femington

Int. J. Intell. Syst. Data Sci. (2026) 1(3):1–13
Article ID: IJISDS-2026-022

Multicore SMT-BMC for Embedded Co-Design: MaxSMT-Aided Hardware/Software Partitioning at Scale

Monisha Rengaraj
monisha98rengaraj@gmail.com
Independent Researcher

Abstract

This paper targets hardware/software partitioning in embedded systems as a first-class design automation problem, presenting a multicore Bounded Model Checking (BMC) workflow that accelerates exact optimization under real system constraints. The proposed approach parallelizes Satisfiability Modulo Theories (SMT)-based verification with Open Multi-Processing (OpenMP) and augments it with a Maximum Satisfiability Modulo Theories (MaxSMT) backend (νZ) to minimize hardware cost while satisfying software and communication constraints on task-graph models used in co-design. A family of implementations, including the sequential Efficient SMT-Based Context-Bounded Model Checker (ESBMC), multicore ESBMC with sequential, worker, and binary-search controllers, and an ESBMC–νZ integration, is developed to exploit contemporary multicore processors without modifying the underlying solver backends. Comparative evaluation against Integer Linear Programming (ILP) and Genetic Algorithm (GA) approaches across standard embedded-system benchmarks (e.g., CRC32, Patricia, Dijkstra, Clustering, RC6, Fuzzy, and Mars) quantifies runtime, optimality, and scalability. The results demonstrate that the MaxSMT-based ESBMC approach achieves superior performance on small-to-medium problem instances, whereas the multicore ESBMC with a parallel binary-search controller (ESBMC-PB) provides substantial speedups for larger task graphs. These findings offer practical guidance for hardware/software co-design workflows: employ ESBMC–νZ when exact optimality and rapid turnaround are priorities, utilize multicore ESBMC-PB for large design spaces, and reserve GA for scenarios where approximate solutions are acceptable. Overall, the proposed contributions establish a practical Computer Engineering (CE) design-automation pipeline that effectively integrates formal optimization with parallel verification to address hardware/software trade-offs in embedded system architectures.

Keywords

• Hardware–Software Co-Design • Hardware–Software Partitioning • Optimization • Model Checking • Multi-Core Computing • OpenMP • SMT • MaxSMT

1. Introduction

Embedded systems development faces increasing complexity as applications demand higher performance under stringent power, area, and timing constraints. The hardware-software (HW-SW) partitioning problem represents a critical phase in co-design methodologies, determining which system components execute in hardware versus software to optimize overall system metrics. Traditional approaches often rely on manual partitioning decisions, leading to suboptimal implementations, design iterations, and extended development cycles.

The HW-SW partitioning problem is fundamentally an optimization challenge where designers must balance hardware implementation costs (area, power) against software execution times while considering communication overhead between partitioned components. Early research established foundational models and algorithms for this problem [1], but increasing system complexity demands more sophisticated approaches that can handle larger design spaces while guaranteeing solution optimality. Recent work has explored various optimization techniques, including immune algorithms [2], cooperative game theory [3], and swarm intelligence [4], each offering different trade-offs between solution quality and computational cost.

Bounded Model Checking (BMC) based on Satisfiability Modulo Theories (SMT) has emerged as a powerful verification technique that can be adapted for optimization problems. The Efficient SMT-Based Context-Bounded Model Checker (ESBMC) provides a robust framework for software verification that can be extended to solve HW-SW partitioning through constraint formulation and property checking. However, traditional single-core implementations face scalability limitations when applied to complex embedded systems with hundreds of components.

This paper presents a comprehensive multicore SMT-BMC approach that leverages modern processor architectures through OpenMP parallelization while integrating Maximum SMT (MaxSMT) capabilities via the νZ solver. Our contributions address the scalability challenges in exact HW-SW partitioning through three parallel strategies: sequential search (ESBMC-SS), worker-based parallel search (ESBMC-PS), and binary search (ESBMC-PB), alongside a MaxSMT integration (ESBMC-νZ) that reformulates the optimization problem within the SMT solver itself.

Experimental evaluation demonstrates that no single approach dominates across all problem sizes. ESBMC-νZ achieves superior performance on small to medium benchmarks, while multicore ESBMC implementations provide significant speedups for larger instances. Compared to established Integer Linear Programming (ILP) and Genetic Algorithm (GA) approaches, our methods offer competitive performance with guaranteed optimality, addressing a critical need in embedded system design automation.

Recent advances in co-design frameworks highlight the growing importance of hardware-software co-design in modern AI and high-performance computing systems [5–7]. Additionally, the evolution of hardware description languages [8] and compiler toolchains [9] underscore the need for formal methods and verification in modern system design. The remainder of this paper is organized as follows: Section 2 provides background on optimization techniques and tools. Section 3 formalizes the HW-SW partitioning problem. Section 4 details our multicore SMT-BMC approaches. Section 5 presents experimental methodology and results. Section 6 discusses related work, and Section 7 concludes with future research directions.

2. Background and Related Tools

2.1 Optimization Fundamentals

Optimization represents the process of finding the best solution to a problem under specified constraints [1]. In HW-SW partitioning, the objective typically involves minimizing hardware cost while satisfying software performance constraints, or vice versa. Linear programming provides a mathematical framework for such optimization problems where both the objective function and constraints are linear functions of decision variables.

Integer Linear Programming (ILP) represents a specialized case where variables are restricted to integer values, making it suitable for binary partitioning decisions.

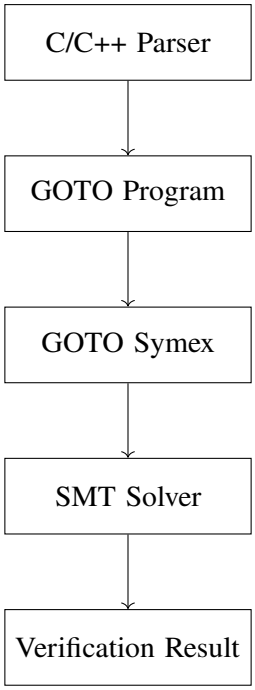

Figure 1: ESBMC architecture showing the verification pipeline from source code to SMT solving.

$$\min f^t x \quad \text{subject to} \quad \begin{cases} A \cdot x \leq b, \\ A_{eq} \cdot x = b_{eq}, \\ x \geq 0. \end{cases} \tag{1}$$

While ILP guarantees optimal solutions, computational complexity can become prohibitive for large problem instances. Heuristic approaches like Genetic Algorithms (GA) offer faster solutions but sacrifice optimality guarantees [4]. SMT-based verification provides an alternative pathway that maintains optimality while potentially offering better scalability through sophisticated reasoning techniques.

2.2 Bounded Model Checking with ESBMC

Bounded Model Checking (BMC) combines model checking with satisfiability solving to explore the state space of systems up to a specified depth [10]. Unlike traditional model checkers that may suffer from state explosion, BMC translates verification problems into satisfiability instances that can be efficiently solved by modern SAT and SMT solvers.

ESBMC implements SMT-based BMC for C/C++ programs, compiling source code into GOTO programs that undergo symbolic execution to generate verification conditions. The tool supports multiple SMT solvers including Z3, Boolector, MathSAT, CVC4, and Yices. For optimization problems, ESBMC utilizes ASSUME and ASSERT directives to encode constraints and objective functions, enabling the discovery of optimal solutions through property violation checks.

Various approaches have been proposed to improve verification scalability, including layered and parallelized model checking techniques [11], and scalable formal verification strategies have been surveyed for specific architectures such as RISC-V [12]. The connection between model checking and hardware generation has been explored through tools that bridge formal verification and hardware design [9]. Additionally, SMT proof checking frameworks [13] contribute to the reliability of verification outcomes.

2.3 OpenMP Parallel Programming

OpenMP provides a portable API for shared-memory parallel programming in C/C++ and Fortran. The fork-join execution model allows sequential programs to spawn parallel regions where multiple threads execute concurrently, synchronizing upon completion of parallel sections. This model aligns well with verification tasks that can be partitioned into independent subproblems.

The basic parallel for directive distributes loop iterations across available threads, while critical regions ensure mutual exclusion for shared resources. Our multicore ESBMC implementations leverage these constructs to parallelize the search for optimal partitioning solutions without modifying the underlying SMT solver backends.

2.4 Maximum SMT with νZ

The νZ tool extends the Z3 SMT solver with optimization capabilities through Maximum SMT (MaxSMT) [14]. MaxSMT extends traditional SMT solving by finding models that satisfy hard constraints while maximizing satisfaction of soft constraints, or directly optimizing objective functions. Key functions include:

- `maximize(T)` and `minimize(T)` for objective optimization
- `assert-soft F : weight n` for weighted soft constraints

νZ combines multiple techniques including Maximum Resiliency (MaxRes) for MaxSAT problems and simplex for linear arithmetic optimization. The tool reformulates SMT formulas with objectives into pseudo-Boolean optimization (PBO) problems, invoking appropriate solvers based on constraint types. This integration enables native optimization within the SMT solving process, potentially offering performance advantages over external optimization loops.

3. Hardware-Software Partitioning Formulation

3.1 Problem Definition

The HW-SW partitioning problem addresses the assignment of system components to hardware or software implementation domains to optimize system metrics under constraints [1]. We consider a first-generation co-design scenario with one general-purpose processor (software context) and one hardware accelerator (hardware context). Components mapped to hardware incur area costs but execute with negligible timing, while software components contribute to execution time. Communication costs arise when interconnected components are assigned to different domains.

This formulation assumes that scheduling can be neglected due to the single processor and instantaneous hardware execution. The focus remains purely on the partitioning decision, which suffices for many embedded systems where hardware acceleration targets specific computational kernels.

3.2 Mathematical Model

The partitioning problem is formalized using a directed task graph $G = (V, E)$, where vertices $V = \{V_1, V_2, \ldots, V_n\}$ represent system components and edges E represent communication dependencies. Each vertex V_i has associated hardware cost h_i (area, power) and software cost s_i (execution time). Edge (V_i, V_j) has communication cost $c(V_i, V_j)$ incurred when endpoints are in different domains.

A hardware-software partition is a bipartition $P = (V_h, V_s)$ where $V_h \cup V_s = V$ and $V_h \cap V_s = \emptyset$. The crossing edges $E_p = \{(V_i, V_j) : V_i \in V_s, V_j \in V_h \text{ or } V_i \in V_h, V_j \in V_s\}$ represent inter-domain communication. The total hardware cost is:

$$H_p = \sum_{V_i \in V_h} h_i, \tag{2}$$

and the total software cost (including communication) is:

$$S_p = \sum_{V_i \in V_s} s_i + \sum_{(V_i, V_j) \in E_p} c(V_i, V_j). \tag{3}$$

We focus on the optimization problem: given a software cost bound S_0, find partition P that minimizes H_p subject to $S_p \leq S_0$. This formulation addresses systems with real-time constraints where software execution must meet timing requirements. The problem is known to be NP-hard [3], justifying the use of sophisticated optimization techniques.

3.3 Constraint Formulation

Using decision variables $x_i \in \{0, 1\}$ where $x_i = 1$ indicates hardware implementation, the constraints can be reformulated as:

$$s(1 - x) + c|Ex| \leq S_0, \tag{4}$$

where s and h are vectors of software and hardware costs, c is the communication cost vector, and E is the incidence matrix. The objective function becomes:

$$\min \sum_{i=1}^{n} h_i x_i. \tag{5}$$

This formulation enables solution through various optimization techniques, including ILP, SMT-based verification, and MaxSMT, each with different computational characteristics and scalability properties.

4. Multicore SMT-BMC Approaches

4.1 Sequential ESBMC Implementation

The baseline sequential ESBMC approach formulates the HW-SW partitioning problem as a verification task. Algorithm 1 outlines the pseudocode, which declares hardware, software, and communication costs along with the software constraint S_0. Decision variables x_i are populated with non-deterministic Boolean values, generating potential solutions that satisfy the constraints through ASSUME directives.

The algorithm searches for the minimum hardware cost by iterating from zero to the maximum possible hardware cost (H_{max}). Each iteration tests whether a solution exists with hardware cost $TipH$, using the ASSERT directive to trigger a property violation when a solution is found. The counterexample generated upon violation provides the optimal partition.

Unlike ILP approaches, this method requires no slack variables, reducing problem complexity. However, verification time grows with the solution space, motivating parallel implementations.

Algorithm 1 Sequential ESBMC-Based HW/SW Partitioning

Input: Task graph $G(V, E)$; hardware costs **h**; software costs **s**; communication costs **c**; software-cost bound S_0

Output: Optimal hardware/software partition satisfying the software constraint

Initialize task graph parameters and cost vectors
Construct the transposed incidence matrix E
Define Boolean decision variables $x_i \in \{0, 1\}$ for each task
for $TipH \leftarrow 0$ **to** $H_{\max}$ **do**
    Generate nondeterministic assignments for all x_i
    Compute software cost

$$S = s(1 - x) + c|Ex|$$

    Assume $S \leq S_0$ ▷ *[r]Enforce software-cost constraint
    Compute hardware cost

$$H_p = hx$$

    Assert $H_p > TipH$ ▷ *[r]Continue search until optimum is found
return Optimal HW/SW partition

4.2 Multicore ESBMC with Sequential Search (ESBMC-SS)

ESBMC-SS leverages OpenMP to parallelize the sequential approach across available processor cores. As shown in Figure 2, the problem specification is processed by a controller that forks multiple ESBMC instances, each checking a different candidate hardware cost value ($TipH$).

Threads execute independently without communication, managed by OpenMP's thread lifecycle control. Each thread batch processes a range of $TipH$ values sequentially. If any thread finds a violation (solution), execution terminates and the result is returned. Otherwise, new batches are processed until the solution is found.

This approach utilizes idle cores but may suffer from load imbalance if solutions are unevenly distributed across the search space. The sequential nature of batch processing can still leave cores idle between batches.

4.3 Multicore ESBMC with Worker Threads (ESBMC-PS)

ESBMC-PS addresses the idle time limitation by implementing worker threads that continuously fetch new $TipH$ values as cores become available. As shown in Figure 3, a central controller manages a work queue, distributing tasks to worker threads that execute ESBMC instances.

Worker threads immediately request new values upon completion, minimizing processor idle time. This approach improves resource utilization but provides limited benefits when verification times are relatively uniform across different $TipH$ values, as is often the case in HW-SW partitioning.

4.4 Multicore ESBMC with Binary Search (ESBMC-PB)

ESBMC-PB implements a parallelized binary search to reduce the number of verification steps. The controller maintains intervals of candidate values and returns the median of the largest interval when a core becomes available. This approach dramatically reduces the search space compared to linear strategies.

Algorithm 2 shows the step calculation process. Workers continuously execute steps retrieved from the controller, which also checks if running steps remain necessary based on results from other threads. Unnecessary verifications are terminated early, saving computational resources.

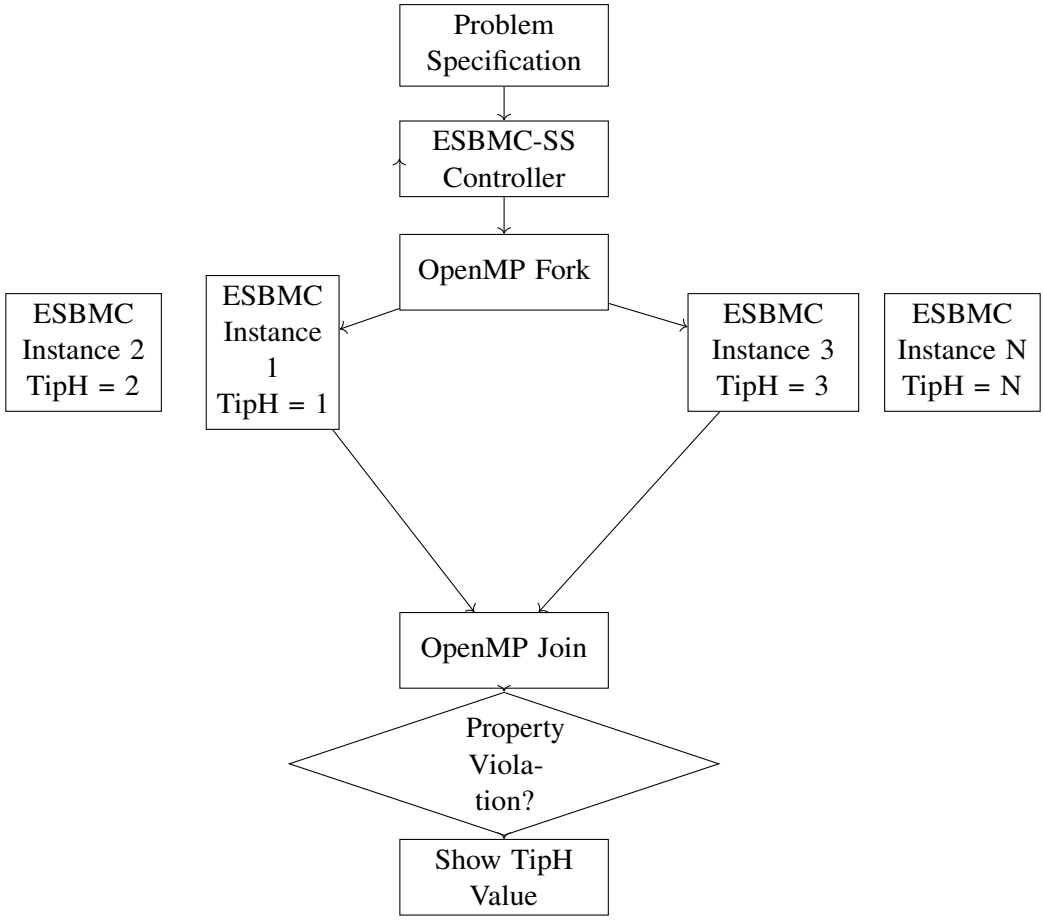

Figure 2: ESBMC-SS architecture with parallel instances checking different hardware cost values.

Algorithm 2 Binary Step Selection in ESBMC-PB

Input: Current set of search intervals (chunks)

Output: Next hardware-cost threshold ($median$)

$largestChunk \leftarrow -1$

foreach $chunk \in chunks$ **do**

  if $(chunk.right - chunk.left) > largestChunk$ **then**

    $largestChunk \leftarrow chunk.right - chunk.left \;\; largest \leftarrow chunk$

Remove $largest$ from $chunks$

Compute the midpoint

$$median = largest.left + \left\lfloor \frac{largest.right - largest.left}{2} \right\rfloor$$

if $median > 0$ **then**

  if $largest.right - largest.left > 1$ **then**

    Insert interval $[largest.left,\ median - 1]$ into $chunks$

  if $largest.right \neq largest.left$ **then**

    Insert interval $[median + 1,\ largest.right]$ into $chunks$

return $median$

Algorithm 3 ESBMC–νZ-Based HW/SW Partitioning

Input: Task graph $G(V, E)$; hardware cost vector $\mathbf{h}$; software cost vector $\mathbf{s}$; communication cost vector $\mathbf{c}$; software-cost bound S_0

Output: Optimal hardware/software partition minimizing hardware cost while satisfying the software-cost constraint

Initialize the νZ optimization context

Create binary decision vector $x = \{x_1, x_2, \ldots, x_n\}$

Initialize the number of tasks (n) and communication edges (m)

Construct the transposed incidence matrix E

Compute the software cost

$$SC = s(1 - x)$$

Compute the communication cost

$$CC = c|Ex|$$

Define the objective function

$$F_{\text{obj}} = \sum_{i=1}^{n} h_i x_i$$

Add the constraint

$$SC + CC \leq S_0$$

Minimize F_{obj}

Invoke the νZ optimizer to solve the model

Retrieve and print the optimal HW/SW partition

return Optimal partition and minimum hardware cost

The binary search strategy proves particularly effective for HW-SW partitioning where the objective function is monotonic with respect to the hardware cost, allowing large portions of the search space to be eliminated based on intermediate results.

4.5 MaxSMT Integration with ESBMC-νZ

ESBMC-νZ integrates the MaxSMT solver directly into the verification workflow, reformulating the optimization problem using νZ's native optimization functions. Algorithm 3 outlines the approach, which creates a νZ context and declares variables, costs, and constraints.

The key advantage lies in νZ's ability to optimize directly within the SMT solving process, avoiding external search loops. The tool handles both hard constraints (which must be satisfied) and soft constraints (which are optimized), applying specialized algorithms for MaxSAT and linear arithmetic optimization [14].

This integration typically outperforms external optimization approaches for problems where the SMT solver can efficiently reason about the constraint structure, though scalability may still challenge very large instances.

5. Experimental Evaluation

5.1 Methodology

Our experimental setup used ESBMC v2.0 on 64-bit Ubuntu 14.04.1 LTS with Boolector 2.0.1 as the default SMT solver. We implemented ESBMC-SS, ESBMC-PS, and ESBMC-PB in C++11 with OpenMP,

Table 1: Benchmark Characteristics

Name	Nodes	Edges	Description
CRC32	25	32	32-bit cyclic redundancy check
Patricia	21	48	Routine to insert values in Patricia Tree
Dijkstra	26	69	Compute shortest paths in a graph
Clustering	150	331	Image segmentation algorithm
RC6	329	448	RC6 cryptography algorithm
Fuzzy	261	422	Fuzzy logic clustering algorithm
Mars	417	600	MARS cipher from IBM

while ESBMC-νZ used the built-in Z3 integration. Comparative evaluation included MATLAB R2013a with Parallel Computing Toolbox for ILP and GA implementations.

All experiments ran on an Intel Core i7-2600 (8 cores, 3.4 GHz) with 24 GB RAM. We measured each configuration three times, reporting averages with statistical confidence between 91.7% and 95.9%. Timeout (TO) was set at 3600 seconds, with memory-out (MO) at 15 GB.

5.2 Benchmark Characteristics

We employed seven standard HW-SW partitioning benchmarks from embedded applications, as detailed in Table I. These represent diverse computational patterns and complexity levels, from small control-oriented tasks to large computational kernels.

Benchmarks include both classical algorithms (Dijkstra, CRC32) and modern embedded applications (RC6, Mars), providing a comprehensive test suite. Software costs represent execution time, hardware costs represent area, and communication costs model data transfer overhead.

5.3 Results and Analysis

Table II presents experimental results comparing all approaches across the benchmark suite. The exact solution column shows optimal hardware costs (H_p) and software costs (S_p) for reference.

ESBMC-νZ demonstrated superior performance on small to medium benchmarks (CRC32, Patricia, Dijkstra, Clustering), solving them 2-5x faster than ILP and significantly outperforming all multicore ESBMC variants. The integration with MaxSMT solving avoids external search loops, providing native optimization within the constraint solver.

For the Clustering benchmark (150 nodes), ESBMC-PB achieved a 3x speedup over ILP, though it remained 2.5x slower than ESBMC-νZ. This demonstrates the effectiveness of binary search parallelization for intermediate-sized problems, where the reduced verification steps compensate for parallelization overhead.

ILP solved 5 of 7 benchmarks, showing robustness across problem sizes but struggling with the largest instances (Fuzzy, Mars). When successful, ILP guaranteed optimality but with longer runtimes than ESBMC-νZ for supported benchmarks.

GA solved all benchmarks except Mars but exhibited significant optimality errors (-37.6% to 29.0%). While providing faster solutions for complex problems, these errors may be unacceptable in constrained embedded systems where resource utilization directly impacts product viability.

The RC6 benchmark (329 nodes) caused timeouts for all ESBMC approaches, while ILP solved it in approximately 30 minutes. This suggests a scalability limit around 300 nodes for current SMT-based

Table 2: Experimental Results Comparison (Time in Seconds)

Approach	CRC32	Patricia	Dijkstra	Clustering	RC6	Fuzzy	Mars
Exact Solution							
H_p	15	47	31	241	692	13820	876
S_p	19	4	19	46	533	4231	297
ILP							
Time (s)	1.6	1.3	1.6	648.9	1806.2	TO	TO
H_p	15	47	31	241	692	-	-
GA							
Time (s)	6.7	7.4	8.8	340.4	2050.0	1371.9	TO
Error %	13.3	0.0	29.0	1.7	-6.5	-37.6	-
Sequential ESBMC							
Time (s)	30.3	313.7	324.7	MO	MO	MO	MO
H_p	15	47	31	-	-	-	-
ESBMC-SS							
Time (s)	2.2	5.8	7.0	1609.3	TO	TO	TO
H_p	15	47	31	241	-	-	-
ESBMC-PS							
Time (s)	3.7	10.0	12.0	2468.0	TO	TO	TO
H_p	15	47	31	241	-	-	-
ESBMC-PB							
Time (s)	4.3	4.7	6.3	218.7	TO	TO	TO
H_p	15	47	38	241	-	-	-
ESBMC-νZ							
Time (s)	0.3	0.3	0.7	86.4	TO	TO	TO
H_p	15	47	31	241	-	-	-

approaches, though ESBMC-PB showed promising results for intermediate sizes.

Memory limitations affected sequential ESBMC on the Clustering benchmark, while multicore approaches successfully solved it through distributed verification. This highlights the memory efficiency advantages of parallelization, where individual verifications handle smaller subproblems.

6. Discussion

HW-SW partitioning research has evolved along three main trajectories: exact solutions, heuristic approaches, and hybrid methods [1]. Our work contributes to exact solutions through SMT-based verification, guaranteeing optimality while improving scalability through parallelization.

Early exact approaches focused on ILP formulations [2], providing optimal solutions but facing combinatorial explosion with problem size. Heuristic methods including genetic algorithms [4], simulated annealing, and particle swarm optimization offered better scalability but sacrificed optimality guarantees. Cooperative game-theoretic approaches [3] have also been proposed to model the interactions between hardware and software components. SMT-based verification for optimization represents a more recent direction. Previous work adapted BMC for property checking, but our contribution extends this to optimization through external search loops and MaxSMT integration. The multicore implementations

specifically address the performance limitations of sequential SMT solving. Layered and parallelized model checking techniques [11] and scalable verification strategies [12] provide complementary perspectives on improving verification performance. Parallel model checking has been explored through swarm verification and distributed memory approaches. However, these focused on property verification rather than optimization. Our work adapts parallel patterns specifically for optimization tasks, demonstrating their effectiveness for HW-SW partitioning. MaxSMT solving with νZ has shown promise for various optimization problems [14]. Applications include compiler error localization and software configuration. Our integration with ESBMC creates a unified framework for verification and optimization, particularly beneficial for embedded systems with formal constraints. The reliability of such verification outcomes is further supported by SMT proof checking frameworks [13]. Comparative studies of optimization tools have highlighted performance differences across problem types. Our evaluation confirms that problem characteristics significantly impact tool performance, justifying a portfolio approach where multiple techniques are available based on problem size and structure. Recent advances in hardware-software co-design frameworks [5–7] emphasize the importance of scalable partitioning techniques in modern AI and chiplet-based systems, further motivating our work. The evolution of hardware description languages such as Anvil [8] and compiler toolchains like Murphi2Chisel [9] also underscore the growing need for formal methods and verification in system design.

7. Conclusion and Future Work

We presented and evaluated five SMT-based approaches for HW-SW partitioning, demonstrating that multicore verification and MaxSMT integration can significantly accelerate exact optimization for embedded co-design. Experimental results provide clear guidance for practitioners: ESBMC-νZ excels for small to medium problems (under 150 nodes), multicore ESBMC-PB offers the best performance for intermediate sizes (150-300 nodes), and ILP remains necessary for the largest instances, while GA serves only when approximate solutions suffice.

The research contributions include: (1) multicore ESBMC implementations that leverage modern processors without modifying solver backends, (2) integration of MaxSMT solving through νZ for native optimization [14], and (3) comprehensive evaluation across standard benchmarks quantifying performance-optimality tradeoffs.

Future work will focus on several directions. First, we plan to implement parallelization within ESBMC's symbolic execution engine rather than as an external layer, potentially reducing overhead and improving scalability. Second, we will explore hybrid approaches that combine exact and heuristic methods, using fast approximations to guide the exact search. Finally, we will investigate incremental SMT solving techniques that reuse learned constraints across verification instances, particularly beneficial for the binary search strategy.

The granularity of partitioning models represents another important dimension. Fine-grained models (instruction-level) enable better optimization but increase problem size exponentially. Coarse-grained models (component-level) reduce complexity but may miss optimization opportunities. Adaptive granularity approaches that dynamically adjust model detail based on computational resources offer promising middle ground.

As embedded systems continue growing in complexity, the marriage of formal methods with parallel computing provides a pathway to maintain design quality while managing computational costs. Our work demonstrates that multicore SMT-BMC with MaxSMT support represents a viable addition to the embedded designer's toolkit, particularly for early design space exploration where multiple partitioning

alternatives must be evaluated under strict constraints. Optimization problems arise in diverse fields, from embedded systems to healthcare. The techniques developed here may also find applicability beyond traditional embedded domains, for instance in large language model fact-checking and optimization, opening avenues for cross-disciplinary research.

References

[1] S. Chen, L. Huang, G. Xie, R. Li, and K. Li. Application of uncertain programming in hardware/-software partitioning: Model and algorithm. *Journal of Circuits, Systems and Computers*, 32(06): 2350105, 2023.

[2] P. Cheng. Hardware and software partitioning method of embedded system based on immune algorithm. In *2023 International Conference on Applied Intelligence and Sustainable Computing (ICAISC)*, pages 1–6. IEEE, 2023.

[3] S. Deng, S. Xiao, H. Tao, and J. Zhou. A cooperative game-theoretic approach for hardware–software partitioning in multichiplet integrated systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2026.

[4] S. Deng, S. Xiao, Q. Deng, and H. Lu. A hovering swarm particle swarm optimization algorithm based on node resource attributes for hardware/software partitioning. *The Journal of Supercomputing*, 80(4):4625–4647, 2024.

[5] M. Vaithianathan. Hardware-software co-design for performance optimization in embedded systems. *International Journal of Emerging Research in Engineering and Technology*, 6(1):29–35, 2025.

[6] M. L. Varshika, A. K. Mishra, N. Kandasamy, and A. Das. Hardware-software co-design for on-chip learning in AI systems. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pages 624–631, 2023.

[7] Y. Du, Y. Wang, M. Wang, X. Li, and Y. Han. Chiplever: A hardware-software co-design framework towards extension of chiplet system for fully homomorphic encryption. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[8] J. Z. Yu, A. R. Jha, U. Mathur, T. E. Carlson, and P. Saxena. Anvil: A general-purpose timing-safe hardware description language. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 110–136, March 2026.

[9] Z. Cai, Y. Li, and Y. Zhao. Murphi2chisel: A protocol compiler from murphi to chisel. In *Proceedings of the 15th Asia-Pacific Symposium on Internetware*, pages 209–218, July 2024.

[10] C. Hensel, S. Junges, J. P. Katoen, T. Quatmann, and M. Volk. The probabilistic model checker storm. *International Journal on Software Tools for Technology Transfer*, 24(4):589–610, 2022.

[11] Y. Phyo, M. N. Aung, C. M. Do, and K. Ogata. A layered and parallelized method of eventual model checking. *Information*, 14(7):384, 2023.

[12] A. Mohan. Scalable formal verification strategies for risc-v based control paths: A review. n.d.

[13] D. Oe and A. Stump. Combining a logical framework with an rup checker for smt proofs. In *Satisfiability Modulo Theories (SMT)*, page 40, 2011.

[14] E. Albert, P. Gordillo, A. Hernández-Cerezo, and A. Rubio. A max-smt superoptimizer for EVM handling memory and storage. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 201–219. Springer International Publishing, 2022.