International Conference on Machine Learning and Computational Intelligence-2017
International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 7 | ISSN : 2456-3307

Optimization of Worst-Case Execution Time for ASIP using Genetic Algorithm

Mood Venkanna*[1], Rameshwar Rao[2], P. Chandra Sekhar[3]
[1]Department of ECE, UCE, Osmania University, Hyderabad, India
[2]Former VC JNTUH, Department of ECE, UCE, Osmania University, Hyderabad, India
[3]Professor, Department of ECE, UCE, Osmania University, Hyderabad, India

ABSTRACT

The use of Application Specific Processor is available almost in all the areas. Research and developments on ASIP has been progressed since last two decades. However, the minute analysis of these processors is still a great challenge for the engineers and researchers in current scenario. Embedded processor application spread over different areas with a high desire of fast and accurate execution. It requires enhancing the execution time of the processor. The worst-case execution time (WCET) evaluation satisfies the desire of user end along with the hardware and software application of the processor. As a result the reconfiguration of processor architecture can be modified and perfect task scheduling can be performed. For WCET, upper bound on execution time is to be focused. An attempt is made to optimize the WCET to enhance the performance of the processor along with less occupation of space. Genetic Algorithm (GA) as the popular optimization technique is utilized to optimize that can help to the reconfigurable processor performance and also the control flow of the instruction to the processors.

Keywords : Embedded processor, ASIP, Optimization, Genetic Algorithm, WCET.

I. INTRODUCTION

An embedded system relies on Application-specific instruction set processor (ASIP) design to meet its desired performance and cost effectiveness. Additionally, these processors found useful in cellular phones, avionics, automobile control systems etc. in which a slight change in performance or cost may impact drastically the productivity. There are different ASIP designs available such as Co Ware Lisa Tek products, TensilicaXtensa processor, Target Compiler Technologies products and Xilinx Micro Blaze. These ASIPs remain sole solution for the physical constraints and for the desired functions due to programmability and high flexibility. This paper will provide a brief explanation how the ASIP systems can be improved further.

Program Execution time is a crucial component in any real-time system as it may result in catastrophically consequences in case the deadline is missed. The worst execution time measurements using the worst possible program input remains unreliable today. Further, a program's worst execution path may not be not captured during the measurements. A number of researches in WCET (Worst Case Execution Time) analysis and theoretically estimate of WCET models has been conducted during last few years (Asavoae,M.et.al., 2013, Cl´ementBallabriga et al., 2010, Banerjee,A., et.al. 2013, Armin Biere et al., 2013) . However, application of actual real-time operating system code models has been least considered.

In embedded systems, WCET of a program need to be less than a specific threshold particular important in case of synchronous active control loops (Bjørner,N. Dutertre,B. and Moura, L., 2008). For a program, the WCET is computed as a combination of low-level, micro architectural reasoning. This involves pipeline, busses, cache states, cycle-accurate timing as well as higher-level reasoning such as the loop counts, program control flow and the variable pointers. It requires application of abstract interpretation with respect to the micro architecture, deduce elementary block's worst case timings and reassemble to global WCET using the control flow, maximal iteration counts by means of integer linear programming (Cadar, C. et. al, 2008).
In this paper, static method is optimized using genetic algorithm. The control flow and the path analysis is the major focus for optimized WCET. Rest of the paper is organized as the subsections for technique of WCET that follows the optimization method. Next to it the result has been explained. Finally it concludes this piece of wok.

II. TECNIQUES OF WCET AND THE MODEL

WCET has been used in many real-time systems due to safety, reliability, and surfacing of software in automotive systems. It serves as an input to schedule ability analysis in system design. Few of the automated approaches for WCET computation includes:

- ✓ Analytical techniques for test cases that boost the confidence for end to end measurements
- ✓ Static analysis of the software.
- ✓ combined or hybrid approaches that include both measurements and structural analysis
- ✓ Worst-case path determination

a. Maps control flow graph to an integer linear program
b. Determines upper bound and associated path

For accurate WCET computation the possible program flow involving function calls and loop iterations including their effects corresponding to hardware features need to be known.

WCET calculation:
For program average-case execution time improvement modern processors contains features such as order execution and cache hierarchies (Chattopadhyay, S., and Roychoudhury, A., 2013, Chu, D., and Jaffar, J., 2011, Wilhelm, R., 2006, Reinhard Wilhelm et al., 2008). Nevertheless, for atight WCET, these features make the system complex. Addition of more complex architectures in model checker increases the number of states which makes the track more prone to state explosion problems. However, availability of sophisticated tools like as Chronos cangues a better running of WCET in single core processors (Chaki, S., and Ivers, J., 2010). On the other hand, Multi core systems posses an additional complexity because of the shared resources or shared memories. Use of shared memory makes problems to obtain tight WCET.

Finding a method for WCET involve approximations thus, the exact WCET can be regarded as unachievable (Wankang Zhao et al., 2006, Kim, S.K. et.al.,1996, White,R., et.al.,1997, Colin, A. et.al., 2000)]. Finding the WCET are based on estimates which may be pessimistic. In such cases, the estimated WCET believes to be higher than the corresponding real or desired WCET. Hence, mostly in WCET analysis an attempt is made to reduce the pessimism with a low enough estimated value that can be of real interest to the system designer.

Static methods takes the task code in hand and do not depend on the executing code involving on a simulators or hardwares. Together with some annotations, the method analyzes the possible control flow through the given task, attempts to combine the control flow with hardware architecture abstract model so as to obtain the desired upper bounds.

Control-Flow Analysis

The control flow analysis is finite and aims to accumulate information on possible execution paths. Any superset can be considered as a safe approximation since the exact paths are not possibly determined. The analysis is difficult on machine code in comparison to the source code than as it is cumbersome to map the machine-code program results because of compilation, change of code optimization and linking in the structure. The basic concept of flow graph has been shown in Figure 1.

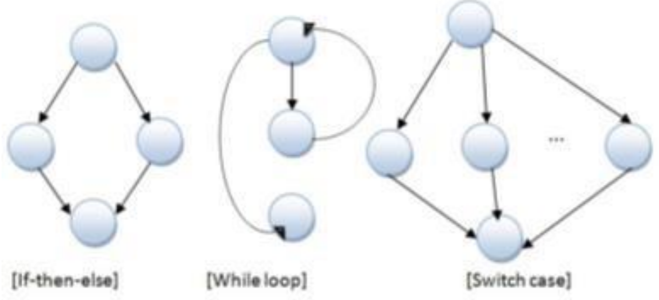

Figure 1. The Basic Concept of Flow Graph

It aims to estimate the WCET in dynamic approaches which may be underestimated since a subset of entire executions has been used for estimation. In static approaches also known as Bound Calculation an upper bound is computed for the entire execution times of task relying on the previous phase flow and timing information. There are three major classes such as the path-based, the structure-based, implicit path enumeration (IPET) based on analytically determination of end-to-end estimate times.

The structure-based approach cannot express every control flow through the syntax tree thus, assumes a straightforward relationship between the source and target program structures. Further, it is not feasible to incorporate additional flow information unlike the IPET. On the other hand, IPET can handle different flow information. It uses constraint programming or integer linear programming techniques and in this flow facts are converted to constraints whose size grows with the number of flow facts.

Let CI be the set of all CIs. We assume a specific configuration j of a CI k ∈CI in hardware to have a constant delay $t_{k,\ j}$ to require area on the reconfigurable fabric $a_{k,\ j}$∈[1, A], and to take a constant reconfiguration delay $r_{k\ j}$ for configuring it on the fabric. For a constant reconfiguration delay, a constant bandwidth for transferring configuration data to the reconfigurable fabric's configuration memory needs to be guaranteed. We assume the CPU to be delayed during reconfiguration in this work, and therefore the system bus could be utilized for reconfiguration at a guaranteed bandwidth. Along with hardware configurations, a CI can be implemented using its original software code j = 0. Since it has been implemented with a software, it does not have a constant delay $t_{k,0,}$ because of specific cache and pipeline analysis. (i.e., $a_{k,0}= r_{k,0} = 0$).

In order to provide flexibility to execute the original software for generated CIs, we introduce CI super blocks. The CI superblocks begin with a conditional branch before every CI (the actual instruction in the binary) which jumps to the functionally equivalent software code when the CI is not implemented in hardware. If a configuration for the CI is available on the reconfigurable fabric, then it is executed instead of jumping to the software. The CI super block ends by joining paths of hardware CI and software. Multiple CI superblocks in the binary can execute the same CI k. Let B be the set of all blocks, that is, basic blocks (not contained in super blocks) as well as super blocks. The function ci(i) determines which CI k is executed by a super block i ∈B, that is, ci: B → CI ∪ {0}, i → k, with ci(i) = 0 ∈CI if i is a basic block

To ensure that exactly one implementation is chosen potentially in software (j= 0)or hardware (j >0) with m_k being the number of hardware configurations of CI $_k$. To only allow solutions that do fit onto the reconfigurable fabric, we introduce the area constraint is the sum of area on the reconfigurable fabric $a_{k,j}$ required to implement all CIs using the selected implementation j (for which $y_{k,j}$ = 1) needs to be lower than or equal to the total fabric area A. For a program with a count of *N* basic blocks, the objective function is given 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.

We extend the ILP formulation of IPET for capturing the implementation alternatives of a CI *k* ∈CI. We introduce new variables $y_{k,j}$ ∈ {0, 1} for every implementation *j* with $y_{k,j}$= 1 if CI *k* is implemented using alternative *j* and $y_{k,j}$= 0 otherwise

$$\sum 0<j<n \; y_{k,}=1 \qquad (2)$$

The total cycle contribution of CI *k*'s super block *i* to the WCET bound is given as follows:

$$\sum^{|B|}_{\substack{1\le i \le |B| \\ 0<j<me(i)}} e_{i,j} y_{c(i),j}, x_i \qquad (3)$$

The WCET for a given selection *y* without accounting for reconfiguration delay can be determined as follows:

$$\mathrm{WCET}(y)=\max x\left(\sum^{|B|}_{\substack{i=1 \\ Ci(i)\in Cj}} c_i x_i + \sum^{|B|}_{\substack{1\le i \le |B| \\ 0<j<me(i)}} e_{i,j} y_{c(i),j}, x_i \,)\right) \qquad (4)$$

Every CI utilized in a kernel is configured exactly once before entering the kernel (with zero reconfiguration delay for software implementation). Therefore, we obtain the WCET including reconfiguration delay as:

$$\mathrm{WCET}(y) = \mathrm{WCET}'(y) + \sum_{\substack{0\le i \le m \\ 0<j<n}} y_{k,j}\, r_{k,j} \qquad (5)$$

III. OPTIMIZATION USING GA

Genetic Algorithm is a population-based search method in which the candidate solutions are termed as chromosomes, and the solution is termed as genes in the chromosomes. A search space has been formed using possible chromosomes. These are involved with corresponding fitness function that represents solutions encoded in the corresponding chromosome. The search continues by computing the fitness of a population of chromosomes followed by mutations and recombination with respect to successful chromosomes. The GA execution starts with a set of random initial population which are sampled for a particular task. The process of selection, crossover and mutation are applied on the initial population to get a new and better generation.

The basic Genetic Algorithm:

[Start]: random population of n chromosomes is generated that gives suitable solutions for the task.

[Fitness]: The fitness (*x*)with respect to each chromosome *x* in the population is evaluated.

[New population]: A new population is created using the below steps and repeating them till completion of the new population.

[Selection]: Two parent chromosomes are selected from a population as per their fitness).

[Crossover]: Cross over the parents with across over probability form a new offspring (children). In case of

no crossover the offspring is an exact replica of parents.
[Mutation] : Mutate new offspring with a mutation probability at each locus that gives the position in chromosome.
[Accepting]: The new offspring in a new population is placed.
[Replace]: The new generated population is used for a algorithm to be run further.
[Test]: When the end condition is satisfactory, stop, and return to the best solution with respect to the current population.
[Loop]: Go to fitness step.

The three basic steps for Genetic Algorithm, as shown above, are:
1. Selection: In selection (also known as reproduction), the chromosomes from the population to be parents are selected to cross over and produce offspring.
The various methods for parents to cross over are:

I. Roulette-wheel selection
II. Boltzmann selection
III. Tournament selection
IV. Rank selection
V. Steady-state selection

2. Cross over: The off springs are enriched with suitable individuals after the selection phase. Cross over process is continued to the mating pool and expected to create a better string. It also has three steps; firstly, the reproduction stage selects randomly a pair of two individual strings for mating. Secondly, a random cross-site is selected along the string length and at last their position values are swapped between those two strings. Different cross over types are:

I. Single-site cross over
II. Two-point cross over
III. Multi-point cross over
IV. Uniform cross over
V. Matrix cross over

3. Mutation: The strings are mutated once the cross over process is completed. It involves flipping of bits between 0 to 1 and vice versa using a small mutation probability Pm. A number is chosen between 0 to 1 randomly and the bit is changed if the number is less than Pm, otherwise it is unaltered.

Generating optimal test data using GA based on fitness function: On the basis of basis paths, the developed system automatically generates the optimal test data in the CFG. The WCET analysis tool architecture is shown in Figure 1.

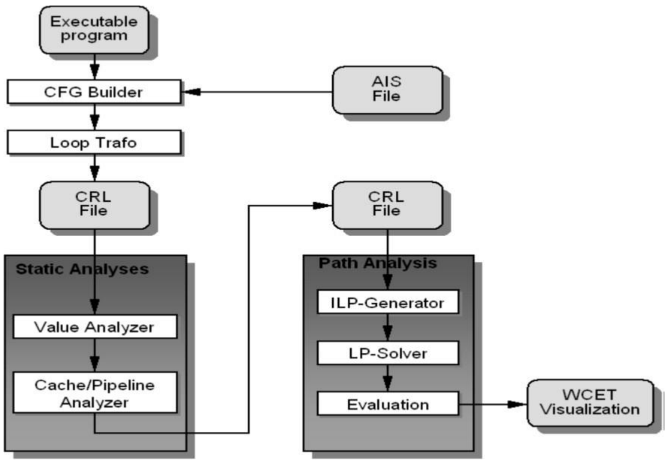

Figure 2. WCET Analysis Tool Architecture

IV. RESULTS AND DISCUSSION

The processor is reconfigured with the optimization. The parameter for genetic algorithm is given in Table-I and related convergence for hardware and software is shown in Figure 3.

Table 1. The parameter for genetic algorithm

Parameter	Value
Generations	350
Population Size	200
Chromosome Length	300
Selection mechanism Tournament	size=2
Crossover	0.85 (fixed point)
Mutation	0.02

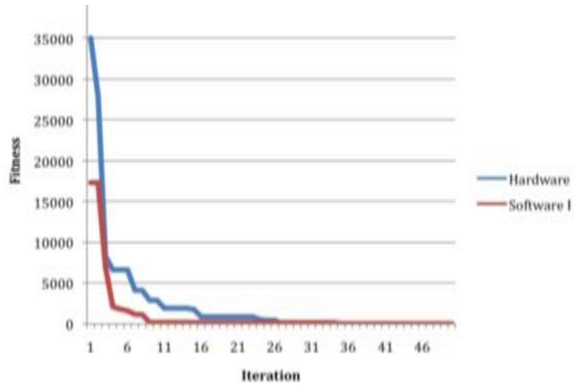

Figure 3. The Fitness for the Specific Function

V. CONCLUSION

It is observed that using satisfy ability modulo theory (SMT), the optimization has been a feasible approach in case of the bounding the WCET of the corresponding loop-free programs i.e., the programs in which the loops may be unrolled. To best of our knowledge, such an approach has been applied successfully for the first time. In all these levels we propose an evolutionary algorithm as the optimization engine, which is helped by other applications, either in a closed loop, either in off-line phases. The development of the computer-aided design or the CAD and compilation tools is one of the major challenges for mapping any application into an effective reconfigurable computing system. This desires the determination of application parts for mapping into the fabric and into the processor. Time of determination and its frequency of mapping into the reconfigurable fabric need to be emphasized in future.

VI. REFERENCES

[1]. Asavoae, M., Maiza,C. Raymond, P., 2013, Program Semantics in Model-Based WCET Analysis: A State of the Art Perspective. WCET Ed. by Claire Maiza. OASICS. 30, 32–41.

[2]. ClementBallabriga et al., 2010, OTAWA: An Open Toolbox for Adaptive WCET Analysis. SEUS. LNCS. Springer, 6399, 35–46.

[3]. Banerjee,A., Chattopadhyay,S. and Roychoudhury,A., 2013, Precise micro architectural modeling for WCET analysis via AI+SAT. IEEE Real-Time and Embedded Technology and Applications Sym- posium (RTAS), IEEE Computer Society, 87–96.

[4]. Armin Biere et al., 2013, The Auspicious Couple: Symbolic Execution and WCET Analysis. WCET, OASIcs. IBFI Schloss Dagstuhl, 30. 53–63. http://drops.dagstuhl.de/opus/volltexte/2013/4122.

[5]. Bjorner,N. Dutertre,B. and Moura, L., 2008, Accelerating lemma learning using joins - DPLL(⊔). Appeared as short paper in LPAR 2008, outside of proceedings.

[6]. Cadar, C. and Sen, K.., 2013, Symbolic Execution for Software Testing: Three Decades Later. Commun. ACM 56.2, 82–90.

[7]. Caspi,P., Raymond P., and Tripakis, S., 2008, Synchronous Programming. Handbook of Real-Time and Embedded Systems. Chapman & Hall / CRC, Chap. 14.

[8]. Chaki, S., and Ivers, J., 2010, Software model checking without source code. English. Innovations in Systems and Software Engineering 6.3, 233–242. ISSN: 1614-5046. doi: 10.1007/s11334-010- 0125-0.

[9]. Chattopadhyay, S., and Roychoudhury, A., 2013 Scalable and precise refinement of cache timing analysis via path-sensitive verification. Real-Time Systems 49.4, 517–562.

[10]. Chu, D., and Jaffar, J., 2011, Symbolic simulation on complicated loops for WCET Path Analysis.EMSOFT. 319–328. ISBN: 978-1-4503-0714-7. doi: 10.1145/2038642.2038692.

[11]. Wilhelm, R., 2006, Determining Bounds on Execution Times. Handbook on Embedded Systems. CRC Press, Chap. 14.

[12]. Reinhard Wilhelm et al., 2008, The worst-case execution-time problem - overview of methods

and survey of tools. ACM Trans. Embedded Comput. Syst.7.3.

[13]. Wankang Zhao et al., 2006, Improving WCET by applying worst-case path optimizations. Real- Time Systems 34.2, 129–152.

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

[15]. White,R., Muller, F. , Healy,C., Whalley,D., and Harmon, M.,1997, Timing Analysis for Data Caches and Set-Associative Caches. IEEE Real-Time Technology and Applications Symposium (RTAS'97), 192–202.

[16]. Colin, A. and Puaut, I., 2000, Worst Case Execution Time Analysis for a Processor with Branch Prediction.Journal of Real-Time Systems, 18, 2/3, 249–274.

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