International Journal of Engineering & Technology, *7 (2.33) (2018) 252-254*

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

PSO based optimization of worst-case execution time for ASIP application

Mood Venkanna [1] *, Rameshwar Rao [2], P. Chandra Sekhar [3]

[1] Dept of ECE UCE, Osmania University Hyderabad, India
[2] Former VC, JNTUH Dept of ECE UCE, Osmania University Hyderabad, India
[3] Professor, Dept of ECE, UCE, Osmania University Hyderabad, India
**Corresponding author E-mail: venkatmood03@gmail.com*

Abstract

Industrial requires hard real-time systems for safety and critical applications like automotive, Aeronautics, manufacturing control and train industries. Hard Real-Time Systems' embedded controllers are with expectation of complete the tasks within a certain time bounds reliably including task scheduling. The estimation of upper bound limits corresponding to the execution times is often termed as the Worst-Case Execution Times (WCETs). It is an essential step in developing and validating the hard real-time systems. Particularly, the upper bounds need to satisfy these constraints related to the execution times. However, it is often not feasible many times to set upper bounds on execution times for programs. In present work, the problem of choosing reconfigurable Custom Instructions (CIs) is accomplished by optimizing the WCET corresponding to an application. This issue is designed using Particle Swarm Optimization (PSO) based program for a path analysis. The work emphasizes on the effectiveness of optimizing the WCET when applied to a reconfigurable processor. It evaluates a compound application of multimedia with a host of reconfigurable CIs corresponding to a number of hardware parameters.

Keywords: Hard Real-Time System; Embedded Controller; ASIP; Optimization; PSO; Space Exploration.

1. Introduction

In today's complex applications including multimedia application involving multiple hot spots require different form of hardware accelerators in order to accomplish the intended performance. In this regard the Application Specific Instruction, set Processors (ASIPs) facilitate a developer to design specific pre-fabrication customizations so as to improve the desired degree of specialization to meet the actual application environments such as the computational hot spots. Nevertheless, the ASIP can be kept within its reasonable size by only a few subsets of hot spots.

It is essential to know the WCET of a program for the design and verify the real-time systems. An efficient WCET method of analysis need to consider the feasible program flow that involves the loop iterations, the function calls and the timing effects of the caches and pipelines. One of the essential parameter of WCET analysis is the calculation as the combination of the flow as well as the timing information ofhardwareto computes a corresponding program in WCET analysis. The precision of WCET estimate largely depends on the type of flow information. In conventional methods, a trade-off is made between the global calculations precision and the computational complexity. However, such traditional methods are although fast but unable to account for all such flow information.

The flow information are essentially acts as a set of flow facts, each of which gives the constraints with respect to the program flow that include the infeasible paths, loop bounds, execution dependencies, etc. The local nature of the flow facts often express information confined to smaller regions such as an if-statement or a single loop of the intended program. In some cases, sometimes these smaller regions may occupy the structures of the basic program.

The task of estimating a suitable upper bound for WCET remains complex in nature due to the performance enhancing features like Pipelining, Caches and Branch predictions made available in the modern day processor. The presence of micro architectural state creates latency in instruction execution based on the dependency level in the past. When micro-architectural component, are in a worst-case scenario like a cache miss wherein the micro-architectural state is unable to determinethe result statically in a safe WCET bound for the complete task.

We would like to present an efficient approach to select the reconfigurable CIs statically in view of optimize the WCET bound of a task. Further, the time analysis of the high-performance architectures has been made to advance the research. A major issue is to resolve the worst-case path volatility that selects the WCET and optimize the CIs. To accomplish this task the worst-case path latency is by inserting a CI and to generate an entirely new different path.

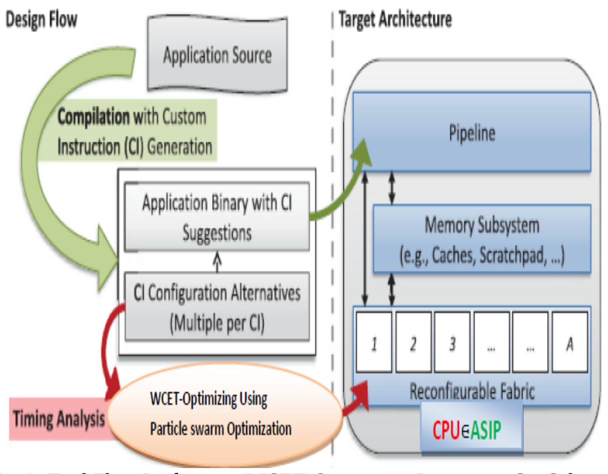
Design Flow
Application Source
Compilation with Custom Instruction (CI) Generation
Application Binary with CI Suggestions
CI Configuration Alternatives (Multiple per CI)
Timing Analysis
WCET-Optimizing Using Particle swarm Optimization
Target Architecture
Pipeline
Memory Subsystem (e.g., Caches, Scratchpad, ...)
1
2
3
...
...
A
Reconfigurable Fabric
CPU∈ASIP

Fig. 1: Tool Flow Performing WCET-Optimizing Instruction Set Selection Integrated with Timing Analysis.

As shown in Figure 1, an integral component of WCET-optimizing CI selection is the WCET bound estimation. In this, the CI or the instruction set selection is another major step used inthe extension problem of the instruction set [1].
The paper is sectioned as explained. In section [1-2] the concerned literature is presented. Section 1.3 explains the optimization technique using PSO. The experiment result discussion is presented on section 1.4. The conclusion part is described on section 1.5.

2. Related literature

The flow, low-level analysis and the calculation are the essential components for the generation of the WCET estimate. The flow analysis (FA) phase aims to represent a program's dynamic behavior. The process involves the function to be called, number of loop iterations how many times loops iterate in case air-statements dependencies exist. All possible program executions need to be safe approximated as the FA is unaware of the execution path which with respect to the longest execution time. These information are extracted using either the integrated in the programming language (manual annotations) [2] or provided separately [3-5]), or by automatic flow analysis techniques [6-10]. Conventionally, the FA is referred as high-level analysis as it is mostly accomplished using the source code although it may be possible to use the machine code or intermediate level.
Analysis of instruction caches, data caches, and branch predictors corresponding to global low-level analysis has been a major source of inspiration in this direction [5-16].
WCET calculation:
To improve a program's average-case execution time, advanced processors use the cache hierarchies as well as out of order execution although these features incorporated difficulties for determination of tight WCET. On the other hand, complex architectures require more states in the model checker to be prone to state explosion problems. For such cases, the timing anomalies with respect to the shared resources or memory accesses are desired. As an example, let us consider a separate worst-case path ABD where A, B and D are the basic building blocks. However, availability of shared L2 cache may result in other paths such as ACD as the worst-case path in case more instructions are evicted by L2 cache from C than B. It is essential to take into account the tasks of different cores together for multicore analyzing of WCET that in increases the computation time.
The modeling of a similar problem with Execution Flow Graphs and Trees has been proposed in [17-18]. The authors used the function level calls in modeling the execution flow in their work. Similarly, in [19], the authors proposed the optimization of WCET instruction set selection applied to extensible processors with custom functional units. These units are configurable for implementation of instruction patterns used frequently. It also speeds up the process by exploiting instruction level parallelism and operator chaining. While optimizing the WCET instruction set for a particular task that is executed, the instruction set remains fixed. Thus, selected pattern configuration cost is not considered when this approach is executed. In this piece of work, a dynamic reconfiguration based custom instructions has been targeted in which the area demands varies.

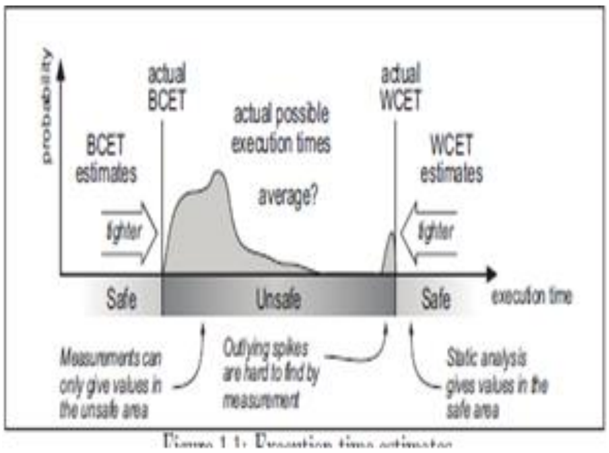
probability
actual BCET
actual possible execution times
average?
actual WCET
BCET estimates
WCET estimates
tighter
tighter
Safe
Unsafe
Safe
execution time
Measurements can only give values in the unsafe area
Outlying spikes are hard to find by measurement
Static analysis gives values in the safe area

Fig. 2: Execution Time Estimates

3. Optimization using particle swarm optimization (PSO)

In order to allow solutions in the reconfigurable fabric this work introduces an area constraint as the reconfigurable fabricsum $a_{k,j}$which need to be implemented the corresponding CIs where, jdenotes the selected implementation in which case the term $y_{k,j} = 1$. It needs to be either low or equal to the total fabric area a [20].
Consider a program having N basic blocks. The objective function can be represented as

$$max \sum_{i=1}^{N} c_i x_i \qquad (1)$$

Selecting an instruction set to optimize the WCET bound essentially means. we aim to minimize the WCET over all possible selections, that is, we aim to minimize the maximum execution time.
Steps for Optimization
The population based adaptive stochastic optimization method is PSO. The technique is based on collective swarm intelligence (SI).
The steps of PSO algorithm are as follows:
Step 1: Start and define solution space
Step 2: Represent potential solution to particle
Step 3: Initialize the population with respect to the random position x as well as the velocityv vectors
Step 4: Evaluate the fitness for each agent.
Step 5: If fitness (x) > fitness (gbest) then gbest=x
Step 6: If fitness (x) > fitness (pbest) then pbest=x
Step 7: Update position and velocity
Step 8: gbest is best solution of parameters
In this work, the use of PSO is based on [24], initialize the particle with position vector (x) and velocity vector (v) respectively. The updated position vector and velocity vector for particle (i) are given by:

$$x_i^{k+1} = x_i^k + v_i^{k+1} \Delta t$$

$$v_i^{k+1} = w^k v_i^k + c_1^* rand_{1,i}^k (p_i^k - x_i^k) + c_2^* rand_{2,i}^k (g_i^k - x_i^k)$$

Where, k represents present iteration, i is particle index with x_i^k and v_i^k are present positions and velocities respectively. The updated velocity corresponds to the previous velocity weighted by inertia w^k corresponding with the parameters$c_1^* and c_2^*$, which are cognitive and social rates respectively. The cognitive and social rates represent whether the individual agent has capabletowards the pbest position and towards the gbest position. p_i^kRepresents personal best position and g_i^krepresents global optimum position of each individual particle. The experience of each individual particle is (c_1^*) w.r.t its pbest position, where as the gbest position is found by neighborhood of each individual particle or by whole swarm related to social parameter(c_2^*). $rand_{1,i}^k$In addition,$rand_{2,i}^k$ are two random numbers in the range [0, 1], which are uniformly distributed. ΔtIs time difference, which is unity?

4. Result and discussion

The number of possible hardware configurations m_k per CI $_k$changes drastically. As an example, in this work, we encountered a minimum CIs value of [1] with a corresponding maximum value of 80= M implementations. It includes the entire software implementation with respect to a single kernel. When the CI value has been 80 different implementations, different degrees of parallelism have been observed with latencies. The area and the reconfiguration delay have been same during synthesization of the reconfigurable fabric. In case of minimum-latency implementation of the fabric the proposed algorithm canprovide a number of relevant implementations.

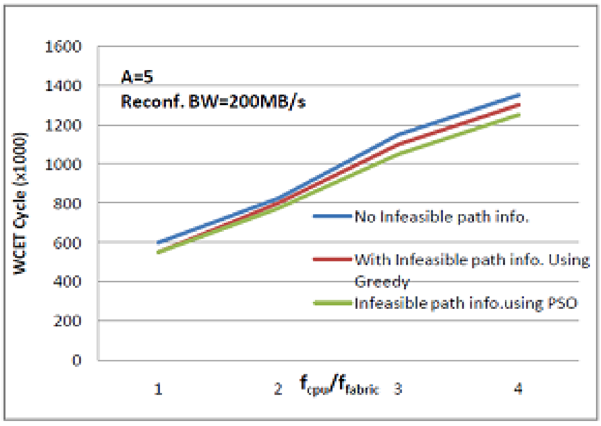

Fig.3: Optimal Results for H.264 Encoder and Different Values off CPU/Ffabric, As Well as Reconfiguration Bandwidth.

The work evaluates the results of the heuristic with respect to an optimal search. To practically demonstrate our approach the runtimes have been studied.

5. Conclusion

This paper attempts a new method to compute the WCET of a program. The technique is a hybridization of the fast computation techniques like the tree and path-based algorithms, are less precise, and potentially slow global IPET. It aims to find the smallest possible program components that can be handled for precision. The method of calculation for every individual component is not fixed and is dependent on the flow information characteristics and the program structure. Due to smaller parts as compared to the overall program, the technique provides the desired precision and is fast while arbitrary boundaries are introduced similar to the tree and path-based approaches.

References

[1] C. Galuzzi**,** K.L.M. Bertels**,** The Instruction-Set Extension Problem: A Survey (December 2010), *ACM Transactions on Reconfigurable Technology and Systems (TRETS), volume 4, issue 2, 2010.*

[2] R. Kirner and P. Puschner, "Transformation of Path Information for WCET Analysis during Compilation," in Proc. 13thEuromicro Conference of Real-Time Systems, (ECRTS'01). IEEE Computer Society Press, Jun 2001.

[3] J. Engblom and A. Ermedahl, "Modeling Complex Flows for Worst-Case Execution Time Analysis," in Proc. 21th IEEE Real-Time Systems Symposium (RTSS'00), Nov 2000.

[4] C. Ferdinand, F. Martin, and R. Wilhelm, "Applying Compiler Techniques to Cache Behavior Prediction," in Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (LCT-RTS'97), 1997.

[5] Y.-T. S. Li and S. Malik, "Performance Analysis of Embedded Software Using Implicit Path Enumeration," in Proc. Of the 32: nd Design Automation Conference, 1995, pp. 456–461.

[6] J. Gustafsson, B. Lisper, C. Sandberg, and N. Bermudo, "A Tool for Automatic Flow Analysis of C-programs for WCET Calculation," in 8th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS'03), Jan 2003.

[7] C. Healy, R. Arnold, F. M¨uller, D. Whalley, and M. Harmon, "Bounding Pipeline and Instruction Cache Performance," IEEE Transactions on Computers, vol. 48, no. 1, Jan 1999.

[8] N. Holsti, T. L°angbacka, and S. Saarinen, "Worst-Case Execution-Time Analysis for Digital Signal Processors," in Proceedings of the EUSIPCO 2000 Conference (X European Signal Processing Conference), Sep 2000.

[9] T. Lundqvist and P. Stenstr¨om, "An Integrated Path and Timing Analysis Method based on Cycle-Level Symbolic Execution," Journal of Real-Time Systems, May 2000.

[10] F. Stappert and P. Altenbernd, "Complete Worst-Case Execution Time Analysis of Straight-line Hard Real-Time Programs," Journal of Systems Architecture, vol. 46, no. 4, pp. 339–355, 2000.

[11] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm, "The Influence of Processor Architecture on the Design and the Results of WCET Tools," IEEE Proceedings on Real-Time Systems, 2003.

[12] S.-S. Lim, Y. H. Bae, C. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Ki, "An Accurate Worst-Case Timing Analysis for RISC Processors," IEEE Transactions on Software Engineering, vol. 21, no. 7, pp. 593–604, Jul 1995.

[13] S.-K. Kim, S. L. Min, and R. Ha, "Efficient Worst Case Timing Analysis of Data Caching," in Proc. 2nd IEEE Real-Time Technology and Applications Symposium (RTAS'96). IEEE, 1996, pp. 230–240.

[14] R. White, F. M¨uller, C. Healy, D. Whalley, and M. Harmon, "Timing Analysis for Data Caches and Set-Associative Caches," in Proc. 3rd IEEE Real-Time Technology and Applications Symposium (RTAS'97), Jun 1997, pp. 192–202.

[15] A. Colin and I. Puaut, "Worst Case Execution Time Analysis for a Processor with Branch Prediction," Journal of Real-Time Systems, vol. 18, no. 2/3, pp. 249–274, May 2000.

[16] T. Mitra and A. Roychoudhury, "Effects of Branch Prediction on Worst Case Execution Time of Programs," National University of Singapore (NUS), Tech. Rep. 11-01, Nov 2001.

[17] Falk, H., Plazar, S., &Theiling, H. (2007, September). Compile-time decided instruction cache locking using worst-case execution paths. In Proceedings of the fifth IEEE/ACM international conference on Hardware/software codesign and system synthesis (pp. 143-148), 2007. ACM.

[18] Liu, T., Li, M., &Xue, C. J. (2009, April). Minimizing WCET for real-time embedded systems via static instruction cache locking. In Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE (pp. 35-44), 2009. IEEE.

[19] Mitra, T., & Yu, P. (2005, September). Satisfying real-time constraints with custom instructions. In Hardware/Software Codesign and System Synthesis, 2005. CODES+ ISSS'05. Third IEEE/ACM/IFIP International Conference on (pp. 166-171). IEEE.

[20] Kennedy, J. (2011). Particle swarm optimization. In Encyclopedia of machine learning (pp. 760-766). Springer US, 2011.