Scalable Compile-Time Scheduler for Multi-core Architectures

Maxime Pelcat* Pierrick Menuet* Slaheddine Aridhi** Jean-François Nezan*

*IETR/INSA, UMR CNRS 6164,
Rennes, France
mpelcat, pmenuet, jnezan@insa-rennes.fr

**Texas Instruments, CIV Division,
Villeneuve Loubet, France
saridhi@ti.com

Abstract

As the number of cores continues to grow in both digital signal and general purpose processors, tools which perform automatic scheduling from model-based designs are of increasing interest. This scheduling consists of statically distributing the tasks that constitute an application between available cores in a multi-core architecture in order to minimize the final latency. This problem has been proven to be NP-complete. A static scheduling algorithm is usually described as a monolithic process, and carries out two distinct functionalities: choosing the core to execute a specific function and evaluating the cost of the generated solutions. This paper describes a scheduling module which splits these functionalities into two sub-modules. This division produces an advanced scalability in terms of schedule quality and computation time, and also separates the heuristic complexity from the architecture model precision.

1. Introduction

Currently, real-time complex signal processing algorithms mostly run on multi-core architectures. Consequently, these algorithms must be divided into tasks and these tasks must then be distributed between the cores to be executed in parallel.

Manually exploring the potential parallelism of an application using new architectures with a high number of cores is greatly time-consuming. Happily, dataflow models have proven to be efficient representations of these applications [1][2], thus allowing the automation of scheduling signal processing algorithms onto complex architectures. The scheduling process is also known as distribution/scheduling or mapping/scheduling.

The role of the scheduler is to statically choose the core to execute each task, with the overall goal of minimizing the global latency. This problem has been proven to be NP-complete in [3] for realistic cases (more than 2 cores, tasks with different timings, and so on).

Scheduling algorithms have been the subject of intense study for the past few years. In his PhD thesis [4], Y. K. Kwok presents high-performance algorithms focusing on their complexity and on the analysis capacity to achieve the smallest latency. The model he used to represent the architecture behaviour is a basic one: the architecture is homogeneous and communications have a given fixed cost for the transfer between cores. In [5], Oliver Sinnen analyses the models that should be used to simulate implementation during scheduling. He introduces the idea of routes and edge scheduling to model realistic systems.

The Institut d'Electronique et de Télécommunications de Rennes (IETR), in collaboration with Texas Instruments, is currently developing an architecture exploration tool. This tool is called Parallel Real-time Embedded Executives Scheduling Method (PREESM) and is based on the Algorithm Architecture Matching (AAM) methodology (formerly known as AAA [1]). Initial results were obtained on the 3G Long Term Evolution (LTE [6]) preamble detection application with a first version of the PREESM tool, and show the possibilities of a Synchronous Dataflow Graph (SDF) description with automatic mapping [7]. The scheduler discussed here is part of a new version of the tool based on plug-ins and the Eclipse framework. This is a combination of scheduling and implementation simulation ideas. This approach improves reusability and upgradeability of mapping algorithms in the rapid prototyping framework. The tool also has a test capability, allowing the investigation of existing advanced scheduling algorithms with precise architecture models.

This paper aims to give an overview of the scheduling process used by the PREESM tool. Firstly, the objectives of this scheduler are explained. Then, the scheduler and its sub-modules are described. Finally, Gantt charts of scheduling results are discussed and future work introduced.

2. Overview and objectives of the scheduler

978-3-9810801-5-5/DATE09 © 2009 EDAA

2.1. The PREESM tool

The objective of the PREESM tool is to model signal processing algorithms and platform architectures, as well as generating the mapping and associated code for parallel architectures. The first architectures targeted for this study are the multi-core processors from Texas Instruments which use an Enhanced Direct Memory Access (EDMA) peripheral for message-passing between cores.

The tool was developed with two goals: rapid prototyping and effective code generation. Good scalability in the workflow is important to ensure the user can either roughly evaluate the behaviour of his application within minutes or finely deploy his application on a multi-core architecture at the cost of hours of work and processing.

The PREESM tool requires descriptions of the target architecture, the studied algorithm or application, and the scenario of the study. A workflow is then applied to this input information and generates a simulation and/or code.

2.2. A usual workflow

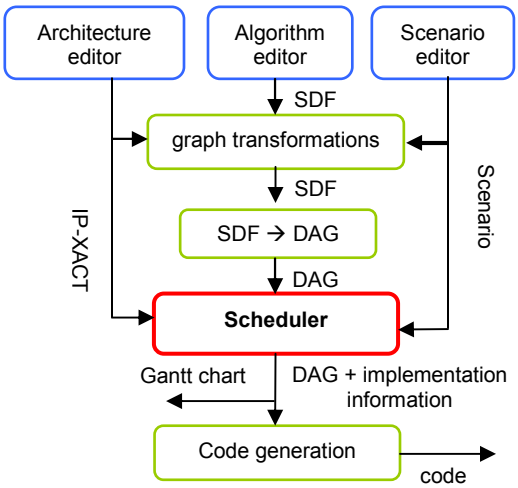

Fig. 1. From SDF and IP-XACT descriptions to code

Fig. 1 describes a classic workflow which can be applied in the PREESM tool. The dataflow model chosen to describe applications in PREESM is the SDF model. This model, described in [8], has the great advantage of allowing formal verification of static schedulability. The typical number of vertices to schedule in PREESM is between a hundred and a thousand. The architecture is described using IP-XACT language, an IEEE standard from the SPIRIT consortium [9]. The typical size of an architecture in PREESM is between a few cores and a few dozens of cores. For the present context, a scenario is defined as a set of parameters and constraints, to specify the conditions under which the implementation will run.

The purpose of the graph transformations module is to modify the SDF graph. It generates clusters [10] which simplify the mapping by reducing the number of vertices to schedule and then it prepares a loop-compressed representation in the generated code. Subsequently, the SDF graph is converted into a Directed Acyclic Graph (DAG), which has a lower expressivity than SDF graph but is the input suitable for the scheduler [11]. As a result of the implementation, code is generated and a Gantt chart of the execution is displayed. The generated code consists of function calls. The functions themselves are hand-written. The originality of the PREESM scheduler is in its structure, discussed in the next section.

3. The PREESM scheduler

3.1. Scheduler structure

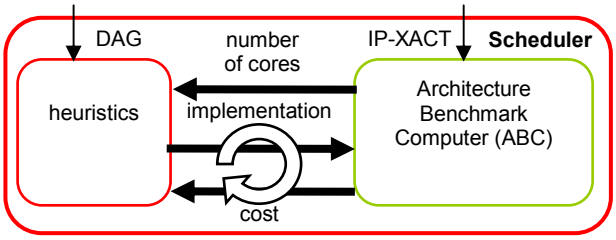

Fig. 2. Scheduler module structure

The PREESM scheduler is divided in two sub-modules which share a minimal interface: the heuristics and Architecture Benchmark Computer (ABC) sub-modules. The heuristics sub-module determines a scheduling solution of the application tasks onto the architecture cores and then questions the ABC sub-module to evaluate the cost of the proposed solution. The advantage of this approach is that any heuristic may be combined with any ABC, leading to many different scheduling possibilities.

The interface offered by the ABC is minimal: it gives the heuristic the number of available cores, receives a DAG with mapping instructions and returns its cost (infinite if the implementation is impossible).

3.2. Scheduling heuristics

Three algorithms are currently coded; all of which are defined in [4]:

- A **list scheduling** algorithm schedules the tasks in the order of a list constructed by calculating a critical path. Once a mapping choice has been made, it will never be modified. This algorithm is quite fast but has limitations due to this last property. List scheduling is used as a starting point for other refinement algorithms.
- The **FAST algorithm** is a refinement of the list scheduling solution using probabilistic hops. It can run until stopped by the user and keeps the best latency found. The algorithm is multi-threaded to exploit the multi-core parallelism of a computer.

- A **genetic algorithm** is coded as a refinement of the FAST algorithm. The n best solutions of FAST are used as the base population for the genetic algorithm. Again, the user is free to stop the processing at any time. This algorithm is also multi-threaded.

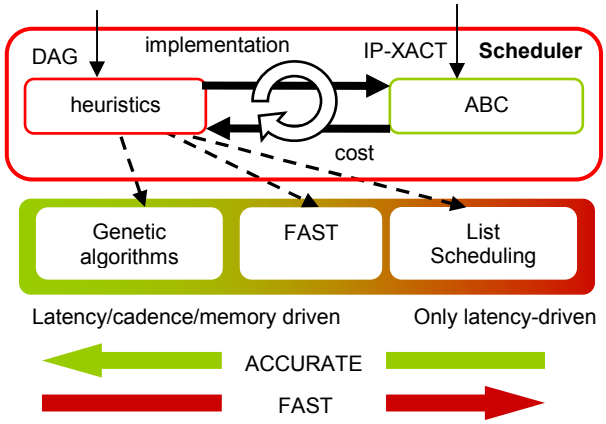

Fig. 3. Switchable scheduling algorithms

3.3. Architecture Benchmark Computer

Scheduling often requires much time. Testing intermediate solutions with precision is an especially time-consuming operation. The ABC was created as a reuse of the concept of time scalability introduced in SystemC Transaction Level Modelling (TLM) [12]. This language defines several levels of system temporal simulation, from untimed to cycle-accurate precision.

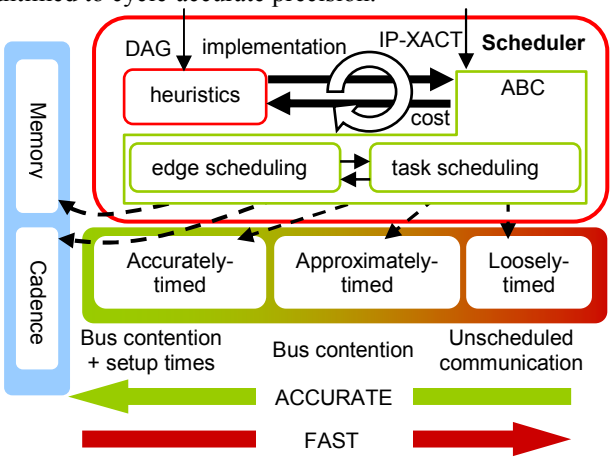

Fig. 4. Switchable ABC models

Three ABC models, with different timings, are currently coded: The **loosely-timed model** takes into account task and transfer times but no transfer contention. The **approximately-timed model** associates each inter-core communication medium with its constant rate and simulates contentions. The **accurately-timed model** adds set-up times which simulate the duration necessary to initialize a parallel transfer controller like Texas Instruments EDMA. This set-up time is scheduled in the core sending the transfer and is a new feature compared to existing models in the literature. These three models target heterogeneous architectures totally connected via routes. The media are assumed to be half-duplex message passing systems, which closely model EDMA behaviour. All ABC models can evaluate the latency of both partially and totally scheduled implementations. This is necessary because in list scheduling, the scheduling choice of a task n exploits the cost of a partial schedule of the n-1 previous tasks. The task and architecture properties feeding the ABC are experimentally evaluated. They include media transfer speed, set-up times and tasks timings.

To these ABCs, a cycle accurate model for Texas Instruments platforms has been added. This model calls a TI simulation tool to allow a precise evaluation of the application latency. Moreover, ABC models evaluating other parameters than latency are planed in order to minimize memory size, memory accesses, cadence (i.e. average runtime), and so on. Minimizing parameters other than latency cannot be performed with the list scheduling algorithm because these costs cannot be evaluated on partial implementations.

The ABC itself contains two parts. The first part, referred to as task scheduling, evaluates the task schedules provided by the chosen heuristic. The second part, referred to as edge scheduling, schedules the edges on the media and resolves the routes between cores. Edge scheduling can also be executed with different algorithms of varying complexity, which results in another level of scalability.

The main advantage of this scheduler structure is the total independence of algorithm type from cost type and benchmark complexity. The performance of scheduling algorithms is also improved, providing realistic cost inputs.

4. Using the scheduler framework

Using an SDF example of [4], Fig. 5 demonstrates that using different ABCs results in significantly different benchmarks. When model complexity increases, new contentions appear, increasing the simulated latency between Fig. 5 a) and c). With a complex ABC, list scheduling becomes inadequate; the local mapping choices ignore future tasks, thus risking problems with future communications. It is for this reason that list scheduling algorithm combined with accurately-timed ABC returns an implementation with a latency of 45 (Fig. 5 c)) while executing all tasks on one core would lead to a latency of 30 (Fig. 5 e)).

The same example was run using the SynDEx tool [1]. Its scheduler is a list scheduler accounting for communication contention in a similar way to the approximately-timed model. The obtained latency of 30 uses 3 cores (Fig. 5 d)) and can be compared to the latency of 40 of the PREESM list scheduling algorithm (Fig. 5 b)). The latency of 40 shows that the list scheduling algorithm from [4] is not ideally suited to be used directly with a contention

aware ABC. This is not a limitation of the scheduler framework but simply an indication that a more advanced list scheduling algorithm should be implemented. Running the scheduler until genetic algorithm refinements produce the best possible scheduling in terms of latency of this implementation using the approximately-timed ABC with a latency of 30 on one core (Fig. 5 e)).

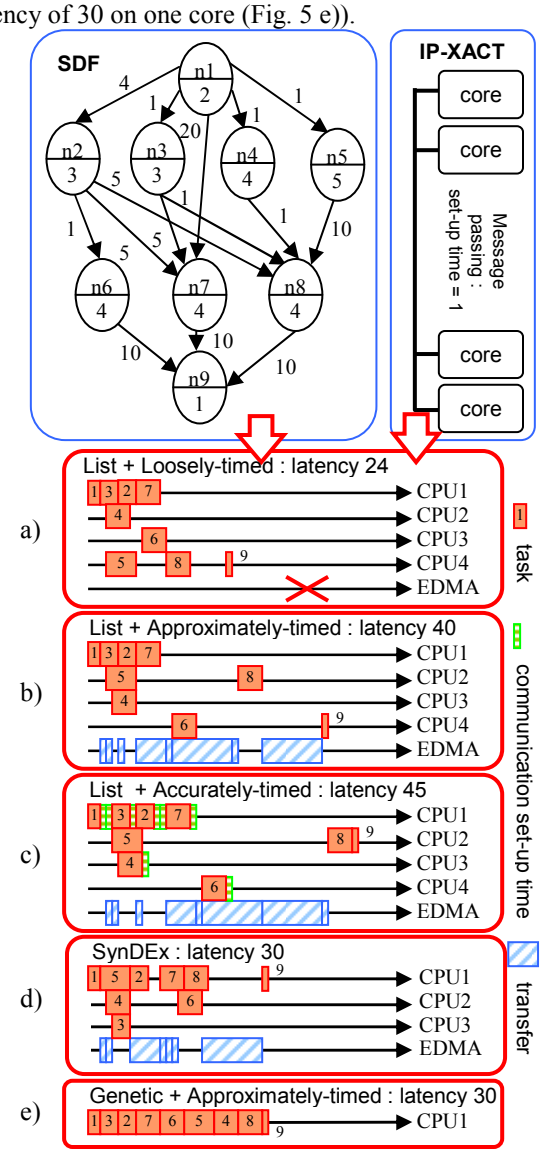

Fig. 5. Gantt charts of schedules with different Algorithms and ABCs

5. Future Work

Using more advanced algorithms than the list scheduling means that the mapping solution may be refined, and hence a number of parameters (including cadence and memory) may be optimized in new ABCs . For instance, given a list scheduling result of a latency of n, we could execute a genetic algorithm minimizing memory use or cadence while forcing the latency under n + 10%. As shown by the experimental results of [7], limited internal memory reduces the simulation quality because external memory accesses are correspondingly dramatically slowed. Minimizing memory size during scheduling in conjunction with memory accesses optimization is thus important.

6. Conclusion

The separation of the scheduler module into two distinct sub-modules produces a flexibility that could not be previously achieved. It enables a trade-off between scheduling convergence speed, schedule quality and model accuracy. It is the scheduler architecture that allows new scheduling heuristics to be implemented without taking into account the way the implementations are evaluated. Additionally, new architecture models may be developed in the Architecture Benchmark Computer (ABC) and tested with all available heuristics. This flexibility is an important feature in an automatic code partitioning tool for multi-core architectures.

References

[1] S. S. Bhattacharyya and R. Leupers and P. Marwedel, Software Synthesis and Code Generation for Signal Processing Systems, IEEE Transactions on Circuits and Systems, 2000.

[2] T. Grandpierre and Y. Sorel, *From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations*, In First ACM and IEEE Inter-national Conference on Formal Methods and Models for Co-Design, June 2003.

[3] Garey and Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman, 1979.

[4] Y.-K. Kwok, *High-performance algorithms for compile-time scheduling of parallel processors*, PhD. Thesis, 1997.

[5] O. Sinnen, *Task Scheduling for Parallel Systems*, Wiley, May 2007.

[6] *3GPP technical specification group radio access network; Evolved universal terrestrial radio access (E-UTRA)* (Release 8), 3GPP, TS36.211 (V 8.1.0).

[7] M. Pelcat, S. Aridhi and J.-F. Nezan, *Optimization of automatically generated multi-core code for the LTE RACH-PD algorithm*, DASIP conference, November 2008.

[8] E.A. Lee and D.G. Messerschmitt, *Synchronous Data Flow*, IEEE Proceedings of the IEEE volume 75, numéro 9, 1987.

[9] The SPIRIT Consortium, *IP-XACT v1.4: A specification for XML meta-data and tool interfaces*, March 2008.

[10] J. L. Pino and E. A. Lee, *Hierarchical static scheduling of dataflow graphs onto multiple processors*, In IEEE ICASSP, Detroit, Michigan, USA, 1995.

[11] J. L. Pino, S. S. Bhattacharyya and E. A. Lee, *A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs*, Laboratory, University of California at Berkeley, 1995.

[12] F.Ghenassia, *Transaction Level Modeling With SystemC: TLM Concepts And Applications for Embedded Systems*, Springer, 2006.