Dataflow-Based Rapid Prototyping for Multicore DSP Systems

Technical Report PREESM/2014-05TR01, 2014

Maxime Pelcat, Karol Desnos, Julien Heulot,
Clément Guy, Jean-François Nezan, Slaheddine Aridhi

1 Introduction

The currently available high performance signal processing systems manufactured by Texas Instruments, including the Keystone I [15] and Keystone II [14] architectures, as well as the TDA2x with Embedded Vision Engine (EVE) [13], are heterogeneous multiprocessor architectures. These architectures currently embed up to 12 heterogeneous cores and this number is likely to increase in the next generations in order to cope with the ever more complex applications.

In this context, training engineers to program Digital Signal Processor (DSP) architectures is a challenging task because the impact of application and architecture bottlenecks on system performance is high and difficult to predict. The main challenges to overcome when designing a multicore signal processing system are:

- to exploit enough algorithm parallelism (task, data and pipeline parallelisms) to minimize latency in general,
- to choose the right core for each application subtask,
- to provide data where and when needed so as to avoid stalling cores and underusing the hardware, hence maximizing the usage efficiency.

PREESM is an Eclipse-based framework that provides dataflow-based methods to study a multicore DSP system. The framework is open-source and provided with extensive tutorials for easy initiation of C/C++ programmers to multicore DSP programming. PREESM provides a system designer with high level rapid prototyping information on algorithm parallelism and latency, as well as on system memory requirements. Moreover, a code generation is provided to transform the dataflow representation into a compilable code. Actors are manually implemented by the system designer in the language supported by the architecture compiler (e.g. C or C++ code for the Texas Instruments TMS320C6678 DSP) and the executable resulting from compiling together the generated and the manual code constitutes a multicore system prototype that is guaranteed deadlock-free and can be retargeted to a different number of cores within a few minutes.

Dataflow Models of Computation (MoCS) are a promising solution to the limitation of imperative languages (C, C++...) to represent application parallelism. The Parameterized and Interfaced Synchronous Dataflow (PiSDF) [5] dataflow MoC tested within the PREESM framework Version 2 divides an application into parallel processes called *actors*. These actors communicate through data *First In, First Out data queues (FIFOs)* and their data production and consumption rates on these FIFOs can be modified by *parameters*.

The paper is organized as follows: after a presentation of related works in Section 2, an overview of the PREESM framework is given in Section 3. A focus is put on the algorithm and architecture models in Section 4 and the most important rapid prototyping task are explained in 5. Experimental results of the PREESM rapid prototyping process are discussed in Section 6. The chosen use case is a stereo matching algorithm running on an 8-core TMS320C6678 DSP.

2 Related Works

OpenMP [3] is becoming the defacto standard for multiprocessor programming. Most C-based toolchains, including the Texas Instruments toolchain for multi-C66x core DSPs, support OpenMP. OpenMP is a set of pragmas for parallelizing loops or sections of an imperative (e.g. C) code. Current OpenMP implementations are oriented towards the dynamic creation of tasks by a runtime support based on pragma information. In this context, no rapid prototyping is available to check the system conformance to performance constraints.

The creation of the PREESM rapid prototyping framework has been inspired by the Algorithm-Architecture Matching methodology (AAM, also sometimes called AAA) [8]. AAM consists in simultaneously searching the best software and hardware configurations for respecting the system constraints. The SynDEx tool [7] is also based on the AAM methodology but it differs from PREESM on several ways: SynDEx is not open source, has a unique dataflow Model of Computation that does not support schedulability analysis and the function of code generation is possible but not provided with the tool. Schedulability analysis is an important feature of PREESM because it ensures deadlock freeness in the generated code.

The Open RVC-CAL Compiler (Orcc) [16] is an open-source tool that generates different types of hardware and software code from a unique dataflow-based language named RVC-CAL. The recent TURNUS tool [2] is an exploration tool that complements the Orcc compiler by offering RVC-CAL design space exploration. An important difference between Orcc, TURNUS and PREESM is the MoC of the algorithm description. While PREESM uses the decidable [1] PiSDF MoC, the MoC implemented in RVC-CAL is not decidable and thus, in the general case, no guarantee can be given in Orcc and TURNUS on the deadlock-freeness and memory boundedness of the generated code.

SDF3 [12] is an open-source dataflow analysis tool that supports the Synchronous Dataflow (SDF), Cyclo-Static Dataflow (CSDF) and Scenario-Aware Dataflow (SADF) MoCS. SDF3 is oriented towards model analysis and simulation while PREESM aims at both simulating the system and generating an executable prototype.

The features that differentiate PREESM from the related works and similar tools are:

- the tool is open source and accessible online[1];
- the algorithm description is based on a single well-known and predictable model of computation;
- the scheduling is totally automatic;
- the functional code for heterogeneous multi-core embedded systems is generated automatically;
- rapid prototyping metrics are generated to help the system designer to take decisions;
- the PiSDF algorithm model provides a helpful hierarchical encapsulation and parameterization, thus simplifying the scheduling;

[1] http://preesm.sourceforge.net/website/

- the System-Level Architecture Model (S-LAM) architecture model provides a high-level architecture description to study system bottlenecks.

Next Sections cover these features through a presentation of the rapid prototyping tool chain of PREESM.

3 Rapid Prototyping Overview

Figure 1 shows a PREESM typical rapid prototyping process described in the PREESM tool by a graphical *workflow*. A PREESM workflow is a graph connecting rapid prototyping tasks such as scheduling and simulation. Each task is implemented in a different Eclipse plug-in, providing a high scalability to the tool. Workflow support is a feature that makes PREESM scalable and adaptable to designer's needs. A developer tutorial[2] provides all necessary information to create new workflow tasks and adapt PREESM to a designer's needs (e.g. for exporting a graph in a custom syntax or for experimenting new scheduling methods).

An algorithm graph description (in PiSDF), an architecture graph description (in S-LAM), and a scenario are retrieved from XML files (left hand side of Figure 1). Graphic editors are provided within PREESM for all the rapid prototyping inputs. These editors ease manipulation and edition of algorithms and architectures and thus the exploration of the design space.

A scenario is a database providing all information to link an algorithm and an architecture. It enables a clear separation of concerns between algorithm and architecture design. For instance, the scenario contains the deterministic execution time of each actor on each type of cores.

Algorithm and architecture models undergo several transformations (including a graph flattening, equivalent to loop unwinding in a compilation process) to expose parallelism to the scheduling process. They are then passed to a static scheduling and a memory optimization tasks. The static scheduling transformation generates a periodic multicore schedule that will be repeated indefinitely to process the input data stream. At the end of each iteration of the periodic multicore schedule, the FIFO queues are back in their initial state, ensuring deadlock freeness. The memory optimization task computes a memory exclusion graph in order to authorize memory reuse between FIFOs in the intermediate representation used for code generation.

Finally, the simulation and the code generation tasks provide respectively metrics for system design and a prototype for testing the multicore execution of the system. Next Section presents in more details the algorithm and architecture models used in PREESM.

4 Rapid Prototyping Input Models

4.1 The Parameterized and Interfaced Synchronous Dataflow (PiSDF) Model of Computation

The PiSDF dataflow model [5] used to describe algorithms aims at providing coarse grain parallel descriptions of algorithms specifying precisely the data

[2] http://preesm.sourceforge.net/website/new-workflow-task

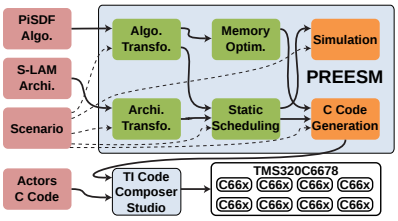

Figure 1: PREESM Rapid Prototyping Process: An Example of a Workflow

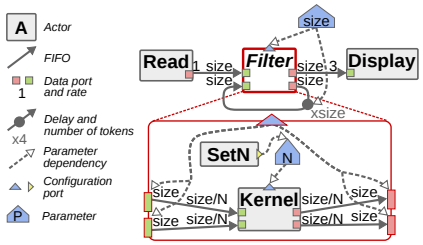

Figure 2: An Example of a PiSDF Model

flowing between actors and offering a tradeoff between dynamic behavior and predictability.

An example of a PiSDF model is shown in Figure 2. This example corresponds to a filter that takes a fixed *size* number of data tokens (indivisible unit of data) as inputs and internally distributes these tokens to N kernels that effectively execute the filtering. This example can for instance be used to execute the filtering of an image by slice (without overlapping between slices) so as to provide data parallelism.

The internal behavior of actors is programmed in plain C code. Actors are authorized to access their input and output data tokens in any order. This construction automates the computation of the number of executions (or *firings*) of the actors per invocation of the graph. This computation is based on the production and consumption of data and can ensure deadlock-freeness at compile-time.

Actors are stateless and the only possibility for an actor to keep a state information between two executions is to send information to its future iterations. The feedback loop of the actor $Filter$ with a delay of *size* tokens provides the $Kernel$ actor with the corresponding slice of the preceding processed image.

Production and consumption rates of PiSDF actors, as delays of FIFOs between actors, can be parameterized depending on the need of the application. In our example, *size* and N are two parameters enabling to vary the number of tokens filtered and the number of kernels executing the filtering. The value of these parameters can be modified at run-time, in which case we talk about *dynamic parameters*.

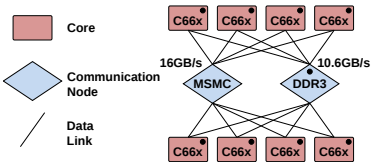

Figure 3: An Example of a S-LAM Model

4.2 The System-Level Architecture Model (S-LAM)

With the convergence of hardware and software languages, the reconfigurable architectures and the High-Level Synthesis (HLS), the distinction between hardware and software is becoming unclear. S-LAM describes architectures in a graph as *a set of cooperative processing elements offering processing capabilities to dataflow actors and a set of communication features offering communication services to dataflow FIFOs* [10]. This definition authorizes the use of S-LAM for describing parallel architectures at different levels of granularity: a set of PCs communicating through ethernet, a set of cores communicating through shared memory, etc.

An example of an S-LAM model is shown in Figure 3. It describes a TMS320C6678 architecture by describing the 8 C66x cores and the capacity of each core to communicate with any other core through two parallel communication nodes: an internal shared memory (Multicore Shared Memory Controller (MSMC)) and an external shared memory (Double Data Rate SDRAM (DDR3)). The graph does not represent the architecture itself but the services it provides to the dataflow algorithm. The same hardware architecture can thus be represented in several ways depending on the modeled communication capacities.

The black dots represent Time Division Multiple Access (TDMA) for the data transfers on the communication nodes and for the actor executions on the processing elements. The absence of a black dot on the MSMC node means that simulation will consider it as capable of managing any number of communication simultaneously. The motivation for this representation is that MSMC controller has an independent link for each core at 16 GB/s while the external DDR3 has a unique link to the processor at 10.6 GB/s. More advanced features are representable in S-LAM such as the delegation of communications to a Direct Memory Access (DMA). Next section describes the main PREESM workflow tasks which process PiSDF algorithm models and S-LAM architecture models.

5 Rapid Prototyping Tasks

Rapid prototyping consists in exploring the design space of a target system in order to minimize its cost and guarantee the respect of different constraints, the most common ones being: latency, throughput, memory, and energy consumption. Other constraints may exist, such as jitter or signal simultaneity. The diversity of constraints invalidates a unique approach targeting all types of signal processing systems. However, it fosters frameworks such as PREESM with plugged-in functionalities that adapt to different targets.

The PREESM multicore scheduler implements the *List* and *Fast* scheduling methods described by Kwok [9]. The current PREESM plug-ins are focusing on latency dominated systems. A latency dominated system is a system where the respect of the latency constraints of the processing assures the respect of its throughput constraints [6]. In this case, the main processing iteration does not need to be pipelined (i.e. only one iteration of the processing is alive at any moment) but it may need to be parallelized if the work to execute is longer in time than the latency constraint. In the current PREESM code generation, all cores are synchronized with a barrier between two application iterations. It is important to note that while inter-iteration pipelining is not supported, intra-iteration pipelining of actors is already available. This constraint may be relaxed in the future, at the cost of a more complex memory and time analysis.

Bounded memory execution is an important property of decidable MoCS [1]. Without this property, unexpected deadlocks can appear when a FIFO becomes out of memory. PREESM currently generates guaranteed deadlock-free code for PiSDF algorithms with purely static parameters. An extension to more dynamic parameters is foreseen in the short term. An advanced memory optimization mechanism based on a memory exclusion graph [4] is available in PREESM to avoid preserving fifo memory spaces that are useless for the correct system execution.

The PREESM simulation offers to the system designer a simulated Gantt chart of the code execution on the parallel architecture, a *speedup assessment chart* that draws the expected algorithm execution speedup depending on the number of cores, and an evaluation of the memory necessary for the execution. The code generation produces a self-timed code [11], i.e. a static code for each core with automatic inter-core communication, cache management and synchronization. This code necessitates communication libraries, which are provided for the TMS320C6678 processor.

A more complete description of PREESM rapid prototyping tasks can be found in [10]. Next Section will provide some experimental results using the tasks described above.

6 Experimental Results

Through the example of a state-of-the-art computer vision application, this Section presents how PREESM can be used to develop an application and automatically deploy a prototype implementation on a multicore Keystone architecture.

6.1 Use Case: Stereo Matching Algorithm

The application studied is a stereo matching algorithm. The purpose of stereo matching algorithm is to process a pair of images taken by two cameras in order to produce a disparity map that corresponds to the 3^{rd} dimension (the depth) of the captured scene. The large computation and memory requirements of stereo matching algorithms, as well as their promising use in Advanced Driver Assistance Systems (ADAS) [13], make them interesting case studies to illustrate the efficiency of the PREESM rapid prototyping framework.

Figure 4 presents the top-level PiSDF graph of a stereo-matching algorithm. Two parameters are used to configure this PiSDF graph: **size**, which corre-

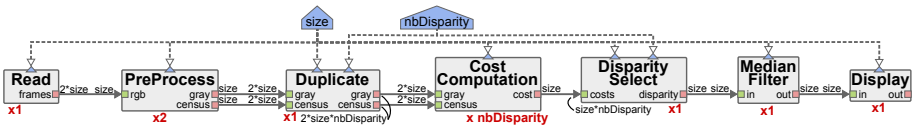

Figure 4: PiSDF graph of the stereo matching application

sponds to the number of pixels of each image of the input stereo pair processed by the algorithm; and **nbDisparity**, which represents the number of distinct values that can be found in the output disparity map.

The stereo matching PiSDF graph presented in Figure 4 contains 7 actors:

- **Read** produces the 2 input frames by reading a stream or a file.
- **PreProcess** converts an RGB image into its grayscale equivalent. This actor also produces an 8-bit signature, called census, for each pixel of an input image. This signature results from the comparison of each pixel with its 8 neighbors.
- **Duplicate** produces a configurable number of copies of the data received on its input ports.
- **CostComputation** computes the matching cost of each pixel for a given disparity. This actor is called as many times as the number of tested disparities.
- **DisparitySelect** produces a disparity map by selecting the disparity of the input cost map with the lowest matching cost for each pixel.
- **MedianFilter** applies a 3x3 pixels median filter to the input disparity map to smooth the results.
- **Display** displays the result of the algorithm or writes it in a file.

Below each actor is a repetition factor which indicates the number of executions of this actor for each iteration of the graph. This number of executions is computed from the data production and consumption rates of actors. The PiSDF description of the algorithm provides a high degree of data and task parallelism since it is possible to execute in parallel the repetitions of the most computationally intensive actors, namely *PreProcess* and *CostComputation*. In addition to the parallelism expressed in this top-level graphs, several actors can be refined with hierarchical subgraphs also containing parallelism. For example, the *PreProcess* and *Median* actors can both be implemented in such a way that several parts of their input images are processed simultaneously in parallel.

6.2 Preesm results

The results presented in this section are obtained by applying the PiSDF graph of Figure 4 to stereo pairs of $size = 450 * 375$ pixels, with $nbDisparity = 60$. In this configuration, 300 actor firings must be scheduled for each iteration of the complete PiSDF graph, and 987 FIFOs must be allocated in shared memory.

The time taken by the Preesm framework to automatically deploy the stereo matching application on the eight cores of a TMS320C6678 chip is 45 seconds.

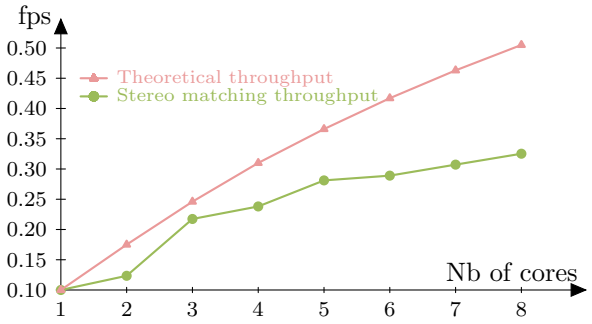

Figure 5: Throughput of the stereo matching application depending on the number of targeted C6x cores.

The scheduling process is responsible for 53% of this time, memory optimization for 36%, graph transformations for 9%, and code generation for 2%. The execution time of the rapid prototyping process remains relatively low compared to the 140 seconds needed to compile the application for the multicore DSP. The fast execution of the Preesm framework is a key feature to accelerate the testing of different deployment scenarios and ease the design space exploration on different multicore architectures.

Figure 5 shows the performance obtained by deploying the stereo matching algorithm on a variable number of cores of the TMS320C6678 multicore DSP chip. On eight cores, a throughput of 0.33 frames per second (fps) is reached. This throughput corresponds to a speed-up by a factor 3.3 compared to the execution of the application on one DSP core.

Figure 5 also plots the theoretical greedy scheduling throughputs [10] computed by Preesm for the stereo matching application. In the current version, the computation of this theoretical throughput does not take into account inter-core communications nor cache operations. It could be considered as an upper bound for the achievable throughput. Consequently, the actual throughput of the stereo matching algorithm appears to be inferior to the theoretical throughput.

Figure 6 shows the data memory footprint allocated for the execution of the stereo matching algorithm on a variable number of cores of the TMS320C6678 multicore DSP chip. The smallest memory footprint of 68.4 MBytes is obtained when the application is executed on a single core of the architecture. When the number of cores executing the application is increased, more parallelism of the application is preserved, and the allocated memory footprint is increased. As illustrated in Figure 6, the memory optimization techniques used in Preesm [4] limit this increase of the memory footprint allocated for the application, and only 79.7 MBytes of memory are needed to execute the application on 3 to 8 cores. Without the memory optimization techniques of Preesm, more than 1.4 GBytes would be needed to execute the application, which is far more than the 512 MBytes available on the EVMC6678.

These experimental results show that the Preesm framework can be used to deploy and optimize real applications on multicore architectures within minutes.

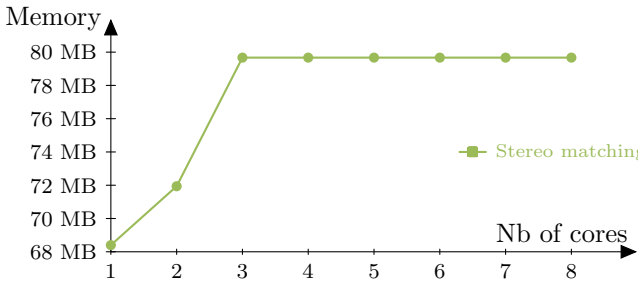

Figure 6: Memory footprint of the stereo matching application depending on the number of targeted C6x cores.

Several tutorials[3] are available online to demonstrate the framework functionalities.

7 Conclusion

PREESM provides a complete rapid prototyping framework for multicore DSP system design. As a C-based open-source framework distributed with complete tutorials, it aims at initiating C programmers and system designers to dataflow methods. Furthermore, the PREESM framework has demonstrated capabilities for developing an optimized real world application (e.g. the stereo matching algorithm) for multicore DSP architectures.

[3]http://preesm.sourceforge.net/website/tutorials

Bibliography

[1] Bhattacharyya, S.S., Levine, W.S.: Optimization of signal processing software for control system implementation. In: IEEE International Symposium on Intelligent Control, pp. 1562–1567 (2006)

[2] Brunei, S.C., Mattavelli, M., Janneck, J.W.: TURNUS: a design exploration framework for dataflow system design. In: ISCAS 2013, pp. 654–654. IEEE (2013)

[3] Chapman, B., Jost, G., Van Der Pas, R.: Using OpenMP: portable shared memory parallel programming, vol. 10. MIT press (2008)

[4] Desnos, K., Pelcat, M., Nezan, J.F., Aridhi, S.: Pre-and post-scheduling memory allocation strategies on MPSoCs. In: ESLsyn 2013. IEEE (2013)

[5] Desnos, K., Pelcat, M., Nezan, J.F., Bhattacharyya, S.S., Aridhi, S.: Pimm: Parameterized and interfaced dataflow meta-model for mpsocs runtime reconfiguration. In: SAMOS XIII, pp. 41–48 (2013)

[6] Ghamarian, A.H., Geilen, M.C.W., Stuijk, S., Basten, T., Moonen, A.J.M., Bekooij, M.J., Theelen, B.D., Mousavi, M.: Throughput analysis of synchronous data flow graphs. In: ACSD 2006, pp. 25–36 (2006)

[7] Grandpierre, T., Lavarenne, C., Sorel, Y.: Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In: CODES. ACM (1999)

[8] Grandpierre, T., Sorel, Y.: From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations. In: MEMOCODE (2003)

[9] Kwok, Y.K.: High-performance algorithms for compile-time scheduling of parallel processors (1997)

[10] Pelcat, M., Aridhi, S., Piat, J., Nezan, J.F.: Physical Layer Multi-Core Prototyping: A Dataflow-Based Approach for LTE eNodeB, vol. 171. Springer (2012)

[11] Sriram, S., Bhattacharyya, S.S.: Embedded multiprocessors: Scheduling and synchronization. CRC press (2012)

[12] Stuijk, S., Geilen, M., Basten, T.: SDF^3: SDF For Free. In: ACSD 2006 (2006)

[13] Texas Instruments: Empowering automotive vision with TIs Vision AccelerationPac - SPRY251. URL http://www.ti.com/lit/pdf/spry251(accessed05/2014)

[14] Texas Instruments: Multicore DSP+ARM KeyStone II System-on-Chip (SoC) - SPRS866E. URL http://www.ti.com/lit/pdf/sprs866e(accessed05/2014)

[15] Texas Instruments: Multicore Fixed and Floating-Point Digital Signal Processor - SPRS691E. URL http://www.ti.com/lit/pdf/sprs691e(accessed04/2014)

[16] Wipliez, M.: Compilation infrastructure for dataflow programs. Ph.D. thesis, Ph. D. thesis, INSA de Rennes (2010)