Relative Roles of Instruction Count and Cycles Per Instruction in WCET Estimation

Archana Ravindar Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science
Bangalore-12, India
{archana,srikant}@csa.iisc.ernet.in

ABSTRACT

Most of the existing WCET estimation methods directly estimate execution time, ET, in cycles. We propose to study ET as a product of two factors, ET = IC * CPI, where IC is instruction count and CPI is cycles per instruction. Considering directly the estimation of ET may lead to a highly pessimistic estimate since implicitly these methods may be using *worst case* IC and *worst case* CPI. We hypothesize that there exists a functional relationship between CPI and IC such that CPI=f(IC). This is ascertained by computing the covariance matrix and studying the scatter plots of CPI versus IC. IC and CPI values are obtained by running benchmarks with a large number of inputs using the cycle accurate architectural simulator, *Simplescalar* on two different architectures. It is shown that the benchmarks can be grouped into different classes based on the CPI versus IC relationship. For some benchmarks like FFT, FIR etc., both IC and CPI are almost a constant irrespective of the input. There are other benchmarks that exhibit a direct or an inverse relationship between CPI and IC. In such a case, one can predict CPI for a given IC as CPI=f(IC). We derive the *theoretical worst case* IC for a program, denoted as SWIC, using integer linear programming(ILP) and estimate WCET as SWIC*f(SWIC). However, if CPI decreases sharply with IC then measured maximum cycles is observed to be a better estimate. For certain other benchmarks, it is observed that the CPI versus IC relationship is either random or CPI remains constant with varying IC. In such cases, WCET is estimated as the product of SWIC and measured maximum CPI. It is observed that use of the proposed method results in tighter WCET estimates than *Chronos*, a static WCET analyzer, for most benchmarks for the two architectures considered in this paper.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

General Terms

Experimentation, Measurement, Performance

Keywords

CPI, WCET analysis, benchmarking, covariance matrix, scatter plot, soft real-time systems

1. INTRODUCTION

The goal of *worst case execution time* (WCET) analysis is to estimate the longest execution time of a program on a given architecture. WCET analysis is valuable in real-time system design where programs are expected to meet stringent performance goals. It is also valuable in systems using dynamic task scheduling. The input which causes a program to execute for the longest amount of time is termed as *worst case input*. As WCET depends on both program structure and effect of the underlying architecture along various paths, deriving the worst case input is not easy.

Ideally, the estimated WCET is expected to be both *safe* and *tight*. Safe estimates are always greater than or equal to actual WCET and are critical to hard real time systems. A tight estimate is typically within a few percent of WCET and ensures effective resource management. Tight estimates that are rarely unsafe, are suited for *soft real-time* systems that can miss deadlines occasionally without causing significant change in system behavior.

The survey in [11] summarizes WCET analysis techniques in use today. A static WCET analyzer typically divides a program into basic blocks and estimates execution time of each basic block by modeling the architecture statically. Techniques like integer linear programming are employed to compute the worst case path in a program[11]. The static WCET is given by a weighted sum of the execution time of basic blocks along the worst case path. Although the estimate given by a static WCET analyzer is safe, it tends to be pessimistic due to conservative assumptions about program behavior. Moreover it is difficult to model all architectural features statically.

A measurement-based WCET analyzer measures execution time of smaller components like basic blocks[6, 9] or segments[8] by running the program with a large set of test inputs directly on the target architecture. These execution times are typically combined in a manner similar to static WCET analyzers to provide an estimate of WCET. Measurement based WCET analyzers rely on the availability of a comprehensive test input set ensuring adequate coverage.

The worst case input is often unknown thereby making it difficult to make any definitive statement on the safety or tightness of the estimate. Even if test inputs assure 95% path coverage, the worst case input could lie in the remaining 5%.

Many a times, the actual WCET may occur only very rarely and it is advantageous to look at WCET associated with probability of occurrence[6]. For this purpose, probabilistic methods are applied on measured execution times obtained using various test inputs. The method can be parametric or non-parametric. In a non-parametric method[10], the probability distribution of execution times of a program with various inputs is drawn in the form of a histogram. Rather than estimating just a single WCET value, the method allows one to estimate WCET at each percentile of probability. However the underlying assumption again is adequate coverage.

In a parametric method, a probabilistic model is fitted on the cumulative distribution function[7] of the measured execution times. The model curve is extrapolated to the point determined by the desired probability of an execution time ever exceeding a particular WCET value. However, if the tail of the distribution is very long, this might lead to a pessimistic estimate. Also, the accuracy of the parameters of the fitted model need to be validated.

2. PROPOSED STRATEGY

1. A common approach in static and measurement-based WCET analyzers is to divide a program into basic blocks. The worst case execution time of a basic block is computed as a product of the maximum frequency of the basic block and estimated execution time of a single instance of the basic block on the target architecture. A sum of such products for all the blocks gives the estimated WCET.

2. In contrast to point 1 above, the proposed method views an *entire program* as a single basic block. We will revert to the issue of dividing the program into blocks later. We measure program execution time, ET, and instruction count (IC) for various test inputs using the cycle accurate simulator, *Simplescalar*. All the architectural buffers are assumed to be empty before running the programs. The cycles per instruction(CPI) of the program for each test input is computed as a ratio of program execution time (ET) in cycles to the number of instructions executed(IC) since,

$$ET = IC \ * \ CPI \tag{1}$$

3. From equation (1), estimated WCET is equal to,

$$\widehat{WCET} = Worst(IC) \ * \ Worst(CPI) \tag{2}$$

But this estimate can be too pessimistic since both factors correspond to the worst case. This paper describes our efforts to improvise on this estimate.

4. We derive program worst case instruction count, SWIC, statically, by modifying the integer linear program formulation, used to derive WCET in static WCET analyzers. SWIC is a *theoretical upper bound* on IC and depends only on program structure and instruction set architecture. Hence it can be derived for any target with ease. The method used to obtain SWIC is described in the appendix.

5. Validation of a measurement-based WCET analyzer is an intractable problem. If the worst case input, WI, is known, estimating WCET is trivial since the program can be run with WI and execution time can be measured directly. When WI is unknown, there remains always an uncertainty regarding estimated WCET. Unless complete path and state coverage is achieved, no guarantee can be made regarding either safety or tightness of estimated WCET[11]. Hence it becomes essential to take a pragmatic approach and make certain reasonable assumptions in order to evaluate measurement-based WCET analyzers.

6. Assume for a program, that there exists a relation between CPI and IC for various inputs, given by an equation of the form, CPI = f(IC). Note that such a functional relationship is derived based on direct measurements and hence reliable. This implies that the CPI can be predicted for any given IC. If there are multiple CPI points for a given IC, we consider the maximum CPI to determine the functional relation.

7. Hence, estimated execution time,

$$\widehat{ET} = IC \ * \ f(IC) \tag{3}$$

Since we are interested in determining Max $\widehat{ET}$, assuming f(IC) to be continuous and differentiable, this occurs at IC = IC_{max} obtained by solving the equation,

$$\frac{d(\widehat{ET})}{d(IC)} = 0 \tag{4}$$

thereby setting, $\frac{d(IC*f(IC))}{d(IC)}$=0, we get,

$$Max \ \widehat{ET} = IC_{max} * f(IC_{max}) \tag{5}$$

The value of IC_{max} very much depends upon the functional relationship f(IC).

8. Since SWIC is a theoretical upper bound on IC, we consider SWIC as a candidate for IC_{max}. Hence estimated execution time based on SWIC is computed as,

$$\widehat{ET} = SWIC * f(SWIC) \tag{6}$$

9. However there may be a function f(IC) such that Max $\widehat{ET}$ computed by (5) is greater than that computed by (6) in which case Max $\widehat{ET}$ can be taken as $\widehat{WCET}$. If IC_{max} corresponds to one of the measured points then this estimate coincides with measured maximum cycles, *M*.

10. If there is no functional relationship between IC and CPI then $\widehat{WCET}$ is the maximum of
a) *SWIC * Maximum of measured CPI* and
b) *Measured maximum cycles,* M

Now we will demonstrate experimentally that a functional relationship exists between IC and CPI for many of the benchmarks considered in this paper.

3. CPI VERSUS IC RELATION

In this section we investigate the relationship between CPI and IC on two target architectures for each benchmark shown in Table 1. Work on other architectures and benchmarks are in progress. The test inputs are derived based on coverage of statements, decisions, conditions and modified conditions-decisions [5].

In addition to inputs satisfying coverage, we include inputs that make the program execute maximum number of

Benchmark	Description	Number of Inputs
bez[4]	(Bezier): Draw a set of 200 lines of 4 reference points on a 800 X 600 image	500 different inputs
bitc[2]	(Bitcount): Performs bit operations on an array of 1K numbers having various distributions	500 arrays
bs[3]	(Binary search): Search for a key in an array of 10K numbers using binary search	20K (key, array) pairs
bub[3]	(Bubble sort): Sort an array of size 3K	500 arrays
crc[3]	(CRC): Cyclic redundancy check on a char array of 16KB	500 arrays
cnt[3]	(Cnt): Counts positive numbers in a 200 X 200 matrix	500 matrices
dij[2]	(Dijkstra): Finds 100 shortest paths in a graph of 200 vertices using Dijkstra's algorithm	500 graphs
edn[3]	(Edn): Implements jpegdct algorithm together with other signal processing algorithms	500 arrays
fir[3]	(FIR): Finite impulse response filter algorithms over a 400items long sample	500 real-life audio samples
fft[2]	(FFT): Fast fourier transform on a wave of size 16K	500 real-life wave samples
ins[3]	(Insertion sort): Sort an array of size 3K	500 arrays
jan[3]	(Janne_complex): contains a nested loop whose inner loop max iterations depend on outer loop	500 pairs of a, b
lms[3]	(LMS) adaptive signal enhancement: The input signal is an array of 1000 coefficients	500 real-life samples
lud[3]	(LUD): LU Decomposition algorithm on a 200 X 200 matrix	500 matrices
minv[3]	(Minver): Matrix inversion on a 200 X 200 matrix	500 matrices
mat[3]	(Matmul): Matrix multiplication of two 200 X 200 matrices	500 matrices
ndes[3]	(NDES): Linear Search in a (29 X 29 X 29) array	500 arrays
nsch[3]	(Nsch): Simulate an extended petrinet	$dummy_i$ = 32, 500 initial states
qsort[3]	(Quick sort): sort a 3K array	500 arrays
Architecture	**Description**	
A	Issue, decode and commit width=1, RUU size=8, I-cache 1 KB L1 direct mapped 2 level branch predictor, Fetch Queue size=4, No D-cache	
B	In-order issue, Issue, decode & commit width=1, RUU size=8, I-cache 8KB L1 direct mapped, D-cache 8KB L1 2-way set associative, Unified 64KB 8-way associative L2 cache 2 level branch predictor, Fetch Queue size=4	

Table 1: Inputs tested for each benchmark and target architectures on which they are tested.

instructions whenever possible. For Ex: *bub*(Bubble sort) executes maximum instructions when the input array is reverse sorted. Executing the programs with test inputs generate IC and CPI vectors that consist of observed instruction counts, observed average CPI for each test input. The CPI of a program is influenced mainly by the target architecture on which it runs. When a program begins to execute, it takes time for architectural buffers to be filled with program related data, until then program CPI is comparatively high. This is referred to as *warmup* CPI. The measurements reported in this paper include the effect of warmup.

3.1 Covariance Matrix

The instruction count values are orders of magnitude greater than corresponding CPI values. Hence we normalize both IC and CPI vectors with respect to their respective measured maximum values before computing the covariance matrix. σ_{11}, σ_{22} denote the covariance in IC and CPI respectively. $\sigma_{12}(\sigma_{21})$ denotes the cross covariance between IC and CPI values. The elements of the covariance matrix of the benchmarks for both architectures are as shown in Table 2. Since the instruction set architecture is common for A and B, there is only one column for σ_{11}. While comparing the values for different benchmarks, the number of inputs used must be noted.

3.2 Scatter Plots

Scatter plots of CPI against IC are drawn. Some typical patterns are shown in Figures 1-6. The input (IC,CPI) that caused the program to run for the maximum number of cycles, *M*, is denoted by the symbol '$\lhd$' in the scatter plot. A vertical dashed line is drawn at IC=SWIC. A horizontal dashed line is drawn at CPI used to estimate WCET by the proposed method and is denoted by f(SWIC) in the scatter plot. The point (SWIC, f(SWIC)) is denoted by a '$\Box$'. The product of SWIC and f(SWIC) is the WCET estimated by the proposed method, $\widehat{WCET}$, unless otherwise specified. $\widehat{WCET}$ is compared with the estimate made by the static

Benchmark	σ_{11}	σ_{22}		$\sigma_{12}(\sigma_{21})$	
		A	B	A	B
Negligible variance in IC and CPI					
crc	0.0000	0.0005	0.0000	0.0000	0.0000
edn	0.0000	0.0087	0.0002	0.0000	0.0000
fft	0.0000	0.0000	0.0000	0.0000	0.0000
fir	0.0000	0.0007	0.0000	0.0000	0.0000
jan	0.0000	0.0002	0.0001	0.0000	0.0000
lms	0.0000	0.0001	0.0000	0.0000	0.0000
mat	0.0000	0.0000	0.0000	0.0000	0.0000
ndes	0.0000	0.0041	0.0000	0.0000	0.0000
Negative correlation					
bez	0.0005		0.0012		-0.0008
bitc	1.3144	0.0388	0.0769	-0.2059	-0.2992
bub	6.22	0.3252	1.3191	-1.0191	-2.0332
bs	0.1808	2.0984	0.1229	-0.6088	-0.123
cnt	0.0117	0.0035	0.0048	-0.006	-0.0073
ins	7.194		0.6061		-1.12
minv	1.9851		0.0947		-0.4261
nsch	20.144	3.9855		-7.8	
Positive correlation					
lud	11.49	0.8259	0.2332	2.6828	1.0353
bez	0.0005	0.0009		0.0002	
nsch	20.144		7.1128		11.4716
Constant CPI with varying IC					
ins	7.194	0.0277		0.2516	
minv	1.9851	0.0022		-0.0613	
Random correlation					
dij	3.559	0.0172	0.0640	-0.1305	-0.3289
qsort	0.0463	0.1101	0.017	0.0074	-0.0024

Table 2: Elements of Covariance Matrix for architectures A and B. Grouping is based on values of covariance matrix and scatter plots.

WCET analyzer *Chronos*[1] denoted by Chronos_Est and the results are shown in Table 3. The inherent CPI used by *Chronos*, computed as $\frac{Cycles\ estimated\ by\ Chronos}{SWIC}$ is indicated by a horizontal dashed line at CPI=$\text{CPI}_{chronos}$. The estimated WCET is validated by comparing it with measured maximum cycles, *M*. Due to the non-availability of measurement based WCET analyzers in the public domain, comparisons with other measurement based WCET analysis methods are not presented.

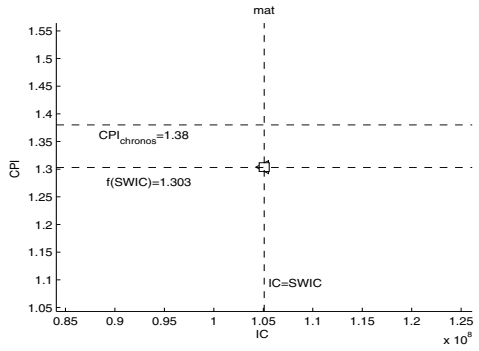

Figure 1: Scatter plot for *mat* for Architecture A.

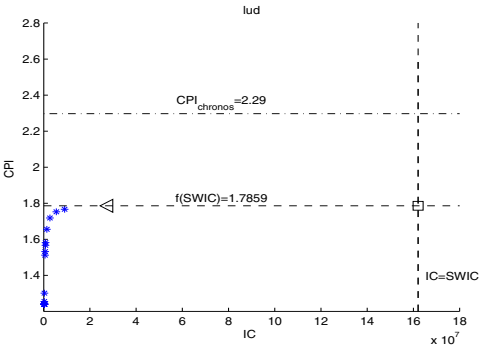

Figure 3: Scatter plot for *lud* for Architecture A.

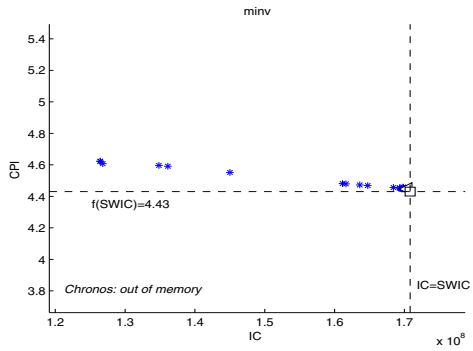

Figure 2: Scatter plot for *minv* for Architecture B.

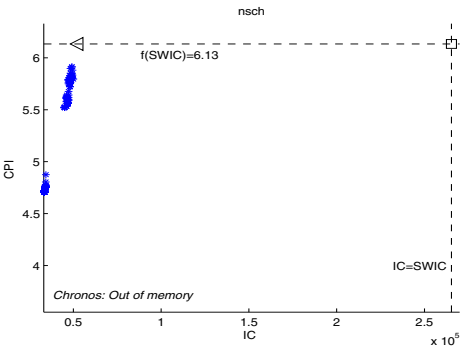

Figure 4: Scatter plot for *nsch* for Architecture B.

3.3 Benchmark Classification and WCET Estimation

From the covariance matrix and scatter plots it is noted that there are four groups of benchmarks that exhibit a definite relationship between IC and CPI and one group in which there exists no predictable correlation between IC and CPI. In the following discussion, a (benchmark, architecture) combination is denoted by *name(arch)*.

(i) Negligible Variance and Cross Correlation

Benchmarks *crc*, *edn*, *fir*, *fft*, *jan*, *lms*, *mat* and *ndes* show very little variance in both IC and CPI irrespective of input for both architectures. The scatter plot of *mat(A)* is shown in Figure 1. Although 500 test inputs are considered, only one point is seen in the scatter plot as all measurements of (CPI,IC) coincide. There is only one CPI that corresponds to maximum measured cycles M, in the scatter plot which is considered as f(SWIC).

For *mat(A)*, note that $\text{CPI}_{chronos}$ is higher than f(SWIC) as seen in Figure 1. It can be observed from Table 3 that *Chronos* overestimates WCET for all benchmarks in this class, especially *edn*(A and B), *fir*(A and B), *lms*(A and B) and *mat(B)*. For *ndes(B)*, *Chronos* goes out of memory while statically analyzing the program.

(ii) Negative Cross Correlation

Benchmarks *bitc*, *bub*, *bs*, *cnt* and *minv* show an inverse relationship between IC and CPI for both architectures. *bez(B)*, *ins(B)* and *nsch(A)* show an inverse relationship between IC and CPI. Figure 2 shows the scatter plot for *minv(B)*. Although 500 test inputs are considered, a large number of measurements overlap resulting in fewer points on the scatter plot. An almost linear trend with negative correlation is seen. In this paper, optimum curve fitting has not been done. Instead we assume that if SWIC lies close to (IC,CPI) corresponding to measured maximum cycles, M, we use the CPI corresponding to M as f(SWIC). If that is not the case, we assume piece-wise linear fit and extrapolate the curve up to SWIC to find f(SWIC). In case of *minv*, SWIC lies very close to point corresponding to M on the scatter plot. Hence we use the CPI corresponding to M as f(SWIC).

Referring to Table 3, *Chronos* gives a more pessimistic WCET for *bub(B)*, *cnt(B)*, *minv(A)*. For *minv(B)*, *Chronos* goes out of memory. It is important to note that if the decrease of CPI with IC is very steep then SWIC*f(SWIC) may be lower than M (as already mentioned in Sec. 2, point 9). This happens for *bs(B)*. In such a case, M is taken as WCET. This is a classic example of a case where worst case input is different from the input that causes the worst case path.

(iii) Positive Cross Correlation

Benchmark *lud* exhibits a direct relationship between IC and CPI on both architectures. Benchmarks *bez(A)* and *ins(A)* exhibit a direct relationship between IC and CPI. The scatter plot for *lud(A)* is shown in Figure 3. The CPI appears to be saturating as IC increases. This shows that the relationship between IC and CPI need not always be linear. By choosing the saturated CPI as f(SWIC), we observe that the proposed method gives a more tighter WCET estimate compared to *Chronos*.

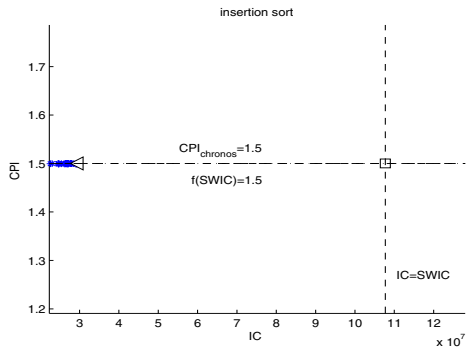

Figure 5: Scatter plot for *ins* for Architecture A.

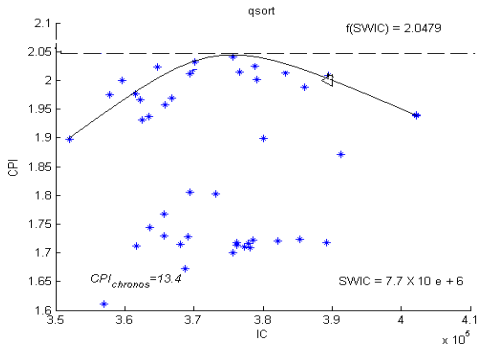

Figure 6: Scatter plot for *qsort* for Architecture A.

Benchmark *nsch(B)* shows a direct relationship between IC and CPI (Figure 4). The slope of the curve happens to be very steep. SWIC is much greater than IC corresponding to *M*. Ideally, we need more measurements to fill up this gap. But if SWIC is an overestimate then no matter how many measurements we make, the gap may not get filled at all. In the absence of a better alternative, we use the maximum observed CPI as f(SWIC) in this case. Given that our assumption of using maximum observed CPI is reasonable, *Chronos* gives a more pessimistic estimate for *bez(B)*, *ins(B)* and *nsch(A)* and goes out of memory while analyzing *nsch(B)*.

(iv) Constant CPI with varying IC

Benchmarks *ins(A)* and *minv(A)* exhibit an almost constant CPI with varying IC. The scatter plot of *ins(A)* is shown in Figure 5. The constant CPI is chosen to be f(SWIC). Interestingly $CPI_{chronos}$ also corresponds to f(SWIC) in this case.

(v) Random Cross Correlation

For both architectures, the scatter plot for benchmarks *dij* and *qsort* appear as a random distributed cluster of points to which no functional relationship can be ascribed. Figure 6 shows the scatter plot for *qsort(A)*. There are multiple values of CPI for nearly the same IC. Since we are interested in WCET, we consider only the maximum CPI for each IC. However the covariance matrix elements have been computed retaining all measurements. Considering only the top most points, the distribution now appears to exhibit a parabolic relationship between CPI and IC. We choose the peak point in this case, which is the maximum observed CPI as f(SWIC). Referring to Table 3, *Chronos* gives a more pessimistic estimate for *qsort* on both architectures compared to the proposed method.

Benchmark	$\frac{\widehat{WCET}}{Chronos\_Est}$		$\frac{\widehat{WCET}}{M}$		Class
	A	B	A	B	
crc	0.96	*OOM*	1.04	1.04	
edn	0.74	0.79	1.0	1.02	
fft	0.97	*OOM*	1.0	1.01	
fir	0.73	0.86	1.06	1.05	
jan	0.99	0.99	1.0	1.0	(i)
lms	0.79	0.505	1.0	1.0	
mat	0.94	0.13	1.0	1.00	
ndes	0.83	*OOM*	1.0	1.01	
bitc	0.93	0.99	1.02	1.02	
bub	0.82	0.35	1.01	1.0	
bs	1.0	0.97	1.0	1.01	(ii)
cnt	0.87	0.15	1.0	1.00	
minv	0.78	*OOM*	1.0	1.00	(iv) for arch A, (ii) for arch B
lud	0.77	1.06	5.78	5.7	(iii)
bez	1.0	0.69	1.0	1.0	(iii) for arch A, (ii) for arch B
ins	1.0	0.33	3.6	3.56	(iv) for arch A, (ii) for arch B
nsch	0.75	*OOM*	3.29	5.0	(ii) for arch A, (iii) for arch B
dij	0.99	*OOM*	1.14	6.75	(v)
qsort	0.32	0.2	5.5	5.9	

Table 3: Comparison of proposed method with *Chronos* regarding tightness of WCET.

For *lud*, *nsch* and *qsort*, the proposed method gives a much higher estimate compared to *M*. It needs to be investigated if this is indeed the case or could this arise because of overestimation of SWIC. On an average, the proposed method yields a WCET estimate that is 0.85 times the corresponding *Chronos* estimate for architecture A and 0.6 times the corresponding *Chronos* estimate for architecture B.

4. RELATED WORK

There has been prior research in the study of factors influencing WCET. Colin et al[4] ascertain the influence of cache, branch predictor, pipeline etc on overall estimated WCET. Among various architectural components it identifies the data and instruction caches and their properties like size, organization etc to be having the highest influence on WCET. In this work however, we investigate the relationship between IC and CPI of a program and how it influences WCET estimation.

Bünte et al[12] describe the desirable features of a WCET benchmarking suite tool-set in the context of measurement based analysis. These features are at the program structure level. Our work tries to characterize the benchmarks in terms of variability in IC and CPI and how they correlate to each other. Benchmarks that exhibit more variability in IC and CPI across different inputs are more interesting and challenging for a WCET analyzer than benchmarks that display constant behavior in IC and CPI across inputs.

5. CONCLUSIONS AND FUTURE WORK

We begin by considering execution time, ET as a product of IC and CPI so that WCET is estimated by a product of *Worst case*(IC) and *Worst case*(CPI). Existing methods may be implicitly using worst case values for both IC and

CPI thus resulting in a highly pessimistic estimate. The paper describes a way to improvise the tightness of the estimate. The theoretical upper bound on the program instruction count, SWIC, is obtained by appropriately modifying the integer linear programming framework used by static WCET analyzers. Measurements of IC and CPI of a program for a large number of inputs indicate that there indeed exists a functional relationship between CPI and IC for many of the benchmarks. In lieu of such a relationship, CPI can be predicted for a given IC. The proposed method estimates WCET as a product of SWIC*f(SWIC) to avoid pessimism in both factors IC and CPI. In benchmarks where CPI decreases sharply with an increase in IC, measured maximum cycles is taken to be the WCET.

The proposed method is observed to give a WCET that is 0.85 times the corresponding *Chronos* estimate for an architecture with only an instruction cache. For an architecture with both instruction and data cache, the proposed method gives a WCET that is 0.6 times the corresponding *Chronos* estimate. Apart from helping in WCET estimation, the CPI versus IC relationship also helps in grouping benchmarks into well defined classes such that one from each class may be studied in detail.

This study opens up large number of issues for further investigation. What is the rationale behind the relationship between CPI and IC? Does the derived relationship between CPI and IC keep changing with increasing number of measurements or does it get stabilized? If the values of variance of IC, variance of CPI and their cross-covariance eventually saturate, it would mean that we have considered sufficient number of measurements. We could make use of *Pearson coefficient* or *Eigen value decomposition* to infer if a definite predictable relationship exists between CPI and IC. If need be we could even consider non-linear scales such as semi-log or log-log scale to examine if a definite relationship can be derived.

In this work, the entire program is considered as a single block to infer any relationship between CPI and IC. This may be so because many of the benchmarks considered have either an underlying mathematical algorithm or a clearly well-defined functionality. Large applications may have to be divided into functional blocks to infer a relationship between CPI and IC. Once a relationship is seen to exist between CPI and IC, either a curve fitting or a functional modeling has to be derived to capture the relationship optimally in mean square sense. We are examining many of these issues as on-going research.

6. ACKNOWLEDGMENTS

The authors would like to thank Dr. T. V. Ananthapadmanabha for his valuable suggestions on several aspects of this paper. We would also like to thank the anonymous reviewers for their suggestions and feedback.

7. REFERENCES

[1] "http://www.comp.nus.edu.sg/∼rpembed/chronos/download.html".
[2] "http://www.eecs.umich.edu/mibench".
[3] "http://www.mrtc.mdh.se/projects/wcet".
[4] A. Colin et al. Experimental evaluation of code properties for WCET analysis. In *RTSS*, pages 190–199, 2008.
[5] A. Dupuy et al. An empirical evaluation of the mc/dc coverage criterion on the hete-2 satellite software. In *DASC*, 2000.
[6] G. Bernat et al. pWCET: a tool for probabilistic worst case execution time analysis of real-time systems. In *Technical Report YCS-2003-353*, Univ. of York, UK.
[7] J. Hansen et al. Statistical based WCET estimation and validation. In *ECRTS*, pages 123–133, 2009.
[8] M. Zolda et al. Towards adaptable control flow segmentation for measurement-based execution time analysis. In *RTNS*, 2009.
[9] Matteo Corti et al. Approximation of worst-case execution time for preemptive multitasking systems. In *LCTES*, pages 178–198, 2000.
[10] P. Keim et al. Extending the path analysis technique to obtain a soft WCET. In *ECRTS*, pages 134–142, 2009.
[11] R. Wilhelm et al. The worst-case execution time problem- overview of methods and survey of tools. *ACM Trans. Embedded. Comp. Syst.*, 7(3):36–53, April 2008.
[12] S. Bünte et al. A benchmarking suite for measurement based WCET analysis. In *ICSTW*, pages 353–356, 2008.
[13] V. Suhendra et al. Efficient detection and exploitation of infeasible paths for software timing analysis. In *DAC*, pages 358–363, 2006.

APPENDIX

We now discuss the procedure for obtaining the static worst case instruction count(SWIC) for a given program. Several static WCET analyzers including *Chronos* use integer linear programming to estimate static worst case execution time. We use a similar formulation to estimate SWIC. Our input is the control flow graph (CFG) which is constructed from the program binary. Each basic block, B is associated with an integer variable N_B that indicates B's execution count and an integer constant W_B that indicates the weight of the basic block(number of instructions constituting the basic block B). The linear objective function is given by,

$$Maximize\ \Sigma_{\forall B}(\ N_B * W_B\ ) \tag{7}$$

The parameter N_B is controlled by how the control flows among the edges in the CFG and is constrained by,

$$\Sigma_{B' \to B}(E_{B' \to B}) = N_B = \Sigma_{B \to B''}(E_{B \to B''}) \tag{8}$$

The parameter E_B is bounded by the maximum number of times the loop can iterate, L, if it happens to reside inside a loop else it takes the value *1* by default. We assume availability of loop iteration bounds for all the loops in the CFG. Any instances of recursion are converted to iteration.

$$E_{i \to j} <= L \tag{9}$$

To improve accuracy of WCET, we account for *infeasible paths* and *context sensitivity*. Infeasible paths are identified using the approach proposed by Vivy et al[13] and figure as additional constraints in our linear system of equations. To deal with context sensitivity, we perform procedure cloning if a procedure is called in two different locations and treat each call as a separate one. In this work, we do not perform any context sensitivity analysis at the loop level.