Explorando a estratégia híbrida modular para determinar o Worst-Case Execution Time

Felipe D. A. Brito, Gustavo C. E. Santo, Ingrid F. V. Oliveira, Lucas G. B. Cunha

[1]Centro de Informática - Universidade Federal de Pernambuco (UFPE) - Pernambuco, Brasil
Programa de Especialização em Software Embarcado
Embraer - Brasil

{fdab,gces,ifvo,lgbc}@cin.ufpe.br

Abstract. *The technological advance has ushered in a significant increase in the adoption of real-time systems across diverse applications. These systems must ensure the accuracy of their outputs and, more critically, that the outputs are generated within the designed time. In hard real-time systems, these temporal requirementes are more stringent. Failure to meet deadlines in these systems can lead to significant damage or loss of lives. Consequently, having a precise understanding of the Worst-Case Execution Time (WCET) for a task is crucial. This article describes a method for establishing upper bounds for the WCET of a program coded in the C language, using measurement and static analysis techniques to explore hardware and software variability in execution time.*

Resumo. *O avanço tecnológico possibilitou o aumento significativo na utilização de sistemas de tempo real em diversas aplicações. Esses sistemas precisam garantir que suas saídas estejam corretas e, principalmente, que essas sejam geradas no tempo esperado. No caso de sistemas de tempo real críticos, esses requisitos temporais são mais rigorosos, visto que o não cumprimento dos seus prazos pode acarretar danos graves ou até mesmo a perda de vidas. Como consequência, conhecer o o Pior Tempo de Execução (WCET) de uma tarefa é primordial para esses sistemas. Portanto, este artigo descreve um método de obtenção dos limites superiores para o WCET de um programa em linguagem C. Métodos de medição e análise estática são utilizados para explorar a variabilidade de hardware e software no tempo de execução.*

Sumário

1. Introdução

Os sistemas de tempo real (STRs) são sistemas computacionais que atendem a requisitos temporais. Esses requisitos são expressos como prazos (*deadlines*) para execução de tarefas do sistema, onde a confiabilidade do sistema está diretamente relacionada com sua habilidade de executar as tarefas dentro dos prazos, mesmo no pior cenário de execução. Caso esses prazos não sejam atendidos, dependendo da criticalidade do sistema de tempo real, as consequências podem ser desde inexistentes até acarretar na perda de vidas.

Um passo fundamental para quantificar a confiabilidade de um STR é determinar os limites superiores para os tempos de execução das tarefas no pior caso (*Worst Case Execution Time - WCET*), ou seja, o tempo máximo que pode ser gasto pelo hardware alvo (*target hardware*) para executá-las [Arcaro et al. 2018]. Entretanto, determinar o WCET tem se tornado um desafio a medida que os sistemas modernos adotam arquiteturas de computadores mais complexas. Desta forma, diversas abordagens foram propostas na literatura para determinação dos limites de WCET.

A abordagem estática para determinar o WCET é conhecida por gerar informações confiáveis, resultantes da modelagem do sistema, baseando-se em quatro etapas principais: reconstrução do fluxo de controle do software, análise de valor, análise da microarquitetura e obtenção do pior caminho. No entanto, a medida em que os processadores tornam-se mais complexos (memória cache, *branch predictors*, *pipelines*), a modelagem completa é intratável devido à grande quantidade de estados. Como resultado, a modelagem imprecisa aumenta o pessimismo, tornando-se um fator dominante neste tipo de análise [Silva 2015]. Já a abordagem baseada em medição analisa os tempos de execução das tarefas produzidos em *runtime*, levando à redução de esforços quanto a análise e sendo potencialmente aplicável a arquiteturas complexas. Porém, neste caso, o valor obtido para o maior tempo de execução (*High Water Mark - HWM*), dificilmente será o WCET. Deste modo, é necessário estabelecer margens de segurança que levam em consideração os casos de tempo possivelmente não observados durante o processo de coleta das medições [Arcaro et al. 2018].

A análise de tempo probabilística baseada em medição (*Measurement-Based Probabilistic Timing Analysis - MBPTA*) visa estabelecer limites para os WCETs das tarefas através da análise de medições de tempo de execução usando abordagem estatística e, desta forma, estimar um limite superior de tempo de execução, no qual a probabilidade de excedência esteja abaixo de um limiar aceitável para o projeto ($10^{-9} \sim 10^{-15}$). A teoria dos valores extremos (*Extreme Value Theory - EVT*) é uma das abordagens mais utilizadas, consistindo na análise estatística capaz de estimar a probabilidade de acontecimento de eventos extremos incomuns, a partir da observação de desvios da modelagem do comportamento típico do fenômeno analisado [Arcaro et al. 2018].

Por fim, a análise híbrida combina elementos de análise estática e análise base-

ada em medição. O objetivo dessa análise é superar as desvantagens da análise estática e dos métodos dinâmicos. A utilização de técnicas de análise estática como determinação do grafo de fluxo de controle, em conjunto com a análise de valor (execução abstrata), permite explorar o problema da variabilidade de software e assim encontrar os valores de entrada que exercitam o caminho do pior caso. Além disso, com a execução de um conjunto de medições no hardware alvo, utilizando as entradas obtidas para o caminho crítico, a variabilidade causada pelo hardware do sistema pode ser contemplada, e desta forma, um pWCET, a probabilidade de excedência para um determinado valor de WCET, pode ser obtido a partir de um tratamento estatístico adequado [Davis and Cucu-Grosjean 2019].

Deste modo, este trabalho descreve a utilização de um método híbrido e modular para a obtenção de um limite superior para o WCET de programas escritos em linguagem C. O termo "Híbrido" deriva do fato de que métodos de análise dinâmica e estática foram concebidos para serem utilizados simultaneamente, aproveitando assim as vantagens que cada um oferece, ao mesmo tempo que as deficiências são compensadas de maneira análoga. O termo "Modular" expressa a característica de generalização da abordagem, que se apresenta como um conjunto de etapas coesas com saídas bem definidas e que podem ser substituídas conforme as necessidades de cada caso, novas ideias e novos trabalhos surgirem.

A sequência deste artigo está organizada da seguinte forma: na Seção 2 são abordados os trabalhos relacionados disponíveis na literatura. Na Seção 3 é apresentada a metodologia utilizada no desenvolvimento desse trabalho. A metodologia é dividida em cinco etapas, onde na Subseção 3.1 é apresentada a primeira etapa de medição dos blocos básicos, na Subseção 3.2 é descrita a segunda etapa referente a execução abstrata, a Subseção 3.3 apresenta a terceira etapa da metodologia que é responsável pela determinação das entradas, já na Subseção 3.4 é descrita a quarta etapa que refere-se a medição no hardware, e na Subseção 3.5 é apresentada a quinta etapa responsável pelo MBPTA. Por fim, através da aplicação da metodologia descrita anteriormente, na Seção 4 são apresentados e discutidos os testes realizados e os resultados obtidos.

2. Trabalhos Relacionados

A literatura explora o problema do WCET em três principais vertentes: métodos de medição, métodos estáticos e uma técnica de otimização denominada IPET (*Implicit Path Enumeration Technique*). Cada estratégia apresenta suas vantagens e desafios. Além disso, diversas ferramentas foram desenvolvidas com o objetivo de abordar a problemática da obtenção do WCET, tanto no âmbito comercial quanto acadêmico. Um levantamento e análise das ferramentas disponíveis na literatura é realizado em [Wilhelm et al. 2008]. Esse artigo reúne os principais aspectos relacionados ao desafio da determinação do WCET, além de proporcionar uma visão abrangente das principais práticas da indústria

nesse contexto.

Em [Ermedahl et al. 2011] é mencionado que a abordagem do IPET é a preferida na análise do WCET atualmente. Nessa técnica, a análise de fluxo de um programa é traduzida em um conjunto de restrições lineares envolvendo variáveis contadoras para partes do programa, por exemplo, blocos básicos. Essas restrições, em conjunto com uma função objetivo, formam um problema de programação linear inteira (*Integer Linear Programming Problem - ILP*), cuja solução maximiza o valor do WCET. O autor ainda cita que o desafio dessa técnica decorre do fato de que, para estimativas mais próximas do WCET real, as restrições de fluxo precisam ser detalhadas. Entretanto, é complexo incorporar no IPET os efeitos da variabilidade no tempo de execução provocados pelo hardware, como memória cache e *pipeline*. Além disso, considerando que o ILP possui uma complexidade *NP-hard*, o problema pode tornar-se computacionalmente inviável.

Além dos principais métodos, apresentados acima, que exploram o problema da determinação do pior tempo de execução, um método alternativo é encontrado na literatura: a utilização de uma abordagem híbrida [Wilhelm et al. 2008] [Davis and Cucu-Grosjean 2019]. Essa abordagem procura, através da junção de duas ou mais técnicas, potencializar as suas vantagens e reduzir os seus aspectos problemáticos, resultando assim em uma estratégia híbrida, que devido a sua versatilidade foi a opção escolhida e desenvolvida neste trabalho.

A determinação explícita dos caminhos é uma etapa geralmente presente em métodos de análise estática, mas que pode ser vantajosamente utilizada na estratégia modular híbrida. Os desafios associados a essa etapa estão relacionados com o crescimento exponencial de possíveis caminhos quando estruturas, como loops e condicionais, são utilizadas de maneira aninhada [Wilhelm et al. 2008]. Embora o problema de enumerar todos os possíveis caminhos seja desafiador [Yen et al. 1989], a determinação de um subconjunto de caminhos que atenda a determinados critérios pode ser uma estratégia viável.

Considerando a possibilidade de representação de um software a partir da modelagem de um grafo (por exemplo, *Control Flow Graph - CFG*), de acordo com [de Rezende 2014] a busca pelo caminho mais longo em grafos acíclicos e direcionados possui uma complexidade conhecida e uma abordagem matemática formalizada, tratando-se de um problema algorítmico amplamente explorado na literatura. Em [Yen et al. 1989] é proposta a busca dos k maiores caminhos em um grafo direcionado acíclico, sendo que o critério de tamanho é determinado pelo tempo total necessário para percorrer cada caminho.

No geral, o fluxo de controle em software é naturalmente direcionado devido a sua característica de execução sequencial de instruções, o que em parte valida a mode-

lagem por meio de grafos. No entanto, a segunda restrição, referente a aciclicidade do grafo, limita a modelagem do software, visto que estruturas como loops precisariam ser desdobradas em segmentos sequenciais.

Em [Altenbernd 1996], o autor discute que simplesmente encontrar o maior caminho estrutural pode levar à superestimação do WCET caso esse caminho não seja realizável em uma execução real. Ademais, o autor propõe a ideia de analisar as instruções do código, como cabeçalhos de condicionais, na busca pelo maior caminho realizável através da utilização da **"execução simbólica"**.

A execução simbólica consiste na exploração de todos os caminhos de execução possíveis/viáveis de um programa. A cada condicional verificada no programa, novos caminhos (estados) de execução são criados com restrições de viabilidade. Essas restrições são analisadas e, caso não sejam atendidas, a execução do caminho é interrompida [Martins 2018]. Por fim, [Ermedahl et al. 2011] propõe e implementa, em uma ferramenta acadêmica chamada SWEET, o que denominou de **Execução Abstrata**, uma forma de execução simbólica baseada no *framework* de interpretação abstrata e que é utilizada para obtenção de *flow-facts* e exploração dos possíveis caminhos de execução do software.

3. Metodologia

Conforme visto na Seção 1, o objetivo deste trabalho é conceber um processo que, a partir de um código-fonte em linguagem C, atinja o limite superior do WCET. Considerando as questões levantadas na Seção 1 a respeito da problemática na determinação do WCET e as vantagens e desafios das abordagens existentes na literatura discutidas na Seção 2, optou-se por desenvolver esse processo utilizando uma abordagem híbrida adotando a estratégia de modularização das camadas.

A abordagem modular em camadas possui como características principais: a manutenibilidade e modificabilidade. A manutenibilidade define que eventuais problemas identificados estão restritos a um subconjunto de camadas. Já a modificabilidade prevê que a substituição de um subprocesso, definido em qualquer camada, pode ocorrer conforme as necessidades do momento, desde que a interfaces padronizadas entre as camadas sejam preservadas. Dessa forma, projetos futuros podem reutilizar a estrutura geral do processo e adaptá-la conforme seja necessário.

A metodologia desenvolvida pode ser dividida em cinco principais etapas que envolvem conceitos de análise estática para o estudo do fluxo do programa e de análise dinâmica com o uso de medições. As três primeiras etapas são aplicáveis para programas que apresentam fluxo múltiplo de execução dependente de entradas. Enquanto que no caso de programas de fluxo único de execução, a metodologia deve ser observada a partir da quarta etapa, conforme ilustrado no fluxograma da Figura 6 no Apêndice A. Es-

sas etapas são elencadas abaixo e serão exploradas com mais detalhes ao decorrer desta seção.

- **Etapa 1:** Medição dos blocos básicos do código.
- **Etapa 2:** Execução abstrata para obtenção do pior caminho (*Worst-Case Execution Path - WCEP*).
- **Etapa 3:** Obtenção das entradas que provocam o WCEP.
- **Etapa 4:** Medição do código completo no Hardware.
- **Etapa 5:** Análise estatística (MBPTA).

Para realizar as etapas descritas acima, buscou-se o auxílio de ferramentas de software e hardware para montar o processo. Essa busca resultou no desdobramento de um conjunto de passos. Primeiramente, buscou-se a conversão do código em linguagem C para um arquivo ALF (linguagem de representação intermediária. Em seguida é realizada a criação do arquivo TDB, um arquivo *template* para o tempo de execução dos blocos básicos (BBs). Então, cria-se o arquivo ANN, um arquivo *template* para as anotações no código. Após a criação desses arquivos, realiza-se a análise de correspondência dos BBs do programa, ALF para C. A partir de então é realizada a medição do tempo de execução de cada bloco básico e realiza-se a execução abstrata utilizando a ferramenta SWEET.

Na ferramenta SWEET são inseridos os arquivos ALF, TDB e ANN para obtenção do WCEP, em termos dos BBs do código ALF, além de uma estimativa do WCET. A partir do conhecimento obtido em uma primeira execução do SWEET, realiza-se uma nova execução abstrata para inferir as entradas que estimulam o pior caminho. Através do uso das entradas resultantes do passo anterior, realiza-se a medição do tempo de execução do WCEP, de ponta a ponta, em hardware físico. E por fim, é realizado um tratamento estatístico dos dados de WCET obtidos. Esse conjunto de passos podem ser observados no fluxograma ilustrado na Figura 7 do Apêndice A.

3.1. Etapa 01 - Identificação e Medição dos blocos básicos

3.1.1. Sobre o ALF

Para utilização da ferramenta SWEET, a qual é discutida na Subseção 3.2, é necessária a preparação de arquivos auxiliares, a partir do código C a ser avaliado, os quais são utilizados como arquivos de entrada para execução da ferramenta. Dentre eles, o principal é o arquivo de representação intermediária chamado ALF (*ARTIST2 Language for WCET Flow Analysis*).

O ALF é uma linguagem intermediária, desenvolvida pela equipe da Universidade de Mälardalen, no intuito de ser usada em análises de fluxo para cálculo do WCET (*Worst Case Execution Time*). De modo geral, a análise de fluxo pode ser realizada tanto

em código-fonte, código intermediário ou código binário. No entanto, a análise feita em cada um dos formatos de código pode trazer diferentes informações referentes a um mesmo artefato em questão:

- **Código Binário:** A análise do código binário garante que seja encontrado o fluxo de informações correto que é executado no processador. No entanto, as informações disponíveis no código-fonte podem ser perdidas no binário, como por exemplo, informações acerca do tipo de dado (inteiro, ponto flutuante, ponteiros) e estrutura de controle, o que pode levar a uma análise de fluxo menos precisa;
- **Código-Fonte:** O código-fonte normalmente pode ser analisado com mais detalhes, uma vez que há muito mais informações acerca do tipo de dados, fluxo de controle, estruturas de dados, definição de funções e chamadas. Por outro lado, as otimizações do compilador podem fazer com que a estrutura de controle do código binário gerado seja diferente daquele do código fonte;
- **Código Intermediário:** O código-fonte e o código intermediário gerado a partir dele geralmente são aproximadamente isomórficos. Além disso, após as otimizações, o código intermediário tem um fluxo de programa próximo ao fluxo do código binário gerado. Essas propriedades tornam o código intermediário um candidato interessante para análise de fluxo, uma vez que pode ser analisado quanto às propriedades de fluxo do código-fonte e binário.

Desse modo, a proposta do ALF é ser uma linguagem genérica para análise de fluxo utilizada em WCET, a qual pode ser gerada a partir de qualquer uma das representações de código mencionadas acima, agrupando as principais informações que a análise isolada em cada formato de código trariam, em um único código [Gustafsson et al. 2009].

ALF faz parte do desenvolvimento do projeto ALL-TIMES, cuja proposta consiste na interoperabilidade das diversas ferramentas dos principais fornecedores comerciais (Rapita, AbsInt) e universidades, a fim de desenvolver ferramentas integradas usando estruturas e interfaces de ferramentas open-source.

3.1.2. Geração dos arquivos

Como abordado anteriormente, o SWEET utiliza o formato ALF, que é uma representação intermediária do código, por possuir características próximas tanto do código-fonte, quanto do código binário, permitindo uma análise mais precisa do programa.

Para conversão do código em C para ALF, é utilizada a ferramenta **AlfBackend**, um tradutor de C para ALF baseado em LLVM, desenvolvido na Universidade Técnica de

Viena, na Áustria [Gustafsson 2019]. Além disso, a ferramenta permite a criação de um arquivo de mapeamento entre o formato ALF e o código-fonte original em C. O arquivo de mapeamento realiza a correspondência entre blocos básicos no código ALF para o código em C. O código ALF é então utilizado no SWEET em todas as etapas posteriores.

O primeiro arquivo gerado é o template de tempos de execução dos BBs (arquivo TDB), o qual organiza, lado a lado, os blocos básicos no formato ALF com o tempo de execução de cada bloco Figura 1. A técnica para obtenção do tempo de execução de cada bloco é explorada na Subsubseção 3.1.3.

```
main::bb 0
main::bb::1 0
binary_search::bb 0
binary_search::bb1 0
binary_search::bb2 0
binary_search::bb9 0
binary_search::bb14 0
binary_search::bb19 0
binary_search::bb21 0
binary_search::bb23 0
binary_search::bb24 0
binary_search::bb25 0
```

Figura 1. Exemplo ilustrativo do formato tdb.

O segundo arquivo gerado é o template para anotação de código (arquivo ANN), o qual permite indicar quais valores ou intervalos a execução abstrata deverá explorar para certas variáveis em determinados locais. A Figura 2 apresenta um exemplo de anotação, indicando que, na entrada da função *"binary_search"*, a variável x poderá assumir os valores inteiros de 1 a 20.

```
/* ------------------------------------------------------- */
/* Annotation templates for imported data (frefs)          */
/* ------------------------------------------------------- */

/* Given annotations should provide a safe upper bounds on what
   value a data, or specific fields of a data, may hold.
   The annotations should be valid for any execution of the
   program (for all its possible input value combinations).
   Please run: sweet -h topic=annot  for annotation syntax   */

FUNC_ENTRY "binary_search" ASSIGN "%x" INT 1 20;
```

Figura 2. Exemplo ilustrativo do formato ann.

A utilização do arquivo de anotações é crucial para estabelecer limitações na execução abstrata, cuja ideia principal é utilizá-lo para **limitar a faixa** de valores que

uma ou mais **entradas** poderão assumir na execução do código. Essa funcionalidade é explorada também na Subseção 3.3 para identificação das entradas que provocam o WCEP.

No Seção 4.1.3 é fornecido um tutorial passo a passo para instalação das ferramentas, e algumas dicas de uso, como o uso de scripts para Linux que automatizam a criação dos arquivos ALF, mapeamento, e templates TDB e ANN.

Para que seja possível realizar a estimação do WCET, uma das etapas necessárias é a definição do tempo de execução de cada um dos blocos básicos (BB) do código avaliado. Dito isso, diversas são as maneiras para se obter tais medidas, como a utilização de um osciloscópio ou a instrumentação do código, sendo esta última, o foco da metodologia elaborada.

A instrumentação do código consiste na inserção de códigos que visam capturar a quantidade de tempo, ou ciclos, que um determinado conjunto de instruções demorou para ser executado, tendo que no cenário estudado, cada conjunto de instruções instrumentado é um bloco básico. Tal processo, pode ser realizado em diferentes níveis de abstração, desde cenários mais amigáveis, porém menos precisos, como o código fonte, até a instrumentação do próprio *assembly*, que, apesar de mais complexo, possibilita uma maior precisão.

Além disso, é importante salientar que, embora haja um ganho de precisão, através da instrumentação em assembly, o processo de compilação ainda realizará transformações no código, o que, por sua vez, pode acarretar em otimizações que culminem na alteração da ordem de execução de determinadas instruções do programa binário. Consequentemente, a precisão dos dados obtidos através da instrumentação pode ser afetada, uma vez que tal comportamento pode proporcionar uma superestimação neles, isto é, um conjunto de instruções pode acabar sendo contabilizado com mais ciclos do que realmente consumiu [Gustafsson et al. 2009].

Nesta etapa, foram medidos os tempos de execução de partes do programa. Definiu-se essas partes como sendo os blocos básicos do código (BB), seções compostas apenas por instruções sequenciais sem desvios de fluxo (condicionais e chamadas de funções).

A saída desta etapa é uma tabela que mapeia BBs para o tempo de execução correspondente.

3.1.3. Medida no código fonte C

A instrumentação do código fonte em C consiste, basicamente, em computar, com o auxílio de funções, o número de ciclos gastos entre uma instrução inicial, a qual demarca

o começo de um BB equivalente no ALF, e uma instrução final que corresponde ao fechamento do mesmo BB (através do arquivo de mapeamento de código ALF para código C). Ou seja, para medir o custo de um BB no ALF utiliza-se a medição no código C em um trecho que corresponda ao BB em questão. Para isso, foi utilizado a biblioteca disponível em [Malich 2017], que implementa um contador de ticks para microcontroladores AVR.

A biblioteca em questão, implementa quatro funções, as quais podem ser vistas abaixo:

- void ResetTickCounter(void);
- void StartTickCounter(void);
- void StopTickCounter(void);
- ticks_t GetTicks(void);

A função ResetTickCounter é responsável por reiniciar o contador de ticks, a função StartTickCounter tem como objetivo iniciar a contagem de ticks, StopTickCounter pausa a contagem de ticks até que a mesma seja, novamente, iniciada. Por último, GetTicks retorna a quantidade de ticks contabilizados no período entre a chamada de StartTickCounter e StopTickCounter.

Com o código instrumentado, é necessário realizar o cross-compiling, visando as configurações do microcontrolador AVR desejado. Para isso, foi utilizado o compilador avr-gcc, que possibilita a geração de código para AVR a partir de um código-fonte em linguagem C, especificando a geração para o microcontrolador Atmega328, o qual não possui memória cache e pipeline.

Após a compilação, é possível realizar a execução no microcontrolador, para isso foi utilizado o programa SimulAVR, um simulador para microcontroladores da família AVR, que possibilita a instanciação de um Atmega328 e a, subsequente, execução do código compilado. Por conseguinte, é possível obter o número de ciclos gastos para a execução do trecho de código em C que corresponde a um BB no código intermediário ALF.

A relação de tempos medidos dos BBs, correspondentes ao código ALF, é então reunida em um arquivo TDB para a realização da execução abstrata na ferramenta SWEET Subseção 3.2.

3.1.4. Problemas, Limitações e Ressalvas

Durante a realização das medições notou-se uma limitação no SimulAVR, relacionada ao tamanho da RAM, de 32 KB, haja visto que certos Benchmarks ultrapassavam esse valor e portanto não era possível carregá-los na memória. A fim de abordar este problema, os

benchmarks que se enquadravam nesse cenário foram submetidos a uma das seguintes abordagens:

1. Não prosseguir com o Benchmark;
2. Ajustar o código do Benchmark para ocupar menos espaço na RAM.

Para a última, abordagens como reduzir o tamanho das estruturas de dados instanciadas pelo Benchmark foram utilizadas, uma vez que, entendendo o objetivo do código avaliado, era possível alterá-lo, sem modificar o WCEP obtido pelos custos de tempo dos BBs que o compõem.

Além disso, é importante frisar que os valores obtidos pelo calculo dos BBs servem como uma estimativa do custo relativo dos blocos, custo esse que passa a depender da plataforma em que as medições foram realizadas, nesse caso relacionadas ao Atmega328, emulado pelo SimulAVR.

Quanto à instrumentação a nível do código-fonte, utilizando como referência os BBs definidos pelo mapeamento gerado a partir do ALFBackend, vale ressaltar que concessões foram feitas para trechos em que a medição não poderia ser realizada, como nos casos em que o mapeamento de um bloco básico, gerado a partir do código ALF, apontava para instruções em código C, como **return**, **else if** ou chaves de abertura/fechamentos - **{}**.

Haja visto que, o processo de instrumentação é feito a partir dos BBs definidos no arquivo ALF, e devido às concessões necessárias, comentadas anteriormente, a etapa de instrumentação dos blocos básicos em C foi realizada manualmente, uma vez que se fez necessário checar o arquivo ALF para uma melhor interpretação da correspondência dos BBs, como nos casos em que mais de um BB apontava para uma mesma instrução do código C.

3.2. Etapa 02 - Execução Abstrata e o SWEET

A *execução abstrata* é um método que permite o cálculo de restrições de fluxo em um código, a geração de fatos de fluxo (flow facts) e, ao mesmo tempo, uma análise de valor nas variáveis do programa. Esse método foi implementado na ferramenta open source SWEET. Em adição a isso, ao adicionar o tempo como uma variável explícita e incrementá-la para cada bloco básico processado, o método da *execução abstrata* devolve tanto o WCET quanto o BCET, junto dos caminhos explícitos que os realiza. [Ermedahl et al. 2011]

A ferramenta SWEET (SWEdish Execution Time analysis tool) foi desenvolvida na Mälardalen University na Suécia para prover métodos de cálculo do WCET para software de tempo real crítico [Lisper 2014]. O software implementa ferramentas de análise de fluxo (como a execução abstrata, foco desse trabalho) e análises de baixo nível para os processadores NECV850E e ARM9. Figura 3

Com o código no formato intermediário ALF, os templates de anotações e custos de BBs preenchidos, a etapa atual propõe encontrar o caminho crítico WCEP (Worst Case Execution Path) por meio da execução abstrata. A ferramenta consegue ao executar o programa em um domínio abstrato calcular possíveis valores para variáveis em diferentes pontos do programa de maneira contextual, associando-os à valores abstratos. Por exemplo, se a execução abstrata utilizar o domínio de intervalos, o conteúdo de uma variável é o intervalo com todos os valores que ela pode assumir em um ponto do programa. A coleção de todos as variáveis abstratas e seus valores possíveis corresponde a um estado abstrato.

A execução abstrata permite determinar:

- Limites inferiores e superiores para loops e loops aninhados.
- Limites superiores para a execução de BBs.
- Pares de BBs inviáveis.
- Caminhos inviáveis.
- O pior caminho de execução (WCEP) e o seu tempo (WCET).
- O melhor caminho de execução (BCEP) e o seu tempo (BCET).

Para esse trabalho a técnica da execução abstrata foi utilizada pelo SWEET para a produção de uma estimativa para o WCET e a determinação do pior caminho de execução WCEP que o provoca [Ermedahl et al. 2011].

A Figura 3 resume o fluxo do uso da ferramenta, evidenciando o uso dos arquivos TDB, ANN e ALF como entradas, identificados como **timing model**, **input value constraints** e **.alf**, correspondentes com os tempos dos BBs medidos, o arquivo de anotação e o código em ALF, respectivamente. A saída da execução abstrata identificada como **WCET bound** é o valor para o WCET e o pior caminho de execução WCEP.

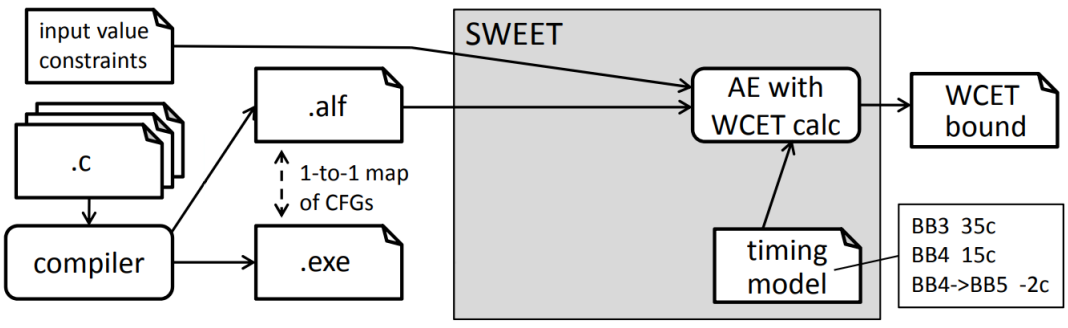

Figura 3. Uso do SWEET para Execução Abstrata.

3.2.1. Problemas, Limitações e Ressalvas

Como introduzido na Subseção 3.2 a Execução Abstrata lida com estados abstratos, que é uma coleção de todos as variáveis abstratas e seus possíveis valores em um determinado ponto no programa. Cada caminho de execução pode corresponder a um estado. Por exemplo, se houver uma estrutura condicional ***if***, que pode ser tanto avaliada como verdadeiro ou falsa, a execução abstrata seguirá os dois fluxos de forma independente, cada um representando um estado diferente.

Na execução abstrata, todos os caminhos de execução possíveis (estados) são tratados simultaneamente. Além disso, certas estruturas como loops com condicionais internas resultam em um rápido aumento no número de estados [Wilhelm et al. 2008]. Técnicas de *merge* permitem a fusão dos estados abstratos em pontos específicos, acelerando a execução abstrata, no entanto, ao custo da perda de informação, afetando a precisão da estimativa do WCET, tornando-a mais pessimista. Portanto, há um trade-off entre o tempo de análise da execução abstrata e a precisão dos valores obtidos para o WCET e WCEP [Ermedahl et al. 2011].

Dado que a busca se baseia em obter um WCEP e as entradas que o estimulam, é importante que o WCEP encontrado seja realizável e não apenas uma aproximação. Isso garante com que as entradas obtidas possam ser efetivamente usadas posteriormente na etapa de medição no hardware final, e realmente manifestem o WCEP. No entanto, nem sempre é possível atingir esse determinismo. Dependendo do estilo de programação, quantidade de combinações de entrada e o uso de entradas do tipo FLOAT (Subsubseção 3.3.2) a expansão de todos os estados abstratos se torna computacionalmente inviável, impossibilitando a conclusão da execução abstrata. Assim, optou-se por inicialmente não utilizar as técnicas de merge nos códigos benchmarks, aumentando a precisão do WCEP obtido. Caso a análise não seja concluída, então a estratégia de merge é adotada com a consciência de que o WCEP retornado pode não ser realizável semanticamente no software.

Um grande obstáculo no uso do SWEET é a falta de documentação detalhada sobre a ferramenta. Há a falta, por exemplo, de uma seção em que sejam explicados as condições e códigos de erro e o que significam. Por exemplo, em algumas análises a execução abstrata finaliza com o retorno *"Killed"* sem maiores detalhes do motivo. A seção sobre Abstract Input Annotations documenta bem os usos, mas é pobre em exemplos práticos, o que levou a dúvidas sobre como assinalar valores em estruturas compostas como Arrays. A seção sobre Debug também é deficiente e as possibilidades de monitorar o que realmente ocorre na execução abstrata limitada.

Por fim, o maior dos problemas é que há casos em que a execução abstrata não finaliza e causa travamentos na máquina. Isso ocorreu em programas que apresentam

muitos estados a serem trabalhados simultaneamente, principalmente devido à abundante quantidade de combinações de entradas e uso de variáveis de ponto flutuante, o que levou ao esgotamento dos recursos do sistema. Na execução abstrata dos exemplos **que trabalhavam com tipos FLOAT**, além do terminal do SWEET, vários outros processos abertos encerraram e a interface gráfica do Linux foi reiniciada. Notou-se no entanto, ao análisar os mesmos benchmarks em um computador com maior quantidade de recursos (principalmente memória RAM) que a execução abstrata passou a ser bem sucedida. Apenas em dois casos nos benchmarks **15-insertsort** e **28-qurt** acompanhamos o SWEET consumir os 54GB de RAM disponíveis da máquina sem alcançar a conclusão da análise.

3.3. Etapa 03 - Determinação das entradas

O último desafio antes da medição no hardware real é a obtenção das entradas que provocam o WCEP obtido na seção acima. Dado a generalidade que é proposta nesse trabalho, não foi determinado um único método que seja abrangente para qualquer tipo de fluxo de controle, pelo contrário, derivou-se um conjunto de processos que podem ser utilizados caso a caso, a medida em que se adéquem melhor ao problema em estudo.

Em geral, esses processos operam de forma iterativa, isto é realizam a *execução abstrata*, capturam as informações produzidas e as utilizam em execuções subsequentes do método. Além disso, o fato do método retornar os fatos de fluxo (flow-facts) e o próprio caminho crítico (WCEP) pode já fornecer informação suficiente para o testador determinar as entradas necessárias. Segue um resumo dos itens que auxiliam a restringir a busca pelas entradas e que serão detalhados nos sub itens a seguir:

- A execução abstrata deriva o WCEP e o WCET.
- A execução abstrata calcula os fatos de fluxo (flow-facts) que indicam o número máximo de iterações de loops e caminhos/combinações de BBs inviáveis.
- Uma inspeção visual do código, seguindo o WCEP e auxiliado pelos fatos de fluxo (flow-facts) pode fornecer indícios de quais são as entradas que realizam o caminho, ou pelo menos restringir os intervalos que as contém.
- Realizar a execução abstrata novamente com o conhecimento do WCEP, do WCET e o maior conhecimento das entradas, restringindo os intervalos é possível descobrir se a direção da escolha/indício está correta.

3.3.1. Reduzir o intervalo de valores para as entradas iterativamente

A determinação das entradas pode ser feita por um método iterativo no próprio SWEET, seguindo um esquema manual de busca binária. Como explicado anteriormente, a execução abstrata funciona sobre um intervalo de possibilidades para as entradas fornecidos via um arquivo de anotações. Após a primeira execução é retornado o BCEP, o WCEP, o BCET e o WCET. Sabendo que o WCET é o pior tempo de execução para o con-

junto de entradas escolhido, ao reduzir o intervalo de entradas pela metade no arquivo de anotação e fazer a execução abstrata novamente necessariamente o novo valor do WCET será um dos dois a seguir:

- **O novo WCET é menor que o anterior**: Caso isso ocorra significa que as entradas que provocam o caminho crítico com o maior WCET possível estavam no intervalo das entradas rejeitado, isto é, não estão mais definidas no arquivo de anotações.
- **O novo WCET é igual ao anterior**: Se isso ocorrer significa que as entradas que provocam o caminho crítico continuam no intervalo definido no arquivo de anotações.

O objetivo é que a cada redução do intervalo de entradas no arquivo de anotações e nova execução abstrata o WCET permaneça o mesmo. A técnica permite então reduzir o intervalo de busca pela metade a cada execução, isolando as entradas que provocam o WCEP. O fluxograma representativo da busca pode ser consultado nos apêndices Figura 8.

3.3.2. Problemas, Limitações e Ressalvas

Há uma ressalva sobre os tipos de dados que a execução abstrata suporta manipular quando as estratégias de *merge* estão desabilitadas. No capítulo da documentação [Gustafsson 2019] que explica sobre o *arquivo de anotações* é indicada a possibilidade do uso de FLOATs para as variáveis. Na prática, no entanto, execuções de códigos do SWEET com inputs desse tipo não atingiram a conclusão da execução do algoritmo, finalizando em estado de erro. Isso restringe o uso de entradas para o tipo INTEIRO para uso da execução abstrata em máxima precisão, isto é, obter o WCEP que é semanticamente realizável.

3.4. Etapa 04 - Medida no hardware

Com as entradas, que levam ao WCEP, definidas, o benchmark está pronto para ter o tempo de execução de seu WCEP medido no hardware, visando a analise posterior de tais medições através do método MBPTA Subseção 3.5. Para essa etapa, utilizou-se o BeagleBone Black, um kit de desenvolvimento baseado no processador AM3358X que conta com um ARM Cortex-A8. Esse ultimo, contendo uma cache de dois níveis, sendo uma memoria cache L1 de 32KB e uma memoria L2 de 256KB. Além disso, ele também conta com um pipeline de 13 estágios.

A fim de evitar o overhead proporcionado por sistemas operacionais, as medições foram conduzidas em um ambiente bare-metal. Tendo em vista tal objetivo, foi utilizado uma imagem do Uboot, um bootloader de primeiro e segundo estagio capaz de inicializar o hardware, carregar e executar o código da aplicação em um endereço desejado.

Tendo o ambiente de execução configurado, é necessário preparar o código a ser medido, isto é, na abordagem estudada, instrumentá-lo a fim de obter o número de ciclos decorridos de seu fluxo de execução. Além disso, a instrumentação também será responsável por desativar a cache L2, e realizar uma rotina de manutenção da cache L1 a cada fluxo completo.

Haja visto que a memória cache irá aumentar o desempenho da execução e, consequentemente, diminuir o número de ciclos, devido ao rápido acesso, quando comparado a uma arquitetura sem cache, é de suma importância que, para a coleta das amostras, a cache seja colocada sempre em seu pior estado, antes de cada nova execução do programa alvo, a fim de evitar medições que possam acarretar em um WCET muito otimista. Sendo assim, o pior estado da cache pode variar entre uma cache suja, para caches que possuam políticas *write-back*, e cache limpas ou sujas para caches com políticas *write-through*. No cenário estudado, uma vez que a cache L1 está configurada com a política *write-back*, foi necessário implementar uma rotina de manutenção da cache, que consiste na invalidação da cache de instruções e na ocupação da cache de dados, de tal forma que seu conteúdo não fosse útil para a próxima execução do programa avaliado.

3.4.1. Problemas, Limitações e Ressalvas

Suporte do StarterWare.

Problema inerente ao hardware que faz com que os dados não sejam randomizados...

3.5. Etapa 05 - Measurement-Based Probabilistic Timing Analysis - MBPTA

O MBPTA consiste em um método para determinação de um limite superior do pior tempo de execução. Esse método baseia-se em análises probabilísticas de medidas dos tempos de execução utilizando a Teoria do Valor Extremo (TVE), que é o ramo da estatística inicialmente projetado para estimar a probabilidade de eventos naturais incomuns/extremos.

A análise permite obter uma estimativa do WCET associado a uma baixa probabilidade de excedência aleatória, ou seja, o pWCET (*Probabilistic Worst Case Execution Time*) [Arcaro et al. 2018]. Essa estimativa é derivada de uma amostra de observações de tempo de execução de um programa realizadas em um hardware alvo, ou em um simulador de hardware, de acordo com um protocolo de medição apropriado. O protocolo de medição executa o programa múltiplas vezes de acordo com alguma(s) sequência(s) de estados de entrada viáveis e estados iniciais de hardware, amostrando um ou mais cenários possíveis de operação [Davis and Cucu-Grosjean 2019]. Portanto, neste trabalho, o cenário explorado consiste na combinação de entradas e estado do hardware que representam o pior caso de execução do programa.

Após a coleta da amostra de observações de tempo de execução, é necessário verificar se a aplicação da TVE atende a alguns requisitos. Esses requisitos tem como propósito verificar se o comportamento dos dados analisados é de fato compatível com o modelo estatístico empregado na análise. Primeiramente, o fenômeno analisado deve ser modelável por variáveis aleatórias. E por fim, a amostra de observações utilizada deve ser independente e identicamente distribuída (i.i.d.).

A verificação da aplicabilidade da TVE é feita a partir da realização de testes estatísticos apropriados. Para verificação da propriedade i.i.d. dos dados foram implementados os seguintes testes em um código *python* [Costa 2021]:

- **Teste de Anderson-Darling**: testa se os valores da amostra pertencem a uma mesma distribuição;
- **Teste Kolmogorov-Smirnov**: avalia se duas amostras são oriundas de uma mesma distribuição;
- **Teste Ljung-Box**: testa a independência nas observações a partir da análise de autocorrelação de resíduos.
- **Teste Wald-Wolfowitz runs**: testa a aleatoriedade da amostra;

Após comprovação da aplicabilidade da TVE, seleciona-se os dados a partir da amostra por meio de dois métodos empregados na literatura sobre MPBTA. Esses métodos são o de máximo de blocos (*Block Maxima - BM*), que é baseado no teorema de Fisher-Tippett-Gnedenko [Davis and Cucu-Grosjean 2019], e o de picos acima do limiar (*Peaks-over-Threshold - PoT*), que é baseado no teorema de Pickands-Balkema-de Haan [Davis and Cucu-Grosjean 2019]. Esses teoremas nos quais o TVE se baseia mostram que a distribuição assintótica da cauda de uma amostra de variáveis aleatórias, independentes e identicamente distribuídas, converge para famílias de distribuições conhecidas como Valor Extremo Generalizado (*Generalized Extreme Value - GEV*) e Distribuição de Pareto Generalizada (*Generalized Pareto Distribution - GPD*), respectivamente para o uso dos métodos BM e PoT [Costa 2021].

O método Máximo de Blocos divide a amostra em blocos de tamanho fixo e obtém o valor máximo para cada bloco. A partir dos dados selecionados, é obtida a função de densidade de probabilidade (PDF), sendo ela ajustada pela curva GEV, como ilustrado na Figura 4 abaixo.

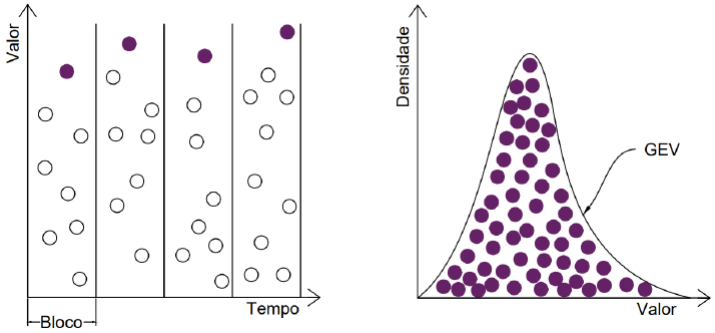

Figura 4. Exemplo de aplicação do método BM.

Fonte: [Costa 2021].

A GEV é uma família de distribuições que, de acordo com os valores dos seus parâmetros, converge para uma distribuição específica, podendo ser uma invertida de Weibull, Gumbel ou Fréchet, dependendo do parâmetro de forma ξ. A forma paramétrica da GEV é dada por [Costa 2021]:

$$G(x;\xi,\mu,\sigma) = \frac{1}{\sigma}\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\xi-1} e^{-\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\xi}} \tag{1}$$

$$G(x;\xi=0,\mu,\sigma) = \frac{1}{\sigma}\exp\left[-\exp\left(-\frac{x-\mu}{\sigma}\right)\right]\exp\left(-\frac{x-\mu}{\sigma}\right) \tag{2}$$

Os parâmetros ξ, σ e μ, apresentados nas equações acima, são conhecidos como forma, escala e local, respectivamente. De acordo com o valor de ξ, a distribuição resultante pode ser:

- $\xi = 0$: Gumbel (exponencial ou de cauda leve)
- $\xi > 0$: Fréchet (cauda pesada)
- $\xi < 0$: Weibull (cauda curta)

Já o método PoT utiliza um valor escolhido como limite para desempenhar o papel de filtro, em que apenas observações que excedam o valor limite são selecionadas. De maneira análoga, obtendo a PDF dos dados, a curva é ajustada por uma curva GPD, como ilustrado na Figura 5.

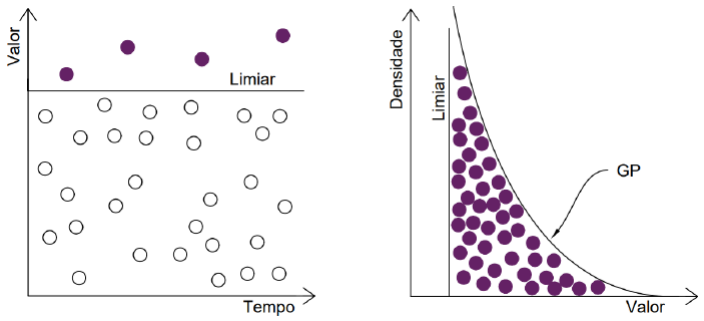

Figura 5. Exemplo de aplicação do método PoT.

Fonte: [Costa 2021].

A GPD é uma família de distribuições que incluem, Exponencial, Beta e Pareto, dependendo do parâmetro de forma ξ. A forma paramétrica da GPD é dada por [Costa 2021]:

$$H(x;\xi,\beta,\sigma) = \frac{1}{\sigma}\left(1 + \frac{\xi(x-\beta)}{\sigma}\right)^{-1/\xi-1} \tag{3}$$

$$H(x;\xi=0,\beta,\sigma) = \frac{1}{\sigma}e^{-\frac{x-\beta}{\sigma}} \tag{4}$$

Analogamente a GEV os parâmetros ξ, σ e β são conhecidos como forma, escala e local, respectivamente. De acordo com o valor de ξ, a distribuição resultante pode ser:

- $\xi = 0$: Exponencial
- $\xi > 0$: Pareto
- $\xi < 0$: Beta

Neste trabalho, optou-se por utilizar o método de Máximo de Blocos. Na literatura é possível encontrar discussões quanto à confiabilidade e acurácia desse método comparado ao método PoT. Alguns trabalhos afirmam que PoT é uma técnica mais complexa e limitado do que a BM [Machado 2021].

De acordo com [Davis and Cucu-Grosjean 2019], o método de Máximo de Blocos pode ser resumido da seguinte forma:

- Obtenha uma amostra de observações de tempo de execução usando um protocolo de medição apropriado.
- Verifique, através de testes estatísticos apropriados, se as observações coletadas do tempo de execução são analisáveis usando TVE.

- Divida a amostra em blocos de tamanho fixo e obtenha o valor máximo para cada bloco.
- Ajuste uma distribuição de valor extremo generalizado (GEV) à distribuição dos valores máximos obtidos.
- Alternativamente, caso o parâmetro de forma obtido no ajuste anterior seja $\xi \leq 0$, ajuste uma distribuição Gumbel, ou seja, parâmetro de forma fixado em zero ($\xi = 0$).
- Verifique a qualidade do ajuste entre os máximos e o GEV ajustado (por exemplo, usando gráficos de quantis).
- A partir da distribuição GEV obtida para os valores extremos, estime a distribuição pWCET.

O valor de WCET é estimado a partir da função de confiabilidade (*Reliability Function*), dada por $R(t) = 1 - CDF$, onde CDF é a função de distribuição acumulada. Logo, o WCET será dado por:

$$WCET = arg(R(t) = pWCET) \tag{5}$$

Onde *pWCET* é a probabilidade de excedência fornecida. De acordo com a literatura, são utilizados valores entre $10^{-9} \sim 10^{-15}$.

3.5.1. Problemas, Limitações e Ressalvas

Uma observação importante a ser destacada, além do cumprimento dos requisitos mínimos para aplicação da TVE, é a escolha do tamanho de bloco. Não existe ao certo um tamanho de bloco ideal para o método BM, no entanto, é reportado na literatura tamanhos de bloco entre 50 a 250 medições, a depender do tamanho total da amostra. O que deve-se atentar é que, tamanhos de bloco muito pequenos acabam por capturar ruídos indesejáveis à curva de distribuição, enquanto que, tamanhos de bloco muito grandes levam a perda de informações relevantes, além de resultar em um baixo número de dados para a composição da curva PDF. Desse modo, o ideal é analisar caso à caso qual tamanho de bloco permite a obtenção da curva PDF suficiente para o ajuste do modelo GEV.

3.6. Avaliação da Metodologia

A metodologia proposta se separa em duas frentes de estudo, a investigação da variabilidade de software e de hardware, decisão essa, tomada a fim de modularizar o processo. Ao estudar o fluxo de programa, o objetivo é encontrar o caminho crítico de execução (WCEP), isto é, aquele que resultará no maior custo de execução. Com o WCEP definido, e a variabilidade de software, consequentemente, explorada, o desafio se torna medir o tempo de execução total no hardware final, realizando a subsequente análise visando

obter o pior tempo de execução (WCET).

A abordagem de tratar os problemas de forma modularizada tem vantagens, como explorado na Seção 3, em especial a separação de responsabilidades e resolução de problemas. No entanto, essa abordagem pode levar a perda da precisão dos resultados, pois a variabilidade de software e hardware estão fortemente relacionadas.

- **Software**: O pior caminho do software pode depender de como o hardware lidará com certos conjuntos de instruções, isto é, um conjunto pode levar menos tempo para executar simplesmente porque o hardware conseguiu aplicar estratégias para aumentar a eficiência da execução. Por exemplo, em um loop que opera em um array, as características de localidade espacial aumentam a probabilidade de que as próximas posições a serem acessadas já estejam na memória cache otimizando o tempo de resposta.
- **Hardware**: Elementos como cache, pipeline, previsão de saltos e execução fora de ordem podem acelerar a execução de certos trechos de código ao aproveitar características como localidade temporal e espacial.

Logo, o hardware pode alterar o pior caminho de execução, e as estruturas do software, recursos utilizados e estilo de programação, podem afetar a forma como o hardware as executará. Nesse panorama tratar a variabilidade de software e hardware de forma separadas pode ser visto como um *trade-off*, se por um lado há o ganho em modularização, modificabilidade e simplificação em cada uma das etapas, por outro há a possibilidade de perda de informação no processo.

4. Testes e Resultados

4.1. Análise dos Benchmarks

Dos 33 benchmarks, 13 se revelaram adequados para serem submetidos ao processo completo da metodologia, ou seja, às cinco etapas, enquanto 17 foram considerados independentes das entradas e passaram a fazer parte do processo metodológico a partir da etapa 04. Veja a Seção 3 e Figura 7.

4.1.1. Execução Abstrata

Foram utilizados para a execução abstrata com o SWEET as seguintes máquinas:

- **Máquina 01:** 8 GB de RAM, processador Intel I5-6200U 2.30GHz, 1Tb de SSD;
- **Máquina 02:** 54 GB de RAM, processador Ryzen 5 5600G 3.90GHz, 2Tb de SSD, M.2.

Notou-se um maior tempo de processamento da execução abstrata e o aumento no uso da memória RAM da máquina de testes para os benchmarks que produziram mais estados e/ou fizeram o uso de variáveis do tipo *FLOAT* ou *DOUBLE*. Essa diferença do

custo da análise levou a necessidade de utilização de um ambiente de testes apropriado ao benchmark em estudo, isto é, que possua hardware suficiente para sua execução.

Assim, dos 13 benchmarks submetidos a execução abstrata, 7 foram processados pela Máquina 01, os 6 restantes por demandarem mais recursos foram explorados na máquina 02 e são comentados no detalhe a seguir.

O benchmark **33-Ud** foi explorarado para a entrada n no intervalo de 0 a 50. Após cerca de 40 minutos de análise a execução do algoritmo foi finalizada com sucesso.

Para o benchmark **15-Insertsort** a primeira tentativa foi explorar para as 11 posições do array os intervalos de 0 a 10. Isso indica que o SWEET deveria testar as 10^{12} possibilidades de entradas e lidar com os estados gerados. Após cerca de 2 horas de execução e 11219665 estados explorados a máquina 02 ocupou os 54GB de RAM disponíveis e o terminal fechou junto do processo do SWEET, impossibilitando a conclusão do algoritmo.

A experiência de executar o **15-Insertsort** sem sucesso remete ao discutido no segundo parágrafo da Subsubseção 3.2.1, quando a complexidade do programa é grande, o tratamento completo da execução abstrata se torna computacionalmente inviável. Então nesse benchmark seguimos uma estratégia de *merge*, isto é, junção de estados ao longo da execução para reduzir a complexidade da execução e uso de recursos da máquina ao custo de precisão nos resultados. O programa em questão utiliza loops while aninhados, então a estratégia escolhida foi a de realizar o *merge* em pontos de saída de loops, utilizando o parâmetro do sweet *merge=le*.

Para o benchmark **28-qurt** que possui declarações de variáveis do tipo *FLOAT* ou *DOUBLE*, assim como no exemplo anterior, a análise sem estratégias de merge não produziu resultados. No geral, após cerca de 1 hora e meia de execução na máquina 02 toda a RAM foi consumida e o processo do SWEET encerrado sem a conclusão do algoritmo. Logo, a estratégia de *merge* também foi aplicada. Nesse caso, o programa realiza algumas chamadas de função em meio aos cálculos, como funções que calculam o absoluto e a raiz quadrada de um número. A estratégia escolhida foi a de realizar o *merge* em pontos de saída de retorno de funções, utilizando o parâmetro do sweet *merge=fr*.

Para a execução dos benchmarks **30-sqrt, 20-ludcmp e 29-select** mesmo envolvendo o uso de variáveis do tipo FLOAT obtiveram resultados sem o uso de merge, o primeiro finalizou a execução após 1 hora, o segundo em cerca de 15 minutos, e o terceiro em menos de 1 segundo.

A conclusão da execução abstrata em cada um dos benchmarks foi condição para a metodologia de busca das entradas responsáveis pelo WCEP apresentada na Subseção 3.3. Os resultados para a análise dos 13 benchmarks em questão são resumidos na Tabela 1.

Programa	Máquina AE	Nº de comb. de entradas	Entradas	Estratégia de Merge	Tempo AE
bs	Máquina 01	20	a = 0	-	<1 s
cover	Máquina 01	1000	cnt = 0	-	<5 s
expint	Máquina 01	2500	n = 50, x = 1	-	<10 s
insertsort	Máquina 02	1x10^11	a[0] = 5, a[1] = 8, a[2] = [7..8], a[3] = 7, a[4] = 7, a[5] = 7, a[6] = 7, a[7] = 7, a[8] = 7, a[9] = 6, a[10] = 5	LE	<5 s
janne_complex	Máquina 01	900	a = 0, b = 5	-	<20 s
lcdnum	Máquina 01	16	a=0	-	<1 s
ludcmp	Máquina 02	10	n = 49	-	∼15 m
ns	Máquina 01	500	x = 5	-	<2 s
prime	Máquina 01	1x10^6	x = 997, y = 997	-	<5 s
qurt	Máquina 02	1000	a = 7, b=9, c=[2..3]	FR	<5 s
select	Máquina 02	20	k = 5	-	<1 s
sqrt	Máquina 02	10	valor = 1	-	∼1 h
ud	Máquina 02	50	n = 49	-	∼40 m

Tabela 1. Resultados das análises dos Benchmarks após AE.

4.1.2. Medição no Hardware

4.1.3. MBPTA

Para a análise MBPTA, a primeira etapa é a verificação da aplicabilidade dos dados coletados ao ajuste pelo modelo TVE. A avaliação dos teste estatísticos foi feita pela comparação do *pvalue* com o nível de significância α ($0.01 \leq \alpha \leq 0.05$), onde tem-se que:

- $pvalue \geq \alpha$: Hipótese-nula do teste não pode ser rejeitada;
- $pvalue < \alpha$: Hipótese-nula do teste pode ser rejeitada;

Os testes foram implementados em um código python, retornando os valores de *pvalue* para os testes executados, para a amostra de dados fornecida ao programa. Neste trabalho, foi considerado nível de significância $\alpha = 0.05$. Os resultados dos testes aplicados aos dados coletados dos benchmarks estão apresentados na **??**.

Benchmarks	KS	AD	WW	LB
bs	0,187	0,059	0,373	0,950
bsort100	0,001	0,001	0,022	0,980
cnt	0,464	0,250	0,063	0,980
crc	0,105	0,094	0,000	0,558
duff	0,869	0,250	0,000	0,132
fdct	0,622	0,250	0,061	0,998
jfdctint	0,886	0,250	0,109	0,984
matmult	0,392	0,250	0,267	0,398
fir	0,000	0,000	0,000	0,988
select	0,585	0,250	0,063	1,000
ud	0,001	0,001	0,013	0,964
janne_complex	0,720	0,250	0,000	0,997
prime	0,000	0,001	0,000	0,980
statemate	0,000	0,001	0,087	0,999
nsichneu	0,017	0,001	0,000	0,003

Tabela 2. Resultados dos testes estatísticos para verificação de aplucabilidade da TVE.

De acordo com os resultados acima, os testes foram atendidos apenas para os benchmarks LISTAR AQUI OS BENCHMARKS.

Em seguida, a análise da TVE foi empregada nas medições coletadas. Vale ressaltar que para os arquivos de dados que não atenderam aos requisitos verificados pelos testes estatísticos, não é possível garantir confiabilidade para os resultados de WCET gerados, ao passo que, para os que atenderam, é possível garantir tal aspecto.

Referências

Altenbernd, P. (1996). On the false path problem in hard real-time programs. In *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, pages 102–107.

Arcaro, L. F., Silva, K., and de Oliveira, R. (2018). A reliability evaluation method for probabilistic wcet estimates based on the comparison of empirical exceedance densities. In *Anais do VIII Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, pages 129–133, Porto Alegre, RS, Brasil. SBC.

Costa, J. J. S. (2021). Emprego de medição na estimação do tempo de execução no pior caso para sistemas de tempo real -orientador, rômulo silva de oliveira, coorientador, luís fernando arcaro. https://repositorio.ufsc.br/handle/123456789/229274. Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2021.

Davis, R. I. and Cucu-Grosjean, L. (2019). A survey of probabilistic timing analysis techniques for real-time systems. *Leibniz Transactions on Embedded Systems*, 6(1):03:1–03:60.

de Rezende, S. F. (2014). Caminhos mais longos em grafos. Master's thesis, USP, https://repositorio.ufsc.br/xmlui/handle/123456789/133239. Karila Palma Silva ; orientador, Rômulo Silva de Oliveira - Florianópolis, SC, 2015. Dissertação (mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

Ermedahl, A., Gustafsson, J., and Lisper, B. (2011). Deriving wcet bounds by abstract execution. In Healy, C., editor, *Proc. 11th International Workshop on Worst-Case Execution Time (WCET) Analysis (WCET 2011)*. Austrian Computer Society (OCG).

Felipe Brito, Gustavo Correia, I. F. e. L. C. (2023). Pes-tcc-estratégia-híbrida-modular. https://github.com/sw3luke/PES_TCC_WCET-Estrategia_Hibrida_Modular/tree/master.

Gustafsson, J. (2019). Sweet manual. In *SWEET manual*. Västerås Sweden.

Gustafsson, J., Ermedahl, A., Lisper, B., Sandberg, C., and Källberg, L. (2009). ALF - A Language for WCET Flow Analysis. In Holsti, N., editor, *9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*, volume 10 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–11, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. also published in print by Austrian Computer Society (OCG) with ISBN 978-3-85403-252-6.

Lisper, B. (2014). Sweet – a tool for wcet flow analysis. In Steffen, B., editor, *6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 482–485. Springer-Verlag.

Machado, G. I. D. (2021). Analysis of the extreme value theory on the estimation of probabilistic wcet. Master's thesis, PUC-RS, https://tede2.pucrs.br/tede2/handle/tede/10022.

Malich, M. (2017). Avr-tick-counter. https://github.com/malcom/AVR-Tick-Counter.

Martins, G. N. (2018). Validando modelos para verificação de programas p4 por execução simbólica. Master's thesis, UFRGS, https://lume.ufrgs.br/bitstream/handle/10183/190199/001088714.pdf?sequence=1.

Silva, K. P. (2015). Análise de valor para determinação do tempo de execução no pior caso (wcet) de tarefas em sistemas de tempo real. Master's thesis, UFSC, https://repositorio.ufsc.br/xmlui/handle/123456789/133239. Karila Palma Silva ; orientador, Rômulo Silva de Oliveira - Florianópolis, SC, 2015. Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., and Stenstrom, P. (2008). The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.*, 7.

Yen, S., Du, D., and Ghanta, S. (1989). Efficient algorithms for extracting the k most critical paths in timing analysis. In *26th ACM/IEEE Design Automation Conference*, pages 649–654.

Apêndices

A. Fluxogramas

A.1. Fluxograma da metodologia

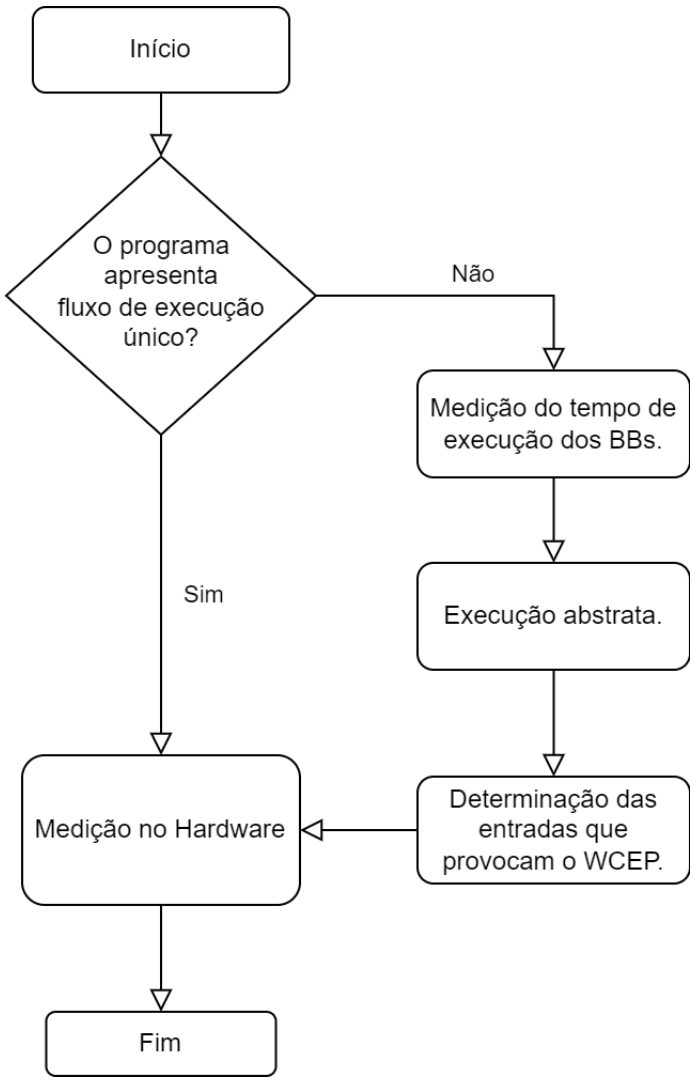

Figura 6. Fluxograma da metodologia.

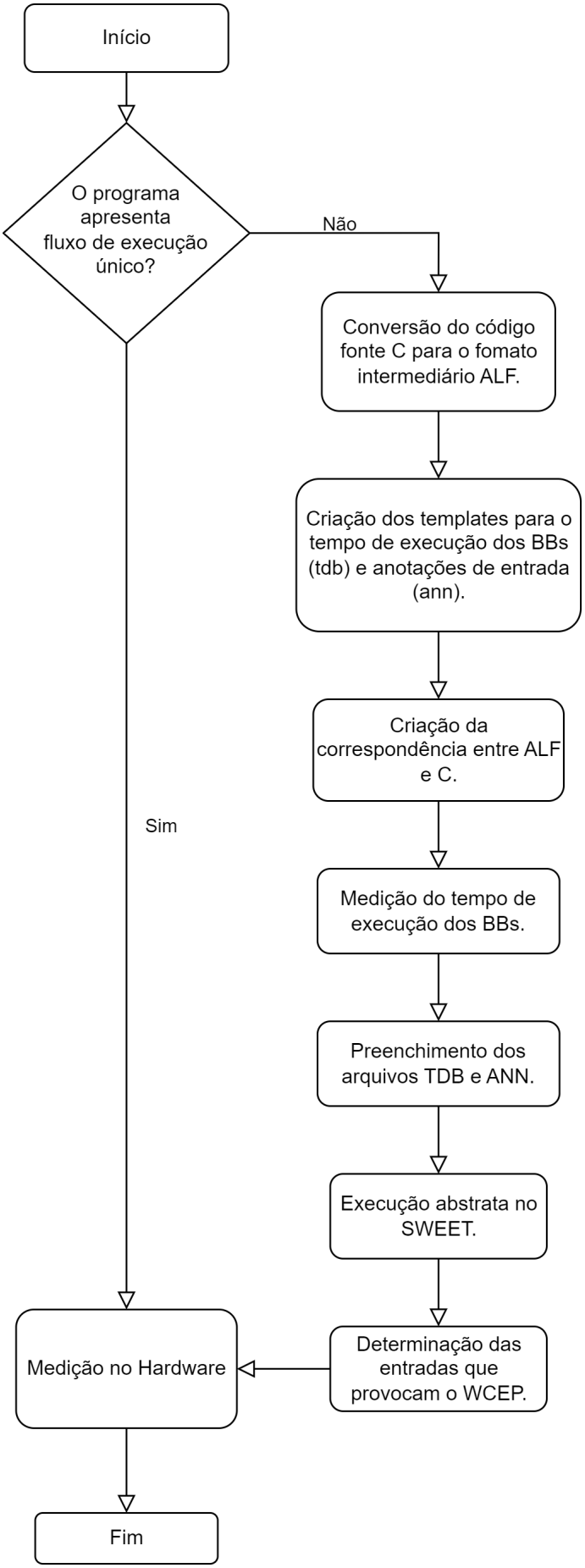

Figura 7. Fluxograma detalhado da metodologia.

A.2. Fluxograma para busca de entradas na Etapa 03

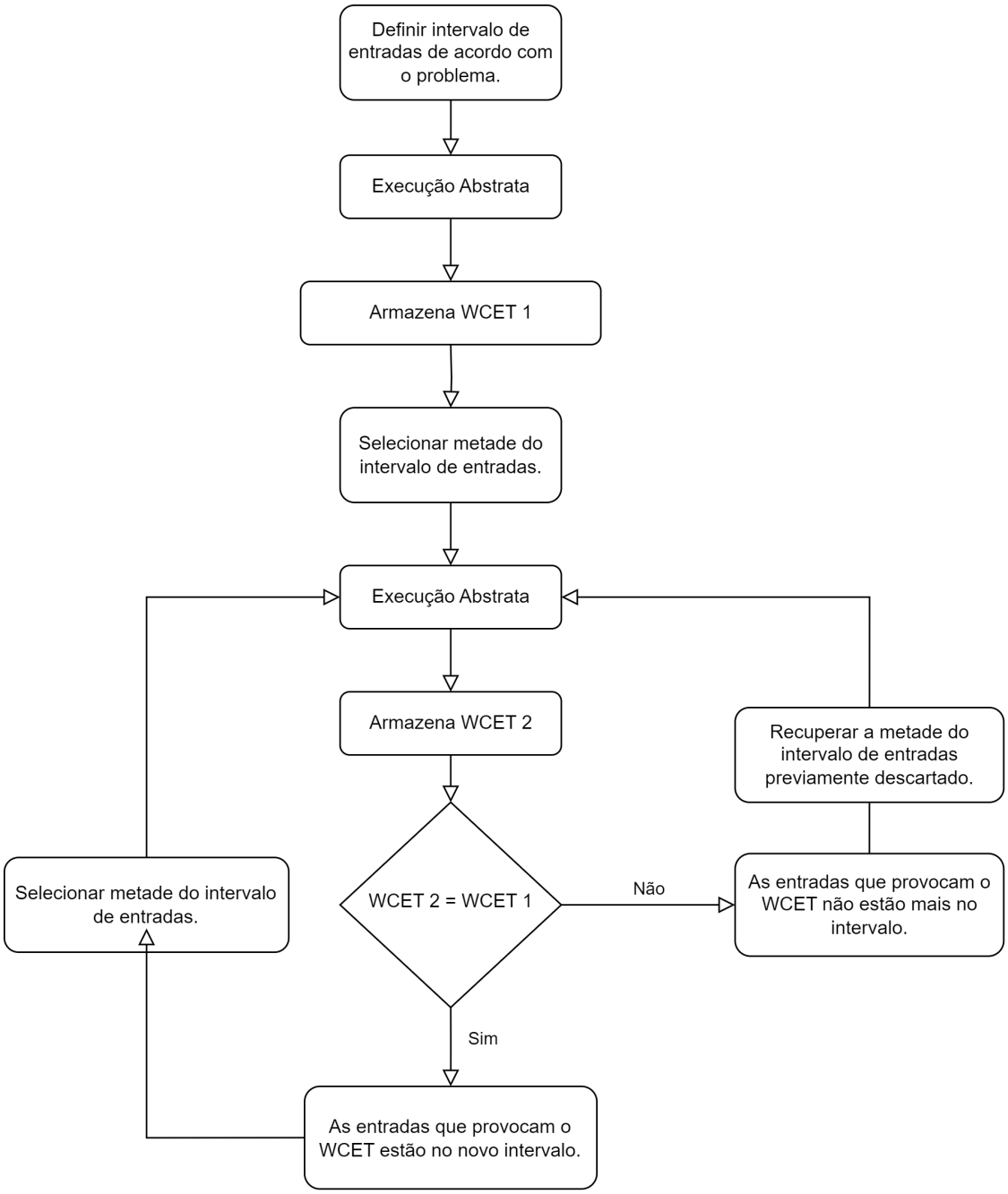

Figura 8. Fluxograma da técnica de busca binária para isolar as entradas que provocam o WCEP.

B. Requisitos e arquitetura de ferramentas

Os requisitos de alto nível que guiaram o desenvolvimento do processo e framework foram:

- HLR01: O framework deve identificar os blocos básicos relacionados ao código fonte em C fornecido como entrada.
- HLR02: Caso o programa analisado apresente múltiplos caminhos cuja execução depende das entradas, os blocos básicos devem ser instrumentados para medição de seus respectivos tempos de execução.
- HLR03: O framework deve identificar o WCEP baseado na técnica de Execução Abstrata, caso o programa apresente múltiplos caminhos.
- HLR04: A técnica de Execução Abstrata deve ter como entrada os seguintes arquivos: Código a ser analisado, tempos de execução dos blocos básicos e arquivo de anotações com os ranges possíveis das entradas.
- HLR05: O framework deve determinar as entradas que provocam o WCEP.
- HLR06: O fluxo de execução correspondente ao WCEP deve ser provocado pelas entradas responsáveis e medido em Hardware.
- HLR07: O framework deve realizar a análise probabilística utilizando a Teoria do Valor Extremo caso o conjunto de medidas do WCEP atinjam os requisitos mínimos do método.
- HLR08: O framework deve permitir a realização de testes estatísticos para verificação da aplicabilidade da TVE, seguindo requisitos mínimos de dados I.I.D ou estacionários;
- HLR09: A análise MBPTA deve empregar o método de seleção de dados Máximo de Blocos (BM).
- HLR10: A análise MBPTA deve receber como entradas o arquivo de dados de tempo de execução, tamanho de bloco referente ao método BM, e valor de probabilidade de excedência (pWCET), retornando como saída o valor de WCET associado à pWCET.
- HLR11: O framework deve ser desenvolvido a partir da composição de componentes modulares.

A arquitetura geral do processo/framework que reflete diretamente os requisitos é apresentada na Figura 9.

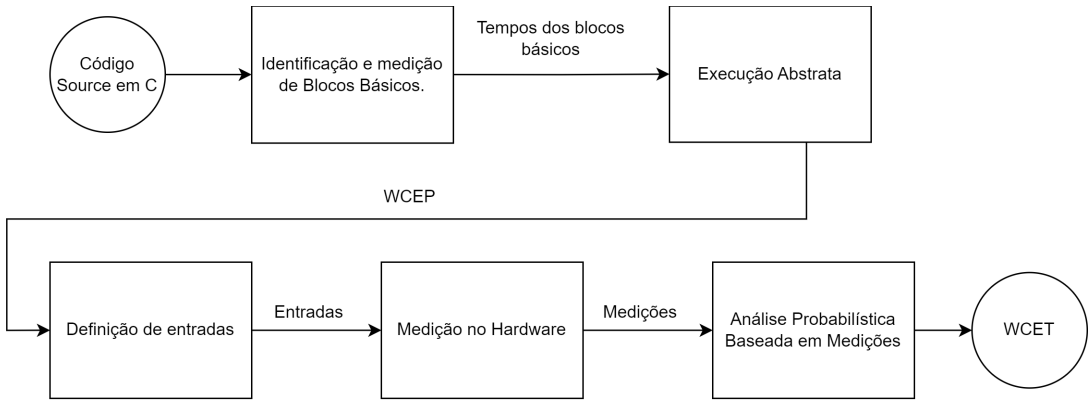

Figura 9. Arquitetura do Framework.

Na etapa de **Identificação e medição de Blocos Básicos** são utilizados os programas ALF-Backend, Python e SimulAVR. O ALF-Backend atua na conversão do código em C para o formato intermediário ALF utilizado posteriormente na etapa de execução abstrata. O Python é utilizado para a execução de um script que entrega a equivalência entre os dois formatos C e ALF. Por fim, SimulAVR é o simulador responsável por medir os BBs no código-fonte C.

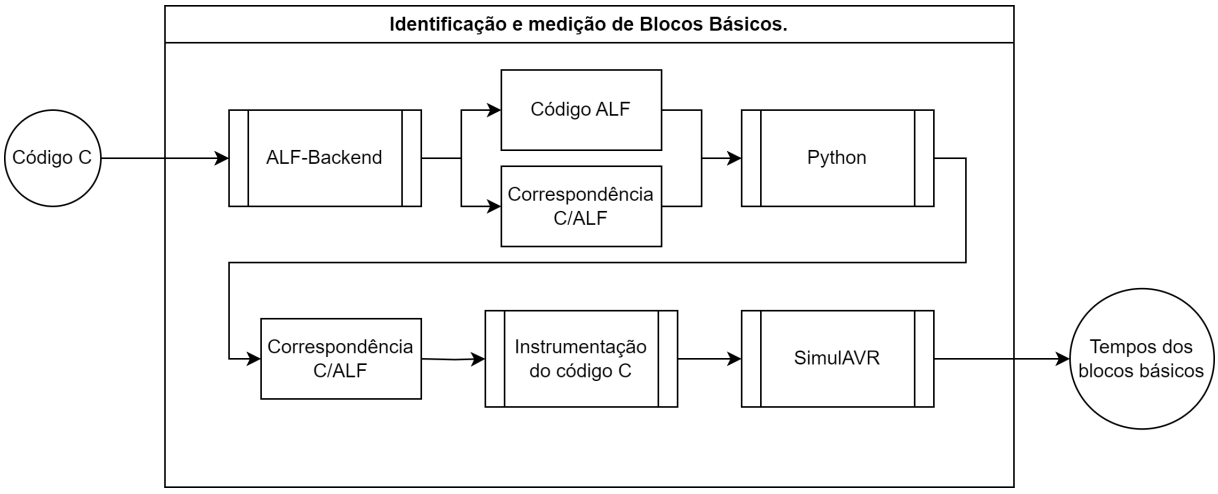

Figura 10. Etapa de Identificação e medição dos Blocos Básicos.

Na etapa da **Execução Abstrata** é utilizado o programa SWEET que realiza a execução abstrata mediante com o código no formato intermediário e os arquivos TDB e ANN. O foco dessa etapa é determinar as entradas que provocam o WCEP, assim a execução abstrata é feita de forma iterativa restringindo a cada repetição a faixa de entradas possíveis para o WCEP.

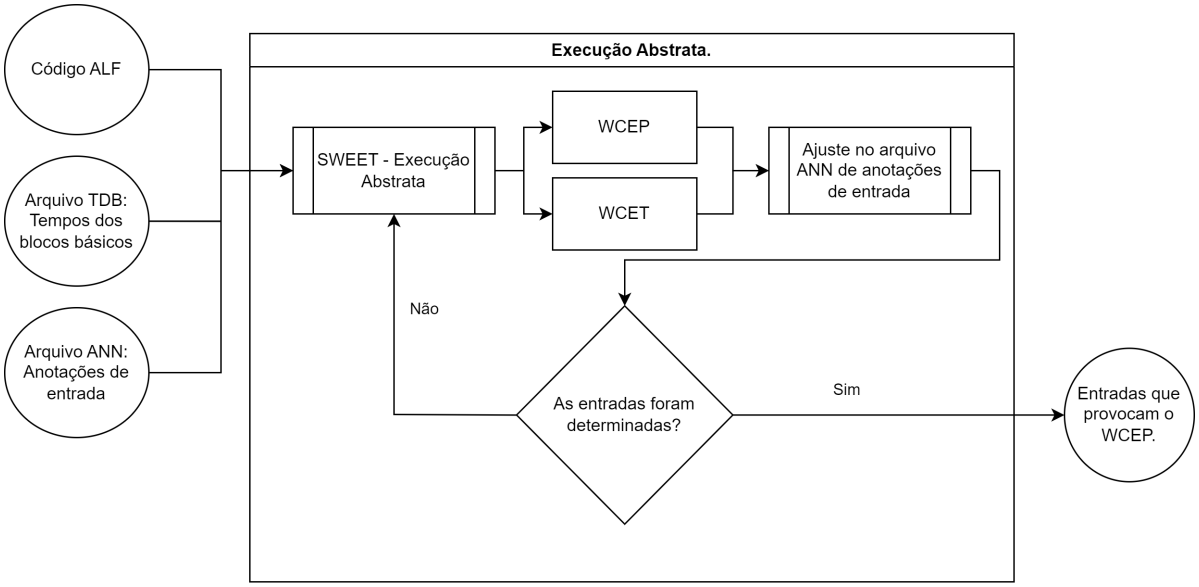

Figura 11. Etapa da Execução Abstrata.

Na etapa da **Medidas no Hardware** é utilizado o código fonte em C e as entradas que provocam o caminho crítico no software. O programa é executado sucessivamente de início a fim e mensurado. Entre cada execução há o cuidado de invalidar as caches de instruções e dados com o objetivo de colocar o ambiente no pior estado possível para a execução. O resultado do processo de medidas são 2 blocos de 50 mil medições cada.

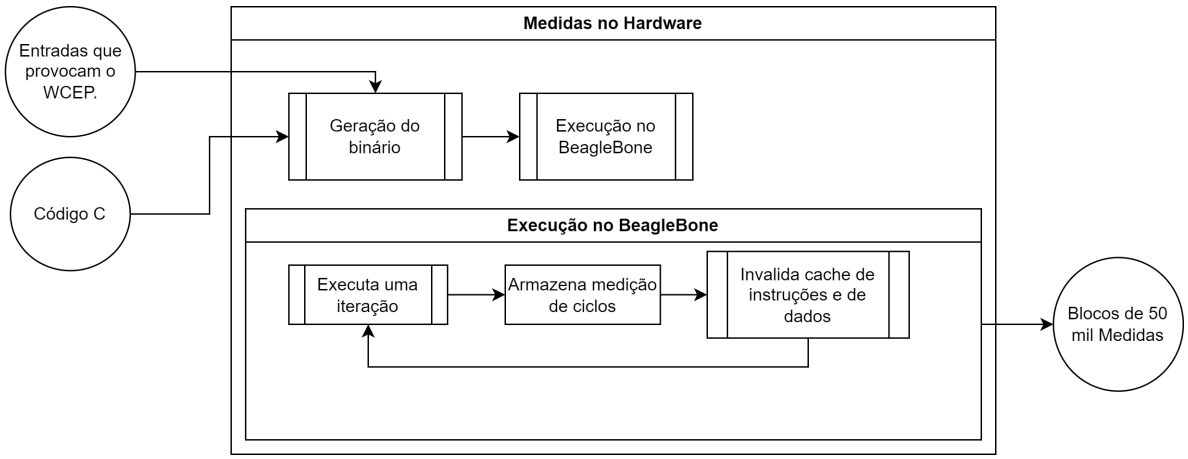

Figura 12. Etapa de Medição no Hardware.

Na etapa da **Análise MBPTA**, utilizando o conjunto de dados coletado, são realizados os testes estatísticos de I.I.D. e estacionariedade dos dados para posterior aplicação da modelagem TVE. Analisando os resultados dos testes, uma vez que os requisitos de aplicação são atendidos, a modelagem é feita, fornecendo a amostra de observações, valor

de tamanho de bloco, e valor de probabilidade de excedência, retornando, por fim, o valor de WCET.

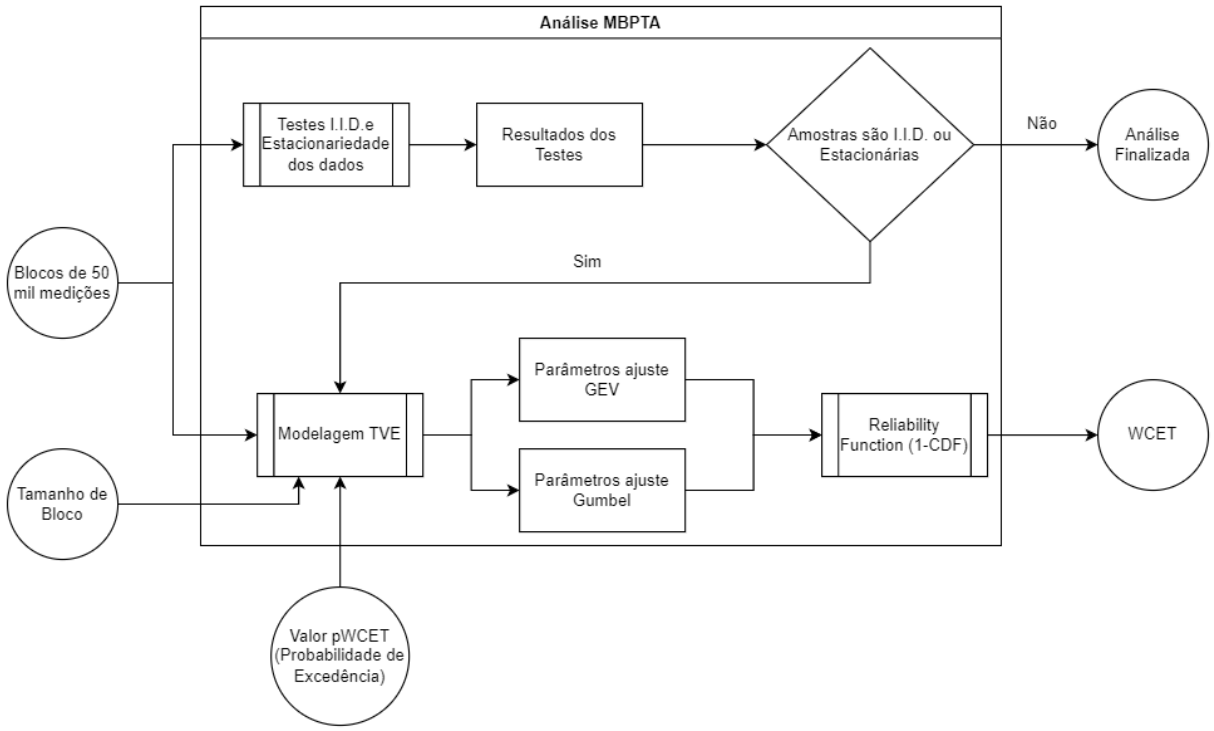

Figura 13. Etapa da Análise MBPTA.

C. Instalações

O passo a passo das instalações e procedimentos que fizemos nas ferramentas, além dos principais comandos estão disponíveis em [Felipe Brito 2023]. As instruções de instalação para o ALF-Backend e o SWEET se aplicam a distribuição Ubuntu 22.04.3 do Linux.