Estimation of Probabilistic Bounds on Phase CPI and Relevance in WCET Analysis

Archana Ravindar Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science
Bangalore-560012, India
{archana,srikant}@csa.iisc.ernet.in

ABSTRACT

Estimating program worst case execution time(WCET) accurately and efficiently is a challenging task. Several programs exhibit *phase behavior* wherein cycles per instruction (CPI) varies in phases during execution. Recent work has suggested the use of phases in such programs to estimate WCET with minimal instrumentation. However the suggested model uses a function of mean CPI that has no probabilistic guarantees. We propose to use Chebyshev's inequality that can be applied to any arbitrary distribution of CPI samples, to probabilistically bound CPI of a phase.

Applying Chebyshev's inequality to phases that exhibit high CPI variation leads to pessimistic upper bounds. We propose a mechanism that refines such phases into sub-phases based on program counter(PC) *signatures* collected using profiling and also allows the user to control variance of CPI within a sub-phase. We describe a WCET analyzer built on these lines and evaluate it with standard WCET and embedded benchmark suites on two different architectures for three chosen probabilities, p={0.9, 0.95 and 0.99}. For p=0.99, refinement based on PC signatures alone, reduces average pessimism of WCET estimate by 36%(77%) on *Arch1* (*Arch2*). Compared to *Chronos*, an open source static WCET analyzer, estimates obtained by refinement are more accurate by 5%(125%) on *Arch1*(*Arch2*). On limiting variance of CPI within a sub-phase to {50%, 10%, 5% and 1%} of its original value, accuracy of WCET estimate improves further to {9%, 11%, 12% and 13%} respectively, on *Arch1*. On *Arch2*, accuracy of WCET improves to 159% when CPI variance is limited to 50% of its original value and improvement is marginal beyond that point.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Measurement, Performance Evaluation

Keywords

phase behavior, CPI, WCET analysis, profiling, soft real-time systems, probabilistic bounds, Chebyshev inequality, confidence intervals

1. INTRODUCTION

The worst case execution time (WCET) of a program is the maximum time a program will ever take to execute on a given architecture. WCET estimates are necessary to design real-time systems where programs have deadlines to adhere to. WCET estimates help build an optimal schedule that ensures effective resource utilization. WCET analysis is non-trivial as it depends on several factors like program structure, input and complexity of the architecture.

Generally, WCET analyzers work on components of a program. Static WCET analyzers[19] estimate WCET of program components on an analytical model of the architecture built for this purpose. Measurement based WCET analyzers[11, 18, 20, 16, 22] measure execution time of these components directly on the architecture either by native execution or simulation. The overall program WCET is estimated by combining these costs using program structural analysis. Statistical WCET analyzers measure end to end execution times and fit models to estimate WCET[21, 12, 24, 23]. Instead of an absolute WCET estimate, one can estimate WCET at various probabilities, especially useful, when tasks with different priorities exist. Bernat et al[11] probabilistically combine worst case effects of basic blocks under three different scenarios and build the program worst case path to estimate probabilistic WCET.

Each WCET analysis technique is applicable in a specific domain and has its own set of concerns. While static WCET analysis guarantees safe WCET estimates, absence of runtime information forces the analysis to make conservative assumptions that might lead to pessimistic WCET estimates. Statistical WCET techniques need to make their model close to the real world as much as possible[10]. A measurements based WCET analyzer might make unsafe estimates of WCET due to incomplete coverage of functions, statements and conditions.

The amount of instrumentation remains a concern in measurement based analyzers which typically measure basic blocks or group of basic blocks of a program[5, 6]. Several programs exhibit *phase behavior* that refers to phase-like vari-

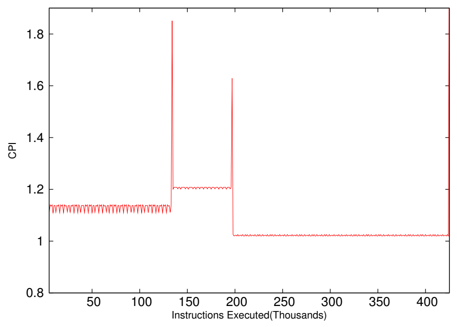


Figure 1: Variation of *Bitcount* over time on *Arch1*.

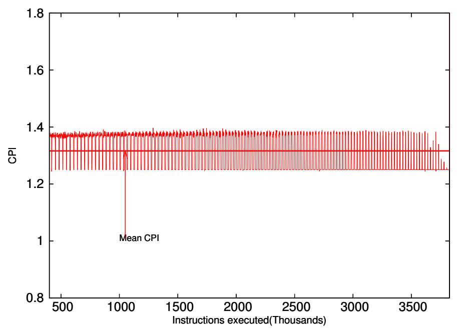


Figure 2: Deviation of CPI around the mean in *Bubble sort* on *Arch1*.

ation of CPI (cycles per instruction) observed during their dynamic execution. In [8], we have demonstrated that program phase behavior can be used to reduce instrumentation in measurement based WCET analysis. Figure 1 plots cycles per instruction(CPI) for every 1000 instructions executed for *Bitcount* (Table 2). Within each phase, CPI varies homogeneously and is distinct across phases as shown. In order to make use of phase behavior, we consider execution time (processor cycles) as a product of instructions executed(IC) and cycles per instruction(CPI).

The distinct CPI behavior of each phase drives the formulation of program WCET as a sum of WCET of it's constituent phases. The homogeneity and repeatability of CPI behavior within a phase helps in obtaining CPI of a phase with minimal instrumentation. Code structural analysis is used to mark phases in the program[14] that hold across different inputs. We instrument programs at every thousand instructions in [8] resulting in an instrumentation ratio of 0.1%. The product of worst case number of instructions executed within a phase, *Max(IC)* and worst case CPI of a phase, *Max(CPI)* is the WCET of that phase. Although the idea is simple and promising, the method uses maximum of mean CPI observed as *Max(CPI)* resulting in an approximate WCET estimate that has no probabilistic guarantees[8].

Our objective is to improve the phase based WCET analyzer to yield WCET estimates associated with probabilistic guarantees. For this purpose, we compute probabilistic upper bound of phase CPI that is multiplied by Max(IC) to yield a probabilistic WCET estimate. For each phase, CPI samples are collected by measurement at numerous points by running benchmarks with a large number of test inputs. The true probability distribution of these CPI samples is not known. We know that the samples have finite mean and finite variance. Hence we use Chebyshev's inequality to bound CPI of a phase within a confidence interval for a probability, p. Applying Chebyshev's inequality to benchmarks with stable CPI behavior (coefficient of variation or CoV of CPI $< 0.5\%$) results in accurate WCET estimates (that are within 1% of maximum observed cycles even at p=0.99).

Some benchmarks like *Bubble_sort* (Table 2) exhibit high variation in CPI during execution(Figure 2). Applying Chebyshev's inequality directly for such phases yields a wide confidence interval for CPI leading to highly pessimistic WCET estimates, as execution time is directly proportional to CPI. It is observed that deviations in CPI correspond to deviations in the program counter even at a granularity of a few tens of instructions. Using this observation, we *refine* such phases into smaller sub-phases based on PC(program counter) signatures, collected using profiling. These signatures basically encode path information of loop iterations in a concise manner and are analyzed to isolate high deviations in CPI. Re-applying Chebyshev's inequality on CPI samples for each sub-phase gives us a tight bound on CPI thereby resulting in an accurate WCET estimate. The refinement process also allows the user to control CPI variance within a sub-phase and hence accuracy of WCET.

For evaluation, we choose two architecture, *Arch1* and *Arch2* as shown in Table 1 with benchmarks taken from Mälardalen WCET project suite[3] and embedded benchmark suite, Mibench[1] (Table 2). All measurements are carried out using cycle accurate simulator, *Simplescalar Version 3.0*[4]. We compute bounds on CPI for unrefined and refined phases respectively at three chosen probability values, p={0.9, 0.95, 0.99}.

For benchmarks that exhibit high variation of CPI, refinement is observed to reduce pessimism in WCET estimates by 36%(*Arch1*) and 77%(*Arch2*), on an average, compared to unrefined phases. Refinement of sub-phases further limiting CPI variance to {50%, 10%, 5%, 1%} of original sub-phase CPI variance is observed to improve accuracy of WCET further, on both architectures, compared to refinement based on PC signature alone. Compared to *Chronos*, at p=0.99, WCET estimates computed using phase refinement based on PC signature and limiting CPI variance of sub-phase to (50%, 10%, 5%, 1%) of original sub-phase CPI variance, are tighter by {15%, 20%, 21%, 23%, 24%} on *Arch1* and by {149%, 159%, 159%, 159%, 159%} on *Arch2* respectively.

We address the following questions in this paper.
1. How can we obtain robust WCET estimates that are also accurate in the phase based timing model?
2. How can we isolate points of high CPI variation within a phase?
3. How can we control CPI variation within a phase?
4. What is the impact of 2 and 3 on WCET accuracy?

The rest of the paper is organized as follows. Section 2 outlines the phase based WCET analyzer. Section 3 describes the basic framework to compute probabilistic bound on CPI of a phase. Section 4 describes phase refinement in detail and how refined phases can be used to estimate WCET. The proposed technique is evaluated in Section 5 and compared with related work in Section 6. The paper is finally concluded in Section 7.

Table 1: Architectural configurations used for experimentation.

Common Parameters	Issue, decode and commit width=1, Register update unit (RUU) size=8, Fetch Queue size=4
Arch1	8KB direct mapped Instruction cache, Out of order Issue, 2-level Branch Predictor
Arch2	In-order Issue, 8KB direct mapped Instruction Instruction cache, 8KB 2-way set associative Data cache, Unified 64KB 8-way associative, L2 cache, Perfect Branch Prediction

2. PHASE-BASED TIMING MODEL

In [8], we propose to estimate program WCET as a sum of WCET of it's phases. A phase corresponds to a static code region detected by code structure analysis[14]. The unit of analysis is a hierarchical call loop(HCL) graph, created out of the program binary. The program is executed with various inputs to ensure coverage of all functions and conditions. Profile data is used to annotate the HCL graph with hierarchical information regarding number of calls, loop iteration counts, variance in instructions executed every time each call/loop is executed. The HCL graph is analyzed to pick phase marker edges. The code region lying between a marker edge *e1* and the following marker edge *e2* comprises the phase associated with *e1*.
The WCET of a program is estimated as,

$$WCET = \Sigma_{i \in \{1..p\}} WCET_i \tag{1}$$

where p is the number of phases of the program.
WCET of the i-th phase, is estimated as,

$$WCET_i = T_i \times Max(IC_i) \times Max(CPI_i) \tag{2}$$

where, $\mathbf{T}_i$**:** Maximum number of times phase i occurs during execution.
Max(IC$_i$): Worst case instruction count(IC) of phase i. *Max(IC$_i$)* is either the theoretical upper bound on IC derived using static analysis(SWIC$_i$) or the maximum observed IC of phase i(MIC$_i$)[8].
Max(CPI$_i$): Worst case CPI of phase i. CPI is measured within each phase at various points and maximum of mean CPI across all tested inputs is taken as *Max(CPI$_i$)*. In this work, we use probabilistically bounded CPI instead of maximum of mean CPI to obtain a more robust WCET estimate.

A program depending on its structural complexity, can execute different code regions(phases) on execution with different inputs thereby exhibiting multiple phase sequences across inputs[8]. In that case, WCET is estimated as the maximum among WCET of all possible sequences.

3. COMPUTING BOUNDS ON PHASE CPI

We now describe how CPI of a phase is bounded for a given probability, p. To bound CPI, we collect n CPI samples for each phase(static code region) to form the sample set, $\widehat{S_i}$, by running the program with a large number of test inputs. CPI is measured at intervals ranging from 100 to 1000 instruction depending on the program dynamic execution length.

On an average, CPI samples are observed to be within 10% of the sample mean($\widehat{\mu}$) on both architectures. Our main objective is to quantify the amount by which a future CPI sample can be away from $\widehat{\mu}$ for a given probability(p). Had we known the true probability distribution of the samples (ascertained only if true population set, S$_i$, built by exercising *all* paths within phase i is known), we could apply an appropriate probability density function to compute the confidence interval to contain a future CPI sample for probability p. Building S$_i$ is computationally expensive. Hence we use Chebyshev's inequality as it can be applied to any arbitrary distribution. Chebyshev's inequality only requires the random variable(CPI) to have finite mean and finite variance. If variance is small, bounds obtained using Chebyshev's inequality are tight.
Chebyshev's inequality: The inequality states that p, probability of a future sample, cpi$_x$, being greater than mean of S$_i$(μ), is as follows,

$$P(|cpi_x - \mu| \geq C) \ \leq \ \frac{\sigma^2}{C^2} \tag{3}$$

Where, C is an arbitrary constant, μ is true mean of the distribution and σ^2 is true variance of the distribution. We can use sample mean, $\widehat{\mu}$ and sample variance, $\widehat{\sigma}^2$ in Eq.3 provided variance of sample mean, Var($\widehat{\mu}$) is small.

$$Var(\widehat{\mu}) \ = \frac{\sigma^2}{n} \tag{4}$$

Var($\widehat{\mu}$) is inversely proportional to the number of samples, n. Hence with increasing n, Var($\widehat{\mu}$) decreases[25]. Since we have a large number of samples, we can confidently use $\widehat{\mu}$ and $\widehat{\sigma}^2$ in place of μ and σ^2 to give the following equation.

$$P(|cpi_x - \widehat{\mu}| \geq C) \ \leq \ \frac{\widehat{\sigma}^2}{C^2} \tag{5}$$

Applying Chebyshev's inequality to $\widehat{S_i}$, we obtain a confidence interval [CPI$_{i,l}$, CPI$_{i,u}$] within which a future CPI sample will lie with probability p. As WCET is to be estimated, we use the upper bound of the interval, CPI$_{i,u}$, which for probability p, is referred as *PrCPI*$_\mathsf{p}$. Hence original timing equation, Eq.2 is modified to,

$$WCET_i = T_i \ \times \ Max(IC_i) \ \times \ PrCPI_\mathsf{p} \tag{6}$$

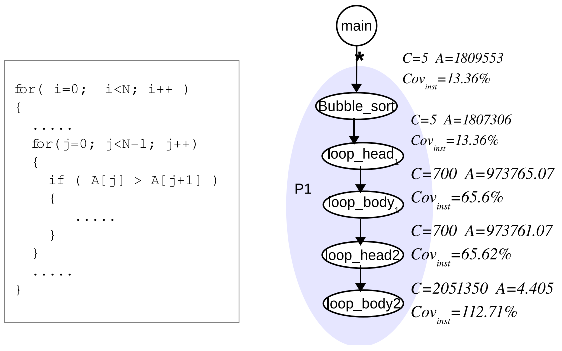
Since we use theoretically bounded *Max(IC$_i$)*, which is a constant and is multiplied by *PrCPI*$_\mathsf{p}$, the probabilistic guarantee applies to the resultant WCET as well. For benchmarks that exhibit low variance in CPI, Chebyshev's inequality tightly bounds phase CPI. Applying the inequality to phases with high variance in CPI results in a wide confidence interval leading to higher *PrCPI*$_\mathsf{P}$ values and hence pessimistic WCET estimates. In the next section, we will see how such phases can be divided into smaller sub-phases to obtain tighter CPI bounds and hence tighter WCET estimates.

4. PHASE REFINEMENT

Code structure analysis[14] ensures that variation in instructions executed within a phase is much lesser than the corresponding variation across different phases. However, presence of if-conditions in loops or calls can cause high variance in instructions executed across loop iterations or call invocations. The if-condition of the inner-loop in *Bubble_sort* (Figure 3), when *true*, executes additional code compared to when the condition is *false*. The HCL graph for the routine created by profiling with a set of 5 different inputs, is shown in the same figure. Each loop is represented by a loop head

Table 2: Benchmarks and their characteristics.

Benchmark	Description	Phase Sequence	Static Length	Avg. Dynamic Length
Bezier (*Bez*)	Draws a set of 200 lines of 4 reference points on a 800×600 image.	P1 P2	114	107901512
Bitcount (*Bit*)	Performs bit operations on a 1K bit-vector a thousand times[1].	P1 P2 P3 P4 P5 P6	257	404910
Binary Search (*Bs*)	Search for a key in a 50K number vector[3].	single	51	6329
Bubble sort (*Bub*)	Sort an array of size 3K[3].	single	55	41472125
CNT (*Cnt*)	Counts positive numbers in a 200×200 matrix[3].	P1 P2	72	653672
CRC (*Crc*)	Cyclic redundancy check on a 16KB char vector[3].	P1 P2	84	583140
FIR (*Fir*)	Finite impulse response filter over a signal of size 400[3].	single	272	148171
FFT (*Fft*)	Fast fourier transform on a wave of size 16K[1].	P1 P2	277	8508909
Insertion Sort (*Ins*)	Sort a 3K number vector[3].	single	38	24899477
Janne_complex (*Jan*)	A nested loop program, *a*, *b* are input parameters	single	35	1201362
LMS (*Lms*)	Adaptive signal enhancement[3].	single	142	567565
Matmul (*Mat*)	Matrix multiplication of two 200×200 matrices[3].	P1 P2	106	105090836



```
for( i=0;  i<N; i++ )
{
  .....
  for(j=0; j<N-1; j++)
  {
    if ( A[j] > A[j+1] )
    {
       .....
    }
  }
  .....
}
```

Figure 3: Code structure of *Bubble_sort* routine and its HCL graph.

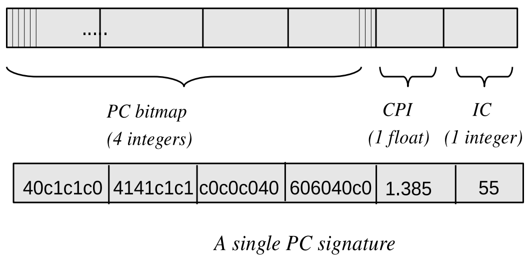


Figure 4: Format of a single PC signature.

node and a loop body node in the HCL graph. The edge associated with loop head node stores information about the number of times loop head was executed(**C**), hierarchical average number of instructions(**A**) and CoV in instructions executed($\mathbf{CoV}_{inst}$) over different executions. Similarly the edge associated with loop body node stores these information pertaining to a loop iteration. It can be observed that loop edges have a very high CoV_{inst} hence do not qualify as software markers resulting in the whole routine being selected as a single phase[14]. Considering the entire loop as a single phase exhibits high variation in CPI(Figure 2).

In such cases, we could statically mark the code region associated with each iteration and hence each path per iteration as a different phase. But that would not work because,
a) The underlying architecture is based on a pipeline consisting of several stages. Multiple instructions are in flight at the same time. A phase should be lengthy enough to allow at least few instructions to completely execute to facilitate calculation of CPI of the phase.
b) Instrumenting every few instructions can hamper performance of the program that we are trying to measure.
Hence it appears that we need to find a mid point where phases are big enough, at the same time, small enough to obtain tight bounds on CPI.

An intuitive approach is to consider every x consecutive iterations of a loop, L, as a potential phase, which we term as a *window*, W. We emit dynamic execution information of every window, W, in the form of a triple, defined as a *PC signature* (Figure 4), storing the following information.
PCbitmap: The bitmap is a vector of 4 integers (128 bits). The simulator hashes every instruction PC encountered and stores it into the bitmap. 127 bits are sufficient to map PC addresses in each phase of the benchmarks considered in this paper.
CPI: CPI represents observed cycles per instruction while instructions belonging to W are executed.
IC: IC represents number of instructions executed that belong to W.

4.1 Refinement Based on PC Signature

Refinement consists of three steps: trace generation, trace compression and classification of compressed trace into subphases.
1) Trace Generation: In order to generate a trace, we first identify the branch instruction that iterates loop L of the phase. If L is nested with several levels, we select the innermost loop. The simulator is modified to count x consecutive executions of the branch instruction of L. If *Min(*$|L_i|$*)* denotes the minimum number of instructions executed in each loop iteration i of L, number of iterations that make up a single window, x is defined as,

$$x = \left\lceil \frac{Phase\_length}{Min(|L_i|)} \right\rceil \quad (7)$$

where, *Phase_length* is the number of instructions that make up a phase. On a given architecture, *Phase_length* should be greater than the minimum number of instructions that have to be executed for at least one instruction commit. A *Phase_length* of *50* instructions suffices for both architectures considered in this paper (Table 1). x being a ceiling value, Eq.7 might cause some windows to be composed of more than *50* instructions. x will not always be an exact multiple of $|L|$. Hence the execution time of the last few

Trace of a single run of Bub

```
   PC Bitmap                                              CPI        IC
..........
202020002020202020                                     1.250000     64
202020002020202020                                     1.250000     64
202020002020202020                                     1.250000     64
202020002020202020                                     1.250000     64
202020002020202020                                     1.250000     64
20202020202020203030202030303030                       1.451610     93
2020200020202020202020                                 1.291140     79
202020002020202020                                     1.250000     64
202020002020202020                                     1.250000     64
202020002020202020                                     1.250000     64
202020002020202020                                     1.250000     64
..........
```

Compressed trace of a single run of Bub

```
   PC Bitmap                                     CPI        IC   #duplicates
..........
202020002020202020                            1.250000     64    147
20202020202020203030202030303030              1.451610     93    1
2020200020202020202020                        1.291140     79    1
202020002020202020                            1.250000     64    93
..........
```

Figure 5: Signature trace of a single run of *Bubble sort* and its compressed version

iterations will have to be added separately. If the phase has multiple loops, the same procedure is repeated for all loops within the phase.

A loop with small $|L_i|$ will have windows comprising of a large number of iterations. If $|L_i|$ is greater than minimum *Phase_length*, every iteration forms a window. If $|L_i|$ is well beyond minimum *Phase_length*, we can use code structure analysis to break it into smaller phases. The benchmarks considered in this paper comprises of loops where $|L_i| \leq$ minimum *Phase_length*. The cycles taken by code preceding loop L of phase P, if any, is added separately.

When a program is simulated by *Simplesim-3.0* modified as explained above, we obtain a trace consisting of $\frac{|L|}{x}$ such signatures, where $|L|$ is the loop iteration count of L. The modifications to simulator does not impact execution cycles of the program as PC values are read off the pipeline and processed in parallel. In the worst case, for every 50 instructions executed, a trace comprising of 6 words, is emitted out and the 4-word hash table is reset.

An integer vector that stores occurrences of each PC encountered would be more accurate to represent path information of every window, instead of the existing bitmap. But that would clearly not scale with x and would lead to huge traces. The bitmap is imprecise as we shall now see with an example.

In Figure 3, assume the inner loop executes 18 instructions when if-condition evaluates to *true* and 10 instructions when it evaluates to *false*. Let window size, x be 4 iterations. Consider two such windows W_1 and W_2. Assume in W_1: if-condition is *true* once and false three times. In W_2, if-condition was *true* three times and *false* once. The PCbitmap will be identical in both cases but IC(W_1) = 18 × 1 + 10 × 3 = 48. IC(W_2) = 18 × 3 + 10 × 1 = 64. Hence IC serves to store extra information without bloating the trace. Although seeming imprecise, the combination of IC and PCbitmap is observed to be sufficient to isolate high CPI variations in most cases.

2) Trace Compression: Lengthy program runs can produce megabytes of trace. But they are easily compressible owing to the repetitive nature of phases. A large number of consecutive windows have identical PC signatures which can be compressed(Figure 5). We look for consecutive triples that repeat to compress them. The time complexity of the compression algorithm is linear to the trace size.

3) Trace Classification: A one-to-one correspondence is observed between <PC-bitmap, IC> and CPI in the trace for program *Bubble sort* (Figure 5), which is observed in

```
/****************   Refine sub-phase based on CPI *************/
/* Inputs:                                                      */
/* <CPI data for each sub-phase>                                */
/* <Variance_threshold>                                         */
/* Output:                                                      */
/* <New Bounds on CPI for each new sub-phase>                   */
/***************************************************************/

Procedure Split(sub-phase_CPI_vector)
 Compute variance of CPI;
 While (variance > Variance_threshold) do
   Split sub-phase_CPI_vector into two depending on range of values
 /* vector_1 range: [lower..mean] vector_2 range: [mean..upper] */
   Split(vector_1);
   Split(vector_2);
  end for
end Split
Procedure Bounds(sub-phase_CPI_file)
 Compute mean of CPI;
 Compute Variance of CPI;
 Compute Chebyshev bounds on CPI;
end Bounds
Procedure main
 for each sub-phase, i, do
  Split(i); // generates new sub-phases
  for each new-sub-phase, j, do
   Bounds(j);
  end for
end for
end main
```

Figure 6: Algorithm to refine sub-phase based on CPI variance.

other benchmarks as well. This happens because CPI is largely determined by the instructions that execute[14]. Based on this, we define a *sub-phase* as a unique pair of <PCbitmap, IC> values. All windows with the same <PCbitmap, IC> value belong to one sub-phase. For each such sub-phase, new confidence intervals are computed by applying Chebyshev's inequality on CPI samples pertaining to that sub-phase.

The time taken by the classification algorithm is O(m × n), where m is the number of unique <PCbitmap, IC> pairs (sub-phases) and n is the number of entries in the compressed trace. On an average, number of sub-phases, m, detected for benchmarks used in this paper is 15.04(15.33) for *Arch1*(*Arch2*) even if number of windows for some benchmarks go upto a few thousands. The size of compressed trace obtained across all inputs for a program, n, ranges from 2 MB to 1.8GB. The average sub-phase size observed across all benchmarks is 62 resulting in an average instrumentation overhead of 1.6%.

4.2 Refinement Based on CPI Variance

Inspite of refinement based on PC signature, certain subphases exhibit high variance of CPI. Hence we add another level of refinement wherein the user can control the variance of CPI within the sub-phase. The classification will now be based on <PCbitmap, IC, CPI-range>. The procedure repeatedly splits the sub-phase until the CPI values fall in the desired range giving rise to variance well within the specified limit(Figure 6). The time complexity of *Split* is O(n × *log(n)*), where n is the number of entries in sub-phase CPI file. The overall time complexity of the refinement procedure is O(m × n × *log(n)*) where m is the number of original sub-phases.

4.3 WCET Estimation Using Sub-Phases

A phase represents a static code region. Whereas a subphase represents a group of consecutive loop iterations. Every single loop iteration is included in the analysis. Subphases do not overlap as each of them represent a different group of loop iterations. In order to estimate WCET in terms of sub-phases, Eq.6 has to be suitably modified. Different phases can occur on execution with different inputs[8]. The same holds for sub-phases. The set of sub-phases that occur for a particular program run with input i forms a sub-phase sequence $\mathcal{S}_i$. Note that we are not interested in the exact order in which sub-phases occur.

A sub-phase sequence ($\mathcal{S}_i$) obtained with input i, takes the form of an integer vector, $[s_{i.0},\ s_{i.1},\ \ldots\ ,\ s_{i.sp}]$ where sp is the total number of sub-phases appearing across all inputs. $s_{i.j}$ indicates the number of times sub-phase j occurs in the execution run of program with input i. Among two sequences, $\mathcal{S}_a$ and $\mathcal{S}_b$, obtained with inputs a and b, such that $s_{a.k} \geq s_{b.k}\ \forall$ k = {0,..,sp}, we include only $\mathcal{S}_a$. The number of unique sub-phase sequences that can occur range from 1 to over a hundred.
For each sequence, $\mathcal{S}_i$, $WCET_i$, is estimated as-

$$WCET_i \ = \ \Sigma_{j\in\{0,..,sp\}}\ (s_j \ \times \ Max(IC_j) \ \times \ PrCPI_{\mathsf{p}}) \quad (8)$$

s_j is the sub-phase counterpart of T$_i$(in Eq.6). Since sub-phase is a dynamic entity, we use maximum observed IC in a window, occurring for the bitmap corresponding to sub-phase j, across all inputs as $Max(IC_j)$. For s possible sequences, overall WCET is estimated as,

$$WCET \ = \ max(WCET_1, ..., WCET_s) \qquad (9)$$

Equation 8 applies to loops that iterate the same number of times for all inputs. However, the loop can terminate sooner than intended depending on data. It is hence useful to compute WCET in a situation when iterations reach the loop bound. For this purpose, we calculate the theoretical upper bound on the number of windows, SWW, possible for a given loop making up a program phase. If $|L|$ denotes the loop bound of L, x denotes the number of iterations per window, SWW is computed as,

$$SWW \ = \frac{|L|}{x}$$

For each unique sub-phase sequence, $\mathcal{S}_i$, we calculate the weight of each sub-phase, k, occurring in that sequence as,

$$w_k \ = \ \frac{s_{i,k}}{\Sigma_{j\in\{0,..,sp\}}\, s_{i,j}}$$

And consequently, $WCET_i$ is estimated as,

$$WCET_i \ = \ \Sigma_{j\in\{0,..,sp\}}\ (w_j \times SWW \times Max(IC_j)\times PrCPI_{\mathsf{p}}\,) \quad (10)$$

4.4 Context Sensitivity

An analysis of a program fragment is said to be context sensitive if it takes into account the context in which the fragment appears. Context sensitive analysis has been observed to improve precision of WCET analysis significantly[22]. Context sensitive analysis is typically applied for procedures and loops. In this paper, we treat a procedure appearing in two different contexts as two different procedures. It is observed that the first iteration of a loop takes more time to execute (greater CPI) than rest of the iterations[17]. Hence we treat the first window (that includes the first iteration) of a loop as a separate sub-phase.

5. EVALUATION

All our experiments are performed on benchmarks taken from *Mibench* and Mälardalen standard WCET project benchmark suite(Table 2), for architectures mentioned in Table 1. The benchmarks are compiled to MIPS PISA binaries with -O2 -static flags. *Simplescalar Version 3.0* is used to obtain CPI samples and generate traces of PC signatures with

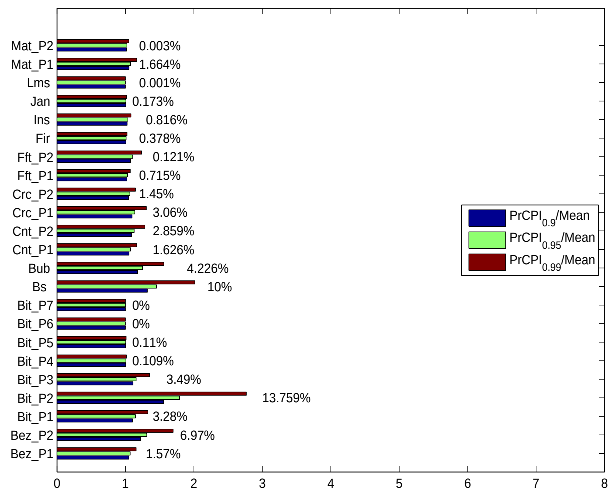


Figure 7: Ratio of probabilistic CPI upper bound to mean CPI at p={0.9, 0.95, 0.99} on *Arch1*. Percentages indicated next to bars refer to coefficient of variation of CPI.

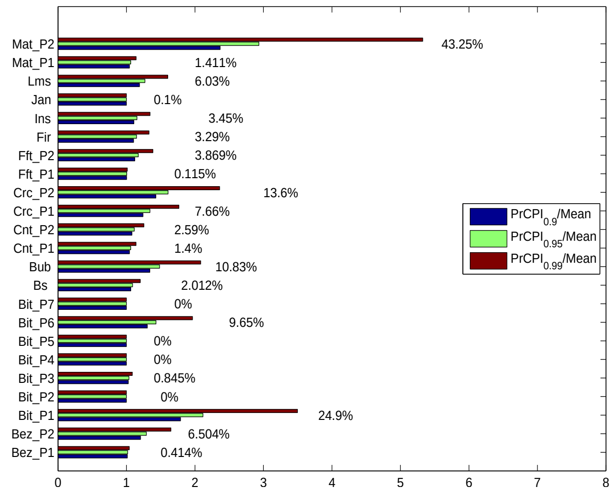


Figure 8: Ratio of probabilistic CPI upper bound to mean CPI at p={0.9, 0.95, 0.99} on *Arch2*. Percentages indicated next to bars refer to coefficient of variation of CPI.

modifications described in Section 4.1. Input selection is done primarily on the basis of MC/DC coverage criteria. Randomly generated inputs are also used. Each (benchmark, input) pair is executed with 500 different inputs multiple number of times to model different initial states[7] and atleast one million CPI samples per phase are generated. Invalid inputs and inputs that terminate execution early are not considered for analysis. The resulting estimates are compared with the open source static WCET analyzer *Chronos* as it models the MIPS architecture.

5.1 Impact of Coefficient of Variation of CPI on Probabilistic Upper Bound of CPI

Chebyshev's inequality yields tight CPI bounds for phases that exhibit low CoV(CPI) as can be seen from Figures 7 and 8 which plot PrCPI$_{\mathsf{p}}$ at p={0.9, 0.95, 0.99}, normal-

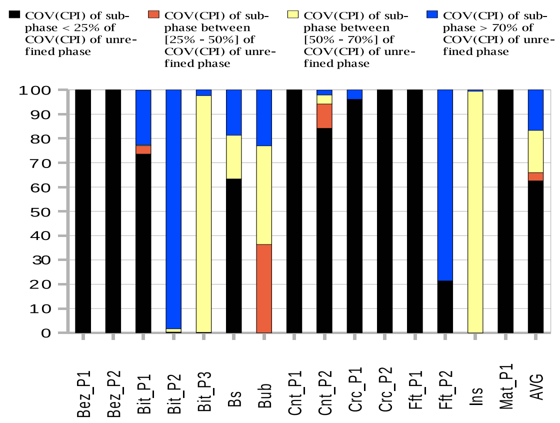


Figure 9: *Arch1:* Percentage breakup of sub-phases based on CoV(CPI).

ized to the mean CPI, for all program phases on *Arch1* and *Arch2* respectively. However, applying the inequality directly to phases like *Mat_P2*(*Arch2*) with high CoV(CPI) results in pessimistic upper bounds of CPI, as can be seen from Figure 8. Hence we need to refine such phases into smaller sub-phases. This will reduce CPI variance within a sub-phase and help yield tighter CPI bounds.

5.2 Impact of Refinement on Coefficient of Variation of CPI

We now compare sub-phases obtained using refinement based on unique <PCBitmap, IC> pairs with the corresponding unrefined phase based on their CoV(CPI). Figures 9 and 10 group sub-phases into four categories as shown. The breakup of only those unrefined phases that exhibit high CPI variance is shown. Post refinement, 63%(87%) of sub-phases exhibit CoV(CPI) that is less than 25% of the corresponding unrefined phase CoV(CPI) on *Arch1*(*Arch2*). Sub-phases of *Bit_P2(Arch1)*, *Fft_P2*, *Bit_P6 (Arch2)* and *Ins(Arch2)* continue to exhibit high CoV(CPI). Such sub-phases are further refined into smaller sub-phases based on CPI variance as outlined in Section 4.2.

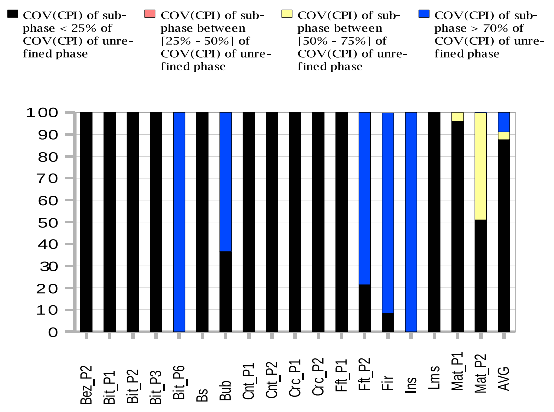


Figure 10: *Arch2:* Percentage breakup of sub-phases based on CoV(CPI).

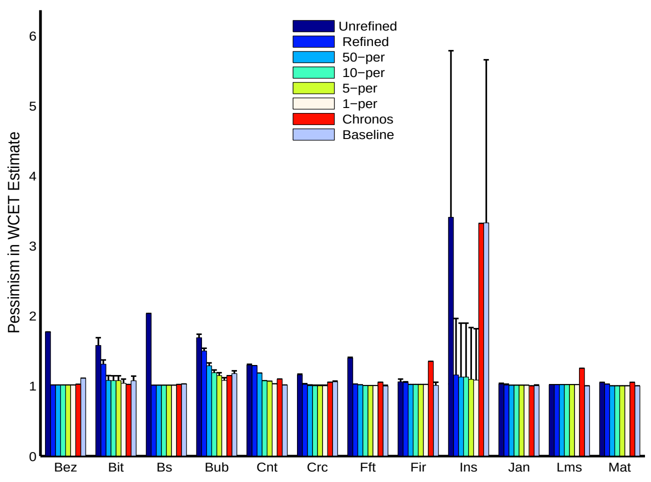


Figure 11: Comparison of WCET estimates using proposed method with *Chronos* and Baseline estimates on *Arch1*.

5.3 Accuracy of WCET

Figures 11 and 12 plot the ratio of estimated WCET to maximum observed cycles (*Pessimism in the WCET estimate*) observed when the proposed phases/sub-phases are used for p=0.99. *Unrefined* and *Refined* bars represent the pessimism observed using unrefined phases and phases refined based on PC signature respectively. The *50-per*, *10-per*, *5-per* and *1-per* bars indicate pessimism in WCET obtained using refined sub-phases with variance of CPI limited at {50%, 10%, 5% and 1%} of CPI variance of original sub-phase respectively. Alongside the bars, WCET estimated by *Chronos* (*chronos_WCET*) and our original phase based model (Eq.2)[8] (*Baseline*) are also plotted.

The theoretical maximum IC executed coincides with maximum observed IC for benchmarks with straight line code. However they differ for programs with complex conditions and loops. The bars in Figures 11 and 12 are plotted using maximum observed IC (for phases) and maximum observed windows (for sub-phases). Similarly the upper limit of the bar is plotted using theoretical upper bound on IC (for phases) and theoretical maximum windows (for sub-phases). CoV(CPI) for benchmarks like *Fir(Arch1)*, *Lms(Arch1)* and *Jan(Arch2)* is less than 1%. As a result, the phase CPI bounds obtained by Chebyshev's inequality are tight enough and improvement by refinement is very marginal.

Applying Chebyshev's inequality to phases with high variation of CPI leads to pessimistic unrefined WCET estimates as shown in Figures 11 and 12. Refinement based on PC signature reduces pessimism considerably(Table 3). The reduction is less on *Arch1* as CPI variation is more scattered possibly due to an out-of-order pipeline and a realistic branch predictor. Refinement based on CPI variance continues to reduce pessimism further. The average improvement in accuracy of WCET estimate compared with *Chronos* for p=0.99 is also shown. As expected, Eq.10 gives a more pessimistic WCET as it uses theoretical upper limit of windows.

Figure 13 plots level of refinement needed to reach a point of zero CPI variance in every sub-phase of the benchmark. Refinement beyond this point will not improve accuracy

Table 3: Impact of Refinement on pessimism of WCET and comparison with *Chronos*

p	100-per	50-per	10-per	5-per	1-per
(Arch1)					
% Reduction in pessimism compared to unrefined WCET					
0.9	20.13	21.81	22.3	22.66	23.25
0.95	23.3	25.64	26.39	26.82	27.56
0.99	35.74	40.63	42.48	43.19	44.82
% Average Pessimism of all refined estimates using Eq.8					
0.99	11.99	6.41	4.53	3.86	2.56
% Average Pessimism of all refined estimates using Eq.10					
0.99	19.84	13.85	11.97	11.06	9.63
% Improvement in accuracy compared to Chronos using Eq.8					
0.99	15.46	20.02	21.56	22.63	24.02
% Improvement in accuracy compared to Chronos using Eq.10					
0.99	4.91	9.25	10.75	11.57	12.84
(Arch2)					
% Reduction in pessimism compared to unrefined WCET					
0.9	35.83	42.23	42.37	42.37	42.37
0.95	44.37	52.46	52.65	52.65	52.65
0.99	77.11	92.09	92.5	92.5	92.52
% Average Pessimism due to refinement using Eq.8					
0.99	20.66	4.82	3.97	3.97	3.96
% Average Pessimism due to refinement using Eq.10					
0.99	31.35	11.63	11.23	11.23	11.22
% Improvement in accuracy compared to Chronos using Eq.8					
0.99	148.88	191.62	191.62	191.62	191.65
% Improvement in accuracy compared to Chronos using Eq.10					
0.99	125.5	158.97	158.97	158.97	159

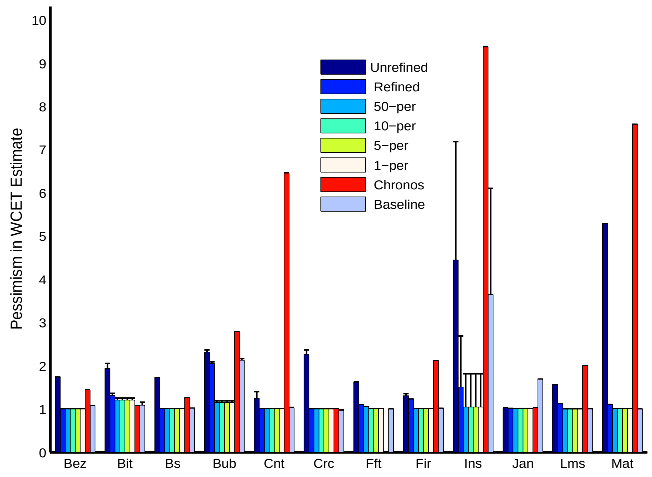


Figure 12: Comparison of WCET estimates using proposed method with *Chronos* and Baseline estimates on *Arch2*.(**Chronos goes out of memory while analyzing *Fft(Arch2)*.**)

of WCET. The benchmarks falling under the grey band have accurate WCET estimates either without refinement or when refined based on PC signatures alone and hence not considered for refinement based on CPI variance. *Bubble sort*, *Bitcount* and *Cnt*(*Arch1*) continue to show variance in CPI even beyond a point when CPI variance is limited to 1% of CPI variance of the original sub-phase. With CPI variance of a sub-phase limited to 10% of original sub-phase CPI variance, 4 out of 9(5 out of 7) benchmarks reach the point of maximum WCET accuracy on *Arch1*(*Arch2*).

The WCET estimates obtained by different kinds of refinement as described in the paper are evaluated with respect

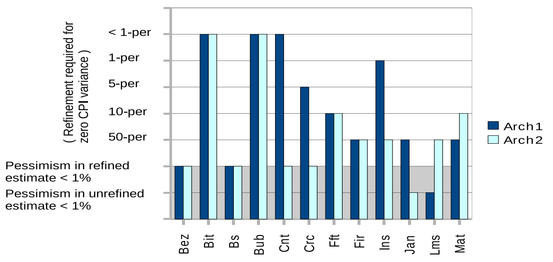


Figure 13: Amount of refinement required to reach zero variance of CPI within a sub-phase.

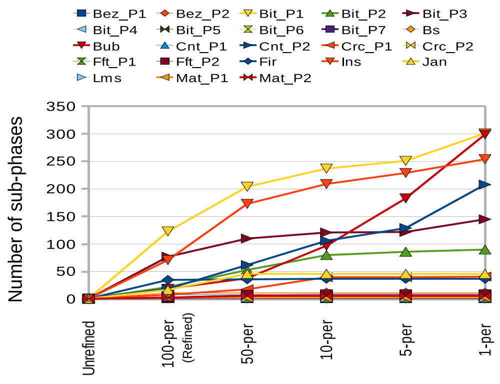


Figure 14: Impact of refinement on number of sub-phases on *Arch1*.

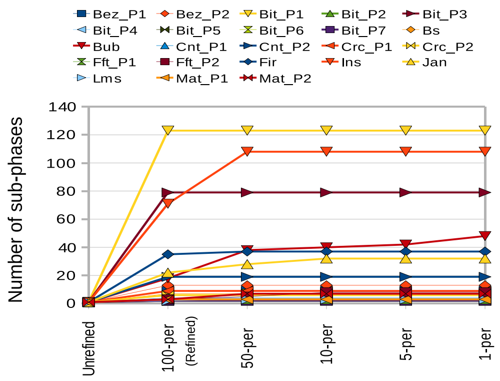


Figure 15: Impact of refinement on number of sub-phases on *Arch2*.

to *safety* by running the benchmarks with a new set of 1000 inputs that were not considered for estimating CPI bounds. None of the estimates (refined based on PC signature, refined based on CPI variance) fall below maximum observed cycles at p={0.9, 0.95 and 0.99} on both architectures.

5.4 Impact of Refinement on Sub-phases

Refinement splits a phase into smaller sub-phases based on PC signature. When a sub-phase is refined based on CPI variance, many more smaller sub-phases are generated. Figures 14 and 15 plot the increase in number of sub-phases due to refinement based on PC signature(indicated by *100-per(Refined)* and refinement based on CPI variance (*50-per* to *1-per*). Number of sub-phases reaches a saturation point for 69%(96%) of phases by the time CPI variance of a sub-phase is limited to 50% of CPI variance in the sub-phase obtained by refinement based on PC signature alone on *Arch1*(*Arch2*). The reason could be that *Arch2* has an *in-order* processor with perfect branch prediction.

5.5 Compression

Table 4 compares the average sizes of trace obtained across all inputs before and after compression on both architectures. On an average, traces are compressed by a factor of 24.86(24.21) on *Arch1*(*Arch2*).

6. RELATED WORK

Program Phase Behavior: In this paper, we extend the phase based WCET analyzer that we proposed in [8] to use probabilistically bounded phase CPI to obtain more robust WCET estimates. The earlier model uses a function of mean CPI which has no probabilistic guarantees and hence underestimates WCET for a few programs[8]. We also profile PC, CPI and IC of loop iterations to refine a phase with high CPI variation into smaller sub-phases. Phases described in [8] are architecture independent. The minimum sub-phase size in our case is architecture dependent. Although the PC bitmap which is determined by code structure is independent of the architecture, number of instructions executed within each sub-phase and CPI of a sub-phase is determined by the architecture.

Davies et al[9] record instruction pointers encountered during execution in the form of an integer vector(EIP) that are classified into phases based on grouping of EIP values. However their purpose is to have minimal error in estimating phase CPI by collecting least number of samples in a phase to reduce simulation effort. We consider all CPI samples in a phase, our objective being WCET estimation. Moreover we benefit from a large sample set, as confidence of Chebyshev bounds increases with the number of CPI samples(Eq.4). The phases in [9] are large(>=100 million instructions) compared to our phases (50-100 instructions). The existence of phase behavior at different levels was first studied by [13] that use *sequitur* to classify the trace consisting of loop branch, procedure call and return instructions in the context of reducing simulation effort. In addition to PC Bitmap and IC of loop iterations, we classify phases based on measured CPI as well.

Measurement Based WCET Analysis: Measurements are taken either at the whole program level or at the level of basic blocks[11, 16, 22], program segments[18, 17] or paths[20]. Measurement usually generates a timing trace from which cost of each component is derived. These costs are combined using structural analysis and techniques like IPET[19] to estimate WCET. The location of instrumentation points influence trace size and accuracy of WCET estimate considerably[6]. Further, instrumentation should be least intrusive. The repetitive manner in which CPI varies in programs that exhibit phase behavior can be used to reduce instrumentation required in WCET analysis of such programs. We propose to instrument at the level of groups of loop iterations leading to a low instrumentation overhead of 1-2%. The number of iterations per group can be varied as per the requirement. Phases also help in compressing PC signatures considerably(Table 4). The issue of adequate program coverage is equally important in this work as it is for any other measurement based WCET analyzer.

Table 4: Average trace size across inputs before and after compression.

Phase	Trace size before compression	Trace size after compression		Compression factor	
		Arch1	*Arch2*	*Arch1*	*Arch2*
Bez_P1	1.8M	92K	68K	20:1	27:1
Bez_P2	50M	504K	312K	101:1	27:1
Bit_P1	24K	24K	24K	1:1	1:1
Bit_P2	48K	46K	23K	1.04:1	2.4:1
Bit_P3	160K	104K	96K	1.53:1	1.66:1
Bit_P4	28K	8K	8K	3.5:1	3.5:1
Bit_P5	32K	8K	8K	4:1	4:1
Bit_P6	32K	8K	8K	4:1	4:1
Bit_P7	32K	8K	8K	4:1	4:1
Bs	8K	8K	8K	1:1	1:1
Bub	23M	17M	11M	1.35:1	2.09:1
Cnt_P1	180K	28K	24K	6.42:1	7.5:1
Cnt_P2	212K	24K	24K	8.83:1	8.83:1
Crc_P1	8K	8K	8K	1:1	1:1
Crc_P2	780K	12K	8K	65:1	97.5:1
Fft_P1	220K	8K	20K	27.5:1	11:1
Fft_P2	220K	8K	20K	27.5:1	11:1
Fir	28K	8K	8K	3.5:1	3.5:1
Ins	21M	408K	328K	52.7:1	65.5:1
Jan	904K	4K	4K	223:1	223:1
Mat_P1	316K	48K	36K	6.58:1	8.77:1
Mat_P2	75M	5.2M	3.6M	14.4:1	20.8:1

Statistical WCET Analysis: Bernat et al[11] measure execution time of basic blocks(execution time profiles or ETPs) and note their relative frequencies. The ETPs are convolved together to give probabilistic WCET estimates using three different scenarios- ETPs are mutually independent, ETPs are dependent, dependency is not known. The phase based timing model views execution time as a product of instruction count(IC) and CPI and estimates program WCET in terms of phases instead of blocks, instructions, segments or paths. We use probabilistic bounds on phase CPI to compute WCET of a phase.

Edgar et al[21], Hansen et al[12] and Lu et al[24, 23] work with end to end program execution time samples and try to fit these samples into a *Gumbel* distribution using extreme value theory(EVT). Once the parameters of the distribution are computed, the estimate of WCET at various probabilities is available. Our work neither assumes any probability distribution of CPI samples nor tries to fit these samples into any distribution. We use Chebyshev's inequality that is applicable to *any* distribution, to compute bounds on CPI. The precision of our results will definitely improve if information regarding *true probability distribution* of CPI samples is available.

7. CONCLUSIONS AND FUTURE WORK

The repetitive manner in which CPI varies in programs exhibiting phase behavior can be used to reduce instrumentation in WCET analysis of such programs We propose a basic model in [8] that uses maximum of mean CPI observed across inputs to estimate WCET. However, WCET estimated thus, is approximate and has no probabilistic guarantees. In this paper, we extend this model to use probabilistically bounded phase CPI. Using CPI bound along with maximum IC results in a robust WCET that can be estimated at the desired probability. The proposed method assumes no probability distribution of CPI samples and uses Chebyshev's inequality to compute bounds of CPI. The accuracy of CPI bound will certainly improve if the true probability distribution is known. Chebyshev's inequality works well with phases that exhibit low variance in CPI resulting in tight CPI bounds and accurate WCET estimates (Examples: *Fir(Arch1)*, *Jan(Arch2)* and *Lms(Arch1)*). Some phases exhibit high variance in CPI. Applying Chebyshev's inequality for such phases results in pessimistic WCET estimates (*Mat(Arch2)*). To isolate points of high variation in CPI, we refine such phases into smaller sub-phases based on PC signatures collected using profiling. We observe the following results for p=0.99. Refinement reduces pessimism by 36%(77%) on *Arch1*(*Arch2*). Using maximum observed windows, accuracy in refined WCET estimate improves by 15%(149%) on an average compared to *Chronos* on *Arch1* (*Arch2*). Using theoretically bounded windows, accuracy in refined WCET estimate improves by 5%(125%) on an average compared to *Chronos* on *Arch1*(*Arch2*).

Refinement is designed to allow the user to control variance of CPI within a sub-phase, which is useful in programs like *Bubble sort* wherein CPI varies throughout program execution and points of high variation of CPI cannot be isolated based on PC signatures alone. We split a sub-phase into four levels (CPI variance within the sub-phase is limited to 50%, 10%, 5% and 1% of average CPI variance of the sub-phase obtained by refinement based on PC signature). Refining *Bubble sort (Arch1)* at these four levels reduces pessimism by 21%, 31%, 35% and 42% respectively.

Compared to *Chronos*, accuracy of WCET continues to improve following refinement at each of these four levels of CPI variance on *Arch1*(by {20%, 21%, 23% and 24%} using maximum observed windows and by {9%, 11%, 12% and 13%} using theoretical maximum windows). On *Arch2*, compared to *Chronos*, accuracy improves by refinement at the first level(50%) by 192% (using maximum observed windows) and 159% using theoretical maximum windows. The improvement is marginal beyond the first level.

The process of collecting PC signatures through profiling is completely independent of program phase detection and classification. It would be interesting to see if PC signatures can subsume program phase detection and classification. The amount of instrumentation in the proposed method can be varied by modifying the window size depending on the accuracy requirement and availability of resources. Future work will analyze the impact of window size on accuracy of WCET and evaluate the proposed method on larger programs and more complex architectures.

Acknowledgements: We thank Indrajit Bhattacharya, T. Matthew Jacob and Rupesh Nasre for their valuable suggestions on several aspects of this work. We are grateful for the financial support provided by the IMPECS project during the course of this research. We would also like to thank the anonymous reviewers for their suggestions to improve this paper.

8. REFERENCES

[1] http://euler.slu.edu/∼fritts/mediabench.

[2] http://www.comp.nus.edu.sg/∼rpembed/chronos/download.html.

[3] http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

[4] http://www.simplescalar.com.

[5] A. Betts et al. Hybrid measurement-based WCET analysis at the source level using object-level traces. In *Proceedings of WCET 2010*, pages 54–63.

[6] A. Betts et al. WCET Analysis of Component-Based Systems using Timing Traces. In *Proceedings of ICECCS 2011*, pages 13–22.

[7] A. Colin et al. Experimental Evaluation of Code Properties for WCET Analysis. In *RTSS 2008*, pages 190–199.

[8] A. Ravindar et al. Implications of Program Phase Behavior on Timing Analysis. In *Proceedings of INTERACT 2011*, pages 71–79.

[9] B. Davies et al. iPART: An Automated Phase Detection and Recognition Tool. *Technical Report.*, IR-TR-2004-1.

[10] D. Griffin et al. Realism in Statistical Analysis of Worst Case Execution Times. In *Proceedings of WCET 2010*, pages 44–53.

[11] G. Bernat et al. WCET Analysis of Probabilistic Hard Real-Time Systems. In *Proceedings of RTSS 2002*, pages 279–288.

[12] J. Hansen et al. Statistical Based WCET Estimation and Validation. In *Proceedings of ECRTS 2009*, pages 123–133.

[13] J. Lau et al. Motivation for Variable Length Intervals and Hierarchical Phase Behavior. In *ISPASS 2005*, pages 135–146.

[14] J. Lau et al. Selecting Software Phase Markers with Code Structural Analysis. In *Proceedings of CGO 2006*, pages 135–146.

[15] K. Ghani et al. Automatic Test Data Generation for Multiple Condition and MC/DC Coverage. In *Proceedings of ICSEA 2009*, pages 152–157.

[16] M. Corti et al. Approximation of Worst-Case Execution Time for Preemptive Multitasking Systems. In *Proceedings of LCTES 2000*, pages 178–198.

[17] M. Zolda et al. Context-Sensitive Measurement-Based Worst-Case Execution Time Estimation. In *RTCSA 2011*, pages 243–250.

[18] M. Zolda et al. Towards Adaptable Control Flow Segmentation for Measurement-Based Execution Time Analysis. In *Proceedings of RTNS 2009.*

[19] R. Wilhelm et al. The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools. *ACM Trans. Embed. Syst.*, 7(3), April 2008.

[20] S. A. Seshia et al. Game-Theoretic Timing Analysis. In *Proceedings of ICCAD 2008*, pages 575–582.

[21] S. Edgar et al. Statistical Analysis of WCET for Scheduling. In *Proceedings of RTSS 2001*, pages 215–224.

[22] S. Stattelmann et al. On the Use of Context Information for Precise Measurement-Based Execution Time Estimation. In *Proceedings of WCET 2010*, pages 64–76.

[23] Y. Lu et al. A Trace-Based Statistical Worst-Case Execution Time Analysis of Component-Based Real-Time Embedded Systems. In *Proceedings of ETFA 2011.*

[24] Y. Lu et al. A New Way about using Statistical Analysis of Worst-Case Execution Times. *ACM SIGBED Review*, 8(2), September 2011.

[25] S. M. Ross. *Introduction to Probability and Statistics for Engineers and Scientists.* Wiley, 2009.