Applying Multi-Core Model Checking to Hardware-Software Partitioning in Embedded Systems

Alessandro Trindade, Hussama Ismail, and Lucas Cordeiro
Federal University of Amazonas - Manaus, Amazonas, Brazil
{alessandro.b.trindade, hussamaismail}@gmail.com, lucascordeiro@ufam.edu.br

***Abstract*—We present an alternative approach to solve the hardware and software partitioning problem, which uses Bounded Model Checking (BMC) based on Satisfiability Modulo Theories (SMT) in conjunction with a multi-core support using Open Multi-Processing. The multi-core approach allows initializing many verification instances based on processors cores numbers available to the model checker. Each instance checks for a different optimum value until the optimization problem is satisfied. The goal is to show that multi-core model-checking techniques can be effective, in particular cases, to find the optimal solution of the hardware-software partitioning problem. We compare the experimental results of our proposed approach with conventional algorithms.**

Keywords- hardware-software co-design; hardware-software partitioning; optimization; model checking; multi-core; OpenMP

I. Introduction

With the strong development of embedded systems, the design phase plays an important role nowadays. The partitioning decision process, which deals with decisions upon which parts of the application have to be designed in hardware (HW) and which one in software (SW), must be supported by any well-structured methodology. If not, this leads to a number of issues that affects the overall development proces. Starting at the 1990s, intensive research was performed, and several approaches proposed, as shown in [1] and [2]. In [3] was shown that it is possible to use Bounded Model Checking (BMC) based on Satisfiability Modulo Theories (SMT) to perform HW-SW partitioning in embedded systems. The present work improves the SMT-based verification method. Here, we exploit the availability of multi-core processors. In particular, a multi-core SMT-based BMC method is applied to the HW-SW partitioning and then is compared to the results with classical integer linear programming (ILP) and genetic algorithm (GA) using a multi-core tool as well. To the best of our knowledge, this is the first work to use a multi-core SMT-based verification to solve a HW-SW partitioning problem in embedded systems. We implement our ideas with the Efficient SMT-based Bounded Model Checker (ESBMC) [9].

II. Background

A. Optimization

Optimization is the act of obtaining the optimal solution under given circumstances [5]. In engineering system, the ultimate goal is either to minimize the effort required or to maximize the desired benefit. Eq. (1) shows a typical linear programming problem, where A and b are vectors or matrixes that describe the constraints.

$$\min f^T x \text{ such that } \begin{cases} A.x \leq b, \\ Aeq.x = beq, \\ x \geq 0. \end{cases} \qquad (1)$$

If the optimization problem is complex, some heuristics can be used to solve the same problem faster [5]. The drawback is that the found solution may not be the exact.

B. Bounded Model Checking with ESBMC

Bounded model checking (BMC) combines model checking with satisfiability solving. And for some types of problems, it offers large performance improvements over previous approaches, as shown in [6]. BMC checks the negation of a given property at a given depth: given a transition system M, a property ϕ, and a bound k, BMC unrolls the system k times and translates it into a verification condition (VC) φ such that φ is satisfiable if and only if ϕ has a counterexample of depth k or less [6]. To cope with increasing software complexity, SMT solvers can be used as back-ends for solving the generated VCs, as shown in [7], and [8]. Two directives in C/C++ that can guide a model checker to solve an optimization problem: ASSUME ensures the compliance of constraints (software costs); and ASSERT controls the code violation (minimum hardware cost).

C. Multi-core ESBMC with OpenMP

Although recent CPUs have a modern multi-core architecture, ESBMC verification runs are still performed only in a single-core. Fig.1 shows the ESBMC architecture, which consists of the C/C++ parser, GOTO Program, GOTO Symex, and SMT solver [10]. ESBMC compiles the C/C++ code into equivalent GOTO-programs (*i.e.*, control-flow graphs) using a gcc-compliant style. The GOTO-programs can be processed by the symbolic execution engine, called GOTO Symex, where two recursive functions compute constraints (C) and properties (P); finally it generates two sets of equations (*i.e.*, $C \wedge \neg P$) which are checked by an SMT solver. The main factor for ESBMC to use only a single-core relies on its back-end (*i.e.*, SMT Solver).

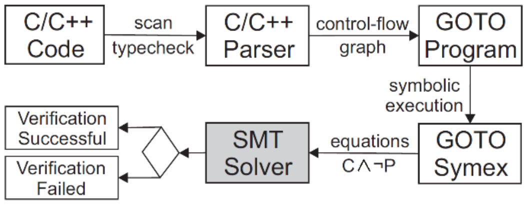

Figure 1. ESBMC architecture

To optimize the CPU utilization without modifying the underlying SMT Solver, the Open Multi-Processing (OpenMP) library [15] is used as a front-end for ESBMC. In OpenMP, the implementation is based on the fork-join model. The main thread executes the sequential parts of the program; if a parallel region is encountered, then it forks a team of worker threads. After the parallel region finishes, then the main procedure gets back to the single-threaded execution mode [4]. Fig. 2 shows our approach called Multi-core ESBMC (ESBMC-MC).

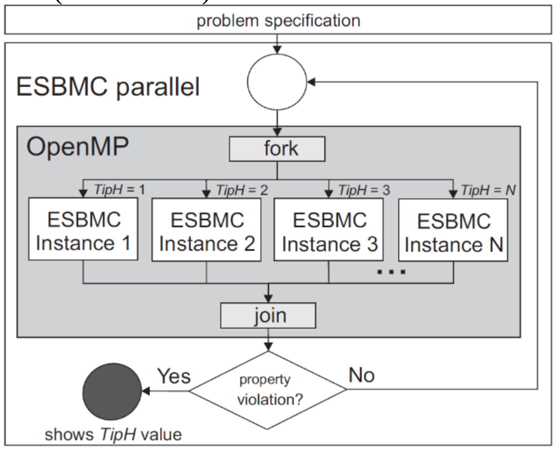

Figure 2. Multi-core ESBMC Approach

ESBMC-MC obtains the problem specification represented by a C program, which is violated when the correct optimum value ($TipH$) parameter is reached; ESBMC-MC starts a parallel region with N different instances, based on the number of available processing cores. All these ESBMC instances run independently of each other, as shown in Fig. 2; there is no shared-memory or message-passing. In particular, different threads are managed by OpenMP using different $TipH$ values as condition. After executing N instances, if there is no code violation, then ESBMC-MC starts N new instances. During the parallel region execution, if a violation is found, then it presents a counterexample. If all threads of the batch processing are terminated, then ESBMC-MC finishes its execution.

III. Mathematical Modeling

The mathematical modeling was taken from [1], [2].

A. *Informal Model (or Assumptions)*

First, there is only one software context, *i.e.*, there is just one general-purpose processor, and there is only one hardware context. The components of the system must be mapped to either one of these two contexts. Second, the software implementation of a component is associated with a software cost (running time). Third, the hardware implementation of a component has a hardware cost (area, heat dissipation, or energy consumption). Fourth, based on the premise that hardware is significantly faster than software, the running time of the components in hardware is considered as zero. Finally, if two components are mapped to the same context, then there is no overhead of communication between them. The consequence of these assumptions is that scheduling does not need to be addressed. The focus is only on the partitioning problem. That configuration describes the first-generation co-design, where the focus is on bipartitioning [11].

B. *Formal Model*

A directed simple graph $G = (V, E)$, called the task graph of the system, is necessary. The vertices $V = \{v_1, v_2, \dots, v_n\}$ represent the nodes that are the components of the system that will be partitioned. The edges (E) represent communication between components. Each node v_i has a cost $h(v_i)$ (or h_i) of hardware (if implemented in hardware) and a cost $s(v_i)$ (or s_i) of software (if implemented in software). $c(v_i, v_j)$ represents the communication cost between v_i and v_j if they are implemented in different contexts (hardware or software). Based on [1], P is called a hardware-software partition if it is a bipartition of V: $P = (V_H, V_S)$, where $V_H \cup V_S = V$ and $V_H \cap V_S = \emptyset$. The crossing edges are $E_P = \{(v_i, v_j) \colon v_i \in V_S, v_j \in V_H \text{ or } v_i \in V_H, v_j \in V_S\}$. The hardware cost of P is given by Eq. (2), and the software cost of P is given by Eq. (3):

$$H_P = \sum_{v_i \in V_H} h_i \tag{2}$$

$$S_P = \sum_{v_i \in V_S} s_i + \sum_{(v_i, v_j) \in E_P} c(v_i, v_j) \tag{3}$$

In this paper, the focus is on the case that S_0 is given, *i.e.*, to find a P HW-SW partitioning so that $S_P \leq S_0$ and H_P is minimal (system with hard real-time constraints). So, based on Eq. (1) and Eq. (3) the optimization problem's restrictions can be reformulated as: $s(1 - x) + c|Ex| \leq S_0$, where x is the decision variable. Concerning the complexity of this problem, reference [1] demonstrates that it is NP-Hard.

IV. Analysis of the Partitioning Problem

A. *Using ILP-based and Genetic Algorithms*

The ILP and GA were taken from [3]. Both use slack variables to represent constraints and to use commercial tools. However, GA had improvements to increase the solution accuracy without producing timeout. The tuning was performed by empirical tests and resulted in changing of three parameters of MATLAB function *ga* [12]: the population size was set from 300 to 500, the Elite count changed from 2 to 50, and the number of Generations changed from 100* *NumberOfVariables* (default) to 75.

B. *Using ESBMC*

ESBMC pseudocode shows the algorithm with the same restrictions and conditions placed on ILP and GA. Two values must be controlled to obtain the results and to perform the optimization. One is the initial software cost, as defined in Section III.B. The other is the halting condition (code violation) that stops the algorithm. The ESBMC algorithm starts with the declarations of hardware, software, and communication costs. S_0 must also be defined, as the transposed incidence matrix and the identity matrix. Here, the matrices A and b are generated. At that point, the ESBMC algorithm starts to differ from ILP and GA presented in [3]. It is possible to tell ESBMC with which type of values the variables are tested. There is a declaration to populate all decision variables x with non-deterministic Boolean values. Those values that change for each test will generate a possible solution and obey the restrictions. If this is achieved, then a feasible solution is found and the ASSUME directive ensures the compliance of constrains ($i.e., A.x \leq b$).

A loop controls the cost of hardware hint, starting with zero and reaching the maximum value considering the case where all nodes are partitioned to hardware. To every test performed, the hardware hint is compared to the feasible solution. This is accomplished by an ASSERT statement at the end of the algorithm (the halt condition or true-false statement). The ASSERT statement tests the objective function (hardware cost), and will stop if the hardware cost found is lower than or equal to the optimal solution. However, if ASSERT returns a TRUE condition (hardware cost is higher than the optimal solution), then the model-checking algorithm restarts and a new possible solution is generated and tested until the ASSERT generates a FALSE condition. When the FALSE condition happens, the execution code is aborted and ESBMC presents the counterexample that caused the condition to be broken. That is the point in which the solution is presented (minimum HW cost). In the ESBMC algorithm, it is not necessary to add slack variables because the modulus operation is kept.

ESBMC Pseudocode

```
01 Initialize variables
02 Declare number of nodes and edges
03 Declare hardware cost of each node as array ($h$)
04 Declare software cost of each node as array ($s$)
05 Declare communication cost of each edge ($c$)
06 Declare the initial software cost ($S_0$)
07 Declare transposed incidence matrix graph $G$ ($E$)
08 Define the solutions variables ($x_i$) as Boolean
09 For $TipH = 0$ to $Hmax$ do {
10   Populate $x_i$ with nondeterministic/test values
11   Calculate $s(1-x) + c * |Ex|$ and store at $variable$
12   Requirement insured by $ASSUME$ ($variable \leq S_0$)
13   Calculate $H_P$ cost based on value tested of $x_i$
14   Violation check with ASSERT ($H_P > TipH$)
15 }
```

In the multi-core ESBMC (ESBMC-MC) algorithm, the only difference is the fact that the value of $TipH$ and its range is not declared in the algorithm. The approach is invoked as follows:

esbmc-parallel $<filename.c>$ $<hmin\_value>$ $<Hmax>$

Where $< filename.c >$ is the optimization problem described in ANSI-C format, $<hmin\_value>$ is the minimum (zero to HW-SW partitioning problem) and $<Hmax>$ is the maximum hardware cost for the specified problem. Therefore, the algorithm starts N different instances of ESBMC using the different optimization values, in ascending order, for $Hmax$ in order to find a violation. If all instances finish and no violation is found, then multi-core ESBMC starts new N instances. When a violation is found, it reports time and hardware cost. If multi-core ESBMC tests all the possibilities for the hardware cost and has not found a violation, then it reports: "Violation not found".

V. EXPERIMENTAL EVALUATION

Testing platform: desktop with 15GB of RAM and i7 (8-cores) from Intel with clock of 3.40 GHz; ESBMC 1.24; 64-bit Ubuntu 14.04.1 LTS; version 2.0.1 of Boolector SMT-solver [16]; MATLAB R2013a from MathWorks with Parallel Computing Toolbox was used [12] for the ILP and GA formulations. The ESBMC algorithms were implemented in C++ [1]. Each time was measured 3 times and average taken. Statistical confidence reached 92%, based on standard deviation, and confidence interval for every benchmark. A time out condition (TO) is reached when the running time is longer than 7,200 seconds. A memory out (MO) occurs when the tool reaches 15GB of memory. TABLE I. lists the benchmarks[1]. Those were the same used by related work [1], [2], and [14].

TABLE I. DESCRIPTION OF BENCHMARKS

Name	Nodes	Edges	Description
CRC32	25	32	32-bit cyclic redundancy check [13]
Patricia Insert	21	48	Routine to insert values [13]
Dijkstra	26	69	Computer shortest paths in a graph [13]
Clustering	150	331	Image segmentation algorithm in a medical application
RC6	329	448	RC6 cryptography graph
Fuzzy	261	422	Clustering algorithm based on fuzzy logic
Mars	417	600	MARS cipher from IBM

The nodes correspond to high-level language instructions. SW and communication costs are time dimensional, and HW costs represent the occupied area. The first three benchmarks came from MiBench [13]. Clustering and Fuzzy benchmarks were designed from [2]. From the same authors, very complex benchmarks to test the limits of the applicability of techniques were used (RC6 and Mars).

TABLE II. shows that ILP produces the best results, with a limit of 329 nodes or less. GA was able to solve all benchmarks, but the error from the exact solution varies from -38% to 29%. ESBMC-MC had a better performance than that of pure ESBMC. The relative speedup obtained ranged from 14 to 54. Until the number of 150 nodes is reached, the ESBMC technique, mainly ESBMC-MC, has shown itself to be a good choice to solve HW-SW partitioning. This is because the exact solution was found and the execution time was mostly closer to ILP. Pure ESBMC algorithm has the

[1] Available at: http://www.esbmc.org/benchmarks/

drawback of creating more complex problems, because it increases the states produced.

TABLE II. RESULTS OF THE BENCHMARKS

		CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
	Nodes	25	21	26	150	329	261	417
	Edges	32	48	69	331	448	422	600
	S_0	20	10	20	50	600	4578	300
Exact Solution	H_P	15	47	31	241	692	13820	876
	S_P	19	4	19	46	533	4231	297
ILP	Time (s)	2	1	2	649	1806	TO	5429
	H_P	15	47	31	241	692	-	876
GA	Time (s)	7	7	9	340	2050	1372	5000
	Error %	13	0	29	2	-7	-38	-28
ESBMC	Time (s)	30	314	325	MO	MO	MO	MO
	H_P	15	47	31	-	-	-	-
ESBMC-MC	Time (s)	2	6	7	1609	TO	TO	TO
	H_P	15	47	31	241	-	-	-
ESBMC Relative speedup		14	54	47	-	-	-	-

Legend: TO = Time out and MO = memory out

ESBMC-MC was unable to solve some benchmarks. Pure ESBMC had even a worse performance. This is a clear indication that the prune method adopted by ILP is more efficient than the adopted by ESBMC solver.

VI. RELATED WORK

Since the 2000s, three paths have been tracked to solve the optimization of HW-SW partitioning, *i.e.*, to find the exact solution [2], to use heuristics to speed up performance time [1], and hybrid ones [14]. This paper belongs to the group that founds the exact solution. In terms of SMT-based verification, in [10] was presented a bounded model checker for C++ programs, which is an evolution of dealing with C programs and [9] uses the ESBMC model checker for embedded ANSI-C software. In [3] it was proven that it is possible to use ESBMC to solve HW-SW partitioning. There are related studies focused on decreasing the verification time of model checkers [17], and modifications of internal search engines to support parallelism [18], but there is the need for initiatives related to parallel SMT solvers [19]. Recently, SMT solver Z3 was extended to solve optimization problems [20].

VII. CONCLUSIONS

None of tools is indicated to partition problems with more than 400 nodes. If we consider less than 400 nodes, so ILP is the best tool. If the problem to be solved has 150 nodes or less, then ESBMC represents a feasible alternative. ESBMC has a BSD-style license and can be downloaded and used free. Concerning the two versions of ESBMC, it is possible to conclude that Multi-core ESBMC had better performance results than pure ESBMC (speedup reached 47 times).

ACKNOWLEDGEMENTS

Part of the results were sponsored by Samsung Eletrônica da Amazônia Ltda ruled by Brazilian federal law 8.387/91.

REFERENCES

[1] Arató, P., Juhász, S., Mann, Z.A., Orbán, A., Papp, D.: Hardware/software partitioning in embedded system design. In: WISP, pp. 192-202, 2003

[2] Mann, Z.A., Orbán, A., Arató, P.: Finding optimal hardware/software partitions. In: FMSD, vol. 31, pp. 241-263, 2007

[3] Trindade, A., Cordeiro. L.: Applying SMT-based verification to hardware/software partitioning in embedded systems. In: DAES, 2015

[4] Wu, M., Wu, W., Tai, N., Zhao, H., Fan, J., Yuan, N.: Research on OpenMP model of the parallel programming technology for homogeneous multicore DSP. In: ICSESS, pp. 921,924, 2014

[5] Rao, S.: Engineering Optimization: Theory and Practice. 4th edition. John Wiley & Sons, Hoboken, 2009

[6] Biere, A.: Bounded model checking. In: Biere, A., Heule, M., van Marren, H., Walsh, T. (org.) Handbook of Satisfiability, IOS Press, pp. 457–481, Amsterdam, 2009

[7] Armando, A., Mantovani, J., Platania, L.: Bounded model checking of software using SMT solvers instead of SAT solvers. In: STTT, Vol. 11, n. 1, pp. 69-83, 2009

[8] Ganai, M., Gupta, A.: Accelerating high-level bounded model checking. In: ICCAD, pp. 794–801, 2006

[9] Cordeiro, L., Fischer, B., Marques-Silva, J.: SMT-based bounded model checking for embedded ANSI-C software. In: IEEE TSE, vol. 38, ed. 4, pp. 957-974, 2012

[10] Ramalho, M., Freitas, M., Souza, F., Marques, H., Cordeiro, L.: SMT-Based Bounded Model Checking of C++ Programs. In: ECBS, pp. 147-156, IEEE, 2013

[11] Teich, J.: Hardware/Software Codesign: The Past, the Present, and Predicting the Future. In: Proc. of the IEEE, vol. 100, pp. 1411-1430, 2012

[12] The MathWorks, Inc: MATLAB (version R2013a). Natick, MA, 2013

[13] Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., Brown, R.: MiBench: a free, commercially representative embedded benchmark suite. In: WWC, pp. 3-14, 2001

[14] Arató, P., Mann, Z.A., Orbán, A.: Algorithmic aspects of hardware/software partitioning. In: ACM TODAES, vol. 10, pp. 136–156, 2005

[15] Dagum, L., Menon, R.: OpenMP: an industry-standard API for shared-memory programming. In: Computational Science & Engineering, vol. 5, issue 1, pp. 46-55, 1998

[16] Brummayer, R., Biere, A.: Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: TACAS, LNCS 5505, pp. 174–177, 2009

[17] Holzmann, G.J., Joshi, R., Groce, A.: Swarm Verification Techniques. In: IEEE TSE, vol. 37, issue 6, pp. 845, 857, 2011

[18] Holzmann, G.: Parallelizing the spin model checker. In: SPIN, LNCS 7385, pp. 155-171, 2012

[19] Wintersteiger, C., Hamadi, Y., De Moura, L.: A Concurrent Portfolio Approach to SMT Solving. In: CAV, LNCS 5643, pp. 715-720, 2009

[20] Bjørner, N., Phan, A-D., Fleckenstein, L.: νZ - An Optimizing SMT Solver. In: TACAS, LNCS 9035, pp. 194-199, 2015