POLITECNICO DI TORINO
Repository ISTITUZIONALE

Improving the design flow for parallel and heterogeneous architectures running real-time applications: The PHARAON FP7 project

Original
Improving the design flow for parallel and heterogeneous architectures running real-time applications: The PHARAON FP7 project / Héctor, P., Alejandro, N., Pablo, P., Eugenio, V., Florian, B., Michel, B., Albert, C., Lazarescu, M.T., Lavagno, L., Andrei, T., Miguel, G., Manuel, P.. - In: MICROPROCESSORS AND MICROSYSTEMS. - ISSN 0141-9331. - ELETTRONICO. - 38:8(2014), pp. 960-975. [10.1016/j.micpro.2014.05.003]

Availability:
This version is available at: 11583/2552336 since: 2020-10-22T20:39:13Z

Publisher:
Elsevier

Published
DOI:10.1016/j.micpro.2014.05.003

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Elsevier postprint/Author's Accepted Manuscript

© 2014. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/.The final authenticated version is available online at:
http://dx.doi.org/10.1016/j.micpro.2014.05.003

(Article begins on next page)

08 September 2026

Improving the design flow for parallel and heterogeneous architectures running real-time applications: The PHARAON FP7 project

Héctor Posadas [a,*], Alejandro Nicolás [a], Pablo Peñil [a], Eugenio Villar [a], Florian Broekaert [b], Michel Bourdelles [b], Albert Cohen [c], Mihai T. Lazarescu [d], Luciano Lavagno [d], Andrei Terechko [e], Miguel Glassee [f], Manuel Prieto [g]

[a] *University of Cantabria, Santander, Spain*
[b] *Thales Communications & Security, Gennevilliers, France*
[c] *INRIA and École Normale Supérieure, Paris, France*
[d] *Politecnico di Torino, Torino, Italy*
[e] *Vector Fabrics, Eindhoven, The Netherlands*
[f] *IMEC, Leuven, Belgium*
[g] *Tedesys, Santander, Spain*

A B S T R A C T

In this article, we present the work-in-progress of the EU FP7 PHARAON project, started in September 2011. The first objective of the project is the development of new techniques and tools capable to guide and assist the designer in the development process, from UML specifications to implementation and debug on multicore platform. This tool chain will offer the possibility to propose and implement several parallelization strategies and drive the designer into implementation steps. The second objective of the project is to develop monitoring and control techniques in the middleware of the system capable to automatically adapt platform services to applications requirements and therefore reduce power consumption in a transparent manner for applications.

1. Introduction

Recent market data show that a critical increase in the number of multicore architectures used in projects is currently taking place [1]. During the last decade, those architectures have expanded from only targeting some very specific domains with very high processing needs (e.g. engine control), to become the actual implementation paradigm for mainstream embedded systems. This kind of architectures is getting increasing acceptance into the computing industry, and has become very common in the notebook and tablet markets, among others. This enabled the latest and greatest embedded systems to integrate a growing range of complex functionalities. A smart phone, for example, is capable to communicate through 3G and WIFI connections while running other applications on Android or Windows Phone. It integrates phone services with high performance graphics and sophisticated software applications such as real-time video and audio.

Designers are facing challenging problems as hardware architectures are evolving faster than multicore software development techniques. These techniques are not yet capable to provide efficient methodologies to exploit the full potential of multicore architectures satisfying all the requirements of embedded systems, including performance and power consumption. Accurately predicting the performance of an application implemented on such architectures has become very difficult, because of numerous factors such as cache coherency. Moreover, commonly taught programming models, that are generally based on sequential languages, are no longer sufficient, since early consideration of parallelism in applications has become critical. The lack of efficient software design techniques increases both software development costs and implementation risk in terms of costs and delays. Parallelism, heterogeneity, complex memory structures, efficient power monitors and controllers, are among the list of new functionalities provided by recent multicore systems that require to be adequately tackled by new design

* Corresponding author.
E-mail addresses: posadash@teisa.unican.es (H. Posadas), nicolasa@teisa.unican.es (A. Nicolás), pablop@teisa.unican.es (P. Peñil), villar@teisa.unican.es (E. Villar), florian.broekaert@thalesgroup.com (F. Broekaert), michel.bourdelles@thalesgroup.com (M. Bourdelles), Albert.Cohen@inria.fr (A. Cohen), mihai.lazarescu@polito.it (M. T. Lazarescu), luciano.lavagno@polito.it (L. Lavagno), andrei@vectorfabrics.com (A. Terechko), glasseem@imec.be (M. Glassee), mprieto@tedesys.com (M. Prieto).

tools, as proposed by the PHARAON (Parallel and Heterogeneous Architecture for Real-time ApplicatiONs) project.

1.1. The PHARAON project

The PHARAON project is a European collaborative initiative between universities, research labs and companies that is aimed at proposing solutions to overcome these limitations. It is sponsored by the European Commission that covers part of the costs and assists partners in the project management.

The objective of PHARAON is to achieve a breakthrough towards broader adoption of multicore architectures and to enable the development of complex systems with high processing needs and low-power requirements. For such purpose, the project focuses on solving two major problems appearing in these types of systems. First, the development of parallel software, capable of exploiting multiple processor cores, is much more complex and, therefore, more expensive than traditional sequential software, which increases the product cost. Second, the increased complexity of services provided by the systems requires more energy and, hence, is associated with a reduction of autonomy.

To overcome these problems, the PHARAON project targets the development of two different sets of techniques and tools, aiming at best exploiting the low-power capabilities of modern multi-core processors, both at design time and at runtime (Fig. 1). These techniques and tools tackle both the programming and power management challenges mentioned previously.

The first set directly affects the design flow, from UML/MARTE specifications to implementation on multicore platforms. The objective is to assist the designer in finding the most adequate software architecture while taking into account hardware constraints at design time. To do so, tools developed in PHARAON can evaluate the parallel structure of an application and propose improvements, in terms of parallelization constructs. At the same time, the toolset will be able of automatically generating the multi-processor embedded code required to deploy the communicating SW components on the processing cores of the system, including DSPs and GP–GPUs.

The second set of techniques and tools affect the runtime behavior of the application. The objective is to adapt the performance of the platform (frequency and voltage, for example) in order to consume only the required energy. For this purpose, project partners are developing monitoring and control techniques that are integrated in the code generated at design time to map the SW to the processors of the platform. This middleware automatically adapts platform services to application requirements during execution, and therefore reduces power consumption. A reconfiguration system and a low power scheduler are integrated with other run-time components on top of the platform to do so.

As a result, the PHARAON project has the goal of reducing the software development cost by 25% and to increase the battery life of embedded systems by nearly 20%.

The project is coordinated by Thales Communications & Security which is a large French company. Tedesys (Spain) and Vector Fabrics (Netherlands) are two SMEs completing the industrial partners. Academic partners include Politecnico di Torino (Italy), Ecole Normale Supérieure (France) and University of Cantabria (Spain). Finally, the Interuniversitary Micro-Electronics Centrum research institute (Belgium) completes the consortium.

As a result, the next sections present the improvements achieved during the first two years of the project, in which design tools have been developed and their application to the project use cases have started. The second section summarizes the state of the art in the area. Then, the design flow proposed in the project is detailed in the third section. The fourth section presents the design-time tools developed during the project, including their results. The fifth section is devoted to the runtime management tools. Then the application of the tools to several industrial use cases is described. Finally the conclusions highlight the project perspectives.

2. Evolution beyond the state of the art

Code parallelization is one of the most widely studied topics in compilers for parallel machines since the 1970s. However, the level of parallelism that can be identified using automated techniques is very limited, since they require specific coding styles (e.g. perfectly nested loops, no conditionals and affine indexing) and hence have limited applicability.

Recent approaches like the Compaan project at the University of Leiden [2], or the Pico Express high-level synthesis software from

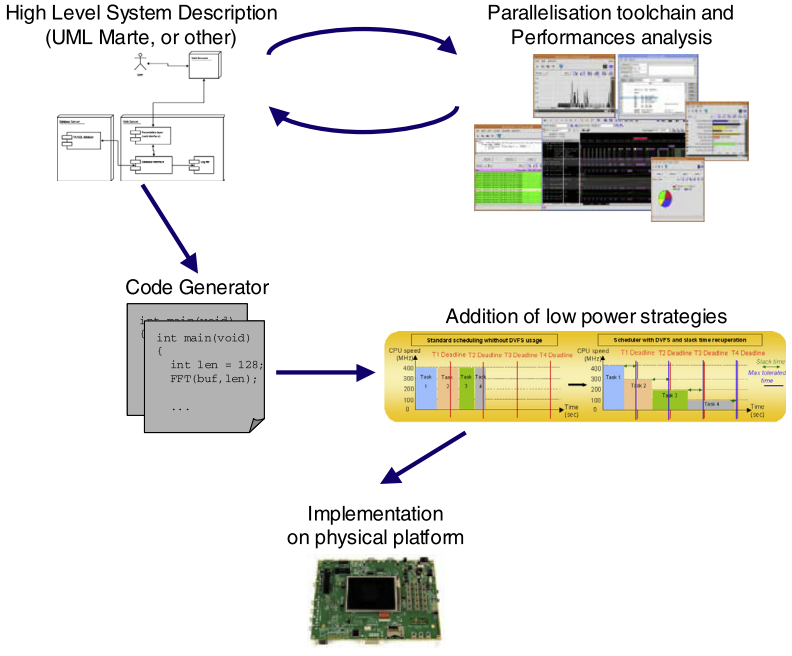

Fig. 1. PHARAON global approach and tools interactions.

Synopsys [3] are interesting examples in this area. Other efforts like MORPHEUS [5], CRISP [6] and MEGHA [7], produce parallel code for execution on embedded or other special type platforms but usually are tailored to the target architecture. Several compilation and debugging tools, often based on dedicated extensions of the C language like OpenCL and CUDA, have also been proposed for GPU architectures.

The ARTEMIS ASAM project [8] is aimed at performing efficient design space exploration in order to optimize Application Specific Instruction-set Processors (ASIPs) to specific applications. The goal of the ALMA project [4] is to tailor the application to the available hardware. The FP7 TERAFLUX project [9] proposes new programming models and solutions for harnessing large-scale multicore platforms in the area of thousand cores in an efficient way by exploiting data-flow parallelism.

On the other hand, the goal of the PHARAON project is not limited to parallelism discovery and exploitation, but integrates it with solutions oriented at optimizing power consumption, while using heterogeneous cores of the target platform. The PHARAON flow starts from UML/MARTE models and combines code synthesis, parallelism analysis and stream-oriented OpenMP extensions with power and performance evaluation tools and run-time managers. That combination of design-time and run-time tools enables solving the trade-off between performance improvements and power consumption minimization.

UML is a very common solution for high-level system design. [20] pays special attention to the importance of UML models for industrial applications and the effort that they require. UML is a very broad and flexible language. Hence different profiles have been derived for specific application area. Currently MARTE [21] is the OMG standard profile for Modeling and Analysis of Real-Time and Embedded systems. Several UML-based methodologies also focus on HW/SW communication synthesis. In [22], a semi-automatic solution using Remote Method Invocation (RMI) semantics for generating HW/SW infrastructure from UML models is presented. Similarly, [23] describes a flow to generate code from high-level MARTE for implementation on dynamically reconfigurable SoCs.

Regarding code parallelism analysis, the ParTools toolset has been developed during the PHARAON project. It goes beyond previous techniques, such as the one proposed in [41] by providing techniques based on data compaction and advanced visualization. They are meant to effectively display huge trace data sets, and thus improve the developer analysis time while searching for the best parallelization opportunities. In the same vein, the tools from CriticalBlue can predict application performance under different thread decompositions, and display the corresponding inter-thread dependencies. Like in our case, the assessment of parallelization opportunities is bound by the quality of the test bench used. However, the visualization capabilities are closer to those of a traditional profiler.

After analyzing parallelization opportunities, solutions to implement them are required. OpenMP has been selected as a background technology, since, it is the de facto standard to program shared memory parallel computers. An extension of OpenMP for data-flow and stream computing, called OpenStream, has been developed in the project, leveraging previous results in the area [48,49].

Finally, in the context of run-time management for low-power design, different task-scheduling techniques have been also developed before the project. These techniques cover solutions such as the proportional, integral and derivative (PID) controllers [24], or fuzzy logic controllers [25]. Again, various European projects have tackled these issues [26–31]. Traditional approaches can be roughly classified into either pure design-time approaches or pure run-time approaches. In general, they suffer from the following drawbacks.

First, some of them are applicable only for single-processor platforms [44], or for homogeneous multi-processor platforms [45], but not for heterogeneous multi-processor platforms. Second, none of the existing approaches proposes a complete framework. Some of them are based only on task mapping and scheduling (a good overview of traditional scheduling algorithms can be found in [46]). Some others are based only on slowing or shutting down the platform resources [32] and on Dynamic Voltage and Frequency Scaling (DVFS) [33–36]. Third, the objective of the majority of these approaches is performance optimization [37–40,50], and not power consumption optimization. Finally, design-time approaches involve slow heuristics [36,42,43] using Integer Linear Programming (ILP) algorithms and cannot be used at run time.

The addition of parallelism to the set of platform parameters significantly increases the design space of application implementations. Thus, the PHARAON project developed innovative and efficient techniques for run-time power management, which are needed to extend the traditional approaches for power consumption optimization [47].

3. PHARAON system design flow

The targeted design flow in PHARAON drives the design from UML specifications to implementation of cross-compiled code onto the target platform. During this process, parallelization analysis, code synthesis and power management components are added to the original functional code in order to optimize the use of the target platform.

As depicted in Fig. 2, the proposed flow starts by modeling the top-level application architecture with a high level component-based approach (UML/MARTE). The use of this methodology enables the PHARAON flow to separately map various application components to resources within homogeneous, heterogeneous and distributed platforms. In order to enable this, the business code for each component (C/C++ files) has to be provided.

This approach enables a two level design approach, combining a coarse grain, and a fine grain level. The component-based approach allows the user to select different coarse-grained deployment strategies and to explore parallelization between relatively large components (e.g. the layers of a protocol stack) though automatic code generation. Then, the internals of each component (e.g. a specific MAC or PHY algorithm) can be analyzed and optimized at a finer grain using the following tools of the flow.

For the coarse grain level, a code generator has been developed to automatically generate the wrapper code used for deployment of and communication among the different SW components. The tool produces the source files that are used as inputs to the subsequent stages of the toolchain, including analysis and optimization on the host computer and later target platform mapping.

In a first stage, the C code of a UML component to be further parallelized is sent to a performance simulator in order to evaluate the execution time and power cost of the different statements of the code.

In a second stage, the parallelization tool allows the designer to understand the underlying computational structure of the C code, and use this to further parallelize the internal code of the component. Based on user decisions, the parallelization tool then generates code integrating OpenMP/OpenStream parallelization directives.

In a third stage, the optimized code is simulated again on the performance evaluation tool both to evaluate the quality of the parallelization (and optionally improve it by using the parallelization tool again), and to obtain the information required for run-time optimization. Alternatively, the code can be implemented and measured onto the physical platform, if that is already available.

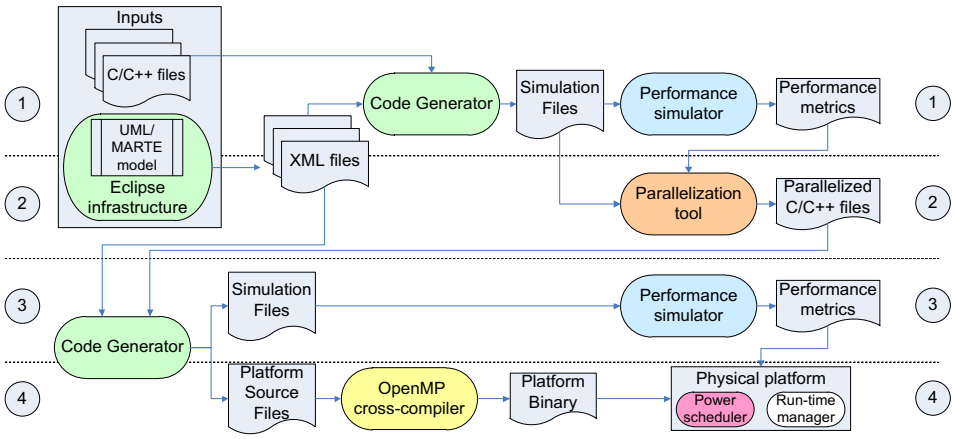

Fig. 2. PHARAON design flow.

Finally, different runtime managers (a reconfiguration manager and a low-power scheduler) are deployed on the physical platform in order to reduce power consumption while ensuring required application performance. Here, performance and power traces collected by the simulator help to refine the power management strategy.

The project has been organized around 6 different workpackages, as detailed in Fig. 3. The first step in the project has been to list all industrial requirements and to specify both the interfaces among the tools and the demonstrators that will be used to assess them. This work is covered in the first workpackage of the project, and constitutes the skeleton of the project and produces all the reference documents that will drive the development in the following workpackages. Secondly (WP2), the project focuses on the implementation of design-time methods and tools that facilitate the development and implementation of applications targeting parallel and heterogeneous platforms, as described in Section 4.

At the same time, WP3 is aimed at runtime resource and power management. One goal of these run-time solutions is to develop techniques to schedule multiple applications on the platform. Another goal is to develop software components, integrated in the middleware of the system, capable to adapt hardware platform configuration to application needs depending on the required quality of service.

To evaluate the validity of these tools, three case studies are begin designed to demonstrate the efficiency, validity and applicability of the developed tools and methods. It also integrates an industrial evaluation that ensures the industrial viability of the proposed solutions. This work focuses WP4.

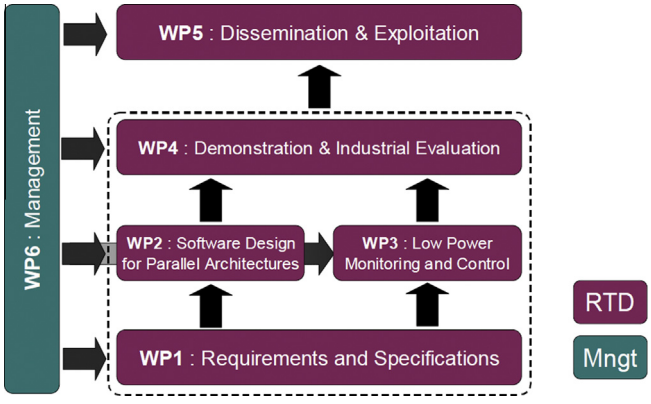

Fig. 3. PHARAON workpackages organization.

Dissemination and exploitation of results is covered in WP5, with main focus on ensuring the widest visibility of the project results and promoting the industrial exploitation of the most promising results. Finally, WP6 deals with project management and integrates scientific and technical management as well as communications with the European Commission.

4. Design-time tools for parallelization and heterogeneous platform support

4.1. UML/MARTE modeling

In order to support all the different stages of the flow, a powerful high-level modeling methodology has been defined. It is based on UML and it follows a component-based approach applying the Model-Driven Architecture (MDA) principles to the development of HW/SW embedded systems. Additionally, the MARTE profile has been used to consider all the specific characteristics specifically related to embedded system design (Fig. 4).

The proposed methodology is software centric, as it assumes an allocation of components to programmable processors.

Following the proposed methodology designers can completely describe the system, enabling automatic generation of the input code required by the different tools of the design flow. For such purpose, designers must describe in various UML/MARTE views the system functionality, the target platform and the resource allocation.

However, since the methodology has to support a broad variety of platforms, several extensions to the basic UML/MARTE profile were required. The main new issues to be covered are heterogeneity, parallelization, I/O support and run-time power management. Thus, specific enhancements are proposed for all these points.

In order to support adequate mapping to heterogeneous systems, three major issues have been detected. First, it is required to generate different executables with the components mapped to each resource. Second, it is required to ensure the correct access to shared information, maintaining the memory architecture of the original source code. And third, the model must handle multiple file versions for the same component, each one optimized for each possible mapped resource (e.g. GPU, DSP and CPU), including files for host simulation.

In order to solve the first two points, the system mapping is performed in two steps: first components are mapped to memory spaces and then these memory spaces are mapped to resources (Fig. 5). As a result, different executables are generated for the system, one for each memory space. Additionally, to support different

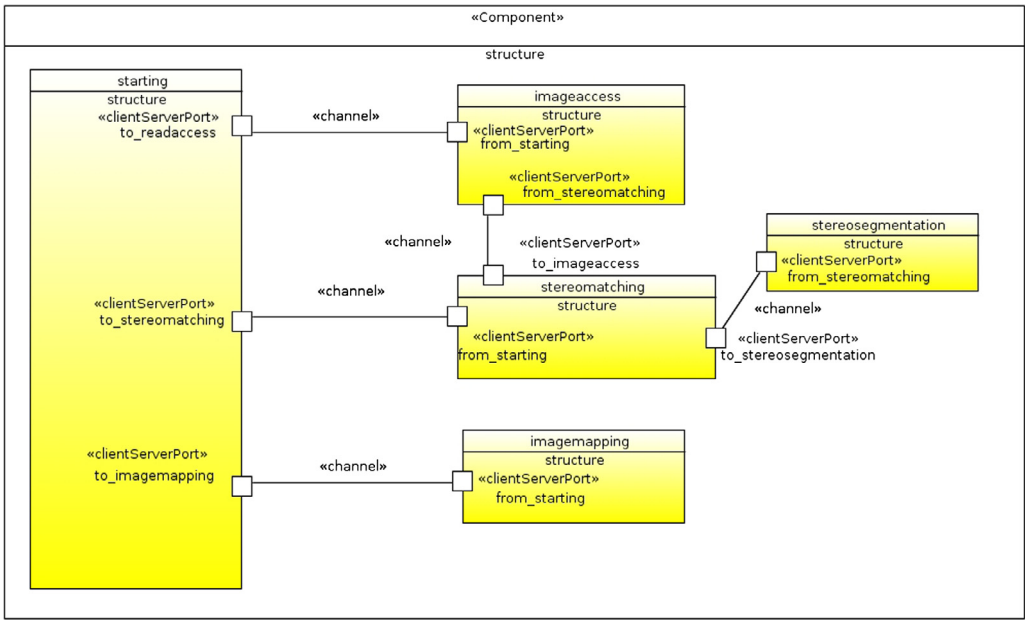

Fig. 4. Excerpt of PIM of the stereovision application.

files for the same component, different attributes have been added to the "file" stereotype.

Additionally, the information described in the UML model also enables the automatic generation of ad-hoc communications infrastructures. To optimize it, different channel semantics (listed below) have been added to MARTE profile. This allows designer to optimize the system concurrent architecture at a coarse grain, by modifying the relationships between the system components.

4.2. Code generator

From the information included in the UML/MARTE graphical model, the inputs for the different tools of the flow are created. This generation process is performed in two steps. First, an Eclipse plugin has been developed, capable of transforming the graphical model into a set of XML files. From these files, the generator produces a set of C files that includes the code required to initialize all the components mapped to each memory space, the C wrappers that enable the communication among the application components, the different agents handling incoming communication requests, and the platform-specific compilation scripts.

The interface wrappers use the facilities provided by a communication library that has been specifically developed to implement the various communication mechanisms. These wrappers are implemented in a three layer structure, in order to have enough flexibility to support multiple communication semantics and mappings. One layer implements communication semantics. Characteristics such as synchronous or asynchronous calls, FIFOs, data joining or splitting and synchronized or prioritized accesses from different clients are implemented in this step. Then, arguments are adapted to be transferred depending on the communication type (within the memory space, in different spaces of the same OS, in different OSs or resource types). Finally, the infrastructure obtains from the communication library the generic transfer functions for the required communication types required on each case.

At the same time, the automatic generation of wrappers enables easy mapping of components to different resources, considering distributed platforms and heterogeneous systems, which can contain devices such as DSPs, co-processors such as the NEON or GP–GPUs. The mapping to these heterogeneous resources has been thoroughly verified with the project use cases, as described in Section 6.

This work has been described through several papers, such as [51–56], where more details and specific results can be found.

4.3. Pareon's performance and energy simulator

Within the PHARAON project, the performance analysis of C applications on the target hardware platform is performed by the Pareon tool, which also estimates energy consumption. The

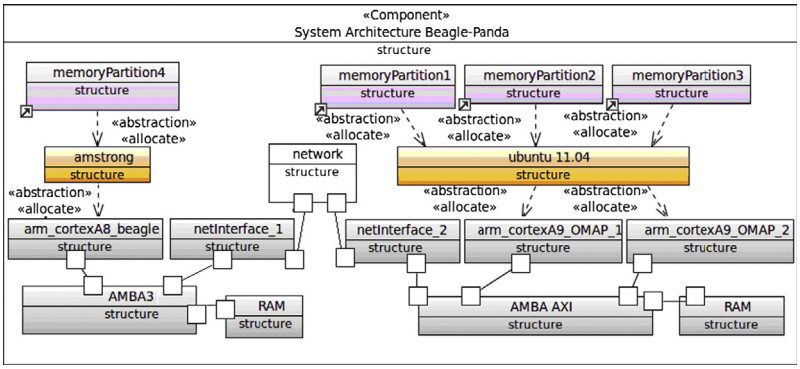

Fig. 5. Platform mapping example.

estimates are fed into the parallelization tool to help parallelize performance and optimize memory bottlenecks of the code, while tracking effects on power consumption. Furthermore, the energy estimates are used by the low power scheduler that can select the most power-efficient operating mode of the system. The modeled target hardware in the context of PHARAON includes ARM Cortex A9 and Intel Core 5 multicore processors. The Pareon tool-suite also features leading-edge interactive parallelization capabilities (akin to those described in the next section), which are, however, outside the project's scope.

The Pareon tool is a collection of command line interface (CLI) tools and a GUI. Within the PHARAON project the CLI tools are used in the automated toolchain, while the GUI enables human inspection of the modeling results. The input to the tools is the source code of a C or C++ program. The input program should comply with the ANSI C99 or ANSI C++98 standards and may contain selected POSIX function calls. In particular, parallel programs with POSIX threads such as pthread_create() and pthread_join() can be analyzed in terms of their timing and energy consumption. Furthermore, the latest release of Pareon supports OpenMP pragmas, in order to handle the fine grained parallelization strategy used in PHARAON. Overall, analysis of parallel code enables closing the loop in the PHARAON toolchain and optimizing already parallelized applications by looking at their performance and energy consumption estimates.

The *vfcc* compiler translates the input source code into a generic executable for an intermediate instruction set architecture, which is independent of the target processor. Then the generic executable is run in the Pareon simulator with the provided execution environment, including necessary test data, input files, environment variables, etc. During the execution various statistics such as instruction counts and memory behavior are collected. Finally, the Pareon report command converts these statistics into estimates for a particular hardware target platform and generates an XML output file with performance and power estimates of the input program, to be used by the parallelization tool.

The internal Pareon toolflow for performance analysis is shown in Fig. 6 and an extensive documentation of the Pareon functionality is available online at [10].

Pareon has been successfully used with PHARAON case studies, such the Software Defined Radio and the depth estimation algorithm, as described in Section 6. Furthermore, Pareon's parallelization capabilities were applied to analyze available concurrency in complex industrial applications of the project, as well as to construct their multithreaded implementations. Currently Pareon results are being integrated in the PHARAON optimization flow to steer parallelization and power management.

4.4. ParTools parallelisation toolset

The ParTools toolset [16,17] addresses the parallelization of legacy sequential C software that can include also complex control structures, pointer operations, and dynamic memory allocation. It can discover both task and data parallelization opportunities and can be used for any parallelization technique, including in particular both the UML/MARTE-based method used in the PHARAON flow for coarse grain parallelization the OpenMP/OpenStream pragmas used for fine grain parallelization.

The toolset flow shown in Fig. 7 is divided in four stages: (I) source instrumentation, (II) run-time execution trace profile and data dependency collection and compaction, (III) graphical visualization and analysis of execution data, and (IV) source code parallelization. Its operation is controlled from the Code::Blocks IDE for C and C++. The IDE supports also cross-referencing between the execution trace visualized in stage III and the sequential C project source.

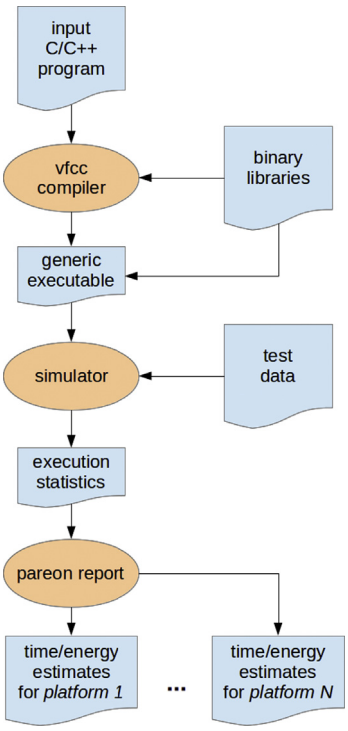

Fig. 6. Pareon performance analysis toolflow.

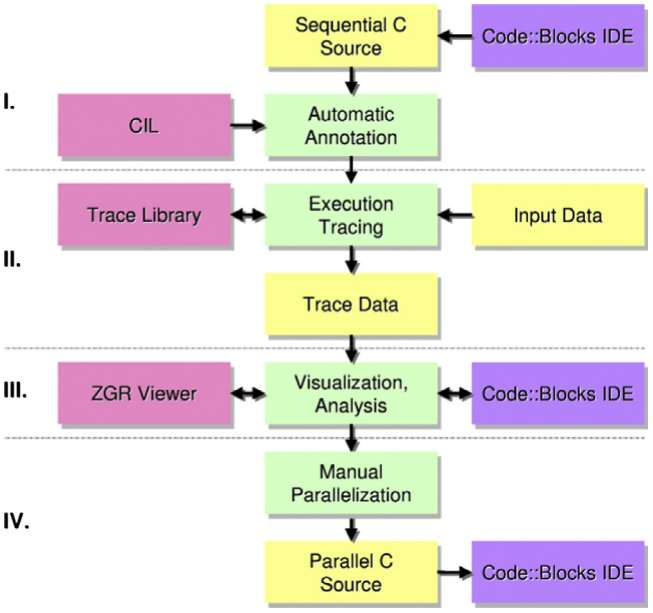

Fig. 7. ParTools toolset parallelization flow.

The automatic annotator used in stage I instruments the sequential source for run-time data dependency collection and can be easily integrated into make-based projects. During program run, the data retrieved by the instrumentation are collected, analyzed and compacted by a library linked with the instrumented program. At the end of program execution, the data collected are saved for use by the graphical visualization and analysis interface.

The graphical visualization interface displays both the program execution profile and the data dependencies to facilitate the search for parallelization opportunities. These are shown as a graph with nodes representing program control (e.g., statements, loops,

function calls) and edges representing the data dependencies. All elements are uniquified based on the execution call stack. In order to make the huge data dependency graph easy to visualize and understand, the nodes for complex program structures (e.g., loops, function calls) can be "folded" to represent the cumulative data (both execution and data dependencies) for all the execution call stacks rooted there.

Figs. 8 and 9 show an analysis view for the stereovision application presented in Section 6. The rectangular nodes correspond to loops (folded with their underlying call stack), while the elliptic ones represent function calls (also folded with their underlying call stack). Both types of nodes (loops and functions) that fold all data dependencies below them are called node folds in the following. The two loop folds with stronger colorization include 53% and 18% of the program execution time respectively, which makes them significant candidates for parallelization. Moreover, they have no strong data dependency among them and may be suitable for data-parallel rewriting.

This abstraction mechanism, including graph re-rooting at any level of interest, is essential to compactly show the most important points for parallelization opportunities of a DDG (Data Dependence Graph) that can have millions of nodes and edges in its fully exploded form.

The data dependency view of a selected DDG node is another important feature for parallelization candidate analysis. As mentioned above, a DDG node often represents not just a C statement, but rather the fold of a whole call stack, i.e., a collapsed view of the statement and its descendants in the call tree, including its nested statements, and those of all functions called by it. For any fold in the current scope of interest (i.e. not included in a fold above it), the data dependency view shows a summary of the input and output data dependencies only of that node. This is an essential information for any parallelization mechanism, language and style. The data dependency view of a given call stack (i.e., folded node), as shown in Fig. 10, is organized in layers:

1. The top layer displays the leaf nodes (C statements) that produce the incoming data dependencies.
2. The next layer displays the data produced by these statements, in parallelogram-shaped boxes.
3. The middle layer displays the statements in the selected fold node that consume or produce data.
4. The next layer displays the data produced by selected fold statements, again in parallelogram-shaped boxes.
5. The bottom layer displays the leaf nodes that consume the outbound data dependencies.

This view can substantially speed up the parallelization decisions made by the developers. Note that these dependencies are typically difficult to extract through code inspection or static code analysis, since the producers and consumers can be at various depths in different call stacks, and the dependencies can be through any type of data (dynamic, local, global, etc.) and complex control structures. The ParTools toolset tracks data dependencies through variables inside any scope or storage class, including those that are dynamically allocated on the heap.

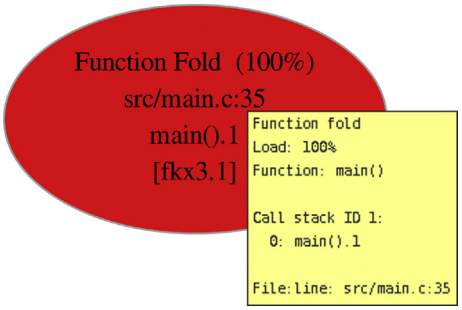

Fig. 8. Initial view folds all execution and dependencies under main() function.

4.5. OpenMP extension for data-flow and stream parallelism

OpenStream (http://www.di.ens.fr/OpenStream) is a stream programming language, designed as an incremental extension to the OpenMP parallel programming language [12]. It allows expressing arbitrary task-level data flow dependence patterns. Programmers expose task parallelism and provide data-flow information to the compiler through compiler annotations (pragmas), used to generate code that dynamically builds a streaming program. The language supports nested task creation, modular composition, variable and unbounded sets of producers/consumers, and first-class streams. These features, enabled by an original GCC-based compilation flow, allow translating high-level parallel programming patterns into efficient data-flow code.

Data-flow execution is essential to reduce energy consumption, one of the primary focuses of the PHARAON project, by reducing the severity of the memory wall. This is achieved in two complementary ways: (1) thread-level data flow naturally hides latency; and (2) decoupled producer–consumer pipelines favor on-chip communication, bypassing global memory. Furthermore, OpenStream has shown excellent performance in comparison with state-of-the-art parallel programming environments like StarSs, as illustrated in Fig. 11, showing[1] the speedups achieved by OpenStream (red) and StarSs (blue) against sequential execution for a block-sparse matrix LU factorization on a dual-socket AMD Opteron 6164HE machine with 2×12 cores at 1.7 GHz. Key to the efficient execution of OpenStream programs is our optimized runtime system, providing low-overhead synchronization and work-stealing scheduling.

Work stealing is a central component of the OpenStream runtime library, allowing for efficient lock-free scheduling of lightweight tasks. The dynamic scheduler has been ported to the $\times 86$ and ARM architectures, with a focus on correctness and performance. Improving on Chase and Lev's concurrent doubly-ended queue, OpenStream includes a state-of-the-art work stealing implementation. The ARM version of the algorithm is specifically optimized for its weak memory model. Moreover, based on recent progress in the formalization of memory consistency, we established the first proof of the relaxed double-ended queue for such a processor [11].

Our experiments show that the optimized ARM code, of which two versions have been written in C11 and native inline assembly, generally outperforms the original sequentially consistent Chase-Lev in a variety of benchmarks, including a selection of standard fine-grained task-parallel computations (Fig. 12). These results provide the foundation for a robust parallel library, and pave the way for further research into correct lock-free algorithms for run-time support.

From this successful experience, we went on with one other critical concurrent data structure for parallel languages and embedded multiprocessors: Single-Producer, Single-Consumer (SPSC) FIFO queues. They arise from a variety of parallel design patterns and from the distribution of Kahn process networks over multiprocessor architectures. A fine-tuned FIFO implementation translates into higher communication bandwidth and lower communication latency. The latter is key to facilitate the satisfaction of real-time constraints and reduce the memory footprint of in-flight computations, a critical asset for memory-starved embedded processors and

[1] For interpretation of color in Fig. 11, the reader is referred to the web version of this article.

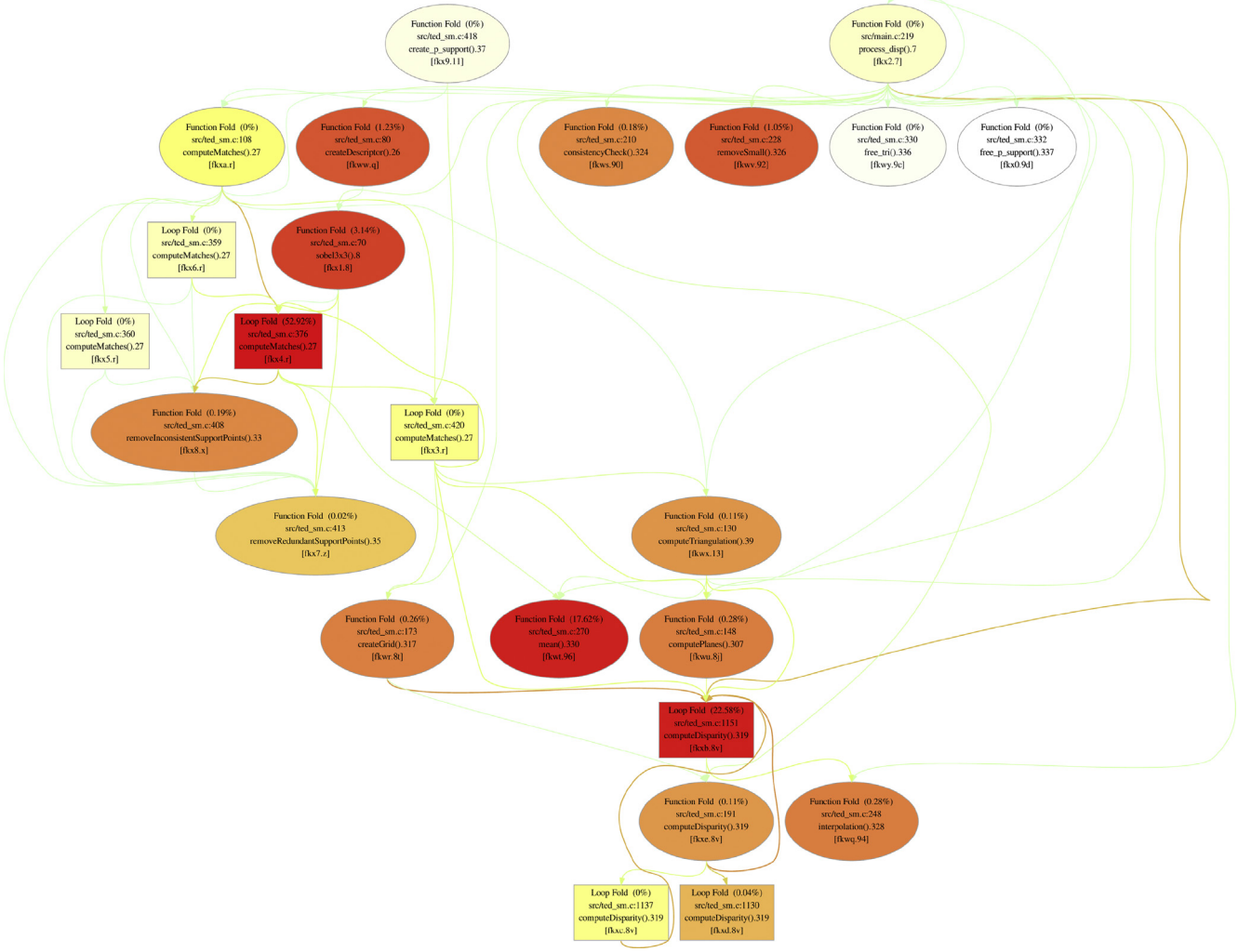

Fig. 9. Analysis of the stereovision application. The two loop folds with stronger colourization include 53% and 18% of the program execution time, which makes them significant candidates for parallelization. Moreover, they have no strong data dependency among them and may be suitable for data-parallel rewriting.

many-core architectures. This motivates the search for the FIFO queue with the highest throughput for a given buffer and batch size.

Formal reasoning about SPSC bounded queues dates back to the seminal work of Lamport, who proved that this algorithm does not need any additional synchronization primitives such as locks to work properly. Our goal is twofold: to offer portability and correctness through a proven, concurrent implementation in C11, and to offer performance through advanced caching and batching extensions of the algorithm, as well as relaxing the hypotheses on memory ordering, leveraging the low-level atomics in C11 with relaxed memory consistency.

The solution we propose is called WeakRB [19]. Along with a complete proof using an axiomatic memory model of C11, we validate its portability and performance is validated over 3 architectures with diverse hardware memory models, including 2 embedded platforms. Our experiments demonstrate consistent improvements over state-of-the-art algorithms for a wide range of buffer and batch sizes. As shown in Fig. 13, WeakRB outperforms one of the state of the art algorithms, MCRB [18], sustaining close-to peak throughput in core-to-core streaming communications.

Overall, our foray into streaming data-flow languages has led to the design of a tightly integrated collection of compilation, code generation, and concurrent runtime algorithms for task-level parallel programming. The complete design has proven particularly effective on embedded multicores. In the future, we will work on complementing these techniques with real-time scheduling policies and low-power adaptation schemes.

4.6. Data-flow synchronous programming of parallel embedded systems

The PHARAON project also investigates longer-term research directions, such as the design and implementation of safety–critical embedded software running on parallel multicore processors. Heptagon is a data-flow synchronous language devoted to the design and implementation of embedded software. Its ancestors Lustre and Scade have met a large success in the field of safety–critical real-time systems, offering a clean semantics, with a robust, efficient, and traceable compilation flow, while enforcing bounded resource and bounded reaction-time guarantees. However, compilation schemes for such languages lead to very efficient, but sequential code. Various distribution and parallelization approaches can be applied a posteriori, at the price of performing a non-modular and hardly scalable static analysis of the generated code to guarantee efficiency and correctness. We provide a clean alternative to these approaches, giving the designer explicit control on the de-synchronization and on the distribution of the program (or model) [13].

Classical issues are summed up in the classical "slow_fast" example sketched in Figs. 14 and 15. A slow process communicates with a faster one at the rate of the slow process. Parallel execution

Fig. 10. Detail of the data dependency view.

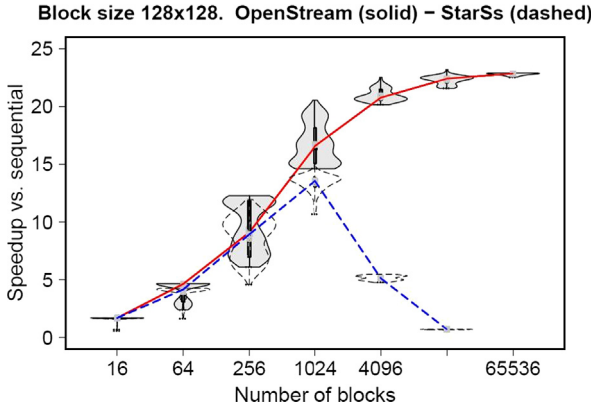

Fig. 11. Speed-up comparison OpenStream and StarSs.

is clearly possible, from the observation of the dependence graph, but the effective distribution mandates decoupling processes executing at different rates. Usual synchronous compilation leads to poor performance with the fast process waiting for the completion of the slower one, as seen in the first figure.

To leverage all the advantages of Heptagon while allowing for parallel code generation, we extend it with futures. At the source code level, futures may be seen as simple annotations leaving the functional semantics of the program unchanged. During the compilation phase, they are key to enable asynchronous calls while preserving memory boundedness. As seen in the second figure, our example can be efficiently compiled to parallel code by adding lightweight, semantics-preserving future annotations.

Operations on arrays are frequent in embedded applications, as example applications studied in the PHARAON project show. It is thus very important when designing a dedicated programming language to offer high-level support together with efficient compilation techniques. In practice, this means reducing the number of

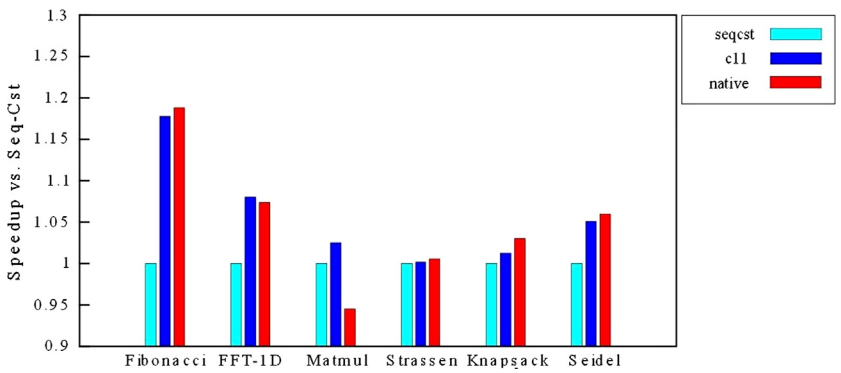

Fig. 12. Speed-up Vs Seq-Cst on various benchmarks.

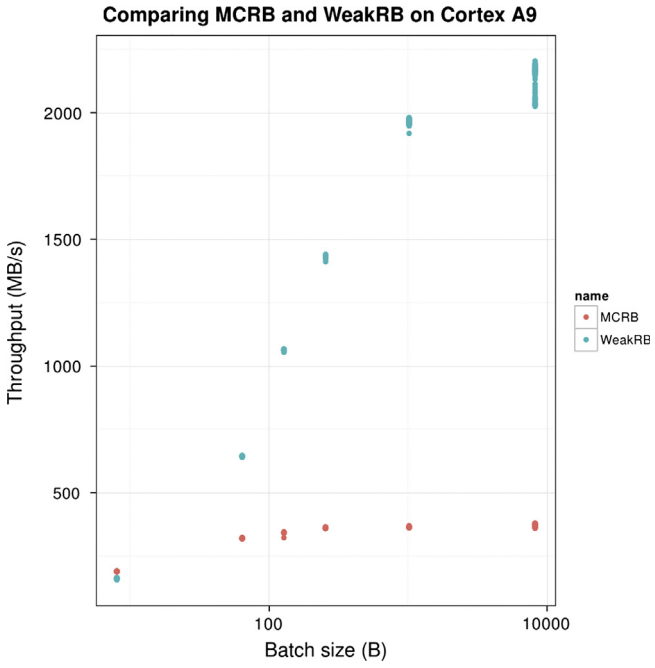

Fig. 13. Comparison between MCRB and WeakRB on Cortex A9.

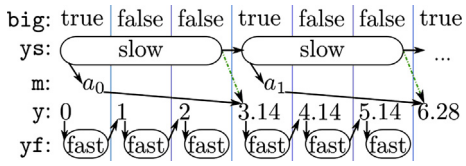

Fig. 14. "slow_fast" Async flow.

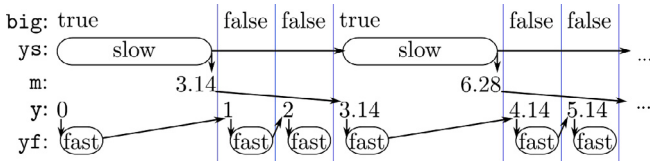

Fig. 15. "slow_fast" Sync flow.

array copies. The Heptagon compiler implements original techniques to this aim, based on a programmer-guided, modular inter-procedural memory allocation procedure [14].

Finally, we are now studying the marriage of the data-flow synchronous paradigm with computational models dedicated to high-performance regular algorithms, such as SDF/CSDF graphs, with the intuition that the result will be more than the sum of its parts. We believe that considering communication rates as a first class citizen of a language semantics is key to next generation tools for embedded platforms, reconciling programmer productivity, efficient and predictable compilers and analyzers, and parallel hardware.

5. Runtime power management tools (RTPM)

After performing design-time optimizations, run-time management gives additional capabilities to improve system operation. Global run-time management methodology used in the context of PHARAON project consists of two phases:

- Phase 1: a full design space is explored for each application at design time to derive set of optimal design points. This phase is out of scope of this paper.
- Phase 2: critical decisions about all active applications are taken at run time. This run-time phase is explored in PHARAON project.

Separating actions in two phases allows minimizing overhead introduced by the runtime decisions through pre-defined optimal configurations which are computed at design time. Then, at runtime, only extra optimization based on dynamic inputs will be done to improve power consumption gains with low performance overhead.

In this approach, the following assumptions on applications are considered:

- Ideally, for any application, all functionalities should be accessible at any time. However, based on the user requirements, the available platform resources, the limited power/energy budget of the platform, and the platform autonomy, it may not be possible to integrate all these functionalities on the platform at the same time. Hence the application developer has to organize the application into *application modes*, each one specifying a different subset of functionalities. For example, a video codec can be implemented in many ways using different number of parallel threads (single-threaded, 2-threaded, 4-threaded and so on). In that case, each implementation (code version) will correspond to a separate video codec mode.
- Whereas the functional specification of an application mode is fixed, there may be several specific algorithms or implementations for a given task (e.g. a Fast Fourier Transform (FFT)).

Additionally, to alleviate the run-time decision making, the RTPM must obtain information from design-time exploration. This exploration is performed per application on a representative set of available input data for all possible application modes, allowed QoS requirements, application parallelizations and data managements. This leads to a multi-dimensional set of Pareto-optimal application configurations, illustrated in Fig. 16. The average costs and the platform resource usage on this representative set are also reported in the multi-dimension set. The Pareto set of each application is an input for the RTPM.

5.1. RTPM approach

During the application run, various opportunities can be exploited by the global run-time manager to optimize application and hardware platform performance. Such run-time decisions during the lifetime of applications are organized into two layers: the coarse grained level L1 includes decisions triggered by dynamic events, while the fine grained level L2 includes decisions to improve application performance. L1 decisions include optimal selection of application configurations and then mapping one or more tasks in those configurations on the platform resources. These decisions are more costly to perform and usually involve reconfiguration of platform hardware. They are triggered by dynamic events generated due to change in the environment e.g. user moves from roaming with LTE network into a WiFI hotspot. On the other hand, L2 decisions correspond to fine-tuning application performance. The control knobs available with the platform (e.g. DVFS) and with the application-specific parameters (e.g. changing the frames per second rate in an MPEG4 encoder to trade-off quality with performance) are tuned iteratively to optimize application and platform performance.

5.2. Decision making at run-time

During the application run, there are various opportunities to optimize application and hardware platform performance. The Global Run-time Manager (GRM) can decide to change platform

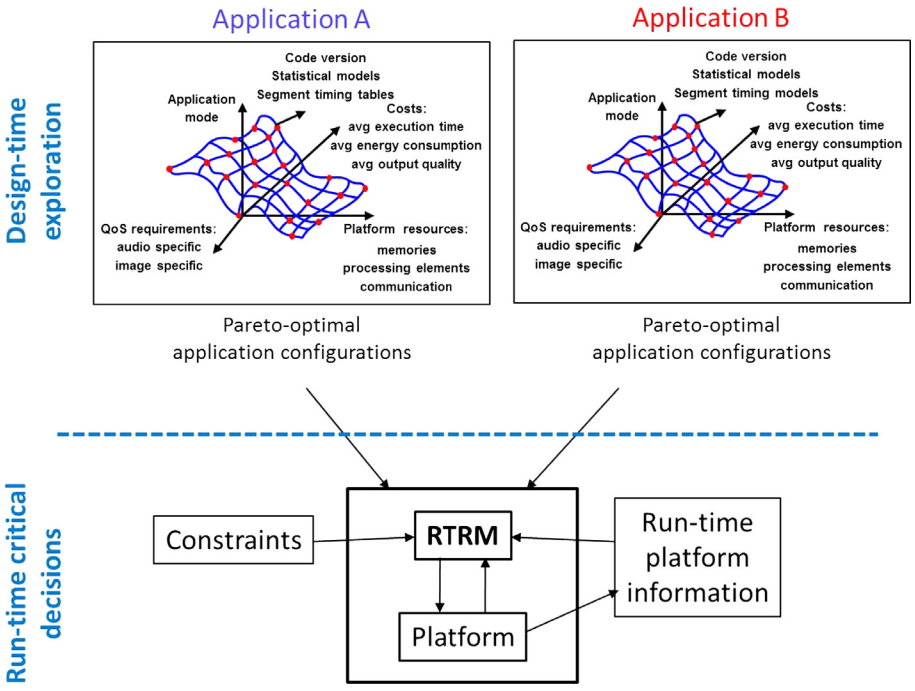

Fig. 16. RTPM global approach.

and application parameters at run-time to exploit these opportunities. The GRM makes these decisions in a systematic way by using coarse-grained and fine-grained decisions.

The high-level algorithm for the runtime decision making, which will typically run on the control processor of the hardware platform, is as follows.

Inputs:

A. Hardware platform information, e.g. available resources (both for computation and for communication), and the different knobs of those resources that can be tuned.
B. Application information, e.g. deadlines (both hard and soft), multiple optimal operating points and their corresponding resource usage, Quality of Service (QoS) constraints.
C. External inputs regarding environment changes e.g. temperature, remaining battery energy.

Algorithm:

1. Decide the allocation of platform resources to applications.
2. Select the optimum operating configuration for each application using the allocated resources.
3. Decide for each application, how the platform resources will be allocated for each task of the application.
4. Perform (partial or full) dynamic reconfiguration of the platform to load application code on those chosen resources.
5. Start executing the application.
6. Monitor observable performance parameters both for application and platform.
7. Perform fine-tuning of platform DVFS modes depending on actual application slack time.
8. In case of environment changes, or dynamic events, or the inability to achieve the expected performance, go to Step 1.
9. Go to Step 6.

In PHARAON, this run-time decision-making flow is enabled by using the RTPM architecture shown in Fig. 17. The low-power scheduler that is described next corresponds to the lower level operations (layer 2).

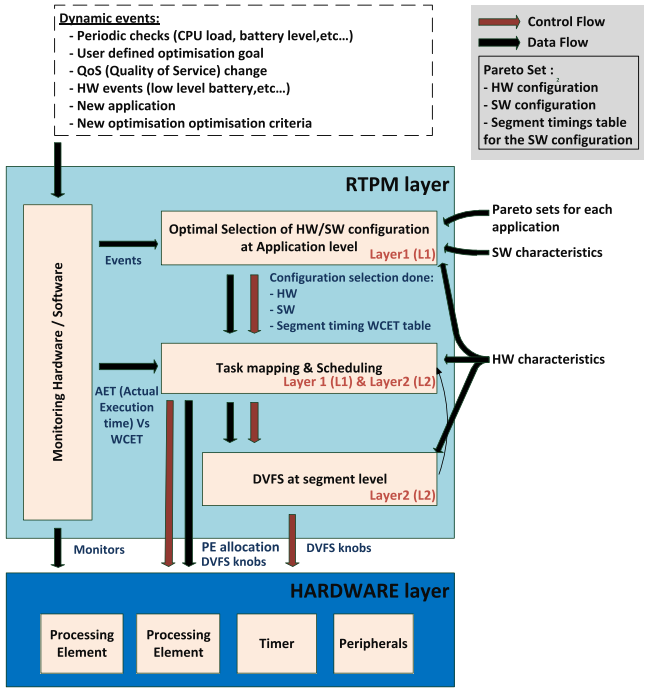

Fig. 17. RTPM architecture.

5.3. Low power scheduler

The low-power scheduler takes as an input the selected application configuration mode, defined by the RTPM. It takes into account a predefined SW configuration (application mode, associated deadlines, tables with the expected timing for various code segments) and HW configuration (number of active cores, voltage/frequency mode, task affinity to specific cores). The low-power scheduler developed in PHARAON actually combines a classical Earliest

Deadline First (EDF) policy with Dynamic Voltage and Frequency Scaling (DVFS) mechanisms.

As depicted in Fig. 17, the power management policy relies on: (1) monitoring the application Actual Execution Time (AET) and (2) comparing it with the Worst Case Execution Time (WCET). To reduce power consumption while still complying with the application deadline, the scheduler tries to minimize the core idle time by: (A) spreading tasks on as many active cores within the SMP as possible, and (B) lowering as much as possible the core voltages and frequencies. The results in terms of energy savings depends on the amount of application idle times and their duration (which can be important if the WCET is very far from the AET) and are thus actually very application dependent. For example, an H264 codec executed on an ARM Cortex A8 shows gains from 20% to 80% depending on the required application QoS (see Fig. 18).

The first task of the application is executed with the core configuration as applied by the dynamic reconfiguration manager. Then modifications of the core state may occur during the task execution only at some specific points, which mark the boundary between "segments".

A segment is a section of code with an associated timing constraint, which enables the scheduler to retrieve information about worst-case and actual execution times at runtime. Thanks to this method, it is possible to monitor and exploit the different execution paths taken by a task at runtime. By feeding the scheduler with this information, the scheduler gets an accurate vision of the portions of job which have already been done and which remain to be done. For that purpose, specific APIs have been defined as well as an extension of POSIX threads, to back-annotate a task with timing information.

From a user perspective, applications that must benefit from this power optimization must be instrumented with segment boundaries through the above mentioned APIs call. The more segments are in task, the better the scheduler will be able to monitor progress of this task, but also the more timing overheads are introduced. A table used at runtime by the low-power scheduler must also be provided by the user. It contains the names of all the segments in the application, together with their associated WCETs at the different core operating points and their deadlines.

Implementation of this scheduler has been done in user space for portability reasons, but implementation in kernel space could also be possible. It requires a FIFO priority-based task scheduler, a hardware platform with DVFS capabilities and an application with POSIX threads. In PHARAON, experimentations will be done on a multicore Cortex-A9 platform with a Linux-SMP OS and the CPUfreq framework. Relying on CPUfreq, which provides a standard HW layer abstraction, allows the approach to be transparently compatible, among others, with future HMP platforms based on big.LITTLE architectures.

In the scope of the PHARAON project, applications must be manually instrumented with the defined APIs. There is some ongoing work on: (1) generating automatically these calls thanks to the code generator previously presented; (2) estimating the application segment WCET via the Pareon tool; (3) integrating the APIs in the OpenStream runtime.

6. Use cases

As previously described, PHARAON targets the development of a set of methods and tools enabling the industrial use of parallel and heterogeneous platforms. Industrial use-cases are thus required to demonstrate both the efficiency of the developed methods and their applicability in an industrial flow.

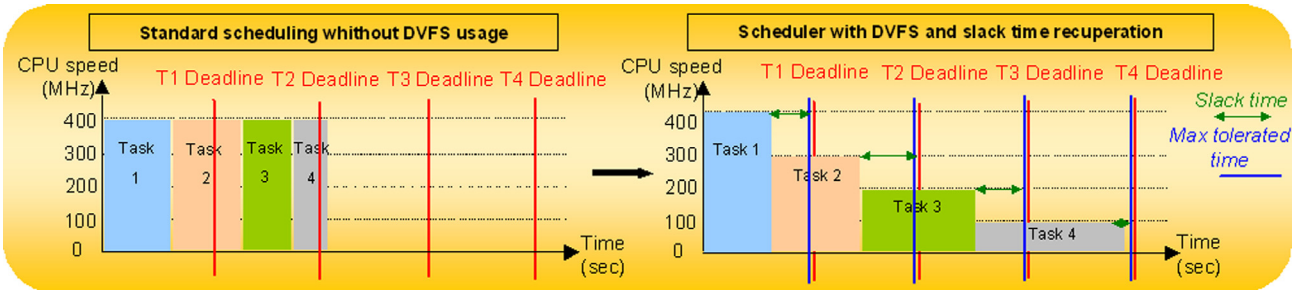

Fig. 18. DVFS scheduling compared to regular scheduling.

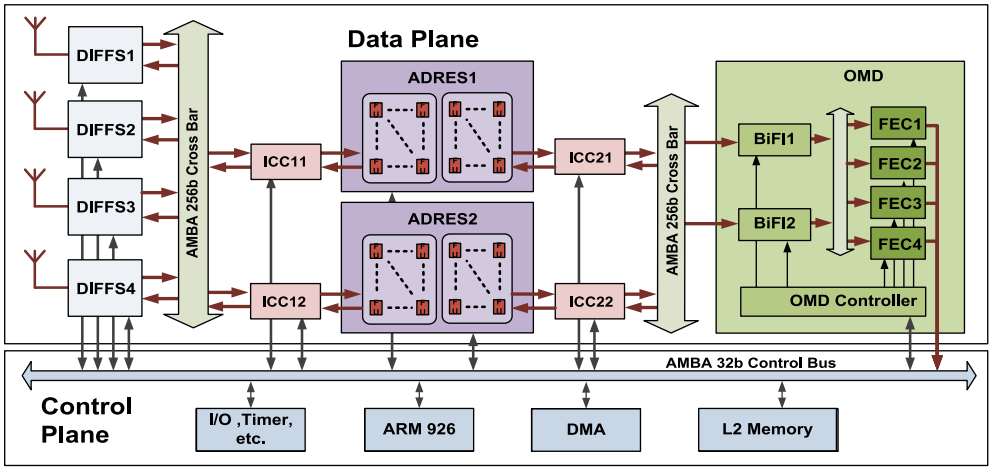

Fig. 19. IMEC COBRA Software Defined Radio platform.

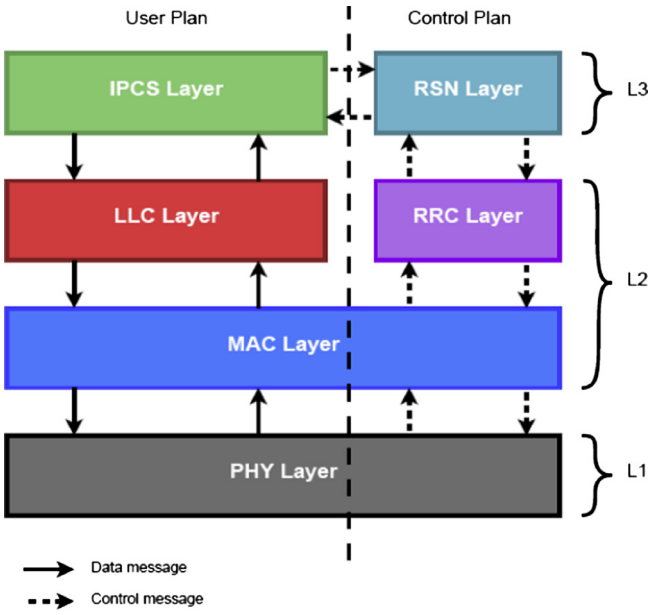

Fig. 20. Radio use case #2.

To ensure a wide coverage of research activities and developed methods, PHARAON targets several application domains: namely advanced 3D video processing and radio communications. On the one hand, most multimedia applications have inherent parallelism and their implementation on a parallel platform is rather direct. On the other hand, radio applications have strong real-time requirements and hard-to-parallelize data dependencies.

Moreover, because of heavy processing loads, implementing real-time 3D video applications with high-definition features becomes more and more difficult on single core architectures. As a consequence, their implementation on a parallel platform is more challenging and the need for automation-assisted parallelization stronger. Furthermore, the two proposed radio applications are complementary. The first one targets the implementation of a physical layer (L1) with real time reconfiguration and multi-stream capabilities, while the second one concentrates on the implementation of the MAC layer (L2 and L3) with a cross-layering approach offering more flexibility. Thus, the three use cases covered (two radio applications and video processing) are described now.

6.1. Radio applications

Two complementary radio applications (use case 1 and 2) are being applied to study the effects of the techniques and tools described in the previous sections. The first one targets the implementation of a physical layer (PHY) with real-time reconfiguration and multi-stream capabilities. The platform architecture for PHY implementation is shown in Fig. 19 [15]. It contains digital front-end (DIFFS) connected to antennas, a baseband processor (ADRES) and an outer modem (OMD) containing Forward Error Correcting (FEC) blocks. The data are exchanged between the blocks in a flexible and programmable way by the 256-bit wide AMBA AHB buses and the interconnect controllers (ICC). All these platform blocks contain in-house developed domain-specific processors which can be programmed by the ARM processor in the Control plane. This programmability of the components makes the platform capable of handling multiple and/or concurrent data streams. The application chosen to show the full capability of this platform switches from receiving a WLAN 802.11n packet to receiving an LTE Cat 4 packet. Our wireless receiver platform can achieve this switch in 52 μs by reconfiguring the firmware of the all platform components.

The second test application (use case 2) concentrates on the implementation of upper protocol layers. It handles IP packets with a TDMA (Time Division Multiple Access) protocol and targets ad-hoc networks. The use of the PHARAON tool suite will help to improve civil protection services (police, fire brigades, medical services), by improvements in the battery life and quality of the radio communications (see Fig. 20).

Although the application behaves at the high level as a dataflow pipeline, it is in fact fully control flow based, with potential cross-layer optimizations, including inter-layer data dependencies. This class of radio protocol applications must be implemented on multicore heterogeneous platforms with new power consumption monitoring capabilities. Both the mapping to a specific platform and the addition of the power management calls to also ensure satisfaction of timing constraints are not straightforward. The application of the PHARAON design flow presented in Section 3 helps the system engineers to perform a proper analysis to find the best mapping to a given platform.

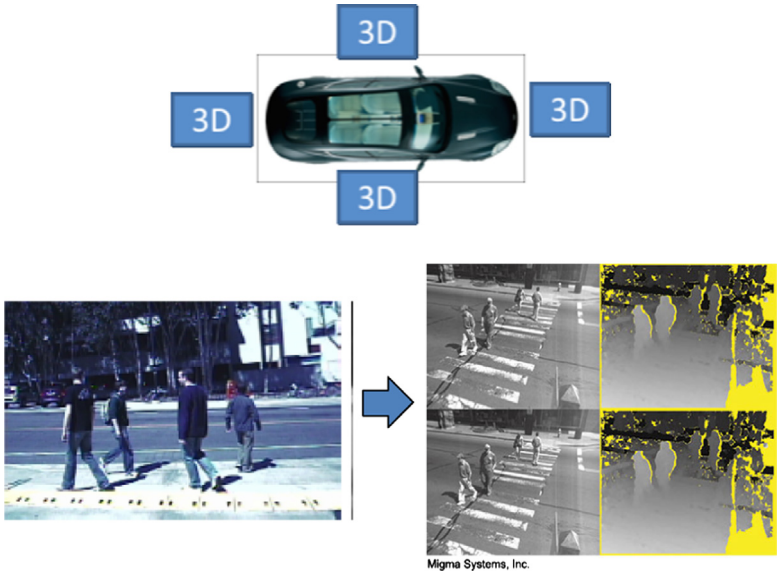

Fig. 21. Stereovision use case.

Table 1
Overview of PHARAON tools used for industrial use cases.

	Radio applications	Stereovision application
UML specification and code generation (UC)	X	X
PAREON Tool (VF)	X (partially)	X
ParTool (POLITO)	X	X
OpenMP extension (ENS)	X	X
Heptagon language (ENS)	X (partially)	
Low-power scheduler (TCS)	X	X (partially)
Run-time manager (IMEC)	X	

6.2. Stereovision application

Another demonstrator (use case 3) relies on advanced 3D stereoscopic applications with real-time and high definition constraints, targeting the automotive domain for human and obstacle detection (see Fig. 21).

The application infers the 3D scene geometry from the images provided by two twin cameras placed in a known configuration. Several steps are required in order to compensate the distortion introduced by the physical characteristics of the sensors, to align the images and find the depth map with enough accuracy to be used in safety critical environments. The application of the design flow described in Section 3 will allow finding the optimal solution to be implemented on a multicore platform. This use case will demonstrate the impact of our system design flow on critical aspects of embedded systems design.

6.3. Design flow application to use cases

Table 1 gives an overview on the tools in the PHARAON design flow being applied on the three use cases (the two radio applications and the stereovision application). Although the project is still on-going, various tests have been performed using the tools presented above. Their results are described next.

Considering use case 1, the application code for the wireless communication modes has been parallelized and optimized for the Software Defined Radio platform and this use case is being primarily used with the global run-time manager to validate the potential for energy savings.

The integration of the application code with the run-time manager showed that the reconfiguration from an 802.11n scenario to a 3GPP-LTE scenario can be successfully executed by the run-time manager.

Considering use case 2, the following steps have been successfully executed or are currently in progress:

- UML modeling with the structural and deployment view.
- Performance analysis of the first version of the application provided.
- Preliminary steps of low-power scheduling.
- Preliminary steps of parallelization opportunity identification, namely performance analysis and data/control dependency analysis.

The use of GP–GPUs for performance optimization has also been explored for the physical layer application (see Fig. 22).

Finally, regarding the stereo vision use case 3, some initial manual parallelization has also been completed. It is used as a baseline reference of effort and performance to evaluate the tools provided by the partners.

Fig. 23 shows that on a 2 core platform the performance gain of the initial manual parallelization is slightly above 1.7X. Note that since the target platform will be a 4 core machine, this should be considered just a preliminary estimate of the manual parallelization quality.

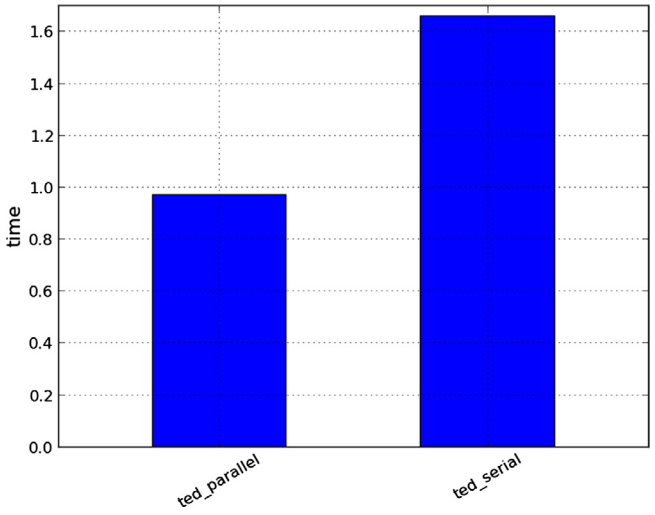

Fig. 23. Performance comparison of sequential and manually parallelized versions of the stereovision use case on a 2 core machine.

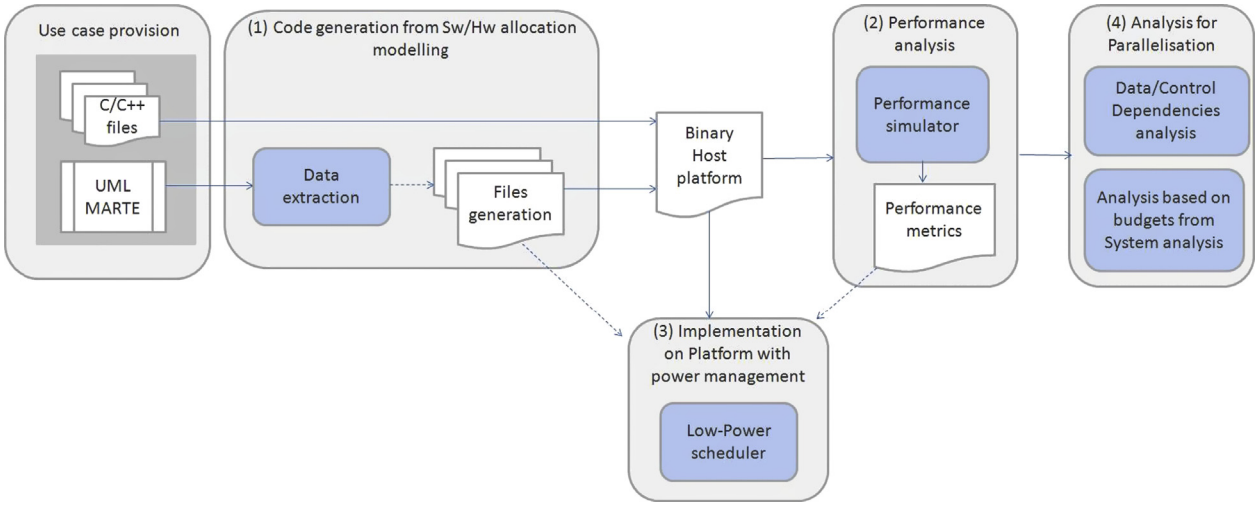

Fig. 22. Current status of experiments done on THALES use cases.

Table 2
Execution times of the stereovision use case.

HW platform	Original code (sec)		Optimized code (sec)	
I/O type	Test-bench	Camera	Test-bench	Camera
Intel platform	16.3	–	25.2	–
Beagle: no Neon compilation flags	305.2	313.9	199.7	202.9
Beagle-Panda: no Neon comp. flags	278.7	286.8	164.8	165.8
Beagle: Neon compilation flags	84.7	85.2	68.8	74.8
Beagle-Panda: Neon comp. flags	23.5	26.6	31.3	32.40
SPEAr-600	262.6	265.8	–	–

Table 3
Processing times of the stereovision application for different parallelization solutions.

Optimisations	Time	Speedup
Sequential code (no OMP & no SIMD)	3.4 s	Baseline
OpenMP parallelization but no SIMD	1.9 s	1.8×
SIMD vectorization but no OpenMP	1.15 s	3×
Both OpenMP parallelization and SIMD vectorization	0.8 s	4.×

Two platforms, both the initially proposed OMAP4 platform (Panda board, with dual-core ARM and GPU) and an ASUS P8H77-I powered by an Intel I5-3570T, have been tried for use case 3. These two possibilities were motivated by the preliminary performance analysis of the application and the recent availability of quad-core architectures. On the one hand, the OMAP4 enables the verification of various mappings of components to DSP cores, or the use of NEON co-processors, in order to check heterogeneity support. On the other hand, the Intel platform enables increasing the computational power and physical parallelism to better check the effectiveness of the parallelization tools.

The UML/MARTE modeling and code generation infrastructure has been applied to the stereovision use case, enabling the exploration of different platforms, and supporting the generation of optimized code for each platform. As a result, not only the proposed platforms, but also other platforms, as Beagle and SPEAr boards have been evaluated, which demonstrates the flexibility of the approach. Some results are shown in the next table (see Table 2).

At the same time, the ParTools suite was used to discover the best parallelization candidates and to analyze their data dependencies for the stereo vision use case. Based on these results, two parallelization strategies were decided for a target architecture made of 2 i5 cores running at 1.20 GHz (i5-3230 M):

1. Four data-parallel hotspots, using OpenMP.
2. Several SIMD conversions using the target architecture support for vector operations.

The processing times of the test images of 1024×768 pixels can be found in Table 3. As can be shown, the combined optimizations achieved a maximum speed-up of 4 times.

7. Conclusions

In this work various tools, techniques and runtime solutions developed to help mapping applications to multiprocessor platforms have been presented. These tools implement a complete flow, from UML modeling to final implementation. The goals of the flow are to reduce development time, increase performance and reduce power consumption.

To facilitate the usage of this methodology, a model-driven design methodology starting from UML is used to generate automatically the various data files used by the design-time tools and the runtime environment. The use of a component-based approach enables to separately handle different parts of the system, enabling different mappings to homogeneous, heterogeneous and distributed systems.

This approach also enables a two level design flow, combining a coarse grain and a fine grain level. The component-based approach allows the user to select different deployment strategies and to explore parallelization between components though automatic code generation. Then, the internals of each component can be analyzed and optimized using the parallelization tools.

Our code generation, parallelization analysis and implementation tools have been developed in combination with fast performance estimations to improve design-time results and speed up the design cycle.

Finally, different runtime managers (a reconfiguration manager and a low-power scheduler) are deployed on the target physical platform in order to reduce power consumption while ensuring the fulfillment of real-time constraints. Furthermore, timing and power traces collected by the performance simulator help to refine the power management strategy.

Till now, several results have been obtained separately for the different tools. The rest of the project will be devoted to further integration tests and to measure more globally the quality of the achieved results.

Acknowledgments

This work is being performed in the framework of the FP7-288307 project PHARAON.

References

[1] Steve Balacco, Next generation embedded hardware architectures: driving onset of project delays, cost overruns, and software development challenges, Klockwork Inc.
[2] Bart Kienhuis, Compaan: Deriving process networks from Matlab for embedded signal processing architectures, in: Proceedings of the 8th International Workshop on Hardware/Software Codesign, 2000.
[3] Vinod Kathail, Shail Aditya, Rob Schreiber, B. Ramakrisha (Bob) Rau, Darren Cronquist, Mukund Sivaraman, PICO (Program In, Chip Out): automatically designing custom computers, IEEE Computer 35 (9) (2002) 39–47.
[4] Timo Stripf et al., Compiling Scilab to high performance embedded multicore systems, Microprocessors Microsyst. (2013).
[5] F. Thoma, et al., MORPHEUS: Heterogeneous Reconfigurable Computing, in: Field Programmable Logic and Applications (FPL). International Conference on, aug. 2007, pp. 409–414.
[6] T. Ahonen et al., CRISP: cutting edge reconfigurable ICs for stream processing, in: J.M.P. Cardoso, M. Hübner (Eds.), Reconfigurable Computing:From FPGAs to Hardware/Software Codesign, Springer Verlag, London, 2011, pp. 211–238.
[7] A. Prasad, J. Anantpur, R. Govindarajan, Automatic compilation of matlab programs for synergistic execution on heterogeneous processors, SIGPLAN Not. 46 (6) (2011) 152–163.
[8] L. Józwiak et al., ASAM: Automatic Architecture Synthesis and Application Mapping, in: 15th Euromicro Conference on Digital System Design (DSD), Cesme, Izmir, Turkey, September 2012, pp. 216–225.
[9] Marco Solinas, Rosa M. Badia, Francois Bodin, Albert Cohen, The TERAFLUX Project: Exploiting the DataFlow Paradigm in Next Generation Teradevices, Euromicro Conference on Digital System Design (DSD), 2013.
[10] Pareon documentation, <http://www.vectorfabrics.com/docs/pareon/current/>.
[11] Nhat Minh Lê, Antoniu Pop, Albert Cohen, Francesco Zappa Nardelli, Correct and efficient work-stealing for weak memory models, in: Symp. on Principles and Practice of Parallel Programming(PPoPP), Shenzhen, China, February 2013.

[12] Antoniu Pop, Albert Cohen, OpenStream: Expressiveness and data-flow compilation of OpenMP-streaming programs, ACM Transactions on Architecture and Code Optimization (TACO), January 2013. HiPEAC 2013 Conf.
[13] Albert Cohen, Léonard Gérard, Marc Pouzet, Programming parallelism with futures in Lustre, in: ACM Conf. on Embedded Software (EMSOFT), Tampere, Finland, October 2012. Best paper award.
[14] L. Gérard, A. Guatto, C. Pasteur et M. Pouzet, Modular Memory Optimization for Synchronous Data-flow Languages, in: Languages, Compilers and Tools for Embedded Systems (LCTES), Taipei, Taiwan, June 2012, Best paper award.
[15] J. Declerck, P. Avasare, M. Glassee, A. Amin, E. Umans, A. Dewilde, P. Raghavan, M. Palkovic, A flexible platform architecture for Gbps Wireless Communication, International Symposium on System-on-Chip (SoC), Tampere, Finland, 2012.
[16] M.T. Lazarescu, L. Lavagno, Dynamic trace-based data dependency analysis for parallelization of C programs, in: Proceedings of 12th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM), Riva del Garda, Italy, September 2012.
[17] L. Lavagno, M.T. Lazarescu, I. Papaefstathiou, A. Brokalakis, J. Walters, B. Kienhuis, F. Schäfer, HEAP: a highly efficient adaptive multi-processor framework, in: Microprocessors and Microsystems: Embedded Hardware Design (MICPRO), 2013.
[18] P.P.C. Lee, T. Bu, G. Chandranmenon, A lock-free, cache-efficient shared ring buffer for multi-core architectures, in: Proc. of the 5th Symp. on Architectures for Networking and Communications Systems, New York, NY, 2009.
[19] N.M. Lê, A. Guatto, A. Cohen, A. Pop, Correct and Efficient Bounded FIFO Queues, Research Report RR-8365, INRIA, Sept. 2013.
[20] W. Mueller, Y. Vanderperren, UML and model driven development for SoC design, CODES+ISSS'06.
[21] OMG: "UML Profile for MARTE", <www.omgmarte.org>, 2013.
[22] J. Barba, F. Rincón, F. Moya, J.D. Dondo, J.C. López, A comprehensive integration infrastructure for embedded system design, Microprocessors Microsyst. (2012).
[23] S. Kang, H. Kim, J. Baik, H. Choi, C. Keum, Transformation Rules for Synthesis of UML Activity Diagram from Scenario-Based Specification, 34th Annual Computer Software and Applications Conference (COMPSAC), 2010.
[24] E. Karl, D. Blaauw, D. Sylvester, T. Mudge, Reliability Modeling and Management in Dynamic Microprocessor-Based System, Proc. Design and Automation Conf. (DAC), pp. 1057–1060, 2006.
[25] H.R. Pourshaghaghi, J.P. de Gyvez, Fuzzy-controlled voltage scaling based on supply current tracking, IEEE Trans. Comput. 62 (12) (2013) 2397–2410.
[26] MULTICUBE FP7-216693 project, Multi-objective Design Space Exploration of Multiprocessor SOC Architectures for Embedded Multimedia Applications, <http://www.multicube.eu>, January 2008 – June 2010.
[27] 2PARMA FP7-248716 project, PARallel PAradigms and Run-time MAnagement techniques for Many-core Architectures, <http://www.2parma.eu>, January 2010 – March 2012.
[28] Kim Grüttner et al., The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration, Microprocessors Microsyst. (2013).
[29] GEODES ITEA2-07013 project, Global Energy Optimization for Distributed Embedded Systems, <http://geodes.ict.tuwien.ac.at>, September 2008 – September 2011.
[30] PHERMA ANR project, Parallel Heterogeneous Energy efficient Real-time Multiprocessor Architecture, <http://pherma.irccyn.ec-nantes.fr/>, September 2007 – September 2010.
[31] SCALOPES ARTEMIS project, SCAlable Low Power Platforms, <http://www.scalopes.eu/>, January 2009 – January 2011.
[32] L. Benini, R. Bogliolo, G. De Micheli, A survey of design techniques for system-level dynamic power management, IEEE Trans. VLSI Syst. 8 (2000) 299–316.
[33] J.J. Chen, C.Y. Yang, T.W. Kuo, C.S. Shih, Energy-Efficient Real-Time Task Scheduling in Multiprocessor DVS Systems, IEEE ASP DAC, Yokohama, Japan, January 2007, pp. 342–349.
[34] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, M. Martonosi, An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget, 39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006, pp. 347–358.
[35] J. Luo, N.K. Jha, Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems, IEEE ASP-DAC (2002) 719.
[36] V.K. Prasanna, Power-aware resource allocation for independent tasks in heterogeneous real-time systems, IEEE ICPADS (2002) 341.
[37] T.P. Baker, An analysis of EDF schedulability on a multiprocessor, IEEE Trans. Parallel Distributed Syst. (2005).
[38] A. Buchard, Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems, Technical Report, University of Virginia, Charlottesville, VA, USA, 1994.
[39] H.L. Chan, Non-Migratory Online Deadline Scheduling on Multiprocessors, SODA, 2004, pp. 970–979.
[40] S. Lauzac, Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor, EUROMICRO, 1998, pp. 188–195.
[41] W. Thies, V. Chandrasekhar, S. Amarasinghe, A practical approach to exploiting coarse-grained pipeline parallelism in C programs, in: 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2007.
[42] D. Shin, J. Kim, Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems, ACM International Symposium on Low Power Electronics and Design, Augustus 2003, pp. 408–413.
[43] M. Schmitz, Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems, IEEE DATE, 2002.
[44] A. Sinha, A. Chandrakasan, Jouletrack – a Web Based Tool for Software Energy Profiling, IEEE DAC, 2001.
[45] Y. Zhang, X. Hu, D. Chen, Task Scheduling and Voltage Selection for Energy Minimization, IEEE DAC, 2002, pp. 183–188.
[46] K. Ramamritham, G. Fohler, J.M. Adan, Issues in the static allocation and scheduling of complex periodic tasks, IEEE Real-Time Systems Newsletter 9 (1993) 11–16.
[47] Ch. Ykman-Couvreur, V. Nollet, Fr. Catthoor, H. Corporaal, Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management, aCM Transactions on Embedded Computing Systems, February 2011.
[48] A. Pop, A. Cohen, A stream-computing extension to OpenMP, in: Intl. Conf. on High Performance Embedded Architectures and Compilers (HiPEAC'11), January 2011.
[49] C. Miranda, P. Dumont, A. Cohen, M. Duranton, A. Pop, Erbium: A deterministic, concurrent intermediate representation to map data-flow tasks to scalable, persistent streaming processes, in: Intl. Conf. on Compilers Architectures and Synthesis for Embedded Systems (CASES'10), October 2010.
[50] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM 20 (1) (1973) 46–61.
[51] H. Posadas, P. Peñil, A. Nicolás, E. Villar, Automatic synthesis from UML/MARTE models using channel semantics, ACM/IEEE 15th International Conference on Model Driven Engineering Languages & Systems, MODELS, 2012.
[52] H. Posadas, P. Peñil, A. Nicolás, E. Villar, UML/MARTE methodology for high-level system estimation and optimal synthesis, International Workshop on Metamodelling and Code Generation for Embedded Systems, MeCoES 2012.
[53] H. Posadas, P. Peñil, A. Nicolás, E. Villar, System Synthesis from UML/MARTE Models: The PHARAON approach, in: The Conference on Electronic System Level Synthesis, ESLSyn 2013.
[54] H. Posadas, P. Peñil, A. Nicolás, E. Villar, Code Synthesis Of Uml/Marte Models For Physical Platforms Considering Resource-Specific Optimized Codes, in: The Embedded Computing Conference, JCE 2013.
[55] H. Posadas, P. Peñil, A. Nicolás, E. Villar, Automatic Concurrency Generation Through Communication Data Splitting Based On Uml/Marte Models", Conference on Design of Circuits and Integrated Systems, DCIS 2013.
[56] H. Posadas, P. Peñil, A. Nicolás, E. Villar, Automatic synthesis of embedded SW Communications from UML/MARTE models supporting memory space separation, Conference on Design of Circuits and Integrated Systems, DCIS, 2012.