

FACSE: a Framework for Architecture and Compilation Space Exploration

Smail Niar, Nicolas Inglart, Mahdi Chaker*, Said Hanafi, Nasser Benameur
University of Valenciennes, Le Mont Houy, 59313 VALENCIENNES Cedex 9, France

*Ecole Nationale d'Ingénieurs de Sfax, B.P w.3038, Sfax, Tunisia

smail.niar, nicolas.inglart, said.hanafi, nasser.benameur@univ-valenciennes.fr, mahdichaker@yahoo.fr

Phone : +33 327511948, Fax : +33 327511940

Abstract-

To exploit microelectronics progresses for next processor generations, it is crucial to design new tools allowing a better adequacy between architectures and applications. The Facse project, which we present in this paper, strives to achieve this objective. As the two aspects of the system (hardware and software) are taken into account simultaneously, it is possible to increase the performances significantly. Facse is based on the use of adapted heuristics to guide search towards the most promising architecture and compiler optimization configurations in one side and rapid evaluation methods of these configurations on the other side.

Keywords: Architecture, processors, simulation, optimization, exploration.

I. INTRODUCTION

The next generations of processors will provide a very high degree of integration (over 1 billion of transistors on-chip) that is used for embedding several units inside the chip (processors, caches, interconnection network, etc.).

For instance, the recent Intel Itanium dual-core processor (alias Montecito) contains close to 2G transistors and 3 cache levels for a total of more than 20MB of on-chip cache. In the embedded processor area, the tendency is also towards the integration of more and more cores (processors, caches, I/O interfaces, network-on-chip, etc.) to provide a complete high performance integrated system. For applications that require an important computation and complex algorithms, Multi-Processor System on Chip (or MPSoC) represents an interesting alternative to ASIC. For designers of such architectures, the challenge focuses around two points: 1) Finding the optimal configuration of these units in order to: react rapidly to the user needs, increase performances and reduce fabrication costs and power consumption and 2) Offering to these units enough useful work to benefit from these technological advances.

Nevertheless, several obstacles prevent to succeed in these challenges. The first obstacle concerns the ever increasing design time. If currently, up to 2 years are required to end a (multi-)processor design phase, the usage of existing design tools for the next generation of processors will exacerbate the design and test phase delay. This time is extremely valuable and a long design phase can have a negative influence on the penetration into the market of a new product. Three factors affect this design and test phase:

- The important number of architectural configuration to take into account: This makes so that the search for the best processor configuration corresponds to exploring a very large space of possible micro-architecture configurations. Each configuration in this space can be characterized by the parameters such as: the data and instructions cache sizes, the branch prediction technique, the number of ALU, etc. In the Tensilica Xtensa [Gonzalez00] ASIP for instance, the designer has the ability to configure the instruction set, the caches, etc. For evident economic reasons, this design space exploration is generally performed using a cycle-accurate simulator. In SimpleScalar, one of the most popular micro-architecture simulators [Burger97], more than 100 parameters are available. If we consider that each parameter may have 4 different values, this makes up a space of 100^4 configurations to explore.
- An important number of applications and/or data input sets. For each, processor configuration, it is required to measure the performance of the target applications with it different input data sets. The EEMBC [Levy05] test benchmark contains 14 applications with 5 data input sets in average per application. This gives a total of 6700 application/data input combinations. Moreover, each application can be compiled using different options such as loop unrolling, software pipelining, tiling, etc.
- Last but not least, the third factor concerns the increasing processor architecture complexity. In the next generation of MPSoC, an important number of tasks are accomplished in parallel. Nevertheless, when a simulator is used to test and evaluate these architectures, tasks are accomplished sequentially. Thus more the platform to design is complex more time will be needed for simulation.

The second obstacle concerns the lack of tools to reach the second challenge. In order to permit to the applications to take advantage of all the available resources, it is necessary to use software (programming language, compiler, etc.) to generate parallel tasks and thus to increase the system efficiency.

The FACSE (for Framework for Architecture and Compilation Space Exploration) project intends to take up the two challenges previously mentioned. More precisely, it aims to design a flexible framework for an adaptation of the micro-architecture and the compilation process in one side to the application in the other side.

This paper gives an overview of the capabilities of FACSE and presents its philosophy and its general structure.

II. FACSE : A GENERAL OVERVIEW

FACSE provides, to the user, tools permitting to achieve: 1) A fast Design Space Exploration (DSE) of processor micro-architectures to find a "good" micro-architecture configuration for a specific application or set of applications. 2) A fast Compilation Option Space Exploration (COSE) to find interesting compilation options for a given application executed on a given target micro-architecture. One of the main advantages of FACSE is the fact that both of the two aspects (COSE and DSE) are taken into account simultaneously and in the same framework. The efficiency of the solution found in DSE is evaluated in terms of execution time and power consumption. Other metrics such as code size (for COSE) or die area (for DSE), will be considered in the future.

To achieve these objectives, FACSE is based on the use of the two following tools: 1) Different techniques for a fast evaluation of application performance and power consumption on a target micro-architecture. 2) Meta-heuristics for solving the COSE and the DSE problems in order to find a "close to optimal" micro-architecture configuration and a "close to optimal" compilation options in a reasonable time.

These two tools correspond to the two components of FACSE depicted in figure 1, i.e. a performance and power consumption estimator in one side and a configuration exploration method in the other side. In the performance and power estimation phase, the objective is the generation, in a relatively short time delay, of different statistics (execution time and the total power consumption). These statistics measure the adequacy between: the architecture configuration, the compiler option configuration and to the applications under test.

In the second phase, the statistics are recovered to generate eventually a new architecture configuration for DSE and/or a new compilation transformations configuration for COSE. At this stage, we use meta-heuristics (such as genetic algorithms and taboo search) to forward the exploration to exploration areas containing interesting solutions. The objective here is to reduce the set of evaluated configurations in DSE and COSE, with out degrading the search accuracy

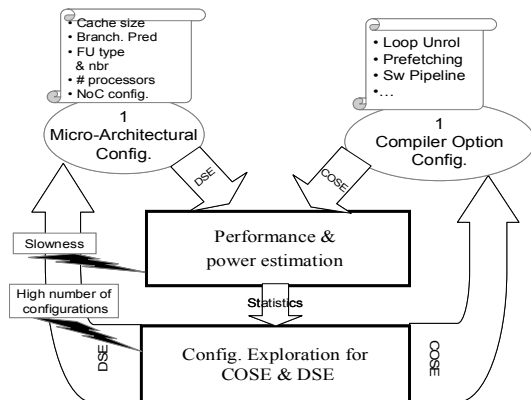


Fig. 1: FACSE general overview

III. DESIGN SPACE EXPLORATION (DSE) FOR EMBEDDED PROCESSOR DESIGN

In this section, we will present the gain when applying FACSE in an Intel Xscale core based embedded system design. Even though, the use of FACSE is interesting for any type of processor (Itanium, IA-32, Multicores, Multithreads, etc.), its use in a DSE for embedded system processors is important for several reasons.

As these embedded systems increases in complexity, fast and early simulation and performance estimation methods like those proposed in FACSE are necessary since the first design flow steps. These methods tend to guide the design towards efficient choices, much before the embedded system physical realization. Thus important savings at execution time, power consumption and system cost can be obtained. However, to be relevant, the DSE process must have the ability to evaluate a large number of design alternatives in a small time period all with the guaranty of a reasonable accuracy.

In the FACSE framework, three methods for rapid performance and power consumption evaluation in embedded system design have been evaluated. These methods are intended to offer both acceleration and accuracy at the cycle-accurate simulation level.

The basic idea in all three is to avoid performing the entire simulation of an application on a slow cycle accurate simulator, by using several rapid partial simulators (or profilers) instead. Because these profilers evaluate only one specific aspect of the architecture, such as cache configuration, their execution times are shorter. The results of these rapid simulations are then combined and adjusted to estimate the application's performance and power consumption. Thus, exploring the configuration space of one specific parameter (cache exploration, for instance) does not require executing the entire set of simulators but only the simulator concerned by the parameter (the cache simulator, for instance), which leads to an even higher simulation acceleration factor.

The first method uses statistical simulation (SS) to generate a short synthetic trace from the application profile. It offers a relative high simulation speed up. The second method is based on analytical modeling (AM) of performance and power consumption. Using this method, estimation errors are reduced by the usage of Detailed Simulation (AMDS) on an ideal configuration (zero cache and branch prediction misses) and by estimating elementary penalties of cache and branch predictor misses. Compared to the previous method, this method offers a better performance and power consumption accuracy but with limited acceleration factor. Finally, the third method represents a hybridization of the previous two methods and offers a good compromise between simulation speed up and performance and power estimation accuracy. This method accesses to both analytical modeling and statistical simulation (AMSS). It employs statistical simulation to accelerate the first phase in AMDS. Experiments with these three methods for some benchmark in MediaBench and Mibench have shown that a simulation accelerating factor up 289 can be obtained. For

these benchmarks, the accuracy errors are smaller than 3.8% in average and 8% in the worst case [Niar06].

IV. COSE FOR VLIW ARCHITECTURE

In VLIW architectures, such as the Intel Itanium® architecture or the Texas Instruments C6000 DSP's, high performances are obtained by a better exploitation of the available instruction-level parallelism (ILP) through the compiler [IntelUserGd].

Usually, the compilation process consists of several modules using efficient and complex techniques of code transformation and optimization. Loop unrolling and software pipelining are examples of the possible optimizations [Allen07]. The effect of these optimizations is not always obvious. Although, when individual application of these optimizations improves the execution speed, the combination of some of them produces negative effects on the global performance.

In traditional compilers, heuristics are used to predict the impact of every possible optimization and so to decide if this optimization must be applied or not. As it is impossible to obtain the best optimization sequence for every application, the use of predictive heuristic determines the best sequence of optimizations in average. This approach is interesting but the optimizations remain suboptimal for most applications.

To avoid this drawback, some researchers recommended the utilization of iterative compilation [O'Boyle02]. This method consists in evaluating several configurations and to choose the best one for future execution. Thus, instead of using "a priori" selected compilation options, iterative compilation finds out the best options by evaluating several configurations. As such, for commercial and high performance applications this process requires several days or even several weeks. In FACSE, this second solution is used. Nevertheless, in order to reduce the number of configuration to explore, different methods are used to guide the search towards interesting solution area in the exploration space. By consequent, this project aims to develop new heuristics [Osman02] to approximate the optimal sequence of code transformations.

As shown in figure 2, our COSE framework has three components. The first component is the « best configuration search heuristic ». This bloc has as function to determine the sub-space of interesting configurations to evaluate. For this purpose several meta-heuristic are being adapted and implemented for COSE. Each candidate configuration produced by this component is presented by a vector of binary values representing the setting of the different compilation options. The second component is the compiler, which receives a configuration candidate and the application, and generates the corresponding binary code. Finally, the performance evaluator realizes the evaluation of the "fitness" of the received binary code and sends the evaluation to the search heuristic for a new possible iteration. We will detail in the following the structure of this component. Our proposed method for performance evaluation within COSE for FACSE is represented by two phases:

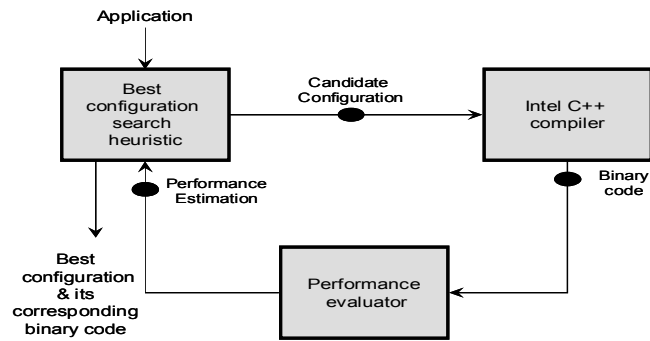


Fig 2: Functional representation of COSE in FACSE

The first phase consists in executing the whole application in order to determine its dynamic aspect. Statistics that are obtained in this phase are called global statistics and must be usable by all the following static estimations. For this reason, all the global statistics concern the source code level and are only dependent on the input data. To achieve this objective, source code instrumentation is performed for measuring the execution occurrences of each instruction. These occurrences are used during the static analysis (second phase). A trace of executed instructions is also generated.

The second phase is divided into two steps and is realized for each COSE configuration. The first step links source code blocks to binary code blocks while the second step consists in obtaining static statistics (called specific statistics) of the binary code and combining them with global statistic. At the end of the two phases, the performance estimation is computed following this formula:

$$CPU_CYCLES = WORK_CYCLES + BR_STALL + ICACHE_STALL + DCACHE_STALL$$

In this formula, *CPU_CYCLES* corresponds to the total number of cycles to execute the application and *WORK_CYCLES* is the number of cycles to execute the application with a perfect branch predictor and a perfect cache. *BR_STALL* represents the time penalties due to branch miss-predictions while *ICACHE_STALL* and *DATA_STALL* correspond to time penalty due to instruction and data cache misses. All these penalties are expressed in processor cycles.

The calculation of *WORK_CYCLES* is obtained without execution. In the case of perfect caches and perfect branch prediction, performance depends only on instruction parallelism and producer-consumer instruction distances. The static instruction scheduling used in VLIW architectures (such as Intel Itanium and TI C6000 DSP's) facilitates the estimation.

BR_STALL, *ICACHE_STALL* and *DATA_STALL* are calculated by a rapid execution of the trace generated in the first phase on a branch and cache simulators. Here the source code trace is analyzed in order to replace source code instructions by basic blocks of assembly instructions.

Figure 3 shows the capacity of the method to find the best configuration for 6 benchmarks taken from the SPEC CPU2000 and the Mediabench benchmarks. We used the Itanium2 platform and the intel Icc compiler with 6 different

compilation options giving a total space size of 216 different configurations.

For each benchmark, the first histogram (estimation error) represents the gap between the real best configuration execution time (RBET) found by iterative compilation and the execution time of the best configuration estimated by our method. The second histogram (Average error) represents the gap between RBET and the average real execution time for all the configurations. The third histogram (Maximum error) represents the gap between RBET and the real worst configuration execution time.

V. DEVELOPING METAHEURISTIC FOR DSE

We design a new meta-heuristic for DSE considering 15 different processor architectural parameters. These parameters concern instruction and data caches (total size, associativity, bloc size), number of rename registers, number of integer and floating point functional units, structure of branch predictor, bandwidth and number of memory ports, instruction issue and fetch widths, .etc. Each parameter can take between 3 and 7 possible values and are related to the simple scalar ARM processor simulator [Burger97]. They lead to a design space of $1.49 \cdot 10^9$ different configurations. As we are interested by reducing both execution time and power consumption, the purpose here is to find pareto-optimal solutions. The first phase of the meta-heuristic consists in eliminating unfeasible solutions by applying the following micro-architectural constraints such as [Burger97]: the instruction issue width must be smaller or equal to the instruction fetch queue size.

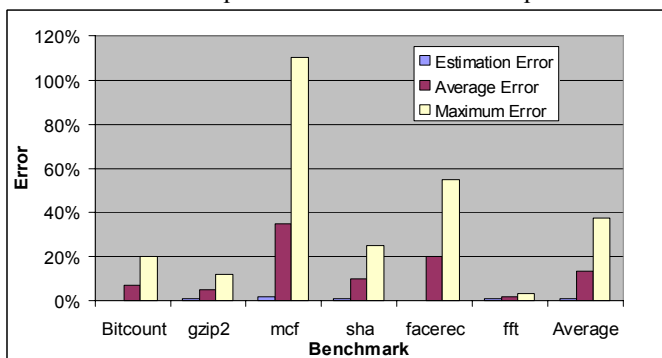


Fig 3: Performance estimation accuracy of the developed model for COSE

This filtering reduces the design space to $1.33 \cdot 10^6$ configurations. In the second phase, a dependency graph [Givargis01] has been used to perform two explorations using two phases: 1) A local search for the instruction and data cache parameters and 2) A genetic algorithm to explore the rest of architectural parameters.

The local search allowed finding 4 among 5 pareto optimal cache configurations. Only 43 different cache configurations have been simulated. On the other side, the genetic algorithm demonstrated a rapid convergence and only 1128 configurations have been simulated. Using the mibench suite, our hybrid meta-heuristic allows the localization of almost all pareto optimal solutions and needs only to explore

$1128+43=1171$ configurations among $1.33 \cdot 10^6$. This corresponds to a gain of 99.99997%.

VI. CONCLUSION AND PERSPECTIVES

The FACSE presented here, is a very ambitious project. Its concerns several domains: processor micro-architecture, compilation techniques, code optimization, and operational research. The contributions of this project are numerous.

The tool that we propose to achieve COSE in the FACSE project can easily be integrated into different application development flows. COSE using FACSE can be useful in two cases. In the first case, the application developer lacks of time and/or sufficient knowledge on the processor micro-architecture. In this case our tool will be integrated directly in the compiler and permits to obtain code with good performance. In the second case, the developer will simplify the work of the detailed analysis that he projects to make afterward. In this case the use of our tool can be considered as a preliminary phase for a detailed profiling phase (using for example VTune).

The utilization of the tools that we propose to develop for the DSE in FACSE is also very useful at different levels. The first application area concerns the design phase of future processors. The second application area concerns the utilization of these tools in platform customization such as in Application Specific Instruction set Processor (ASIP).

ACKNOWLEDGMENT

The authors would like to thank Prof. Koen De Bosschere, Dr. Lieven Eeckhout (Ghent University, Belgium) and Dr. Albert Cohen (INRIA-Futurs Paris) for their precious comments and suggestions. We would also like to thank Intel Corporation, for the support given to this project.

VII. REFERENCES

- [Allen07] R. Allen, K. Kennedy, Optimizing Compilers for Modern Architectures, 2007, Elsevier Pub.
- [Burger97] D.Buger, T.Austin, The SimpleScalar Tool Set, Version 2.0, University of Wisconsin-Madison, CS dep. Tech. Report #1342, 1997.
- [Givargis01] T. Givargis, et Al: System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip, ICCAD, 2001.
- [Gonzales00] R.E.Gonzales: Xtensa; A Configurable and Extensible Oricessor. IEEE Micro, March/April 2000.
- [Osman01] I.Osman and J.Kelly "Meta-Heuristics Theory and Applications", Kluwer Academic Pub. 2001.
- [IntelUserGd] Intel Corporation, Intel C++ compiler for Windows systems - User's Guide. 2004 (<http://www.intel.com/software/products/compilers/>).
- [Kahl05] J. A. Kahle, et Al. Introduction to the Cell multiprocessor IBM, Journal of research and development, Vol. 49 Number 4/5 2005.
- [Levy05] M.Levy, Evaluating Digital Entertainment System Performance IEEE Computer journal July-August 2005
- [Martin99] G.Martin et Al, Surviving the SOC Revolution - A Guide to Platform-Based Design, Springer, November 1999.
- [Mei04] B. Mei, S.Vernalde, D. Verkest, R.Lauwereins, Design Methodology for a Tightly Coupled VLIW/Reconfigurable. DATE 2004.
- [Niar06] S.Niar, N.Inglart, Rapid Performance and Power Consumption Estimation Methods for Embedded System Design, 17th IEEE International Workshop on Rapid System Prototyping, June 2006.
- [O'Boyle02] M.F.P. O'Boyle, P.M.W.Knijnenberg Integrating Loop and Data Transformations for Global Optimisation, Journal of Parallel and Distributed Computing, 2002.