Muttillo *et al. EURASIP Journal on Embedded Systems* (2016) 2016:15
DOI 10.1186/s13639-016-0051-9

EURASIP Journal on Embedded Systems

RESEARCH Open Access

A design methodology for soft-core platforms on FPGA with SMP Linux, OpenMP support, and distributed hardware profiling system

Vittoriano Muttillo[1*], Giacomo Valente[1], Fabio Federici[1], Luigi Pomante[1], Marco Faccio[1], Carlo Tieri[2] and Serenella Ferri[2]

Abstract

In recent years, the use of multiprocessor systems has become increasingly common. Even in the embedded domain, the development of platforms based on multiprocessor systems or the porting of legacy single-core applications are frequent needs. However, such designs are often complicated, as embedded systems are characterized by numerous non-functional requirements and a tight hardware/software integration. This work proposes a methodology for the development and validation of an embedded multiprocessor system. Specifically, the proposed method assumes the use of a portable, open source API to support the parallelization and the possibility of prototyping the system on a field-programmable gate array. On this basis, the proposed flow allows an early exploration of the hardware configuration space, a preliminary estimate of performance, and the rapid development of a system able to satisfy the design specifications. An accurate assessment of the actual performance of the system is then enforced by the use of an hardware-based profiling subsystem. The proposed design flow is described, and a version specifically designed for LEON3 processor is presented and validated. The application of the proposed methodology in a real case of industrial study is then presented and analyzed.

Keywords: Multicore architectures, Performance evaluation, Embedded systems, Parallelization framework, Reconfigurable logics

1 Introduction

In the embedded systems domain, a proper tailoring of platform resources is always more frequently required in order to better exploit the whole system. For example, with the right mapping of tasks (i.e., processes or threads) onto a customized number, type, and configuration of processing cores, performance can be pushed to the theoretical limit. However, an application parallelization task is explicitly required to the programmer, so they have to take care of platform details. To facilitate this operation, different libraries have been proposed in the market, such as *OpenMP* [1], *OpenCL* [2], and *OpenMPI* [3]. *OpenMP* is widely adopted both at industrial and research levels and supports a wide range of programming languages, processing architectures, and operating systems. Moreover, reconfigurable logics have gained a wide diffusion in the embedded systems domain, due to the possibility to perform a quick system customization. In particular, nowadays, field-programmable gate arrays (FPGAs) are widely diffused both as prototypal elements, due to their lower non-recurring engineering costs, and as support for application execution, since they can be used to realize ad hoc accelerators for time-critical functionalities. In this context, the possibility of building multiprocessor systems by exploiting soft-cores increases the range of the applications that can be implemented on FPGAs. Soft-cores can be intended as general-purpose processors (i.e., soft-processors) or co-processors. Specifically, the use of a soft-processor allows to exploit a programmable

*Correspondence: vittoriano.muttillo@graduate.univaq.it
[1]University of L'Aquila, Center of Excellence DEWS, Via Giovanni Di Vincenzo 16/B, 67100 L'Aquila, Italy
Full list of author information is available at the end of the article

component with customized peripherals, inserting only necessary parts. For this, various FPGA vendors provide soft-processors optimized for their reconfigurable logic: for example, *Xilinx* offers *MicroBlaze* [4] and *PicoBlaze* [5], and *Altera* has *Nios-II* [6]. Moreover, third party vendors offer soft-processors targeting specific domains. For example, *Gaisler Research* provides the LEON family [7] for avionic systems.

With the increment of logic and memory elements available on FPGAs, complex multiprocessor architectures can be developed (e.g., uniform memory architecture (UMA); not uniform memory architecture (NUMA); network-on-chip (NOC)) by exploiting also proper operating systems. On UMA architectures, symmetric multiprocessing (SMP) Linux-based operating systems can be adopted to exploit shared-memory multicore architectures on FPGA. Then, given a multicore architecture, by splitting the workload on various cores, it is possible to obtain a relevant speed-up for multi-threaded applications. However, to easily perform this task, a parallel programming model should be used. Unfortunately, the support to several parallel programming models have been provided to a lot of embedded ASIC architectures, but working in reconfigurable logic area, this kind of support is still not well mature. In particular, as described later, in the context of this work it has been needed to explicitly port the well-known OpenMP library to the proposed execution environment.

In such a scenario, the main contribution of this work is the definition of a design flow to support a designer that needs to improve performance of its embedded application when implemented on a reconfigurable platform (i.e., FPGA). It is assumed that such an application already runs on a single-core platform on FPGA. Starting from this entry point, the flow allows the designer to evaluate the speed-up reachable by parallelizing the application (by means of OpenMP) and by running it on a shared-memory multicore architecture (based on multiple instances of the same starting soft-core). Then, the flow allows the designer to realize such a multicore platform on FPGA by providing also an SMP Linux with OpenMP support. Moreover, the flow allows the designer to integrate in the platform a distributed HW profiling system, tailored for the multicore scenario on FPGA, that does not introduce software overhead. Such a profiling system allows to further monitor, both at design time and run time, system performances and to consider possible run-time reconfigurations. The presented design flow has been used to support the development of the multicore platform in the context of CRAFTERS project [8]. In particular, in the first design phases, it has been customized for LEON3 processor, but it can be easily extended to other soft-processors, as described in the final considerations.

This paper is organized as follows. Section 2 describes the state of art related to frameworks for parallelization, multicore platforms based on soft-processors, and HW/SW profiling approaches for multicore embedded systems. Section 3 describes the proposed design flow, and Section 4 presents the customization of the proposed flow for a LEON3-based multicore system. Section 5 presents the validation of the customized design flow while Section 6 shows its exploitation to develop a real-world industrial use case. Finally, Section 7 reports some conclusive considerations and presents future activities on the topic.

2 Related work

As stated before, this work presents a design flow targeting multicore hardware on FPGA for speeding up execution of embedded applications. A profiling system is also integrated in the final platform. So, this section reviews and analyzes some relevant examples of frameworks that allow to adapt an application to work on multi-/many core system (parallelization). The second subsection reports some existing multicore platforms based on soft-processors. Finally, some relevant profiling system mechanisms existing in the literature are reported.

2.1 Frameworks for parallelization

Multicore architectures are increasingly used in an embedded system, in order to speed up the execution of an application and to perform energy saving. Aldinucci et al. [9] presented a porting of a decision tree algorithm implementation to a multicore platform using FastFlow [10]. The focus was, by using high-level programming layers, to exploit parallelization of a server computer using different strategies. Our work differs because it targets embedded system domain and allows to model the hardware to tailor platforms to the real needs. A framework called PEMAP [11] allows to estimate performance of serial application when parallelized to work on a graphics processing unit (GPU). Our work differs because it targets multiprocessor systems but assuming that hardware can be configured. The advantages of such a hypothesis will be shown in the following sections.

2.2 Multicore platforms based on soft-cores

The advantage to exploit a multicore platform on FPGA is given by the possibility to configure exactly what is needed for the execution of an application. Caches, MMU, and ALU parameters can be customized to better match the requirements, especially the non-functional ones. In fact, different multicore platforms based on soft-processors have been developed up to now. For example, *Poly-Blaze* [12] is a multicore MicroBlaze-based system with SMP Linux support on FPGA. Huerta et al. described another work related to MicroBlaze in SMP configuration

[13]: the system consists of 4 MicroBlaze soft-cores, shared block-ram through on-chip peripheral bus (OPB), local memory for each processor and a hardware synchronization mechanism. David et al. analyzed different systems on chip (SoC) based on multiprocessor architectures [14]. They focus on two reference architectures: one with shared memory used only for task migration, that requests a complex programming model, and another one with shared memory connected through a multiport memory controller that allows reading and writing memory locations simultaneously. Serres et al. [15] presented a reconfigurable multicore scenario based on LEON3 soft-processor and proposed a co-processor interface for each core.

2.3 Software- and hardware-based profiling approaches

One of the goals of profiling is the measurements of time intervals related to the running code, such as response time (i.e., the elapsed time between the start of the application execution and its end, including also the time needed to serve interrupts and context switches time) or execution time (i.e., the time needed to the processor to execute the application, without considering interrupts and context switches) of an application [16], at various levels of granularity (i.e., task level, instruction level, etc.). In general, profiling approaches can be based on software techniques or can rely on direct hardware support. Software-based solutions usually follow the general approach of classic profiling tools, like GPROF [17], the GNU statistical profiler able to estimate the execution time of functions that compose an application and to count the number of times they have been called. In the case of GPROF, instrumentation of source code allows to generate interrupts and it samples the program counter, obtaining various statistics about software execution. Software profiling necessarily introduces some overhead on occupied area in memory (due to application instrumentation), and some overhead on execution time (because the sampling is done in an interrupt service routine executed by the processor). Moreover, in the case of GPROF, there is a grade of statistical inaccuracy essentially due to the sampling frequency.

Hardware profilers are instead based on dedicated hardware resources able to carry on the profiling action. This means that no (or very limited) source code instrumentation is needed, and the software execution by the central processing unit is not altered. Thus, no overhead on execution time is introduced. For the same reason, hardware solutions can guarantee the best accuracy in performance analysis. However, these solutions require a larger silicon area occupation for system implementation. Other possible disadvantages are the difficulty to correlate low-level measurements to source code performance metrics and the limited number of available hardware resources that often force to collect desired performance metrics by means of multiple tests. Various examples of hardware-based profiling approaches have been presented in literature. For example, SnoopP [18] and Airwolf [19] are two function-level profilers for software applications running on soft-core processors. In a multicore scenario, Shannon et al. [20] adapted the ABACUS profiling system to work on heterogeneous platforms composed of multiple cores and accelerators, while Nam Ho et al. [21] proposed an infrastructure for performance monitoring of LEON3 in multicore configuration. Considerations on the use of such solutions in the proposed flow are presented in Section 3.

3 Design flow

As reported in Section 2, there exist various implementations of multicore platforms based on soft-core processors. Instead, this work does not focus only on a specific implementation but it also defines a design flow that addresses the problem to implement a multicore platform on FPGA able to support the OpenMP library and that can be also analyzed by means of a distributed HW profiling system. In particular, the main goal is to support a designer that needs to improve performance of its embedded application. So, starting from functional and non-functional (i.e., in this case, the required execution speed-up) requirements, the main steps of such a flow are shown in Fig. 1.

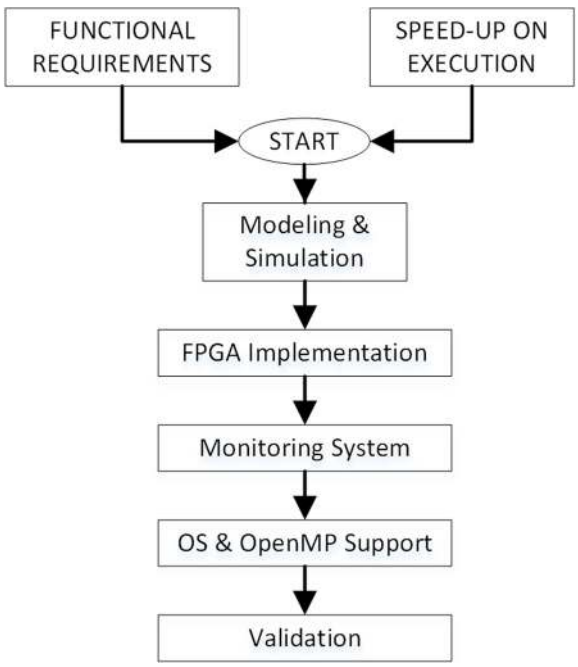

Fig. 1 Reference system design flow

The entry point is a target application (e.g., a program written in C/C++ code), running on a single core on FPGA and already able to satisfy functional requirements. The first step is related to evaluate if, by means of OpenMP parallelization (over a variable number of cores), it is possible to satisfy the (non-functional) speed-up requirement. Strictly related to this analysis is the identification of the architectural parameters (e.g., cache organization, bus bandwidth) that could have effects on the same requirement. Finally, once such parameters are identified, the last action is related to evaluate their optimal values. All this can be performed by means of a proper system-level simulator. In other words, this step allows to perform a design space exploration with respect to several ways to exploit OpenMP features and different architectural parameters. The second step is related to the effective implementation of the identified multicore architecture on FPGA: starting from the results of the first step (i.e., the multicore architecture and its parameters), all the elements are instantiated and connected on FPGA. The third step is related to the selection and the integration of a monitoring solution able to measure parameters useful to evaluate at run-time actual speed-up of execution on the target system. The fourth step is related to the integration of an operating system and the components needed to provide the support to OpenMP-based applications. The last step is related to requirement validation on the final target. In the next subsections, each step of the design flow is better explained.

3.1 Modeling and simulation

The first step is the modeling and simulation one. It consists of the modeling of the target architecture in order to estimate system performance in the execution of the target application by means of simulation. For this purpose, the modeling of HW/SW elements can be done at different abstraction levels by using block diagrams, *UML*, *SystemC*, or other modeling languages. Normally, the accuracy of the results depends on such abstraction level that, unfortunately, can also have effects on simulation time. Moreover, this one depends on the simulator itself and the features of the machine (the host) used to perform the simulations. In the context of this work, VIrtual Parallel platform for Performance Analysis (VIPPE) simulator [22] has been selected. It is an electronic design automation tool for HW/SW simulation that provides a library of multicore platforms that can be extended and allows the simulation of an operating system and the simulation of applications that use OpenMP. It provides run-time simulation statistics such as execution time, cache behavior, and power dissipation. VIPPE relies on platform modeling based on UML/MARTE. By means of this modeling language, it is possible to model the platform, the mapping between tasks and processing cores. Moreover, it is also possible to model an operating system and specific libraries (such as OpenMP). From an operative point of view, after the modeling phase is completed, the target application is compiled by means of LLVM [23] (or a different source compiler, depending on what is supported by the target processor) in order to obtain related assembly instructions. Then, for each assembly instruction, VIPPE considers a cost from the point of view of execution time and energy consumption. Such costs depend on another modeling file that describes the processor under simulation. It contains the list of instructions and the associated costs.

The design space exploration that can be performed by using the simulator is illustrated in Fig. 2. Starting from the speed-up requirement for the target application, it is possible to identify the number of cores, the cache parameters, and the OpenMP clauses needed to satisfy the requirement. Considering also the simulation time, it is worth nothing that VIPPE allows to fully exploit the (hopefully multicore) host machine by making a host-based simulation [24]. In this type of simulation, each target thread is mapped on a host thread. VIPPE adds another thread, called kernel, that communicates with other threads managing the simulation and collecting performance of execution [22]. In conclusion, VIPPE can be used to analyze the correctness of the target application while executed on a multicore platform, to make a design space exploration with respect to architectural parameters, and to evaluate the impact of different choices in the use of OpenMP.

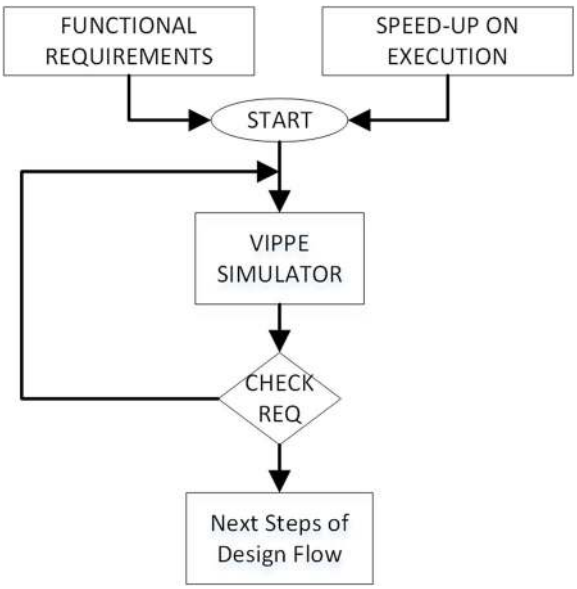

Fig. 2 Simulation-driven design space exploration

3.2 FPGA implementation

After the modeling and simulation step, it is then possible to implement the platform on FPGA by considering the identified parameters about the multicore architecture. The actual process and the toolchain to be used are strictly dependent on target technologies.

3.3 Monitoring system

Starting from the multicore platform developed in the previous step, in order to evaluate the system speed-up, response time on the real target has to be measured. For this, a run-time monitoring system should be integrated in the final system. As described in Section 2, several options are available. In order to avoid as much as possible introducing overhead in the software execution, the proposed flow is based on a hardware solution. In fact, it exploits a fully customizable and portable distributed HW solution based on the library called AIPHS ([25, 26]): specifically, it is a library of elements to be used to realize a monitoring solution. In fact, it allows to consider architectures based on different soft-processors while changing only few hardware components. Such a choice allows to overcome some limitations of existing approaches. For example, the solution proposed in [21] lacks portability among different soft-processors, while ABACUS, a profiling solution adapted in multicore scenario [20], although represents a smart profiling solution portable among different architectures, presented high area occupation because it is intended to be used during development phases. In this work, the monitoring system to be added in the final multicore architecture is intended to be left in the final platform: so, the hardware overhead has to be kept into account, as will be shown in the next sections.

The customization of a monitoring solution for the proposed LEON3-based implementation is shown in the next section.

3.4 OS and OpenMP support

Once the HW multicore architecture is implemented, by customizing and interconnecting soft-processors as suggested by the simulation results, and the hardware monitoring mechanism is inserted, there is the need to customize an SMP Linux operating system. Such an OS is needed to support OpenMP application: OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify high-level parallelism in FORTRAN and C/C++ programs. It instructs the compiler to organize parallel sections of code in a specific manner, and helps to parallelize execution of an application. It is based on a fork-join model. The implementation of OpenMP based on GCC, called *libgomp* [27], has been selected to provide support to the execution of parallelized applications that use this library. This motivates the need of an SMP Linux distribution. In order to provide *libgomp* on the target, the porting of the required SW components has to be done by cross-compiling source files and inserting results in the kernel.

4 LEON3-oriented design flow specification

In this section, the design flow defined in Section 3 is furtherly specified to target a multicore LEON3 architecture. It is worth noting that the flow can be easily

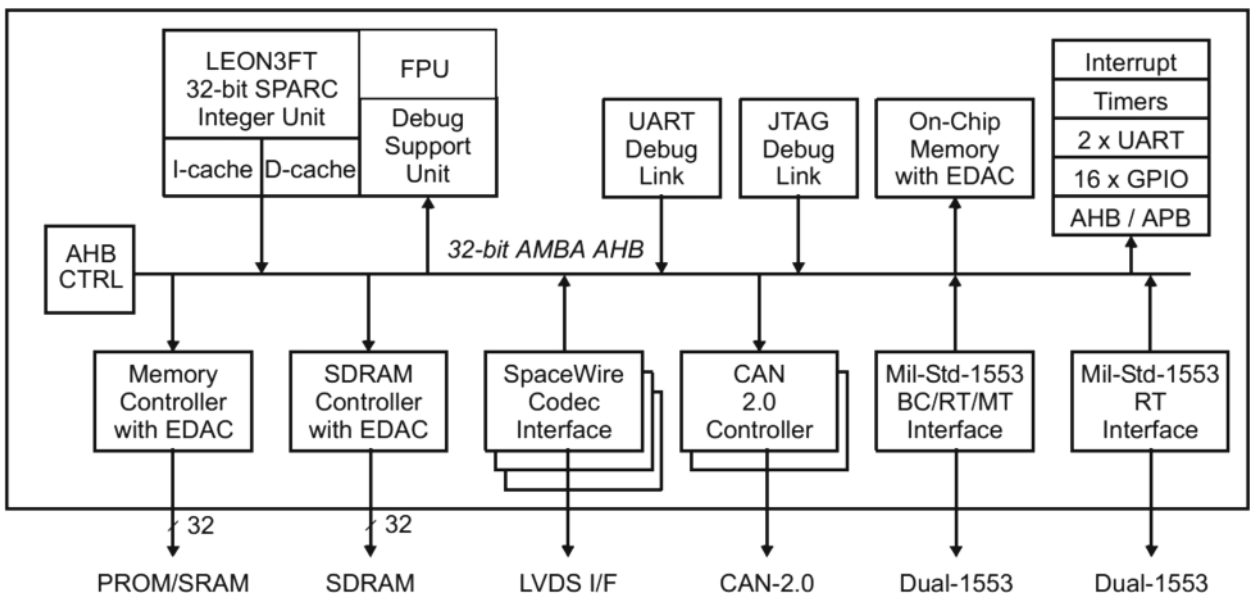

Fig. 3 Typical implementation of the LEON3 processor [35]

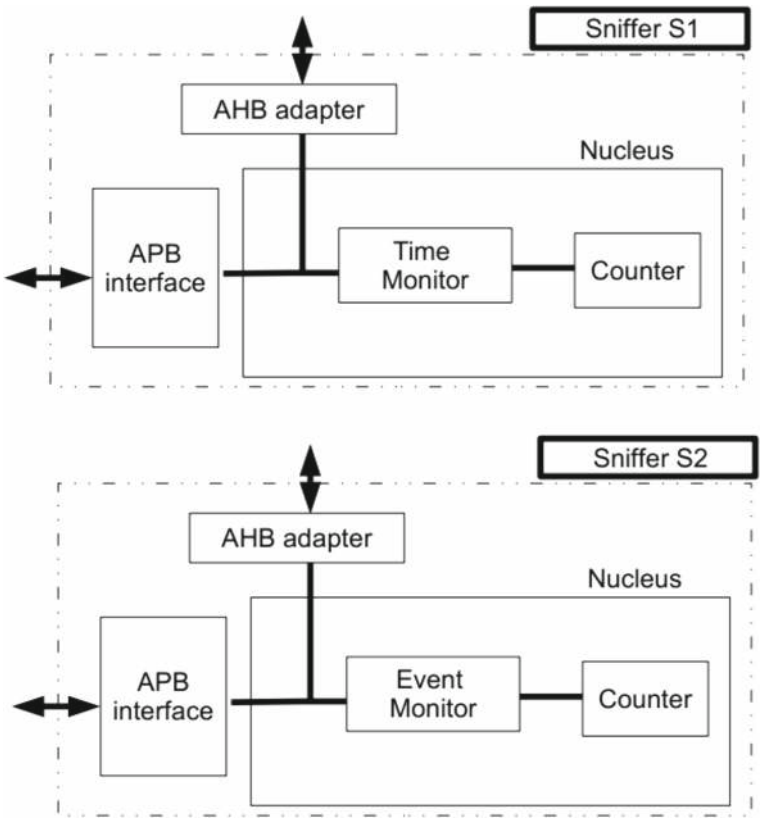

Fig. 4 Internal architecture of S1 and S2

applied to other soft-processors, as discussed in the final considerations.

LEON3 [7] is a 32-bit synthesizable soft-processor that is compatible with SPARC V8 architecture: it has a seven-stage pipeline and Harvard architecture. It uses separate instruction and data cache memories and supports multiprocessor configurations: in particular, an SMP-aware configuration is well supported thanks to available memory management unit and snooping unit for cache coherence. It represents a soft-processor for aerospace applications. LEON3 is described by means of an open-source VHDL model and provides full configurability by means of the Gaisler Research IP Library (GRLIB).

LEON3 is also diffused as ASIC implementation. The proposed flow targeting LEON3 is intended to be used during the prototyping phases, where FPGAs give great effort because of their lower non-recurring engineering (NRE) costs. The first step is to configure the simulator to work with LEON3-based multicore platform. So the related UML/MARTE model has been created. Then, to associate costs to the whole LEON3 instruction set, it has been considered an ideal pipelined execution (LEON3 is a RISC processor). This leads to consider a cost of one clock cycle for each assembly instructions. In the context of early design space exploration (as the one considered in the presented flow), such an hypothesis leads to an accuracy good enough to perform meaningful comparisons among different design choices (as will be shown in the validation section). A different option could be to refer to some average metrics (e.g., MIPS, CPI). However, such values would be dependent on the benchmarks used to evaluate them, so, in the context of this work, it has been preferred the ideal approach. Then, among the options provided by VIPPE, it has been considered the model of an SMP Linux operating system running on the multicore platform while providing also support to OpenMP.

Fig. 5 General view of monitoring operation

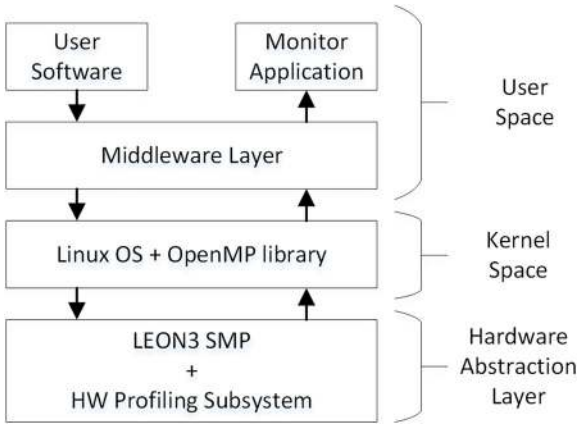

Fig. 6 SMP LEON3-based final platform architecture

After the modeling and simulation step, it is possible to implement the platform on FPGA. A typical single-core implementation of the LEON3 processor is reported in Fig. 3.

The LEON3 processor, together with some peripherals (USB, JTAG, Ethernet controllers), is connected on a system bus, the advanced high-performance bus (AHB) [28]. Since AHB is a high-performance bus, for low-speed peripherals, another bus is used: the advanced peripheral bus (APB) [28] that is connected to the system bus using an AHB/APB Bridge. In order to obtain a multicore platform, other LEON3 processors can be connected to the AHB, acting as multi-masters. Indeed, AHB has a controller (the AHB controller in Fig. 3) that contains an arbiter to manage multiple masters. The multicore implementation proposed by the simulator is then implemented to FPGA by extending the scheme on Fig. 3.

Starting from such a multicore platform, a monitoring system has been designed for the proposed LEON3-based implementation considering elements of AIPHS library. It is composed of two subsystems: a bus analysis subsystem (constituted by various dedicated hardware sniffers) and a global monitor subsystem (GMS). Each sniffer detects and collects selected events and notifies them to GMS. Each sniffer is able to monitor a specific system interconnection (e.g., the memory bus, the communication bus between two processing elements). In order to measure response time, a proper sniffer (named S1) monitors the system bus and identifies the start of application execution and its end. In fact, S1 is able to measure elapsed time between two triggering conditions. Other sniffers are not strictly needed, but another one (S2) has been inserted in the platform to consider a parameter that can be very useful at run time: a sniffer that measures the number of bus accesses allowing also to monitor effective bus bandwidth. This parameter provides information on whether the bus can accept more processing cores without becoming a bottleneck. The internal architectures of S1 and S2 are reported in Fig. 4. The target system bus is the AHB and it is connected to a target bus adapter section that gives necessary signals for the *time monitor*, that measures the time elapsed between two occurred events, and the *event monitor*, that counts the number of events.

Finally, a control logic block, named *Nucleus*, controls a *counter* and the whole sniffer behavior. It is worth noting that, in the proposed configuration, the role of GMS is considered to be played by one of the LEON3 processors already existing in the architecture. GMS carries out the start-up sniffer configuration before the execution of the monitored application and collects results at the end. Although one of the LEON3 is used as GMS, this does not cause overhead during the monitoring action, since the startup and the collection of results are done outside from the monitored application execution. Communication between LEON3 and sniffers is done by means of the APB; therefore, also an APB interface exists in the sniffer architecture. A general view of the monitoring operations is given in Fig. 5. The black part represents, respectively, the initialization phase (from t1 to t2) and collection of results (from t4 to t5) of the sniffers done by GMS, in this case the LEON3 processor. The gray part is the application execution, where response time (from t2 to t4) and bandwidth are measured by means of the monitoring system. The dotted lines represent the time execution of other

Table 1 DSE 1st benchmark

Tag	#Cores	Sets/WS/LS/CS		OMP	DC Rat	#Instr.	SU
		IC	DC				
C1	1	1-1-16-1	2-4-16-8	–	1 %	240000034	–
C2	1	1-1-16-1	2-4-16-8	Reduction	1 %	310000178	0.8×
C3	2	1-1-16-1	2-4-16-8	Reduction	1 %	155000175	1.5×
C4	3	1-1-16-1	2-4-16-8	Reduction	1 %	103333447	2.3×
C5	4	1-1-16-1	2-4-16-8	Reduction	1 %	77500112	3.1×
C6	4	1-1-16-1	2-4-16-8	SPMD/FS	1 %	77500152	3.1×
C7	4	1-1-16-1	2-4-16-8	SPMD/NFS	1 %	62500061	3.8×

Table 2 Configuration parameters for LEON3

Parameter	Attribute	Value
Clock generation	System clock frequency	75 MHz
Processor	Number of processors	4
Integer unit	SPARC register windows	8
FPU	Enable FPU	Yes
I-cache	Sets/WS/LS/CS	1 1 16 1
D-cache	Sets/WS/LS/CS	2 4 16 8

applications (with higher priority than the monitored one) and include even the context switch overhead. Once the HW multicore architecture is finalized, there is the need to customize an SMP Linux operating system and port the libgomp (needed to support OpenMP) to this distribution. In this case, a Linux distribution has been built starting from *LEON LINUX Kernel 3.10*.

The kernel is customized to work with the multicore platform in SMP mode and has been developed using *Buildroot Tool* [29] and the cross-compiler toolchain [30] provided by Cobham Gaisler. In order to implement *libgomp* for the target, the needed libraries have been cross-compiled and integrated into the selected Linux distribution.

Finally, in order to allow the applications to interact with the monitoring system by means of both kernel and user space Linux processes, a proper Middleware API and related drivers have been defined and developed. With this API, it is possible to initialize the sniffers and retrieve information on response time (during the black slots in Fig. 5). The final platform architecture appears in Fig. 6.

5 Validation

In order to validate the LEON3 customized design flow, some benchmarks (e.g., PI calculation, matrix multiplication, and FFT) have been selected and used as an entry point. Specifically, during this test, the speed-up obtained inserting OpenMP clauses in the original code (to parallelize it among a certain number of cores) has been firstly estimated with VIPPE and then, by means of the unobtrusive monitoring system, accurately evaluated in the final platform. The comparison between the estimated and real values of speed-up aims to demonstrate the effectiveness of the proposed approach. In this section, the PI benchmark and the results obtained by running it on the proposed LEON3 multicore platform implemented on a *Xilinx Virtex 6* FPGA (*ML605 Dev. Board*) [31] are described.

The benchmark is constituted of an algorithm that performs a numerical integration able to calculate the PI value [1]. The algorithm was highly parallelizable, and it was selected to perform a first basic test of the entire flow. The initial value of response time for the application on single LEON3 core was 8394 ms. Supposing a speed-up of 3× is required, by using a multicore platform based on LEON3 and OpenMP, the proposed flow has been applied. Simulation with VIPPE has been performed on the model of LEON3 processor described in the previous section, and results of the design space exploration (DSE) are reported in Table 1. In Table 1, #Cores indicates

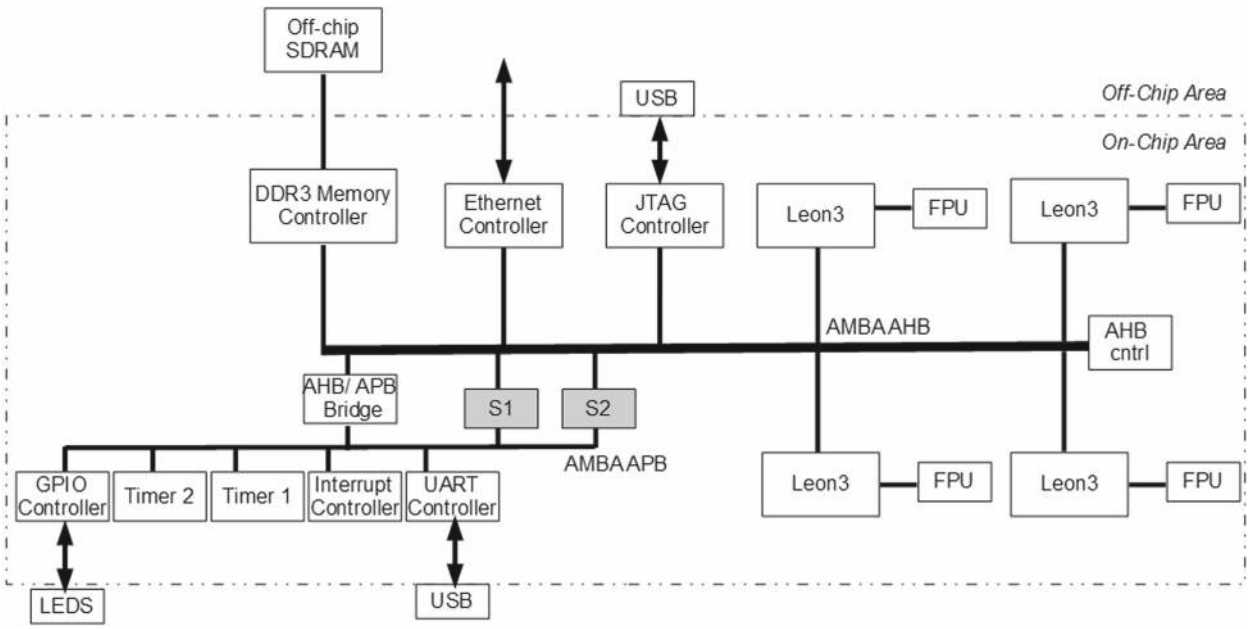

Fig. 7 Final implemented architecture for PI benchmark

Table 3 Response time of implemented architecture

Configuration	Response time	Speed-up
C1	8394 ms	–
C2	9322 ms	0.9×
C3	4751 ms	1.7×
C4	3198 ms	2.6×
C5	2474 ms	3.4×
C7	2257 ms	3.7×

the number of cores, while the third column reports cache parameters: itSets is number of sets, itWS is the way size, itLS is the line size, and itCS is the cache size, that are reported for itIC and itDC (respectively itinstruction cache and itdata cache). itOMP indicates the parallelization technique used, itDC Rat is the ratio between data cache misses and total accesses on data cache. it#Instr is the number of executed instructions and itSU is the speed-up. Since first VIPPE results on one core (i.e., C1) shows that the ratio of data cache misses is not high (1 %), cache configuration has not been changed. Then, OpenMP has been used to modify the source code. The simulation of the application with OpenMP on one core, that represents the C2 configuration, estimates a speed-up of 0.8×. This is worse than the serial execution but it is justified by the growing of the number of instructions (numinstructions) to be executed on the same core. Then, while increasing the number of cores (i.e., C3), also the speed-up starts to improve (i.e., 1.5× with two cores). OpenMP has been used with the reduction clause, that allows to organize the operation to be done in a loop (in this case it acts on multiple sums). At C5 configuration, four cores provide a speed-up of 3.1×: this value satisfied the requirement of speed-up equal or greater of 3×; however the value was near to this threshold and, due to a not completely detailed model of the processor, a stronger band guard has been searched. Specifically, another way to work with OpenMP has been exploited, i.e., the Single Process Multiple Data (SPMD) technique that allows a better control of the parallelization process of the main application loop program. This technique has led to a speed-up still equal to 3.1×, as indicated in C6 configuration, but, when removing also the false sharing problem in C7, the speed-up has reached its max equal to 3.8× and very close to the ideal one. The false sharing problem is a performance degradation caused by a repetitive cache flush, due to the fact that threads read and write data contained on the same cache line. We forced, in C7, the concurrent threads to work with separate cache lines. The total invested time to perform the Design Space Exploration has been about 3 h, also considering the study of the source code in order to identify sections suitable for parallelization. So, this is the configuration to be implemented on FPGA, that satisfies the speed-up requirement of 3× with a sufficient guard band. The parameters identified for LEON3, following the suggestions obtained by the simulation and the naming convention proposed by GRLIB, are reported in Table 2.

The final multicore architecture is shown in Fig. 7. Each core has one cache level (for data and instructions) and a memory management unit, as required from the selected

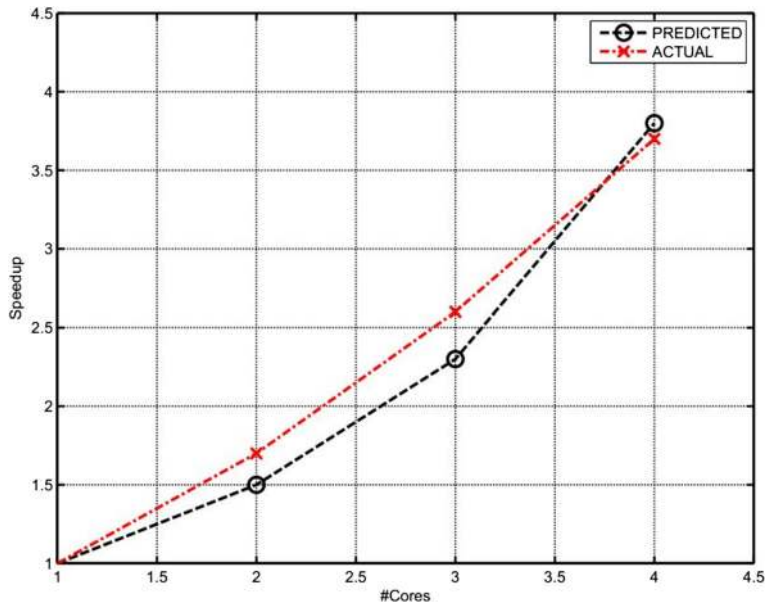

Fig. 8 Speed-up predicted and actual

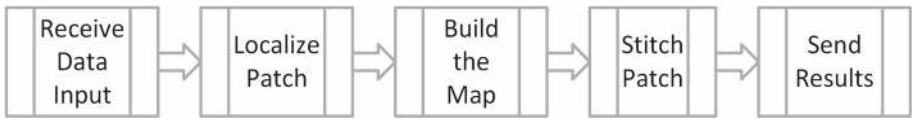

Fig. 9 Main steps of industrial application

Linux distribution. A shared *Ethernet* controller has been added too, to be used as input/output link for data processing and to make debugging of user-space code. For each core, there is a dedicated floating point unit (FPU): this has been inserted because the considered benchmark involves floating point operations. The monitoring system has been inserted in the architecture (i.e., the shaded S1 and S2 components). The area occupation on Virtex6 on ML605 Dev. Board is equal to 37,000 slice registers (12 %) and 88,000 slice LUTs (58 %). This conduced, after implementation, to a total number of occupied slices equal to 80 %. In order to show the validity of the model used during the simulation to understand the speed-up trend, some of the tests done during simulation have been performed also on the final target (other than the final test to verify the final speed-up of response time). Cache dimension is fixed in the final hardware, so the tests have been done only changing the number of threads and the use of OpenMP. Results on response time have been collected using hardware profiling API. They are illustrated in Table 3.

The comparison between the speed-up predicted by simulation and the real trend is reported in Fig. 8.

The trend is the same, while there is a slight difference in the values. This is due to the model of the processor that has been loaded on VIPPE and to the approximation of the cost of each instruction equal to one clock cycle. The important thing is that, by using the proposed flow, in only one implementation step on FPGA, a suitable platform able to provide better performance on multicore has been obtained. Finally, by tailoring the multicore platform parameters to the value predicted by the simulation, using OpenMP and removing the false sharing phenomena, a speed-up equal to 3.7× has been reached, that provided a response time equal to 2257 ms.

6 Real-world industrial application

This section presents the proposed flow applied to an industrial application (that is called benchmark in the following). Specifications to run this benchmark have been provided in the context of CRAFTERS project [8]. It is represented by an algorithm that supports a real-time location system (RTLS) in an indoor scenario. More specifically, the algorithm is related to an indoor positioning system, for anchor-free (i.e., infrastructure less) localization in a mobile ad hoc network (MANET). It is worth noting that in an anchor-free scenario, there is more demanding for computational power, so it is a good use case for the proposed flow. Moreover, the prototyping phase applied to LEON3-oriented design flow had the purpose to test the algorithm on a multicore architecture able to provide certain response times. Future works have been planned in order to realize a SoC starting from the multicore prototyping platform identified by applying the proposed flow. The use of LEON3 in this case has multiple advantages: it is a processor that implements the SPARCv8 architecture, and OS targeting this architecture are stable. Another focus point was the fact that LEON3 is available under GPL license and it is well tested. In the proposed embedded application, the execution platform is

Table 4 DSE for industrial application for 50 nodes localization

Tag	# Cores	Sets/WS/LS/CS		IC Rat.	DC Rat	SU
		IC	DC			
C1	1	1 1 16 1	1 1 16 1	1 %	23 %	–
C2	1	2 4 32 8	1 1 16 1	1 %	21 %	1.042445467
C3	1	2 8 32 16	1 1 16 1	–	21 %	1.029811464
C4	1	2 8 32 16	2 8 32 16	–	17 %	0.909814107
C5	1	2 8 32 16	2 4 32 8	–	19 %	1.101375597
C6	2	2 8 32 16	2 4 32 8	–	19 % per core	1.39721928
C7	3	2 8 32 16	2 4 32 8	–	21 % per core	1.132754151
C8	4	2 8 32 16	2 4 32 8	–	18.6 % per core	1.853288014
C9	4	2 8 32 16	4 8 32 32	–	1.8 % per core	4.945024246
C10	4	2 8 32 16	4 4 32 16	–	2 % per core	2.311683089

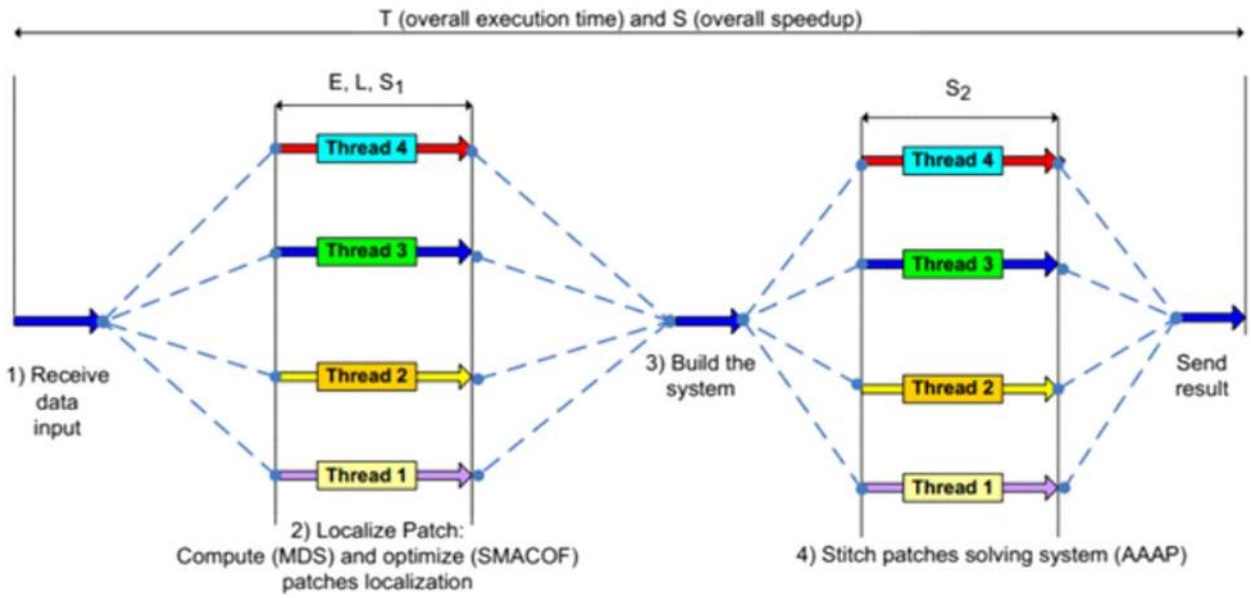

Fig. 10 Parallelization of industrial application

considered to be the node of a MANET in an indoor scenario, called itmaster node, that has some *neighbor nodes*. The main phases of the application are shown in Fig. 9 and described in the following steps:

1. *Receive data input*: the node receives a distance pairs matrix, obtained by UWB ranging. In this case, data are provided from a host computer connected to a database of measures.
2. *Localize patch*: starting from the distance pairs matrix, a Multi-Dimensional Scaling (MDS-MAP) algorithm [32–34] is executed on the node to obtain a local map of the neighbor nodes. Next, a least square minimization is performed using a technique called Scaling by MAjorizing a COmplicated Function (SMACOF).
3. *Build the map*: each node evaluates a first global map of the scenario.
4. *Stich patches*: each node refines the global map using an As Affine As Possible (AAAP) algorithm.
5. *Send results*: the nodes send data back to the host computer.

In the source code of the application, there are different parameters to be changed in order to model different scenarios. One of these is the number of nodes viewed by the node (that is executing the algorithm). The initial value of response time for the application on single LEON3 core was 1644 ms for a scenario with 50 nodes. Given the requirement of a speed-up of 2× for 50 nodes by using a multicore platform based on LEON3 and OpenMP, the proposed flow has been applied. Simulations with VIPPE have been performed on the LEON3 processor model described in the previous section, with results indicated in Table 4.

Table 4 columns have the same meaning as those of Table 1 in the previous subsection. In this case, IC Rat and DC Rat indicate the ratio between cache miss and total number of cache accesses for, respectively, instruction cache and data cache. Starting from C1, in C2 and C3 a modification of the instruction cache organization has

Table 5 Simulation results for different nodes

Nodes	Configuration	Speed-up
50	C1	–
50	C10	2.3×
80	C1	–
80	C10	3.26×
100	C1	–
100	C10	4.67×

Table 6 Configuration parameters for LEON3

Parameter	Attribute	Value
Clock generation	System clock frequency	75 MHz
Processor	Number of processors	4
Integer unit	SPARC register windows	8
FPU	Enable FPU	Yes
I-Cache	Sets/WS/LS/CS	2 8 32 16
D-Cache	Sets/WS/LS/CS	4 4 32 16

Table 7 Response time on the target

Nodes	Configuration	RT	Threads	Speed-up
20	C10	57 ms	4	–
30	C10	100 ms	4	–
40	C10	166 ms	4	–
50	C1	1644 ms	–	–
50	C10	839 ms	2	–
50	C10	634 ms	4	2.6×
60	C10	1046 ms	4	–
70	C10	1263 ms	4	–
80	C1	10,279 ms	–	–
80	C10	2989 ms	4	3.4×
90	C10	4319 ms	4	–
100	C1	8100 ms	–	–
100	C10	1838 ms	4	4.4×

been considered, that conduced to a lower IC Rat value, that did not cause a speed-up on performance. In C4 and C5, also data cache has been reorganized, so obtaining a performance speed-up in the latter. Then, the number of cores has been raised: in C8 we can see a speed-up of 1.8×. The requirement was to try to have a speed-up 2× for 50 nodes. To do this, we changed the data cache organization inserting an associativity of 4. This conduced, initially, to a C9, that unfortunately resulted infeasible: the problem was due to a toolchain limitation. In fact, it supports only operating systems with page size of 4 kB. This conduces to a maximum data cache dimension of 16 kB, while in C9 we exceeded this value (CS = 32 kB). In C10, we solved reducing the way size. We obtained a speed-up of 2.3×, that satisfies the requirement. Figure 10 shows how the application has been parallelized using OpenMP. The first and last phases are intrinsically serial. The second and fourth have been parallelized using OpenMP.

The third has been left serial: this is because in this part of the algorithm there are operations on sparse matrices, and, after a depth analysis, we stated that the implementation, in this phase, was not parallelizable.

Finally, for evaluation purposes, some simulations have been performed using the configuration C10 also for scenarios with 80 nodes and 100 nodes. Results are reported in Table 5. The total invested time to perform the design space exploration has been about 2 h, not considering the study of the source code in order to identify sections suitable for parallelization. The configuration parameters of LEON3, following the suggestion obtained from the simulation and the naming convention proposed by GRLIB, are reported in Table 6.

The complete multicore architecture is similar to that shown in Fig. 7. The area occupation on Virtex6 on ML605 Dev. Board is equal to 38,000 slice registers (12 %) and 91,000 slice LUTs (60 %). This conduced, after implementation, to a total number of occupied slices equal to 80 %. Results on response time have been collected using hardware profiling API. They are illustrated in Table 7.

Tests on the final target have been done for different number of nodes. The speed-up requirement for 50 nodes is satisfied on the target, showing the validity of the

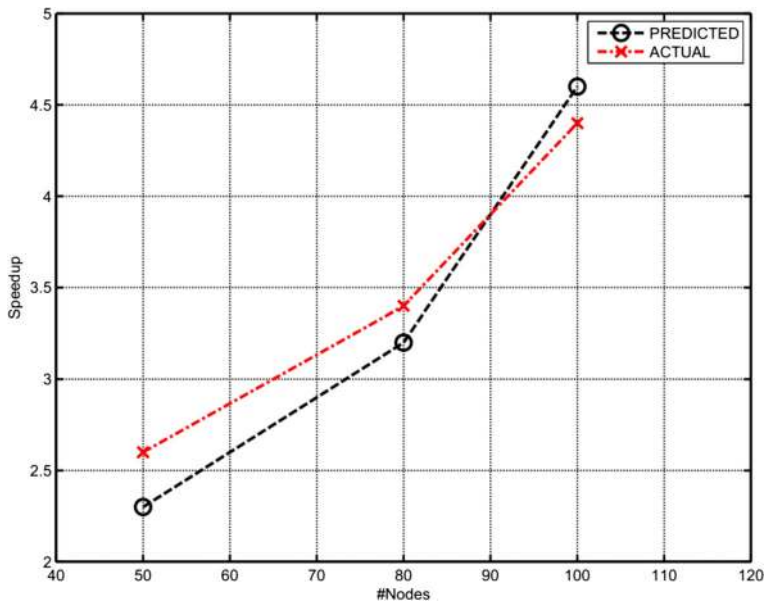

Fig. 11 Speed-up predicted and actual

Table 8 Bandwidth of system bus

Nodes	Bandwidth (b/s)
80	619,923,284
100	562,312,437

proposed flow to make a first design space exploration and to implement a multicore platform on FPGA. Speed-up for 80 nodes and 100 tends to be higher, this is due to the fact that the parallel sections of the algorithm (shown in Fig. 10) grow in percentage while number of nodes grow.

The comparison between the speed-up predicted by simulation and the real trend is reported in Fig. 11. The trend is the same, while there is a small difference in the values. This is mainly due to the model of the processor that has been loaded on VIPPE and to the approximation of the cost of each instruction equal to one clock cycle. The important thing is that by using the proposed flow, in only one implementation step on FPGA, a suitable platform for having better performance on multicore has been obtained. Specifically, by tailoring the multicore platform parameters to the values predicted by the simulation and using OpenMP, a speed-up equal to 2.6× has been reached, that provided a response time equal to 822 ms.

By using the sniffer S2 of the monitoring system, the bandwidth of the system bus has been measured during application execution. This is a useful measure that allows to provide traffic information on the bus for the arbiter. Indeed, in the next step of the work, an integration in the proposed flow will be the possibility to add dedicated IP cores at run time by using dynamic partial reconfiguration. In this context, the bandwidth measure allows to understand if the system bus could represent a bottleneck. Results of bandwidth measure are reported in Table 8.

7 Conclusions

This work has defined a design flow to support the development of a system for performance speed-up using OpenMP on a shared memory multicore architectures. The system is intended to be implemented on FPGA. An assumption that the application already worked on a single-core platform on FPGA has been done. Integrated in the final platform, a hardware monitoring system to measure response time without software overhead and to provide other useful metrics has been proposed. The entire flow has been customized for LEON3-based multicore system. The application of the proposed flow to two useful benchmarks, one from OpenMP website and another from a real industrial application, have been proposed.

The application of the flow to other soft-processors can be considered: in order to allow a symmetric multiprocessing with soft-processors, there are hardware and software considerations to be done. From the hardware point of view, there is the need of a system bus that can accept multiple masters. Then, each CPU must have a unique ID. Atomic accesses must be provided from CPUs and cache coherency has to be guaranteed. Also IPI has to be guaranteed in hardware. Each CPU has to be provided with 1 reset signal. These things are required, in general, by a Linux SMP distribution. From the software point of view, there are modifications to be done in hardware dependent source files, to match the hardware modifications, and there is the need of a toolchain to cross-compile applications. Future developments of this work are the refinement of the model of the LEON3 processor in order to have higher precision on the simulation results (in particular, in the execution time) and the porting of the flow to the Nios2 processor.

Acknowledgements
This work has been partially supported by the Artemis-JU ASP 2011 CRAFTERS (GA 295371) and the Artemis-JU AIPP 2013 EMC2 (GA 621429) projects.

Competing interests
The authors declare that they have no competing interests.

Author details
[1] University of L'Aquila, Center of Excellence DEWS, Via Giovanni Di Vincenzo 16/B, 67100 L'Aquila, Italy. [2] Thales Italia, Via Enrico Mattei 20, 66100 Chieti, Italy.

Received: 29 February 2016 Accepted: 4 September 2016
Published online: 15 September 2016

References

1. The OpenMP® API Specification for Parallel Programming. http://openmp.org/wp/. Accessed 29 June 2016
2. The Open Standard for Parallel Programming of Heterogeneous Systems. https://www.khronos.org/opencl/. Accessed 29 June 2016
3. Open MPI: Open Source High Performance Computing. https://www.khronos.org/opencl/. Accessed 29 June 2016
4. MicroBlaze Soft Processor Core. http://www.xilinx.com/products/design-tools/microblaze.html. Accessed 29 June 2016
5. PicoBlaze 8-bit Microcontroller. http://www.xilinx.com/products/intellectual-property/picoblaze.html. Accessed 29 June 2016
6. Nios II Processor. https://www.altera.com/products/processors/overview.html. Accessed 29 June 2016
7. LEON3 Processor. http://www.gaisler.com/index.php/products/processors/leon3. Accessed 29 June 2016
8. CRAFTERS Project. http://www.crafters-project.org/. Accessed 29 June 2016
9. M Aldinucci, S Ruggieri, M Torquati, Porting decision tree algorithms to multicore using fastflow. Lect. Notes Comput. Sci. **6321**, 7–23 (2010). doi:10.1007/978-3-642-15880-3_7
10. FastFlow: Programming Multi-core. https://sourceforge.net/projects/mc-fastflow/. Accessed 30 June 2016
11. Y Ardila, N Kawai, T Nakamura, Y Tamura, in *Design Automation Conference, 18th Asia and South Pacific*. Support tools for porting legacy applications to multicore, (2013), pp. 568–573. doi:10.1109/ASPDAC.2013.6509658
12. E Matthews, L Shannon, A Fedorova, in *22nd International Conference on Field Programmable Logic and Applications (FPL)*. Polyblaze: From one to many bringing the microblaze into the multicore era with linux smp support (IEEE, 2012), pp. 224–230. doi:10.1109/FPL.2012.6339185
13. P Huerta, J Castillo, C Sánchez, JI Martínez, in *International Conference on Reconfigurable Computing and FPGAs*. Operating system for symmetric multiprocessors on fpga (IEEE, 2008), pp. 157–162. doi:10.1109/ReConFig.2008.43
14. S-D David, A-T David, Evaluation of mpsoc operating systems. Int. J. Eng. Pract. Res. (IJEPR). **2**, 49–54 (2013)

15. O Serres, VK Narayana, T El-Ghazawi, in *International Conference on Reconfigurable Computing and FPGAs*. An architecture for reconfigurable multi-core explorations (IEEE, 2011), pp. 105–110. doi:10.1109/ReConFig.2011.10
16. Rapita Blog: Explaining the Difference Between Execution Times and Response Times. https://www.rapitasystems.com/blog/difference_between_execution_times_and_response_times/. Accessed 29 June 2016
17. GNU Gprof. https://sourceware.org/binutils/docs/gprof/. Accessed 29 June 2016
18. L Shannon, P Chow, in *Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*. Using reconfigurability to achieve real-time profiling for hardware/software codesign, (2004), pp. 190–199. doi:10.1145/968280.968308
19. J Tong, M Khalid, Profiling tools for fpga-based embedded systems: Survey and quantitative comparison. J. Comput. **3**, 1–14 (2008). doi:10.4304/jcp.3.6.1-14
20. L Shannon, E Matthews, NC Doyle, A Fedorova, Performance monitoring for multicore embedded computing systems on fpgas. CoRR. **abs/1508.07126**, 68–72 (2015)
21. N Ho, P Kaufmann, M Platzner, in *Field Programmable Logic and Applications (FPL), 24th International Conference On*. A hardware/software infrastructure for performance monitoring on leon3 multicore platforms, (2014), pp. 1–4. doi:10.1109/FPL.2014.6927437
22. L Diaz, E Gonzalez, E Villar, P Sanchez, in *Design of Circuits and Integrated Circuits (DCIS), Conference On*. Vippe, parallel simulation and performance analysis of multi-core embedded systems on multi-core platforms, (2014), pp. 1–7. doi:10.1109/DCIS.2014.7035584
23. The LLVM Compiler Infrastructure. http://llvm.org/. Accessed 29 June 2016
24. A Gerstlauer, in *Rapid System Protyping, 21st IEEE International Symposium On*. Host-compiled simulation of multi-core platforms, (2010), pp. 1–6. doi:10.1109/RSP.2010.5656352
25. G Valente, V Muttillo, L Pomante, F Federici, M Faccio, A Moro, S Ferri, C Tieri, in *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. A flexible profiling sub-system for reconfigurable logic architectures, (2016), pp. 373–376. doi:10.1109/PDP.2016.86
26. A Moro, F Federici, G Valente, L Pomante, M Faccio, V Muttillo, in *Intelligent Solutions in Embedded Systems (WISES), 12th International Workshop On*. Hardware performance sniffers for embedded systems profiling, (2015), pp. 29–34. http://ieeexplore.ieee.org/document/7356977/
27. GNU Libgomp. https://gcc.gnu.org/onlinedocs/libgomp/. Accessed 29 June 2016
28. AMBA 2.0 Specifications. https://www.arm.com/products/amba-open-specifications.php. Accessed 30 June 2016
29. Buildroot Website. https://buildroot.org/. Accessed 30 June 2016
30. SPARC LINUX 4.4.2. http://www.gaisler.com/anonftp/linux/linux-2.6/toolchains/. Accessed 30 June 2016
31. ML605 Development Board. http://www.xilinx.com/products/boards-and-kits/ek-v6-ml605-g.html. Accessed 30 June 2016
32. Y Shang, W Ruml, Y Zhang, MPJ Fromherz, in *ACM MobilHoc, Annapolis*. Localization from mere connectivity (ACM, 2003), pp. 201–212. doi:10.1145/778415.778439
33. Y Shang, W Ruml, in *INFOCOM. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Improved mds-based localization, (2004), pp. 2640–2651. doi:10.1109/INFCOM.2004.1354683
34. B Wei, W Chen, X Ding, in *Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS)*. Advanced mds based localization algorithm for location based services in wireless sensor network, (2010), pp. 1–8. doi:10.1109/UPINLBS.2010.5654301
35. SPARC V8 32-bit Processor. http://www.microsemi.com/products/fpga-soc/design-resources/ip-cores/. Accessed 30 June 2016