Uma Ferramenta Cinco em Um para Cálculo de Worst-Case Execution Time (WCET) no Setor Aeronáutico

Daniel Ianni, Henrique Lopes, Lucas Menezes, Lucas Parreira e Manuela Matos

[1]Centro de Informática - Universidade Federal de Pernambuco (UFPE)
Av. Jorn. Aníbal Fernandes, s/n - Cidade Universitária, Recife - PE, 50740-560

dsi@cin.ufpe.br, hlc3@cin.ufpe.br, lkmgm@cin.ufpe.br,

lpap@cin.ufpe.br, mmcs@cin.ufpe.br

Resumo. *Em um cenário em que sistemas embarcados desempenham funções críticas na indústria aeronáutica, a determinação precisa do WCET é necessária para garantir a confiabilidade e a previsibilidade desses sistemas. Abordagens tradicionais de análise estática e dinâmica enfrentam desafios significativos, como a necessidade de modelos detalhados do hardware-alvo e a falta de garantias de segurança nas estimativas. Este trabalho apresenta a Ferramenta de Análise de WCET Cinco em Um (FiOWAT), uma nova ferramenta para estimar o WCET de programas em linguagem ANSI C. O FiOWAT supera essas limitações implementando cinco metodologias distintas que dispensam a necessidade de modelos teóricos de hardwares-alvo complexos e proporcionam estimativas seguras pela Teoria de Valores Extremos. Experimentos com os algoritmos Bubble Sort e Insertion Sort validaram as metodologias no microcontrolador ATmega328, revelando trade-offs entre precisão e tempo de execução. O algoritmo de inversão de matriz foi investigado em uma análise probabilística nos microcontroladores ESP32-S3, ESP32-C3 e STM32F103C8T6 obtendo resultados otimistas para arquiteturas complexas e com hardware que introduz variabilidade no tempo de execução. Finalmente, o FiOWAT destaca-se como uma ferramenta automática e adaptável com alto potencial para aplicações industriais de cálculo de WCET.*

Keywords— Algoritmo Genético, Teoria dos Valores Extremos, Técnica da Enumeração de Caminho Implícito, Solucionador de restrições

Abstract. *In a context where embedded systems play critical roles in the aviation industry, accurate determination of WCET is necessary to guarantee the reliability and predictability of these systems. Traditional static and dynamic analysis approaches face significant challenges, such as the need for detailed models of the target hardware and the lack of assurance in the estimates. This paper presents the Five-in-One WCET Analysis Tool (FiOWAT), a new tool for estimating the WCET of ANSI C language programs. FiOWAT overcomes these limitations by implementing five distinct methodologies that dispense the need for theoretical models of complex target hardwares and provide safe estimates by Extreme Value Theory. Experiments with the Bubble Sort and Insertion Sort algorithms validated the methodologies on the ATmega328 microcontroller, revealing trade-offs between accuracy and execution time. The Matrix Inversion algorithm was investigated in a probabilistic analysis on microcontrollers ESP32-S3, ESP32-C3 and STM32F103C8T6, achieving optimistics results in challenging settings*

characterized by complex architectures and hardware that introduces variability in runtime. Finally, FiOWAT stands out as an automatic and adaptable tool with high potential for industrial WCET calculation applications.

Keywords— Genetic Algorithm, Extreme Value Theory, Implicit Path Enumeration Technique, Constraint Solver

1. Introdução

O uso de sistemas embarcados em uma variedade cada vez maior de dispositivos eletrônicos é uma marca da sociedade contemporânea e da chamada 4ª Revolução Industrial. Esses sistemas são caracterizados por desempenhar tarefas específicas e dedicadas, geralmente como parte de um sistema maior que requer algum grau de automação. Com frequência, precisam atender a exigências particulares do contexto em que são empregados, como tamanho reduzido, baixo consumo de energia, alta velocidade de processamento, entre outros.

Na indústria aeronáutica, os sistemas embarcados precisam ser confiáveis e previsíveis porque frequentemente desempenham tarefas críticas. Para isso, é fundamental garantir que a execução de uma tarefa ocorra dentro de um prazo determinado [Andersson et al. 2023, Wartel et al. 2013]. No entanto, essa garantia requer conhecer o maior tempo de execução possível de um programa (WCET, do inglês *Worst-Case Execution Time*), o que é um problema incomputável, uma vez que implicaria afirmar que o programa sempre encerra sua execução, o que, por sua vez, resolveria o Problema da Parada [Griffin and Burns 2010].

Assim, na impossibilidade de estabelecer um WCET exato, utilizam-se métodos para estimar esse valor. Não obstante, calcular uma boa estimativa de WCET é um desafio considerável, porque o tempo de execução de um programa é determinado por fatores complexos de analisar, como o tempo tomado por tarefas externas, os possíveis valores de entrada, mapeamentos, inicializações de memória e outras configurações iniciais de hardware. Além disso, uma boa estimativa deve ser segura e precisa, o que são características difíceis de equilibrar dada suas naturezas antagônicas [Wilhelm 2020].

Tradicionalmente, os métodos para obtenção de estimativas são organizados em duas grandes classes: análise estática e análise dinâmica [Silva et al. 2015]. Na primeira, determina-se o pior caminho de execução do programa a partir do código-fonte e calcula-se a velocidade de execução com modelos teóricos do hardware-alvo. Essa técnica gera estimativas bastante seguras, mas depende de modelos fiéis do hardware-alvo do programa, o que é cada vez mais difícil para processadores modernos [Puaut]. Na falta de modelos fiéis, as estimativas acabam sendo pouco precisas, *i.e.* superestimadas.

Por outro lado, na análise dinâmica, a estimativa é obtida medindo o tempo de execução do programa no hardware-alvo, sem considerações analíticas sobre caminhos ou componentes do processador como *cache* e *pipeline*. Essa abordagem é menos custosa do ponto de vista teórico e fornece valores mais precisos do que a análise estática. Entretanto, sem considerações analíticas, não se pode garantir que o tempo medido corresponda ao pior caso, o que torna essa estimativa não-segura [Silva et al. 2015]. Assim, somando-se a dificuldade de criar modelos analíticos que gerem estimativas precisas à necessidade de garantir a segurança do valor estimado, é necessário desenvolver novas metodologias para cálculo de WCET que tratem essas limitações.

Este trabalho propõe uma nova ferramenta para estimar o WCET de um programa em linguagem ANSI C. A Ferramenta de Análise de WCET Cinco em Um (FiOWAT, do inglês *Five in One WCET Analysis Tool*) visa enfrentar as limitações das abordagens tradicionais apresentadas no parágrafo anterior. Para isso, implementa cinco metodologias diferentes que mesclam

técnicas de análise estática e dinâmica, dispensando a necessidade de um modelo teórico do hardware-alvo, mas garantindo a segurança das medições com modelos estatísticos. O FiOWAT é uma ferramenta automática que pode ser integrada a outras ferramentas e adaptada a diferentes arquiteturas-alvo sem grandes modificações, o que a torna uma ferramenta de alto valor industrial [Andersson et al. 2023].

Finalmente, este artigo está organizado da seguinte forma: a seção 2 traz uma revisão da literatura e das técnicas relacionadas às metodologias propostas; a seção 3 apresenta em detalhes cada uma das cinco metodologias implementadas na ferramenta FiOWAT; a seção 4 descreve os experimentos realizados para avaliar cada metodologia e seus resultados; a seção 6 reúne as conclusões finais e as sugestões para trabalhos futuros.

2. Revisão bibliográfica

Esta seção fornece uma visão abrangente da literatura relacionada ao cálculo de WCET, explorando tanto os métodos tradicionais quanto inovações mais recentes. Pretende-se identificar lacunas de pesquisa e tendências atuais que fundamentam as metodologias desenvolvidas nesse trabalho.

2.1. Análise de tempo baseada em medição

A Análise de Tempo Baseada em Medição (MBTA, do inglês *Measurement-based Timing Analysis*) é uma abordagem híbrida de análise de WCET, ou seja, combina análise estática com uma parte dinâmica [Wenzel et al. 2008]. A parte dinâmica consiste em medir o tempo de execução do programa ou trechos dele no hardware-alvo. Dessa forma, elimina-se a necessidade de um modelo teórico do processador, uma vez que os efeitos de componentes como cache e pipeline são capturados pela medição.

Neste trabalho, as metodologias *Hybrid IPET* e *WPEVT* são MBTAs. Ambas utilizam a Técnica de Enumeração de Caminho Implícito (IPET, do inglês *Implicit Path Enumeration Technique*) como a parte estática da metodologia, assim como Zola *et al.* [Zolda and Kirner 2016] e Wenzel *et al.* [Wenzel et al. 2008]. Para a parte dinâmica, mede-se o tempo de trechos do programa original, como em [Bernat et al. 2002] e em [Wenzel et al. 2008]. No entanto, difere-se pelo método de segmentação do programa: enquanto nos dois trabalhos citados os segmentos são definidos por aspectos funcionais (*e.g.* o maior trecho que gera uma quantidade tratável de caminhos), neste são blocos básicos, *i.e.* uma seção de código executada de forma contígua e sequencial, sem desvios condicionais.

Por fim, este último trabalho levanta uma problemática dos métodos MBTA que segmentam o código original, que também é descrita em [Bünte et al. 2011]. Ambos ressaltam a necessidade e o desafio de gerar entradas para o programa que permitam alcançar todos os trechos do código alvo. Neste trabalho, as metodologias MBTAs utilizam dois geradores de entrada para alcançar a maior cobertura possível, um de abordagem aleatória e outro usando a ferramenta CBMC [Kroening et al. 2023] para cobertura de linhas.

2.2. IPET

O IPET é uma técnica que permite modelar o fluxo de execução de um programa como um conjunto de restrições que podem ser solucionadas como um sistema linear inteiro [Theiling 2002, Wilhelm et al. 2008].

De forma breve, considera-se que cada função ou programa tem um fluxo de execução que percorre somente uma vez seu ponto de entrada e seu ponto de saída. Além disso, a quantidade de vezes que o fluxo entra e sai de cada bloco básico do programa deve ser a mesma. Por fim, laços

de repetição devem possuir limites de execuções previamente definidos. Com essas considerações, monta-se um sistema linear inteiro que representa o fluxo de execução do programa, com variáveis livres representando quantas vezes cada bloco é executado e ponderadas pelo custo associado de cada bloco.

A partir daí, o objetivo do IPET é achar a solução que maximiza esse sistema, *i.e.* o número de execuções de cada bloco básico (que determina um caminho no programa) que gera o maior valor total do fluxo (WCET). Para solucionar esse sistema linear inteiro, algoritmos como *branch and bound* podem ser utilizados [Toledo and Costa 2019].

Essa técnica é muito utilizada atualmente porque permite modelar vários tipos de programas independente do seu fluxo de execução ou otimizações do compilador. Além disso, tem um custo computacional baixo o suficiente para retornar um WCET de programas de escala industrial em um tempo razoável de análise [Carminati et al. 2017, Puaut].

2.3. Algoritmo Genético

Algoritmos Genéticos (GAs, do inglês *Genetic Algorithms*) são algoritmos matemáticos inspirados nos mecanismos de evolução natural e recombinação genética [Soares 1997]. Portanto, são uma técnica de busca e otimização pelo cromossoma artificial melhor adaptado ao seu meio. Ou seja, dado um determinado valor inicial de entrada e um critério de avaliação (*fitness function* ou função objetivo), o GA irá construir uma população e avaliá-la. Depois realizará o cruzamento entre os cromossomas mais bem avaliados e desenvolverá uma nova população, utilizando métodos de cruzamento (*crossover*) ou de mutação (*mutation*), que será novamente testada, cada etapa concluída desse ciclo é chamada de geração. Esse processo evoluirá até um critério de parada ser atingido, podendo esse ser um valor de avaliação alvo ou uma pré-determinada quantidade de gerações que resultaram na mesma pontuação para o indivíduo mais bem avaliado de cada uma.

Pensando no contexto do pior caso de execução, algoritmos evolutivos podem ser úteis devido a sua mecânica de dado um critério, por exemplo o tempo de execução, buscarem o pior vetor de entrada possível para maximizar o objetivo. Devido ao critério de avaliação ser o tempo de execução, essa metodologia é nomeada de teste temporal (*temporal testing*), como definido por [Wegener 1996].

Atualmente, técnicas evolutivas (como o GA) são frequentemente utilizadas para explorar exaustivamente todas as possíveis execuções de um teste temporal de sistemas de tempo real [Rashida et al. 2020]. Entretanto, como proposto por [Bate and Khan 2011], é possível utilizar outros critérios de otimização além do tempo de execução, alguns desses critérios são a quantidade de laços, de ramificações ou até de execuções de cada instrução.

2.4. Constraint Solver

Ao tentar limitar a variabilidade causada pelo software determinando qual é o pior caminho que levaria ao maior tempo de execução, gera-se um novo problema que é determinar quais são os valores de entrada que levam o fluxo de execução a percorrer esse pior caminho [Oliveira 2020].

Para solucionar esse problema, existem ferramentas como os *constraints solvers*, que consistem em programas que realizam a execução simbólica ou concólica, de modo a verificar quais são os valores válidos que satisfazem todas as restrições encontradas naqueles caminhos (condicionais, laços de repetição, *def-use*) [Baldoni et al. 2018].

A grande vantagem da execução simbólica é determinar a satisfabilidade de uma grande faixa de valores, porém esse tipo de execução é restringida pela capacidade da ferramenta, a explosão de caminhos possíveis, memória para armazenar todos os estados. Esses fatores

acabam dificultando de usá-los para programas reais em escalas maiores [Baldoni et al. 2018, Bjørner et al. 2019, Cadar et al. 2008].

O FiOWAT possui um *constraints solver* que segue por uma nova abordagem, que consiste em utilizar um GA para buscar valores de entrada que mais se aproximem do pior caminho encontrado pelo IPET.

2.5. EVT

No domínio da MBTA, ao utilizar-se de análise probabilística dos tempos de medição é conferida a esta a denominação de Análise de Tempo Probabilística Baseada em Medição (MBPTA, do inglês *Measurement-Based Probabilistic Timing Analysis*), que consiste na utilização da teoria conhecida como Teoria dos Valores Extremos (EVT, do inglês *Extreme Value Theory*) para análise temporal.

Essa teoria propõe uma abordagem para determinar a probabilidade de ocorrência de valores extremos através de uma série histórica de eventos medidos [Coles 2001]. O uso de EVT já foi demonstrado como forma eficiente na redução de eventos necessários para obtenção do pWCET (*probabilistic WCET*), que é derivado através da estimação de uma distribuição de probabilidade de tempos de execução de uma certa tarefa [Cucu-Grosjean et al. 2012, Davis et al. 2014, Acaro 2019].

A metodologia provê o valor de pWCET que pode ser excedido com uma dada probabilidade de ocorrência (e.g., 10^{-12} de ocorrer uma vez). A utilização do EVT permite predizer a probabilidade de exceder-se certo valor de tempo de execução, mesmo que ele não tenha ocorrido durante as execuções anteriores.

O uso adequado dessa metodologia é atrelado a diversos requisitos, um deles é que a plataforma ou *target* em questão apresente, via hardware ou software, variabilidade ou respostas de eventos randomizados durante a execução [Abella et al. 2014, Kosmidis et al. 2016]. O uso de plataformas com variabilidade induzida de hardware e/ou software já conta com diversas ocorrências na literatura, em que o uso do EVT se mostrou bem sucedido, respeitando-se as suas premissas de aplicabilidade[Berezovskyi et al. 2014]. Entretanto, algumas patologias de comportamento temporal de certos hardwares tornam o uso de EVT não-adequado à confiabilidade desejada, por exemplo em sistemas críticos de tempo real [Kosmidis et al. 2016, Cazorla et al. 2016].

Outra aplicação de EVT que tem ganhado atenção é para obtenção de pWCET em arquiteturas mistas ou *multi-cores*, que compartilham recursos e temporalidade, e que por conta da complexidade e não-determinismo representam uma barreira ao uso de métodos estáticos tradicionais de obtenção do WCET. Neste contexto, resultados otimistas foram obtidos com MBTPA em certas arquiteturas não-determinísticas, por exemplo em [Abella et al. 2014], em que o pWCET obtido em arquiteturas mistas superou o pWCET obtido na arquitetura determinística em apenas 12% do tempo de execução.

Neste artigo, o uso de MBPTA concentrou-se em mostrar uma comparação entre os resultados obtidos em diferentes arquiteturas ou seja, com diferentes complexidades, para testar a validade e aplicabilidade desse método, seja sozinho ou combinado com outra metodologia da ferramenta FiOWAT.

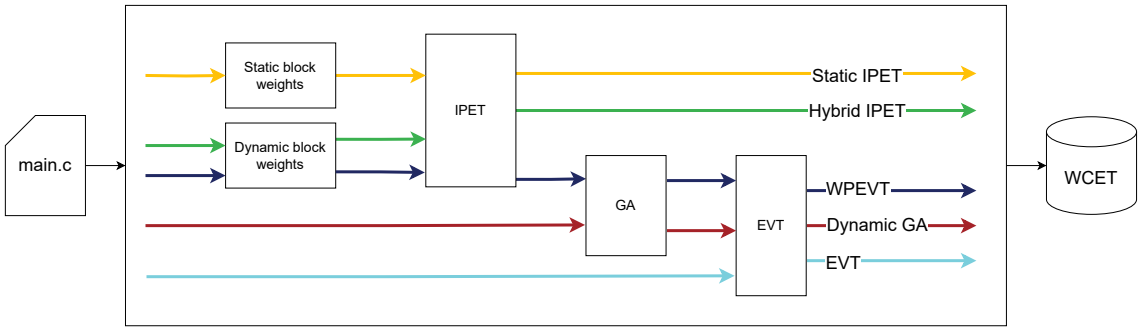
3. Metodologias

O FiOWAT foi desenvolvido para ser o mais automático possível em suas várias etapas de análise, evitando modificar o fluxo de trabalho do usuário, entregando resultados de fácil

interpretação e tratando possíveis erros de uso. Sua implementação teve como base a norma DO-330 e todas as ferramentas externas que foram agregadas ao projeto tiveram suas licenças previamente analisadas [Rierson 2013]. Sua arquitetura é modularizada e facilmente extensível. Ao longo do desenvolvimento, vários artefatos foram gerados, por exemplo, o Manual do Usuário demonstrando a instalação e utilização da ferramenta ou o Relatório de Falhas, apresentando quais outras ferramentas e ideias foram testadas, mas não foram levadas a diante e suas respectivas justificativas. Todos esses documentos estão presentes na pasta compactada "artefatos.zip"dentro do repositório privado do projeto no Github [1].

Nesta seção, serão apresentadas em detalhes as cinco metodologias implementadas na ferramenta. A Figura 1 é uma representação em alto nível do percurso que cada metodologia segue pelos principais módulos da ferramenta. As metodologias *Static IPET* e *Hybrid IPET* utilizam IPET para obter uma estimativa de WCET e se diferenciam entre si pelo método de obtenção dos pesos dos blocos básicos: a primeira usa análise estática e a segunda medição no hardware-alvo. As metodologias *EVT* e *Dynamic GA* fornecem estimativas probabilísticas do WCET, aplicando EVT a uma série de medições do tempo de execução do programa. Por fim, a metodologia *WPEVT* é a central e mais complexa do trabalho, e utiliza todos os módulos da ferramenta para calcular também um WCET probabilístico. Nas seções subsequentes, cada uma delas será descrita em detalhes.

Figura 1. Diagrama com as cinco metodologias implementadas na ferramenta FiOWAT



3.1. Metodologia *Static IPET*

A metodologia do IPET estático consiste em gerar o código em Representação Intermediária (IR, do inglês *Intermediate Representation*) com a ferramenta LLVM [LLVM 2023], e a partir dele obter as informações necessárias para montar um sistema linear inteiro equivalente ao fluxo de execução do programa, por exemplo as fronteiras dos blocos básicos, limites dos laços de repetição, ordem das chamadas de funções, entre outros [Rufino 2009].

Para determinar os limites dos laços de repetição, o código-fonte deve estar anotado com o símbolo //@ seguido do número máximo de vezes que aquele laço de repetição executa. Caso o usuário esqueça de anotar todos os laços, o FiOWAT possui um analisador de laços simples que tenta deduzir o limite do laço automaticamente.

Em seguida, determina-se a quantidade de ciclos utilizada por cada bloco básico relacionando as instruções da IR com o seu respectivo CPI (Ciclo por instrução) para a arquitetura-alvo, obtido do trabalho de [Fachini 2011].

[1] https://github.com/daniel-cin/TCC-PES

A próxima etapa é determinar a quantidade necessária de ciclos para executar o bloco básico todo e para isso considera-se que suas instruções são executadas sequencialmente, sem *pipeline*, de modo a tornar a análise mais segura [Fachini 2011]. Por fim, o FiOWAT calcula o WCET encontrando a solução que maximiza o sistema linear inteiro montado pelo IPET para cada função, percorrendo o grafo de chamada do programa em pós-ordem.

Dessa forma, solucionando o sistema por função, a complexidade computacional do algoritmo diminui, já que os métodos de solução para sistemas lineares inteiros possuem uma ordem de complexidade não-linear, ou seja, executar esse algoritmo por partes tem custo menor que executar uma única vez o algoritmo no programa inteiro [Queiroz 2014, Silva et al. 2015, Toledo and Costa 2019].

Outra vantagem de aplicar essa técnica é que funções que são chamadas em vários pontos do programa são calculadas uma única vez, ou seja, mesmo que elas possam seguir por fluxos menores em trechos diferentes, a análise do FiOWAT considera que a função sempre vai percorrer o seu pior caminho independentemente de qual trecho de código a chame.

Assim, pode-se escolher estimar o sWCET (WCET estático) de uma função determinada pelo usuário, ou se estimar o sWCET da função ”main”, é o mesmo que estimar o WCET do programa inteiro.

3.2. Metodologia *Hybrid IPET*

A metodologia *Hybrid IPET* difere da anterior apenas na geração dos pesos dos blocos básicos do programa, que compõem o sistema linear inteiro solucionado pelo IPET. Na abordagem híbrida, esses pesos são determinados por medição, ao invés da soma dos CPIs das instruções do bloco. Assim, obtém-se valores mais precisos, *i.e.* menos superestimados, pois os efeitos de *cache e pipeline* são diretamente considerados nos tempos medidos.

O processo de medição começa segmentando o arquivo-fonte em blocos básicos, para posteriormente inserir instrumentação nas suas fronteiras. Essa segmentação é realizada pelo *front-end* do compilador clang do LLVM, que gera uma representação intermediária (IR) do código organizada como um grafo de fluxo de controle. Em seguida, para introduzir a instrumentação na IR, a ferramenta opt do LLVM carrega e executa um ”passe”sobre ela.

O opt pode carregar passes nativos do LLVM (otimizações e transformações já implementadas) ou passes personalizados, criados com a API LLVM. Para as medições do FiOWAT, implementou-se um passe personalizado que insere instruções de medição no início de cada bloco básico.

Essas instruções de medição consistem em manipular o contador de ciclos de *clock* da arquitetura-alvo, como, por exemplo, imprimir seu valor na tela ou zerá-lo após a impressão. Para garantir a compatibilidade com várias arquiteturas, essas instruções foram encapsuladas em funções definidas em arquivos externos, específicos de arquitetura, de forma que o passe insira chamadas às funções com nomes padronizados, mas cuja implementação é decidida em tempo de ligação.

Após a geração da IR e a instrumentação com o passe personalizado, o processo de compilação segue as etapas convencionais: os arquivos externos são ligados ao programa, e a IR é compilada para um executável específico da arquitetura. Então, executa-se o código instrumentado no hardware-alvo, que para cada bloco básico no caminho de execução imprime o nome do bloco e valor corrente do contador. Finalmente, com os valores impressos durante a execução, calcula-se o tempo dos blocos.

Vale observar que, nessa abordagem, o fluxo de execução do programa precisa exe-

cutar o trecho de código para que seja medido, como apontado por [Wenzel et al. 2008] e [Bünte et al. 2011]. Por esse motivo, o *Hybrid IPET* determina o peso dos blocos integrando os valores obtidos em execuções de diferentes entradas geradas pela ferramenta, de modo que cubra o máximo de instruções possíveis.

Finalmente, com os pesos atribuídos a cada bloco do programa, o IPET determina o WCET da mesma maneira que na metodologia do *Static IPET* (subseção 3.1), ou seja, resolvendo o sistema linear inteiro correspondente ao grafo de fluxo de controle anotado com esses pesos.

3.3. Metodologia *EVT*

A metodologia *EVT* pode ser segmentada em etapas, nesse artigo dividido em seis etapas. A primeira etapa consiste na obtenção de uma amostra representativa de tempos de execução, que seja suficientemente grande e que atenda aos requisitos impostos pelo próprio método e descritos na etapa seguinte de verificação de adequação.

Para obtenção do tempo de execução, faz se desejável que o par *benchmark* e dados de entrada sejam favoráveis à obtenção de caminhos extensos de execução do programa, por exemplo para o caso de testes com o *benchmark bubblesort*, um vetor inversamente ordenado seria o pior dado de entrada escolhido para obtenção das amostras.

Outro fator que faz-se importante notar é a configuração do hardware que deve ser conhecida e comum a toda amostra, de forma a se estudar o mesmo cenário dentro da amostra. Conhecendo-se a quantidade de valores da amostra como uma questão ainda aberta na literatura, e dado a margens de confiabilidade estreitas que sistemas críticos necessitam ($\sim 10^{-9}$), definiu-se uma quantidade mínima de 5.000 amostras na ferramenta, porém para os testes presentes aqui o mínimo foram 10.000 amostras para qualquer metodologia que use o EVT [Jiménez Gil et al. 2017].

Em seguida, a adequação da amostra é avaliada em termos de aderência ao critério Independente e Identicamente Distribuída (i.i.d). Esse critério é atendido mediante a um conjunto de testes estatísticos de hipótese que precisam atender ao valor crítico, $p \geq 95\%$ nesse trabalho.

Os testes de independência de amostragem utilizados foram os testes *Wald-Wolfowitz* (WW) e o *Ljung-Box*(LB), enquanto para os testes de similaridade, foram os testes Kolmogorov-Smirnov(KS) e k-sample Anderson-Darling(AD). Para esses testes utilizou-se sub amostras randômicas contendo 1.000 elementos cada e calculadas 100 vezes de forma a obter-se uma distribuição dos resultados, exibidos como um *boxplot* na seção de resultados[Coles 2001, Acaro 2019].

A terceira etapa consiste em selecionar os extremos da amostra, de forma a reduzir o espaço amostral para concentrar-se na distribuição próxima da região assintótica, ou também conhecida como cauda da distribuição. Entre os métodos empregados estão, o Máximo por Blocos (*Block Maxima*, BM) e o Picos Acima do Limiar (*Peaks Over Threshold*, POT).

Os dois métodos são executados nesta etapa, onde somente um é escolhido na etapa seguinte para seguir na execução. A escolha é feita utilizando-se o método que propiciar o melhor ajuste entre a distribuição empírica e a distribuição estimada pelo processo de ajuste de curva, *fitting*.

Cada um dos métodos, utiliza um parâmetro chave para sua aplicação que pode ou não estar otimizado. Para o POT esse valor é o limiar, ou *threshold*, e no caso do BM, é o tamanho do bloco. A definição desses parâmetros pode ser complexa para estimar-se de forma adequada assim como estudos demonstraram [Abella et al. 2017, Silva et al. 2015, Acaro 2019]. Com o objetivo de reduzir variáveis da utilização do método, utilizou-se para o limiar do POT o valor

correspondente a 0.98 do quantil das amostras, e para o tamanho de bloco do BM, blocos com 1% do tamanho da amostra original.

A etapa de ajuste da curva de distribuição, a quarta etapa, é executada uma vez para cada método, POT e BM, em que cada método baseia-se em uma família de distribuição diferente. Para o POT a teoria aplicada do EVT baseia-se em uma Distribuição Generalizada de Pareto (GPD), enquanto para BM, em uma família de distribuições conhecida como Distribuição Generalizada de Valores Extremos (GEV). Para obtenção dos parâmetros das distribuições foi utilizada a ferramenta, FIT, da biblioteca Scipy-Stats que realiza o melhor ajuste desses parâmetros, através do método de Máxima Semelhança de Estimação Generalizada (GMLE).

Após os dois métodos aplicados obterem os parâmetros das distribuições estimadas, a próxima etapa realiza a avaliação numérica da etapa anterior, através do valor de Kolmogorov-Smirnov (KS) da distribuição estimada comparada a distribuição empírica das amostras, em que o método com o melhor ajuste segue para as última etapa do procedimento.

Outra forma de fazer a avaliação do ajuste é após a execução da ferramenta utilizando a comparação dos quantis e percentis da amostra com os da distribuição pelos gráficos QQ e PP plot disponíveis no relatório gerado pela ferramenta. Os dois gráficos exibem uma reta de 45 graus entre os eixos das abcissas e ordenadas que representa o melhor ajuste entre o valor empírico da amostra e o valor obtido pela distribuição de probabilidade ajustada para um quantil ou percentil desejado. O quantil é a contagem de valores que há até aquele valor da amostra e o percentil similar mas em valores percentuais. A última etapa consiste na obtenção do pWCET, através da projeção da cauda da distribuição empírica [Kosmidis et al. 2016]. Esse valor é obtido calculando-se o inverso da função acumulada de probabilidade da distribuição obtida, 1- CDF, para as probabilidades de excedência desejados de se obter.

3.4. Metodologia *Dynamic GA*

Baseada na combinação do Algoritmo Genético e da Teoria do Valor Extremo, essa metodologia consiste em obter a pior entrada para o programa através do algoritmo evolutivo e depois calcular um WCET probabilístico a partir de dos tempos de execução do programa com essa pior entrada usando o EVT explicado na seção 3.3.

Optou-se por utilizar a implementação do Algoritmo Genético da biblioteca PyGaD[PyGaD 2023] por ser uma biblioteca de código-aberto e com amplo respaldo na comunidade científica [2]. Ela permite a personalização de uma função objetivo para cada problema, a definição dos hiperparâmetros utilizados em algoritmos genéticos e a implementação de critérios de parada. Nessa parte serão explicadas a escolha da função objetivo e do critério de parada, pois a determinação dos hiper-parâmetros do GA será tratada no capítulo seguinte.

A cada geração da execução do PyGaD, uma certa quantidade de indivíduos será analisada sobre o seu desempenho, a função que realizará essa avaliação é chamada de função objetivo. No caso tratado, essa possuirá apenas um objetivo, o número de ciclos de execução do programa alvo. Para a determinação desse valor será necessário informar o tamanho de um cromossoma, ou seja, a quantidade de variáveis que um vetor de entrada do programa alvo possui e o intervalo em que eles podem variar. Portanto, três variáveis devem ser preenchidas, a primeira a quantidade de variáveis, sendo essa obrigatoriamente um valor inteiro, a segunda o menor valor possível que o GA poderá testar, chamada de *bound-min* e a terceira o maior valor possível, chamada de *bound-max*.

Dessa forma, 3 valores de entrada serão transmitidos ao algoritmo genético e após o termino de sua execução serão disponibilizados dois resultados, um é a maior quantidade de ciclos ao

[2] https://pygad.readthedocs.io/en/latest/releases.html#research-papers-using-pygad

longo de toda execução e o outro é o vetor que a gerou. Além disso, é possível a seleção de quatro hiperparâmetros pelo usuário: a forma que os pais são selecionados para a próxima geração, como o cruzamento entre eles é realizado, como a nova geração sofrerá mutação e a percentagem dessa geração que sofrerá mutação. Entretanto, sugere-se um valor padrão baseados nas investigações feitas neste trabalho, que serão descritas na seção 4.

Caso nada seja definido, o algoritmo genético encerrará sua execução quando o número de gerações for igual ao hiper-parâmetro relativo a quantidade máxima de gerações. Entretanto, é possível que o código encerre antes disso caso algum critério de parada seja atingido, oficialmente o PyGaD disponibiliza duas opções para a definição do critério de parada. O primeiro acontece quando o valor de um indivíduo for igual ao valor definido, esse é um critério recomendado para quando se conhece o WCET e deseja-se saber qual indivíduo é responsável por essa quantidade de ciclos. Enquanto o segundo é um critério de estabilidade, que consiste em uma análise de por quantas gerações o valor da função objetivo não é alterado, por exemplo, se após mil gerações a quantidade máxima de ciclos executados não foi alterada, o algoritmo encerra sua execução apresentando o resultado atual como final devido ao critério de convergência.

Após a conclusão do algoritmo genético, a metodologia segue com a realização do EVT. Com o pior valor de entrada descoberto pelo GA, o programa é executado múltiplas vezes gerando uma série de medições de tempo de execução. A partir dessa série, o EVT calcula o WCET probabilístico conforme descrito na subseção 3.3.

3.5. Metodologia *Worst Path EVT*

O *Worst Path EVT* (WPEVT) é a metodologia original e central do FiOWAT. Ela é a razão pela qual todas as outras foram desenvolvidas, já que reutilizam componentes criados para ela.

A WPEVT começa com a medição dos tempos de execução de cada bloco básico do programa como explicado na subseção 3.2. Em seguida, executa-se o IPET sob a estrutura de grafo do programa com os pesos obtidos. No entanto, esse IPET diferencia-se das abordagens anteriores por armazenar na memória os valores correspondentes ao pior caminho encontrado com a resolução do sistema linear, ao invés de apenas fornecer o tempo desse caminho.

A partir desses valores, utiliza-se um Algoritmo Genético para determinar o valor de entrada que gera esse pior fluxo de execução. Nesse processo, o GA utiliza uma versão instrumentada do programa para tentar maximizar a correspondência entre o pior caminho gerado pelo IPET na etapa anterior e o caminho percorrido pelo programa com o indivíduo recém gerado como entrada.

Ao atingir o critério de parada de estabilidade explicado em 3.4, o programa é executado uma série de vezes com o possível pior valor de entrada para obter uma sequência de tempos de execução. Por fim, o método aplica o EVT a essa série de medições e determina um WCET probabilístico.

Nessa metodologia, o WCET é estimado considerando o pior caminho dentro do fluxo de execução do programa com o IPET, *i.e.* pior cenário a nível de software, e considerando as possíveis variações de hardware que a arquitetura possa apresentar, ao aplicar o EVT [Oliveira 2020].

3.6. Recomendações de aplicação para cada metodologia

Cada metodologia possui um ponto forte e um ponto fraco, seguindo a regra geral, as metodologias que utilizam de análise estática somente analisam arquiteturas que conseguem dar suporte, enquanto as dinâmicas conseguem dar suporte a qualquer arquitetura que se consegue estimar o tempo de execução, exceto as que tem hardware determinístico, o que impede de aplicar a técnica do EVT para gerar uma estimativa de WCET segura.

O *Static IPET* é a mais rápida e menos precisa de todas, sendo útil para estimar o WCET com o menor custo de tempo. A única dificuldade é preparar as anotações dos limites dos laços.

O *Hybrid IPET* demora um pouco mais, porém é rápida o suficiente para fornecer estimativas bem precisas. Possui o mesmo problema de exigir anotações manuais do código-fonte.

O *WPEVT* é a mais demorada de todas, por exigir o trabalho de preparar alguns códigos-fontes anotados, instrumentar eles, achar o pior caminho e aplicar o EVT. Apesar de todo esse custo, ele gera as estimativas mais seguras e precisas comparada a todas as outras metodologias.

O *Dynamic GA* é utilizada quando não se sabe os piores valores de entrada, e o hardware é difícil de ser modelado por outras técnicas. Ele tem um custo intermediário de tempo, porém com uma medida precisa.

Por fim, o *EVT* é utilizado quando o usuário conhece os piores valores de entrada, portanto ele pode de maneira rápida e segura estimar o WCET. Apesar dela não ter um *trade-off* diretamente, ela demanda que o usuário já saiba de antemão por outras maneiras quais são os piores valores de entrada para aquele programa.

4. Experimentos

Para validar a capacidade das metodologias de estimar WCETs corretamente, *i.e.* dentro de uma margem de precisão aceitável, foram realizados experimentos com programas ANSI C em diferentes arquiteturas-alvo. Além disso, também foi feita uma análise do tempo de execução das metodologias, uma vez que esse é um especto relevante para uma ferramenta de análise de WCET [Andersson et al. 2023].

4.1. Benchmarks

A avaliação experimental da ferramenta FiOWAT teve como base *benchmarks* disponibilizados pelo *Mälardalen WCET Research Group*, que possui uma série de códigos-fonte em C que incluem recursos da linguagem como laços aninhados, recursão, condicionais, entre outros [Gustafsson et al. 2010]. Para os experimentos deste trabalho, foram utilizados os *benchmarks* *Bubble Sort* (bsort), *Insertion Sort* (insort) e *Matrix Inversion* (minvers).

Esses *benchmarks* foram escolhidos por serem programas pequenos e de fácil validação dos resultados obtidos. Por exemplo, a pior entrada possível é conhecida para o *Bubble Sort*, o que permite avaliar os resultados do Algoritmo Genético e do IPET. Além disso, devido ao tamanho reduzido, é possível validar manualmente os resultados gerados para arquiteturas simples.

Todos os *benchmarks* foram adaptados para que se tornassem programas de múltiplos caminhos, uma vez que suas versões no repositório do *Mälardalen WCET Research Group* são de caminho único. Essa adaptação foi feita para tornar o problema do WCET mais parecido com um problema real, em que o tempo de execução do programa é afetado pelas entradas que recebe.

Além da inserção do código para receber essas entradas, foram adicionadas instruções de instrumentação para medir o tempo da função-alvo do programa. É importante ressaltar que essas modificações foram feitas na função *main* do programa e que não fazem parte do código da função-alvo para o qual o WCET é estimado no FiOWAT.

4.2. Arquiteturas-alvo

A arquitetura AVR foi a primeira a ser suportada pela ferramenta. Utilizou-se a implementação do microcontrolador ATmega328 [Atmel Corporation 2016], que não possui *cache, pipeline* ou outros aceleradores de execução. Os experimentos para essa arquitetura foram

executados no simulador SimulAVR[3] e em uma placa de desenvolvimento Arduino UNO. Embora o cálculo de WCET não seja desafiador em AVR, ela é uma arquitetura simples e determinística que permite validar os resultados de WCET em programas pequenos.

Posteriormente, três outras plataformas com mais recursos foram adotadas: ARM Cortex-M3, RISC-V e Xtensa, através dos *chips* STM32F103C8T6 [STMicroelectronics 2021], ESP32-S3 [Espressif Systems 2023b] e ESP32-C3 [Espressif Systems 2023a]. Os experimentos foram executados em placas de desenvolvimento, porém apenas as metodologias Dynamic GA e EVT dão suporte a essas arquiteturas, pelos motivos explicados no documento suplementar de desenvolvimento da ferramenta. Na tabela 1, estão resumidas as principais características de cada microcontrolador e o suporte provido pela FiOWAT.

Tabela 1. Resumo dos processadores utilizados nos experimentos.

	Núcleo	Cache	Pipeline	Serial	Suporte FiOWAT
ATmega328	Single-core 8-bit AVR, até 16 MHz	Não possui	Não possui	UART	[1][2][3][4][5]
STM32F103C8T6	Single-core Arm® 32-bit Cortex-M3, até 72 MHz	Não possui	3 estágios	UART	[4][5]
ESP32-C3	Single-core 32-bit RISC-V, até 160 MHz	16 KB SRAM	4 estágios	USB CDC	[4][5]
ESP32-S3	Dual-core 32-bit Xtensa LX6, até 240 MHz	32 KB SRAM	5 estágios	USB CDC	[4][5]

As metodologias estão representadas na última coluna por: Static IPET [1], Hybrid IPET [2], WPEVT [3], Dynamic GA [4] e EVT [5]

4.3. Hiperparâmetros do GA

As metodologias baseadas em algoritmos genéticos requerem a definição de hiperparâmetros para sua execução. Após a decisão do *benchmark* e da arquitetura alvo, pode-se utilizar os dois para múltiplas execuções, essas geraram dados para a definição dos seguintes hiperpârametros: *Parent Selection Type*, *Mutation Type*, *Crossover Type* e *Random Seed*.

A biblioteca PyGaD ([PyGaD 2023]) utilizada permite que a seleção de pais seja feita através das metodologias: (sss — random — rank — sus — tournament — rws); que o cruzamento dos genes aconteça via: (two_points — scattered — single_point — uniform); e que a mutação de cada cromossomo realize-se utilizando: (adaptive — inversion — random — scramble — swap).

Além disso, a *Random Seed* ou semente possui como limitação apenas o seu tipo inteiro, porém escolheu-se variá-la de 1 a 10. O número total de execuções foi de $6 \times 4 \times 5 \times 10 = 1200$, portanto para a definição desses hiper-parâmetros ser realizada em um tempo hábil no cronograma, escolheu-se o *benchmark BubbleSort* com 10 elementos como *input*, uma população com 50 elementos e 100 gerações de execução, resultando em um tempo total de execução igual 131660 segundos, ou aproximadamente 36 horas.

Com o intuito de analisar o critério de convergência do GA, aproveitou-se que a função *BubbleSort* possui um pior *input* conhecido (Vetor com valores decrescentes) e executou-se o *benchmark* obtendo um pior tempo de execução de 7802 ciclos através do desenvolvimento da biblioteca *tick-counter*, durante a implementação do programa principal, alterou-se o compilador e o WCET desse programa específico foi reduzido para 7560 ciclos.

Após a definição do WCET alvo para esse teste, utilizou-se o critério de parada *"Reach_7082"*, assim quando a função objetivo atingisse esse valor o código encerraria e o tempo de execução, juntamente com outros dados, seriam armazenados automaticamente em um arquivo Excel. A partir desses dados, construiu-se os gráficos 5, 6 e 7 a serem analisados, esses foram construídos através da plataforma Power BI.

[3]https://savannah.nongnu.org/projects/simulavr/

A figura 5 apresenta a quantidade de vezes que uma mesma seleção de pais convergiu para um determinado número de ciclos e o tamanho de cada círculo é a quantidade de tipos de mutação diferentes presentes naquele mesmo ciclo.

Portanto, observa-se que o critério *rank* não está disposto nas opções, pois como mostrado na figura 6, ele não demonstrou desempenho suficiente para continuar no processo de análise e foi removido.

Dessa forma, analisando os critérios restantes é notório que os cinco alcançaram o patamar de pior caso de execução. Entretanto, o critério *sss* foi o que apresentou maior número de repetições, ou seja, foi o critério que mais vezes determinou corretamente o pior *input* do programa.

A comprovação dessa escolha veio através do gráfico 6, nesse apresenta-se colunas com a média de execução de cada tipo diferente de cruzamento de genes em função da seleção de pais e da mutação. Além disso, mostra-se a variação entre o maior e o menor valor encontrado para cada um. A melhor combinação de hiperparâmetros apresentada é *sss* para o método de seleção de pais e *adaptive* para o de mutação, pois independentemente do método de cruzamento alcançou-se o critério de parada.

Portanto, os dois últimos critérios a serem analisados são do *crossover* e da semente geradora, para isso analisa-se a figura 7. Os dados mostrados na imagem foram previamente filtrados para método de seleção de pais igual a *sss* e metodologia de mutação igual a *adaptive*, as quatro formas disponíveis de *crossover* serão analisadas na média de *High Water Mark* obtido, enquanto o hiper-parâmetro da semente geradora será escolhido pelo tempo de convergência do algoritmo genético.

Observa-se que a metodologia *two_points* alcançou o WCET desejado duas vezes e em outras 6 ocasiões ficou a 90% disso, portanto, por ter apresentado a maior média, essa foi a escolhida. Dentro dela, escolheu-se a *Random-Seed* igual a 4, por ter convergido ao valor de 7802 ciclos, mas possuir tempo de execução menor do que a semente 9, que também convergiu.

Após essas determinações, variou-se o tamanho da população entre duas a dez vezes a quantidade de variáveis de entrada, enquanto o número máximo de gerações foi variado entre a primeira e a quarta potência da quantidade de variáveis de entrada. Dado que o *BubbleSort* possui complexidade de ordem n^2, observou-se que havia convergência entre a potência quadrática e a cúbica, portanto optou-se por escolher que o número máximo de gerações seria o cubo do tamanho do vetor de entrada (n_i), mas o critério de parada seria que se n_i^2 gerações obtivessem o mesmo valor na *fitness function*, o algoritmo genético consideraria uma convergência e encerraria a otimização considerando aquele o melhor resultado.

Observando o tamanho da população, não se apresentou ganho significativo quando essa era maior que o triplo de n_i, portanto escolheu-se que o tamanho da população seria o triplo do tamanho do vetor de entrada. Dessa forma, os hiper-parâmetros escolhido para o *benchmark BubbleSort* foram “sss”, “adaptive”, “two_points”, semente igual à 4, tamanho de população igual ao triplo, o número máximo de gerações igual ao cubo e o critério de parada igual ao quadrado da quantidade de variáveis de entrada do programa.

5. Resultados

Nessa seção, serão apresentados os resultados dos experimentos conduzidos com a ferramenta FiOWAT. Em um primeiro momento, foram conduzidos experimentos de validação e comparação de desempenho das metodologias propostas. Em um segundo momento, investigou-se a capacidade das metodologias *Dynamic GA* e EVT lidarem com arquiteturas mais complexas,

i.e. com recursos como *cache*, *pipeline* ou mais de um núcleo.

5.1. Validação e desempenho comparativo das metodologias em AVR (ATmega328)

Neste primeiro experimento, como explicado na seção 4, buscou-se validar a ferramenta a partir de uma arquitetura simples e determinística. Para isso, foram executados os programas *Bubble Sort* e *Insertion Sort* com diferentes tamanhos de vetores de entrada.

As tabelas 2 e 3 mostram os resultados de cada metodologia em termos do WCET estimado e do seu desvio com relação ao WCET real, calculado previamente com a pior entrada de cada algoritmo. O cálculo do WCET real foi possível nesse experimento devido ao uso da arquitetura AVR, em que o pior tempo de execução é determinado apenas pela pior entrada. Por outro lado, por ser determinístico, o hardware-alvo não produz variabilidade suficiente para o uso da metodologia *EVT*. Assim, a comparação foi feita apenas para as demais metodologias e, naquelas munidas de uma etapa final de *EVT*, ele não foi executado.

Tabela 2. WCET estimado e desvio do WCET real - Bubble Sort

	Static IPET		Hybrid IPET		WPEVT		Dynamic GA	
Tamanho vetor	WCET	Dev (%)	WCET	Dev (%)	WCET	Dev (%)	WCET	Dev (%)
10	52607	596.32	9242	22.33	**7555**	**0.0**	**7555**	**0.0**
20	189627	532.93	33457	11.67	**29960**	**0.0**	**29960**	**0.0**
30	411847	511.93	72772	8.13	59460	-11.65	**66403**	**-1.34**

Tabela 3. WCET estimado e desvio do WCET real - Insertion Sort

	Static IPET		Hybrid IPET		WPEVT		Dynamic GA	
Tamanho vetor	WCET	Dev (%)	WCET	Dev (%)	WCET	Dev (%)	WCET	Dev (%)
10	26652	411.95	6372.0	22.4	4951	-4.9	**5155**	**-0.98**
20	98322.0	393.31	23812	19.47	18471	-7.33	**19727**	**-1.02**
30	214792	386.44	52252	18.33	40768	-7.67	**43405**	**-1.7**

A tabela 2 mostra que as metodologias *WPEVT* e *Dynamic GA* estimam o valor do WCET com exatidão para o *Bubble Sort* de tamanho 10 e 20. Já para o tamanho de entrada 30, nenhuma metodologia acerta precisamente o valor. Não obstante, o *Dynamic GA* subestima-o em 1.34%, o que é um resultado não-seguro, mas ainda bastante preciso considerando as margens de erro usuais de WCET.

Para os três tamanhos de entrada, o *Static IPET* superestima o WCET em mais de 500%, mostrando que não é uma técnica precisa para esta tarefa. Isso pode ser explicado pelo valor de CPI usado nessa metodologia, que está associado às instruções da IR do LLVM e não do *assembly*, e que portanto são superestimadas.

Por fim, ainda na tabela 2, o *Hybrid IPET* superestima o valor de WCET em 22.33%, 11.63% e 8.13%, o que significa que estima WCETs seguros, mas não tão precisos quanto das técnicas *WPEVT* e o *Dynamic GA* no geral. A falta de precisão do *Hybrid IPET* pode ser explicada pela simplicidade da análise semântica feita pelo IPET, que, por não eliminar caminhos impossíveis, considera no cálculo da estimativa trechos de código mutuamente excludentes.

A tabela 3 mostra os resultados do mesmo experimento para o algoritmo *Insertion Sort*. Observa-se que a metodologia *Dynamic GA* superou todas as outras em precisão, nos três testes

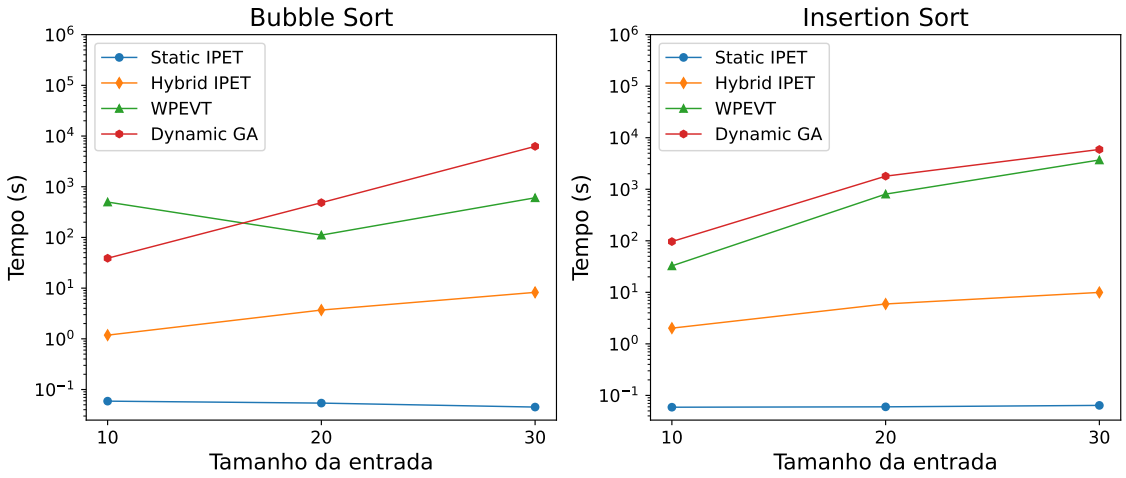
executados. Apesar de não ter obtido resultados perfeitos, a margem de subestimação no pior cenário de teste foi de 1.7%, o que ainda é uma margem pequena para cálculo de WCET. Além disso, a metodologia *WPEVT* também obteve valores com menos de 10% de desvio do valor real.

Como nos resultados anteriores, o *Static IPET* superestimou o WCET para além de uma margem aceitável, o que é explicado pelo mesmo motivo dos resultados anteriores. Por fim, o *Hybrid IPET* atingiu valores de desvio por volta de 20% de superestimação, o que de novo é explicado pela falta de análise semântica do IPET dessa metodologia.

É importante ressaltar, nos experimentos acima, que as metodologias *Dynamic GA* e *WPEVT* não utilizaram a sua parte probabilística, devido ao determinismo do hardware ATmega328. Assim, a estimativa que elas fornecem não são acrescidas de uma margem de erro, o que resulta em um WCET não-seguro. Em um cenário em que o EVT estivesse sendo utilizado, ele inseriria uma margem de erro que tornaria o WCET seguro.

Por fim, uma última análise desses experimentos foi comparar o tempo de execução de cada metodologia. Esses tempos foram medidos em um computador com sistema operacional Ubuntu 22.04 e CPU Intel Core i5-1135G7 11th Gen @ 2.40GHz, usando um único núcleo por vez. Embora os resultados dessas análises sejam pouco determinísticos, pela interferência de tarefas externas ao programa, eles permitem avaliar pelo menos em ordem de grandeza o *trade-off* tempo/acurácia.

Figura 2. Tempo de execução por metodologia



A figura 2 mostra os tempos obtidos para cada metodologia nos algoritmos *Bubble Sort* e no *Insertion Sort*, para diferentes tamanhos de entrada. Em ambos os *benchmarks*, observa-se que o *Static IPET* é a metodologia mais rápida, na ordem de milissegundos, e que seu tempo de execução não depende do tamanho da entrada, o que era esperado para essa metodologia.

A metodologia *Hybrid IPET* estimou o WCET em segundos, em ambos os códigos, crescendo de forma proporcional ao tamanho da entrada no intervalo investigado. Diferente do *Static IPET*, o tempo dessa metodologia é impactado pela complexidade computacional do programa alvo.

As metodologias *WPEVT* e *Dynamic GA* foram as mais demoradas, com tempos indo da ordem de minutos (entradas de tamanho 10 e 20) a horas (entradas de tamanho 30). O tempo de execução de ambas é impactado pela convergência do Algoritmo Genético que executam interna-

mente. Por sua vez, o tempo de convergência é influenciado por hiperparâmetros do GA como o critério de parada, que futuramente serão estudados para melhorar o tempo do caso genérico.

No trabalho [Andersson et al. 2023], consideram que um tempo de análise aceitável para o cálculo de WCET é de até 120 minutos. Pelos resultados obtidos nesses experimentos, apenas as metodologias *Static IPET* e *Hybrid IPET* estariam dentro dessa margem. Além disso, é razoável supor que elas ainda estariam dentro da margem para programas maiores, dado seu comportamento quando aumenta-se o tamanho da entrada.

Já as metodologias *WPEVT* e *Dynamic GA*, mesmo para um hardware simples e programas pequenos, quase cruzam esse limiar (seus maiores tempos registrados foram 61 e 104 minutos, respectivamente), além de aumentarem de forma não-linear com o tamanho da entrada. Portanto, no seu estado atual, essas metodologias não são escaláveis, necessitando de otimizações nos seus algoritmos genéticos.

5.2. Resultados das análises probabilísticas de medição de WCET - (MBPTA)

Nesta seção são apresentados os resultados das metodologias probabilísticas da ferramenta FiOWAT (*Dynamic GA* e *EVT*). Os experimentos foram conduzidos para estudar (1) a viabilidade do uso de análise estatística no cálculo do WCET, (2) o número de amostras suficiente para a análise do EVT e (3) a influência do *benchmark* e da arquitetura na contagem de ciclos.

As metodologias que empregam a análise do *EVT* foram aplicadas em arquiteturas mais complexas, que geram maior variabilidade na contagem de ciclos pelo não-determinismo do hardware. A tabela 4 apresenta os experimentos conduzidos nos chips ESP32-C3 (Riscv), ESP32-S3 (Xtensa) e Cortex-M3 (ARM), utilizando dois *benchmarks* distintos: inversão de matriz (minvers) e o algoritmo de ordenação *Bubble Sort* (bsort). Os resultados nas tabelas 4 e 5 apresentam o valor de pWCET para quatro probabilidades de excedência, 10^{-9},10^{-10}, 10^{-11},10^{-12}.

Para investigar a influência do tamanho das amostras nas inferências estatísticas, foram realizados experimentos com o mesmo *benchmark* variando apenas a quantidade de execuções (10.000, 1.000.000 e 3.000.000), ou seja, a quantidade de marcações na série temporal em que o *EVT* atua. Além disso, foi estudada a influência do tamanho da entrada no algoritmo de inversão de matriz, utilizando para isso uma versão com matriz de ordem 5×5 (minvers5) e outra versão com matriz de ordem 10×10 (minvers10).

Tabela 4. Resultados do pWCET para o *benchmark* minvers em diferentes arquiteturas

		10.000 amostras				1.000.000 amostras				3.000.000 amostras			
		1e-9	1e-10	1e-11	1e-12	1e-9	1e-10	1e-11	1e-12	1e-9	1e-10	1e-11	1e-12
ESP32-C3	minvers5	104376	104377	104377	104377	10438	104388	104388	104389	104384	104384	104385	104385
	minvers10	790761*	793299*	796398*	800184*	x	x	x	x	x	x	x	x
Cortex-M3	minvers5	15372742*	15372742*	15372742*	15372742*	16873562	16873562	16873562	16873562	16873780	16873780	16873780	16873780
	minvers10	17366885	17366885	17366885	17366885	x	x	x	x	x	x	x	x
ESP32-S3	minvers5	80995	80996	80997	80997	81117	81118	81118	81118	81102	81103	81103	81103
	minvers10	80951	80951	80952	80952	x	x	x	x	x	x	x	x

Nota: calculados pWCET para as seguintes probabilidades de excedência: 1e-9, 1e-10, 1e-11 e 1e-12.

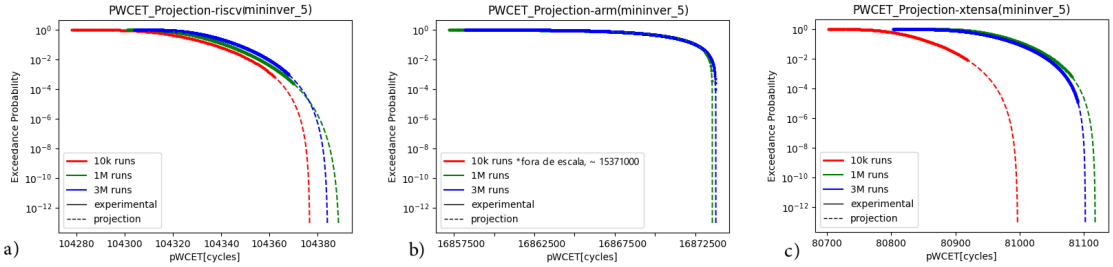
Legenda: (x) não realizado; (*) baixa confiabilidade

Na figura 3 temos o resumo dos resultados da projeção do pWCET para o *benchmark* minvers5, comparando as três arquiteturas utilizadas nessa análise e três quantidades diferentes de execuções. De acordo com os resultados compilados da tabela 4, os valores de pWCET ficam evidentes como assintóticos na escala logarítmica da probabilidade de excedência. Esse resultado assintótico é atribuído a distribuição de probabilidade ajustada dos dados ter o comportamento de cauda leve (*light tailed*) [Abella et al. 2017]. Entre todas as análises de EVT no *benchamrk*

minvers, o método de seleção de máximos melhor adaptado aos dados da amostra foi o BM (*Block Maxima*).

Na figura 3 os resultados com quantidades de execução 10.000 apresentam valores assintóticos de pWCET bem diferentes dos resultados obtidos para números de execução mais expressivos, da ordem de 10^6. Essa diferença do resultado de 10.000 execuções e a média entre os resultados de 1.000.000 e 3.000.000 chegam a 8% para a arquitetura ARM por exemplo, que por questões de visualização não se encontra na figura dentro da escala dos outros dois resultados de 1.000.000 e 3.000.000. A variabilidade desses resultados assintóticos em termos de ciclos de execução não são da mesma ordem de grandeza entre as arquiteturas, por exemplo para o RISC-V esse valor é de 0.01%.

Figura 3. Projeção do PWCET para minvers5



a) RISC-V, b) ARM, c)Xtensa

No contexto estatístico da aplicabilidade do método do *EVT*, todos os resultados da ordem de 10^6 obtiveram êxito no ajuste das curvas em praticamente todas as regiões das distribuições, principalmente no centro e na cauda. Isso fica evidente na figura 4, que traz os gráficos de quantis por quantis e percentis por percentis dos ajustes, e ao lado direito o histograma dos valores máximos extraídos sobreposto da distribuição ajustada. Para além da análise visual da qualidade de ajuste desses casos de testes, a ferramenta não levantou nenhuma exceção de uso e conferiu validade de uso do método à estes casos, dentro da margem de confiabilidade padrão de 0.95%. No caso c), que exibe os resultados de 10.000 execuções, fica evidente a ausência de um bom ajuste entre curva e amostra de máximos, em que dada a baixa quantidade de execuções e a baixa variabilidade dos resultados, não possibilitou uma amostragem de máximos adequada às distribuições pretendidas dentro do método do *EVT*. Este ajuste "fraco" explica os resultados observados anteriormente da divergência de valores assintóticos do Cortex M3 para 10.000 e 1.000.000 e 3.000.000 de execuções. Na figura 8 da seção de Apêndices são apresentados resultados complementares da análise dos casos dessa seção, como por exemplo o gráfico de *box-plot* que contém a distribuição de testes estatísticos das amostras utilizadas.

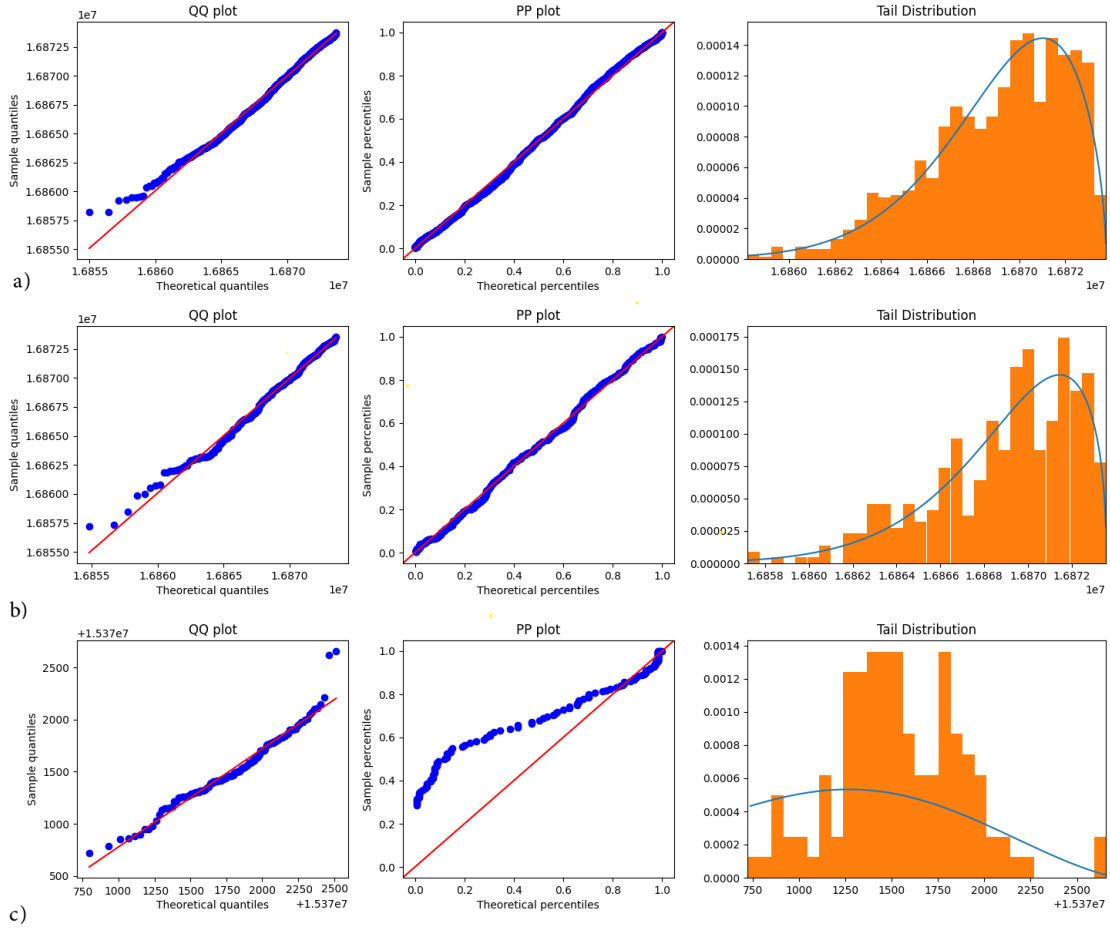
Tabela 5. Resultados do pWCET para o bsort no ESP32-S2 (Xtensa)

		10.000 amostras				30.000 amostras			
		1e-9	1e-10	1e-11	1e-12	1e-9	1e-10	1e-11	1e-12
ESP32-S3	bsort10	o	o	o	o	o	o	o	o
	bsort20	o	o	o	o	o	o	o	o
	bsort30	o	o	o	o	o	o	o	o
	bsort100 wcet	147838660*	658615620*	300368665*	137703392*	x	x	x	x
	bsort100 ga	†	†	†	†	†	†	†	†
	bsort1000	†	†	†	†	†	†	†	†

Nota: calculados pWCET para as seguintes probabilidades de excedência: 1e-9, 1e-10, 1e-11 e 1e-12.

Legenda: (x) não realizado; (o) nenhuma variabilidade; (*) baixa confiabilidade; (†) baixa variabilidade com resultados não representativos

Figura 4. Resultados estatísticos do *benchmark* minvers5 em Cortex-M3



a) 3.000.000, b) 1.000.000, c) 10.000

A metodologia *Dynamic GA* está representada na tabela 5 com a aplicação do algoritmo *Bubble Sort*. Conforme demonstrado na seção 5.1, essa técnica revelou-se adequada para este *benchmark* específico. Os experimentos listados são para as três arquiteturas, no entanto, somente no ESP32-S3 obteve variabilidade mínima para a análise do *EVT*, sendo, portanto, a única arquitetura que apresenta resultados na tabela. Mesmo para 1.000.000 execuções a variabilidade obtida foi marginal para aplicação da metodologia probabilística, inviabilizando o aumento do número de execuções na plataformas alvo pois isso levariam a períodos exponencialmente crescentes. Dentre as arquiteturas testadas o ESP32-S3 apresenta as seguintes complexidades de hardware que podem ter contribuído para esse resultado: caches de instrução e de dados, dois núcleos e *pipeline* com cinco estágios.

Analisando os resultados obtidos na tabela 5, observou-se que o cálculo de WCET probabilístico possui baixa confiabilidade para o *benchmark* bsort. Isso acontece devido a natureza do algoritmo de ordenação e de como ele é implementado no microcontrolador. Embora o GA seja capaz de encontrar uma solução ótima para o *Bubble Sort*, os resultados obtidos pelos experimentos impossibilitaram a aplicação da metodologia *EVT* devido a isso. Adicionalmente, no apêndice é possível observar o histogramas das amostras e alguns resultados estatísticos que foram reprovados na validação do *EVT* aplicado ao bsort.

Seguindo a ideia de análise de custo computacional da metodologia realizada em 5.1, o

EVT em várias arquiteturas e *benchmarks* demora aproximadamente até 1 minuto para finalizar a sua execução. Portanto, é uma metodologia escalável e apesar de não ter sido executado nos experimentos em AVR, o seu tempo de execução é desprezível comparando ao tempo que o Algoritmo Genético necessita para convergir.

6. Conclusão

Este trabalho apresentou a ferramenta FiOWAT, um novo programa para estimar o WCET de um programa ANSI C. O FiOWAT implementa cinco metodologias diferentes que combinam técnicas como IPET, medições dinâmicas de MBTA, Algoritmo Genético e teoria estatística do EVT, para fornecer estimativas precisas e seguras do WCET.

As metodologias foram analisadas através de experimentos com os algoritmos *Bubble Sort*, *Insertion Sort* e *Matrix Inversion*, em diferentes arquiteturas. Inicialmente, foram conduzidos experimentos no microcontrolador ATmega328, que possui uma arquitetura simples e de fácil validação dos resultados.

Em termos de precisão, esses experimentos mostraram que as metodologias *WPEVT* e *Dynamic GA* são as que mais se aproximam do valor real de WCET dos programas. No entanto, elas possuem os maiores tempos de execução e precisariam ser otimizadas para utilização em escala industrial. A metodologia *Hybrid IPET* tem um custo temporal significativamente menor que as duas e superestima o valor do WCET em média em 17%. Por fim, a metodologia *Static IPET* é a mais rápida de todas, mas também a menos precisa, com superestimações em de mais de cinco vezes o valor real.

Na segunda análise deste trabalho, as metodologias probabilísticas *Dynamic GA* e *EVT* foram analisadas com respeito a viabilidade do uso do *EVT* em termos da obtenção de resultados que apresentam confiabilidades altas de 10^{-9} utilizando menos execução em plataforma alvo. Nesse sentido o método mostrou-se eficiente e confiável para alguns dos *benchmarks* testados, em que só mostrou-se a viabilidade de seu uso a partir de 10^6 execuções. Outra investigação foi a aplicabilidade de dois tipos de *benchmarks* em que o *Bubble Sort* foi rejeitado na maior parte dos testes devido a baixa variabilidade dos tempos de execução causados pela combinação de algoritmo e poucas execuções em target para estimação do pWCET, já no caso do minvers, a combinação algoritmo, quantidade de execuções foi adequada para os testes. Por fim, a influência das arquitetura na contagem de ciclos foi confirmada para as arquiteturas que trazem maior variabilidade de comportamento implementados via hardware, no caso dos testes executados.

Finalmente, algumas sugestões para trabalhos futuros são: fornecer suporte a hardwares mais complexos até atingir o *multicore*; incluir a execução concólica para solucionar o problema do pior caminho; incluir um analisador melhor de limite de laços no FiOWAT; aperfeiçoar a interface gráfica; aperfeiçoar o analisador estático para mais arquiteturas, adicionando suporte ao analisador de cache, *branch predictor, pipeline*, entre outros; aumentar o desempenho da execução de programa nas placas de desenvolvimento; e análises sob o mesmo código utilizando técnicas como *checksum*, para somente analisar o programa parcialmente ao invés dele como um todo.

Referências

Abella, J., Padilla, M., Castillo, J. D., and Cazorla, F. J. (2017). Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4).

Abella, J., Quiñones, E., Wartel, F., Vardanega, T., and Cazorla, F. J. (2014). Heart of gold: Making the improbable happen to increase confidence in mbpta. In *IEEE, Euromicro Conference on Real-Time Systems 2014 (ECRTS'14)*, pages 255–265.

Acaro, L. (2019). *Increasing the Reliability and Applicability of Measurement-Based Probabilistic Timing Analysis*. PhD thesis.

Andersson, B., de Niz, D., Moreno, G., Hansen, J., Klein, M., et al. (2023). Assessing the use of machine learning to find the worst-case execution time of avionics software. Technical report, United States. Department of Transportation. Federal Aviation Administration ....

Atmel Corporation (2016). *ATmega328P*.

Baldoni, R., Coppa, E., D'elia, D. C., Demetrescu, C., and Finocchi, I. (2018). A survey of symbolic execution techniques. *ACM Computing Surveys (CSUR)*, 51(3):1–39.

Bate, I. and Khan, U. (2011). Wcet analysis of modern processors using multi-criteria optimisation. *Empir Software Eng*.

Berezovskyi, K., Santinelli, L., Bletsas, K., and Tovar, E. (2014). Wcet measurement-based and extreme value theory characterisation of cuda kernels. In *ACM.Proceedings of the 22nd International Conference on Real-Time Networks and Systems.*, page 279.

Bernat, G., Colin, A., and Petters, S. (2002). Wcet analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 279–288.

Bjørner, N., de Moura, L., Nachmanson, L., and Wintersteiger, C. M. (2019). Programming z3. *Engineering Trustworthy Software Systems: 4th International School, SETSS 2018, Chongqing, China, April 7–12, 2018, Tutorial Lectures 4*, pages 148–201.

Bünte, S., Zolda, M., and Kirner, R. (2011). Let's get less optimistic in measurement-based timing analysis. In *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, pages 204–212.

Cadar, C., Dunbar, D., Engler, D. R., et al. (2008). Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, volume 8, pages 209–224.

Carminati, A. et al. (2017). Contributions to worst-case execution time reduction using compilation techniques. *Universidade Federal de Santa Catarina*, pages 76–77; 142–149.

Cazorla, F. J., Abella, J., Andersson, J., Vardanega, T., Vatrinet, F., Bate, I., Broster, I., Azkarate-Askasua, M., Wartel, F., Cucu, L., Cros, F., Farrall, G., Gogonel, A., Gianarro, A., Triquet, B., Hernandez, C., Lo, C., Maxim, C., Morales, D., Quinones, E., Mezzetti, E., Kosmidis, L., Aguirre, I., Fernandez, M., Slijepcevic, M., Conmy, P., and Talaboulma, W. (2016). Proxima: Improving measurement-based timing analysis through randomisation and probabilistic analysis. In *IEEE, Euromicro Conference on Digital System Design 2016 (DSD'16)*, pages 276–285.

Coles, S. G. (2001). *An Introdution to Statiscal Modeling of Extreme Values*. Springer.

Cucu-Grosjean, L., Santinelli, L., Houston, M., Lo, C., Vardanega, T., Kosmidis, L., Abella, J., Mezzetti, E., Quiñones, E., and Cazorla, F. J. (2012). Measurement-based probabilistic timing analysis for multi-path programs. In *IEEE, Euromicro Conference on Real-Time Systems 2012 (ECRTS'12)*, pages 91–101.

Davis, R., Vardanega, T., Alexanderson, J., Francis, V., Mark, P., Broster, I., Mikel, A.-A., Wartel, F., Cucu-Grosjean, L., Mathieu, P., Glenn, F., and Cazorla, F. (2014). Proxima: A probabilistic

approach to the timing behaviour of mixed-criticality systems. *Ada User Journal (AUJ)*, pages 118–122.

Espressif Systems (2023a). *ESP32-C3 Series - Datasheet*.

Espressif Systems (2023b). *ESP32 Series - Datasheet*.

Fachini, G. J. d. A. (2011). Modular and generic wcet static analysis with llvm framework. *Universidade Federal do Rio Grande do Sul*.

Griffin, D. and Burns, A. (2010). Realism in statistical analysis of worst case execution times. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Gustafsson, J., Betts, A., Ermedahl, A., and Lisper, B. (2010). The Mälardalen WCET benchmarks – past, present and future. In Lisper, B., editor, *WCET2010*, pages 137–147, Brussels, Belgium. OCG.

Jiménez Gil, S., Bate, I., Lima, G., Santinelli, L., Gogonel, A., and Cucu-Grosjean, L. (2017). Open challenges for probabilistic measurement-based worst-case execution time. *IEEE Embedded Systems Letters*, 9(3):69–72.

Kosmidis, L., Quiñones, E., Abella, J., Vardanega, T., Hernandez, C., Gianarro, A., Broster, I., and Cazorla, F. J. (2016). Fitting processor architectures for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems*, 47:287–302.

Kroening, D., Schrammel, P., and Tautschnig, M. (2023). Cbmc: The c bounded model checker.

LLVM (2023). Clang compiler user's manual.

Oliveira, R. S. (2020). *Fundamentos dos Sistemas de Tempo Real*. Independently Published.

Puaut, I. Worst-case execution time (wcet) estimation: from monocore to multi-core architectures.

PyGaD (2023). Pygad documentation.

Queiroz, M. (2014). O método simplex.

Rashida, M., Shah, S. A. B., and Arif, M. (2020). Determination of worst-case data using adaptive surrogate models for real-time system. *Journals of Circuits, Systems and Computers*, 29(11).

Rierson, L. (2013). *Developing Safety-Critical Software: A Pratical Guide for Aviation Software and DO-178C Compliance*. CRC Press.

Rufino, L. M. (2009). Análise de tempo de execução utilizando llvm. *Universidade Federal de Santa Catarina*.

Silva, K. P. et al. (2015). Análise de valor para determinação do tempo de execução no pior caso (wcet) de tarefas em sistemas de tempo real. *Universidade Federal de Santa Catarina, Centro Tecnológico*.

Soares, G. L. (1997). Algoritmo genético: Estudo, novas técnicas e aplicações. Master's thesis.

STMicroelectronics (2021). *RM0008 Reference manual*.

Theiling, H. (2002). Control flow graphs for real-time systems analysis. *Universität des Saarlandes, Diss*.

Toledo, F. and Costa, A. (2019). Branch and bound.

Wartel, F., Kosmidis, L., Lo, C., Triquet, B., Quinones, E., Abella, J., Gogonel, A., Baldovin, A., Mezzetti, E., Cucu, L., et al. (2013). Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 241–248. IEEE.

Wegener, F. e. a. (1996). Systematic testing of real-time systems. *4th International Conference on Software Testing Analysis and Review*.

Wenzel, I., Kirner, R., Rieder, B., and Puschner, P. (2008). Measurement-based timing analysis. volume 17, pages 430–444.

Wilhelm, R. (2020). Determining reliable and precise execution time bounds of real-time software. *IT Professional*, 22(3):64–69.

Wilhelm, R. et al. (2008). The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53.

Zolda, M. and Kirner, R. (2016). Calculating wcet estimates from timed traces. *Real-Time Systems*, 52(1):38 – 87. Cited by: 3; All Open Access, Hybrid Gold Open Access.

7. Apêndice

Segue abaixo, as figuras sobre os gráficos dos hiper-parâmetros do Algoritmo Genético:

Figura 5. Metodologias de seleção de pais por ciclos de execução

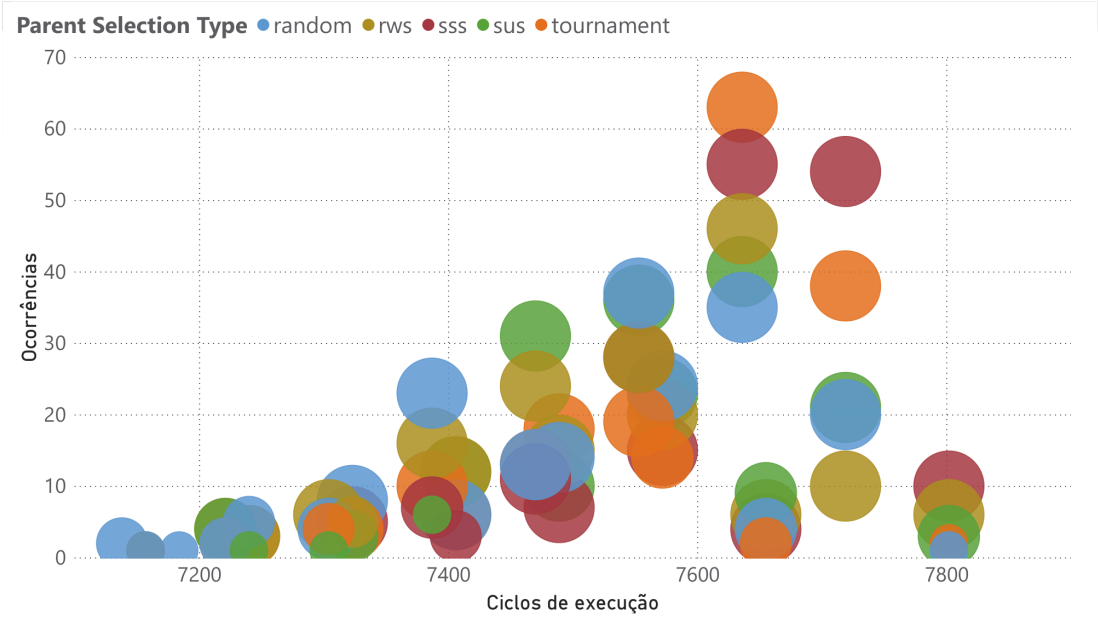


Figura 6. Média de ciclos de execução e média de tempo de execução por metodologia de mutação

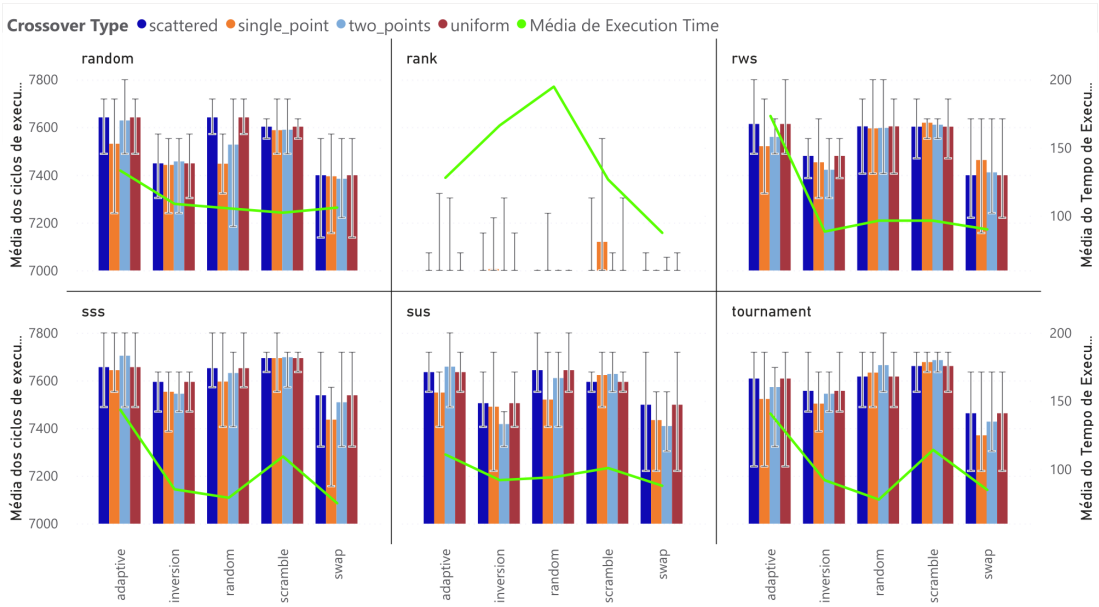


Figura 7. Média de ciclos de execução e média de tempo de execução do Algoritmo Genético por metodologia de cruzamento

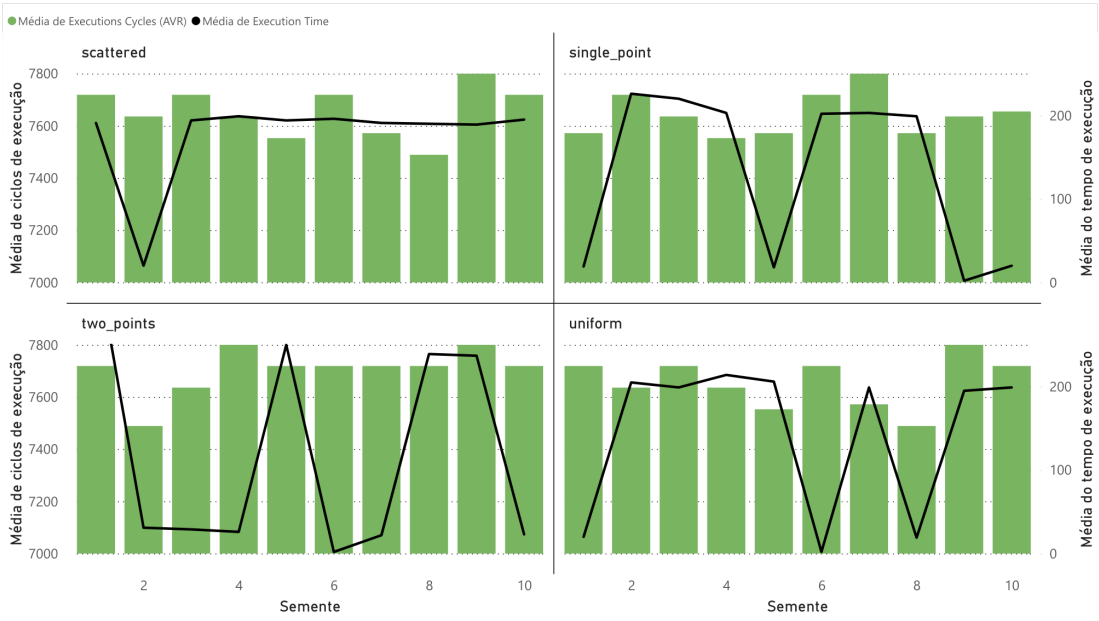
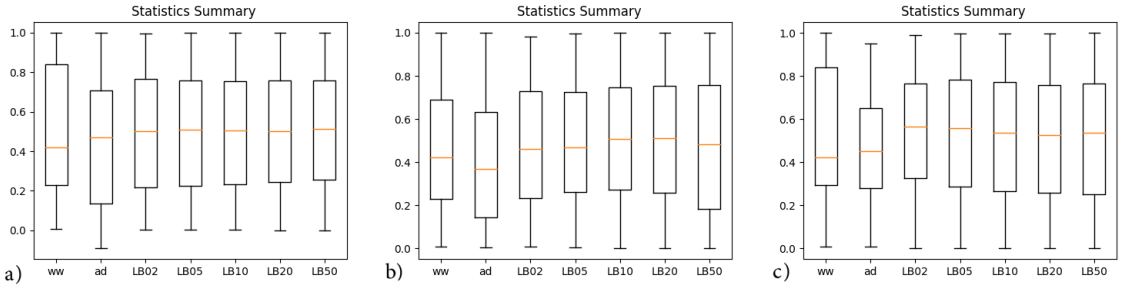


Figura 8. Resumo dos testes estatíticos para Cortex-M3, minverse_5



a) 3.000.000, b) 1.000.000, c)10.000

Figura 9. Resumo do EVT ESP32-S3[Xtensa]- BSort_10_EVT, 10.000 execuções

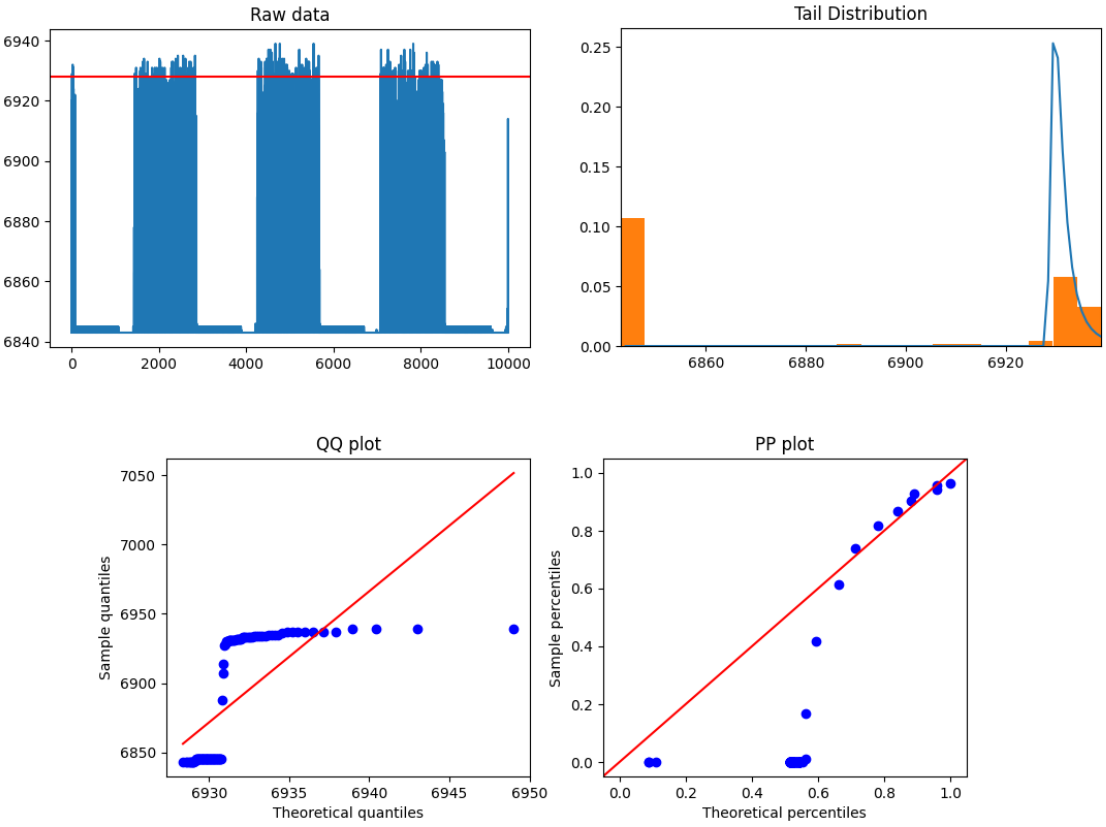


Figura 10. Resumo do EVT ESP32-S3 - BSort_100_EVT, 10.000 execuções

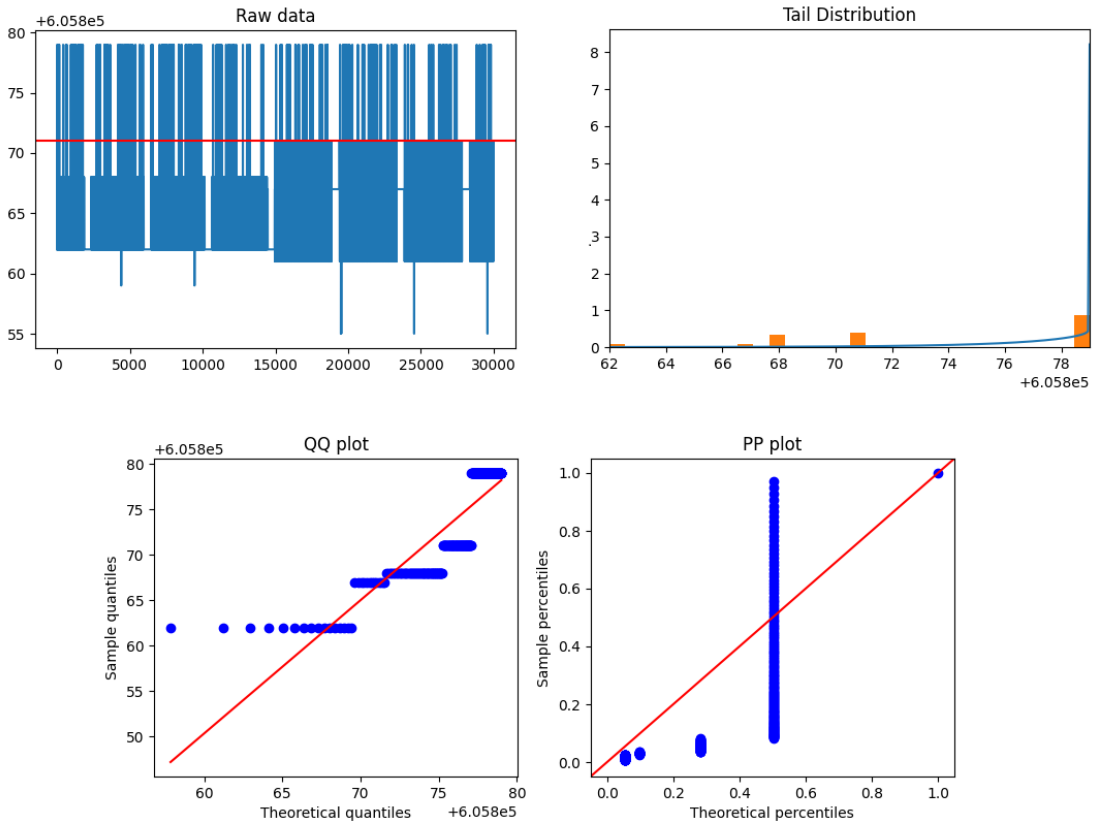


Figura 11. Resumo do EVT ESP32-S3[Xtensa] - BSort_100_*Dynamic GA*, 10.000 execuções