EU FP7-288307 Pharaon Project: Parallel and Heterogeneous Architecture for Real-Time Applications

EU FP7-288307 PHARAON project

Parallel and heterogeneous architecture for real-time applications

Hector Posadas, Eugenio Villar
University of Cantabria Santander
Santander, Spain
{posadash,villar}@teisa.unican.es

Florian Broekaert, Michel Bourdelles
Thales Communications & Security SA
Gennevilliers, France
florian.broekaert thalesgroup.com

Albert Cohen, Antoniu Pop, Nhat Minh Lê, Adrien Guatto
INRIA and École Normale Supérieure
Paris, France

Mihai T. Lazarescu, Luciano Lavagno
Politecnico di Torino
Torino, Italy

Andrei Terechko
Vector Fabrics
Eindhoven, The Netherlands

Miguel Glassee
IMEC
Leuven, Belgium

Daniel Calvo, Edouardo de las Heras
Tedesys
Santander, Spain

***Abstract*—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 assist the designer in the development of parallel embedded systems, from executable specifications to target-specific implementation and debugging on a multicore platform. This tool chain will offer and implement several parallelization strategies, reflecting the functional and non-functional constraints of the system, and driving the designer into incremental parallelization and adaptation 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 application requirements and therefore reduce power consumption transparently.**

Keywords-component; UML; parallelisation; OpenMP; low-power; ressource management; profiling; multiprocessor

I. INTRODUCTION

The PHARAON project is a European collaborative initiative between universities, research labs and industrial companies. It is sponsored by the European Commission, which supports part of the costs and assists partners in the project management. The first section provides a general presentation of the project. Section 2 describes the use cases driving the project. Section 3 surveys our proposed design flow. Detailing this flow, the fourth section presents the High Level System component approach (UML/MARTE) as well as its associated code generator, and the fifth section describes the parallelisation and performance analysis tools. Section 6 presents the runtime power management methodology and tools. Finally the conclusion highlights the project perspectives.

A. Context and objectives

The latest and greatest embedded systems integrate a wide range of very complex functionalities. A smart phone, for example, is capable to communicate through 3G and WIFI connections. It integrates phone services with high performance graphics and sophisticated software applications such as real-time video and audio. To support this application load, new devices make use of recent parallel architectures capable to deliver enough processing power. But these changes have two negative effects. 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 quality of service requires more energy and hence is associated with a reduction of autonomy. The newest smart phones can run for at most about three days in standby mode while former simple phones did not need a recharge for an entire week. The PHARAON project will develop new techniques and tools that will offer the possibility to reduce the software development cost (25% targeted in the project) and increase the autonomy of embedded systems by nearly 20%.

B. Consortium

PHARAON is coordinated by a large company Thales Comunications & Security. Tedesys and Vector Fabrics are two SMEs completing the industrial partners. Academic partners are made of Politecnico di Torino, Ecole Normale Supérieure and University of Cantabria. Finally, the research institute Interuniversitair Micro-Electronica Centrum is completing the consortium.

C. Technical approach

The PHARAON project targets the development of two different sets of techniques and tools, both aiming at best exploiting the low-power capabilities of modern multi-core processors, tackling both the programming and runtime power management challenges mentioned previously. The first set will directly impact the design of the application.

The objective is to assist the designer in finding the most adequate software architecture taking into account hardware constraints.

To do so, tools will be capable to evaluate the parallel structure of an application and propose improvements. A tool will also be capable to handle communications between different processors and generate the multi-processor embedded code. The second set of techniques and tools will impact the runtime behavior of the application. The objective is to adapt the performance of the platform, (frequency & voltage for example) in order to consume only the required energy. A reconfiguration system and a low power scheduler will be integrated with other run-time components on top of the platform.

II. USE CASES

The efficiency of the techniques and tools developed in this project is showcased on three example applications from two different domains: software defined radio and image processing.

A. *Radio applications*

Two complementary radio applications are studied. The first application consists of the implementation of a physical layer (PHY) with real-time reconfiguration and multi-stream capabilities. The platform architecture for the implementation of PHY is shown in Fig.1 0. 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 is exchanged between blocks in a flexible and programmable way through the 256-bit wide AMBA AHB buses and the interconnect controllers (ICC). All these platform blocks contain proprietary domain-specific processors which can be programmed by the ARM processor in the Control plane. Using this versatile feature, the platform is capable of handling multiple and/or concurrent data streams. For example, our wireless receiver platform can switch from receiving a WLAN 11n packet to receiving a LTE Cat 4 packet in 52µs by reconfiguring all of the components' firmwares.

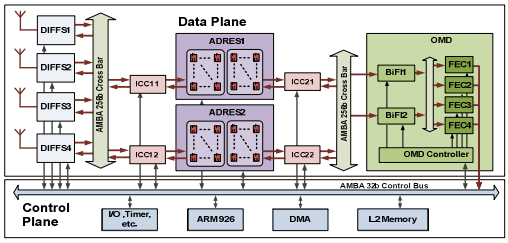

Figure 1. IMEC COBRA Software Defined Radio platform

The second application (Fig.2) concentrates on the implementation of the upper protocol layers. It handles IP packets, relies on TDMA (Time Divion Multiple Access) and targets an ad hoc network. The use of the PHARAON workbench will help to improve civil protection services (police, fire brigades, medical services), by improving the autonomy and quality of radio communications.

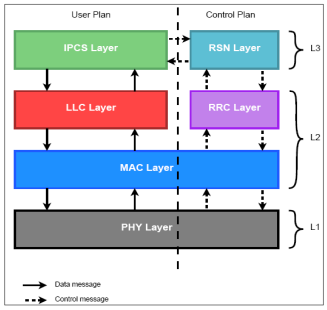

Figure 2. Radio use case #2

Despite the dataflow pipeline appearance of this application at high-level, it is essentially control-flow dependent, with potential cross-layer optimizations and data dependences. Our objective is to provide implementations of these applications for heterogeneous multicore platforms and optimize power consumption based on monitoring data. The key problem lies in optimizing the distribution of applications across processing units while respecting the timing and consumption constraints defined in the application's specification. The design flow developed in the PHARAON project, and presented in section 3, will help the system engineers to analyze the applications and to find an optimal integration for a targeted platform.

B. *Maintaining the Integrity of the Specifications*

Our third example is an advanced 3D stereoscopic application (Fig.3), with real-time and high definition constraints, used in automotive domain for humans/obstacle detection.

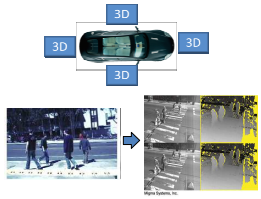

Figure 3. Stereovision use case

The application infers the 3D scene geometry from the images provided by two twin cameras under 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 architectural solution to be implemented on a multicore platform. This use case should demonstrate the impact of our system design flow in critical aspects of embedded systems design.

III. PHARAON SYSTEM DESIGN FLOW

The targeted design flow in PHARAON drives the design from UML specifications to implementation of cross-

compiled code onto the platform. As depicted in Fig.4, the proposed flow starts designing the application with a high level description component based approach (UML/MARTE) with properties structured via XML files. During this step, different deployment strategies can be tested and a coarse grained parallelisation (between components) can be done.

For each component, the associated business code (C/C++ files) has to be developed accordingly the targeted hardware resource. Then, a code generator is developed to automatically generate the wrapper codes required to allocate and run the SW component in the HW platform and interconnect the business code. It also produces the executable files that are used as inputs in the four different stages of the toolchain.

In a first stage, the C code to be parallelized is extracted from the UML model and sent to a performance simulator. This simulator then generates a timing and power analysis of the bloc to be used in the next stage.

In the second stage, the parallelisation tool optimizes the internal code of each component. Basing its analysis on the C code entry and performance information, the parallelization tool generates a parallel code integrating OpenMP/Openstream directives.

In a third stage, the optimized code is simulated on the performance evaluation tool to obtain the information required for run-time optimization. Alternatively, the code can be implemented and measured onto the physical platform.

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

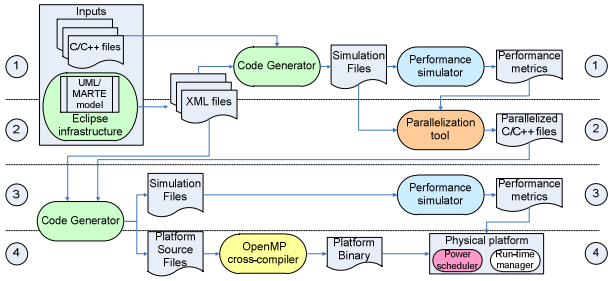

Figure 4. PHARAON design flow

IV. HIGH LEVEL SYSTEM METHODOLOGY

A. *UML/MARTE modeling*

In order to support all the different stages of the flow, a powerful high-level modeling methodology has been defined. Based on UML, it follows a component-based approach, applying the Model-Driven Architecture (MDA)0 principles to the development of HW/SW embedded systems. Additionally, we chose the standard MARTE profile to handle all the specific characteristics related to embedded systems. Moreover, the proposed methodology is software-centric since it considers application-specific HW only when speed or power constraints coming from the designer-supplied mapping of software components to application processors are not met.

Following the proposed methodology, designers can completely describe the system, enabling automatic generation of input data and code required by the different tools of the design flow. For such purpose, designers must describe in different views the system functionality (Fig.5), the target platform and the resource allocation.

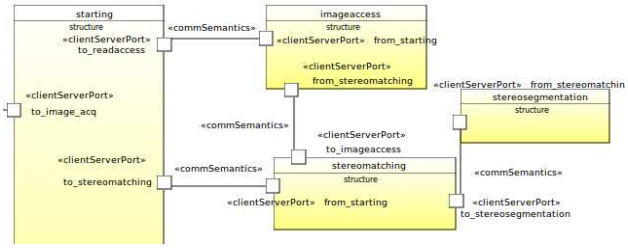

Figure 5. Excerpt of the stereovision application structure

However, since the methodology has to support quite different tools, we have to extend the previous UML/MARTE approaches in several ways. The main issues the flow has to cover are heterogeneity, parallelization, support for different I/O schemes and run-time management. Thus, specific improvements are being proposed for all of these points.

In order to support adequate mappings 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 files for the same component, each one optimized for each possible mapped resource, including files for host simulation.

In order to solve the first two points, the system mapping (Fig.6) is performed in two steps: first components are mapped to memory spaces and then these memory spaces are mapped to resources. As a result, different executables are generated for the system, one for each memory space. More details can be found in 0. Additionally, to support different files for the same component, different attributes have been added to the “file” stereotype.

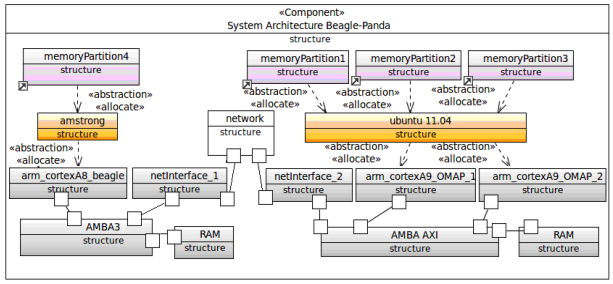

Figure 6. Platform mapping example

Moreover, the information described in the UML model also enables the automatic generation of ad-hoc communications infrastructures. To optimize it, different channel semantics has been added to MARTE profile 0. This allows designer to optimize the system concurrent architecture in a coarse grain, by modifying the relationships between the system components. Finally, different efforts have been started to enable automatic I/O integration and run-time power management support.

B. *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 initializing all the components mapped on each memory space, the C wrappers that enable the communication among the application components, the different agents handling incoming requests and the compilation scripts.

The interface wrappers use the facilities provided by a communication library developed to implement the final 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 priorized accesses from different clients are implemented in this step. Then, argument 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.

This infrastructure has been applied to the stereovision use case, enabling the exploration of different platforms, and supporting the generation of optimized code for this platforms. Some results can be seen in the next table.

TABLE I. EXECUTION TIMES OF THE STEREOVISION USE CASE

HW platform	**Original code (sec)**		**Optimized code (sec)**	
I/O type	Test-bench	Camera	Test-bench	Camera
PC simulation	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	-	-

V. PARALLELISATION TOOLCHAIN AND PERFORMANCE ANALYSIS

A. *PAREON performance 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 estimates are fed into the parallelization tool to parallelize performance and memory bottlenecks of the program, 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 modelled target hardware is an ARM Cortex A9 and Intel Core 5 multicore processors.

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 PHARAON toolchain, while the GUI enables human inspection of the modelling 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. The vfrcc 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 in the provided execution environment, including necessary test data, input files, environment variables, etc. During the execution various statistics such as instruction counts and memory behaviour 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.

The internal Pareon toolflow for performance analysis is shown below in Fig.7 and an extensive documentation of the Pareon functionality is available online at 0.

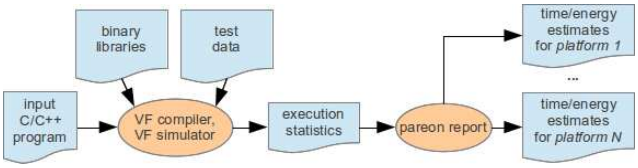

Figure 7. Pareon performance analysis toolflow

In contrast to gate-level back-annotated timing and power modelling tools in the EDA industry, the performance analysis in Pareon is much faster. It delivers the throughput of only few hundred times slower than real-time execution compared to EDA tools, which are orders of magnitude slower. Furthermore, in contrast to state-of-the-art profile-based optimizing compilers, the Pareon can model non-existing chip configurations, for example, with more processor cores than what's available on the market today.

Another important feature of the tools for design space explorations is that the modelling of different platforms or

operating conditions can be done with a single simulation step. Based on this simulation step the necessary statistics are gathered of the program execution and after that the actual estimates are quickly computed using properties of the target hardware.

B. Parallelisation toolset

The toolset supports the parallelization of sequential C software that can include significant control decisions, and pointer and dynamic memory operations. It can help discover both task and data parallelization opportunities, for any parallelization technique.

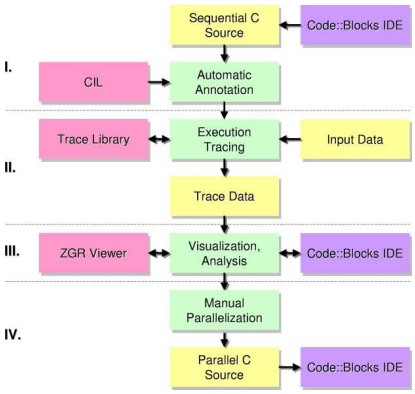

Figure 8. Parallelization toolset flow

The toolset flow shown in Fig. 8 is divided in four stages: (I) source instrumentation, (II) run-time collection and compaction of execution data, (III) execution data graphical visualization and analysis, and (IV) source code parallelization. It is controlled from the Code::Blocks C/C++ IDE that supports also cross-referencing the execution trace data visualized in stage III with the sequential C project source.

The sequential source annotation in stage I supports run-time data dependency collection by means of a special library. Then these data are saved and used by the graphical visualization and analysis interface.

This interface compactly displays in a graph format both the execution profile and the data dependencies to facilitate the search for parallelization opportunities. The nodes are data-processing program elements like statements, loops, function calls, while the edges are the data dependencies between them. The call stacks are also represented, and the nodes can fold an entire execution rooted on them, like for loops and function calls.

Fig. 9 shows an analysis view for the Stereo Matching application presented in Section 2.2. The rectangular nodes are folds of loops, the elliptic are function call folds.

Two of the loop folds, with stronger colourization, include 53% and 18% of program execution. This makes them significant candidates for parallelization especially since they have no strong data dependency between them

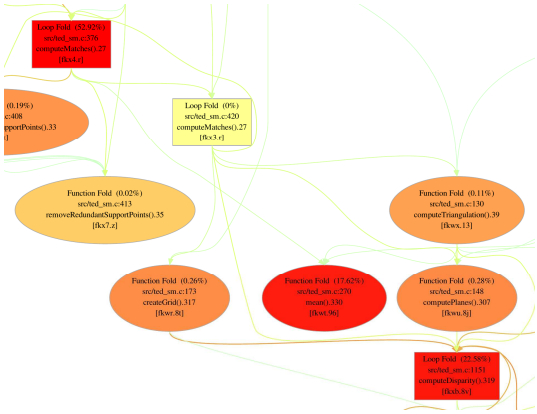

Figure 9. Analysis of the Stereo Matching application

C. 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 0. It allows to express 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 our 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. Key to the efficient execution of OpenStream programs is our optimized runtime system, providing low-overhead synchronization and work-stealing scheduling.

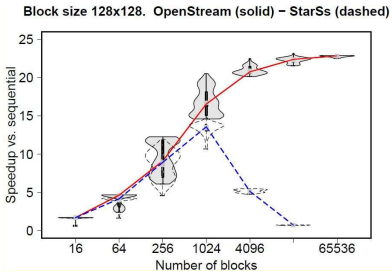

Figure 10. Speed-up comparison OpenStream and StarSs

Work stealing is a central component of the OpenStream run-time library, allowing for efficient lock-free scheduling

of lightweight tasks. The dynamic scheduler has been ported to the x86 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 0.

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.11). 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.

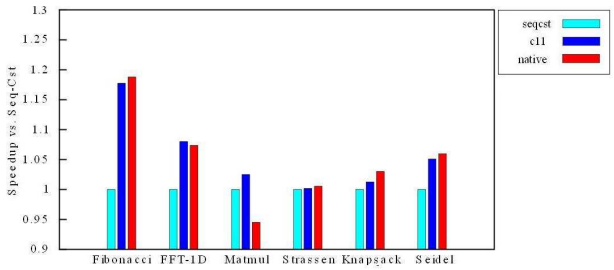

Figure 11. Speed-up Vs Seq-Cst on various benchmarks

OpenStream originated from the TERAFLUX FP7 project. In PHARAON, it is being ported to embedded ARM multicore platforms and extended to support

D. 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 compilaton 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 desynchronization and on the distribution of the program (or model)0.

Classical issues are summed up in the classical slow_fast example sketched in Fig.12 and Fig.13. A slow process communicates with a faster one at the rate of the slow process. Parallel execution 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 on 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.

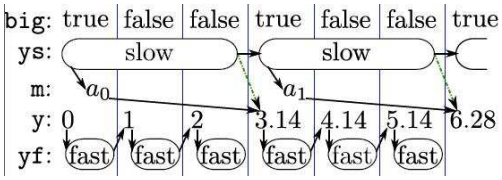

Figure 12. slow_fast async flow

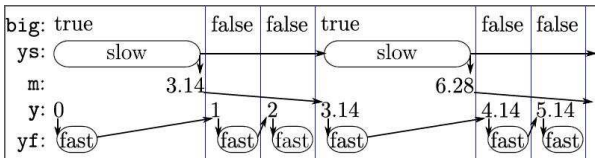

Figure 13. slow_fast sync flow

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 array copies. The Heptagon compiler implements original techniques to this aim, based on a programmer-guided, modular inter-procedural memory allocation procedure 0.

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's semantics is key to next generation tools for embedded platforms, reconciling programmer productivity, efficient and predictable compilers and analyzers, and parallel hardware.

VI. RUNTIME POWER MANAGEMENT TOOLS (RTPM)

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.

A. *RTPM approach*

During the application run, there are various opportunities which can be exploited by global run-time manager to optimise application and hardware platform performance. As depicted by Fig. 14, we propose run-time decisions during the lifetime of applications to be organized into two layers: coarse grained level L1 decisions triggered by dynamic events and fine grained level L2 decisions to improve application performance.

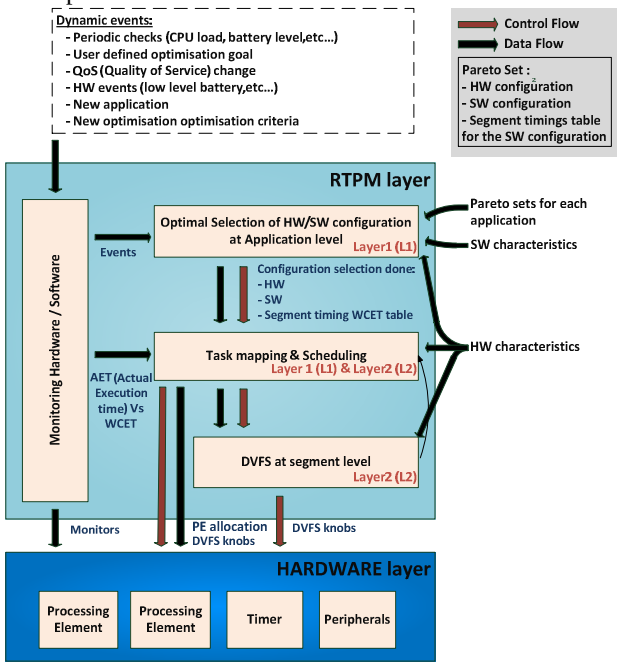

Figure 14. RTPM architecture

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 frames per second constraint in MPEG4 encoder to trade-off quality with performance) are tuned iteratively to optimize application and platform performance.

B. *Decision making at run-time*

During application run, there are various opportunities to optimise application and hardware platform performance. Global Run-time Manager (GRM) can decide to change platform and application parameters at run-time to exploit these opportunities. GRM needs to do this decision-making in a systematic way by using coarse-grained and fine-grained decisions Following is a high-level algorithm for the runtime decision making which will typically run on the controller of the hardware platform. In PHARAON, this run-time decision-making flow is enabled by using proposed RTPM architecture (Fig.14).

Inputs:

a) Hardware platform information e.g. available resources – both computation and communication, different knobs on those resources that can be tuned

b) Application information e.g. constraints – both hard and soft, multiple optimal operating points and their corresponding resource usage, Quality of Service (QoS) of application runs

c) External inputs regarding environment changes e.g. some additional sensors to indicate such changes or the sensors could be built-in on the hardware platform

1. *Decide platform resource allocation to applications*
2. *Select the optimum operating configuration for each application using 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 place 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 change or a dynamic event or inability to achieve expected performance, go to* ***Step 1***
9. *Go to* ***Step 6.***

C. *Low-Power scheduler*

The low-power scheduler is part of layer 2, as presented in section 6.1. It takes as input the selected application mode defined by the reconfiguration manager (see section 6.2), and accounts for the predefined SW and HW configurations. The SW configuration includes the application mode, associated deadline, and segment timing tables, while the HW settings consist of the active core count, voltage and frequency mode, as well as task affinity to specific cores.
The low-power scheduler developed as part of PHARAON combines a classical Earliest Deadline First (EDF) policy with Dynamic Voltage and Frequency Scaling (DVFS) mechanisms.

As depicted in Fig.15, the power management policy relies on the notion of Actual Execution Time (AET). The application AET is monitored and compared to the Worst Case Execution Times (WCET). In order to reduce power consumption while complying with the application deadline, the scheduler attempts to minimize application idle times by spreading tasks over as many active cores as possible within the SMP, as it lowers voltage and/or frequency.

The results in terms of energy savings depend on how often idle times occur, as well as their durations---which may be large if the WCET differs widely from the AET, and are thus very application-specific. For example, our H264 codec sample, running on an ARM Cortex A8, shows improvements ranging from 20% to 80% depending on the desired QoS.

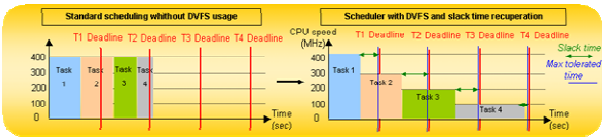

Figure 15. DVFS scheduling compared to regular scheduling

The first task of an application is executed with the core configuration as initially applied by the dynamic reconfiguration operation. During the execution of a task, modifications of the core state may then only occur at certain specific points, termed segment boundaries. A segment is a section of code under timing constraint, which exports timing information at runtime. This decomposition enables the scheduler to identify and monitor the different execution paths taken by a task, in order to maintain an accurate vision of the work already accomplished and the work that remains to be done in a given task. This reporting is achieved through specific additional APIs, in conjunction with an extension of POSIX threads, allowing timing annotations on thread-based tasks.

From a user perspective, to benefit from this runtime, applications have to be instrumented with segments through the aforementioned API calls. Splitting a task into segments constitutes a trade-off. More segments help the scheduler track progression better, but incurs more timing overhead. In addition to these API hooks, the low-power scheduler requires a user-supplied segment table, which maps the name of all the segments in the application to their associated WCET, at the different core operating points, as well as their deadlines.

For portability reasons, the current version of the scheduler has been written to run in user space. It requires a FIFO priority-based host scheduler, hardware support for DVFS capacities, as well as POSIX threads for the application interface. An implementation in kernel space could be built on the same principles.

In the case of PHARAON, experiments are conducted on a multicore Cortex-A9 platform, running Linux-SMP, using the CPUfreq framework for DVFS. By relying on CPUfreq, we leverage the standardized hardware abstraction of Linux, making the approach transparently compatible with future HMP platforms with big.LITTLE architectures.

As part of the PHARAON project, applications defined in section 2 are currently being manually instrumented with the timing APIs. Simultaneously, more ongoing work focuses on: (1) automatically generating these calls based on the code generator presented in section 4.2; (2) estimating the application segment WCET via the Pareon tool; (3) integrating the APIs in the OpenStream runtime.

VII. Conclusion

Various tools, techniques and runtime to help application mapping on multiprocessors platforms have been presented. Results should benefit to reduce development time, increase performance and reduce power consumption. To ease the usage of this methodology, information related to tools are integrated in the UML models in order to generate automatically relevant data used by tools and runtime.

Acknowledgment

This work has been performed in the framework of the EU FP7-288307 funded project PHARAON.

References

[1] Pareon doc, http://www.vectorfabrics.com/docs/pareon/current/

[2] Nhat Minh Lê, Antoniu Pop, Albert Cohen, and 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.

[3] Antoniu Pop and Albert Cohen. OpenStream: Expressiveness and data-flow compilation of OpenMP-streaming programs. ACM Transactions on Architecture and Code Optimization (TACO), January 2013. Selected for presentation at the HiPEAC 2013 Conf.

[4] Albert Cohen, Léonard Gérard, and Marc Pouzet. Programming parallelism with futures in Lustre. In ACM Conf. on Embedded Software (EMSOFT), Tampere, Finland, October 2012. Best paper award.

[5] 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.

[6] 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.

[7] D. C. Schmidt, “Model-driven Engineering” IEEE Computer, vol. 39 no. 2, pp. 25-31, 2006

[8] H. Posadas, P. Peñil, A. Nicolás, E. Villar "Automatic synthesis of Embedded SW Communications from UML/MARTE models supporting memory-space separation", XXVII Conference on Design of Circuits and Integrated Systems, DCIS'12. 2012-11

[9] P. Peñil, H. Posadas, A. Nicolás, E. Villar "Automatic synthesis from UML/MARTE models using channel semantics", International Workshop on Model-Based Arquitecting and Construction of Embedded Systems, ACES-MB 2012. 2012-09