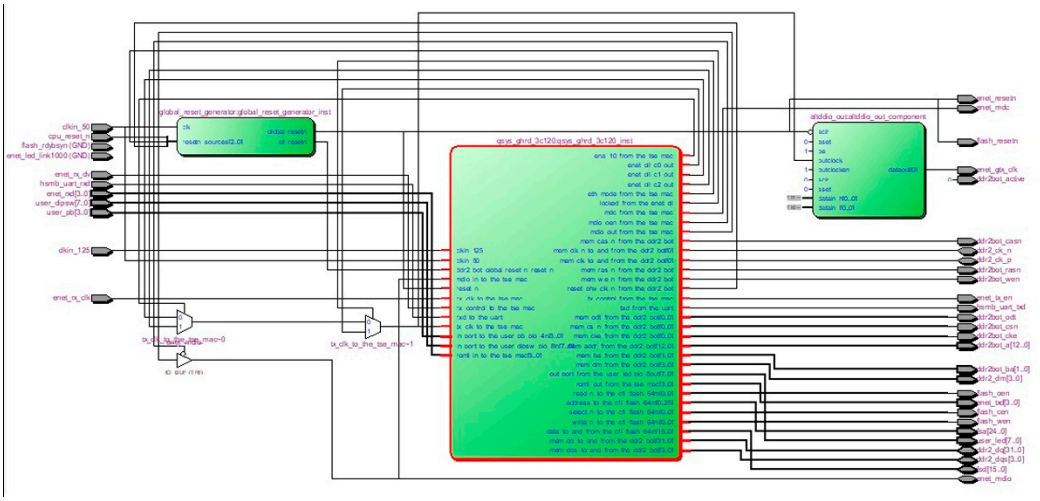

Figure 3. RTL partial view of the Nios II-based architecture.

3.3. Multithreaded Application

One of the approaches used to efficiently exploit an application's parallelism is to partition the application into multiple threads and execute them in parallel. In a multithreaded application, each process was configured and assigned to run in parallel on multiple cores on the FPGA. It was efficiently structured as multiple independent threads of control that perform different sorting tasks running in parallel on multiple cores. Creation and controlling of threads were achieved by making calls to the POSIX Threads API, in order to set the threads real-time parallel execution features, including scheduling policy, CPU affinity, and timing.

Each process scheduled the two sorting tasks (merge and quick sort) as software threads (SW thread1 and SW thread2). Multiple threads were created recursively and executed concurrently on each core, as well as in parallel on multiple cores. The threads accessed common data for sorting operations, merge sort and quick sort, through shared FPGA SDRAM memory. Nevertheless, data movement between cores would affect the thread performance metrics, such as execution time. For this purpose, data arrays of 32-bit integers of size 10^2 to 10^6 elements were allocated for each core in the on-chip memory to support the concurrent execution. Sorted data output was written out in the same memory space during the sorting operations, by overwriting the input. Both algorithms utilize data partitioning in dividing data in subsets for their recursive sorting. This functionality makes them appropriate for parallelization.

Thread scheduling can be performed at different levels of the OS, and even outside of it. Linux implements scheduling at the kernel level. In our study, thread scheduling was implemented by the multithreaded application itself. The application exploited the capabilities offered by the underlying Linux real-time OS scheduler to manage thread migration and placement among the cores. In particular, in terms of thread scheduling, the POSIX standard defined scheduling policies, thread attributes, and system calls that the multithreaded application called to interact with the scheduler.

The multithreaded application equally distributed and allocated the workload of sorting computations and partitioned data to each core for optimal balance. The merge and quicksort algorithms were implemented on the basis of a recursive approach upon which threads were created until the assigned thread's count was achieved (e.g., 2^1 to 2^6 threads). The threads were configured with high priority and SCHED_FIFO real-time policy. The SCHED_FIFO scheduling policy is mainly used for real-time applications. Priority reflects the importance of a thread and, therefore, the relative urgency of a thread to be scheduled compared to others. In real-time contexts, a usually desirable property is the respect of deadlines. Real-time applications perform tasks that must complete before a given time called a deadline. Linux patched with PREEMPT_RT provides such real-time support and guarantees maximal bounds on scheduling delays. Threads were scheduled as real-time threads with the highest real-time priority (priority level: 99).

Figure 4 illustrates the execution flow of the multithreaded application.

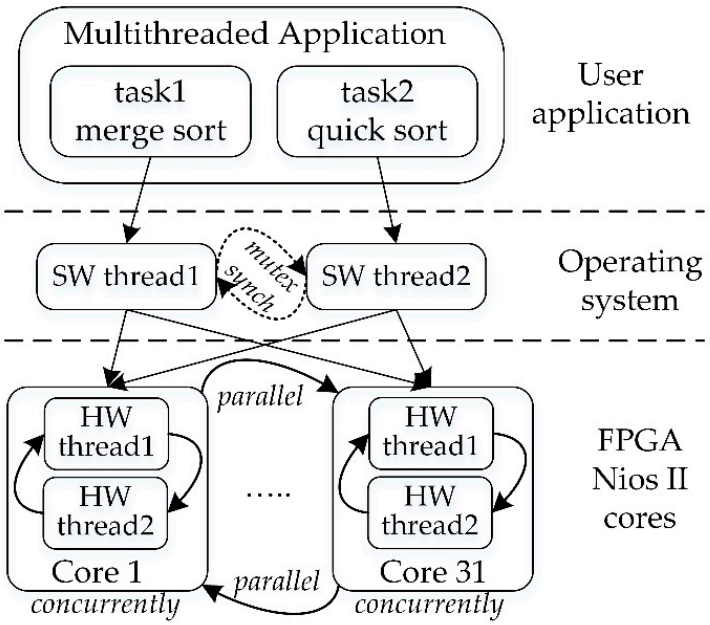

Figure 4. Execution flow of the multithreaded application.

4. Performance Evaluation

4.1. Performance Evaluation Metrics

Performance analysis and evaluation commonly involve benchmarking and empirically measuring performance methods and techniques. A typical method to assess the performance of multicore computing systems is through the parallel scaling behavior of multithreaded applications. The sequential execution of an application serves as the baseline that is used as a reference for analysis and comparisons of parallel threads' executions on multiple cores. The level of thread parallelism substantially impacts the performance of the multithreaded applications through various factors, ranging from hardware-specific and thread implementation-specific to application-specific.

4.1.1. Execution Time

Each thread may be thought of as an independent task having its own memory stack and instructions. The threads are spread out in parallel sets among the available cores reducing the execution time to the maximum execution time of any of the threads, compared to sequential execution version. The work is evenly divided among threads, and the overhead of allocating and scheduling threads is minimal. The time it takes for a thread to execute a single task is essential in performance measurements. The real time is the difference between the end time and the begin time. Therefore, an average value of the

total execution time (t_{exec}) for a given number of execution runs (iterations) is estimated by the following equation:

$$t_{exec} = t_{end} - t_{begin}, \quad (1)$$

where t_{end} is the time it takes to finish the task's execution, and t_{begin} is the initial time the execution is started.

4.1.2. Speedup

Speedup is defined as the execution time of a sequential program divided by the execution time of a parallel program that computes the same algorithm running on multiple cores. According to a given number of execution runs, an average speedup (Sp_{avg}) is calculated on the basis of the following equation:

$$Sp_{avg} = \sum_{i=1}^{n} 100 - \frac{t_{p(i)} \times 100}{t_{seq}(i) \times n}, \quad (2)$$

where t_{seq} is the time it takes to sequentially execute the workload on a single core, t_p is the time it takes to execute the workload in parallel on multiple cores, and n is the number of performed iterations.

4.1.3. Parallelization Efficiency

In multicore architectures, it is essential to know the effectiveness of the parallelization of thread execution on multiple cores. Parallelization efficiency (f_p) measures the time for which a processor is fully utilized. A higher efficiency indicates better utilization of the processors. This is calculated using the following equation:

$$f_p = \frac{t_{seq}}{t_p(n_c) \times n_c}, \quad (3)$$

where t_{seq} is the time it takes to sequentially execute a program on a single core, and t_p is the time it takes to execute a program in parallel on n_c cores.

5. Experimental Results and Discussion

5.1. Evaluation Results

A number of experiments were carried out on an ALTERA DE2-115 development board to evaluate the performance of the proposed soft-core reconfigurable architecture and verify its feasibility for multithreading research. Initially, the application was executed sequentially on a single core with a pair of two threads. Then, the execution proceeded in parallel with the number of threads varying from 4 to 62 on multiple core configurations of 2 to 31 cores, respectively. Both merge and quicksort algorithms were set to sort in ascending order datasets of 32-bit integers of size 10^2 to 10^6 (100 to 1 M) elements.

When running the application sequentially on a single core with two threads, the processor's affinity was set to that specific single core. Thus, each core was shared by at least two threads running concurrently. Although one-to-one thread-core mapping is the most obvious choice to obtain best performance, this running scheme was chosen in order to emulate a commonly adopted by default SMT execution mode [42]. In addition, pairing of threads can be highly beneficial for the workload. For example, two threads working on the same data benefit from sharing caches, thus diminishing the number of on-chip memory accesses performed. In a real-time context, reducing contention over shared resources can allow threads to respect their deadlines more easily.

The results were used as the baseline for comparisons with parallel executions on multiple cores, as well as calculation of the speedup and parallelization efficiency. In the case of parallel execution, the data and sorting operations were equally partitioned and distributed among the threads. Each pair of threads ran on a distinct core, processing a

data subset. Thus, threads were executed in parallel with each other and concurrently on each core.

The experiments were executed for a number of repetitions, approximately for a few thousand iterations, to obtain sufficient values for average estimation. Table 1 presents the execution time results for merge and quicksort algorithms for different datasets and different thread and core variations.

Table 1. Merge and quicksort execution times.

Cores	Threads	Sort Alg	Datasets				
			10^2	10^3	10^4	10^5	10^6
			Execution Times (ms)				
1	2	Merge	3.2	6.5	6.12	8.56	40
		Quick	1.08	1.1	2.79	79	2530
2	4	Merge	2.2	3.31	3.25	4.1	38
		Quick	1.01	1.03	2.78	80	2532
4	8	Merge	1.51	1.71	1.83	3.91	37
		Quick	1	1.04	2.8	79	2525
8	16	Merge	0.9	0.98	1	2.51	39
		Quick	0.95	1.05	2.9	70	2510
16	32	Merge	0.52	0.6	0.7	2.22	24
		Quick	0.61	0.91	2.63	67	2480
31	62	Merge	0.25	0.4	1.2	2.01	22
		Quick	0.45	0.62	1.8	61	2472

The results show the differences in structure and complexity of each sorting algorithm. The merge sort algorithm achieved lower execution times than quicksort, particularly for large datasets ($\geq 10^5$). Table 2 presents the average execution times for both algorithms. What we can see is that the total execution times decreased as the number of threads increased.

Table 2. Average execution times.

		Datasets				
		10^2	10^3	10^4	10^5	10^6
Cores	Threads	Average Execution Times (ms)				
1	2	2.14	3.80	4.46	43.78	1285
2	4	1.61	2.17	3.02	42.05	1283
4	8	1.26	1.38	2.32	41.46	1281
8	16	0.93	1.02	1.95	36.26	1274
16	32	0.57	0.76	1.67	34.61	1252
31	62	0.35	0.51	1.50	31.51	1247

As depicted in Figure 5, data sorting was more efficient as the number of threads increased, particularly for large sets of data ($\geq 10^4$).

Table 3 presents the average speedup and parallelization efficiency calculated for each combination of cores and threads.

Table 3. Average speedup and parallelization.

Cores	Threads	Average Speedup (%)	Average Parallelization (%)
1	2	-	-
2	4	20.8	66.0
4	8	31.8	42.3
8	16	40.9	26.4
16	32	48.0	17.2
31	62	53.5	12.2

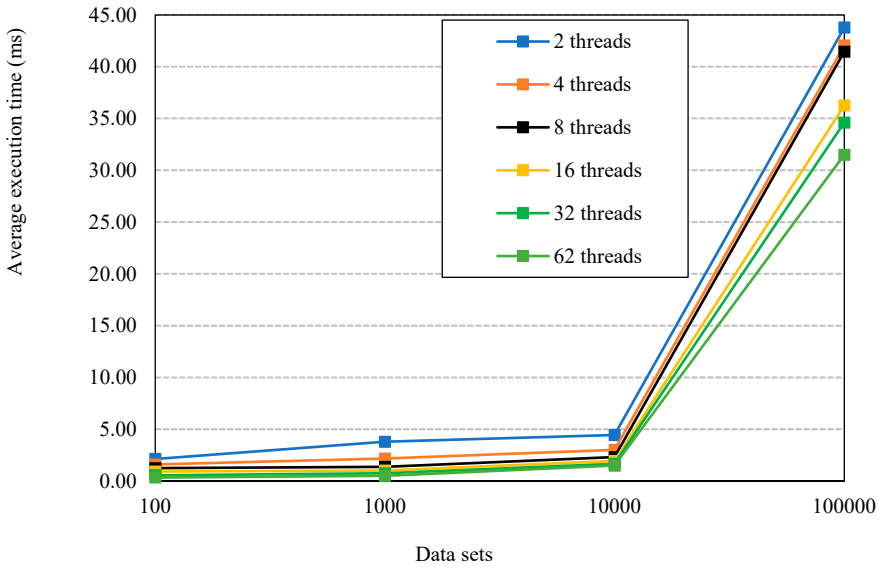

Figure 5. Average execution times in relation to datasets and cores.

As illustrated in Figure 6, each additional core augmented performance, i.e., the speedup increased. However, there was a limit to the parallelization improvement, due to that part of the application which could not be further parallelized. As shown in Figure 6, the maximum parallelization performance (66%) of the threads was bound to four threads and two cores. A further increase in the number of cores and pairs of threads did not increase significantly the overall performance.

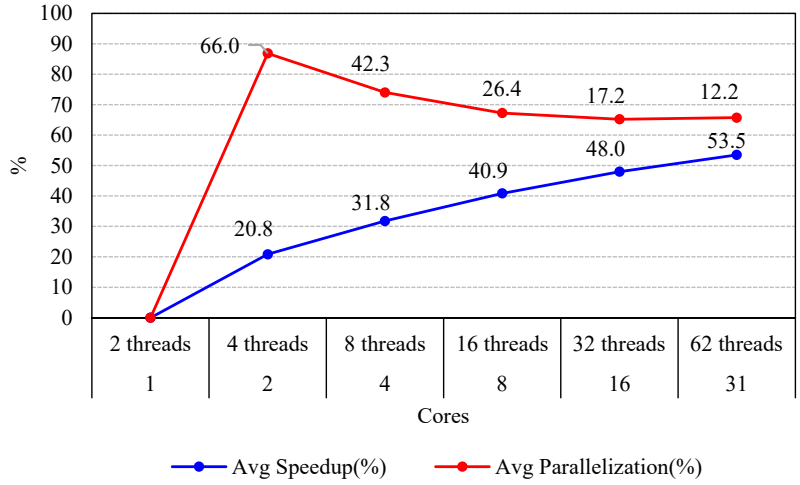

Figure 6. Average speedup and parallelization efficiency.

5.2. Discussion

This research exploited the performance offered by today's soft-core processors built on FPGAs and having a real-time OS with MMU support, especially when running parallel multithreaded tasks. Experiments were conducted upon the development and implementation of a 32-core architecture on an FPGA with MMU and Linux real-time support. The use of Linux patched with PREEMPT_RT in real-time support of multithreaded sorting tasks running in parallel on FPGA architectures seems to be one of the first research efforts of its kind. In this study, both merge and quicksort algorithms utilized data partitioning in dividing data in subsets for their recursive sorting. This functionality made them appropriate for parallelization. However, future research can utilize a variety of sorting algorithms.

Overall, the results verified the validity and efficiency of the proposed FPGA design in testing and evaluating the performance of multithreaded applications. In particular, the results provided insight into thread performance issues related to speedup and parallelization efficiency. Certainly, the parallel implementation of merge sort and quicksort algorithms on multiple cores was more efficient than sequential implementation on a single core. It is interesting that the parallelization efficiency did not improve at the same rate as the speed of the execution time and the speedup. The increase in speedup was proportional to core scaling, reaching a maximum of 53% for the configuration with the highest number of cores and threads (31 cores and 62 threads). However, the maximum magnitude of parallelization (66%) was found to be bounded to a low number of cores and threads (four threads and two cores). It seems that this was limited by the part of the task that could not benefit from the parallelization improvement. Indeed, as explained by Amdahl's law, even a small part with limited thread level parallelism can impose important constraints to the performance.

Previous work was primarily focused on the design and performance assessment of multicore architectures and multithreaded applications on an FPGA platform without MMU and real-time OS support. This work used a multicore FPGA-based architecture with MMU and real-time OS support. In addition, in contrast to studies that employed reconfigurable hardware with MMU and real-time support, with most commonly using benchmarks for efficiency metric assessment, in this work, performance evaluation was based on recursively generated multithreaded sorting applications. This is because there are currently almost no parallel thread applications available for real-time multicore systems, qualifying as benchmarks.

6. Conclusions

The design and implementation of a prototype 32-core architecture on a low-cost FPGA with MMU and Linux OS with real-time support was implemented and used to perform experimental performance tests with a multithreaded sorting application.

The proposed multicore FPGA architecture proved to be effective in exploiting the performance of low-threaded applications (up to 64 instances of threads) on a shared memory framework. The maximum magnitude of the parallelization (66%) was found to be bounded to a low number of cores and threads (four threads and two cores). The research outcomes confirmed the feasibility and validity of the proposed design approach in using multiple soft processors implemented on an FPGA, to execute and evaluate multithreaded tasks with real-time support, and to gain more insight into multicore FPGA multithreading.

Regarding future work, it seems interesting to extend this research with outcomes obtained from comparisons with appropriate real-time benchmarks. Furthermore, current research is exploring the use of distributed memory to model thread performance on microprocessors implemented in several FPGAs.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data has been presented in main text.

Acknowledgments: The author would like to thank the Computer Systems Laboratory (CSLab, https://cslab.ds.uth.gr/) (accessed on 30 April 2022) in the Department of Digital Systems, University of Thessaly, Greece, for the technical support and the resources provided for this experimental research.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Radojkovic, P.; Carpenter, P.M.; Moreto, M.; Cakarevic, V.; Verdu, J.; Pajuelo, A.; Cazorla, F.J.; Nemirovsky, M.; Valero, M. Thread Assignment in Multicore/Multithreaded Processors: A Statistical Approach. *IEEE Trans. Comput.* **2016**, *65*, 256–269. [CrossRef]
2. Fernando, E.; Murad, D.F.; Wijanarko, B.D. Classification and Advantages Parallel Computing in Process Computation: A Systematic Literature Review. In Proceedings of the IEEE International Conference on Computing, Engineering, and Design (ICCED), Bangkok, Thailand, 8 September 2018; pp. 143–147.
3. Amdahl, G.M. Computer Architecture and Amdahl's Law. *IEEE Comput.* **2013**, *46*, 38–46. [CrossRef]
4. Hill, M.D.; Marty, M.R. Amdahl's law in the multicore era. *Computer* **2008**, *41*, 33–38. [CrossRef]
5. Nane, R.; Sima, V.; Pilato, C.; Choi, J.; Fort, B.; Canis, A.; Chen, Y.T.; Hsiao, H.; Brown, S.; Ferrandi, F.; et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2016**, *35*, 1591–1604. [CrossRef]
6. The Linux Foundation Wiki: Real Time Linux. Available online: https://wiki.linuxfoundation.org/realtime/start (accessed on 7 December 2020).
7. Wang, J. *Real-Time Embedded Systems*, 1st ed.; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2017.
8. Sheikh, S.Z.; Pasha, M.A. Energy-Efficient Scheduling for Hard Real-Time Systems: A Survey. *ACM Trans. Embed. Comput. Syst.* **2018**, *17*, 1–26. [CrossRef]
9. Severance, C. Posix: A model for future computing. *IEEE Comput.* **1999**, *32*, 131–132. [CrossRef]
10. Gaillardon, P.-E. *Reconfigurable Logic: Architecture, Tools, and Applications*, 1st ed.; CRC Press: Boca Raton, FL, USA, 2016. [CrossRef]
11. Kirchhoff, M.; Kerling, P.; Streitferdt, D.; Fengler, W. A Real-Time Capable Dynamic Partial Reconfiguration System for an Application-Specific Soft-Core Processor. *Int. J. Reconfig. Comput.* **2019**, *2019*, 4723838. [CrossRef]
12. Cardoso, J.; Hubner, M. *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*; Springer: New York, NY, USA, 2011. [CrossRef]
13. Yan, L.; Wu, B.; Wen, Y.; Zhang, S.; Chen, T. A reconfigurable processor architecture combining multi-core and reconfigurable processing units. *Telecommun. Syst.* **2014**, *55*, 333–344. [CrossRef]
14. Vanderbauwhede, W.; Benkrid, K. *High-Performance Computing Using FPGAs*; Springer: New York, NY, USA, 2014. [CrossRef]
15. Chouliaras, V.A.; Stevens, D.; Dwyer, V.M. VThreads A novel VLIW chip multiprocessor with hardware-assisted PThreads. *Microprocess. Microsyst.* **2016**, *47*, 466–485. [CrossRef]
16. Hassanein, A.; El-Abd, M.; Damaj, I.; Rehman, H. Parallel Hardware Implementation of the Brain Storm Optimization Algorithm using FPGAs. *Microprocess. Microsyst.* **2020**, *74*, 103005. [CrossRef]
17. Muttillo, V.; Valente, G.; Federici, F.; Pomante, L.; Faccio, M.; Tieri, C.; Ferri, S. A design methodology for soft-core platforms on FPGA with SMP Linux, OpenMP support, and distributed hardware profiling system. *EURASIP J. Embed. Syst.* **2016**, *2016*, 15. [CrossRef]
18. LEON3 Processor. Available online: http://www.gaisler.com/index.php/products/processors/leon3 (accessed on 6 December 2021).
19. Wang, K.C. *Embedded and Real-Time Operating Systems*; Springer: Cham, Switzerland, 2017.
20. Seo, S.; Kim, J.; Kim, S.M. An Analysis of Embedded Operating Systems: Windows CE Linux VxWorks uC/OS-II and OSEK/VDX. *Int. J. Appl. Eng. Res.* **2017**, *12*, 7976–7981.
21. Aspencore: 2019 Embedded Markets Study. Available online: www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf (accessed on 16 January 2022).
22. µClinux. Available online: https://en.wikipedia.org/wiki/%CE%9CClinux (accessed on 7 November 2021).
23. MicroC/OS: Micro-Controller Operating Systems. Available online: https://en.wikipedia.org/wiki/Micro-Controller_Operating_Systems (accessed on 2 December 2021).
24. FreeRTOS. Available online: https://en.wikipedia.org/wiki/FreeRTOS (accessed on 12 December 2021).
25. Zhu, S.-H. Hardware Implementation based on FPGA of Semaphore Management in µC/OS-II real-time operating system. *Int. J. Grid Util. Comput.* **2015**, *6*, 192–199. [CrossRef]
26. Matthews, E.; Shannon, L.; Fedorova, A. Shared Memory MicroBlaze System with SMP Linux Support. *ACM Trans. Reconfig. Technol. Syst.* **2016**, *26*, 1–22. [CrossRef]
27. Hahm, O.; Baccelli, E.; Petersen, H.; Tsiftes, N. Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet Things J.* **2016**, *3*, 720–734. [CrossRef]
28. Fradi, M.; Youssef, W.E.; Mohsen, M. The design of an embedded system (SOPC) for an image processing application. In Proceedings of the International Conference on Control, Automation and Diagnosis (ICCAD), Hammamet, Tunisia, 19–21 January 2017; pp. 511–515. [CrossRef]

29. Renesas Electronics Corporation. Microcontrollers & Microprocessors (MCUs, MPUs). 2022. Available online: www.renesas.com/us/en/products/microcontrollers-microprocessors (accessed on 30 April 2022).
30. Găitan, V.G.; Zagan, I. An Overview of the nMPRA and nHSE Microarchitectures for Real-Time Applications. *Sensors* **2021**, *21*, 4500. [CrossRef]
31. Iordanou, K.; Nikolakaki, S.M.; Malakonakis, P.; Dollas, A. A performance evaluation of multi-FPGA architectures for computations of information transfer. In Proceedings of the 18th ACM International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '18), New York, NY, USA, 15–19 July 2018; pp. 1–9. [CrossRef]
32. Belleza, R.R.; Freitas, E.P. Performance study of real-time operating systems for internet of things devices. *IET Softw.* **2018**, *12*, 176–182. [CrossRef]
33. Adam, G.K. Real-Time Performance and Response Latency Measurements of Linux Kernels on Single-Board Computers. *Computers* **2021**, *10*, 64. [CrossRef]
34. Shannon, L.; Matthews, E.; Doyle, N.; Fedorova, A. Performance Monitoring for Embedded Computing Systems on FPGAs. In Proceedings of the 2nd International Workshop on FPGAs for Software Programmers (FSP), London, UK, 1 September 2015; pp. 68–72.
35. Podobas, A.; Sano, K.; Matsuoka, S. A Survey on Coarse-Grained Reconfigurable Architectures. From a Performance Perspective. *IEEE Access* **2020**, *8*, 146719–146743. [CrossRef]
36. Meyer, M.; Kenter, T.; Plessl, C. In-depth FPGA accelerator performance evaluation with single node benchmarks from the HPC challenge benchmark suite for Intel and Xilinx FPGAs using OpenCL. *Parallel Distrib. Comput.* **2022**, *160*, 79–89. [CrossRef]
37. Baklouti, M.; Abid, M. Multi-Softcore Architecture on FPGA. *Int. J. Reconfig. Comput.* **2014**, *2014*, 979327. [CrossRef]
38. Azarian, A.; Cardoso, J.M.P. Pipelining Data-Dependent Tasks in FPGA-Based Multicore Architecture. *Microprocess. Microsyst.* **2016**, *42*, 165–179. [CrossRef]
39. Intel Corporation: Nios II Processor with Memory Management Unit Design Example. Available online: https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/intellectual-property/embedded/nios-ii/exm-mmu.html (accessed on 19 October 2021).
40. Altera DE2-115 Development and Education Board. Available online: www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=502&PartNo=2 (accessed on 1 October 2021).
41. GitHub, Inc., Linux Development Repository for Socfpga. 2020. Available online: https://github.com/altera-opensource/linux-socfpga (accessed on 2 October 2021).
42. Tullsen, D.M.; Eggers, S.J.; Levy, H.M. Simultaneous multithreading: Maximizing on-chip parallelism. In Proceedings of the 22nd IEEE Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Genoa, Italy, 22–24 June 1995; pp. 392–403. [CrossRef]