PGCOMP - Programa de Pós-Graduação em Ciência da Computação
Universidade Federal da Bahia (UFBA)
Av. Milton Santos, s/n - Ondina
Salvador, BA, Brasil, 40170-110

https://pgcomp.ufba.br
pgcomp@ufba.br

Sistemas em tempo real (*Real-Time Systems* - RTS) são compostos por um conjunto de tarefas (trechos de código) sendo lançadas recorrentemente para serem executadas e devem cumprir prazos. Projetar um sistema desse tipo de forma comprovadamente correta requer informações sobre o tempo de execução no pior caso (*Worst-Case Execution Time* - WCET) para cada uma de suas tarefas. No entanto, estimar o WCET está se tornando cada vez mais difícil devido à alta complexidade de hardware e software presentes nas plataformas modernas atuais. Isso tem motivado o uso de técnicas para derivar o tempo de execução probabilístico no pior caso (*Probabilistic Worst-Case Execution Time* - pWCET). A maioria das abordagens existentes se baseia em medir o tempo de execução das tarefas do sistema na plataforma alvo. Como as medições são realizadas durante o tempo de projeto, as amostras coletadas podem levar a estimativas não confiáveis (devido a um possível viés de medição) ou não representativas (devido às dificuldades na reprodução das condições operacionais). A necessidade de tornar as amostras compatíveis com as suposições da modelagem estatística é uma fonte adicional de dificuldade. Dadas as complexidades apresentadas, foram desenvolvidos dois estudos com objetivos distintos. No primeiro estudo, uma representação do tempo de execução é realizada com base em eventos de hardware, considerando diferentes ferramentas de inteligência computacional. Mais especificamente, para um programa sob análise, é mostrado que o tempo de execução $T(n)$ por número n de instruções executadas pode ser correlacionado com ocorrências de eventos relacionados ao hardware. No segundo estudo, uma nova abordagem para análise temporal probabilística baseada em medição (*Measurement-Based Probabilistic Timing Analysis* - MBPTA) é apresentada. Ao contrário do MBPTA usual, que considera apenas $T(n)$ como a variável de interesse, essa nova abordagem incorpora uma variável de interesse que considera tanto $T(n)$ quanto n. Utilizar tuplas $(n, T(n))$ para diferentes valores de n possibilita explorar múltiplos caminhos de execução. Além disso, essa nova abordagem permite que o conjunto de medições seja avaliado e aprimorado. Para esse propósito, redes neurais profundas (*Deep Neural Network* - DNN) foram empregadas. Uma vez que as medições são consideradas representativas, é possível estimar limites probabilísticos do tempo de execução. Ambas as abordagens foram avaliadas considerando diferentes modelos de arquiteturas e programas, e os resultados obtidos demonstraram eficácia nos estudos propostos.

Palavras-chave: tempo de execução de pior caso; inteligência computacional; análise estatística; arquitetura multinúcleo

DSC | 55 | 2025

Métodos estatísticos e de inteligência computacional para análise temporal em sistemas de tempo real

Tadeu Nogueira Costa de Andrade

Métodos estatísticos e de inteligência computacional para análise temporal em sistemas de tempo real

Tadeu Nogueira Costa de Andrade

Tese de Doutorado

Universidade Federal da Bahia

Programa de Pós-Graduação em
Ciência da Computação

Março | 2025

UFBA

Universidade Federal da Bahia
Instituto de Computação

Programa de Pós-Graduação em Ciência da Computação

MÉTODOS ESTATÍSTICOS E DE INTELIGÊNCIA COMPUTACIONAL PARA ANÁLISE TEMPORAL EM SISTEMAS DE TEMPO REAL

Tadeu Nogueira Costa de Andrade

TESE DE DOUTORADO

Salvador
27 de Março de 2025

TADEU NOGUEIRA COSTA DE ANDRADE

MÉTODOS ESTATÍSTICOS E DE INTELIGÊNCIA COMPUTACIONAL PARA ANÁLISE TEMPORAL EM SISTEMAS DE TEMPO REAL

Esta Tese de Doutorado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: George Marconi de Araújo Lima
Co-orientadora: Verônica Maria Cadena Lima

Salvador
27 de Março de 2025

Ficha catalográfica elaborada pela Biblioteca Universitária de Ciências e Tecnologias
Prof. Omar Catunda, SIBI – UFBA.

A553 Andrade, Tadeu Nogueira Costa de
Métodos estatísticos e de inteligência computacional para análise temporal em sistemas de tempo real – Tadeu Nogueira Costa de Andrade. – Salvador, 2025.
118p.: il.

Orientador: Prof. Dr. George Marconi de Araújo Lima.
Co-orientadora: Prof. Dr. Verônica Maria Cadena Lima.

Tese (Doutorado) – Universidade Federal da Bahia.
Instituto de Computação, 2025.

1. Tempo de execução de pior caso. 2. Inteligência computacional. 3. Análise estatística. I. Lima, George Marconi de Araújo. II. Lima, Verônica Maria Cadena. III. Universidade Federal da Bahia. Instituto de Computação. IV. Título.

CDU: 004.8

TERMO DE APROVAÇÃO

TADEU NOGUEIRA COSTA DE ANDRADE

MÉTODOS ESTATÍSTICOS E DE INTELIGÊNCIA COMPUTACIONAL PARA ANÁLISE TEMPORAL EM SISTEMAS DE TEMPO REAL

Esta Tese de Doutorado foi julgada adequada à obtenção do título de Doutor em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia.

Salvador, 27 de Março de 2025

Prof. Dr. George Marconi de Araújo Lima
Orientador - PGCOMP/UFBA

Prof. Dr. Konstantinos Bletsas
CISTER Research Centre - Portugal

Documento assinado digitalmente
GIOVANI GRACIOLI
Data: 01/04/2025 08:43:25-0300
CPF: ***.838.060-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Giovani Gracioli
Universidade Federal de Santa Catarina - UFSC

Documento assinado digitalmente
gov.br
ALLAN EDGARD SILVA FREITAS
Data: 02/04/2025 16:00:11-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Allan Edgard Silva Freitas
Instituto Federal da Bahia - IFBA

Prof. Dr. Maycon Leone Maciel Peixoto
PGCOMP/ UFBA

Dedico esta tese aos meus filhos e à minha esposa, pelo apoio, paciência e compreensão durante todo o doutorado. Agradeço também aos meus pais e irmãos por acreditarem em mim, e aos meus orientadores, cujas contribuições foram fundamentais para o desenvolvimento deste trabalho.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela saúde, persistência e força de vontade que me concedeu para concluir esta importante etapa da minha vida. Aos meus pais, Ivan Matos e Taciara Nogueira, e aos meus irmãos, Ivan Filho e Ítalo Nogueira, por me acompanharem em cada momento. À minha esposa, Verena Lopes, e aos meus filhos, Alice Nogueira e Vinícius Nogueira, que me apoiaram ao longo dessa trajetória, com paciência, compreensão, apoio nas dificuldades e alegria nas vitórias.

Agradeço ao PGCOMP pela oportunidade oferecida, e aos professores pela competência, paciência, qualidade e dedicação nas disciplinas lecionadas. Ao Instituto Federal Baiano, por me proporcionar todo o apoio e a oportunidade de me afastar das minhas atividades em um momento importante dos meus estudos.

Agradeço de forma especial aos meus orientadores, Dr. George Lima e Dra. Verônica Lima, pelo profissionalismo, dedicação, paciência e compreensão, além da capacidade de me inspirar a buscar sempre os melhores resultados, sem me deixar desanimar quando esses resultados não surgiam. Tenho certeza de que, sem o apoio de vocês, nada disso seria possível. Espero levar todos esses ensinamentos adiante na minha carreira.

"All battles are first won or lost, in the mind."

—JOANA D'ARC (1412-1431)

RESUMO

Sistemas em tempo real (RTS) são compostos por um conjunto de tarefas (trechos de código) sendo lançadas recorrentemente para serem executadas e devem cumprir prazos. Projetar um sistema desse tipo de forma comprovadamente correta requer informações sobre o tempo de execução no pior caso (WCET) para cada uma de suas tarefas. No entanto, estimar o WCET está se tornando cada vez mais difícil devido à alta complexidade de hardware e software presentes nas plataformas modernas atuais. Isso tem motivado o uso de técnicas para derivar o tempo de execução probabilístico no pior caso (pWCET). A maioria das abordagens existentes se baseia em medir o tempo de execução das tarefas do sistema na plataforma alvo. Como as medições são realizadas durante o tempo de projeto, as amostras coletadas podem levar a estimativas não confiáveis (devido a um possível viés de medição) ou não representativas (devido às dificuldades na reprodução das condições operacionais). A necessidade de tornar as amostras compatíveis com as suposições da modelagem estatística é uma fonte adicional de dificuldade. Dadas as complexidades apresentadas, foram desenvolvidos dois estudos com objetivos distintos. No primeiro estudo, uma representação do tempo de execução é realizada com base em eventos de hardware, considerando diferentes ferramentas de inteligência computacional. Mais especificamente, para um programa sob análise, é mostrado que o tempo de execução $T(n)$ por número n de instruções executadas pode ser correlacionado com ocorrências de eventos relacionados ao hardware. No segundo estudo, uma nova abordagem para análise temporal probabilística baseada em medição (MBPTA) é apresentada. Ao contrário de MBPTA usual, que considera apenas $T(n)$ como a variável de interesse, essa nova abordagem incorpora uma variável de interesse que considera tanto $T(n)$ quanto n. Utilizar tuplas $(n, T(n))$ para diferentes valores de n possibilita explorar múltiplos caminhos de execução. Além disso, essa nova abordagem permite que o conjunto de medições seja avaliado e aprimorado. Para esse propósito, redes neurais profundas (DNN) são empregadas. Uma vez que as medições são consideradas representativas, é possível estimar limites probabilísticos do tempo de execução. Resultados experimentais indicam uma diferença de até 30% entre as estimativas obtidas com amostras aprimoradas pela abordagem proposta e as amostras não aprimoradas. As abordagens são avaliadas considerando diferentes modelos de arquiteturas e programas, e os resultados obtidos demonstram eficácia nos estudos propostos.

Palavras-chave: tempo de execução de pior caso; inteligência computacional; análise estatística; arquitetura multinúcleo

ABSTRACT

Real-time systems (RTS) are composed of a set of tasks (code segments) that are recurrently launched to be executed and must meet deadlines. Designing such a system in a provably correct manner requires information about the worst-case execution time (WCET) for each of its tasks. However, estimating the WCET is becoming increasingly challenging due to the high complexity of hardware and software in modern platforms. This has motivated the use of techniques to derive the probabilistic worst-case execution time (pWCET). Most existing approaches rely on measuring the execution time of system tasks on the target platform. As measurements are taken during the design time, collected samples may lead to unreliable estimates (due to possible measurement bias) or non-representative ones (due to difficulties in reproducing operational conditions). The need to make samples compatible with the assumptions of statistical modeling is an additional source of difficulty. Given the complexities presented, two studies with distinct objectives were developed. In the first study, a representation of execution time is performed based on hardware events, considering different computational intelligence tools. Specifically, for a program under analysis, it is shown that the execution time $T(n)$ per number n of executed instructions can be correlated with occurrences of hardware-related events. In the second study, a new approach for Measurement-Based Probabilistic Timing Analysis (MBPTA) is presented. Unlike the usual MBPTA, which considers only $T(n)$ as the variable of interest, this new approach incorporates a variable of interest that considers both $T(n)$ and n. Using tuples $(n, T(n))$ for different values of n allows for exploring multiple execution paths. Additionally, this new approach allows the set of measurements to be evaluated and improved. For this purpose, deep neural networks (DNN) were employed. Since the measurements are considered representative, it is possible to estimate probabilistic bounds on the execution time. Experimental results indicate a difference of up to 30% between the estimates obtained using samples refined by the proposed approach and those obtained using non-refined samples. The approaches are evaluated considering different hardware and program models, and the results obtained demonstrate effectiveness in the proposed studies.

Keywords: worst case execution time; computational intelligence; statistical analysis; multicore architecture

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BM	*Block Maxima*	23
CCDF	*Complementary Cumulative Distribution Function*	14
CIG	*Cache-related Interference Generator*	42
DBL-MBPTA	*Distance Between Lines - MBPTA*	59
DL	*Deep Learning*	21
DNN	*Deep Neural Network*	46
DPU	*Data Processing Unit*	85
DWT	*Data Watchpoint and Trace Unit*	114
EPC	*Extended Path Coverage*	27
EVT	*Extreme Values Theory*	3
EV	*Extreme Values*	24
GEV	*Generalized Extreme Value*	31
GP	*Generalized Paret*	17
ILP	*Integer Linear Programming*	11
i.i.d	independente e identicamente distribuida	23
LP	*Linear Programming*	69
MBTA	*Measurement-Based Timing Analysis*	11
MBPTA	*Measurement-Based Probabilistic Timing Analysis*	3
ML	*Machine Learning*	18
MLR	*Multiple Linear Regression*	46
MMQ	Método dos Mínimos Quadrados	19
MNN	*Monotonic Neural Network*	46
MSE	*Mean Squared Error*	19
PCA	*Principal Component Analysis*	30
PMF	*Probability Mass Functions*	14
PMU	*Performance Monitoring Unit*	4
PoT	*Peaks-over Threshold*	23
PTA	*Probabilistic Timing Analysis*	14

Capítulo 1

INTRODUÇÃO

1.1 CONTEXTO GERAL

Sistemas de tempo real (*Real-Time Systems* (RTS)) são sistemas computacionais que precisam executar cada uma de suas tarefas no tempo máximo pré-estabelecido. Uma tarefa é uma unidade de trabalho representada por um trecho de código, onde a sua execução caracteriza o momento em que é iniciada até ser processada e concluída. Em RTS, uma tarefa é considerada corretamente executada somente quando seus requisitos funcionais e temporais são devidamente atendidos. O tempo de execução da tarefa é o tempo que a tarefa leva de processador para ser concluída. Esse tempo pode variar de acordo com o estado do hardware e os caminhos de execução. Os caminhos de execução são as possíveis sequências de instruções que a tarefa pode seguir durante sua execução. O tempo limite para finalizar a tarefa é denominado *deadline* (OLIVEIRA, 2018).

A classificação do RTS está relacionada ao fator pós-*deadline*, que estabelece as consequências decorrentes de ultrapassar o tempo máximo estabelecido e definido previamente. Um RTS pode ser classificado como crítico (*hard real-time*) ou não-crítico (*soft real-time*). Em sistemas críticos, como o trem de pouso em aviões ou os *airbags* de automóveis, a não conformidade com os requisitos temporais pode ter consequência trágica. Em um acidente, os *airbags* precisam ser acionados em um limite de tempo determinado após um impacto. Uma resposta fora desse limite pode resultar em ferimentos graves ao motorista. Em sistemas não-críticos, ultrapassar o tempo máximo estabelecido resulta apenas em uma falha de transmissão, sem maiores gravidades. Em uma videochamada, por exemplo, atrasos no carregamento de imagens não resulta em situações catastróficas, a não ser em uma insatisfação por parte do usuário.

Para atender os requisitos temporais de um projeto tradicional de RTS, é necessário ter conhecimento do tempo de execução no pior caso (*Worst-Case Execution Time* (WCET)) de cada tarefa presente no sistema. Isso envolve estabelecer um limite superior para o tempo de execução, o qual deve ser respeitado em qualquer cenário. Um sistema é considerado determinístico quando seu comportamento temporal é conhecido e previsível. Ou seja, o tempo de resposta para um conjunto de saídas é conhecido (LAPLANTE, 2004).

Nesse caso, é possível determinar com segurança o WCET de um sistema determinístico por meio de medições do tempo de execução no pior cenário.

Embora o WCET possa variar dependendo da arquitetura de hardware e software utilizada, o ideal é que seja obtido independentemente da plataforma. Segundo Davis e Cucu-Grosjean (2019), os métodos tradicionais utilizados para realizar estimativas WCET podem ser divididos em três categorias:

- Análise estática - O código do programa é analisado em conjunto com um modelo abstrato do hardware para determinar os possíveis caminhos de execução e o comportamento do hardware em cada um desses caminhos. Utilizando programação linear inteira (*Integer Linear Programming* (ILP)), os resultados da análise de fluxo de controle e análise microarquitetural são combinados para obter o WCET;

- Análise baseada em medição - Um protocolo de medição é usado para coletar os tempos de execução do programa, registrando o tempo de execução máximo observado. Esse valor pode ser utilizado como uma estimativa para o WCET, ou uma margem de segurança pode ser adicionada para obter uma estimativa mais conservadora;

- Análise híbrida - Uma combinação de elementos da análise estática e da análise por medição é realizada, como, por exemplo, na utilização de tempo máximo observado em pequenos trechos de código.

Em RTS que utiliza arquiteturas modernas, a determinação do WCET é um processo desafiador que dificulta a aplicação dos métodos tradicionais mencionados anteriormente. Isso se deve à complexidade do hardware, à presença de recursos compartilhados e à natureza dinâmica do ambiente de execução. Componentes como memórias cache, *pipelines* de instruções, *branch predictors* e processadores multinúcleo são comumente encontrados nesse modelo de arquitetura. Esses componentes desempenham um papel importante no aumento do desempenho geral do sistema, no entanto, também apresentam desafios significativos no projeto e análise de RTS. Por exemplo, a presença de uma memória cache de último nível compartilhada entre os núcleos introduz uma situação de concorrência e interferência. Quando um dos núcleos insere um conteúdo na cache, outro núcleo pode acessar esta cache e substituir este conteúdo, adicionando informações de outra tarefa. Isso significa que uma tarefa pode ter seus dados transferidos entre diferentes níveis de cache durante sua execução, o que, consequentemente, pode afetar o tempo de execução das operações. Como resultado, é possível que cada execução da tarefa apresente um tempo de execução diferente. Sistemas com essas características são classificados como não determinísticos (LAPLANTE, 2004).

1.2 DESCRIÇÃO DO PROBLEMA

A dificuldade de obter estimativas precisas de WCET usando arquiteturas modernas leva a um aumento no interesse pelo uso de abordagens estatísticas e probabilísticas. Essas abordagens têm a vantagem de lidar com observações empíricas (caixa-preta), sendo

muito mais fáceis de se obter do que um conhecimento detalhado (caixa-branca) dos aspectos internos de uma execução (CAZORLA et al., 2019). A análise temporal baseada em medição (*Measurement-Based Timing Analysis* (MBTA)) é utilizada para obter limites superiores nos tempos de execução das tarefas por meio da coleta de dados durante as medições (DAVIS; CUCU-GROSJEAN, 2019; CAZORLA et al., 2019). A análise temporal probabilística baseada em medição (*Measurement-Based Probabilistic Timing Analysis* (MBPTA)) é uma derivação de MBTA que se destaca ao fornecer estimativas probabilísticas de WCET. Ao utilizar MBPTA para estimar WCET, o tempo de pior caso é referido como pWCET (*Probabilistic Worst-Case Execution Time* (pWCET)). O pWCET estimado para uma tarefa é representado por uma tupla (x, p), em que x é a estimativa do tempo máximo de execução e p é a probabilidade associada. A interpretação correta da tupla (x, p) deve ser a seguinte: a probabilidade de observar um valor maior que x não é superior a p.

Algumas estratégias, inspiradas na desigualdade de Chebyshev, foram discutidas no contexto de MBPTA para realizar estimativas pWCET (V.G. Baranoski; G. Rokne; XU, 2001; RANJBAR et al., 2021; VILARDELL et al., 2022). No entanto, o método mais utilizado para esse propósito é a teoria do valor extremo (*Extreme Values Theory* (EVT)). EVT é uma abordagem estatística baseada no estudo de eventos raros. Embora MBPTA com EVT tenha sido extensivamente estudado e aplicado com sucesso em inúmeros trabalhos, (HANSEN; HISSAM; MORENO, 2009; GRIFFIN; BURNS, 2010; LU et al., 2011; CUCU-GROSJEAN et al., 2012; KOSMIDIS et al., 2013b; ABELLA et al., 2015), alguns pesquisadores têm levantado preocupações (GRIFFIN; BURNS, 2010; LIMA; DIAS; BARROS, 2016; GIL et al., 2017; VASCONCELOS; LIMA, 2022). Essas preocupações abrangem questões relacionadas à discretização dos dados, possíveis dependências nos dados medidos, a garantia de uma aplicação confiável de EVT e a interpretação dos resultados quando ela é aplicada. Além disso, independentemente da ferramenta MBPTA utilizada para realizar estimativas de pWCET, é necessário que as medições do tempo de execução capturem o comportamento do programa em estudo. Isso implica cobrir os diferentes caminhos de execução gerados pelo programa ao processar sua entrada, assim como os possíveis estados do hardware. No entanto, esse tipo de cobertura é complexo, e nem sempre é possível de ser realizado.

1.3 OBJETIVOS E CONTRIBUIÇÕES

Considerando os problemas mencionados anteriormente, o objetivo desta pesquisa consiste em oferecer soluções utilizando técnicas de estatística e inteligência computacional para aprimorar a precisão e confiabilidade das estimativas de pWCET. Para esse propósito, duas abordagens são apresentadas: a representação do tempo de execução com base em eventos de hardware; e uma nova abordagem MBPTA *multipath* que realiza estimativas pWCET utilizando amostras com observações do tempo de execução e do número de instruções executadas. Em ambos os estudos, $T(n)$ representa o tempo de execução (medido em ciclos do processador) e n a quantidade de instruções executadas.

1.3.1 Representação do tempo de execução com base em eventos de hardware

Este estudo visa realizar a modelagem do comportamento de execução de programas em função de eventos relacionados ao hardware. Esses eventos podem ter suas ocorrências monitoradas por meio de unidades de monitoramento de desempenho (*Performance Monitoring Unit* (PMU)), presentes em muitas arquiteturas de hardware, principalmente as mais atuais (DONGARRA et al., 2003). Ao monitorar $T(n)$, n e os eventos de hardware, é possível construir um modelo de execução do programa. O interesse é que $T(n)/n$, denominado neste estudo de *execution pace*, possa ser modelado em função das ocorrências dos eventos de hardware. Para verificar essa hipótese, foi conduzido um experimento utilizando uma arquitetura ARM Cortex-A53 (ARM, 2016). Os tipos e números de eventos relacionados ao hardware que podem ser monitorados, bem como as metodologias para usar esses contadores, variam entre as arquiteturas (WEAVER; MCKEE, 2008). Para o ARM Cortex-A53, nem todos os eventos podem ser monitorados simultaneamente. Essa plataforma possui uma PMU equipada com sete registradores, um dos quais é reservado para contar o evento relacionado ao número de ciclos gastos $T(n)$. Assim, é possível monitorar apenas seis eventos simultaneamente, enquanto existem 59 eventos contáveis. Para superar essa limitação, um método para selecionar um subconjunto dos eventos mais relevantes para cada programa analisado é apresentado.

Em resumo, o método proposto para a seleção de eventos relevantes é o seguinte: o programa analisado é monitorado durante sua execução em um conjunto de execuções, cada uma considerando um grupo de eventos para dois cenários de execução. Um cenário é considerado como base, enquanto o outro, o cenário de estresse, representa possíveis interferências às quais o programa analisado pode estar sujeito. Para os experimentos realizados neste estudo, o cenário de estresse é configurado para induzir interferências relacionadas a acessos à memória e cache. O impacto que cada evento tem no tempo de execução é medido pelas diferenças observadas nas médias e variâncias em suas distribuições empíricas para os cenários base e de estresse. Após selecionar os eventos que apresentaram as maiores diferenças, a abordagem é avaliada verificando se os eventos selecionados podem modelar o *execution pace* associado ao programa analisado. Claramente, ao não utilizar todos os eventos, erros de modelagem são introduzidos. No entanto, evidências empíricas indicam que o modelo ainda pode ser considerado preciso para fins de suporte ao MBTA, conforme demonstramos por meio de um conjunto de experimentos realizados com 15 programas do *benchmark* Mälardalen (GUSTAFSSON et al., 2010).

O modelo apresentado é baseado em técnicas de regressão, para a qual várias abordagens são testadas, desde regressão linear até técnicas de aprendizado de máquina (*Machine Learning* (ML)) mais elaboradas (THEOBALD, 2020; CONWAY; WHITE, 2012; BISHOP, 2011). Os resultados obtidos em cada abordagem são comparados entre si. Como será visto, em geral, todas elas têm um desempenho semelhante.

1.3.2 DBL-MBPTA: Abordagem MBPTA baseada em ciclos e instruções

Neste estudo, é apresentada a *Distance Between Lines - MBPTA* (DBL-MBPTA), uma nova abordagem MBPTA que, ao contrário das abordagens MBPTA atuais, que consideram somente $T(n)$ como variável aleatória de interesse, a DBL-MBPTA também considera

o n em cada medição. Ao incluir n na análise, é proporcionado uma melhor caracterização do comportamento do programa em estudo, especialmente em relação a diferentes caminhos de execução (isto é, diferentes valores de n) e condições de interferência, permitindo que os dados medidos possam ser avaliados e aprimorados. A variável de interesse modelada na DBL-MBPTA é baseada nas distâncias relativas das medições (n,$T(n)$) em relação às linhas que limitam os dados superior e inferiormente, no qual é denominada na abordagem de linhas de referência.

O funcionamento da DBL-MBPTA é o seguinte: primeiramente, são realizadas medições iniciais do programa em estudo. Após a coleta desses dados, a DBL-MBPTA verifica se todo o intervalo de instruções de interesse, previamente especificado pelo analista, foi contemplado. Caso contrário, a DBL-MBPTA, utilizando técnicas de ML, sugere novos conjuntos de dados de entrada que poderiam explorar esses caminhos de execução ainda não observados. Execuções adicionais são então realizadas com base nessas sugestões. Este processo é repetido iterativamente até que todo o intervalo de instruções de interesse seja coberto. Dessa forma, o conjunto final de medições consiste tanto nas medições iniciais quanto nas medições realizadas com os dados de entrada sugeridos via ML. Para evitar possíveis agrupamentos de dados, é extraída uma amostra mais uniformemente distribuída dentro do intervalo de instruções de interesse a partir do conjunto final de medições. Esse procedimento é denominado na abordagem de redução de viés. Com a amostra definida, as linhas de referência são estabelecidas e os limites probabilísticos do tempo de execução são estimados com base na distância relativa dos dados em relação a essas linhas.

Para realizar as estimativas de pWCET, a DBL-MBPTA utiliza a EVT, devido ao amplo conjunto de métodos e ferramentas que ela oferece e que têm sido amplamente aplicados em diversas áreas. A avaliação da abordagem é realizada empregando tanto conjuntos de dados sintéticos quanto reais. Os dados sintéticos foram utilizados para verificar se a DBL-MBPTA consegue fornecer estimativas consistentes, considerando casos em que as estimativas baseadas em EVT são conhecidas. Já os dados reais foram coletados de diferentes programas com características variadas, selecionados a partir dos *benchmarks* Malardalen (GUSTAFSSON et al., 2010) e TACLeBench (FALK et al., 2016), e executados em uma arquitetura ARM Cortex-A53 (ARM, 2016). Os resultados obtidos evidenciam a eficácia do DBL-MBPTA.

1.4 HIPÓTESE

Os estudos realizados neste trabalho foram baseados na seguinte hipótese:

> A combinação de técnicas de otimização, estatísticas e inteligência computacional, aliadas à análise de eventos de hardware, pode melhorar a compreensão do comportamento de tarefas em RTS que operam em arquiteturas modernas. Nesse contexto, ao realizar estimativas de pWCET com MBPTA, melhorias no protocolo de medição podem ser implementadas, resultando em amostras mais representativas. Consequentemente, estimativas mais precisas podem ser obtidas, contribuindo para a construção de sistemas mais eficientes e confiáveis.

1.5 PUBLICAÇÕES

Os estudos realizados neste trabalho resultaram nos seguintes artigos publicados:

1. T. N. C. Andrade, G. Lima, V. M. C. Lima, Y. Abdeddaïm and L. C. Grosjean, 'On the Selection of Relevant Hardware Events for Explaining Execution Time Behavior,' 2021 XI Brazilian Symposium on Computing Systems Engineering (SBESC), Florianopolis, Brazil, 2021, pp. 1-8.

2. T. N. C. Andrade, G. Lima, V. M. C. Lima, S. Ben-Amor, I. Hawila, and L. Cucu-Grosjean, 'On the impact of hardware-related events on the execution of real-time programs', Design Automation for Embedded Systems, vol. 27, no. 4, pp. 275–302, Dec. 2023.

3. T. N. C. Andrade, G. Lima and V. M. C. Lima, Drawing Lines for Measurement-Based Probabilistic Timing Analysis. 45$^{\text{th}}$ IEEE Real-Time Systems Symposium (RTSS), 2024, York, United Kingdom.

1.6 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em 7 capítulos. O Capítulo 1 apresenta o contexto geral em relação aos sistemas de tempo real e introduz as abordagens estudadas. O Capítulo 2 apresenta a fundamentação teórica, detalhando tópicos relacionados a sistemas de tempo real, métodos de análise temporal e inteligência computacional. O Capítulo 3 apresenta a revisão da literatura relacionada à pesquisa. O Capítulo 4 detalha a abordagem de representação do tempo de execução com base em eventos de hardware, explorando as implicações e contribuições. O Capítulo 5 apresenta a abordagem DBL-MBPTA, detalhando cada etapa de sua aplicação, bem como seus requisitos e limitações. O Capítulo 6 utiliza as abordagens dos capítulos 4 e 5 para investigar como as informações fornecidas pelos eventos de hardware podem ser utilizadas para aprimorar os resultados obtidos com MBPTA. Por fim, o Capítulo 7 apresenta a conclusão do trabalho e propõe direcionamentos para pesquisas futuras.

Capítulo

2

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são abordados os principais conceitos relacionados aos estudos desenvolvidos nesta tese. A Seção 2.1 detalha as características e requisitos dos sistemas de tempo real, considerando classificações, aplicações e desafios. A Seção 2.2 descreve a importância da análise temporal para os sistemas de tempo real, bem como as metodologias utilizadas para essa finalidade. Por fim, a Seção 2.3 descreve as técnicas de aprendizado de máquina empregadas nas abordagens apresentadas nos próximos dois capítulos.

2.1 SISTEMAS DE TEMPO REAL

Os sistemas de tempo real (RTS) são sistemas computacionais que precisam concluir suas tarefas sem ultrapassar o tempo máximo de execução pré-estabelecido. O comportamento correto de um RTS depende não somente dos resultados computados, mas também do tempo em que esses resultados são produzidos. Para atender os requisitos temporais, é importante compreender tanto os comportamentos individuais das tarefas quanto o comportamento global do sistema. Por comportamento do sistema, entende-se a maneira como o sistema opera, considerando seus componentes, recursos, processos e tarefas. Geralmente, o tempo de execução de uma tarefa é medido em unidades de ciclos do processador.

RTS vem sendo amplamente aplicado em diversas áreas. Abaixo estão listados alguns exemplos de sua utilização:

- Veículos automotivos — RTS é empregado na indústria automotiva para cumprir diferentes objetivos. Alguns direcionados à segurança do motorista e passageiros, como freio ABS e controle de estabilidade (OLIVEIRA, 2018). Outros estão voltados ao funcionamento do motor, como injeção de combustível e controle de temperatura. Nos últimos anos, também foram inseridos sistemas de RTS relacionados à assistência durante a condução, como o controle de cruzeiro adaptativo e assistente de faixa;

- Telecomunicações — Para manter a qualidade do serviço na telecomunicação, as aplicações exigem pontualidade. São exemplos de RTS aplicados na telecomunicação: transmissão de áudio e vídeo em chamadas de videoconferência, procedimento de encaminhamento de chamadas, detecção automática de falhas (OLIVEIRA, 2018; VIEIRA et al., 2021);

- Mercado Financeiro — Nesta área, o RTS é aplicado principalmente no processamento de informações, negociações automáticas, monitoramento de cotações e taxas de câmbio (OLIVEIRA, 2018). O objetivo é auxiliar em uma tomada de decisão mais assertiva por parte dos investidores;

- Robótica — Grande parte dos sistemas robóticos estão sujeitos a restrições temporais. Para o controle e monitoramento desses equipamentos, é necessário que os eventos sejam respondidos no tempo correto (PINTO; WEHRMEISTER; OLIVEIRA, 2021; FAUDZI et al., 2012). Em uma linha de produção com equipamentos robóticos, por exemplo, a não detecção ou a detecção tardia de um produto reprovado pode ter um impacto negativo na eficiência e na qualidade geral do processo de produção;

- Equipamentos militares — No contexto de equipamentos militares, o RTS é amplamente integrado principalmente nos dispositivos mais modernos. Alguns exemplos: detecção, localização e rastreamento de objetos; otimização de recursos e auxílio nas tomadas de decisões; monitoramento, rastreamento e direcionamento de mísseis (OLIVEIRA, 2018);

- Medicina — As aplicações de RTS na medicina variam desde dispositivos que monitoram os pacientes, registrando os sinais vitais e administrando medicamentos e nutrientes na corrente sanguínea até sistemas que auxiliam em procedimentos cirúrgicos (SERHANI et al., 2020; ELSAYED et al., 2022);

- Automação residencial — Para assegurar o perfeito funcionamento de um sistema de automação residencial, é importante que todas as tarefas respeitem o tempo de resposta pré-estabelecido (BU et al., 2018). São exemplos de aplicações de automação residencial que utilizam de RTS: sensores de presença, sistemas de multimídia e sistemas de eficiência energética.

Uma tarefa é classificada como periódica quando sua liberação é realizada em intervalos regulares. Quando não existe um padrão ou período fixo para ser ativada, ou a tarefa é invocada a partir de um evento (interno ou externo), essa tarefa é classificada como aperiódica. Independentemente dos tipos de tarefas que um RTS precise executar, é necessário que os requisitos temporais sejam respeitados. Um RTS deve ser capaz de responder aos estímulos do seu ambiente em intervalos de tempo determinados por esse ambiente. O prazo máximo em que o resultado de uma tarefa deve ser produzido é chamado de *deadline*. Se um resultado tiver utilidade mesmo após o *deadline* ter expirado, o *deadline* é classificado como *soft*; se consequências graves podem ocorrer caso um *deadline* seja perdido, o *deadline* é chamado de *hard* (KOPETZ, 2022). Assim, um RTS pode ser

classificado de acordo as restrições temporais definidas pela própria aplicação. Aroca (2008) fornece uma descrição detalhada de cada classificação, conforme apresentada a seguir:

- *Soft real-time*: RTS onde a falha ou ultrapassagem do tempo de execução mais longo esperado não resulta em danos significativos, contudo o sistema deixa de cumprir a sua função;
- *Hard real-time*: RTS onde falhas podem resultar em graves consequências, tanto financeiras quanto em perdas de vidas humanas.

A Figura 2.1 ilustra alguns exemplos de aplicação de RTS. O quadro verde (à esquerda) e rosa (à direita) apresentam, respectivamente, exemplos classificados como *soft real-time* e *hard real-time*. Um atraso na emissão de um comprovante em um caixa de supermercado ou no carregamento de uma imagem em uma transmissão ao vivo não ocasiona grandes problemas, a não ser uma insatisfação por parte do usuário. No entanto, se houver uma falha no sistema de controle de cruzeiro adaptativo (ACC), no freio ABS ou na abertura do *airbag* em um veículo, poderá ocorrer danos graves ou fatais ao motorista.

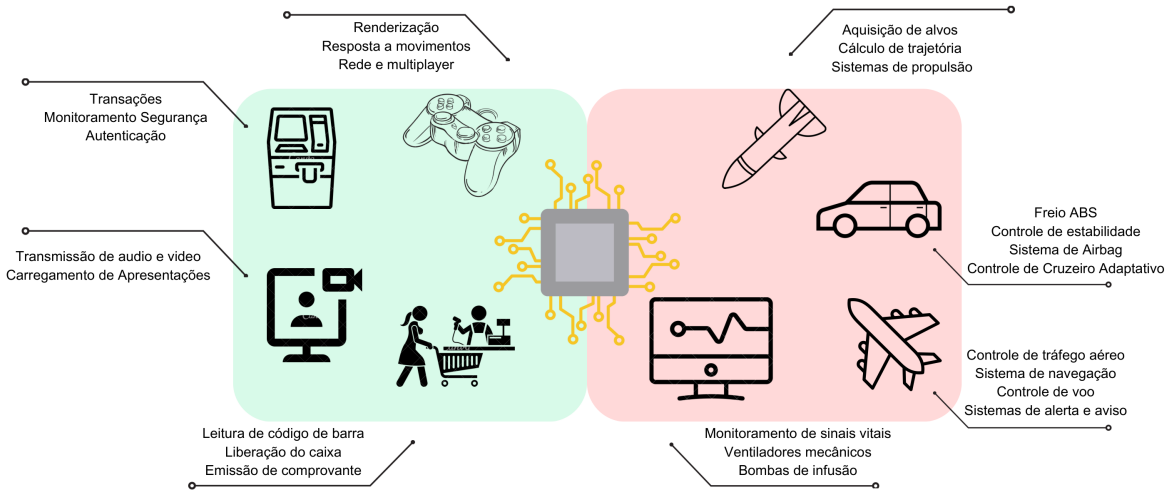


Figura 2.1 Exemplos de RTS *Soft Real Time* (quadro em verde) e *Hard Real Time* (quadro em rosa) presentes em sistemas embarcados na atualidade.

Uma propriedade importante em um RTS é a previsibilidade temporal (*predictability*). Ela está associada com a capacidade do desenvolvedor de poder antecipar, antes do sistema entrar em operação real, se as tarefas serão executadas dentro de seus prazos especificados (OLIVEIRA, 2018). O tempo de execução de pior caso (WCET) é o tempo máximo de execução que uma tarefa pode levar para ser concluída em seu pior cenário. O WCET depende não somente do código da tarefa, mas também da arquitetura utilizada

pelo sistema e dos dados de entrada oferecidos para a tarefa. Na próxima seção, são apresentados os principais conceitos referentes à análise temporal em RTS, bem como as diferentes abordagens empregadas para estimar o WCET de uma tarefa.

2.2 ANÁLISE TEMPORAL

Em estruturas de hardwares simples, a variabilidade do tempo de execução entre diferentes execuções são limitadas pela estrutura da tarefa e pelos dados de entrada (CAZORLA et al., 2019). A estrutura da tarefa determina os possíveis caminhos de execução que podem ser seguidos. Os dados de entrada definem qual caminho de execução deve ser seguido. Em estruturas de hardware modernas, é comum encontrar componentes como memória cache, *pipeline* de instruções e *branch predictors*. Embora esses componentes exerçam um papel importante no aumento do desempenho geral do sistema, eles também apresentam desafios significativos para a determinação do WCET em RTS. Nesses modelos de arquiteturas, além dos atributos relacionados à execução da tarefa (estruturas e dados de entrada), é necessário considerar o estado atual do hardware durante uma análise temporal. O estado do hardware é representado pelo conjunto de atividades em execução no momento, incluindo os tipos de instruções em progresso no *pipeline*, o conteúdo armazenado na cache, entre outros fatores relevantes.

Em RTS, a análise temporal engloba um conjunto de métodos que visam determinar o tempo máximo que as tarefas podem levar, considerando todas as possíveis condições adversas e variações, como diferentes dados de entrada e estados do hardware. Até o momento, esses métodos são classificados em tradicionais e probabilísticos. A seguir, essas classificações são detalhadas.

2.2.1 Métodos tradicionais

Em arquiteturas determinísticas, onde o comportamento temporal é previsível e conhecido, os métodos tradicionais são os mais utilizados. Esses métodos são divididos em três categorias: os estáticos, os baseados em medição e os híbridos.

2.2.1.1 Análise temporal estática. A análise temporal estática (*Static Timing Analysis* (STA)) engloba o conjunto de métodos de análise temporal que realiza estimativas WCET sem a necessidade de executar as tarefas. Ao invés disso, a STA considera os efeitos de todas as entradas possíveis, incluindo possíveis estados do sistema, juntamente com a interação da tarefa com o hardware. As análises se baseiam em modelos matemáticos do software e do hardware envolvidos. Dado que os modelos sejam suficientemente precisos, o resultado é uma estimativa de tempo segura que é maior ou igual ao WCET real (SANDELL et al., 2006).

A análise de WCET por STA é dividida em três fases. Primeiro, é implementado um grafo de fluxo do código da tarefa, contendo todos os possíveis caminhos de execução. O grafo de fluxo não depende da máquina na qual a tarefa é executada, mas das entradas fornecidas à tarefa. O objetivo é extrair o comportamento dinâmico da tarefa, incluindo as dependências entre as instruções, a quantidade de loops percorridos, quais funções e

variáveis são acessadas, entre outros. Em seguida, as características do hardware são consideradas, analisando o comportamento temporal das instruções da tarefa examinada na primeira fase. Esse processo pode ser realizado em uma arquitetura real ou em um simulador. Na terceira e última fase ocorre a combinação do grafo de fluxo gerado na primeira fase com a análise arquitetural realizada na segunda fase para obter o limite superior do WCET da tarefa. A Figura 2.2 apresenta as três fases da STA.

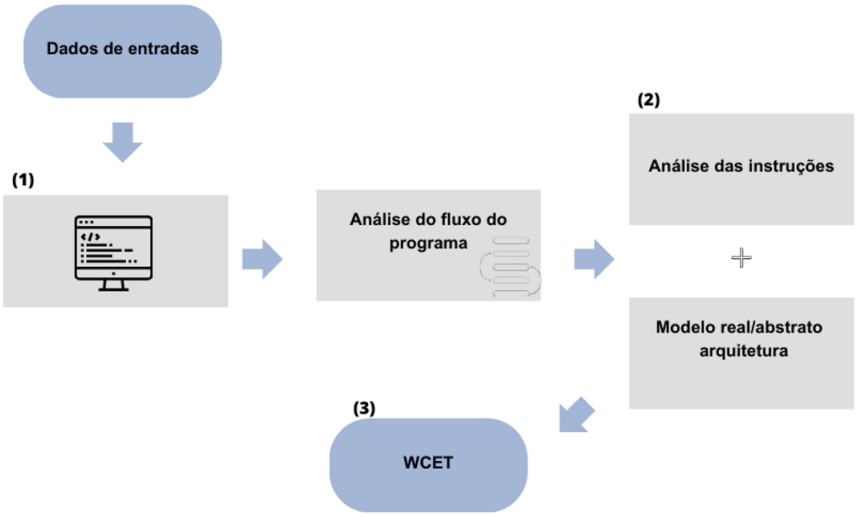


Figura 2.2 Exemplo de uma análise temporal estática (STA). Na fase 1 é realizada uma análise do fluxo da tarefa. Na fase 2 é realizada uma análise dos caminhos do grafo de fluxo no modelo de arquitetura. Na fase 3 uma combinação da fase 1 e 2 é realizada para derivar o WCET da tarefa.

Na terceira fase da STA, a programação linear inteira (*Integer Linear Programming* (ILP)) é empregada para estimar o WCET. O grafo de fluxo do código da tarefa é traduzido em programas lineares inteiros, e as informações sobre o fluxo de controle são convertidas em restrições adicionais. A função objetivo expressa o tempo de execução da tarefa em análise. Seu valor máximo é um limite superior para todos os tempos de execução (WILHELM et al., 2008; ROCHANGE; UHRIG; SAINRAT, 2013).

Em arquiteturas modernas, aplicar STA para determinar o WCET tornou-se inviável. A imprevisibilidade introduzida pelos novos componentes de hardware compromete a precisão das estimativas estáticas, como, por exemplo, quando é necessário determinar qual conteúdo está presente na memória cache, e se o conteúdo possui dependências de dados. Além disso, o avanço contínuo dos softwares, que incluem códigos complexos com múltiplos loops e dependências entre variáveis e dados de entrada, também dificulta a análise temporal por meio da STA.

2.2.1.2 Análise temporal baseada em medição.

A análise temporal baseada em medição (*Measurement-Based Timing Analysis* (MBTA)) envolve a coleta de dados de execução em uma arquitetura real ou em um simulador. Após realizar a coleta, o tempo de execução máximo observado é registrado. Esse valor pode ser utilizado como uma

estimativa do WCET. Alternativamente, uma margem de segurança pode ser adicionada ao tempo máximo observado (DAVIS; CUCU-GROSJEAN, 2019). Normalmente, o dado medido é o número de ciclos de processador necessários para executar as instruções da tarefa em análise. Em uma arquitetura real, as informações da execução podem ser coletadas por meio de contadores de eventos. O uso de um simulador tem a vantagem de permitir a análise em um ambiente controlado, onde, por exemplo, é possível definir os estados do hardware antes de realizar as medições, algo difícil de obter em um hardware real. O estado do hardware tem um impacto direto nos tempos de execução observados. A análise de tempo por meio de simulador pode ser realizada antes da construção do hardware alvo. No entanto, para evitar erros de medição, é necessário realizar uma modelagem precisa da arquitetura do hardware no simulador. Isso pode ser um desafio, uma vez que modificações são esperadas durante a etapa de construção do hardware (ROCHANGE; UHRIG; SAINRAT, 2013).

Devido à sua natureza simplificada, os métodos de MBTA são os mais acessíveis para aplicação, especialmente porque, para encontrar o WCET, não é necessário ter um conhecimento profundo do código da tarefa nem do hardware alvo. De qualquer forma, para que o WCET estimado seja confiável, é necessário que grande parte dos caminhos possíveis de execução sejam explorados. Isso pode ser um problema, uma vez que o caminho mais longo pode não ter sido observado. Ainda assim, é uma abordagem amplamente utilizada em projetos industriais (exceto para aqueles que precisam passar por procedimentos de certificação rigorosos, como no domínio da aviação) (ROCHANGE; UHRIG; SAINRAT, 2013).

Para realizar análise temporal utilizando MBTA, é necessário ter um protocolo de medição estabelecido previamente. Esse protocolo abrange informações como a identificação dos diferentes caminhos de execução a serem explorados, a definição de conjuntos de valores de entrada, as técnicas para coletar informações do hardware e as configurações específicas do próprio hardware. Em plataformas modernas, a definição de um protocolo de medição adequado tornou-se uma tarefa complexa, devido à dificuldade em determinar quais valores das variáveis de entrada, estado do software e estado do hardware levariam a um WCET seguro (DAVIS; CUCU-GROSJEAN, 2019).

2.2.1.3 Análise temporal híbrida. Os métodos de análise temporal híbrida realizam uma combinação entre métodos de STA e MBTA. A abordagem híbrida é geralmente composta por três fases distintas. A primeira fase consiste na decomposição do comportamento da tarefa em segmentos (subconjuntos), garantindo a cobertura adequada em cada um desses segmentos. Na segunda fase, é determinado o tempo de execução para cada segmento. Por fim, na terceira e última fase, ocorre a composição dos segmentos medidos para obter as estimativas do WCET (ZOLDA; BÜNTE; KIRNER, 2009). A Figura 2.3 apresenta um grafo de fluxo contendo as etapas da abordagem temporal híbrida.

As características dos métodos de STA são observadas nas fases inicial e final do processo. Na fase inicial, é necessário analisar o código-fonte para extrair informações sobre os possíveis caminhos de execução e dividir a tarefa em segmentos. Na fase final são empregadas técnicas de análise estática para combinar os tempos medidos em cada segmento com as informações dos caminhos de execução. As características dos métodos

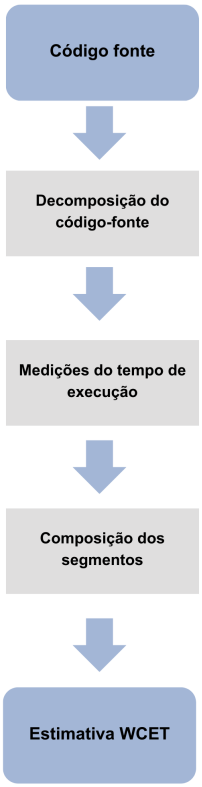


Figura 2.3 Fases da análise temporal híbrida. Imagem adaptada de Zolda, Bünte e Kirner (2009).

de MBTA são observadas na segunda fase, na qual ocorre a medição do tempo de execução de cada segmento.

A análise temporal híbrida requer uma cobertura completa (ou próxima disso) do código-fonte e do estado do hardware. A cobertura do código pode ser alcançada via testes de cobertura, que verificam se os requisitos do código estão completos, se o vetor de entrada de dados é suficiente e se todo o código-fonte está sendo coberto (GIL, 2020). Em arquiteturas modernas, realizar a cobertura do estado do hardware é uma tarefa complexa. O tempo de execução de um segmento pode depender do histórico de execução e, consequentemente, dos segmentos anteriores percorridos. Esse cenário dificulta a composição precisa da estimativa geral do WCET a partir das observações de segmentos individuais, o que pode resultar em uma degradação da precisão dessa estimativa (DAVIS; CUCU-GROSJEAN, 2019).

2.2.2 Análise temporal probabilística

Embora os processadores modernos incluam componentes de hardware que melhoram o desempenho dos sistemas, eles também introduzem uma aleatoriedade que dificulta a aplicação das abordagens tradicionais de análise temporal (KOSMIDIS et al., 2013a). Essas abordagens, por sua natureza determinística, são incapazes de capturar o comportamento

aleatório desses componentes, o que pode levar a estimativas imprecisas.

A Análise temporal probabilística (*Probabilistic Timing Analysis* (PTA)) é um conjunto de métodos para análise temporal que incorpora incertezas relacionadas ao tempo de execução no cálculo do WCET. Os métodos tradicionais, descritos na Seção 2.2.1, estabelecem limites superiores rigorosos para o tempo de execução que poderia ocorrer em uma única execução da tarefa dentre todas as execuções possíveis. Quando os métodos tradicionais são aplicados, é fornecido um único valor de WCET. Os métodos de PTA estabelecem limites superiores rigorosos para a distribuição dos tempos de execução que poderiam ocorrer em algum cenário de operação, dentre todos os cenários possíveis (DAVIS; CUCU-GROSJEAN, 2019). Quando os métodos probabilísticos são utilizados, é retornada uma distribuição probabilística de WCET, denominada de *Probabilistic Worst-Case Execution Time* (pWCET). A probabilidade de exceder um determinado pWCET é chamada de probabilidade de excedência, representada pela função de excedência (2.1), também denominada de função de distribuição cumulativa complementar (*Complementary Cumulative Distribution Function* (CCDF)).

$$p = Pr\{\max(X) > x\} \tag{2.1}$$

A função de excedência calcula a probabilidade p de que o tempo de execução mais longo observado X para uma tarefa exceda um determinado valor x (ALTMEYER; CUCU-GROSJEAN; DAVIS, 2015). A PTA pode ser aplicada tanto em um contexto estático (*Static Probabilistic Timing Analysis* (SPTA)) quanto em um contexto baseado em medições (MBPTA) (KOSMIDIS et al., 2013a).

2.2.2.1 Análise temporal probabilística estática. Semelhante aos métodos de STA, os métodos de SPTA não executam a tarefa no hardware real ou em um simulador. Em vez disso, eles analisam o código da tarefa e as informações sobre os valores de entrada, juntamente com um modelo abstrato do comportamento do hardware. A diferença para STA é que SPTA considera o comportamento aleatório do sistema por meio do uso de distribuições de probabilidade, construindo um limite superior na distribuição dos valores do pWCET (DAVIS; CUCU-GROSJEAN, 2019). Em SPTA, as distribuições de probabilidade do tempo de execução para operações individuais ou componentes são determinadas de forma estática a partir da análise do processador e da tarefa (CAZORLA et al., 2012).

O modelo clássico de SPTA, inicialmente desenvolvido para tarefas de caminho único (ou seja, tarefas que utilizam a mesma entrada em todas as execuções, mantendo o mesmo número de instruções), pode ser resumido em duas etapas. Na primeira etapa, a função de massa de probabilidade (*Probability Mass Functions* (PMF)) é definida, obtendo os valores do pWCET para cada instrução da tarefa. Na segunda etapa, é realizada a convolução entre as PMF de cada instrução (DAVIS, 2013).

Para auxiliar no entendimento, a seguir é apresentado um pequeno exemplo de PMF de duas instruções distintas contidas em um determinado bloco de código, onde é considerada somente a memória cache como componente de hardware moderno. As probabilidades

de ocorrências de cache *miss* são definidas pelas PMF como 0,4 e 0,1 para a primeira e segunda instrução, respectivamente. Nesse exemplo, o custo para um *miss* na memória cache é de 2 ciclos, enquanto para um *hit* é de 1 ciclo. Após o processo de convolução entre as duas PMF, as probabilidades estimadas para executar as instruções são: 0,54% para o custo de dois ciclos, 0,42% para o custo de três ciclos, e 4% para o custo de quatro ciclos.

$$\begin{pmatrix} 1 & 2 \\ 0{,}6 & 0{,}4 \end{pmatrix} \otimes \begin{pmatrix} 1 & 2 \\ 0{,}9 & 0{,}1 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 4 \\ 0{,}54 & 0{,}42 & 0{,}04 \end{pmatrix}$$

Ao analisar somente a memória cache como um componente da arquitetura moderna, é necessário considerar as probabilidades de todas as sequências possíveis de *miss* e *hit* na cache. No entanto, esse processo pode se tornar exponencialmente complexo devido à grande variedade de combinações a serem realizadas (ALTMEYER; CUCU-GROSJEAN; DAVIS, 2015). Além da memória cache, outros componentes de hardware que impactam o desempenho do sistema precisam ser considerados ao realizar as estimativas pWCET para cada PMF. A modelagem dos demais componentes é um dos desafios enfrentados pelos métodos de SPTA (DAVIS; CUCU-GROSJEAN, 2019).

2.2.2.2 Análise temporal probabilística baseada em medição. A análise temporal probabilística baseada em medição (MBPTA) é um conjunto de métodos que combina as metodologias MBTA com modelos probabilísticos para realizar estimativas pWCET. Seu objetivo é contornar as dificuldades encontradas nos métodos tradicionais de análise temporal, derivando limites superiores para os tempos de execução das tarefas com base em dados coletados por meio de medições (CAZORLA et al., 2019; DAVIS; CUCU-GROSJEAN, 2019). Para fins didáticos, neste estudo, o método MBPTA apresentado nesta seção será denominado de método convencional de MBPTA.

As medições das tarefas podem ser realizadas por caminho de execução (*per-path*) ou por programa (*per-program*)(DAVIS; CUCU-GROSJEAN, 2019). A diferença entre os dois modos de medição reside no fato de que, para as medições por caminho de execução, a abordagem MBPTA é aplicada para realizar as estimativas pWCET em cada um dos caminhos, enquanto nas medições realizadas por programa, as estimativas pWCET consideram toda a tarefa. Em ambos os modos, o protocolo de medição adotado precisa abranger todos os possíveis caminhos de execução da tarefa.

Recentemente, MBPTA tem sido empregada em inúmeras pesquisas que envolvem análise temporal através da teoria dos valores extremos (EVT) (CUCU-GROSJEAN et al., 2012; KOSMIDIS et al., 2013; ABELLA et al., 2015; LIMA; BATE, 2017). EVT está preocupada com eventos raros, geralmente situados além do que é observado em amostras coletadas. Para esse propósito, EVT oferece um conjunto robusto de métodos e ferramentas que têm sido aplicados em diversas áreas, abrangendo desde meteorologia até o mercado de ações. Em particular, se X é a variável aleatória de interesse, EVT fornece meios para descrever assintoticamente o comportamento de $\max(X)$. Mais especificamente, se existe uma distribuição de probabilidade não degenerada:

$$F_Z(z) = Pr\{\max(X) \leq z\}$$

então F_Z pode ser modelada como a distribuição de três parâmetros $\mathrm{GEV}_Z(z)$, ou seja, *Generalized Extreme Value* (GEV) (COLES, 2001),

$$\mathrm{GEV}_Z(z) = \exp\left\{-\left[1+\xi\left(\frac{z-\mu}{\sigma}\right)\right]^{-1/\xi}\right\} \tag{2.2}$$

No contexto de MBPTA, vários autores adotam a distribuição de Gumbel (caso específico de GEV onde o parâmetro $\xi = 0$), que pode ser expressa como (COLES, 2001):

$$G_Z(z) = \exp\left\{-\exp\left[-\left(\frac{z-\mu_G}{\sigma_G}\right)\right]\right\}. \tag{2.3}$$

A estimativa dos parâmetros de (2.2) ou (2.3) requer a amostragem de $\max(X)$, que é obtida dividindo uma amostra de X em blocos de tamanhos iguais e selecionando o valor máximo observado em cada bloco. Esse método de obtenção da amostra é denominado de *Block Maxima* (BM). A Figura 2.4 ilustra a aplicação do BM em uma amostra de dados real do programa MSort proveniente do *benchmark* Mälardalen (GUSTAFSSON et al., 2010), executado em uma plataforma Raspberry Pi 3B (RASPBERRY, 2018). Para esse exemplo, a amostra foi dividida em 10 blocos.

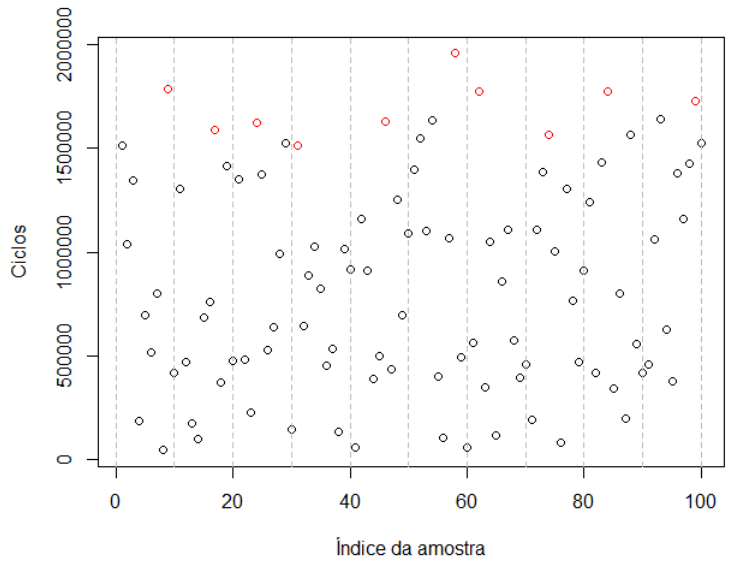


Figura 2.4 Exemplo da aplicação do método BM. A amostra de dados é dividida em blocos e o máximo de cada bloco é selecionado.

Alternativamente, pode-se modelar $\max(X)$ considerando seus valores acima de um determinado limiar u (*Peaks-over Threshold* (PoT)). Nesse caso, o modelo descreve $\Pr\{X > u+y | X > u\}$, que representa a probabilidade condicional de que X esteja acima de $u+y$, dado que $X > u$. Aqui, u é o limiar escolhido, e y o excesso acima desse limiar.

A Figura 2.5 apresenta a aplicação do método PoT na mesma amostra de dados utilizada em Figura 2.4.

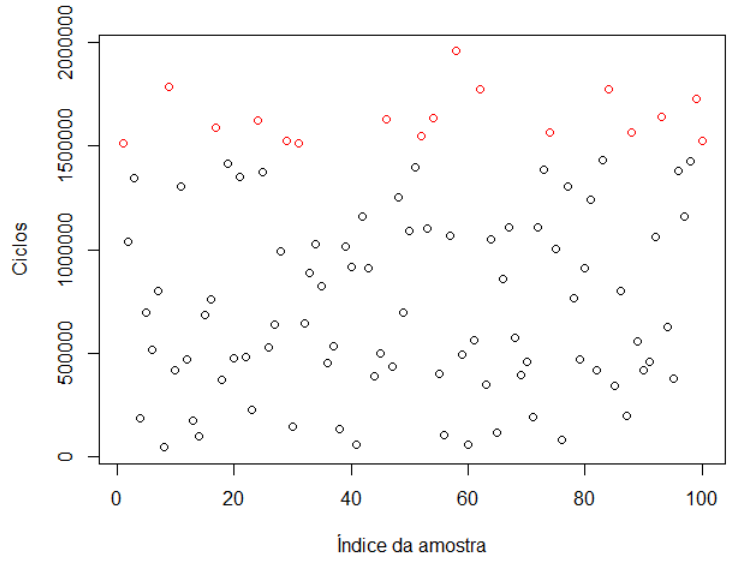


Figura 2.5 Exemplo da aplicação do método PoT com o limiar $w = 1.500.000$.

Ao definir $Y = X - u$, a distribuição de Y para um grande limiar escolhido u pode ser derivada de (2.2) e consiste na distribuição de pareto generalizada (*Generalized Paret* (GP)) de 2 parâmetros (COLES, 2001),

$$H_Y(y) = 1 - \left(1 + \frac{\xi y}{\sigma_H}\right)^{-1/\xi}, \tag{2.4}$$

definida sobre $\{y : y > 0 \text{ e } (1 + \xi y/\sigma_H) > 0\}$, onde $\sigma_H = \sigma + \xi(u - y)$. Ou seja, (2.2) e (2.4) estão relacionadas entre si. Quando $\xi = 0$, (2.4) assume a forma da distribuição Exponencial, um caso especial também usado por alguns autores no contexto de MBPTA (ARCARO; SILVA; OLIVEIRA, 2018; ABELLA et al., 2017).

O uso de (2.2) ou (2.4) deve ser equivalente em termos assintóticos, mas para uma determinada amostra em mãos, pode-se obter melhor qualidade do modelo com um método em relação ao outro. Por exemplo, em alguns casos, pode ser vantajoso utilizar (2.4), uma vez que a amostra de máximos para estimar o modelo (2.2) pode descartar vários máximos por bloco. Ao utilizar (2.4), é necessário escolher, em vez de um tamanho de bloco, um valor de limiar que conduza a uma boa qualidade do modelo. No entanto, os modelos GP são mais vulneráveis a possíveis observações agrupadas (o que pode indicar possíveis dependências de dados).

EVT deve ser aplicada com cautela (LIMA; DIAS; BARROS, 2016; GIL et al., 2017; VASCONCELOS; LIMA, 2022), pois nem todas as amostras são possíveis de serem mo-

deladas via EVT, e as escolhas de tamanhos de blocos e valores de limiar podem ter um efeito relevante na estimativa dos parâmetros do modelo.

2.3 MÉTODOS ESTATÍSTICOS E DE INTELIGÊNCIA COMPUTACIONAL

Esta seção apresenta os conceitos relacionados aos métodos estatísticos e de inteligência computacional utilizados nesta pesquisa. Detalhes sobre as bibliotecas e configurações específicas de cada ferramenta não são fornecidos aqui. Essas informações estão presentes nos capítulos 4 e 5. O objetivo desta seção é fornecer uma base teórica que permita a compreensão dessas técnicas.

Machine Learning (ML) é um campo da inteligência computacional que abrange uma variedade de algoritmos e métodos projetados para inferir padrões e extrair conhecimentos a partir de dados (CONWAY; WHITE, 2012). Após adquiridos, esses conhecimentos são utilizados para realizar previsões, fazer recomendações, estabelecer regras ou obter respostas (JANIESCH; ZSCHECH; HEINRICH, 2021).

ML pode ser categorizado em: supervisionado, não supervisionado e por reforço. No aprendizado supervisionado, o conjunto de dados utilizado durante o processo de aprendizagem possui variáveis de entrada (preditoras) e variáveis de saída (rótulos). No aprendizado não supervisionado, o conjunto de dados não possui saídas conhecidas. O objetivo é encontrar padrões ou estruturas no conjunto de dados para realizar classificações ou relacionamentos. No aprendizado por reforço, o modelo de ML realiza suas ações com base em modificações no ambiente e recebe respostas na forma de penalidades ou recompensas que guiam o modelo no processo de aprendizado (KAELBLING; LITTMAN; MOORE, 1995). Os estudos apresentados nesta tese focam exclusivamente no aprendizado supervisionado, uma vez que todas as amostras de dados trabalhadas contêm variáveis de entrada e variáveis de saída.

No aprendizado supervisionado, os dados são divididos em um conjunto de treinamento e um conjunto de teste. O conjunto de treinamento é utilizado para construir o modelo e configurar os parâmetros de treinamento. O conjunto de teste contém amostras cuja origem é conhecida, mas que não são incorporadas ao processo de modelagem. O conjunto de teste permite que o operador avalie a precisão do modelo (XU; GOODACRE, 2018). Os métodos de ML que se baseiam no aprendizado supervisionado estão divididos em duas categorias: métodos de regressão e métodos de classificação. Os métodos de regressão realiza previsões do valor de uma ou mais variáveis de saída contínuas t, dado o valor de um vetor x de D dimensões de variáveis de entrada. Nos métodos de classificação, a saída t é composta por valores discretos que representam as categorias (ou rótulos) às quais o vetor de entrada deve ser classificado (BISHOP, 2011). Após a fase de treinamento, espera-se que o algoritmo de ML consiga classificar as entradas em sua devida categoria. Em outras palavras, o modelo precisa conseguir atribuir rótulos ou classificações corretas a novos dados não vistos durante o treinamento, com base no conhecimento adquirido a partir dos dados rotulados de treinamento.

As próximas subseções apresentam uma breve introdução dos métodos estatísticos e de inteligência computacional utilizados neste trabalho. Todos os métodos mencionados podem lidar com aprendizado supervisionado, com ênfase na resolução de problemas de

regressão, o qual é o foco das abordagens apresentadas nesta tese.

2.3.1 Regressão linear múltipla

A Regressão linear múltipla (*Multiple Linear Regression* (MLR)) é um método estatístico utilizado para modelar a relação entre a variável dependente e as variáveis independentes. O objetivo é estimar os coeficientes associados a cada variável independente que melhor se ajustem aos dados observados (MONTGOMERY; PECK; VINING, 2006). Os coeficientes podem ser interpretados como a contribuição relativa de cada variável independente na explicação das variações na variável dependente. A equação da MLR pode ser expressa na forma:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon \,,$$

em que Y é a variável dependente, X_i é a i-ésima variável independente, β_i são os parâmetros associados as variáveis X_i e ϵ é o erro aleatório. O Método dos Mínimos Quadrados (MMQ) pode ser utilizado para estimar o vetor de coeficientes β associados ao modelo de MLR (KUTNER et al., 2004). Quando o modelo de regressão linear é composto por somente uma variável independente, ele é denominado regressão linear simples.

2.3.2 Floresta randômica

A floresta randômica (*Random Forest* (RF)) é um modelo de ML que é construído com base em árvores de decisão. Cada árvore na RF se baseia em uma coleção de variáveis aleatórias, que representa um subconjunto de características do conjunto de dados. Durante o treinamento, cada árvore é treinada em um subconjunto dos dados de treinamento. Em problemas de classificação, o rótulo que mais se repetiu entre todas as árvores é selecionado como resultado de saída. Em problemas de regressão, a predição final da RF é obtida pela média das predições de todas as árvores individuais (POLIKAR; ZHANG; MA, 2012; BREIMAN, 2001).

Ao utilizar RF, é necessário determinar o número de árvores que melhor se ajusta aos dados de treinamento. Essa quantidade pode ser obtida através do cálculo do erro médio quadrático (*Mean Squared Error* (MSE)). A Figura 2.6 representa a relação entre o número de árvores e o MSE para uma amostra de dados do programa ISort proveniente do *benchmark* Mälardalen (GUSTAFSSON et al., 2010), executado em um Raspberry Pi 3B (RASPBERRY, 2018). Ao utilizar um total de 200 árvores, o MSE estabiliza, sem mostrar melhorias significativas com a adição de mais árvores. Isso sugere que esse número é suficiente para o treinamento do modelo.

2.3.3 Redes neurais e redes neurais monotônicas

Redes Neurais Artificiais (RNA) consistem em funções de decisão interconectadas, conhecidas como nós, que interagem entre si por meio de conexões de unidades de processamento (neurônios artificiais) (THEOBALD, 2020). Cada neurônio em uma RNA possui uma função de ativação utilizada para introduzir não linearidade ao modelo. Esse

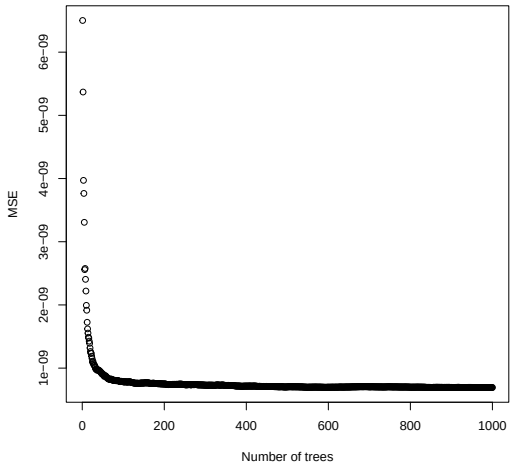


Figura 2.6 Gráfico ilustrando a relação entre o número de árvores em uma RF (eixo x) e o MSE (eixo y) utilizando um conjunto de dados do programa ISort proveniente do *benchmark* Mälardalen (GUSTAFSSON et al., 2010).

neurônio recebe os valores numéricos dos sinais de entrada e os pesos provenientes dos neurônios das camadas anteriores. Em seguida, por meio de uma operação de soma ponderada, o neurônio produz um valor de saída. Sendo k um neurônio artificial específico, sua saída y_k pode ser representada matematicamente por:

$$y_k = \varphi((\sum_{j=1}^{m} \omega_{kj} x_j) + b_k)$$

em que $x_1, x_2, ...x_m$ são os sinais de entrada; $w_{k1}, w_{k2}...w_{km}$ são os pesos sinápticos; b_k é o *bias*; e $\varphi(.)$ é a função de ativação (HAYKIN, 2005). Durante o processo de treinamento, cada peso numérico é ajustado para minimizar o erro entre a saída prevista e a saída desejada.

A monotonicidade entre as variáveis dependentes e independentes é um requisito em alguns domínios, como pontuação de crédito, riscos de seguro e diagnósticos médicos. Por exemplo, espera-se que um indivíduo com um salário mais alto tenha um valor de empréstimo mais alto aprovado. Um modelo sem essa propriedade monotônica pode não ser considerado confiável para fornecer uma base para decisões tão importantes (RUNJE; SHARATH, 2023). Durante a execução de uma tarefa, existem eventos de hardware que possuem relações monotônicas com o tempo de execução. Por exemplo, à medida que o número de ocorrências de cache *miss* aumenta, existe uma tendência de que o tempo de execução também aumente.

As RNA monotônicas (*Monotonic Neural Network* (MNN)) é um modelo de RNA que possui a característica de preservar a monotonicidade na relação entre as variáveis independentes e a variável dependente. A monotonicidade pode ser alcançada, por exemplo, considerando somente valores negativos ou positivos nos pesos (ARCHER; WANG, 1993),

modificando a função de perda (SILL; ABU-MOSTAFA, 1996; GUPTA et al., 2019) ou através da aplicação de programação linear inteira (LIU et al., 2022).

2.3.4 Redes neurais profundas

Em ML, o termo "profundo" (*Deep* do inglês) refere-se à ideia de múltiplos níveis, camadas ou estágios pelos quais os dados são processados para extrair as informações. O aprendizado profundo (*Deep Learning* (DL)) é um campo de ML que ganhou destaque significativo nos últimos anos devido à sua eficácia, especialmente à medida que o volume de dados aumenta (SARKER, 2021; ALZUBAIDI et al., 2021). DL melhorou significativamente o estado-da-arte em reconhecimento de fala, reconhecimento de objetos visuais, detecção de objetos e em muitos outros domínios, como descoberta de medicamentos e genômica (LECUN; BENGIO; HINTON, 2015).

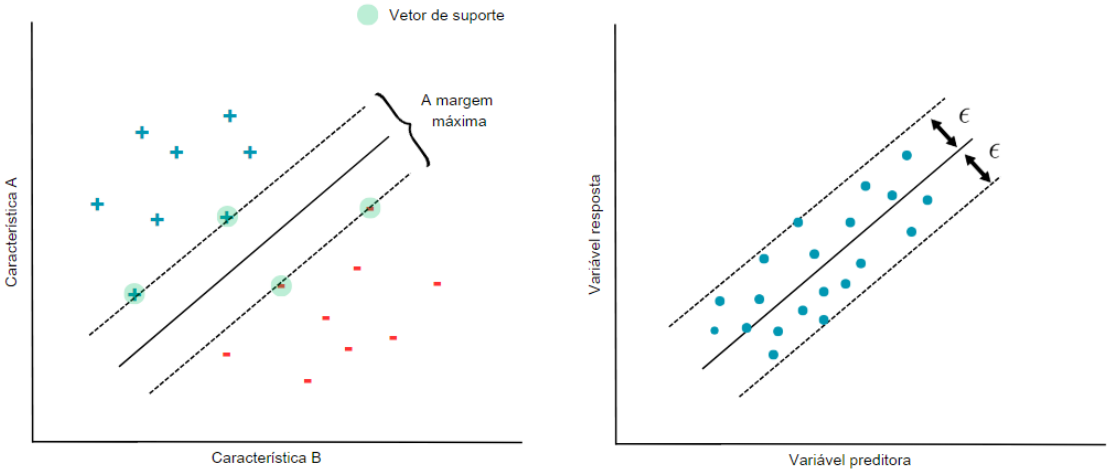
As redes neurais profundas (*Deep Neural Network* (DNN)) são modelos de RNA que exibem características de DL. A principal distinção entre RNA tradicionais e DNN reside no número de neurônios e camadas intermediárias (LIU et al., 2017). Enquanto um modelo tradicional pode ter somente algumas dezenas de neurônios e algumas camadas intermediárias, uma DNN pode facilmente superar esse número, com centenas ou mesmo milhares de neurônios em cada camada e dezenas de camadas intermediárias.

Embora a maioria dos problemas abordados nesta pesquisa apresente características lineares e possa ser resolvida utilizando métodos lineares, como MLR, é importante considerar a possibilidade da existência de relações não-lineares subjacentes que possam afetar os resultados. A aplicação do DNN pode ajudar a identificar essas potenciais relações não-lineares entre as variáveis de entrada. Isso se torna especialmente relevante quando existem interações complexas ou dependências não-lineares nos dados. Portanto, se os resultados obtidos por meio do DNN forem significativamente melhores em comparação com as técnicas lineares de ML, é possível inferir a existência de relações não-lineares complexas no conjunto de dados. Utilizar DNN requer alta capacidade computacional e seu uso é geralmente feito por meio de conexões com servidores externos que disponham de mais recursos computacionais (KIM; DEKA, 2021).

2.3.5 Regressão por vetores de suporte

Máquina de vetor de suporte (*Support Vector Machine* (SVM)), é um método supervisionado de ML utilizado em problemas de classificação. Este método utiliza um hiperplano para separar conjuntos de dados com base nas características identificadas durante a fase de treinamento. Nessa etapa, SVM irá identificar um hiperplano ótimo reproduzível que maximize as margens (ou distâncias) entre os vetores de suporte das classes (PISNER; SCHNYER, 2020). Os vetores de suporte são as observações que estão mais próximas do hiperplano, conforme mostrado em Figura 2.7(a). Esta figura ilustra um hiperplano que separa os vetores de suporte em duas classes, A e B. As características extraídas durante o treinamento são usadas para criar um modelo de decisão que pode classificar dados ainda não observados.

Regressão por vetor de suporte (*Support Vector Regression* (SVR)) é uma extensão de SVM para problemas de regressão. SVR é utilizado para realizar estimativas de



(a) SVM (b) SVR

Figura 2.7 Modelos de hiperplano para (a) SVM e (b) SVR. Em (a) é apresentado um hiperplano que separa maximamente os vetores de suporte correspondentes a cada uma das duas classes a serem previstas. Em (b) é apresentado um modelo de hiperplano onde a função de perda ϵ aproxima todos os pontos do conjunto de dados. Imagem adaptada de Zhang e O'Donnell (2020).

funções contínuas e prever valores numéricos. SVR aplica a ideia básica de SVM, ou seja, o processo é baseado na utilização de um hiperplano definido por alguns vetores de suporte. Mas, em vez de encontrar um hiperplano para separar e classificar um conjunto de dados, SVR introduz uma função de perda para calcular um hiperplano de modo que os valores de respostas previstos no conjunto de dados tenham, no máximo, um desvio de ϵ em relação aos dados de resposta observados (ZHANG; O'DONNELL, 2020). Como pode ser visto em Figura 2.7(b), o parâmetro ϵ define um tubo que deve conter a maior parte dos dados. Os valores absolutos com erros menores que ϵ não são penalizados, tanto acima quanto abaixo das estimativas. Valores fora dessa faixa são penalizados igualmente. O objetivo é encontrar o tubo que melhor aproxima a função contínua, equilibrando a complexidade do modelo e o erro de previsão (AWAD; KHANNA, 2015).

2.4 CONSIDERAÇÕES FINAIS

Neste capítulo, foram abordados os conceitos relacionados a RTS e análise temporal. Além disso, foi feita uma breve introdução ao ML, na qual foram descritos alguns dos métodos dessa área. Esses métodos são utilizados nas abordagens apresentadas nos capítulos 4 e 5. As configurações específicas e as bibliotecas utilizadas para cada método são detalhadas nos respectivos capítulos.

Capítulo 3

REVISÃO DE LITERATURA

Este capítulo apresenta uma revisão da literatura sobre estudos que investigam a análise temporal em RTS. A maioria dos trabalhos discutidos está relacionada à aplicação de MBPTA com EVT, exceto os apresentados nas seções 3.4 e 3.5, que foram inseridos por utilizarem estratégias semelhantes às desta tese, embora com objetivos diferentes. O capítulo está organizado em seis seções, cada uma dedicada a uma linha de pesquisa específica. Alguns estudos podem se enquadrar em mais de uma categoria. Nesses casos, foi selecionada a categoria que melhor reflete a principal contribuição de cada pesquisa.

3.1 EVT PARA ESTIMATIVAS PWCET

O primeiro trabalho que se tem conhecimento que emprega modelagem estatística utilizando EVT para estimar o pWCET por meio de dados observados é o estudo de Burns e Edgar (2000). Os autores mostram que, utilizando o teorema do limite central, para uma amostra independente e identicamente distribuida (i.i.d) proveniente de uma certa população com média μ e variância σ^2, a distribuição da média amostral se aproxima de uma distribuição normal com média μ e variância σ^2. Quanto maior a amostra, mais precisa se torna a aproximação. Como o foco não está na média da distribuição, mas sim nos valores extremos, esses máximos podem ser modelados pelas distribuições dos valores extremos. A distribuição de Gumbel é empregada, cujos parâmetros são estimados utilizando o método da máxima verossimilhança. Não é aplicada técnica para obtenção do conjunto de máximos como *Block Maxima* (BM) e *Peaks-over Threshold* (PoT), e EVT é aplicada diretamente aos dados brutos de medição. A abordagem proposta é aplicada ao programa BSort, executado em uma plataforma com processador Pentium II com dois níveis de cache. Este trabalho serviu como base teórica para as pesquisas posteriores envolvendo EVT nesse domínio.

O primeiro estudo a aplicar MBPTA em programas com múltiplos caminhos é o de Cucu-Grosjean et al. (2012). O trabalho detalha cada etapa de aplicação de EVT, incluindo soluções de adaptação dos dados. Dentre os requisitos apresentados, destacam-se os requerimentos para a plataforma de hardware, a necessidade dos dados serem i.i.d, e

os questionamentos de utilizar uma distribuição contínua na modelagem de dados discretos. Para garantir que as medições atendam aos requisitos de i.i.d, o estado do hardware é redefinido a cada execução. Posteriormente, são aplicados testes estatísticos como o teste de Kolmogorov-Smirnov (FELLER, 1968) e de aleatoriedade (BRADLEY, 1968) para verificar se os requisitos i.i.d são atendidos. Também é apresentada uma abordagem para avaliar o número mínimo de observações necessárias. A distribuição *Extreme Values* (EV) empregada é a distribuição Gumbel, cujos parâmetros são ajustados por meio do teste de calda exponencial (DIEBOLT; GARRIDO; GIRARD, 2007). A seleção dos valores máximos é realizada utilizando o método BM, e o ajuste é feito através do gráfico QQ-plot. Os autores comparam os resultados das estimativas pWCET aplicando SPTA com os resultados utilizando metodologia proposta, mostrando melhorias de até 15%.

O estudo conduzido por Kosmidis et al. (2013b) analisa as implicações de realizar estimativas pWCET utilizando MBPTA em arquiteturas que incorporam recursos de *buffer*. *Buffers* são componentes comumente encontrados em arquiteturas e processadores modernos, sendo responsáveis por armazenar informações de maneira temporária. Os autores demonstram que os *buffers* podem ser empregados conforme os requisitos de MBPTA sem necessidade de alterações, diferentemente de outros recursos de hardware, como a memória cache, que necessitam de randomização temporal. Adicionalmente, são apresentados os conceitos de *Jitter*. *Jitter* é a diferença entre as menores e maiores latências de um recurso de hardware. Os autores classificam as fontes de *Jitter* em seis categorias distintas, baseadas na dependência, no histórico de execuções, ou na natureza probabilística/determinística. Essa classificação pode ser estendida a outros recursos de hardware para avaliar a compatibilidade com MBPTA.

Em Abella et al. (2015) é feito um experimento que aplica EVT para realizar estimativas pWCET em programas de caminho único presentes em sistemas de segurança automotiva, que utilizam arquiteturas multinúcleo. No total, são analisados 16 conjuntos de dados que medem diferentes programas. Os autores empregam o método de PoT para a seleção dos valores de máximos, com o *threshold* escolhido por meio do método CV-Plot (CASTILLO; DAOUDI; LOCKHART, 2014). A distribuição exponencial é usada como modelo com os parâmetros estimados através do método de máxima verossimilhança.

No estudo de Davis e Cucu-Grosjean (2019), os autores realizam um *survey* das técnicas de análise temporal aplicadas em RTS. São examinados os principais resultados de pesquisas nessa área durante o período de 2000 a 2018. A fim de proporcionar uma compreensão mais acessível ao leitor, é fornecida uma base teórica sobre tópicos relevantes nesse contexto. Os trabalhos são categorizados em diferentes métodos, incluindo STA, MBTA, abordagens híbridas, SPTA e MBPTA. Em cada estudo revisado, uma análise crítica das perspectivas dos autores é apresentada. No final, os autores listam alguns questionamentos ainda não respondidos e sugerem possíveis direções para futuras pesquisas.

Uma pesquisa semelhante ao estudo de Davis e Cucu-Grosjean (2019) é realizada por Cazorla et al. (2019). Os autores apresentam o estado da arte de PTA, abordando desde os fundamentos teóricos até a análise e avaliações em aplicações industriais. Além disso, o estudo identifica abordagens que possuem diferenças significativas, mas são agrupadas como se pertencessem a uma única categoria, o que pode levar a uma compreensão incor-

reta por parte do leitor. Ao concluir o levantamento, são indicadas possíveis direções para futuros estudos. Embora compartilhem objetivos semelhantes, é importante considerar tanto o trabalho de Cazorla et al. (2019) quanto o de Davis e Cucu-Grosjean (2019) ao buscar compreender o panorama atual da PTA em RTS.

Em Ferrandez et al. (2023), os autores apresentam a primeira implementação de uma estratégia de randomização de software aplicada a programas executados em GPU, possibilitando a aplicação de EVT para estimativas de pWCET. Até então, a randomização era restrita a aplicações em CPU e, por ser realizada em nível de compilador, a abordagem era dificultada em GPUs devido ao uso de linguagens proprietárias. O estudo foi conduzido na plataforma NVIDIA Xavier, composta por 8 CPUs com três níveis de cache, uma GPU com 8 *streaming multiprocessors* e 32 GB de memória RAM. O caso de uso consistiu em uma aplicação de detecção de pedestres composta por três tarefas principais: aquisição de imagem, processamento e detecção. Além do tempo de execução, o estudo também estendeu a análise para estimativas do pior caso de consumo de energia da aplicação. Os resultados demonstraram que a abordagem adotada proporciona melhorias de até 60% em relação aos limites teóricos de contenção (número de núcleos) e consumo de energia.

Em Bozhko et al. (2023), os autores propõem uma definição rigorosa e axiomática do conceito de pWCET, visando fortalecer suas bases matemáticas. Os autores argumentam que o pWCET serve como uma abstração de modelo que permite a análise de RTS, facilitando a utilização de técnicas da teoria da probabilidade. O estudo defende que a formalização proposta do pWCET é necessária para evitar interpretações incorretas e garantir que as propriedades desejadas de independência probabilística sejam respeitadas. Os autores discutem a relação entre a distribuição pWCET e a distribuição dos tempos probabilísticos verdadeira, destacando a necessidade de propriedades matemáticas para que a análise baseada em pWCET seja válida. Os autores validaram as definições e provas utilizando o verificador de provas Coq.

3.2 REQUISITOS PARA APLICAÇÃO DE EVT

Os trabalhos anteriores a 2009 aplicam EVT diretamente aos dados brutos de tempo de execução para realizar estimativas pWCET. Em Hansen, Hissam e Moreno (2009), os autores demonstram que essa prática é incorreta. Nem todos os tempos de execução observados representam os máximos, e as distribuições de valores extremos (*Extreme Values* - EV) são modelos que descrevem os valores de máximos (ou mínimos). Portanto, não é possível obter um modelo EV adequado ao utilizar os dados brutos. Para solucionar essa questão, os autores adotam, pela primeira vez, o método de BM para selecionar uma amostra de máximo a partir dos dados brutos. Em todo o estudo, é utilizado a distribuição Gumbel, sustentado pelo argumento de que as distribuições de tempos de execução geralmente possuem caudas não pesadas na maioria das amostras, o que faz convergir para esse modelo EV. A validade do estudo é verificada em um conjunto de 200 milhões de amostras de tempo de execução, distribuídas entre 154 tarefas periódicas originadas da indústria. É demonstrado que o método proposto apresenta estimativas pWCET mais precisas quando comparadas às estimativas pWCET derivadas do pior

cenário de tempo de execução observado.

Em Cazorla et al. (2013) são apresentados os requisitos para aplicação efetiva de EVT para realizar estimativas pWCET em programas executados em arquiteturas complexas. Os autores discutem os motivos e a necessidade de que as observações sejam i.i.d, ao mesmo tempo que precisam ser representativas do comportamento temporal real do programa no ambiente de execução da plataforma-alvo. A obtenção de observações i.i.d pode ser alcançada aplicando o processo aleatório nas entradas do programa ou aplicando os processos aleatórios no comportamento interno do hardware/software. O primeiro passo para alcançar a representatividade é compreender a população-alvo na fase de implantação, definindo corretamente a população máxima. Também é discutido sobre as diferentes fontes de variabilidade do tempo de execução, que incluem o vetor de entrada empregado e o estado do hardware/software. O artigo conclui destacando a importância de o analista compreender quais fatores no sistema afetam o tempo de execução em todas as instâncias dos programas sob análise.

O estudo realizado por Santinelli, Guet e Morio (2017) investiga as questões em aberto relacionadas à utilização de MBPTA nas estimativas pWCET, e propõe diretrizes visando à aplicação correta de EVT. Os autores apresentam as hipóteses de aplicação de EVT generalizada: estacionaridade, dependência de curto alcance e dependência de longo alcance. EVT generalizada é uma expansão da EVT clássica, adaptada para permitir a aplicação de EVT em estimativas pWCET. A viabilidade da aplicação é avaliada por meio da abordagem MBPTA-DIAGXTRM (GUET; SANTINELLI; MORIO, 2016), que verifica se as hipóteses anteriormente mencionadas são cumpridas. Além disso, MBPTA-DIAGXTRM seleciona a distribuição EV ideal, estima os seus parâmetros e avalia a qualidade das estimativas pWCET. As propostas são validadas com casos de teste utilizando RTS randomizados e não-randomizados no tempo. Os autores demonstram que, para alguns estudos de caso, a distribuição Gumbel não é confiável, e limitar as estimativas apenas a essa distribuição pode resultar em resultados imprecisos. O método empregado para extrair a amostra de valores máximo é o PoT.

O estudo de Reghenzani et al. (2019) apresenta testes estatísticos para avaliar as hipóteses de aplicação de EVT generalizada em medições de tempo de execução. O primeiro teste é o teste KPSS (KWIATKOWSKI et al., 1992), usado para avaliar a estacionaridade dos dados. Para analisar a independência de curto prazo (dependência de cache entre duas tarefas), os autores recomendam o teste BDS (BROOCK et al., 1996). Para avaliar a dependência de longo prazo, os autores sugerem o uso do índice de Hurst (QIAN; RASHEED, 2005), que pode ser determinado por meio da equação estatística R/S (HURST, 1951). Para avaliação do estudo, são utilizadas amostras de dados sintéticos e reais. As amostras sintéticas são empregadas para testar a capacidade de detecção de cada teste estatístico. As amostras reais são utilizadas para avaliar a previsibilidade probabilística em diferentes arquiteturas utilizando os testes estatísticos recomendados no trabalho. A principal preocupação dos autores está no fato de que alguns trabalhos do estado da arte não seguem os procedimentos necessários para garantir estimativas probabilísticas seguras.

3.3 APRIMORAMENTOS E ADAPTAÇÕES PARA ESTIMATIVAS MAIS CONFIÁVEIS

Em Abella et al. (2014), os autores introduzem uma metodologia chamada *Heart of Gold* (HoG) para identificar padrões de variação no tempo de execução. Segundo os autores, se esses padrões forem identificados, é possível restaurar a confiabilidade e, consequentemente, obter estimativas mais precisas através do MBPTA. O estudo considera como fonte de variação os eventos induzidos por hardware e software. Também são revisados os detalhes relacionados à aplicação de MBPTA, incluindo como os programas ativam eventos aleatórios de hardware. Além disso, é analisado o comportamento da variação temporal ao utilizar políticas de substituição de cache. Os autores ressaltam que a confiabilidade de MBPTA depende da obtenção de uma cobertura suficiente dos eventos aleatórios, apresentando algumas abordagens para alcançar essa cobertura. A distribuição EV empregada é a distribuição Gumbel. Para determinar o número mínimo de execuções necessárias, os autores adotam um método iterativo que interrompe a coleta de dados quando a convergência é alcançada.

Em Kosmidis et al. (2014) é apresentada uma abordagem baseada em MBPTA denominada *Path Upper-Bounding* (PUB), cujo objetivo é proporcionar estimativas pWCET mais confiáveis. Os autores partem da dificuldade em determinar os dados de entrada que levam ao pior tempo de execução de um programa. A abordagem PUB desenvolve uma versão estendida do programa em análise, incorporando instruções ao código original, como adições de números inteiros ou de ponto flutuante, multiplicação, entre outras. As modificações no código são baseadas na análise das propriedades das memórias cache com temporização aleatória. O objetivo é que o tempo de execução de qualquer trajeto do programa modificado estabeleça um limite superior para o tempo de execução do programa original em qualquer trajeto. Conforme os autores, esse procedimento elimina a necessidade de empregar uma grande quantidade de vetores de entrada, que seriam necessários para estabelecer limites probabilísticos confiáveis para o programa original. Os resultados apresentados demonstram que a abordagem proposta aumenta as estimativas pWCET em média 8% quando considerado todos os programas analisados. A necessidade de possuir conhecimento do código-fonte para aplicar as modificações, contrariando as características da abordagem MBPTA, constitui uma desvantagem dessa abordagem.

O trabalho de Ziccardi et al. (2015) também propõe um método para reduzir a necessidade de obter cobertura completa de caminhos, a fim de obter estimativas pWCET mais precisas com MBPTA. O método, denominado de *Extended Path Coverage* (EPC), possibilita obter estimativas pWCET sem depender de uma cobertura completa de caminhos ou da disponibilidade dos dados de entrada que levam ao pior tempo de execução de um programa. A abordagem exige a realização de um conjunto de observações para cada bloco básico do código. Esse processo é realizado extraindo as informações de tempo de cada bloco básico das observações de ponta a ponta do programa analisado. O EPC combina os tempos de execução dos blocos básicos, resultando nos tempos de execução para todo o programa, inclusive para os caminhos de execução não percorridos durante as medições. Com os tempos de execução calculados, EVT é aplicada. Os resultados mostram um aumento médio de 12% nas estimativas pWCET em comparação com a

aplicação convencional de EVT. Também é realizada uma comparação com o método PUB ((KOSMIDIS et al., 2014)), onde os resultados são semelhantes. Uma vantagem do EPC é a ausência da necessidade de modificar o código-fonte original do programa analisado.

Em Lesage et al. (2015), os autores introduzem um framework para avaliar a qualidade das análises de temporização baseadas em medição. Diferentemente de estudos que avaliam a qualidade das estimativas pWCET por meio de EVT utilizando programas e arquiteturas reais, o framework apresentado usa tarefas sintéticas nas quais o WCET real é conhecido, a fim de verificar a qualidade dos resultados obtidos via EVT para esses programas. Ao todo são utilizadas 100 tarefas aleatórias, com duas amostras de 8.000 execuções cada uma. Quando se é utilizado amostras que cobrem todos os caminhos de execução, as estimativas fornecidas via EVT são coerentes com o WCET real das tarefas. Em contrapartida, quando são utilizadas amostras que não possuem boa cobertura dos caminhos, os resultados obtidos por EVT não apresentam confiabilidade. Os autores enfatizam que mesmo em tarefas simples, alcançar uma cobertura completa de caminhos de execução é complexo. Isso é um desafio que precisa ser abordado nas pesquisas.

Para aplicar EVT, além da necessidade da amostra ser i.i.d, também é preciso que os dados provenham de uma distribuição contínua. Geralmente, os dados de medição de tempo de execução possuem uma distribuição discreta. O estudo realizado por Lima e Bate (2017) propõe uma abordagem, denominada IESTA (*Indirect Estimation in Statistical Time Analysis*), que introduz uma aleatorização nos dados de medições. A aleatorização dilui a discretização dos dados e as possíveis relações de dependência entre as observações, aumentando a possibilidade de modelar os dados via EVT. Essa abordagem é avaliada por meio de dois estudos. O primeiro utiliza o programa *BSearch*, onde os dados têm uma natureza discreta. O segundo estudo se baseia em medições de um motor de aeronave real, onde não há aleatorização e as relações de dependência dificultam a análise estatística. Para determinar se, após a aplicação da abordagem, as medições estão em conformidade com a aplicação de EVT, é empregado o teste de Dietrich (DIETRICH; HüSLER, 2002). O grau de aleatoriedade adicionado as medições originais pode ser regulado por meio de um índice denominado no artigo de taxa de dispersão. Os resultados apresentados indicam que, após a aplicação do IESTA, as amostras de dados passam a estar conforme a aplicação de EVT.

3.4 APLICAÇÃO DE APRENDIZADO DE MÁQUINA NA ANÁLISE TEMPORAL

Considerando a importância de uma previsão inicial do WCET, mesmo que imprecisa, nas etapas iniciais do desenvolvimento de RTS, onde apenas o código-fonte do programa está disponível, Bonenfant et al. (2017) propõem um framework baseado em ML para realizar previsões do WCET analisando apenas o código em C. Os autores afirmam que a disponibilidade antecipada de informações na fase inicial da construção de RTS pode ajudar a melhorar a integração e a seleção dos componentes de hardware. Infelizmente, os resultados iniciais apresentados nesse trabalho ainda não são suficientes para sua aplicação prática. Os desafios enfrentados estão relacionados à complexidade do aprendizado pelos modelos de ML utilizados e à seleção dos atributos do código que precisam ser

considerados na análise.

Em Kumar (2021), é apresentada uma abordagem que também visa obter estimativas precoces de WCET. Utilizando um modelo de DNN, o objetivo da abordagem é prever o WCET com base no código-fonte, sem a necessidade de compilá-lo ou executá-lo na arquitetura de hardware. O modelo de previsão é desenvolvido com o framework PyTorch. A abordagem começa medindo o tempo de execução do programa e, em seguida, extrai características que são utilizadas como entrada para a DNN. Exemplos dessas características incluem o número de operações aritméticas e lógicas, o número de chamadas de funções, e o número de operações de carregamento e gravação. As características extraídas são associadas ao tempo de execução obtido, e a DNN é treinada para estimar o WCET. Como esperado, o WCET estimado pode não ser adequado como limite superior. No entanto, os resultados contribuem para o dimensionamento do sistema e a configuração do ambiente de hardware.

Em Amalou, Puaut e Muller (2021), é proposta uma técnica híbrida de estimativa de WCET, chamada WE-HML, que opera em código binário. A técnica divide o código em blocos e mede o tempo de execução de cada um. O WE-HML treina modelos de ML utilizando dados dessas medições e os utiliza para estimar o WCET de cada bloco. O WCET total do programa é então calculado com base nessas estimativas. A abordagem foi avaliada em um Raspberry Pi 3B utilizando 13 programas do *benchmark* TACLeBench (FALK et al., 2016). Os resultados mostraram que as estimativas de WCET realizadas com WE-HML foram superiores a todos os tempos de execução máximos observados nos programas. De qualquer forma, os autores sugerem que o WE-HML seja utilizado em RTS com requisitos de segurança intermediários, onde a estimativa de WCET é necessária, mas o não cumprimento do prazo pode ser tolerado em algumas situações.

Em Kumar (2022), é apresentado um framework que combina algoritmos genéticos e ML para identificar o tempo de execução de um programa. Os algoritmos genéticos são utilizados para determinar os dados de pior caso, que servem como entrada para o modelo de previsão baseado em ML. O modelo ML é responsável por prever o tempo de execução com base nas entradas fornecidas. A metodologia é avaliada por meio de diferentes estudos de caso. Os resultados indicam que o método proposto consegue gerar e prever dados de pior caso que se aproximam dos valores de WCET dos programas estudados.

3.5 MODELAGEM DE PROGRAMAS VIA EVENTOS DE HARDWARE

As unidades de monitoramento de desempenho (PMU) estão presentes na maioria dos processadores modernos. Elas permitem a análise dos estados de hardware, monitorando eventos como leituras e gravações em cache e memória principal, número de instruções executadas, ciclos de processador consumidos, entre outros. Cada fabricante de CPU especifica, em seus manuais técnicos, o número de registradores de PMU e os eventos que podem ser monitorados. Devido à grande quantidade de informações que as PMU fornecem, suas funcionalidades são amplamente utilizadas para otimizações de desempenho de hardware (AZIMI et al., 2009; MOLKA et al., 2017; DANIELSSON et al., 2021) e melhorias na qualidade de software (YILMAZ, 2010; MALONE; ZAHRAN; KARRI,

2011; TINETTI; MéNDEZ, 2014).

Em Lesage et al. (2017) é introduzido um framework que utiliza PMU para identificar os principais fatores de variabilidade no comportamento temporal de uma tarefa. A análise é realizada sem a necessidade de conhecer a plataforma. O framework visa identificar os eventos relacionados às variações no tempo de execução da tarefa, fornecendo fontes de interferências que podem ser exploradas para melhorar a previsibilidade do comportamento da tarefa ou do sistema em geral. Os dados são coletados executando a tarefa de interesse com tarefas concorrentes que causam interferências. Devido ao número limitado de contadores na PMU, os autores utilizam análise de componentes principais (*Principal Component Analysis* (PCA)) para explicar a estrutura de variabilidade dos dados através de combinações lineares de eventos, chamados de componentes principais. A avaliação do framework é realizada em uma placa AURIX. A tarefa é executada várias vezes com as mesmas entradas, mas diferentes eventos são monitorados em cada execução. Os resultados que se apresentam de forma idêntica (isto é, com a mesma quantidade de ciclos), mas com diferentes eventos monitorados, são combinados como se fossem provenientes de uma única execução. Os autores justificam que o erro resultante dessa combinação de eventos pode ser considerado ruído.

O trabalho de Griffin et al. (2017) é uma extensão do estudo realizado por Lesage et al. (2017). Nesta extensão, os autores utilizam os resultados obtidos pelo framework para construir um modelo utilizando técnicas de ML, que permite realizar previsões de interferências às quais o hardware está sujeito, com base nas ocorrências de eventos capturados durante a execução do programa analisado. O processo de identificação dos eventos é realizado de maneira semelhante ao trabalho anterior, e a análise também é executada em uma placa AURIX. Infelizmente, utilizar PCA requer que todos os eventos sejam medidos simultaneamente para que suas ocorrências possam ser relacionadas. Devido ao número limitado de registradores de PMU em plataformas de hardware mais complexas, isso não é possível, e agrupar eventos observados em execuções distintas pode levar a erros de modelagem, uma vez que é improvável que eles se refiram ao mesmo estado da máquina.

Em Pujol et al. (2024), os autores propõem uma metodologia para avaliar a confiabilidade das informações fornecidas em conjunto pelos contadores de eventos de hardware. A abordagem, denominada *Multiple HEM Validation (MHV)*, permite verificar se o comportamento coletivo dos contadores está de acordo com o esperado. A arquitetura utilizada nos experimentos é a NXP T1040, uma plataforma utilizada no setor de aviação. Devido à limitação no número de contadores disponíveis, os autores propõem etapas na abordagem para lidar com a variação nos resultados obtidos em diferentes execuções do mesmo experimento. A abordagem é avaliada com foco nos contadores relacionados à coerência de cache, sendo capaz de identificar contadores confiáveis e não confiáveis com base em seu comportamento observado.

3.6 DESAFIOS E DIVERGÊNCIAS EM MBPTA COM EVT

O estudo de Griffin e Burns (2010) investiga o impacto da falta de realismo e segurança nas estimativas derivadas da distribuição Gumbel, que se manifestam ao analisar as diferenças

entre a distribuição de Gumbel e as características de um RTS. A primeira diferença está ligada ao uso de uma função contínua para modelar o tempo de execução de um programa, que normalmente é discreto. A outra suposição refere-se à exigência de que, para aplicar EVT, os dados precisam ser i.i.d, o que nem sempre se aplica ao tempo de execução de programas. Os autores propõem e justificam algumas soluções para questionamentos anteriores, como alternativas para aproximar uma distribuição discreta de uma distribuição contínua e a aplicação de testes estatísticos para determinar se a suposição i.i.d se mantém nos dados.

Em Lima, Dias e Barros (2016), os autores analisam a viabilidade e eficácia da EVT para estimativa do pWCET. Entre as contribuições do trabalho, incluem-se: a dependência das estimativas em relação à distribuição dos dados de entrada; é demonstrado que restringir a distribuição EV apenas à Gumbel pode levar a estimativas imprecisas, sendo recomendável o uso da distribuição generalizada dos valores extremos (*Generalized Extreme Value* (GEV)); é comprovado que a randomização do hardware favorece a aplicabilidade da EVT, embora não forneça garantias de aplicação. Além disso, o estudo investiga a influência do tamanho da memória cache nas estimativas pWCET, salientando que tamanhos muito pequenos ou grandes podem dificultar a obtenção de estimativas confiáveis. Para verificar a aplicabilidade dos dados à EVT, é utilizado o teste de Dietrich (DIETRICH; HüSLER, 2002). Um dos estudos de caso do artigo usou o programa BSearch, no qual demonstrou que a aplicação da EVT não foi viável, independentemente do uso de plataformas com previsibilidade ou imprevisibilidade no tempo, mesmo ao considerar diferentes tamanhos de cache. O alto nível de discrepância apresentado pelas distribuições dos dados do BSearch, juntamente com seu conjunto limitado de instruções, pode ser uma das razões. Para concluir, os autores ressaltam suas preocupações sobre a coleta de dados durante a fase de análise, sugerindo que podem existir diferenças entre a execução das aplicações em um ambiente de teste e sua execução na prática. Adicionalmente, enfatizam a necessidade de novos estudos relacionados à randomização dos dados para garantir a analisabilidade via EVT.

No estudo de Guet, Santinelli e Morio (2017), os autores examinam tanto a necessidade das amostras de dados serem representativas em relação ao comportamento do sistema para possibilitar a aplicação de EVT quanto a qualidade das estimativas pWCET. São propostas diretrizes com o intuito de viabilizar que EVT e a abordagem MBPTA possam lidar com medições dependentes de tempos de execução e tarefas de múltiplos caminhos. As contribuições são validadas por meio de testes de casos de aplicações industriais, RTS de múltiplos caminhos e arquiteturas multinúcleos. Ao final da pesquisa, os autores enfatizam que as modificações nos dados, feitas para aprimorar a aplicação de EVT, podem comprometer a segurança das estimativas. As medições devem ser representativas e abranger todas as condições de execução, a fim de realizar estimativas pWCET mais precisas.

Em Gil et al. (2017), os autores fornecem uma visão geral do estado da arte para MBPTA e enumeram os principais desafios ainda não resolvidos para assegurar a aplicabilidade de EVT na obtenção de estimativas pWCET. Três desafios são identificados pelos autores. Primeiramente, é mencionada a dificuldade de adquirir uma amostra capaz de representar as situações extremas encontradas no ambiente real. O segundo desafio

é como realizar uma aplicação confiável de EVT, já que não há garantia de que um determinado conjunto de valores máximos possa ser descrito por uma distribuição EV. Os autores ressaltam que as estimativas pWCET são extremamente sensíveis a pequenas variações no método escolhido. O último desafio está relacionado com a interpretação dos resultados alcançados com EVT. O artigo é concluído reiterando a importância de considerar esses desafios em aberto nas pesquisas futuras.

O estudo realizado por Vasconcelos e Lima (2022) aplica EVT em uma plataforma multinúcleo Raspberry Pi 3B, considerando possíveis influências do sistema operacional e de programas executados simultaneamente. O estudo realiza a análise de 11 *benchmarks* utilizando dados coletados de diferentes cenários, variando com base nos recursos de acesso à rede (eth0 e Wi-Fi) e nos núcleos do processador utilizados para execução. Os resultados revelam que, apesar de EVT mostrar eficácia e potencial na obtenção de estimativas pWCET, resultados coerentes e satisfatórios nem sempre são alcançados, destacando desafios como definir tamanhos ótimos de amostra de dados e reconhecer as limitações do critério i.i.d na garantia da adequação das amostras, já que nem sempre se obtêm bons ajustes do modelo, mesmo quando esse critério é cumprido.

3.7 CONSIDERAÇÕES FINAIS

Este capítulo revisou os estudos sobre análise temporal em RTS, com foco na aplicação do MBPTA com EVT para estimativas de pWCET. Foram discutidas diferentes linhas de pesquisa, desde trabalhos relacionados a utilização de contadores de eventos de hardware (LESAGE et al., 2017; GRIFFIN et al., 2017; PUJOL et al., 2024), que mostram como as informações obtidas dos eventos podem ser aproveitadas, até desafios e limitações associadas a aplicação segura de EVT, tais como a necessidade de se obter amostras representativas (GRIFFIN; BURNS, 2010; LIMA; DIAS; BARROS, 2016; GUET; SANTINELLI; MORIO, 2017; GIL et al., 2017; VASCONCELOS; LIMA, 2022).

A Tabela 3.1 apresenta um resumo dos trabalhos relacionados que compartilham características com as abordagens propostas nesta tese. Nela, são detalhadas as contribuições e limitações de cada uma dos trabalhos listados. Em negrito, estão destacadas as abordagens propostas nos capítulos 4 e 5, juntamente com suas respectivas características. A abordagem descrita no Capítulo 4 visa fornecer informações adicionais no nível de hardware, que podem ser incorporadas às estratégias atuais para aprimorar o protocolo de medição e, assim, melhorar a representatividade das amostras. O Capítulo 5 apresenta uma abordagem completa de MBPTA, que orienta o analista desde a etapa de amostragem até a obtenção das estimativas de pWCET. O principal diferencial desta abordagem, em comparação com a abordagem convencional de MBPTA, é que ela considera nos dados amostrados, além do tempo de execução, o número de instruções executadas. Essa inclusão permite uma visualização mais clara dos caminhos de execução e possibilita aprimorar a amostra, visando uma maior representatividade.

Tabela 3.1 Trabalhos relacionados com estratégias semelhantes às abordagens propostas.

Seção	Trabalho	Contribuição	Limitação
Modelagem de programas via eventos de hardware	Lesage et al. (2017)	Framework para identificar eventos relacionados à variação do tempo de execução de uma tarefa	Combina eventos medidos em execuções diferentes como se fossem de uma única execução
	Griffin et al. (2017)	Constrói um modelo de ML para prever possíveis interferências com base na análise de eventos de hardware	Combina eventos medidos em execuções diferentes como se fossem de uma única execução
	Pujol et al. (2024)	Metodologia para avaliar a confiabilidade das informações fornecidas por contadores de eventos agrupados	Combina eventos medidos em execuções diferentes como se fossem de uma única execução
	Cap. 4	**Metodologia para selecionar um subconjunto de eventos mais relevantes para um determinado programa**	**Possíveis erros de modelagem devido à forma como os eventos são monitorados**
Aprimoramentos para estimativas mais confiáveis	Abella et al. (2014)	Metodologia para identificar padrões de variação no tempo de execução	Foco apenas na distribuição Gumbel e ausência de discussão sobre limitações dos contadores da PMU
	Kosmidis et al. (2014)	Metodologia para reduzir a quantidade de vetores de entrada	Alterações no código do programa para reduzir o número de observações
	Ziccardi et al. (2015)	Metodologia para reduzir a quantidade de vetores de entrada	Combinação de tempos de execução de blocos básicos que podem não ocorrer na prática
	Lima e Bate (2017)	Introduz aleatorização nos dados medidos para reduzir a discretização	Modificações diretas nos dados medidos
	Ferrandez et al. (2023)	Estratégia de randomização de software aplicada a programas executados em GPU	Limitações de reprodutibilidade devido ao uso de linguagem proprietária
	Cap. 5	**Utiliza *Deep Learning* para aprimorar as medições**	**Necessita que as medições sejam compostas por caminhos de execução homogêneos**

Apesar dos avanços apresentados pelos estudos revisados, ainda existem lacunas relacionadas à obtenção de amostras representativas e à interpretação dos dados coletados. Os estudos sugerem uma possível direção voltada à melhoria da precisão das estimativas de pWCET. Além disso, a incorporação de informações adicionais que caracterizem melhor os caminhos de execução surge como uma possível estratégia a ser considerada.

Capítulo

4

MODELANDO O COMPORTAMENTO DE EXECUÇÃO VIA EVENTOS DE HARDWARE

Este capítulo introduz uma abordagem que visa modelar o comportamento de execução de programas com base em eventos relacionados ao hardware. A elaboração deste capítulo é baseada no artigo anteriormente publicado, intitulado *'On the Selection of Relevant Hardware Events for Explaining Execution Time Behavior'* (ANDRADE et al., 2021), que detalha e desenvolve este estudo. O artigo foi apresentado no Simpósio Brasileiro em Engenharia de Sistemas de Computação (*Brazilian Symposium on Computing Systems Engineering* (SBESC)), em 2021. Uma versão estendida foi publicada no periódico *Springer Journal of Design Automation for Embedded Systems (DAEM)*, sob o título *'On the impact of hardware-related events on the execution of real-time programs'*. Ao longo deste capítulo, os conceitos, metodologias e resultados discutidos no estudo mencionado são detalhados.

4.1 CONTEXTUALIZAÇÃO E OBJETIVOS DO ESTUDO

A demanda para implementar RTS tem se direcionado para arquiteturas de hardware complexas de múltiplos núcleos (AKESSON et al., 2022). Como discutido nos capítulos anteriores, a adoção desse modelo de arquitetura traz desafios significativos quando se visa realizar uma análise temporal do sistema. Nesse contexto, MBPTA vem sendo tratada como uma solução para superar as dificuldades enfrentadas pelos métodos tradicionais, derivando limites superiores para os tempos de execução de programas com base em dados coletados a partir de medições (DAVIS; CUCU-GROSJEAN, 2019; CAZORLA et al., 2019). No entanto, os resultados obtidos com MBPTA ainda são questionados. (LIMA; DIAS; BARROS, 2016; GIL et al., 2017; VASCONCELOS; LIMA, 2022). Os questionamentos estão centrados na falta de evidências que validem se os dados coletados durante a fase de análise refletem realmente o cenário de pior caso no ambiente real de operação, e se esses dados podem ser adequadamente modelados por meio da teoria dos valores extremos (EVT), entre outras questões.

Cada programa carrega suas próprias características de execução. Alguns programas podem ser mais sensíveis às alterações da memória cache, enquanto outros podem ter os efeitos do *pipeline* e do *branch predictor* como fatores mais relevantes. Ao obter informações dos aspectos que mais afetam o programa examinado, os analistas podem aprimorar a qualidade de suas estimativas probabilísticas do tempo de execução do pior caso (pWCET) ao ajustar, de forma mais precisa, o protocolo de medição. Para atender a essa demanda, neste capítulo é apresentado um procedimento de análise por meio do qual os analistas conseguem entender melhor os efeitos dos componentes de hardware no comportamento de execução dos programas.

Especificamente, este estudo apresenta uma abordagem que visa modelar o tempo de execução dos programas por meio do monitoramento dos eventos de hardware. Esse monitoramento é realizado através da leitura das unidades de monitoramento de desempenho (PMU). A variável derivada pelo modelo é o *execution pace*, que representa a quantidade de ciclos do processador consumidos por instrução. A avaliação do estudo é realizada utilizando uma plataforma ARM Cortex-A53, onde são analisados 15 programas do *benchmark* Mälardalen (GUSTAFSSON et al., 2010). Devido à limitação do ARM Cortex-A53, que não possui registradores de PMU suficientes para monitorar todos os eventos simultaneamente, um método para a seleção dos eventos mais relevantes é proposto. A modelagem é realizada através da aplicação de técnicas de aprendizado de máquina (ML), detalhadas no Capítulo 2.

Em resumo, esse estudo apresenta as seguintes contribuições:

1. O conceito de *execution pace* é definido, que é usado para representar o comportamento de execução do programa analisado. O *execution pace* estabelece a conexão entre a ocorrência de eventos relacionados ao hardware e o comportamento de execução do programa;

2. É demonstrado empiricamente que é possível modelar o comportamento de execução de um programa como uma função de eventos relacionados ao hardware. Evidências que apoiam essa afirmação são apresentadas utilizando uma arquitetura multinúcleo. Para esse modelo de arquitetura, nem todos os eventos podem ser monitorados simultaneamente. A modelagem é realizada por meio de técnicas de ML;

3. Utilizando estratégias relativamente simples e funcionalidades disponíveis em hardware pronto para uso, é descrita uma abordagem para selecionar um subconjunto de eventos relacionados ao hardware capaz de explicar o *execution pace*.

A abordagem descrita neste capítulo difere dos trabalhos citados na Seção 3.5 do Capítulo 3 tanto na metodologia quanto no propósito de seleção de eventos. O foco do estudo apresentado aqui está na seleção dos eventos de hardware (não de componentes) que podem explicar o comportamento em tempo de execução dos programas analisados. O restante deste capítulo está estruturado da seguinte forma. A Seção 4.2 introduz alguns conceitos e define o problema abordado. A Seção 4.3 descreve a plataforma e os *benchmarks* utilizados nos experimentos. Um método para classificar eventos relevantes

relacionados ao hardware é apresentado na Seção 4.4. Os resultados experimentais para os modelos obtidos são fornecidos na Seção 4.5. As considerações finais são apresentadas na Seção 4.6.

4.2 DEFINIÇÕES PRINCIPAIS E PROBLEMA ABORDADO

No presente estudo, a análise da execução de um programa em uma plataforma de hardware segue um protocolo de medição. Os dados são obtidos ao executar o programa com diferentes entradas, sujeitas a execuções concorrentes que podem causar interferências.

Para cada execução, são obtidas sequências de ciclos do processador, quantidade de instruções executadas e os eventos relacionados ao hardware. O foco da análise está na relação entre os eventos de hardware e a variação dos tempos de execução, representada através do *execution pace*.

Para melhor compreensão, sendo $T(n)$ a função que representa o número de ciclos do processador necessário para executar n instruções, é possível considerar as seguintes propriedades:

1. $T(n)$ é uma função crescente em relação a n. Além disso, caminhos de execução mais longos tendem a se beneficiar mais de dispositivos de aceleração de hardware em comparação com caminhos mais curtos. Isso implica que o tempo médio para executar uma instrução em caminhos de execução mais longos tende a ser menor. O valor de $T(1)$ representa, portanto, um limite superior para o número de ciclos necessários para executar uma única instrução, sujeito a variações devido ao estado do hardware;

2. Embora caminhos de execução longos tendam a se beneficiar de dispositivos de hardware que diminuem o tempo médio para executar uma instrução, existe um limite. Portanto, $\lim_{n\to\infty} T(n)/n = A > 0$;

3. Devido à variação no estado do hardware e diferentes tipos de instruções, é improvável que $T(n)$ seja o mesmo para diferentes sequências de n instruções ou para diferentes execuções da mesma sequência.

Seguindo as propriedades acima, o número de ciclos do processador necessários para executar n instruções pode ser modelado aproximadamente pela função:

$$T(n) = A\,n + B\,, \tag{4.1}$$

onde $A > 0$ e $B > 0$ são variáveis aleatórias que representam manifestações do ambiente de execução, incluindo os efeitos da ocorrência de eventos relacionados ao hardware durante a execução de n instruções, bem como as diferenças nos tipos de instruções.

Os termos A e B não são mensuráveis e, portanto, Eq. (4.1) não pode ser diretamente verificada. Eq. (4.1) é tomada como exemplo somente para fins de apresentação, mas este estudo não se baseia nessa função para avaliar o modelo. O modelo não depende da função subjacente que fornece $T(n)$, uma vez que o que está sendo modelado é o *execution*

pace, que, ao assumir Eq. (4.1), é representado por $P(n)$. No caso em que $T(n)$ é dado por (4.1):

$$P(n) = A + B/n, \tag{4.2}$$

A utilização da Eq. (4.1) e da Eq. (4.2) permite observar alguns efeitos associados às propriedades mencionadas anteriormente. Por exemplo, Eq. (4.1) ilustra que o número n de instruções influencia na variabilidade de A (tempo necessário para executar uma instrução). Isso é confirmado na prática ao observar a execução de caminhos de execução curtos e longos. Esse efeito torna mais complexa a correlação entre eventos que ocorrem no nível de hardware e o tempo de execução. A normalização das medições por instrução executada ajuda a diminuir essa complexidade, justificando a utilização de $P(n)$ como uma variável de resposta.

Os valores de $P(n)$ são obtidos a partir de dados medidos ao dividir o número de ciclos observados pelo número n de instruções. Da mesma forma, o número de eventos contabilizados durante o monitoramento é também normalizado. Uma vez que A representa o número de ciclos necessários para executar uma instrução, quanto mais eventos relacionados ao hardware ocorrerem (como cache *miss*, interrupções do processador), maior tenderá a ser o valor de A. Além disso, é esperado que cada ocorrência desses eventos aumente o valor de A por um certo fator sujeito a variações. Considerando que os valores de n são suficientemente grandes, o efeito do termo B/n tende a ser pequeno.

Alguns aspectos importantes não são considerados na modelagem do *execution pace*, como:

- Diferença entre as instruções: $P(n)$ não é afetado somente pelos eventos de hardware. A variação em A e B também é causada pelo tipo de instrução que está sendo executada. No entanto, isso pode não ser relevante, uma vez que os programas usualmente não apresentam instruções com diferenças significativas no tempo de execução;

- Possibilidade de erros de medição: realizar o monitoramento dos eventos de hardware via chamadas de sistema do sistema operacional pode ocasionar erros de medição (ZAPARANUKS; JOVIC; HAUSWIRTH, 2009). Além disso, quando não é possível observar todos os eventos de hardware, erros adicionais podem ser introduzidos. Uma maior quantidade de contadores disponíveis na PMU pode auxiliar a reduzir esse impacto.

4.3 PLATAFORMA E *BENCHMARKS*

Nesta seção, são descritos os 15 programas estudados e detalhadas as características da plataforma ARM Cortex-A53, utilizada nos estudos deste capítulo e do Capítulo 5. O ARM Cortex-A53 é uma plataforma de baixo custo encontrada nos microcomputadores Raspberry Pi 3B. Essa plataforma inclui uma CPU com quatro núcleos de 1.2GHz com o chip Broadcom BCM2837 de 64 bits (ARM, 2016). A plataforma possui 1 GB de RAM, cache de dois níveis, *branch predictors* e um *pipeline* de 8 estágios. Cada núcleo possui

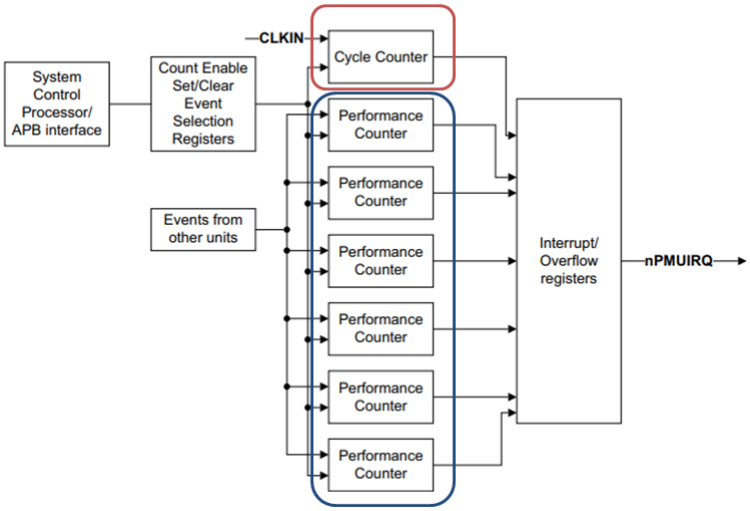


Figura 4.1 Diagrama da PMU do ARM Cortex-A53 destacando o registrador para monitorar a quantidade de ciclos (em vermelho) e os demais registradores (em azul). Imagem adaptada de ARM (2016).

dois caches L1 privados de 32KB, um para armazenar instruções e outro para os dados. A memória cache L2 é compartilhada entre os núcleos, possui um tamanho de 512KB e é utilizada tanto para armazenar instruções quanto para armazenar dados. A política de substituição de cache utilizada é a aleatória, o que significa que qualquer linha de cache pode ser substituída com a mesma probabilidade.

O sistema operacional usado é o Raspbian (FOUNDATION, 2018), versão 4.9.8, baseado no Debian. O monitoramento dos eventos de hardware é conduzido por meio da interface *Perf Event Open* (KERRISK, 2019), presente na maioria das distribuições baseadas em Linux. A plataforma possui PMU com sete registradores, sendo um deles destinado ao monitoramento do número de ciclos. Como Eq. (4.2) também requer a contagem de instruções, restam somente cinco registradores disponíveis para monitorar simultaneamente os demais eventos relacionados ao hardware e ao software. O diagrama de PMU do ARM Cortex-A53 pode ser visualizado em Figura 4.1.

Conforme o manual de referência (ARM, 2016), o ARM Cortex-A53 apresenta um total de 59 eventos relacionados ao hardware que podem ser monitorados. Durante os experimentos, foi constatado que alguns desses eventos possuíam ocorrências constantes (ou próximas disso) entre os cenários de execução configurados, tornando-os irrelevantes para o objetivo do estudo. Adicionalmente, existem eventos cujas ocorrências estão incluídas em outros eventos. Nesses casos, foi decidido considerar somente o evento mais geral. Por exemplo, as ocorrências dos eventos `BusAccessRead` (acesso de leitura no barramento) e `BusAccessWrite` (acesso de escrita no barramento) são observadas no evento `BusAccess` (acesso ao barramento). Assim, é possível desconsiderar os eventos `BusAccessRead` e `BusAccessWrite`. Após essa fase preliminar de filtragem, foram identificados 20 eventos a serem monitorados. Esses eventos estão listados na Tabela 4.1.

Tabela 4.1 Eventos a serem monitorados após filtragem preliminar

Nome	Cod. Hex.	Descrição
`L1IReload`	$0x01$	Recarregamento de cache de instruções L1
`L1DReload`	$0x03$	Recarregamento de cache de dados L1
`L1DAccess`	$0x04$	Acesso à cache de dados L1
`BPSpecWrong`	$0x10$	*Branch prediction* mal previsto executado especulativamente
`BPSpec`	$0x12$	*Branch prediction* executado especulativamente
`MemAccess`	$0x13$	Acesso à memória de dados
`L1IAccess`	$0x14$	Acesso à cache de instruções L1
`L1DWrite`	$0x15$	Escrita de dados na cache L1
`L2Access`	$0x16$	Acesso à cache L2
`L2Reload`	$0x17$	Recarregamento de cache L2
`L2WB`	$0x18$	*Write-back* de dados da cache L2
`BusAccess`	$0x19$	Acesso ao barramento
`ReqMemExt`	$0xC0$	Requisição de memória externa
`BPCondWrong`	$0xCC$	*Branch prediction* condicional mal previsto
`DPUEmpty`	$0xE0$	DPU vazia, não devido à *micro-TLB miss* ou *instruction miss*
`DPUEmptyIMiss`	$0xE1$	DPU vazia e *instruction cache miss* sendo processada
`DPUEmptyTLBMiss`	$0xE2$	DPU vazia e *instruction micro-TLB miss* sendo processada
`INLock`	$0xE4$	Existe *INLock* não devido a instruções de Float
`INLockLS`	$0xE5$	Existe *INLock* devido a instrução de carga/armazenamento
`PipeWaitLMiss`	$0xE7$	Existe uma espera na escrita do *pipeline* devido um *load miss*

Os últimos seis eventos estão relacionados com períodos em que a CPU aguarda devido a condições específicas. Por exemplo, o evento `DPUEmptyIMiss` é registrado sempre que ocorre um período de espera para processar um *miss* na cache L1 de instruções. Esse tipo de informação pode ser mais relevante do que os eventos relacionados diretamente com cache *miss* (`L1IReload`, `L1DReload` e `L2Reload`), uma vez que os atrasos causados pelas ocorrências de *miss* variam conforme o estado da máquina. Considerações semelhantes podem ser aplicadas aos outros cinco eventos. Mais adiante, será visto que alguns desses eventos são particularmente relevantes para representar o tempo de execução durante a modelagem.

Neste estudo, 15 programas do *benchmark* Mälardalen (GUSTAFSSON et al., 2010) são utilizados, cada um com características distintas, conforme apresentado na Tabela 4.2. Enquanto alguns programas, como o BSort, apresentam caminhos de execução mais longos, outros, como o BSearch, possuem caminhos mais curtos. Conforme discutido nas próximas seções, esses aspectos podem ocasionar efeitos colaterais nos resultados.

Na Listagem 4.1 é apresentado um exemplo de código que utiliza a biblioteca *Perf Event Open* para monitorar os eventos de hardware. O identificador `fd` é utilizado para representar o evento líder dentro do grupo de eventos monitorados. Este evento líder pode ser qualquer um dos eventos que estão sendo monitorados. O identificador `IOC_FLAG_GROUP` é usado para indicar que a contagem dos eventos está sendo realizada em grupos. Durante a execução da tarefa em análise, as ocorrências dos eventos são contadas e armazenadas nos registradores de PMU. Ao término do monitoramento, os dados coletados são registrados e armazenados em um arquivo externo. Um script foi elaborado para executar este procedimento de forma automática.

Tabela 4.2 Lista de programas. Valores médio, mínimo e máximo das instruções monitoradas utilizando entradas aleatórias.

		Número de instruções		
Programas	**Características**	**Avg**	**Min**	**Max**
BSort	Uso intenso de memória	20.028.954	19.564.374	20.421.727
ISort	Uso intenso de memória	6.272.023	5.887.219	6.640.230
Recursion	Código recursivo simples.	991.024	28	31.496.234
Cnt	Contagem de não negativos.	671.912	671.909	671.945
MSort	Uso intenso de memória	631.378	630.843	631.876
QSort	Uso intenso de memória	261.716	250.898	280.957
Matmult	Multiplicação de matrizes	162.292	112.426	217.735
Edn	Manipulação de vetores e array	133.737	130.737	136.686
Fft1	Cálculo com pontos flutuantes	18.351	3.490	34.760
Compress	Compreensão em buffer	6.685	4.382	9.231
Ludcmp	Elevado numero de cálculos de Float	3.206	168	8.397
Cover	Um loop contendo muitos switch/case.	1.391	349	2.202
Sqrt	Cálculo numérico simples	810	414	1.141
Fibcall	Fibonacci iterativo. Loop aninhado	559	559	564
BSearch	Busca Binária estruturada	260	63	289

```
1  //reinicia e habilita a contagem
2  ioctl(fd, PERF_EVENT_IOC_RESET, 0);
3  ioctl(fd, PERF_EVENT_IOC_ENABLE, 0);
4
5  //codigo analisado adicionado aqui
6
7  //Finaliza e desabilita contagem
8  ioctl(fd,EVENT_IOC_DISABLE,IOC_FLAG_GROUP);
```

Listing 4.1 Bloco de código utilizado para monitorar os eventos de hardware.

4.4 CLASSIFICAÇÃO DOS EVENTOS RELEVANTES

Conforme destacado na Seção 4.3, a arquitetura ARM Cortex-A53 não permite o monitoramento simultâneo de todos os eventos. Nesta seção é apresentado um procedimento que seleciona o conjunto mais relevante de eventos de hardware para cada programa analisado. A abordagem de seleção é fundamentada em estatísticas derivadas de experimentos realizados.

4.4.1 Configuração dos cenários e programas monitorados

Os experimentos abordados nesta seção têm como foco as potenciais interferências que podem afetar o programa em análise por meio de acessos concorrentes à memória. Além disso, é esperado que essa forma de interferência resulte na ocorrência de outros tipos de eventos de hardware, como acessos ao barramento, interrupções no *pipeline*, entre outros. A ocorrência desses eventos certamente contribui para a variabilidade do tempo de execução.

O programa utilizado para interferir na execução dos demais programas analisados é um gerador de interferência relacionado à cache, que foi utilizado em Bechtel (2019) para verificar potenciais problemas relacionados à segurança na memória cache. Conforme demonstrado na Listagem 4.2, o código consiste em um único loop que efetua atualizações nos elementos de um vetor. Isso é feito via incrementos do tamanho de uma linha de cache (64 bytes) na variável iteradora (i), até atingir o tamanho máximo da cache L2 da arquitetura ARM Cortex-A53 (512kB).

```
1 // men_size = 524.288 bytes
2 // LINE_SIZE = 64 bytes
3 for (int i = 0; i < men_size; i += LINE_SIZE){
4     ptr[i] = 0xff;
5 }
```

Listing 4.2 Código gerador de interferências CIG, utilizado durante o monitoramento dos eventos de hardware.

No presente estudo, o código gerador de interferência é denominado de *Cache-related Interference Generator* (CIG). As ocorrências dos eventos são monitoradas em dois cenários distintos:

- **Cenário Base**: o programa em análise é executado no núcleo 1 do processador, enquanto os outros núcleos permanecem dedicados exclusivamente à execução de tarefas do sistema operacional. É possível que as tarefas do sistema operacional causem interferência na execução do programa em análise;
- **Cenário Stress**: o programa em análise é executado no núcleo 1 do processador, enquanto cada um dos núcleos restantes (2, 3 e 4) executam, simultaneamente, uma instância do CIG.

Com base nos experimentos realizados nesta pesquisa, foi observado que o CIG gera um tráfego de leitura e escrita à memória bastante intenso. Isso provavelmente ocorre devido a uma alta taxa de *miss* na cache L2, bem como a outras fontes de interferência, como o tráfego no barramento e a contenção no acesso à memória. Como resultado, há um aumento significativo no tempo de execução da tarefa analisada quando é executada no cenário Stress, em comparação com o cenário Base.

Na etapa de seleção de eventos relevantes, os dados de entrada são fixados. Isso assegura a existência de um único caminho de execução, sendo que as diferenças nas ocorrências dos eventos entre os dois cenários decorrem exclusivamente devido à execução do CIG nos demais núcleos.

4.4.2 Seleção dos eventos

A abordagem para a seleção de eventos relevantes é baseada nas observações de como as distribuições dos eventos se alteram do cenário Base para o cenário Stress. Se as ocorrências de um determinado evento são semelhantes entre os cenários, é um indicativo de que a interferência causada pelo CIG não é capturada por esse evento. Por outro

lado, eventos nos quais a distribuição observada varia consideravelmente do cenário Base para o cenário Stress são bons candidatos para explicar a interferência induzida pelo CIG, que por sua vez pode estar associada a possíveis variações no tempo de execução do programa analisado. A abordagem de seleção de eventos segue esse raciocínio. Lembrando novamente que, neste estudo, o foco da interferência está relacionado às leituras e escritas na memória cache L2 e na memória principal.

Para identificar a lista de eventos relevantes, cada programa foi executado 10.000 vezes em cada cenário. Devido ao número limitado de registradores de PMU, foram necessários quatro séries de 10.000 execuções para abranger todos os 20 eventos (4×5) da Tabela 4.1. No total, foram realizadas 80.000 execuções para cada programa, sendo 40.000 para cada cenário (Base e Stress). Após a coleta de todos os conjuntos de dados, as distribuições dos eventos observadas são comparadas, analisando em que medida suas médias e variâncias se alteram de um cenário para outro.

Para comparar as distribuições dos eventos entre os cenários, são empregadas as estatísticas Z_0 e F_0. Essas quantidades são, respectivamente, estatísticas convencionais utilizadas para comparar médias e variâncias entre duas distribuições (DEVORE, 2011). Seja E_i um evento para o qual as distribuições amostrais correspondem a dois cenários de execução, $j = 1, 2$, com média $\overline{E}_{ij}$ e variância s^2_{ij}. Os valores de $j = 1, 2$ representam, respectivamente, os cenários Stress e Base. A estatística F_0 expressa a razão entre as variâncias dessas distribuições:

$$F_0 = \frac{s^2_{i1}}{s^2_{i2}} \tag{4.3}$$

enquanto Z_0 é definido como:

$$Z_0 = \frac{\overline{E}_{i1} - \overline{E}_{i2}}{\sqrt{\frac{s^2_{i1}}{m_1} + \frac{s^2_{i2}}{m_2}}} \tag{4.4}$$

em que m_1 e m_2 denotam o tamanho das amostras relativo aos cenários Stress e Base, respectivamente. Como m_1 e m_2 são iguais a 10.000, a Eq. (4.4) pode ser simplificada pela eliminação desses termos. Adicionalmente, como o interesse reside em observar o quão distante a média do evento no cenário Stress está da média do evento no cenário Base, é possível considerar somente o valor absoluto dessas diferenças, isto é:

$$Z_0 = \frac{|\overline{E}_{i1} - \overline{E}_{i2}|}{\sqrt{s^2_{i1} + s^2_{i2}}} \tag{4.5}$$

Eq. (4.3) e Eq. (4.5) são utilizadas para classificar os eventos em ordem decrescente de relevância. Quanto maior for o valor de F_0, maior será a diferença na variância entre os cenários Stress e Base para o evento E_i. Da mesma forma, quanto maior for o valor de Z_0, maior será a diferença na média do evento E_i entre os cenários.

Duas listas ordenadas em ordem decrescente são geradas, uma para Z_0 e outra para F_0. O objetivo é selecionar os cinco primeiros eventos de cada lista, evitando escolher

os eventos altamente correlacionados. Isto é, se o evento E_i aparece antes do evento E_j na classificação, mas eles são altamente correlacionados, o evento E_j não é selecionado e o próximo evento na lista de classificação é considerado. Evitar a seleção de eventos altamente correlacionados é importante para eliminar redundâncias de informações. Se dois eventos são altamente correlacionados, um deles é suficiente para explicar os efeitos na execução do programa analisado. Nesse estudo, foi escolhido o limite de correlação de 0,8. Portanto, se dois eventos possuem um índice de correlação maior ou igual a 0,8, o evento com menor valor de Z_0 (ou F_0) não é considerado na seleção. A Tabela 4.3 apresenta a seleção final de eventos para cada um dos 15 programas em estudo, com os eventos E_1 a E_5 listados em ordem decrescente de relevância, conforme Eq. (4.3) e Eq. (4.5).

Os eventos relacionados ao número de ciclos que a CPU espera por uma condição específica (ou seja, os eventos `DPUEmpty` a `PipeWaitLMiss`) são os que tendem a possuir as mudanças mais significativas do cenário Base para o cenário Stress. Esse comportamento é esperado, uma vez que a alta interferência no cenário Stress aumenta a competição, principalmente por espaço na cache L2, resultando em um maior número de *load miss* e, consequentemente, em mais ocorrências do evento `PipeWaitLMiss`. Da mesma forma, o evento `DPUEmpty` se torna mais frequente, uma vez que a DPU pode ficar ociosa enquanto aguarda a resolução das falhas de cache. Conforme pode ser observado na Tabela 4.3, o evento `PipeWaitLMiss` é selecionado para todos os programas sob ambos os critérios, Z_0 e F_0. Os eventos `DPUEmpty` e `DPUEmptyTLBMiss` estão presentes em todas as classificações baseadas em F_0. Muitos estudos relacionados a interferência da memória cache (VERA; LISPER; XUE, 2003; QUINONES et al., 2009; ALTMEYER; CUCU-GROSJEAN; DAVIS, 2015) são direcionados no pior cenário de execução da memória cache e não consideram informações mais detalhadas, como as mostradas por essas ocorrências de eventos. Os resultados na Tabela 4.3 indicam que essas informações são relevantes e precisam ser consideradas. Uma observação adicional nas listas de classificação é que, mesmo quando o evento `L2Reload` (cache *miss* L2) é considerado relevante, é comum encontrar também o evento `DPUEmptyIMiss`. Além disso, `DPUEmptyIMiss` é considerado relevante em um número maior de programas em comparação à `L2Reload`, conforme indicado na Tabela 4.3.

4.5 MODELANDO O TEMPO DE EXECUÇÃO VIA EVENTOS DE HARDWARE

Depois que os eventos são selecionados, eles são monitorados em cenários onde os programas recebem entradas geradas de forma aleatória. Isso resulta na ativação de vários caminhos de execução. Consequentemente, é avaliado se os eventos escolhidos conseguem representar adequadamente o comportamento da execução nesses cenários.

Nesta seção, são apresentados resultados que demonstram como os eventos relacionados ao hardware podem ser empregados para modelar o comportamento de execução de um programa. O que é modelado é, na realidade, o número médio de ciclos necessário para uma instrução ser executada (conforme apresentado na Seção 4.2). Através da utilização de dados coletados por meio de medições, essa variável resposta é expressa em termos da incidência de eventos ligados ao hardware (por instrução) por intermédio de

Tabela 4.3 Eventos selecionados E_1 a E_5 em ordem decrescente de relevância para Z_0 e F_0. As medições são realizadas utilizando entradas fixas.

		Eventos de Hardware				
Programas		E_1	E_2	E_3	E_4	E_5
BSort	Z_0	L2Reload	PipeWaitLMiss	DPUEmpty	BPSpec	DPUEmptyIMiss
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	L1DWrite	DPUEmpty	L2Reload
ISort	Z_0	L2Reload	PipeWaitLMiss	BusAccess	DPUEmpty	DPUEmptyIMiss
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	L1DWrite	DPUEmptyIMiss	DPUEmpty
Recursion	Z_0	L2Reload	L2WB	PipeWaitLMiss	DPUEmptyIMiss	DPUEmpty
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	DPUEmpty	DPUEmptyIMiss	L2Reload
Cnt	Z_0	L2Reload	L2WB	PipeWaitLMiss	DPUEmptyIMiss	DPUEmpty
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	DPUEmptyIMiss	DPUEmpty	BPSpec
MSort	Z_0	L2Reload	BusAccess	PipeWaitLMiss	DPUEmptyIMiss	L1DReload
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	DPUEmptyIMiss	DPUEmpty	L2Reload
Matmult	Z_0	BusAccess	L2WB	PipeWaitLMiss	DPUEmptyIMiss	DPUEmptyTLBMiss
	F_0	L1DWrite	PipeWaitLMiss	DPUEmptyTLBMiss	DPUEmptyIMiss	DPUEmpty
QSort	Z_0	L2Reload	PipeWaitLMiss	DPUEmptyIMiss	L2WB	DPUEmpty
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	L1DWrite	DPUEmptyIMiss	DPUEmpty
Edn	Z_0	L2Reload	L2WB	BusAccess	DPUEmptyIMiss	PipeWaitLMiss
	F_0	DPUEmptyTLBMiss	PipeWaitLMiss	DPUEmptyIMiss	DPUEmpty	L2Reload
Fft1	Z_0	PipeWaitLMiss	DPUEmptyIMiss	L2WB	DPUEmpty	INLockLS
	F_0	DPUEmptyTLBMiss	PipeWaitLMiss	DPUEmptyIMiss	DPUEmpty	ReqMemExt
Compress	Z_0	L2Reload	L2WB	PipeWaitLMiss	ReqMemExt	L1DReload
	F_0	DPUEmptyTLBMiss	PipeWaitLMiss	DPUEmpty	DPUEmptyIMiss	L2Access
Ludcmp	Z_0	L2Reload	PipeWaitLMiss	DPUEmpty	DPUEmptyTLBMiss	L1IAccess
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	DPUEmptyIMiss	DPUEmpty	BPCondWrong
Cover	Z_0	DPUEmptyIMiss	PipeWaitLMiss	L2WB	BPSpecWrong	L2Reload
	F_0	DPUEmptyTLBMiss	PipeWaitLMiss	DPUEmptyIMiss	DPUEmpty	BPSpec
Sqrt	Z_0	L2WB	L2Reload	DPUEmptyIMiss	DPUEmpty	PipeWaitLMiss
	F_0	DPUEmptyTLBMiss	PipeWaitLMiss	DPUEmpty	DPUEmptyIMiss	L1DWrite
Fibcall	Z_0	L2WB	PipeWaitLMiss	L2Reload	BPCondWrong	DPUEmptyIMiss
	F_0	DPUEmptyTLBMiss	PipeWaitLMiss	DPUEmpty	DPUEmptyIMiss	BPSpec
BSearch	Z_0	BusAccess	PipeWaitLMiss	L2Reload	L2WB	INLock
	F_0	PipeWaitLMiss	DPUEmptyTLBMiss	BusAccess	DPUEmpty	DPUEmptyIMiss

modelos de regressão. As ferramentas empregadas para ajustar os modelos de regressão são as mesmas descritas na Seção 2.3. Na Seção 4.5.1 são apresentadas as configurações adotadas para cada ferramenta utilizada. Os resultados são discutidos na Seção 4.5.2.

4.5.1 Configuração das ferramentas de modelagem

Nesta seção, são detalhadas as configurações adotadas para cada ferramenta empregada na modelagem de $P(n)$ em relação aos eventos monitorados. Mais informações sobre cada ferramenta pode ser visualizada na Seção 2.3. As implementações das ferramentas foram feitas utilizando a linguagem de programação R (R Core Team, 2020). As configurações adotadas foram as seguintes:

- ***Multiple Linear Regression* (MLR)**. Utilizados os procedimentos padrão disponíveis no R. Devido aos aspectos mencionados na Seção 4.2, não se espera que o resíduo da regressão (ϵ) siga uma distribuição normal com média nula e variância constante, como seria o caso habitual da regressão linear (MONTGOMERY; PECK; VINING, 2006);
- ***Random Forest* (RF)**. Implementação realizada através do pacote `randomForest` (LIAW; WIENER, 2002). Foram mantidos os valores padrão da biblioteca para todos os parâmetros de configuração, exceto o número de árvores, que foi selecionado com base na análise do MSE;
- ***Monotonic Neural Network* (MNN)**. Implementação utilizando o pacote `monmlp` (CANNON, 2022). A RNA foi configurada com uma única camada oculta composta por sete neurônios, usando a tangente hiperbólica como função de ativação (não linear). Embora o pacote permita flexibilidade para ajustar o número de neurônios e camadas, os resultados obtidos não apresentaram diferenças significativas em relação à quantidade escolhida. Os demais parâmetros foram mantidos em seus valores padrão conforme fornecidos pelo pacote;
- ***Deep Neural Network* (DNN)**. Implementação realizada através do pacote `h2o` (CANDEL; LEDELL, 2023). Apesar de explorar várias configurações da DNN, não foram observados resultados significativos. Portanto, os parâmetros da rede, incluindo a função de ativação (`Rectifier`) e o número de camadas ocultas (duas camadas ocultas com 100 neurônios cada), foram mantidos em seus valores padrão;
- ***Support Vector Regression* (SVR)**. Utilizado o pacote `e1071` (MEYER et al., 2023). Apenas dois parâmetros foram modificados no pacote. O parâmetro `type='eps-regression'` foi definido para indicar que o problema é de regressão e o parâmetro `kernel='linear'` foi definido para sinalizar a utilização de um kernel linear. Os demais parâmetros foram mantidos em seus valores padrão.

4.5.2 Estratégia de avaliação

Nesta seção, é detalhada a estratégia de avaliação da qualidade dos modelos para o *execution pace*. A estratégia se baseia na análise da discrepância entre as previsões do

Figura 4.2 Etapas de monitoramento, classificação, seleção e avaliação dos eventos de hardware mais relevantes.

modelo e as medições reais. Isso é feito utilizando a fórmula do erro relativo (Err_i para a i-ésima observação). A diferença geral é avaliada por meio da análise do erro médio, AvgE, definido como:

$$\mathrm{Err}_i = \frac{P_i - \hat{P}_i}{P_i} \quad \text{e} \quad \mathrm{AvgE} = \frac{1}{N}\sum_{1}^{N} |\mathrm{Err}_i| \,,$$

em que P_i é o *execution pace* observado, $\hat{P}_i$ é o *execution pace* previsto pelas ferramentas de modelagem e N é o tamanho da amostra.

Depois que os eventos relevantes são selecionados (Seção 4.4), é realizada a avaliação de suas capacidades de explicar o *execution pace*. Ao invés de manter as entradas do programa fixas, são realizadas medições usando entradas aleatórias. Portanto, os eventos selecionados com base em caminhos de execução únicos nos cenários Base e Stress são testados agora no contexto de múltiplos caminhos de execução. Isto é feito para avaliar a eficiência dos critérios de seleção de eventos também nesse contexto. A Figura 4.2 ilustra as etapas do processo de seleção e avaliação dos eventos.

4.5.3 Resultados obtidos

Nesta seção são apresentados os resultados obtidos para cada modelo baseado em ML, considerando as listas F_0 e Z_0 (Tabela 4.3). Os resultados são comparados com previsões utilizando os eventos `L1IReload`, `L1DReload`, `L2Reload`, `BPSpecWrong` e `BPSpec`, os quais são denotados nesta pesquisa como cache-branch (CB). Esses eventos foram selecionados devido à sua forte ligação com ocorrências de cache *miss* e *branch prediction*, que têm sido objetos de investigação em muitos estudos (ALTMEYER; CUCU-GROSJEAN; DAVIS, 2015; VERA; LISPER; XUE, 2003; QUINONES et al., 2009; KOSMIDIS et al., 2013; COLIN; PUAUT, 2000; BATE; REUTEMANN, 2004; BURGUIèRE; ROCHANGE, 2005).

As Tabelas 4.4 e 4.5 apresentam as diferenças entre o que é previsto pelos modelos baseados em ML e o que é observado. Valores de AvgE inferiores a 10^{-3} são geralmente indicativos de boas previsões. No entanto, em cenários em que o intervalo de variação do *execution pace* é pequeno, um AvgE na ordem de 10^{-3} pode não ser suficientemente baixo para caracterizar uma boa previsão. Um exemplo dessa situação pode ser visualizado na Figura 4.3(b), onde, apesar de o valor do AvgE estar na ordem de 10^{-3}, a previsão parece não acompanhar adequadamente as variações do *execution pace* observado. Os modelos baseados em ML que obtiveram o melhor resultado para cada seleção (CB, Z_0 e F_0) estão destacados em negrito. O MNN apresentou um AvgE menor em mais situações em ambas as tabelas. A Tabela 4.6 apresenta a média dos valores de AvgE calculados para cada modelo baseado em ML. Embora o MNN apresente o menor AvgE, seus valores são relativamente próximos aos apresentados por MLR e DNN. Esses três modelos de ML são os que exibem os valores mais baixos de AvgE.

É possível observar a partir das Tabelas 4.4, 4.5 e 4.6 que as listas de eventos selecionados de acordo com Z_0 e F_0 demonstram resultados superiores, com resíduos relativos menores em comparação com os obtidos para CB. Além disso, o critério de seleção F_0 tende a capturar melhor a relevância dos eventos, o que é deduzido a partir de suas métricas, sendo geralmente mais baixas em comparação com as do critério Z_0. No entanto, existem casos em que o critério Z_0 apresenta resultados melhores (por exemplo, no BSort).

Tabela 4.4 AvgE para os modelos de ML descritos na Seção 4.5.1, para os programas da Tabela 4.2, executados na plataforma Raspberry Pi 3B sob o contexto do cenário Base. Em cada seleção (CB, Z_0, e F_0), o modelo que apresenta menor AvgE é destacado.

Programas	Listas	MLR	RF	MNN	DNN	SVR
BSort	CB	$\mathbf{3{,}50 \times 10^{-5}}$	$7{,}06 \times 10^{-5}$	$3{,}41 \times 10^{-4}$	$1{,}11 \times 10^{-4}$	$4{,}07 \times 10^{-5}$
	Z_0	$\mathbf{3{,}41 \times 10^{-5}}$	$8{,}81 \times 10^{-5}$	$3{,}20 \times 10^{-4}$	$5{,}92 \times 10^{-5}$	$3{,}68 \times 10^{-5}$
	F_0	$3{,}59 \times 10^{-4}$	$\mathbf{2{,}01 \times 10^{-4}}$	$3{,}49 \times 10^{-4}$	$5{,}37 \times 10^{-4}$	$3{,}51 \times 10^{-4}$
ISort	CB	$5{,}51 \times 10^{-5}$	$5{,}66 \times 10^{-5}$	$5{,}50 \times 10^{-5}$	$8{,}22 \times 10^{-5}$	$\mathbf{4{,}07 \times 10^{-5}}$
	Z_0	$5{,}25 \times 10^{-6}$	$1{,}03 \times 10^{-5}$	$\mathbf{4{,}11 \times 10^{-6}}$	$8{,}19 \times 10^{-6}$	$5{,}28 \times 10^{-6}$
	F_0	$5{,}21 \times 10^{-6}$	$1{,}04 \times 10^{-5}$	$\mathbf{3{,}91 \times 10^{-6}}$	$8{,}49 \times 10^{-6}$	$5{,}23 \times 10^{-6}$
Recursion	CB	$8{,}52 \times 10^{-2}$	$\mathbf{8{,}00 \times 10^{-2}}$	$8{,}19 \times 10^{-2}$	$7{,}36 \times 10^{-2}$	$1{,}10 \times 10^{-1}$
	Z_0	$1{,}52 \times 10^{-2}$	$2{,}47 \times 10^{-2}$	$\mathbf{6{,}30 \times 10^{-3}}$	$1{,}11 \times 10^{-2}$	$1{,}46 \times 10^{-1}$
	F_0	$1{,}44 \times 10^{-2}$	$2{,}05 \times 10^{-2}$	$\mathbf{5{,}74 \times 10^{-3}}$	$4{,}18 \times 10^{-2}$	$2{,}90 \times 10^{-2}$
Cnt	CB	$1{,}38 \times 10^{-3}$	$1{,}40 \times 10^{-3}$	$1{,}38 \times 10^{-3}$	$1{,}39 \times 10^{-3}$	$\mathbf{1{,}29 \times 10^{-3}}$
	Z_0	$1{,}99 \times 10^{-4}$	$2{,}48 \times 10^{-4}$	$\mathbf{1{,}47 \times 10^{-4}}$	$1{,}84 \times 10^{-4}$	$1{,}75 \times 10^{-4}$
	F_0	$1{,}79 \times 10^{-4}$	$2{,}61 \times 10^{-4}$	$\mathbf{1{,}35 \times 10^{-4}}$	$4{,}50 \times 10^{-4}$	$1{,}63 \times 10^{-4}$
MSort	CB	$8{,}04 \times 10^{-4}$	$8{,}26 \times 10^{-4}$	$8{,}02 \times 10^{-4}$	$7{,}69 \times 10^{-4}$	$\mathbf{6{,}50 \times 10^{-4}}$
	Z_0	$5{,}77 \times 10^{-4}$	$5{,}69 \times 10^{-4}$	$\mathbf{5{,}75 \times 10^{-4}}$	$6{,}02 \times 10^{-4}$	$5{,}76 \times 10^{-4}$
	F_0	$2{,}51 \times 10^{-4}$	$2{,}21 \times 10^{-4}$	$\mathbf{1{,}59 \times 10^{-4}}$	$1{,}93 \times 10^{-4}$	$2{,}14 \times 10^{-4}$
Matmult	CB	$2{,}46 \times 10^{-3}$	$2{,}48 \times 10^{-3}$	$2{,}44 \times 10^{-3}$	$2{,}98 \times 10^{-3}$	$\mathbf{2{,}23 \times 10^{-3}}$
	Z_0	$5{,}01 \times 10^{-4}$	$5{,}67 \times 10^{-4}$	$\mathbf{4{,}20 \times 10^{-4}}$	$6{,}24 \times 10^{-4}$	$4{,}98 \times 10^{-4}$
	F_0	$7{,}79 \times 10^{-5}$	$3{,}71 \times 10^{-4}$	$\mathbf{5{,}91 \times 10^{-5}}$	$1{,}61 \times 10^{-4}$	$1{,}41 \times 10^{-4}$
QSort	CB	$2{,}23 \times 10^{-3}$	$2{,}23 \times 10^{-3}$	$2{,}24 \times 10^{-3}$	$2{,}15 \times 10^{-3}$	$\mathbf{1{,}97 \times 10^{-3}}$
	Z_0	$8{,}30 \times 10^{-4}$	$6{,}59 \times 10^{-4}$	$\mathbf{5{,}50 \times 10^{-4}}$	$6{,}05 \times 10^{-4}$	$7{,}59 \times 10^{-4}$
	F_0	$7{,}54 \times 10^{-4}$	$6{,}53 \times 10^{-4}$	$5{,}02 \times 10^{-4}$	$\mathbf{4{,}87 \times 10^{-4}}$	$6{,}65 \times 10^{-4}$
Edn	CB	$8{,}16 \times 10^{-3}$	$7{,}99 \times 10^{-3}$	$7{,}81 \times 10^{-3}$	$\mathbf{7{,}35 \times 10^{-3}}$	$7{,}72 \times 10^{-3}$
	Z_0	$6{,}15 \times 10^{-4}$	$1{,}48 \times 10^{-3}$	$\mathbf{5{,}52 \times 10^{-4}}$	$5{,}77 \times 10^{-4}$	$1{,}09 \times 10^{-3}$
	F_0	$2{,}49 \times 10^{-4}$	$1{,}23 \times 10^{-3}$	$\mathbf{2{,}35 \times 10^{-4}}$	$5{,}02 \times 10^{-4}$	$1{,}14 \times 10^{-3}$
Fft1	CB	$2{,}22 \times 10^{-2}$	$2{,}21 \times 10^{-2}$	$2{,}16 \times 10^{-2}$	$1{,}98 \times 10^{-2}$	$\mathbf{1{,}94 \times 10^{-2}}$
	Z_0	$7{,}57 \times 10^{-4}$	$3{,}59 \times 10^{-3}$	$\mathbf{7{,}55 \times 10^{-4}}$	$9{,}23 \times 10^{-4}$	$4{,}00 \times 10^{-3}$
	F_0	$8{,}27 \times 10^{-4}$	$4{,}94 \times 10^{-3}$	$\mathbf{7{,}91 \times 10^{-4}}$	$1{,}09 \times 10^{-3}$	$9{,}81 \times 10^{-4}$
Compress	CB	$5{,}65 \times 10^{-2}$	$5{,}49 \times 10^{-2}$	$5{,}71 \times 10^{-2}$	$\mathbf{5{,}47 \times 10^{-2}}$	$5{,}53 \times 10^{-2}$
	Z_0	$4{,}76 \times 10^{-2}$	$4{,}48 \times 10^{-2}$	$4{,}55 \times 10^{-2}$	$\mathbf{4{,}12 \times 10^{-2}}$	$4{,}63 \times 10^{-2}$
	F_0	$\mathbf{9{,}04 \times 10^{-3}}$	$1{,}39 \times 10^{-2}$	$9{,}06 \times 10^{-3}$	$7{,}73 \times 10^{-3}$	$1{,}07 \times 10^{-2}$
Ludcmp	CB	$9{,}85 \times 10^{-2}$	$1{,}00 \times 10^{-1}$	$9{,}63 \times 10^{-2}$	$1{,}02 \times 10^{-1}$	$\mathbf{9{,}38 \times 10^{-2}}$
	Z_0	$1{,}01 \times 10^{-1}$	$9{,}92 \times 10^{-2}$	$\mathbf{9{,}49 \times 10^{-2}}$	$1{,}12 \times 10^{-1}$	$9{,}89 \times 10^{-2}$
	F_0	$3{,}91 \times 10^{-3}$	$3{,}09 \times 10^{-2}$	$\mathbf{2{,}32 \times 10^{-3}}$	$5{,}16 \times 10^{-3}$	$2{,}36 \times 10^{-2}$
Cover	CB	$1{,}20 \times 10^{-1}$	$\mathbf{1{,}18 \times 10^{-1}}$	$1{,}22 \times 10^{-1}$	$1{,}24 \times 10^{-1}$	$1{,}19 \times 10^{-1}$
	Z_0	$\mathbf{7{,}40 \times 10^{-3}}$	$2{,}21 \times 10^{-2}$	$7{,}91 \times 10^{-3}$	$8{,}99 \times 10^{-3}$	$1{,}17 \times 10^{-2}$
	F_0	$8{,}58 \times 10^{-3}$	$1{,}84 \times 10^{-2}$	$8{,}17 \times 10^{-3}$	$\mathbf{7{,}93 \times 10^{-3}}$	$1{,}01 \times 10^{-2}$
Sqrt	CB	$7{,}99 \times 10^{-2}$	$8{,}41 \times 10^{-2}$	$8{,}02 \times 10^{-2}$	$7{,}75 \times 10^{-2}$	$\mathbf{7{,}48 \times 10^{-2}}$
	Z_0	$1{,}31 \times 10^{-2}$	$2{,}16 \times 10^{-2}$	$1{,}29 \times 10^{-2}$	$\mathbf{9{,}13 \times 10^{-3}}$	$1{,}48 \times 10^{-2}$
	F_0	$1{,}54 \times 10^{-2}$	$1{,}74 \times 10^{-2}$	$1{,}02 \times 10^{-2}$	$\mathbf{8{,}21 \times 10^{-3}}$	$1{,}48 \times 10^{-2}$
Fibcall	CB	$1{,}71 \times 10^{-1}$	$1{,}70 \times 10^{-1}$	$1{,}71 \times 10^{-1}$	$1{,}04 \times 10^{-1}$	$\mathbf{8{,}49 \times 10^{-2}}$
	Z_0	$3{,}36 \times 10^{-2}$	$5{,}45 \times 10^{-2}$	$3{,}38 \times 10^{-2}$	$3{,}47 \times 10^{-2}$	$\mathbf{3{,}23 \times 10^{-2}}$
	F_0	$4{,}87 \times 10^{-3}$	$4{,}04 \times 10^{-2}$	$\mathbf{4{,}44 \times 10^{-3}}$	$4{,}93 \times 10^{-3}$	$2{,}16 \times 10^{-2}$
BSearch	CB	$2{,}59 \times 10^{-1}$	$2{,}75 \times 10^{-1}$	$2{,}56 \times 10^{-1}$	$2{,}51 \times 10^{-1}$	$\mathbf{1{,}68 \times 10^{-1}}$
	Z_0	$2{,}00 \times 10^{-1}$	$2{,}02 \times 10^{-1}$	$2{,}02 \times 10^{-1}$	$1{,}90 \times 10^{-1}$	$\mathbf{1{,}39 \times 10^{-1}}$
	F_0	$6{,}35 \times 10^{-3}$	$4{,}03 \times 10^{-2}$	$\mathbf{5{,}27 \times 10^{-3}}$	$1{,}05 \times 10^{-2}$	$2{,}35 \times 10^{-2}$

Tabela 4.5 AvgE para os modelos de ML descritos na Seção 4.5.1, para os programas da Tabela 4.2, executados na plataforma Raspberry Pi 3B sob o contexto do cenário Stress. Em cada seleção (CB, Z_0, e F_0), o modelo que apresenta menor AvgE é destacado.

Programas	Listas	MLR	RF	MNN	DNN	SVR
BSort	CB	$1{,}97 \times 10^{-3}$	$1{,}97 \times 10^{-3}$	$1{,}99 \times 10^{-3}$	$\mathbf{1{,}94 \times 10^{-3}}$	$1{,}97 \times 10^{-3}$
	Z_0	$\mathbf{1{,}37 \times 10^{-4}}$	$4{,}13 \times 10^{-4}$	$3{,}57 \times 10^{-4}$	$1{,}76 \times 10^{-4}$	$1{,}76 \times 10^{-4}$
	F_0	$\mathbf{4{,}38 \times 10^{-4}}$	$6{,}04 \times 10^{-4}$	$4{,}37 \times 10^{-4}$	$4{,}73 \times 10^{-4}$	$4{,}36 \times 10^{-4}$
ISort	CB	$3{,}56 \times 10^{-3}$	$3{,}49 \times 10^{-3}$	$3{,}88 \times 10^{-3}$	$\mathbf{3{,}45 \times 10^{-3}}$	$3{,}50 \times 10^{-3}$
	Z_0	$\mathbf{1{,}00 \times 10^{-4}}$	$6{,}14 \times 10^{-4}$	$\mathbf{1{,}00 \times 10^{-4}}$	$1{,}72 \times 10^{-4}$	$3{,}58 \times 10^{-4}$
	F_0	$\mathbf{8{,}25 \times 10^{-5}}$	$6{,}56 \times 10^{-4}$	$8{,}34 \times 10^{-5}$	$2{,}67 \times 10^{-4}$	$2{,}94 \times 10^{-4}$
Recursion	CB	$\mathbf{3{,}12 \times 10^{-1}}$	$3{,}22 \times 10^{-1}$	$3{,}14 \times 10^{-1}$	$1{,}02 \times 10^{0}$	$3{,}01 \times 10^{0}$
	Z_0	$2{,}79 \times 10^{-1}$	$1{,}19 \times 10^{-1}$	$8{,}02 \times 10^{-2}$	$\mathbf{7{,}86 \times 10^{-2}}$	$2{,}68 \times 10^{0}$
	F_0	$3{,}53 \times 10^{-1}$	$1{,}25 \times 10^{-1}$	$\mathbf{5{,}63 \times 10^{-2}}$	$5{,}18 \times 10^{-1}$	$4{,}07 \times 10^{0}$
Cnt	CB	$1{,}18 \times 10^{-1}$	$\mathbf{1{,}15 \times 10^{-1}}$	$1{,}30 \times 10^{-1}$	$1{,}19 \times 10^{-1}$	$1{,}18 \times 10^{-1}$
	Z_0	$3{,}11 \times 10^{-2}$	$3{,}61 \times 10^{-2}$	$2{,}84 \times 10^{-2}$	$\mathbf{2{,}74 \times 10^{-2}}$	$3{,}03 \times 10^{-2}$
	F_0	$3{,}32 \times 10^{-2}$	$3{,}92 \times 10^{-2}$	$\mathbf{3{,}00 \times 10^{-2}}$	$3{,}22 \times 10^{-2}$	$3{,}24 \times 10^{-2}$
MSort	CB	$5{,}37 \times 10^{-2}$	$5{,}42 \times 10^{-2}$	$\mathbf{5{,}30 \times 10^{-2}}$	$5{,}32 \times 10^{-2}$	$5{,}37 \times 10^{-2}$
	Z_0	$\mathbf{2{,}43 \times 10^{-2}}$	$2{,}60 \times 10^{-2}$	$2{,}45 \times 10^{-2}$	$2{,}44 \times 10^{-2}$	$\mathbf{2{,}43 \times 10^{-2}}$
	F_0	$2{,}48 \times 10^{-2}$	$2{,}64 \times 10^{-2}$	$\mathbf{2{,}40 \times 10^{-2}}$	$2{,}48 \times 10^{-2}$	$2{,}47 \times 10^{-2}$
Matmult	CB	$1{,}11 \times 10^{-1}$	$1{,}12 \times 10^{-1}$	$1{,}31 \times 10^{-1}$	$\mathbf{1{,}10 \times 10^{-1}}$	$1{,}11 \times 10^{-1}$
	Z_0	$5{,}08 \times 10^{-2}$	$5{,}18 \times 10^{-2}$	$\mathbf{4{,}90 \times 10^{-2}}$	$\mathbf{4{,}90 \times 10^{-2}}$	$5{,}00 \times 10^{-2}$
	F_0	$5{,}80 \times 10^{-2}$	$5{,}47 \times 10^{-2}$	$\mathbf{4{,}97 \times 10^{-2}}$	$4{,}99 \times 10^{-2}$	$5{,}53 \times 10^{-2}$
QSort	CB	$6{,}29 \times 10^{-2}$	$6{,}39 \times 10^{-2}$	$6{,}34 \times 10^{-2}$	$6{,}49 \times 10^{-2}$	$\mathbf{6{,}14 \times 10^{-2}}$
	Z_0	$\mathbf{2{,}43 \times 10^{-3}}$	$1{,}00 \times 10^{-2}$	$2{,}44 \times 10^{-3}$	$3{,}51 \times 10^{-3}$	$5{,}70 \times 10^{-3}$
	F_0	$2{,}45 \times 10^{-3}$	$9{,}27 \times 10^{-3}$	$\mathbf{2{,}30 \times 10^{-3}}$	$2{,}59 \times 10^{-3}$	$7{,}14 \times 10^{-3}$
Edn	CB	$6{,}42 \times 10^{-2}$	$6{,}48 \times 10^{-2}$	$6{,}58 \times 10^{-2}$	$6{,}21 \times 10^{-2}$	$\mathbf{5{,}87 \times 10^{-2}}$
	Z_0	$2{,}77 \times 10^{-3}$	$8{,}53 \times 10^{-3}$	$\mathbf{2{,}27 \times 10^{-3}}$	$2{,}83 \times 10^{-3}$	$3{,}00 \times 10^{-3}$
	F_0	$2{,}99 \times 10^{-3}$	$8{,}49 \times 10^{-3}$	$2{,}36 \times 10^{-3}$	$\mathbf{2{,}18 \times 10^{-3}}$	$6{,}20 \times 10^{-3}$
Fft1	CB	$2{,}30 \times 10^{-1}$	$2{,}29 \times 10^{-1}$	$2{,}39 \times 10^{-1}$	$\mathbf{2{,}15 \times 10^{-1}}$	$\mathbf{2{,}15 \times 10^{-1}}$
	Z_0	$1{,}07 \times 10^{-2}$	$4{,}62 \times 10^{-2}$	$\mathbf{1{,}04 \times 10^{-2}}$	$4{,}15 \times 10^{-2}$	$2{,}61 \times 10^{-2}$
	F_0	$\mathbf{6{,}83 \times 10^{-3}}$	$4{,}49 \times 10^{-2}$	$7{,}21 \times 10^{-3}$	$1{,}01 \times 10^{-2}$	$3{,}00 \times 10^{-2}$
Compress	CB	$3{,}34 \times 10^{-1}$	$3{,}42 \times 10^{-1}$	$3{,}59 \times 10^{-1}$	$3{,}35 \times 10^{-1}$	$\mathbf{3{,}27 \times 10^{-1}}$
	Z_0	$1{,}77 \times 10^{-1}$	$1{,}81 \times 10^{-1}$	$2{,}36 \times 10^{-1}$	$\mathbf{1{,}71 \times 10^{-1}}$	$\mathbf{1{,}71 \times 10^{-1}}$
	F_0	$1{,}51 \times 10^{-1}$	$1{,}59 \times 10^{-1}$	$1{,}45 \times 10^{-1}$	$1{,}61 \times 10^{-1}$	$\mathbf{1{,}41 \times 10^{-1}}$
Ludcmp	CB	$3{,}61 \times 10^{-1}$	$\mathbf{3{,}39 \times 10^{-1}}$	$3{,}56 \times 10^{-1}$	$3{,}59 \times 10^{-1}$	$3{,}42 \times 10^{-1}$
	Z_0	$2{,}49 \times 10^{-1}$	$2{,}21 \times 10^{-1}$	$\mathbf{2{,}10 \times 10^{-1}}$	$2{,}54 \times 10^{-1}$	$2{,}93 \times 10^{-1}$
	F_0	$1{,}05 \times 10^{-1}$	$1{,}31 \times 10^{-1}$	$\mathbf{9{,}30 \times 10^{-2}}$	$1{,}11 \times 10^{-1}$	$2{,}56 \times 10^{-1}$
Cover	CB	$4{,}64 \times 10^{-1}$	$4{,}55 \times 10^{-1}$	$4{,}60 \times 10^{-1}$	$4{,}59 \times 10^{-1}$	$\mathbf{4{,}46 \times 10^{-1}}$
	Z_0	$3{,}27 \times 10^{-2}$	$6{,}21 \times 10^{-2}$	$\mathbf{2{,}98 \times 10^{-2}}$	$3{,}89 \times 10^{-2}$	$4{,}03 \times 10^{-2}$
	F_0	$2{,}92 \times 10^{-2}$	$6{,}06 \times 10^{-2}$	$\mathbf{2{,}82 \times 10^{-2}}$	$3{,}20 \times 10^{-2}$	$4{,}33 \times 10^{-2}$
Sqrt	CB	$4{,}82 \times 10^{-1}$	$4{,}98 \times 10^{-1}$	$4{,}84 \times 10^{-1}$	$4{,}55 \times 10^{-1}$	$\mathbf{4{,}03 \times 10^{-1}}$
	Z_0	$4{,}67 \times 10^{-2}$	$9{,}82 \times 10^{-2}$	$\mathbf{4{,}26 \times 10^{-2}}$	$5{,}91 \times 10^{-2}$	$7{,}66 \times 10^{-2}$
	F_0	$3{,}15 \times 10^{-2}$	$7{,}90 \times 10^{-2}$	$\mathbf{3{,}05 \times 10^{-2}}$	$4{,}97 \times 10^{-2}$	$9{,}58 \times 10^{-2}$
Fibcall	CB	$7{,}64 \times 10^{-1}$	$7{,}29 \times 10^{-1}$	$6{,}87 \times 10^{-1}$	$6{,}48 \times 10^{-1}$	$\mathbf{5{,}49 \times 10^{-1}}$
	Z_0	$1{,}51 \times 10^{-1}$	$2{,}84 \times 10^{-1}$	$9{,}56 \times 10^{-2}$	$\mathbf{9{,}45 \times 10^{-2}}$	$1{,}24 \times 10^{-1}$
	F_0	$1{,}07 \times 10^{-1}$	$1{,}47 \times 10^{-1}$	$3{,}59 \times 10^{-2}$	$\mathbf{3{,}45 \times 10^{-2}}$	$9{,}95 \times 10^{-2}$
BSearch	CB	$3{,}63 \times 10^{-1}$	$3{,}75 \times 10^{-1}$	$3{,}67 \times 10^{-1}$	$3{,}64 \times 10^{-1}$	$\mathbf{3{,}45 \times 10^{-1}}$
	Z_0	$5{,}67 \times 10^{-2}$	$7{,}38 \times 10^{-2}$	$5{,}59 \times 10^{-2}$	$7{,}94 \times 10^{-2}$	$\mathbf{5{,}29 \times 10^{-2}}$
	F_0	$2{,}52 \times 10^{-2}$	$5{,}37 \times 10^{-2}$	$\mathbf{9{,}56 \times 10^{-3}}$	$1{,}02 \times 10^{-2}$	$5{,}06 \times 10^{-2}$

Tabela 4.6 Média dos valores de AvgE para os modelos baseados em ML, mencionados na Seção 4.5.1, em relação aos programas listados na Tabela 4.2, quando executados na plataforma Raspberry Pi 3B.

	Cenário Base			**Cenário Stress**		
	CB	Z_0	F_0	CB	Z_0	F_0
MLR	$6{,}06 \times 10^{-2}$	$2{,}81 \times 10^{-2}$	$4{,}35 \times 10^{-3}$	$2{,}48 \times 10^{-1}$	$7{,}44 \times 10^{-2}$	$6{,}21 \times 10^{-2}$
RF	$6{,}13 \times 10^{-2}$	$3{,}17 \times 10^{-2}$	$1{,}26 \times 10^{-2}$	$2{,}47 \times 10^{-1}$	$8{,}81 \times 10^{-2}$	$6{,}26 \times 10^{-2}$
MNN	$6{,}01 \times 10^{-2}$	$2{,}71 \times 10^{-2}$	$3{,}17 \times 10^{-3}$	$2{,}48 \times 10^{-1}$	$5{,}79 \times 10^{-2}$	$3{,}43 \times 10^{-2}$
DNN	$5{,}49 \times 10^{-2}$	$2{,}74 \times 10^{-2}$	$5{,}99 \times 10^{-3}$	$2{,}85 \times 10^{-1}$	$6{,}17 \times 10^{-2}$	$6{,}94 \times 10^{-2}$
SVR	$4{,}93 \times 10^{-2}$	$3{,}32 \times 10^{-2}$	$9{,}15 \times 10^{-3}$	$4{,}03 \times 10^{-1}$	$2{,}39 \times 10^{-1}$	$3{,}27 \times 10^{-1}$

4.5.4 Avaliando a qualidade da previsão em relação ao caminho de execução

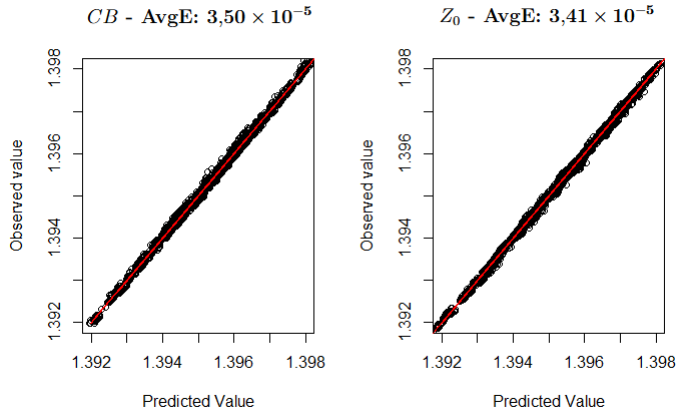
Nesta seção, as previsões feitas com base em Z_0 e F_0 são examinadas e novamente comparadas com as previsões de CB. O objetivo é investigar os impactos relacionados ao tamanho dos caminhos de execução. Para esta análise, o BSort foi escolhido por ter o caminho de execução mais longo, enquanto o BSearch foi selecionado por ter o caminho de execução mais curto. Para padronizar a comparação, somente a ferramenta MLR é considerada. MLR fornece modelos mais simples em comparação com os outros modelos baseados em ML. Além disso, MLR está entre os modelos que apresentaram melhores resultados, conforme a Tabela 4.6.

As Figuras 4.3 e 4.4 apresentam o *execution pace* previsto em relação ao observado para o BSort e o BSearch, respectivamente. É perceptível que Z_0 e F_0 são razoavelmente mais precisos, embora haja uma diminuição nas previsões para o BSearch no cenário Stress. Com exceção BSort no cenário Base, os modelos para o CB não explicam bem o *execution pace*. O número de instruções executadas pode afetar a qualidade das previsões, como será explicado a seguir.

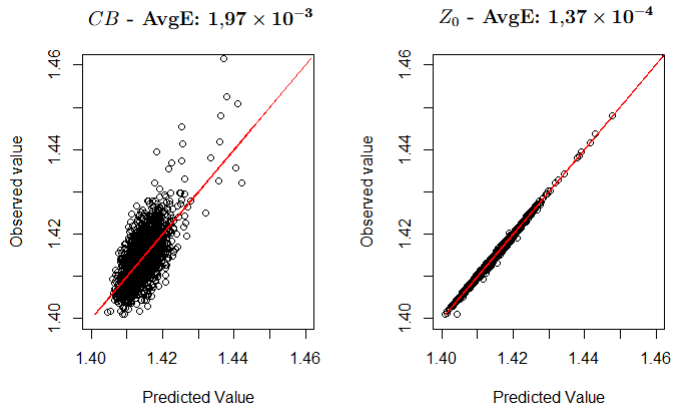
Quanto mais longos forem os caminhos de execução (grandes valores de n), menor tenderá a ser o *execution pace* (convergindo para A). Ou seja, para tais caminhos, os efeitos do termo B/n em Eq. (4.2) tendem a diminuir, justificando os valores baixos de AvgE, mesmo no caso do critério CB.

Por outro lado, para caminhos de execução curtos, o termo B/n pode resultar em alta variabilidade de $P(n)$, a qual tende a aumentar os erros de predição. Esse efeito é ilustrado na Figura 4.5, que mostra o resultado associado às previsões para o programa Recursion (Tabelas 4.4 e 4.5). Conforme pode ser visto na Figura 4.5(a), alguns valores do *execution pace* chegam a 1.500, o que está associado a caminhos curtos sendo acionados durante as medições. Nesse caso, a variabilidade devido a B aparece nos dados medidos, aumentando o erro de predição. O fato de que resíduos menores tendem a ser observados para caminhos de execução mais longos é evidenciado na Figura 4.5(b). A Figura 4.5(c) mostra a nuvem de pontos mais à direita que aparece na Figura 4.5(b). Como pode ser visto, o que parece ser um único ponto na Figura 4.5(b), ainda contém variabilidade. No entanto, os resíduos variam em um intervalo muito menor.

Os efeitos associados ao termo B/n e sua dependência em relação a n estão vinculados às características do programa analisado e à sua estrutura. No caso do programa

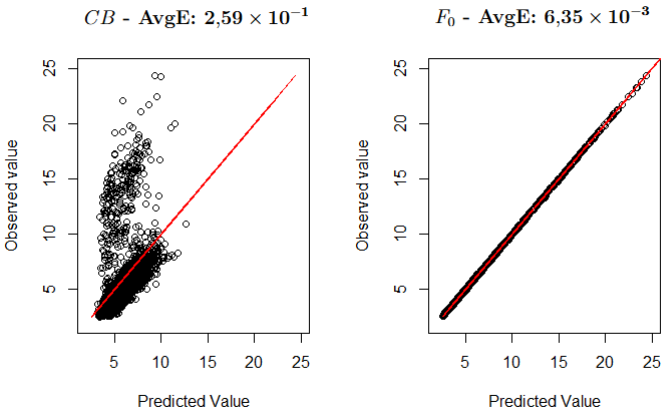


(a) Cenário Base

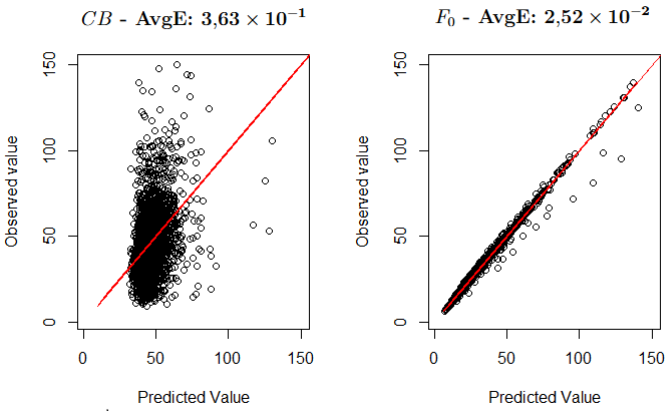


(b) Cenário Stress

Figura 4.3 *Execution pace* previsto (usando MLR) em comparação com o *execution pace* observado para o BSort. O critério de seleção baseado em Z_0 produziu menor AvgE.

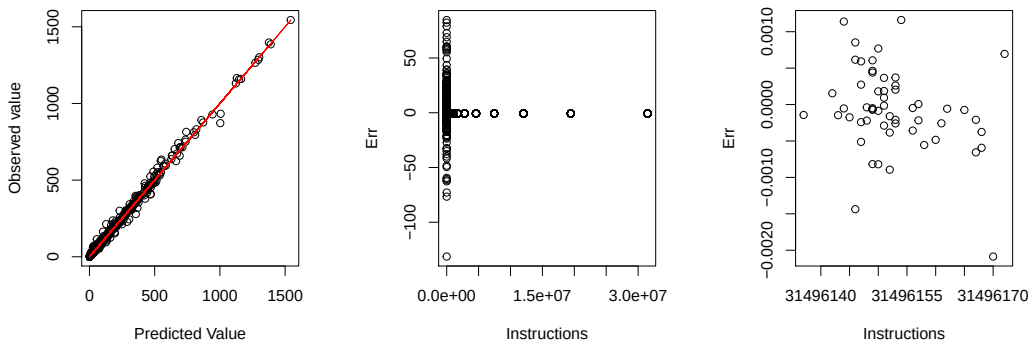


(a) Cenário Base



(b) Cenário Stress

Figura 4.4 *Execution pace* previsto (usando MLR) em comparação com o *execution pace* observado para o BSearch. O critério de seleção baseado em F_0 produziu menor AvgE.



(a) Estimativas do *execution pace*

(b) Resíduos x instruções

(c) Resíduos x instruções (caminhos mais longos)

Figura 4.5 Resultados para o programa Recursion: (a) valores estimados do *execution pace* utilizando MLR (AvgE: $2{,}79 \times 10^{-1}$) com os eventos selecionados por Z_0, sob o cenário Stress; (b) quanto maior a sequência de execução, menores são os resíduos; (c) a faixa de resíduos para os caminhos de execução mais longos.

Recursion, o valor de n deveria ser bem maior para compensar a variabilidade de B, enquanto para o programa BSearch, valores menores de n parecem ser suficientes, como indica a Figura 4.4. Isso se alinha com as características desses programas. Enquanto o BSearch é construído com um loop simples, o Recursion utiliza uma pilha de chamadas recursivas, o que pode resultar em um código menos otimizado para tirar vantagem da memória cache e do *branch predictor*, por exemplo.

Apesar das limitações observadas para alguns casos, como no programa Recursion, e das restrições intrínsecas impostas ao monitorar somente cinco eventos, o gráfico na Figura 4.5 ainda sugere que o modelo MLR estimado prevê de maneira razoável os valores observados. Portanto, considerar a estrutura do programa, bem como compreender os efeitos que os eventos relacionados ao hardware causam em sua execução, parece ser necessário para aumentar a precisão do MBTA, podendo fazer parte de trabalhos futuros.

4.5.5 Análise de sensibilidade na seleção de eventos

Visando verificar a sensibilidade da seleção de eventos, o programa ISort é agora analisado, considerando a lista de eventos indicada por F_0. A Figura 4.6 apresenta a qualidade correspondente das estimativas do *execution pace* (P) no ISort sob o cenário Stress, utilizando MLR como a ferramenta de previsão. Cada gráfico na figura mostra AvgE e a distância entre os valores observados e os valores previstos, considerando um subconjunto do conjunto de eventos selecionados por F_0 (`PipeWaitLMiss`, `DPUEmptyTLBMiss`, `L1DWrite`, `DPUEmptyIMiss` e `DPUEmpty`). Na Figura 4.6(a), todos os cinco eventos selecionados em F_0 são considerados nas previsões. Na Figura 4.6(b), o evento `DPUEmpty` é removido da seleção, e na Figura 4.6(c), os eventos `DPUEmptyIMiss` e `DPUEmpty` são removidos. É observado um aumento gradual em AvgE, chegando até a cinco vezes mais

alto ao comparar os valores das Figuras 4.6(a) e 4.6(c). Na Figura 4.6(d), um modelo de regressão linear simples é estimado considerando somente o evento `PipeWaitLMiss` como variável independente. Os resultados na Figura 4.6(d) mostram similaridades com os encontrados na Figura 4.6(c), sugerindo que os eventos `DPUEmptyTLBMiss` e `L1DWrite` têm baixa relevância para as estimativas de P (no ISort), e que os resultados apresentados nas Figuras 4.6(e) e 4.6(f) são influenciados principalmente pelos eventos `DPUEmptyIMiss` e `DPUEmpty`.

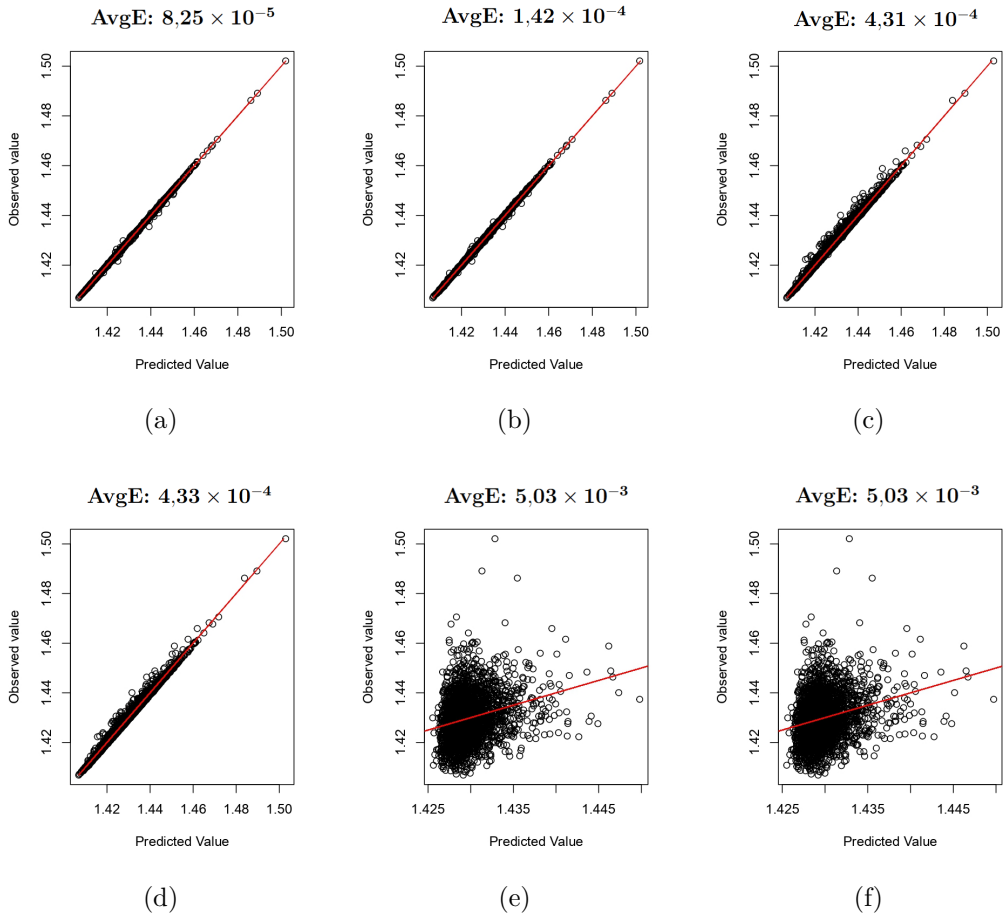


Figura 4.6 Valores observados e previstos do *execution pace* (P) ao utilizar MLR para o programa ISort, considerando os eventos selecionados por F_0 no cenário Stress: (a) utilizando todos os cinco eventos; (b) removendo `DPUEmpty`; (c) removendo `DPUEmptyIMiss` e `DPUEmpty`; (d) apenas `PipeWaitLMiss`; (e) removendo apenas `PipeWaitLMiss`; (f) removendo `PipeWaitLMiss` e `DPUEmptyTLBMiss`.

A Tabela 4.7 apresenta os coeficientes de correlação dos eventos em relação a P para o programa ISort no cenário Stress. O coeficiente de correlação é uma maneira simples de avaliar se existe uma relação linear entre variáveis. Embora `DPUEmptyTLBMiss` e `L1DWrite` apresentem mudanças razoáveis na variância (s^2) entre os cenários Base e Stress (lembrando da Eq. (4.5) em que F_0 representa a razão das variâncias em diferentes

Tabela 4.7 Coeficientes de correlação dos eventos de hardware selecionados por F_0 em relação a P para a amostra de dados do programa ISort no cenário Stress.

	PipeWaitLMiss	DPUEmptyTLBMiss	L1DWrite	DPUEmptyIMiss	DPUEmpty
Coef. Cor.	0,994	0,040	0,009	0,201	0,170

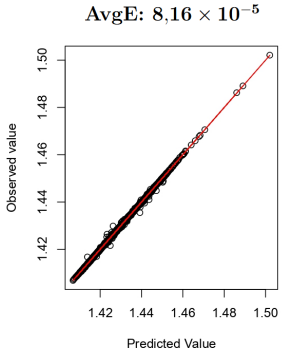


Figura 4.7 Valores observados e previstos de P utilizando MLR para o programa ISort, considerando apenas `PipeWaitLMiss`, `DPUEmptyIMiss` e `DPUEmpty` como variáveis independentes.

cenários), essas mudanças não correspondem às variações em P (coeficientes de correlação próximos a 0).

Ao analisar os valores absolutos do evento `DPUEmptyTLBMiss` na amostra de dados em estudo, é observado que em cerca de 97% desses valores estão abaixo de 100. No entanto, os 3% restantes apresentam valores extremamente altos e ocorrem isoladamente, sem estarem necessariamente relacionados a um aumento no número de ciclos. Essas ocorrências contribuem para mudanças significativas na variância (s^2), justificando a seleção do evento `DPUEmptyTLBMiss` no critério $F0$. Um comportamento semelhante é observado para o evento `L1DWrite`.

O Evento `PipeWaitLMiss` é acionado quando ocorre um impedimento no *pipeline* durante a etapa de escrita devido a uma falha na busca de dados (*load miss*), o que requer que a CPU aguarde a busca correta dos dados na memória. No caso específico do programa ISort, que realiza várias operações de leitura e escrita de dados, é esperado um número significativo de ocorrências do evento `PipeWaitLMiss`, especialmente no cenário Stress. Adicionalmente, também são esperados algumas interrupções da CPU devido ao processamento de um*miss* na cache L1 de instrução (evento `DPUEmptyIMiss`).

Considerando os resultados apresentados, a Figura 4.7 ilustra as estimativas de P utilizando somente os eventos `PipeWaitLMiss`, `DPUEmptyIMiss` e `DPUEmpty`. Os resultados são semelhantes aos mostrados na Figura 4.6(a), indicando que esses três eventos são suficientes para representar P com uma precisão razoável. Conforme demonstrado na Tabela 4.3, esses mesmos eventos também são selecionados pelo critério Z_0.

A limitação na quantidade de registradores de PMU pode restringir a capacidade de

capturar informações importantes sobre o comportamento do sistema, resultando consequentemente em estimativas menos precisas. No entanto, vale destacar que ter mais registradores de PMU não garante automaticamente estimativas mais confiáveis, a menos que o processo de seleção seja apropriado. É necessário identificar os eventos que causam impacto significativo nos resultados, já que alguns podem ser muito mais relevantes do que outros, como o `PipeWaitLMiss` para o programa ISort. É importante considerar cuidadosamente a relevância dos eventos para obter estimativas mais confiáveis.

4.6 CONSIDERAÇÕES FINAIS

Neste capítulo, foi descrita uma metodologia para analisar como o tempo de execução dos programas é influenciado por eventos relacionados ao hardware. Devido à variação nos tipos e quantidades de eventos disponíveis em diferentes arquiteturas, uma abordagem foi apresentada para selecionar os eventos relevantes no nível de hardware em relação aos efeitos no tempo de execução dos programas. Através da utilização de modelos baseados em aprendizado de máquina (ML), foi avaliada a qualidade desta abordagem, constatando-se que as ocorrências dos eventos selecionados podem ser utilizadas para prever o número esperado de ciclos para executar uma instrução (*execution pace*) com razoável precisão. Considerando todos os programas analisados e utilizando apenas MNN como ferramenta de modelagem, os eventos selecionados com F_0 resultaram em um menor AvgE de previsão, com $3,17 \times 10^{-3}$ no cenário Base e $3,43 \times 10^{-2}$ no cenário Stress.

MBTA é uma tendência atual, visto que a complexidade das plataformas de hardware modernas impede a aplicação de métodos tradicionais de análise temporal. A abordagem usual do MBTA estima limites de tempo de execução baseados somente no tempo de execução observado durante medições. Ocorrências de eventos no nível de hardware poderiam ser utilizadas como informações adicionais para derivar limites mais precisos. Pesquisas futuras podem se beneficiar dos resultados apresentados neste capítulo.

Capítulo

5

DBL-MBPTA: UMA ABORDAGEM MBPTA BASEADA EM CICLOS E INSTRUÇÕES

Conforme discutido nos capítulos anteriores, MBPTA se apresenta como uma alternativa promissora para a obtenção de estimativas probabilísticas do tempo de execução do pior caso (pWCET) de tarefas executadas em RTS equipados com arquiteturas complexas e múltiplos núcleos. No entanto, há questionamentos quanto à precisão e confiabilidade dos seus resultados.

Visando aprimorar a credibilidade dos resultados obtidos com MBPTA, este capítulo apresenta a *Distance Between Lines - MBPTA* (DBL-MBPTA), uma abordagem MBPTA *multipath* que se distingue da abordagem convencional. Enquanto a abordagem convencional concentra suas medições apenas no tempo de execução da tarefa, a DBL-MBPTA considera, além do tempo de execução, a quantidade de instruções executadas. Ao considerar o número de instruções, é possível realizar uma análise mais detalhada do comportamento do tempo de execução, o que permite aprimorar as medições e, consequentemente, obter estimativas mais precisas. A elaboração deste capítulo é baseada no artigo intitulado *'Drawing Lines for Measurement-Based Probabilistic Timing Analysis'* que foi publicado no 45^{th} *IEEE Real-Time Systems Symposium (RTSS), 2024, York, United Kingdom.* A abordagem é detalhada ao longo do capítulo, cobrindo cada etapa da implementação, os resultados experimentais e os aspectos críticos relacionados à sua aplicação.

5.1 CONTEXTUALIZAÇÃO E OBJETIVOS DO ESTUDO

A construção de RTS em arquiteturas modernas é desafiadora devido à presença de componentes que, embora projetados para aprimorar o desempenho e a eficiência, introduzem uma certa imprevisibilidade no tempo de execução das tarefas durante as operações. Nesse contexto, os métodos estatísticos surgem como uma escolha natural, ao incorporarem as incertezas relacionadas ao tempo de execução nas estimativas do tempo de execução de pior caso (WCET). Quando os métodos estatísticos são aplicados, um limite de tempo de execução t^* para um programa específico τ é associado a uma probabilidade

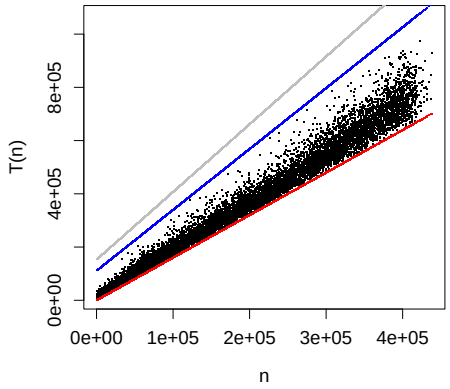


Figura 5.1 QSort sendo executado em um Raspberry Pi 3B no cenário Stress. Os vetores de entrada utilizados possuem tamanhos variados, sendo gerados aleatoriamente. Na ilustração, n indica o número de instruções, e $T(n)$ representa o tempo de execução medido em ciclos do processador. As linhas vermelha e azul representam, respectivamente, os limites inferior e superior das observações. A linha cinza representa as estimativas pWCET para essa amostra.

p, de modo que a probabilidade de τ executar por mais de t^* não é superior a p. O par (t^*, p) é denominado de WCET probabilístico ou pWCET.

MBPTA envolve um conjunto de métodos estatísticos destinados a derivar o pWCET a partir das medições coletadas do programa em estudo. Independentemente do método MBPTA utilizado, é importante que as amostras coletadas representem adequadamente o comportamento do programa quando o objetivo é realizar estimativas seguras de pWCET. É preciso considerar todas as possíveis sequências de instruções, ou seja, os caminhos de execução que o programa pode seguir ao processar suas entradas, bem como os diferentes estados do hardware que podem ocorrer durante a execução. Os estados do hardware dependem das instruções executadas da tarefa analisada e das possíveis instruções de execuções concorrentes. Ainda assim, mesmo que todos os caminhos de execução e estados dos hardwares sejam mapeados, não há garantia de que a distribuição amostrada do tempo de execução seja representativa para o programa em questão. Duas distribuições amostrais, mesmo que apresentem valores dentro da mesma faixa, podem resultar em estimativas diferentes de pWCET.

A Figura 5.1 ilustra uma amostra da execução do programa QSort no cenário Stress (ver seção 4.4.1), utilizando uma plataforma Raspberry Pi 3B. Cada ponto no gráfico representa uma medição do tempo de execução $T(n)$, expresso em ciclos de processador, e do número de instruções executadas n. As linhas vermelha e azul indicam, respectivamente, os limites inferior e superior das medições. O padrão observado no gráfico sugere que o tempo de execução $T(n)$ é uma função do número de instruções executadas n, sendo sujeito a variações no tempo de execução. A inclusão do número de instruções executadas nas medições revela informações importantes que antes eram difíceis de identificar ou não estavam visíveis. É possível obter uma caracterização mais precisa do

comportamento do programa analisado, considerando diferentes caminhos de execução e condições de interferência.

Neste capítulo, é apresentada uma nova abordagem MBPTA, a DBL-MBPTA, que se diferencia da abordagem convencional, as quais modelam somente o tempo de execução $T(n)$ de uma tarefa. Além de considerar $T(n)$, DBL-MBPTA também considera o número de instruções executadas n que resultaram nos respectivos valores de $T(n)$. O nome da abordagem faz referência à variável de interesse modelada, a qual representa a distância relativa das medições em relação às linhas que definem os limites inferior e superior das observações. Essas linhas correspondem aos limites observados para o $T(n)$ em função de n, definindo uma faixa dentro da qual os valores de $T(n)$ tendem a se concentrar. Essas linhas são denominadas na DBL-MBPTA de $L(n)$ (limite inferior) e $U(n)$ (limite superior).

Observando Figura 5.1, a análise da construção das linhas e da distribuição dos pontos observados permite avaliar a qualidade da amostra em relação à representatividade do programa analisado. É possível observar como os caminhos de execução estão sendo cobertos e identificar o impacto das interferências nas execuções. Os caminhos de execução são aproximadamente representados pelo número de instruções executadas, e o intervalo de instruções a ser observado é predefinido pelo analista, com base em inspeções do código, nas configurações operacionais da aplicação e nas variáveis de entrada utilizadas durante as medições. A DBL-MBPTA analisa a amostra, buscando identificar caminhos de execução ainda não percorridos no intervalo de instrução de interesse. Quando esses caminhos não observados são detectados, a DBL-MBPTA utiliza um modelo de redes neurais profundas (DNN) para reconhecer padrões de execução e gerar novos dados de entrada que explorem esses caminhos não cobertos anteriormente. O procedimento é repetido até que todo o intervalo de instruções predefinido seja coberto. Posteriormente, a amostra é submetida a um processo de amostragem proporcional para minimizar potenciais vieses de medição.

Uma vez que o espaço $(n, T(n))$ está bem caracterizado e as linhas de referência $L(n)$ e $U(n)$ são estabelecidas, as distâncias relativas dos dados em relação a essas linhas podem ser obtidas. Por fim, procedimentos estatísticos são aplicados para estimar os valores do pWCET. Na Figura 5.1, essa estimativa é representada pela linha cinza, que pode ser alcançada aplicando alguma técnica capaz de estimar o máximo de uma variável aleatória. Embora algumas abordagens baseadas na desigualdade de Chebyshev sejam descritas no contexto da análise temporal probabilística (V.G. Baranoski; G. Rokne; XU, 2001; RANJBAR et al., 2021; VILARDELL et al., 2022), para os experimentos realizados neste capítulo, é utilizada a teoria dos valores extremos (EVT), devido ao extenso conjunto de métodos e ferramentas disponíveis.

Embora a EVT tenha sido amplamente utilizada no contexto da MBPTA (HANSEN; HISSAM; MORENO, 2009; GRIFFIN; BURNS, 2010; LU et al., 2011; CUCU-GROSJEAN et al., 2012; KOSMIDIS et al., 2013b; ABELLA et al., 2015), pesquisadores levantaram preocupações (GRIFFIN; BURNS, 2010; LIMA; DIAS; BARROS, 2016; GIL et al., 2017). Algumas dessas preocupações estão relacionadas à natureza discreta e à possível dependência dos dados medidos. No entanto, a forma como os dados são medidos neste estudo elimina possível fontes de dependência dos dados. Cada medição é realizada

após reiniciar o programa. As execuções do programa não estão relacionadas. Além disso, a variável modelada $D(n)$ é contínua no intervalo $[0, 1]$. Ainda assim, nem todas as distribuições de dados são adequadas para análise por EVT, e há situações em que a estimativa do pWCET não pode ser obtida por EVT (DIETRICH; HüSLER, 2002).

A abordagem DBL-MBPTA é avaliada utilizando dados sintéticos e reais. Os dados sintéticos são utilizados para verificar a consistência das estimativas obtidas com a abordagem, em cenários onde as estimativas de pWCET são conhecidas. Os dados reais são utilizados para comparar os resultados obtidos com e sem o aprimoramento das amostras. Os dados reais foram coletados de programas com características variadas, selecionados dos *benchmarks* Mälardalen (GUSTAFSSON et al., 2010) e TACLeBench (FALK et al., 2016).

5.2 IMPACTO DA QUALIDADE DA AMOSTRA NAS ESTIMATIVAS PWCET

Em 2016, Lima, Dias e Barros (2016) destacaram em seu estudo que as estimativas pWCET podem ser afetadas por possíveis variações nos protocolos de medição. Utilizando um ambiente de execução baseado em um simulador, os autores obtiveram estimativas pWCET conflitantes por meio de análise *multipath* baseada em EVT, dependendo de como os dados de entrada eram distribuídos. Como de costume, os autores modelaram apenas $T(n)$.

O código monitorado é denominado no estudo de `counter`. Ele consiste em um laço que itera k vezes, onde k é gerado no intervalo $[10^3, 10^4]$.

```
1 void counter(unsigned int k) {
2     unsigned int i;
3     for (i = 0; i < k; i++);
4    }
```

Dois casos são considerados: (a) k segue uma distribuição uniforme no intervalo; (b) k é distribuído de acordo com uma cauda direita de decaimento polinomial pesada e longa, conforme abaixo:

$$(a)k = 10^3 + \left\lfloor \frac{9.000u}{1 - 10^{-5}} \right\rfloor \qquad (b)k = \left\lceil \frac{10^3}{u^{0,2}} \right\rceil$$

sendo u distribuído uniformemente no intervalo de $u = [10^{-5}; 1]$. Esses dois casos podem representar diferentes comportamentos da tarefa durante a execução, dependendo da distribuição das entradas.

Nesta seção, são apresentados os resultados do mesmo experimento, agora executado em uma plataforma real, o Raspberry Pi 3B, detalhada previamente na seção 4.3. O monitoramento do código é realizado no cenário Stress, descrito na seção 4.4.1. Esse cenário proporciona uma melhor visualização dos dados, pois permite observar, para um mesmo valor de n, diferentes valores de $T(n)$, devido à interferência induzida.

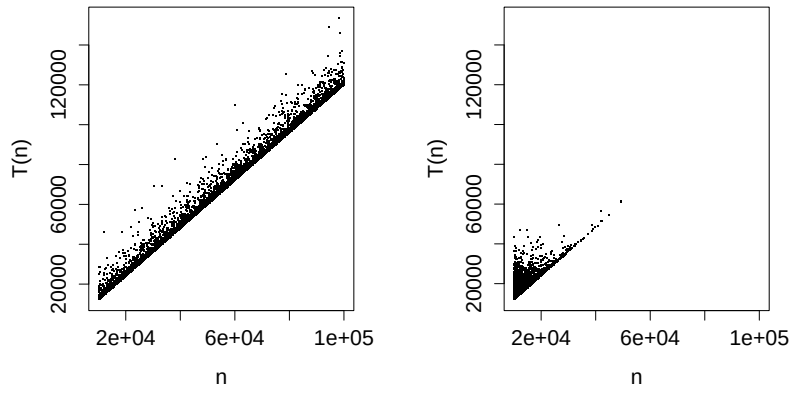
Após monitorar o código e obter os dados medidos, os mesmos efeitos encontrados em Lima, Dias e Barros (2016) utilizando um simulador são observados. Como o número de instruções geradas n é uma função direta de k, é esperado que a distribuição de

$T(n)$ seja aproximadamente a mesma de k. Conforme observações realizadas durante os experimentos, quando $k = 10^4$, o número de instruções geradas é $n^* = 100.017$.

Tabela 5.1 Estimativas de pWCET aplicadas aos valores medidos de $T(n)$ do programa `counter` quando o número de iterações varia dentro do intervalo $[10^3, 10^4]$, de acordo com: uma distribuição uniforme (caso (a)) e uma distribuição de decaimento polinomial (caso (b)).

	MBPTA	
	10^{-2}	10^{-4}
Caso (a)	174782	188798
Caso (b)	49996	75442

A Tabela 5.1 apresenta os resultados das estimativas de pWCET obtidas ao aplicar EVT aos valores medidos de $T(n)$ para o programa `counter`, utilizando dados de entrada gerados nos casos (a) e (b). Como pode ser visualizado na tabela, os resultados conflitantes encontrados em Lima, Dias e Barros (2016) também foram registrados. Isto é, ainda que seja o mesmo programa analisado (`counter`), as estimativas exibem diferenças significativas.



(a) caso (a) (b) caso (b)

Figura 5.2 Programa `counter` rodando em uma plataforma Raspberry Pi 3B em um cenário com interferência induzida. Os dados de entrada foram gerados conforme os casos (a) e (b).

A Figura 5.2 apresenta como as medições do programa `counter` estão distribuídas para os casos (a) e (b). Apesar de k estar dentro do mesmo intervalo para ambos os casos, a forma como k é distribuído em (b) faz com que valores de n maiores que 50.000 não sejam observados, embora sejam possíveis. Além disso, em (b) há uma concentração de medições para n menores que 20.000.

A maneira como os dados do programa `counter` foram medidos no estudo de Lima, Dias e Barros (2016), considerando somente $T(n)$, conforme a abordagem convencional

de MBPTA, impediu uma análise detalhada da distribuição dos dados, dificultando a identificação de possíveis problemas na amostragem. Como pode ser visualizado na Figura 5.2, ao incluir o número de instruções n nas medições, é possível obter informações relevantes, principalmente sobre como o tempo de execução $T(n)$ se distribui ao longo do intervalo de instruções n. Na próxima seção, a abordagem DBL-MBPTA é detalhada, e, utilizando o programa `counter` como exemplo, é demonstrado como os tipos de inconsistências apresentados na Tabela 5.1 e observados na Figura 5.2 podem ser detectados e corrigidos com o uso da DBL-MBPTA.

5.3 DETALHANDO DBL-MBPTA

A abordagem DBL-MBPTA coleta amostras tanto do tempo de execução $T(n)$ quanto do número de instruções n executadas, em pares $(n, T(n))$, para um intervalo específico de instruções. O monitoramento do código analisado pode ser realizado sob diferentes condições de medição, incluindo com presença de interferências. Eventos como cache *miss* e *branch predictions* podem ser provocados, por exemplo, pela execução simultânea de tarefas durante o monitoramento do código. É papel do analista definir o intervalo de instruções n que deseja observar, bem como os cenários de execução. Essas definições podem ser obtidas por meio da análise do ambiente de execução, dos tipos e intervalos de entrada e das características da arquitetura.

Um requisito para aplicar DBL-MBPTA é que os trechos de código analisados gerem caminhos de execução com características homogêneas. Isso significa que as instruções executadas no intervalo de interesse precisam apresentar semelhanças em suas características. Por exemplo, instruções relacionadas ao carregamento e armazenamento, bem como operações aritméticas, se enquadram nessa suposição. A execução dessas instruções requer acesso à memória, barramentos de comunicação, registradores e outros componentes de hardware. Mesmo diante de interferências de hardware, a execução dessas instruções tende a ser consistente ao longo do intervalo observado. Por outro lado, se o intervalo de instruções contiver instruções de I/O, que aparecem somente em algumas faixas de valores, as estimativas obtidas por meio da DBL-MBPTA podem ser imprecisas, levando a resultados pessimistas. Mais detalhes sobre esse requisito são descritos na Seção 5.6, onde também é apresentada uma possível solução para aplicar a DBL-MBPTA em amostras que possuam essas características.

Em qualquer abordagem de MBPTA, é necessário que a amostra coletada seja representativa do comportamento do programa analisado na arquitetura em questão. No entanto, determinar se uma amostra é representativa é uma tarefa complexa. Através da análise dos dados iniciais observados e do intervalo desejado n, a DBL-MBPTA consegue avaliar a amostra e sugerir dados de entrada que cubram caminhos de execução não observados no intervalo de n. Em seguida, estratégias para reduzir possíveis vieses da amostragem são aplicadas e as estimativas de pWCET são obtidas por meio da variável de interesse $D(n)$, que se baseia na tupla $(n, T(n))$.

Nesta seção, a abordagem DBL-MBPTA é descrita, detalhando cada etapa de sua implementação. A seção 5.3.1 apresenta o processo de amostragem e cobertura da amostra, onde é obtido a amostra de dados contendo as medições do programa em análise.

Na seção 5.3.2, é descrito o processo de redução de viés, no qual uma amostra melhor distribuída uniformemente no intervalo de instruções de interesse é gerada a partir das medições. Com a amostra gerada, as linhas de referência podem ser obtidas. Esse processo é descrito na seção 5.3.3. Por fim, a seção 5.3.4 detalha como a variável de interesse $D(n)$ é calculada e como são realizadas as estimativas pWCET.

5.3.1 Amostragem e cobertura da amostra

Esta é a primeira etapa da abordagem DBL-MBPTA. Inicialmente, são realizadas medições de $(n, T(n))$ do programa em análise utilizando dados de entrada gerados aleatoriamente, denominados I_0. As medições realizadas com I_0 resultam na amostra S_0. No entanto, como ilustrado na Figura 5.2(b), considerar somente S_0 pode não ser suficiente. Dessa forma, um modelo de DNN é construído para avaliar S_0 e sugerir novos dados de entrada que permitam cobrir os caminhos de execução não observados, considerando o intervalo de instruções predefinido. A amostra aprimorada com os dados sugeridos pela DNN é denominada de S_1.

Como descrito no capítulo 2, DNN são modelos de RNA que exibem características de aprendizado profundo (DL). A principal diferença entre os modelos de RNA tradicionais e DNN reside no número de neurônios e camadas intermediárias (LIU et al., 2017). Enquanto um modelo tradicional se tem somente algumas dezenas de neurônios e algumas camadas intermediárias, uma DNN pode facilmente ultrapassar esse número, com centenas ou até milhares de neurônios em cada camada e dezenas de camadas intermediárias. A DNN foi utilizada na abordagem DBL-MBPTA para modelar a relação entre os dados de entrada em I_0 e o número resultante de instruções em S_0. Essa modelagem é uma tarefa complexa, uma vez que cada programa em análise se comporta de maneira diferente e possui sua própria estrutura.

Os modelos de DNN utilizados em DBL-MBPTA são construídos com base em aprendizado supervisionado por regressão. Sob essa perspectiva, os dados examinados compreendem variáveis numéricas, onde cada entrada está correlacionada a uma saída conhecida (KAELBLING; LITTMAN; MOORE, 1995; BISHOP, 2011). Para cada um dos programas analisados, um modelo DNN específico é configurado, variando no número de camadas e neurônios dentro de cada camada, conforme necessário. Essas variações são esperadas, uma vez que os programas podem diferir no tipo de dados de entrada que utilizam. Todos os modelos DNN implementados nesse estudo foram desenvolvidos utilizando os pacotes `keras` (KERAS. . . , 2024) e `h2o` (CANDEL; LEDELL, 2023), ambos disponíveis no ambiente R (R Core Team, 2020).

A Figura 5.3 exibe um modelo DNN simplificado contendo um número reduzido de neurônios e camadas. Os modelos DNN utilizados em DBL-MBPTA chegam a ter até 5 camadas e 1.500 neurônios distribuídos entre elas. Na figura, I representa um vetor que indica os dados de entrada ou características dos dados de entrada que resultaram no número observado de instruções. Em resumo, o modelo DNN deve, após analisar n, sugerir o vetor correspondente $I[i_1, i_2, \ldots]$.

O processo de construção dos modelos DNN compreende duas fases: treinamento e teste. Durante o treinamento, utilizando a amostra inicial S_0 e os vetores de entrada

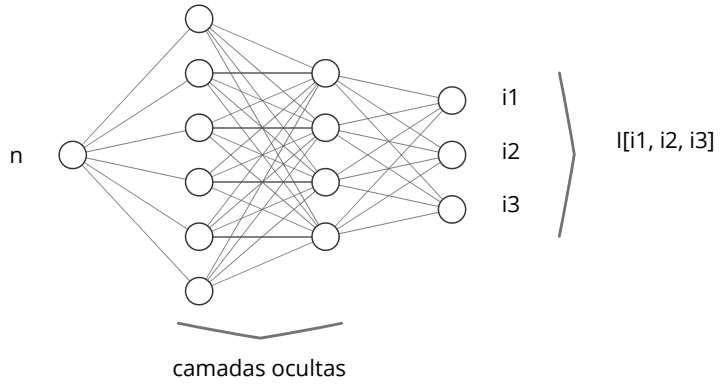


Figura 5.3 Modelo DNN utilizado para prever características dos dados de entrada associadas a números específicos de instruções. A DNN recebe o número de instruções como entrada e sugere os dados (ou características dos dados) que podem levar ao número correspondente de instruções.

correspondentes em I_0, a relação entre n e I é aprendida. Esse processo é realizado pela própria DNN, que ajusta os pesos e parâmetros de seus neurônios. Durante o teste, a qualidade do aprendizado da DNN é avaliada. Essa avaliação é feita calculando o erro médio quadrático (MSE), que representa a diferença entre as previsões e os valores observados reais. Se valores altos de MSE forem observados, o vetor I proposto pelo modelo é inadequado. Nesse caso, diferentes configurações de rede precisam ser consideradas.

Como cada programa possui sua própria estrutura, é necessário aplicar tratamentos distintos durante a construção do modelo DNN. Por exemplo, considere um programa simples, como o `counter`. Para esse programa, o vetor I é composto somente por um elemento que representa o número de interações do laço, e a DNN é treinada para aprender a relação entre o número de interações do laço e o número de instruções n. Em contrapartida, há programas onde a relação é mais complexa, como aqueles que implementam algoritmos de ordenação de vetores. Nesse caso, o número de instruções executadas pode depender não apenas do tamanho do vetor, mas também da ordem de seus elementos. Lidar com essa complexidade exige que a DNN seja treinada para sugerir tanto o número de elementos a serem ordenados quanto o número de permutações a serem realizadas. O vetor correspondente pode, então, ser gerado com base nessas informações. Em resumo, a estrutura e o comportamento do programa devem ser considerados pelo analista para caracterizar I e definir o processo de treinamento da DNN.

Após a construção do modelo DNN, é iniciado o procedimento para avaliar o conjunto de medições S_0. A amostra S_0 é dividida em grupos para os valores de n, onde cada grupo precisa conter um número mínimo de observações, `min_size`. O analista tem liberdade

para definir o valor desse parâmetro. Neste estudo,

$$\texttt{min\_size} = \frac{|S_0|}{\texttt{ns}},$$

onde `ns` denota o número de grupos desejados. O aumento do valor de `ns` pode melhorar a precisão da avaliação, embora possa potencialmente prolongar o tempo de análise. Dado um intervalo requerido para n, cada grupo j deve conter valores de n dentro de $[\texttt{ini[j]}, \texttt{end[j]}]$, com `ini[1]` e `end[ns]` iguais ao valor mínimo e máximo de n no intervalo de interesse estabelecido. O código contendo o processo de cobertura da amostra, realizado para cada grupo j é o seguinte,

```
1 // Para cada grupo j = 1,2,...,ns
2   if(length(j) < min_size){
3     qtd = min_size - lenght(j)
4     inputs[j] = gen_inputs(qtd,ini[j],end[j])
5   }
```

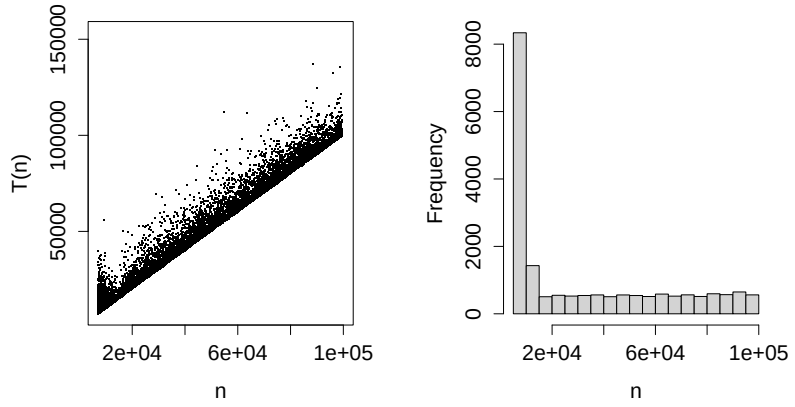
A função `length(j)` retorna o número de observações atuais no grupo j. Quando `length(j)` $<$ `min_size`, o algoritmo calcula quantas medições adicionais são necessárias para que o grupo j tenha a quantidade mínima `min_size` (variável `qtd`). A função `gen_inputs` é onde o modelo DNN é configurado. Quando invocada, ela retorna os dados de entrada a serem utilizados em medições adicionais para cobrir o grupo j. Ao ser executado com essas novas entradas, o programa em análise deve acionar caminhos de execução com o número de instruções dentro do intervalo $[\texttt{ini[j]}, \texttt{end[j]}]$. O processo é repetido para todos os grupos.

Como exemplo ilustrativo, considere S_0 sendo as medições do programa `counter`, utilizando os valores de k gerados no caso (b), conforme mostrado na Figura 5.2(b). O modelo DNN, desenvolvido para `counter`, é treinado para identificar padrões entre o número de instruções n e o número de iterações k. Com base nesse treinamento, o modelo sugere valores de k para obter valores de n ausentes na amostra S_0. Ao medir `counter` com os valores de k sugeridos pela DNN, é obtido a amostra S_1, apresentada na Figura 5.4.

Observando a Figura 5.4(a), é possível confirmar que a amostra S_1 abrange o intervalo de instruções de interesse $[10.000, 100.000]$, igualmente ao caso (a). No entanto, a Figura 5.4(b) indica que há uma concentração de medições com $n < 20.000$. Essas observações são decorrentes de S_0. Além disso, não se pode descartar a possibilidade de que o procedimento de cobertura da amostra também resulte em concentrações de observações, devido a sugestões incorretas dos vetores I pela DNN. Esse agrupamento de observações poderia introduzir inconsistências nas estimativas. O próximo passo da abordagem apresenta uma solução para reduzir o viés nas amostras.

5.3.2 Redução de viés

Considerando S_1 como a amostra coletada na fase de amostragem e cobertura de amostra, conforme descrito na seção 5.3.1, o objetivo da etapa de redução de viés é extrair uma amostra aleatória, de modo que os valores de n estejam distribuídos de forma mais



(a) Amostra S_1 (b) Histograma S_1

Figura 5.4 (a) Amostra S_1 obtida após a aplicação do procedimento de cobertura de amostra nos dados da Figura 5.2(b). (b) Histograma de (a) mostrando uma alta frequência de dados para $n < 20.000$.

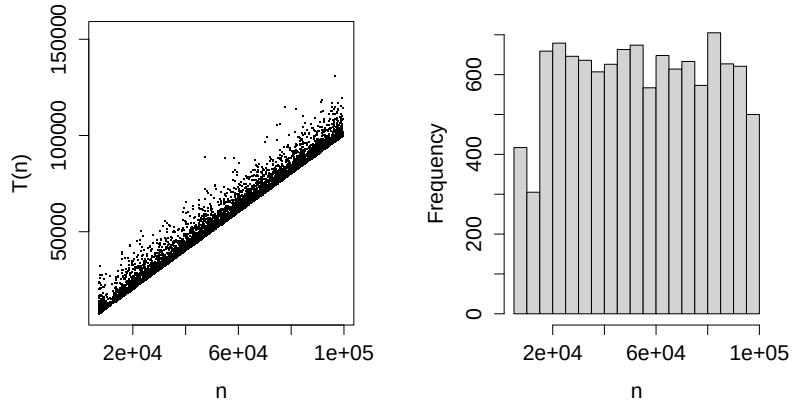
uniforme no intervalo predefinido de instruções, resultando em uma nova amostra, denominada S_2. Em resumo, esta etapa adota uma estratégia de amostragem proporcional, que consiste em dividir S_1 em grupos e calcular o peso de cada grupo em relação ao total de observações. A quantidade de grupos `ns` fica a critério do analista. Nos experimentos realizados nesta pesquisa, foi utilizado o mesmo valor de `ns` definido durante o processo de cobertura de amostra, conforme apresentado na seção anterior. Em seguida, é calculado o valor inversamente proporcional do peso de cada grupo para selecionar os valores de S_1 que formarão S_2.

Seja n_j o número de instruções para cada grupo j, w_j é o peso do grupo j, calculado como $w_j = \frac{n_j}{|S_1|}$. O valor de w_j representa a frequência relativa de observações de S_1 em cada grupo j. Para produzir S_2, é necessário amostrar os valores em S_1 de forma inversamente proporcional aos valores de w_j. Para isso, é necessário calcular w'_j,

$$w'_j = \frac{1 - w_j}{\sum_{i=1}^{ns}(1 - w_i)}\,. \tag{5.1}$$

Com w'_j, é possível amostrar S_2 utilizando, por exemplo, a função `sample` no R (R Core Team, 2020). Qualquer método que consiga gerar uma amostra com base em probabilidades pode ser empregado.

A Figura 5.5 apresenta o gráfico de dispersão e o histograma da amostra S_2, obtidos através do procedimento de redução de viés aplicado à amostra S_1. Como ilustrado na Figura 5.5(b), as observações estão agora melhor distribuídas no intervalo de interesse em comparação ao histograma apresentado na Figura 5.4(b). A Figura 5.6 apresenta a evolução da amostra ao longo de cada etapa da abordagem DBL-MBPTA.



(a) Amostra S_2 (b) Histograma S_2

Figura 5.5 (a) Amostra S_2 obtida após a aplicação do procedimento de redução de viés na amostra S_1, apresentada na Figura 5.4. (b) Histograma da amostra apresentada em (a).

5.3.3 Determinando as linhas de referência

A amostra S_2, gerada na etapa anterior, é considerada suficiente para realizar estimativas de pWCET neste estudo, uma vez que as observações estão razoavelmente bem distribuídas no intervalo de instruções de interesse predefinido. Analisando o gráfico da Figura 5.5(a), deve haver um limite mínimo e máximo para $T(n)$ na execução de cada instrução n. Prosseguindo nesse raciocínio, $T(n)$ pode ser restringido inferior e superiormente por $L(n)$ e $U(n)$.

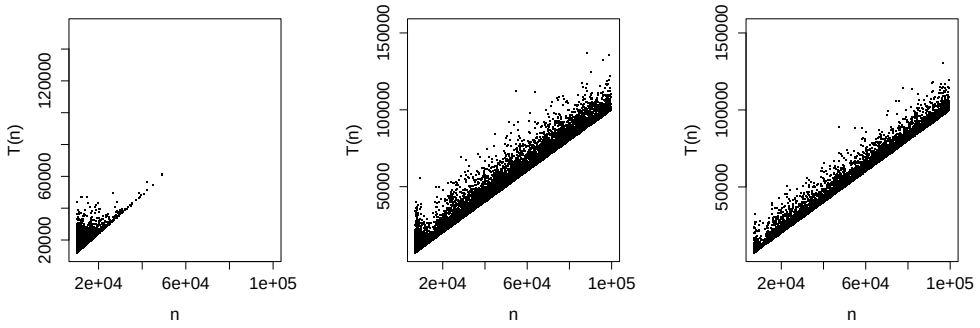
Os limites lineares $L(n) = a^l n + b^l$ e $U(n) = a^u n + b^u$ são obtidos de maneira simples. Considerando uma amostra $\{(n_i, t_i)\}_1^m$ de $(n, T(n))$ com tamanho m, obtida durante as medições, o objetivo é encontrar os valores dos coeficientes a^l, b^l, a^u, e b^u tal que as distâncias entre L, U e os valores observados (n_i, t_i) sejam minimizadas. Mais precisamente, para obter as linhas de referência, é necessário resolver os seguintes problemas de programação linear (*Linear Programming* (LP)),

$$\text{Maximizar} \quad \sum_{i=1}^{m}(a^l\, n_i + b^l) = a^l \sum_{i=1}^{m} n_i + m\, b^l$$

Sujeito a:

$$a^l n_i + b^l \leq t_i, \quad i = 1, 2, \ldots, m \tag{5.2}$$

$$a^l > 0, b^l \geq 0 \tag{5.3}$$



(a) S_0 (b) S_1 (c) S_2

Figura 5.6 Evolução da amostra conforme a aplicação de cada etapa da abordagem DBL-MPBTA. (a) Amostragem. (b) Cobertura da amostra. (c) Redução de viés.

e

$$\text{Minimizar} \quad \sum_{i=1}^{m}(a^u\, n_i + b^u) = a^u \sum_{i=1}^{m} n_i + m\, b^u$$

Sujeito a:

$$a^u n_i + b^u \geq t_i\,, \quad i = 1, 2, \ldots, m \tag{5.4}$$

$$a^u \geq a^l, b^u \geq 0 \tag{5.5}$$

As restrições (5.4) e (5.2) garantem que cada valor observado de $T(n)$ esteja dentro das linhas de referência. As restrições (5.3) e (5.5) decorrem do fato de que o tempo de execução $T(n)$ deve aumentar a medida que n cresce. Além disso, (5.5) assegura que o tempo máximo para executar n instruções não seja menor do que o tempo mínimo correspondente.

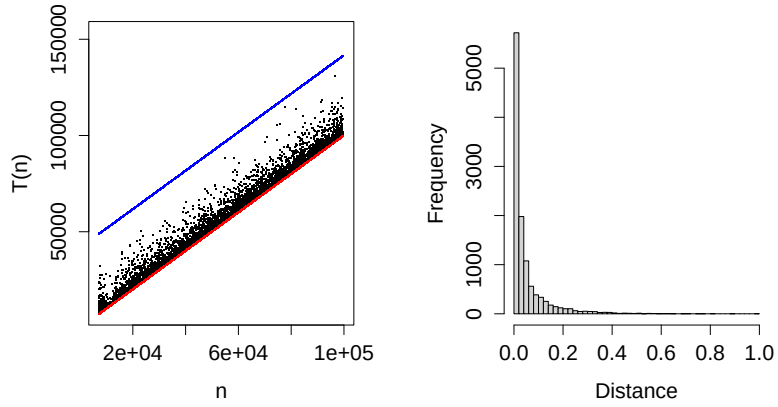
Uma vez que são solucionados os problemas de LP descritos anteriormente, as linhas de referência $U(n)$ e $L(n)$ podem ser obtidas. Para a amostra S_2 apresentada na Figura 5.5, as linhas de referência foram estimadas como $U(n) = 0.99n{+}41741$ e $L(n) = 0.99n{+}90$. A Figura 5.7(a) apresenta a amostra S_2 e as referidas linhas $L(n)$(em vermelho) e $U(n)$(em azul).

5.3.4 Modelando $D(n)$ e estimando o pWCET

Ao limitar $T(n)$ dentro do intervalo $[L(n), U(n)]$, é possível normalizar as observações por meio de uma variável aleatória, que reflete a distância relativa dos valores observados de $T(n)$ em relação a $U(n)$ e $L(n)$. Essa variável é denominada $D(n)$, sendo definida como

$$D(n) = \frac{T(n) - L(n)}{U(n) - L(n)}\,. \tag{5.6}$$

$D(n)$ representa a distribuição dos valores observados entre as linhas $U(n)$ e $L(n)$. Ou seja, para cada observação $\{(n_i, t_i)\}_1^m$ de $(n, T(n))$ é preciso computar



(a) Linhas de referência para S_2 (b) Distância relativa

Figura 5.7 (a) Gráfico de dispersão para S_2 com as linhas de referência correspondentes. (b) Distribuição das distâncias relativas $D(n)$ para os dados apresentados em (a).

$$d_i = \frac{t_i - L(n_i)}{U(n_i) - L(n_i)}\,, \quad i = 1, \ldots, m\,.$$

Para os dados apresentados na Figura 5.7(a), o histograma do conjunto de dados formado com o cálculo das distâncias relativas $D(n)$ é exibido na Figura 5.7(b). Embora se trate de uma variável contínua, esse conjunto pode não atender às suposições de convergência da EVT. Portanto, é preciso verificar possíveis problemas de convergência. Para esse fim, o teste de Dietrich pode ser utilizado (DIETRICH; HüSLER, 2002). A aplicação deste teste aos dados apresentados na Figura 5.7(b) gera os resultados exibidos na Figura 5.8. A estatística do teste (linha sólida preta) é baseada nas k maiores observações da amostra, e seu valor deve estar abaixo do quantil indicado (linha pontilhada vermelha). Como pode ser visualizado na figura, não há evidências para rejeitar a hipótese nula de que os dados provêm de uma distribuição cuja distribuição dos máximos converge assintoticamente para (2.2) ou (2.4).

Por construção, os valores calculados de $D(n)$ estão contidos no intervalo $[0, 1]$. O interesse reside em verificar o menor valor δ tal que, para uma probabilidade alvo p,

$$p \geq \Pr\{\max\{D(n)\} > 1 + \delta\} \tag{5.7}$$

Ao aplicar a EVT para resolver esse problema, é necessário determinar um modelo de máximo que, pelo menos, explique a amostra de máximos produzida por $\{d_i\}_1^m$. Fazendo isso para os dados apresentados na Figura 5.7(b), são produzidos os modelos GEV e GP. Os parâmetros para esses modelos e os respectivos intervalos de confiança de 95% são:

- GEV – Eq. (2.2): $\mu = 0{,}41[0{,}39, 0{,}44]$, $\sigma = 0{,}09[0{,}07, 0{,}11]$; e $\xi = 0{,}12[-0{,}06, 0{,}31]$. Com tamanho de bloco utilizado de 145.

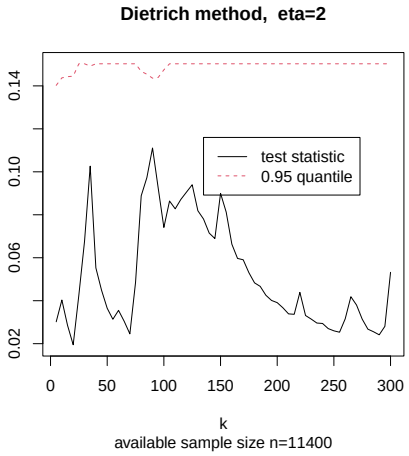


Figura 5.8 Estatística do teste de Dietrich (DIETRICH; HüSLER, 2002) realizada para os dados mostrados na Figura 5.7(b).

- GP – Eq. (2.4): $\sigma_H = 0{,}10[0{,}08, 0{,}12]$; e $\xi = 0{,}04[-0{,}10, 0{,}19]$. Com limiar utilizado de 0,325.

A Figura 5.9 apresenta os gráficos de quantis-quantis (QQ-plots) para os modelos GEV e GP estimados. Esses gráficos permitem avaliar o ajuste dos modelos aos dados. Modelos adequados devem representar bem a amostra de máximos em análise. Como pode ser observado, ambos os modelos se ajustam satisfatoriamente aos dados, com a distribuição GEV apresentando um ajuste ligeiramente superior.

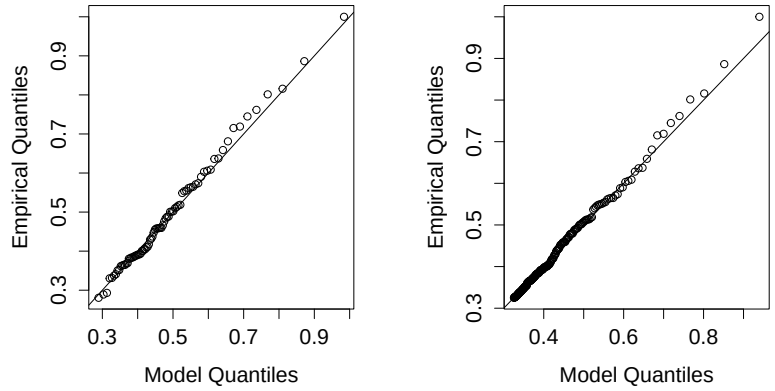
Uma vez que os parâmetros são estimados, é possível obter o quantil δ. Nos casos dos modelos GEV e GP estimados para os dados de distâncias relativas visualizados na Figura 5.7(b), os valores correspondentes de δ com os respectivos intervalos de confiança de 95% são $1{,}100[0{,}423, 2{,}777]$ para GEV e $0{,}792[0{,}653, 1{,}931)]$ para GP, considerando $p = 10^{-4}$.

Uma vez encontrando o quantil δ, é possível obter o valor de pWCET (t^*, p) para um número de instruções executadas n^* como,

$$t^* = L(n^*) + (U(n^*) - L(n^*))(1 + \delta) \tag{5.8}$$

A Tabela 5.2 apresenta as estimativas de pWCET com $p = 10^{-2}$ e $p = 10^{-4}$ para o dados mostrados na Figura 5.7(b). Como pode ser observado, os modelos fornecem estimativas aproximadas de pWCET, embora o modelo GEV tenha demonstrado um ajuste superior, conforme ilustrado na Figura 5.9. As diferenças entre as estimativas estão abaixo de 10%. Além disso, as estimativas apresentadas na Tabela 5.2 estão próximas as apresentadas para o caso (a) na Tabela 5.1. Esses resultados demostram a importância de aplicar as etapas descritas nas seções 5.3.1 e 5.3.2 antes de realizar as estimativas de pWCET.

Uma vez que as linhas de referência são obtidas a partir da amostragem do número de instruções n em um intervalo de interesse predefinido, não é apropriado extrapolar além



(a) GEV (b) GP

Figura 5.9 Gráficos quantis-quantis para os modelos GEV (à esquerda) e GP, cujos parâmetros foram estimados considerando as distâncias relativas mostradas na Figura 5.7(b).

Tabela 5.2 pWCET estimado (t^*, p) com base nos modelos GEV e GP ajustados para os dados mostrados na Figura 5.7(a). O número n^* de instruções é $n^* = 100\,017$.

p	GEV	GP
10^{-2}	142 653	145 615
10^{-4}	187 545	174 734

desse intervalo para prever t^*. Cabe ao analista determinar qual intervalo melhor representa o comportamento operacional do programa em análise. Por exemplo, a análise de WCET pode ser empregada, assumindo que cada instrução leva uma unidade de tempo para ser executada. Alternativamente, medições preliminares com dados de entrada específicos podem ser realizadas. Além disso, pequenas variações nos parâmetros estimados podem produzir diferenças dramáticas na previsão de extremos (KOTZ; NADARAJAH, 2000). Nos experimentos realizados neste capítulo, foram utilizados os valores de probabilidade de excedência p de até 10^{-4}. No entanto, é importante considerar que a escolha dos valores mais adequados ao MBPTA ainda está sujeita a discussões adicionais.

5.4 VERIFICANDO A CONSISTÊNCIA NAS ESTIMATIVAS PWCET

A abordagem DBL-MBPTA estima pWCET indiretamente, calculando um quantil alto das distâncias em relação às linhas de referência $L(n)$ e $U(n)$. Diferentemente da análise convencional de MBPTA de caminho único, que realiza suas estimativas baseadas apenas em observações do tempo de execução. Nesta seção, é investigado se a abordagem DBL-MBPTA é consistente com a análise de caminho único. Mais especificamente, utilizando um experimento controlado para o qual a análise de caminho único fornece

estimativas conhecidas de pWCET, é realizada uma comparação dessas estimativas com as obtidas via DBL-MBPTA.

Seja X uma variável aleatória que representa o tempo necessário para que uma dada máquina hipotética execute uma única instrução. O valor de X pode variar com os estados da máquina durante a execução, o tipo da instrução executada e seus dados processados. É possível imaginar que X segue uma distribuição exponencial com parâmetro λ, uma vez que, na máquina hipotética apresentada, uma sequência de instruções a serem executadas pode ser interpretada como eventos que ocorrem de acordo com uma distribuição de Poisson, a qual, por sua vez, é conhecida por seguir uma distribuição exponencial (ROSS, 2009). Em uma plataforma real, devido à heterogeneidade do hardware e software, isso pode não ser o caso. No entanto, o caso sintético está sendo estabelecido com o único propósito de servir como um teste de conceito, proporcionando uma base de comparação, uma vez que a DBL-MBPTA não depende de qual distribuição os dados são extraídos, desde que quantis extremos possam ser estimados.

Sendo $T(n)$ a representação do tempo que a máquina leva para executar uma sequência de n instruções. Conforme a Eq. (4.1), $T(n)$ pode ser aproximadamente representado pela função $T(n) = An + B$. Deve haver um valor mínimo para a execução de uma instrução, que, por conveniência, é assumido o valor 1. Assim, o coeficiente A pode ser representado por $A = 1 + X$. Adicionalmente, assumindo que $B = 0$ para essa máquina hipotética, $T(n)$ pode ser expresso conforme abaixo:

$$T(n) = (1 + X)n \tag{5.9}$$

A função de probabilidade acumulada é $F_T(n, t)$, que é dada por $\Pr\{T(n) \leq t\} = \Pr\{X \leq t/n - 1\}$. Como $X \sim \texttt{Exp}(\lambda)$,

$$F_T(n, t) = F_X\left(\frac{t - n}{n}\right) = 1 - \exp\left\{\frac{-\lambda(t - n)}{n}\right\} \tag{5.10}$$

É possível encontrar o quantil t através da inversa de F_T, isto é, calculando $F_T^{-1}(p)$. Logo,

$$1 - p = \exp\left\{\frac{-\lambda(t - n)}{n}\right\} \Rightarrow \ln(1 - p) = -\frac{\lambda(t - n)}{n}$$

Portanto,

$$F_T^{-1}(p) = n\left(1 - \frac{\ln(1 - p)}{\lambda}\right) \tag{5.11}$$

O interesse, no entanto, é na distribuição de $\max\{T_i(n, t)\}_1^k$ para uma sequência de k variáveis aleatórias seguindo a Eq. (5.10). Como $T(n, t)$ segue uma distribuição exponencial, é de conhecimento que a distribuição de $\max\{T_i(n, t)\}_1^k$ segue uma distribuição Gumbel G (COLES, 2001), conforme expresso na Eq. (2.3), no Capítulo 2, cujos parâmetros, ou seja, localização μ_G e escala σ_G, devem ser estimados a partir da distribuição

subjacente de $T(n,t)$. Como, por construção, uma sequência de variáveis aleatórias obtidas a partir da Eq. (5.10), $T_1(n,t), T_2(n,t), \ldots, T_k(n,t)$, são independentes,

$$\Pr\{\max_{1}^{k}\{T_i(n,t)\} \leq t\} = \Pr\{T_1(n,t) \leq t, T_2(n,t) \leq t, \ldots, T_k(n,t) \leq t\} = F_t(n,t)^k \,.$$

Para valores grandes de k,

$$F_T(n,t)^k \approx \exp\left\{-\exp\left\{-\frac{t-\mu_G}{\sigma_G}\right\}\right\}, \tag{5.12}$$

onde o lado direito de (5.12) é a distribuição Gumbel. Seja $p = F_T(n,t)^k$. Portanto,

$$p = \exp\left\{-\exp\left\{-\frac{t-\mu_G}{\sigma_G}\right\}\right\} \Rightarrow -\ln p = \exp\left\{-\frac{t-\mu_G}{\sigma_G}\right\},$$

o que leva a

$$-\ln(-\ln p) = \frac{t-\mu_G}{\sigma_G},$$

e assim o quantil $t = \mu_G + G^{-1}(p)\,\sigma_G$, em que

$$G^{-1}(p) = -\ln(-\ln p) \tag{5.13}$$

representa a inversa da distribuição Gumbel padrão. Observe que $p^{1/k} = F_T(n,t)$ e, portanto, $t = F_T^{-1}(p^{1/k})$. F_T^{-1} denota a inversa de $F_T(n,t)$. Assim,

$$\mu_G = F_T^{-1}(p^{1/k}) - G^{-1}(p)\,\sigma_G \,. \tag{5.14}$$

A Eq. (5.14) pode ser usada para estimar os parâmetros de G com uma aproximação razoável, escolhendo dois valores para p, p_1 e p_2, e um valor grande para k. Isto é

$$F_T^{-1}(p_1^{1/k}) - G^{-1}(p_1)\,\hat{\sigma}_G = F_T^{-1}(p_2^{1/k}) - G^{-1}(p_2)\,\hat{\sigma}_G \,,$$

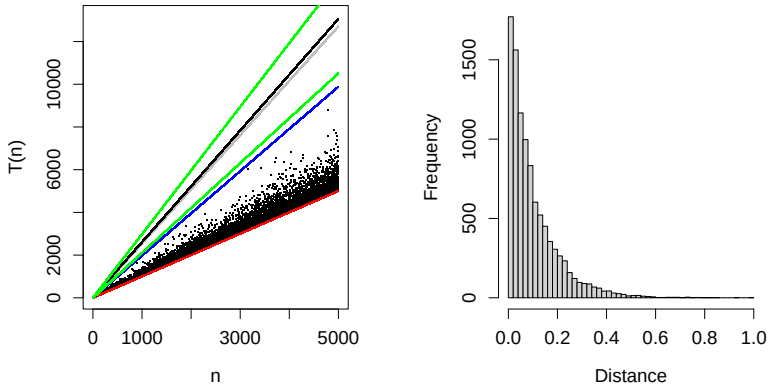
levando a,

$$\hat{\sigma}_G = \frac{F_T^{-1}(p_2^{1/k}) - F_T^{-1}(p_1^{1/k})}{G^{-1}(p_2) - G^{-1}(p_1)} \,. \tag{5.15}$$

Substituindo (5.15) em (5.14), é obtido:

$$\hat{\mu}_G = \frac{G^{-1}(p_2)F_T^{-1}(p_1^{1/k}) - G^{-1}(p_1)F_T^{-1}(p_2^{1/k})}{G^{-1}(p_2) - G^{-1}(p_1)} \,. \tag{5.16}$$

Escolher valores bem espaçados para p_1 e p_2 e um valor razoavelmente grande para k é suficiente para estimar os parâmetros μ_G e σ_G. Aqui, os valores foram definidos como $p_1 = 0{,}25$, $p_2 = 0{,}75$ e $k = 1.000$. Além disso, foram gerados $m = 10.000$ valores de $T(n)$, escolhendo n a partir de uma distribuição discreta uniforme dentro de $[1, 5.000]$, e considerando $X \sim \text{Exp}(\lambda)$, com $\lambda = 10$. Em seguida, as estimativas pWCET foram



(a) Linhas de Referência (b) Distâncias relativas $D(n)$

Figura 5.10 Avaliação da análise: (a) gráfico de dispersão de $(T(n), n)$, linhas de referência e estimativas; e (b) a distribuição de $D(n)$. As estimativas analíticas (linha preta) são comparadas com as fornecidas pelo DBL-MBPTA (linha cinza).

obtidas para todos os valores de n por meio das Eq (5.11), (5.13), (5.15) e (5.16). Essa estimativa é mostrada como uma linha preta na Figura 5.10(a) sendo considerada aqui como o valor verdadeiro para os dados sinteticamente gerados.

Cada ponto no gráfico corresponde a um valor de tempo de execução observado, ou seja, valores gerados de $T(n)$. As linhas vermelha e azul correspondem, respectivamente, às linhas de referência inferior e superior obtidas conforme o procedimento descrito na seção 5.3.3. A Figura 5.10(b) representa o histograma das distâncias relativas observadas associadas à Figura 5.10(a), a partir das quais o valor máximo foi estimado via EVT. Ou seja, esta é a amostra empírica para a distribuição da distância $D(n)$.

Como pode ser observado na Figura 5.10, as estimativas obtidas via DBL-MBPTA (linha cinza) estão próximas das estimativas conhecidas obtidas pelo método convencional de MBPTA de caminho único (linha preta). Conforme indicado pelo intervalo de confiança (linhas verdes), ambas as estimativas podem ser consideradas consistentes uma com a outra.

5.5 RESULTADOS EXPERIMENTAIS

Esta seção apresenta os resultados obtidos com a utilização da abordagem DBL-MBPTA para estimar o pWCET em conjuntos de dados coletados na plataforma Raspberry Pi 3B, descrita na seção 4.3. São examinados um total de 16 programas selecionados dos *benchmarks* Mälardalen (GUSTAFSSON et al., 2010) e TACLeBench (FALK et al., 2016). Além de estimar o pWCET para o conjunto de programas citados, esse experimento também analisa a importância das etapas de cobertura de amostra e redução de viés. Para esse propósito, são realizadas estimativas de pWCET, comparando os resultados com e sem a aplicação dessas duas etapas.

5.5.1 Características do ambiente de execução

Os programas examinados foram configurados para executar sob interferência concorrente induzida. O cenário de execução utilizado é similar ao cenário Stress, descrito na seção 4.4.1. Nesse cenário, o programa em análise é executado no núcleo 1, enquanto o programa gerador de interferência (CIG) é executado simultaneamente nos núcleos 2, 3 e 4. O monitoramento ocorre no ambiente gráfico do sistema operacional Raspbian, com Ethernet, WLAN e Bluetooth habilitados. Esse cenário de execução foi escolhido por representar um desafio adicional na estimativa do pWCET, uma vez que a execução concorrente afeta a execução do programa monitorado ao utilizar os recursos de hardware.

Para aplicar DBL-MBPTA, a plataforma utilizada deve permitir contabilizar a quantidade de instruções (n) e ciclos ($T(n)$). No Raspberry Pi 3B, esse processo pode ser realizado monitorando os eventos `PERF_COUNT_HW_INSTRUCTIONS` e `PERF_COUNT_HW_CPU_CYCLES`, através da biblioteca *Perf Event Open* (KERRISK, 2019). Antes de iniciar a contagem, os dados de entradas são configurados. Para gerar a amostra S_0, os dados de entrada são inicializados de forma aleatória. Para produzir a amostra S_1, as variáveis são inicializadas de acordo os valores sugeridos pela DNN. Por exemplo, em um programa de ordenação, para gerar a amostra S_0, o vetor a ser ordenado é criado com tamanho e elementos gerados aleatoriamente. Já para gerar a amostra S_1, o vetor a ser ordenado é sugerido pelo modelo DNN. Após a contagem, os dados medidos de n e $T(n)$, bem como as variáveis de entrada utilizadas, são salvos em um arquivo externo. As variáveis de entrada são salvas para poderem ser utilizadas no processo de aprendizado da DNN, conforme explicado na seção 5.3.1.

5.5.2 Benchmarks

A Tabela 5.3 apresenta os 16 programas examinados, juntamente com os intervalos de instruções n observados e as características das variáveis de entrada utilizadas. Para cada programa, foi coletada uma amostra S_0 com 10.000 observações. Os primeiros 11 programas foram extraídos do *benchmark* Mälardalen (GUSTAFSSON et al., 2010). Esses programas foram descritos na Tabela 4.2, no Capítulo 4. Os últimos 5 programas foram extraídos do *benchmark* TACLeBench (FALK et al., 2016) sendo escolhidos por representarem funcionalidades comuns em sistemas embarcados: codificadores e decodificadores (Huffman e ADPCM), distância em grafos (Dijkstra), compressão de imagens (Transcodificação JPEG) e controlador baseado em máquinas de estado (Statemate).

As estimativas de pWCET foram obtidas para cada programa apresentado na Tabela 5.3. Duas configurações na DBL-MBPTA foram utilizadas. Na primeira configuração, denominada de configuração A, a modelagem de $D(n)$ é realizada diretamente na amostra S_0. Na segunda configuração, denominada de configuração B, a modelagem de $D(n)$ é realizada na amostra S_2. Isto é, a configuração B utiliza amostras de dados onde as etapas de cobertura de amostra e redução de viés foram aplicadas. As estimativas de pWCET para as duas configurações estão apresentadas na Tabela 5.4. Na tabela, a coluna 'n^*' indica o número de instruções para o qual as estimativas foram obtidas. A coluna 'diff.' apresenta as diferenças entre as estimativas para as configurações A e B, calculadas da seguinte forma,

Tabela 5.3 Lista com os 16 programas examinados. A segunda coluna apresenta o intervalo de instruções executado durante os experimentos, e a terceira coluna apresenta as características dos dados de entrada.

	Programas	**Valores observados de n**	**Tipo de entrada e intervalo**
1	Recursion	$[28, 31.496.367]$	int - $[0, 30]$
2	Fft1	$[46, 171.876]$	int - $[0, 492]$
3	Cnt	$[93, 658.602]$	array size - $[0, 100]$
4	Cover	$[154, 2.550]$	max interations - $[0, 120]$
5	BSearch	$[180, 618]$	array size - $[10, 1.000.000]$
6	Sqrt	$[318, 1.142]$	int - $[0, 10.000]$
7	Fibcall	$[413, 1.500.024]$	int - $[0, 100.000]$
8	Matmult	$[753, 3.729.254]$	array size - $[2, 50]$
9	Insert Sort	$[766, 14.641.126]$	array size - $[10, 1.500]$
10	Quick Sort	$[845, 439.925]$	array size - $[10, 1.500]$
11	Edn	$[130.736, 136.690]$	array size - $[1, 200]$
12	ADPCM Dec.	$[88.479, 17.914.614]$	array size - $[1, 45]$
13	JPEG Transc.	$[5.740.395, 6.536.310]$	array size - $[1, 50]$
14	Dijkstra	$[1.478.519, 46.371.358]$	matrix size - $[20, 80]$
15	Huffman Enc.	$[374.966, 1.160.479]$	array size - $[1, 600]$
16	Statemate	$[758, 540.161]$	loop - $[1, 900]$

$$\text{diff} = \left(\frac{\max(\text{pWCET}_A, \text{pWCET}_B)}{\min(\text{pWCET}_A, \text{pWCET}_B)} - 1 \right) \times 100\,.$$

Como pode ser visto na Tabela 5.4, existem alguns programas para o quais pWCET_A e pWCET_B foram semelhantes, sugerindo que a amostra inicial S_0 apresenta um bom grau de cobertura, sendo suficiente para realizar as estimativas. No entanto, para alguns programas, a não aplicação das etapas de cobertura de amostra e redução de viés afetou as estimativas.

A Figura 5.11 apresenta o gráfico de dispersão e as linhas de referência para os programas BSearch, Recursion e Dijkstra. Conforme pode ser visualizado na Tabela 5.4, esses foram os programas que apresentaram as maiores diferenças entre pWCET_A e pWCET_B. A não observação de alguns caminhos de execução na configuração A resultou em divergências entre as $U(n)$ calculadas. Por exemplo, no programa BSearch, a amostra S_2 (Figura 5.11(d)) possui mais observações próximas a 300 instruções, elevando a probabilidade de observar valores maiores de $T(n)$ nessa faixa de instruções. Efeitos semelhantes, mas para valores altos de n, são observados nos programas Recursion e Dijkstra. Esses resultados indicam que a aplicação da etapa de cobertura de amostra e redução de viés possibilita a obtenção de medições mais representativas e não afetam negativamente a estimativa quando a amostra original S_0 já oferece uma boa caracterização do programa analisado.

Tabela 5.4 Estimativas de pWCET baseadas na DBL-MBPTA sem (A) e com (B) cobertura de amostra e redução de viés

Programas	**n***	**pWCET$_A$**	**pWCET$_B$**	**diff. %**
1 Recursion	40.000	368.824	281.773	30,89
2 Fft1	100.000	490.444	448.310	9,39
3 Cnt	200.000	2.548.313	2.659.193	4,35
4 Cover	2.000	245.566	240.261	2,20
5 BSearch	400	95.168	112.182	17,87
6 Sqrt	300	106.089	92.134	15,14
7 FibCall	1.000.000	1.161.595	1.134.217	2,41
8 Matmult	3.000.000	7.691.013	7.557.081	1,77
9 Insert Sort	2.000.000	3.056.232	3.025.526	1,01
10 Quick Sort	300.000	1.054.955	1.073.299	1,70
11 Edn	133.000	426.861	499.105	16,92
12 ADPCM Dec.	17.000.000	18.386.052	18.550.155	0,89
13 JPEG Transc.	6.400.000	8.825.930	8.808.049	0,20
14 Dijkstra	40.000.000	73.771.601	91.518.800	24,05
15 Huffman Enc.	1.000.000	4.374.557	4.548.237	3,97
16 Statemate	400.000	1.018.895	964.522	5,63

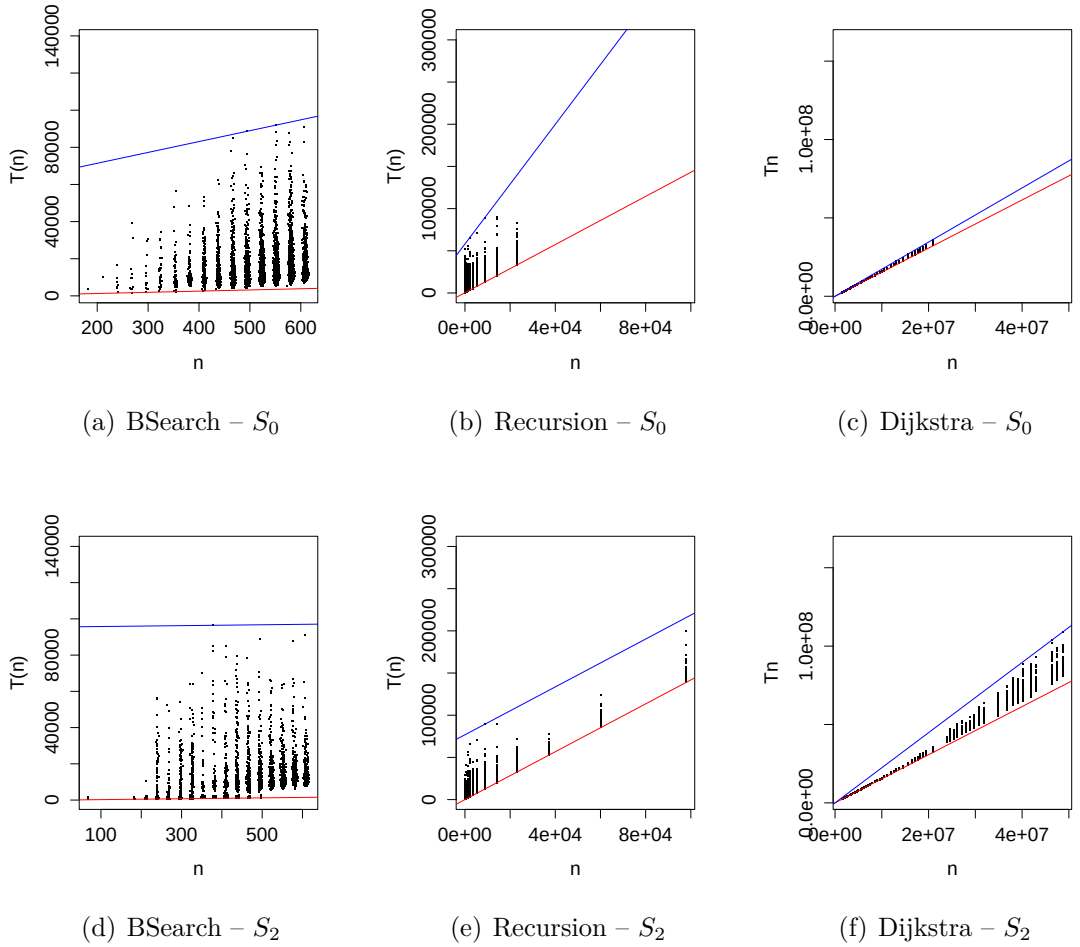
5.6 DIRETRIZES E LIMITAÇÕES DA DBL-MBPTA

A aplicação da DBL-MBPTA envolve alguns aspectos que devem ser cuidadosamente considerados para garantir resultados consistentes na análise. Esses aspectos estão relacionados tanto com as escolhas do analista que usará a abordagem quanto do código ou bloco de código do programa que terá o pWCET estimado. Importante ressaltar que nem todas as amostras permitem aplicar DBL-MBPTA. Esta seção apresenta quais decisões o analista deve tomar ao aplicar DBL-MBPTA e analisa algumas características que as amostras precisam apresentar antes de serem utilizadas na abordagem.

5.6.1 Diretrizes e parâmetros decididos pelo analista

Conforme detalhado na seção 5.3, a abordagem DBL-MBPTA é composta por quatro etapas: amostragem e cobertura da amostra, redução de viés, determinação das linhas de referência e modelagem de $D(n)$. Na fase de amostragem, o analista precisa definir qual intervalo de instrução n deve ser monitorado. O intervalo precisa estar alinhado com o ambiente onde o programa é executado, assim como com as variáveis utilizadas como entrada no programa. Esse intervalo é utilizado pela DNN para verificar quais caminhos de execução não foram cobertos e sugerir novas variáveis de entrada para cobrir esses caminhos.

Na implementação do modelo DNN, o analista precisa identificar quais elementos devem fazer parte do vetor de entrada I. Esses elementos devem estar relacionados ao



(a) BSearch – S_0 (b) Recursion – S_0 (c) Dijkstra – S_0

(d) BSearch – S_2 (e) Recursion – S_2 (f) Dijkstra – S_2

Figura 5.11 Gráficos de dispersão e linhas de referência para BSearch, Recursion e Dijkstra usando as amostras S_0 e S_2.

número de instruções n gerado. Por exemplo, no programa Sqrt, I pode ser composto pelo número do qual se deseja calcular a raiz quadrada. Já no programa Matmult, I pode incluir uma variável que indique o tamanho das matrizes que serão multiplicadas. Ao considerar I, podem ser necessários ajustes nos parâmetros do modelo de DNN para melhorar o aprendizado, como modificar o número de camadas, neurônios, funções de ativação, entre outros. Alguns desafios para o analista podem aparecer nessa etapa. Por exemplo, se o vetor I contiver muitos elementos, pode dificultar o processo de aprendizado. A DNN precisaria sugerir I baseada apenas em n, o que pode não ser possível.

Na etapa de redução de viés, o analista deve definir o tamanho dos grupos a serem utilizados. Embora mais grupos aumente a precisão, isso também pode resultar em um tempo de processamento mais longo. Nos experimentos realizados neste capítulo, foi decidido fixar o número de grupos em 10. Foi observado que tamanhos maiores não mostravam diferença significativa nos resultados. Já na etapa de modelagem de $D(n)$, o analista precisa realizar algumas decisões comuns em abordagens MBPTA. Por exemplo,

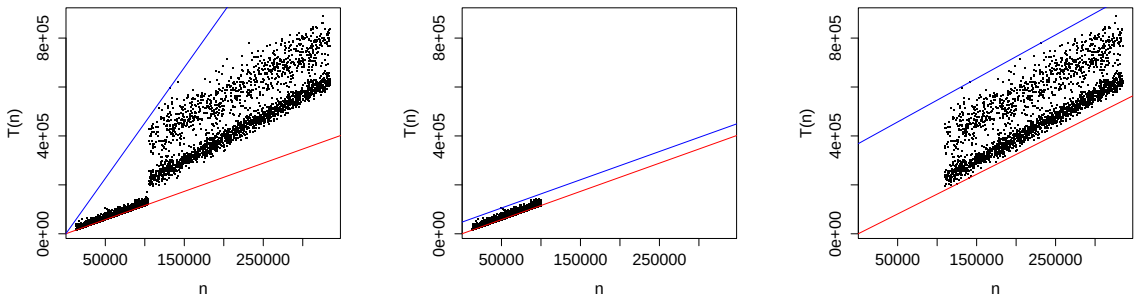
quando se faz uso de EVT, é necessário escolher o tamanho máximo do bloco ou o limiar apropriado para o conjunto de dados utilizado.

5.6.2 Limitações da DBL-MBPTA

Para realizar estimativas de pWCET via DBL-MBPTA, é necessário que as amostras utilizadas sejam compostas por caminhos de execução homogêneos. Isto é, as amostras devem consistir em caminhos de execução que apresentem características semelhantes ao longo do intervalo de instruções de interesse n. Se a amostra contiver caminhos de execução que levem significativamente mais tempo para serem executados do que outros caminhos no mesmo intervalo, a eficácia da DBL-MBPTA pode ser comprometida. Nesse cenário, as linhas de referência $L(n)$ e $U(n)$ podem não se ajustar adequadamente aos dados observados, afetando a precisão das estimativas de pWCET. Por exemplo, considere o seguinte código,

```
1 for(int i=0;i<=k;i++){
2   if(i>8000) memcpy(ptr1, ptr2, 100);
3 }
```

k é distribuído uniformemente no intervalo $[1.000, 10.000]$. Quando $i > 8.000$, a função `memcpy` é chamada para copiar 100 bytes da memória apontada por `ptr2` para a memória apontada por `ptr1`.



(a) $n = [10.000, 350.000]$ (b) $n = [10.000, 100.000]$ (c) $n = [100.000, 350.000]$

Figura 5.12 Exemplo de uma amostra de dados que apresenta caminhos de execução não homogêneos. Os caminhos de execução observados para $n \leq 100.000$ possuem características distintas em relação aos caminhos de execução para $n > 100.000$. Em (a), é mostrado que $L(n)$ e $U(n)$ não se ajustam adequadamente aos dados observados. Em (b) e (c), é apresentada uma possível solução: dividir a amostra e construir $L(n)$ e $U(n)$ para cada subamostra.

A Figura 5.12 apresenta um conjunto de 10.000 medições do código anterior, realizadas no mesmo ambiente de execução descrito na seção 5.5.1. Como pode ser observado, os caminhos de execução contidos nessa amostra não são homogêneos. A operação de manipulação de blocos de memória executada pela função `memcpy` demonstra um desempenho significativamente mais lento quando $n > 100.000$ em comparação com $n <= 100.000$.

Conforme ilustrado na Figura 5.12(a), as linhas $L(n)$ e $U(n)$ não se ajustam adequadamente às observações, o que geraria um certo pessimismo nas estimativas, caso fossem obtidas.

Uma alternativa para aplicar o DBL-MBPTA a conjuntos de dados não homogêneos é dividir as observações com base nas características das instruções. O gráfico de dispersão é uma ferramenta valiosa para identificar e visualizar essa não homogeneidade. Ao analisar a distribuição dos dados na Figura 5.12(a), é possível separá-los em duas subamostras: $n \leq 100.000$ e $n > 100.000$. Em seguida, as linhas $L(n)$ e $U(n)$ podem então ser construídas para cada subamostra, conforme ilustrado nas Figuras 5.12(b) e 5.12(c).

5.7 CONSIDERAÇÕES FINAIS

Neste capítulo, é apresentada a abordagem DBL-MBPTA, uma nova abordagem MBPTA *multipath* que, além de considerar o número de ciclos, considera também o número de instruções em suas medições. Foi demonstrado que, ao incluir o número de instruções, é possível obter informações mais detalhadas sobre as medições. A abordagem utiliza linhas de referência para delimitar os dados, tanto inferior quanto superiormente, e mantém as medições dentro de um intervalo de instruções predefinido. Ao restringir as medições, é possível melhorar sua qualidade e aumentar a representatividade em relação ao bloco de código analisado. Para esse propósito, são empregados métodos baseados em aprendizado de máquina. A consistência da abordagem é verificada por meio de dados sintéticos controlados, e evidências de sua aplicabilidade são apresentadas em experimentos realizados em uma plataforma multinúcleo real.

Capítulo

6

UTILIZANDO EVENTOS DE HARDWARE PARA MELHORAR RESULTADOS COM MBPTA

Nos dois capítulos anteriores foram apresentadas as abordagens centrais desta tese. O Capítulo 5 descreveu a abordagem DBL-MBPTA, e o Capítulo 4 apresentou um método para modelar o tempo de execução dos programas por meio de eventos relacionados ao hardware. Esse método foi validado mediante cinco técnicas de aprendizado de máquina (ML), onde foi comprovado sua eficiência. Ao final do Capítulo 4, foi mencionado que as ocorrências de eventos no nível de hardware poderiam ser utilizadas como informações adicionais para derivar limites probabilísticos mais precisos de pWCET.

Seguindo esse contexto, este capítulo investiga como as informações fornecidas pelos eventos de hardware podem contribuir para a melhoria dos resultados alcançados com MBPTA. Especificamente, uma vez que os eventos possibilitam capturar padrões de comportamento do sistema ao executar um determinado programa — tais como quais recursos de hardware foram utilizados, identificação de gargalos ou bloqueios de execução —, essas informações podem ser utilizadas para compreender o comportamento do sistema, auxiliando o analista na tomada de decisão, no ajuste do protocolo de medição e na identificação de cenários críticos de execução que poderiam não ter sido considerados inicialmente. Consequentemente, medições mais alinhadas ao comportamento real do ambiente podem ser obtidas.

Para atender tal objetivo, um experimento é realizado visando compreender a relação entre os eventos de hardware e o comportamento do tempo de execução do programa em análise, considerando diferentes cenários. Um ambiente monitorado composto por uma arquitetura Raspberry Pi 3B (RASPBERRY, 2018), equipada com um processador quad-core ARM Cortex-A53 é utilizado. Quatro programas são executados nessa arquitetura, com a possibilidade de execução concorrente. O programa MSort é o programa no qual se tem interesse em realizar as estimativas de pWCET. Os programas BSearch, ISort e Fft1 são os outros três programas. Os quatro programas pertencem ao *benchmark* Mälardalen (GUSTAFSSON et al., 2010). Para simplificar a análise e facilitar a didática do experimento, a execução dos programas foi limitada a, no máximo, dois por núcleo de processador, reduzindo o número de cenários possíveis. Assim, no pior caso,

o MSort é executado em conjunto com um dos programas concorrentes em um mesmo núcleo, enquanto os demais programas são executados nos núcleos restantes. Políticas de escalonamento não são abordadas neste capítulo, uma vez que estão fora do escopo desta tese.

Especificamente em relação ao ambiente de execução descrito no parágrafo anterior, a análise dos eventos de hardware, considerando as diferentes concorrências, possibilita compreender o comportamento da arquitetura durante a execução do programa MSort. Assim, espera-se que seja possível identificar gargalos de execução, tipos de interferências geradas pelos programas concorrentes, recursos consumidos e, consequentemente, identificar cenários que tendem a aumentar o tempo de execução desse programa, os quais também precisariam ser incluídos no protocolo de medição utilizado para realizar suas estimativas pWCET. Este capítulo está estruturado da seguinte forma: a Seção 6.1 apresenta a contextualização do experimento; a Seção 6.2 descreve detalhadamente a metodologia utilizada; a Seção 6.3 traz os resultados obtidos; e, por fim, a Seção 6.4 apresenta as conclusões do capítulo.

6.1 CONTEXTUALIZAÇÃO

Em MBPTA, o protocolo de medição representa os procedimentos utilizados para coletar dados de tempo de execução de um programa em uma determinada arquitetura. Esse protocolo define o conjunto de valores de entrada e as configurações iniciais de hardware necessárias para explorar um subconjunto dos caminhos possíveis do código do programa, bem como os estados do hardware que podem influenciar o comportamento temporal do sistema. Para que a distribuição estimada de pWCET seja válida em cenários futuros de operação, é necessário que os dados de entrada e os estados do hardware reflitam aqueles que ocorrerão durante execução no ambiente real de operação (DAVIS; CUCU-GROSJEAN, 2019). A capacidade dos dados observados de representar as condições e comportamentos do ambiente real de execução é o que define a **representatividade**. Um protocolo de medição é considerado representativo se, ao coletar uma amostra de k observações de tempos de execução, a estimativa de pWCET para essa amostra é muito próxima da estimativa de pWCET obtida para o conjunto completo de todas as observações possíveis (MAXIM et al., 2017). Alcançar a representatividade das amostras é um desafio ainda em aberto em MBPTA. Particularmente, quando se utiliza EVT para estimar o pWCET, as distribuições obtidas são fortemente influenciadas pelas condições de execução consideradas durante a análise (SANTINELLI; GUET; MORIO, 2017).

A abordagem DBL-MBPTA visa melhorar a representatividade dos dados de entrada, assegurando a cobertura dos possíveis caminhos de execução de acordo a um intervalo de instruções de interesse pré-definido pelo analista. Todavia, isso pode não ser suficiente. É preciso também considerar o comportamento do programa em relação aos diferentes tipos de interferências que podem afetar sua execução, as variações nos acessos aos recursos do hardware, entre outros. Em estruturas de hardware complexas, como o modelo de Raspberry Pi utilizado nos experimentos desta tese, é ainda mais desafiador garantir a representatividade dos possíveis estados do hardware devido ao comportamento dependente de estado, que pode afetar a latência das instruções que acessam a memória

(DAVIS; CUCU-GROSJEAN, 2019). Entretanto, ignorar essas variações pode resultar em amostras que não refletem adequadamente o que ocorre no ambiente de operação.

A abordagem de seleção de eventos apresentada no Capítulo 4 compara as ocorrências de eventos de hardware em dois cenários de execução distintos. Esses cenários são projetados para referenciar diferentes estados do hardware quando executa o programa em análise. Por exemplo, o cenário Base representa uma condição típica, com baixa carga no sistema, onde, geralmente, somente o programa em análise está em execução. O cenário Stress representa uma condição de alta complexidade, na qual os programas são executados concorrentemente, resultando em utilização intensiva dos recursos de hardware. Ao comparar as ocorrências dos eventos de hardware entre os cenários de execução considerados, é possível identificar padrões de comportamento do programa em análise e compreender como o hardware responde a diferentes condições de operação. Especificamente em relação ao Raspberry Pi 3B, é possível avaliar em quais cenários houve maior consumo de recursos de memória, aumento de cache *miss*, acesso à memória principal ou situações em que a *Data Processing Unit* (DPU) ou *pipeline* permaneceram ociosos ou bloqueados devido à espera por carregamento de informações. Esse tipo de análise pode auxiliar o analista a detectar situações de maior complexidade, que poderiam ser ignoradas inicialmente, mas que são necessárias para obter medições mais alinhadas com as condições reais de execução.

6.2 METODOLOGIA DO EXPERIMENTO

Nesta seção, a metodologia empregada no experimento é detalhada. As manifestações dos eventos de hardware são medidas durante a execução do programa MSort no ambiente monitorado, com diferentes cenários sendo definidos conforme a execução concorrente dos programas BSearch, ISort e Fft1. Os dados observados são analisados e avaliados por meio das estastísticas Z_0 e F_0, complementadas por gráficos de histogramas.

O experimento é conduzido em um ambiente monitorado, utilizando uma plataforma Raspberry Pi 3B (RASPBERRY, 2018), a mesma utilizada nos experimentos dos capítulos anteriores. O sistema operacional utilizado é o Raspbian (FOUNDATION, 2018). Dentre os eventos disponíveis nessa arquitetura, são considerados aqueles relacionados à memória e ao desempenho do sistema, tais como cache *access* e *miss* em diferentes níveis de hierarquia, acessos à memória externa e eventos diretamente associados a bloqueios e ociosidades na CPU (`DPUEmpty` a `PipeWaitStore`). Esses eventos são suficientes para entender o comportamento do hardware e do programa analisado, e viabilizar a identificação das situações mais críticas de execução.

Nas próximas subseções, são apresentados os cenários de execução considerados, de acordo com as restrições impostas, bem como as estratégias de monitoramento dos eventos de hardware e de avaliação do experimento.

6.2.1 Definição dos cenários para monitoramento dos eventos de hardware

Considerando a restrição do ambiente de execução monitorado utilizado no experimento, de que apenas dois programas podem ser executados simultaneamente em um único núcleo

de processador, os seguintes cenários são definidos:

- **Cenário 0**: Similar ao cenário Base utilizado no experimento do Capítulo 4 (Seção 4.4.1). Apenas o programa MSort é executado no núcleo 1. Os programas BSearch, ISort e Fft1 não são executados nesse cenário;
- **Cenário 1**: MSort e Fft1 são executados no núcleo 1, BSearch no núcleo 2 e ISort no núcleo 3;
- **Cenário 2**: MSort e BSearch são executados no núcleo 1, Fft1 no núcleo 2 e ISort no núcleo 3;
- **Cenário 3**: MSort e ISort são executados no núcleo 1, Fft1 no núcleo 2 e ISort no núcleo 3;

Em todos os cenários, o núcleo 4 permanece ocioso. Outros cenários poderiam ser considerados, como, por exemplo, a execução de um programa em cada núcleo. No entanto, durante a realização do experimento, foi constatado que essa e outras configurações não apresentariam relevância para os objetivos do estudo, uma vez que não foram observadas modificações significativas no tempo de execução do programa MSort em comparação ao cenário 0. No entanto, em um ambiente real de operação, é necessário considerar o maior número possível de cenários, a fim de minimizar a probabilidade de negligenciar caminhos de execução críticos.

6.2.2 Monitoramento dos eventos de hardware

O monitoramento dos eventos de hardware é realizado como descrito no Capítulo 4. A entrada do programa MSort é fixada em um vetor de tamanho 10.000, com os elementos já definidos. Os eventos são monitorados por meio da interface *Perf Event Open* (KERRISK, 2019). Como informado no Capítulo 4, essa plataforma possui uma PMU com sete registradores, dos quais um é obrigatoriamente reservado para monitorar o número de ciclos. Apesar de utilizar uma entrada fixa na execução do programa MSort, foi observada uma pequena variação no número de instruções, possivelmente causada por erros de medição. Por essa razão, um dos registradores é utilizado para monitorar o número de instruções, o qual é empregado para normalizar os valores observados dos eventos (processo semelhante ao descrito na Seção 4.3). Consequentemente, somente cinco registradores estão disponíveis para monitorar os demais eventos. Cada evento é medido ao longo de 40.000 execuções, 10.000 para cada cenário. A Tabela 6.1 apresenta os eventos de hardware analisados. Todos relacionados à memória e ao desempenho do sistema. Durante a análise, foi observado que eventos relacionados a *branch predictions* e *Translation Lookaside Buffer* (TLB) não apresentaram variações significativas entre os cenários. Por esse motivo, tais eventos foram desconsiderados especificamente neste experimento. Alguns dos eventos listados na Tabela 6.1 já foram descritos na Tabela 4.1. No entanto, para facilitar a visualização, suas descrições foram replicadas. Todas as informações sobre os eventos foram extraídas da documentação do ARM Cortex-A53 (ARM, 2016).

Tabela 6.1 Eventos a serem monitorados após filtragem preliminar

Nome	Cod. Hex.	Descrição
`L1IReload`	$0x01$	Recarregamento de cache de instruções L1
`L1DReload`	$0x03$	Recarregamento de cache de dados L1
`MemAccess`	$0x13$	Acesso à memória de dados
`L2Access`	$0x16$	Acesso à cache L2
`L2Reload`	$0x17$	Recarregamento de cache L2
`ReqMemExt`	$0xC0$	Requisição de memória externa
`PipeBlockSB`	$0xC7$	Bloqueio no *pipeline* devido ao *store buffer* estar cheio
`DPUEmpty`	$0xE0$	DPU vazia, não devido à *micro-TLB miss* ou *instruction miss*
`DPUEmptyIMiss`	$0xE1$	DPU vazia e *instruction cache miss* sendo processada
`PipeWaitLMiss`	$0xE7$	Espera na escrita do *pipeline* devido um *load miss*
`PipeWaitStore`	$0xE8$	Espera na escrita do *pipeline* devido operações de armazenamento

Os eventos `L1DAccess` e `L1IAccess`, que, conforme a documentação do ARM Cortex-A53, representam, respectivamente, acessos à cache de dados e à cache de instruções no L1, não foram utilizados na análise. Foi observado uma maior ocorrência desses eventos no cenário 0 (sem interferência concorrente), o que seria consistente caso esses eventos representassem cache *hit*, uma vez que a ausência de concorrência aumenta a probabilidade de acertos na cache. Por outro lado, se esses eventos representassem acesso à cache, seria esperada uma ocorrência mais semelhante entre os cenários. Como a documentação não deixa clara a natureza desses eventos, eles foram excluídos da análise para evitar interpretações incorretas. De todo modo, os eventos `L1IReload` e `L1DReload`, que representam, respectivamente, o recarregamento de instruções e de dados na cache L1, já oferecem informações relevantes sobre o que ocorre nesse nível de memória em cada cenário.

6.2.3 Estratégia de avaliação

A estratégia de avaliação do experimento descrito neste capítulo foi projetada para investigar como as informações fornecidas pelos eventos de hardware podem ser utilizadas para aprimorar os resultados obtidos com o MBPTA. O objetivo é, com base nos dados observados dos eventos de hardware, entender o comportamento de execução do programa na arquitetura proposta, identificar possíveis cenários de maior complexidade e analisar as razões por trás dessa complexidade. É esperado que os maiores tempos de execução do programa analisado ocorram em cenários com maiores ocorrências de eventos diretamente relacionados ao desempenho do sistema, como bloqueios no pipeline, DPU ociosa e cache *miss*.

As manifestações dos eventos são analisadas com base nas estatísticas Z_0 e F_0. São as mesmas estatísticas utilizadas no Capítulo 4, replicadas abaixo:

$$Z_0 = \frac{|\overline{E}_{ij} - \overline{E}_{i0}|}{\sqrt{s_{ij}^2 + s_{i0}^2}} \qquad e \qquad F_0 = \frac{s_{ij}^2}{s_{i0}^2},$$

onde $\overline{E}_{i0}$ e s_{i0}^2 representam, respectivamente, a média e a variância das observações

do evento i no cenário 0. O índice j assume os valores 1, 2 e 3, que correspondem aos cenários com concorrência. Dessa forma, $\overline{E}_{ij}$ e s^2_{ij} representam, respectivamente, a média e a variância das observações do evento i nesses cenários.

As estatísticas Z_0 e F_0 são complementares na análise, pois fornecem informações distintas. Enquanto Z_0 mede a diferença entre as médias das ocorrências dos eventos, F_0 analisa a razão entre as variâncias dessas ocorrências. Por exemplo, valores de Z_0 próximos a 0 e F_0 próximos a 1, indicam que o evento não sofreu alterações significativas entre o cenário 0 e o cenário concorrente em estudo. Por outro lado, valores menores em Z_0 combinados com valores maiores de F_0, quando comparados aos demais cenários, podem sugerir que o evento manteve uma média semelhante, mas com uma maior variância. Como será demonstrado na próxima seção, esse padrão foi observado em alguns dos eventos analisados.

6.3 RESULTADOS

Nesta seção, são apresentados os resultados da análise do comportamento dos eventos de hardware listados na Tabela 6.1, medidos durante a execução do programa MSort nos cenários definidos na Seção 6.2.1. Especificamente, as ocorrências desses eventos são avaliadas por meio das estatísticas Z_0 e F_0, e de histogramas, considerando o cenário 0 (sem concorrência) e os cenários concorrentes (1, 2 e 3). Em seguida, os valores de Z_0 e F_0 são comparados entre os cenários concorrentes, com o objetivo de identificar padrões nas ocorrências desses eventos que possam indicar possíveis complexidades na execução do programa MSort, o que, eventualmente, aumentaria seu tempo de execução. Por fim, são realizadas estimativas de pWCET utilizando a abordagem DBL-MBPTA para verificar se os cenários identificados como os mais críticos pela análise dos eventos, correspondem àqueles com as maiores estimativas.

6.3.1 Eventos relacionados à memória

A Tabela 6.2 apresenta os valores de Z_0 e F_0 para os eventos de hardware relacionados à memória, considerando os três cenários de execução concorrente. Conforme descrito no Capítulo 4, a arquitetura ARM Cortex-A53 possui um processador com quatro núcleos, onde cada núcleo contém duas memórias caches L1 privadas de 32KB, uma para armazenar instruções e outra para armazenar dados. A memória cache L2 é compartilhada entre os núcleos e possui um tamanho de 512KB. A política de substituição é a aleatória, onde qualquer linha de cache pode ser substituída com a mesma probabilidade. A plataforma possui 1GB de memória RAM.

O evento `L1IReload` (recarregamento de cache L1 de instruções) apresentou valores de Z_0 mais elevados nos cenários 1 ($7{,}65 \times 10^0$) e 2 ($9{,}30 \times 10^0$) em comparação com o cenário 3 ($4{,}46 \times 10^0$). No entanto, no cenário 3, foi registrado o maior valor de F_0 ($1{,}69 \times 10^5$), indicando uma maior variância nas observações.

A Figura 6.1 apresenta o histograma dos valores observados para o evento `L1IReload`. No cenário 0, onde não há interferência de programas concorrentes, a ocorrência desse evento é próxima de zero, sugerindo que, quando o MSort é executado isoladamente no

Tabela 6.2 Valores das estatísticas Z_0 e F_0 para eventos de hardware relacionados à memória, considerando os cenários descritos na Seção 6.2.1.

		Cenários de Execução		
Eventos		1 - Fft1	2 - BSearch	3 - ISort
`L1IReload`	Z_0	$7{,}65 \times 10^0$	$9{,}30 \times 10^0$	$4{,}46 \times 10^0$
	F_0	$4{,}51 \times 10^4$	$2{,}40 \times 10^4$	$1{,}69 \times 10^5$
`L1DReload`	Z_0	$1{,}27 \times 10^0$	$1{,}18 \times 10^0$	$1{,}44 \times 10^0$
	F_0	$9{,}51 \times 10^{-1}$	$9{,}64 \times 10^{-1}$	$1{,}05 \times 10^0$
`L2Access`	Z_0	$9{,}08 \times 10^0$	$1{,}05 \times 10^1$	$3{,}82 \times 10^0$
	F_0	$1{,}08 \times 10^2$	$6{,}33 \times 10^1$	$8{,}72 \times 10^2$
`L2Reload`	Z_0	$1{,}94 \times 10^0$	$1{,}85 \times 10^0$	$2{,}42 \times 10^0$
	F_0	$5{,}13 \times 10^0$	$5{,}02 \times 10^0$	$1{,}98 \times 10^0$
`ReqMemExt`	Z_0	$9{,}05 \times 10^0$	$1{,}04 \times 10^1$	$3{,}69 \times 10^0$
	F_0	$9{,}76 \times 10^2$	$5{,}83 \times 10^2$	$8{,}18 \times 10^3$
`MemAccess`	Z_0	$7{,}90 \times 10^0$	$8{,}04 \times 10^0$	$3{,}80 \times 10^0$
	F_0	$7{,}97 \times 10^2$	$6{,}10 \times 10^2$	$5{,}44 \times 10^3$

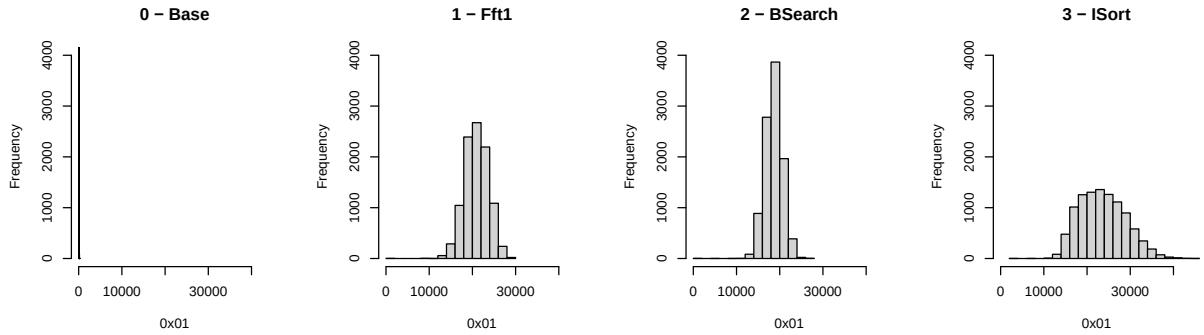


Figura 6.1 Histograma dos valores observados do evento `L1IReload` (recarregamento da cache L1 de instruções) nos quatro cenários analisados. O cenário 3 - ISort apresenta maior variância, com os maiores valores observados.

núcleo, a maioria das instruções necessárias para sua execução está disponível na cache L1.

Nos cenários com interferência, a média das ocorrências do evento `L1IReload` aproxima-se de 20.000, justificando o menor valor de Z_0 no cenário 3. Conforme a equação de Z_0, se as médias forem semelhantes entre os cenários, o cenário com maior variância tende a apresentar o menor valor de Z_0. Um padrão semelhante ao encontrado nas observações de `L1IReload` foi identificado para os eventos `L2Access` (acesso à cache L2), `ReqMemExt` (requisição de memória externa) e `MemAccess` (acesso à memória de dados).

Informações interessantes são observadas para os eventos `L1DReload` (recarregamento da cache L1 de dados) e `L2Reload` (recarregamento da cache L2 de dados). As ocorrências do evento `L1DReload` são razoavelmente semelhantes entre os cenários, sendo o cenário 3 com valores de Z_0 e F_0 levemente superior. Aparentemente, esse comportamento é replicado para o evento `L2Reload`.

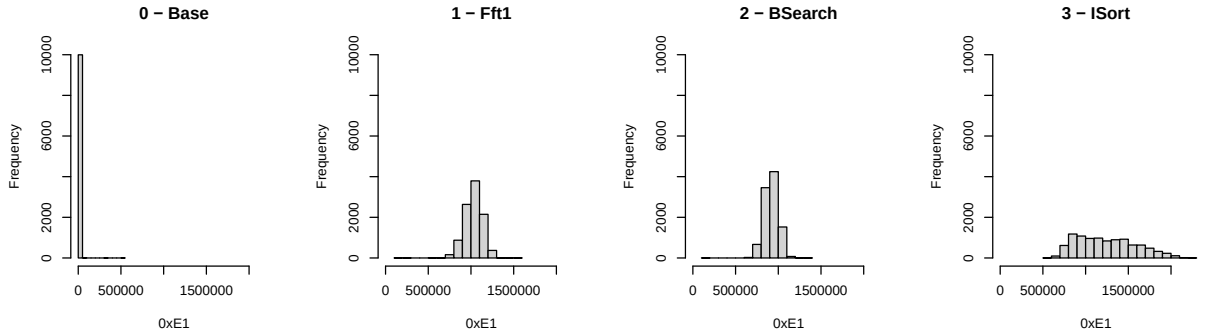
Em resumo, as informações fornecidas pela Tabela 6.2 e pela Figura 6.1 confirmam uma disputa pelos recursos de memória entre o MSort e os programas concorrentes. Apesar das médias das ocorrências desses eventos serem razoavelmente próximas entre os cenários, o cenário 3 apresenta a maior variância, sugerindo uma competição mais intensa nos diferentes níveis de memória. Especificamente, ao analisar separadamente cada nível, a menor eficiência no acesso à cache L1 de instrução resultou em mais acessos à cache L2 e, consequentemente, à memória externa, aumentando a possibilidade de ocorrências de eventos relacionados ao desempenho do sistema, como será visto na próxima seção.

6.3.2 Eventos relacionados ao impacto no desempenho

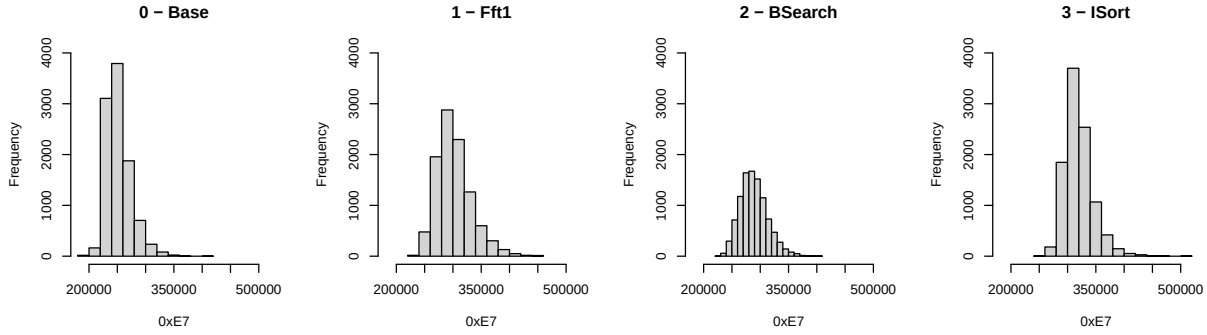
Dentre os eventos de hardware disponibilizados para monitoramento pela arquitetura ARM Cortex-A53, alguns estão diretamente relacionados aos impactos no desempenho durante a execução do programa. Especificamente em relação a essa arquitetura, esse conjunto de eventos é responsável por contabilizar a quantidade de ciclos em que a DPU permaneceu ociosa ou em que ocorreu um bloqueio no *pipeline*, ambos devido a causas específicas. Eventos com essas características estão disponíveis para monitoramento na maioria das arquiteturas modernas (DONGARRA et al., 2003). A análise desses eventos permite a identificação de pontos críticos de execução, onde o desempenho pode ser significativamente afetado. Em algumas situações, as ocorrências desses tipos de eventos estão relacionadas às manifestações de outros eventos, como é o caso dos eventos `DPUEmptyIMiss` e `PipeWaitLMiss`, que estão associados às ocorrências de cache *miss*.

A Tabela 6.3 apresenta os valores de Z_0 e F_0 para os eventos de hardware relacionados ao desempenho do sistema. Os eventos `DPUEmpty` e `DPUEmptyIMiss`, ambos referentes a ociosidade da DPU, exibiram no cenário 3 um comportamento semelhante ao de alguns eventos relacionados à memória, com menor Z_0 e maior F_0. Embora os valores médios de ocorrências desses dois eventos sejam próximos entre os cenários, a variância parece ser significativamente maior no cenário 3, como pode ser confirmado através dos histogramas apresentados na Figura 6.2(a), referente as ocorrências do evento `DPUEmptyIMiss`. É também nesse cenário que são observados os valores mais altos para esses eventos.

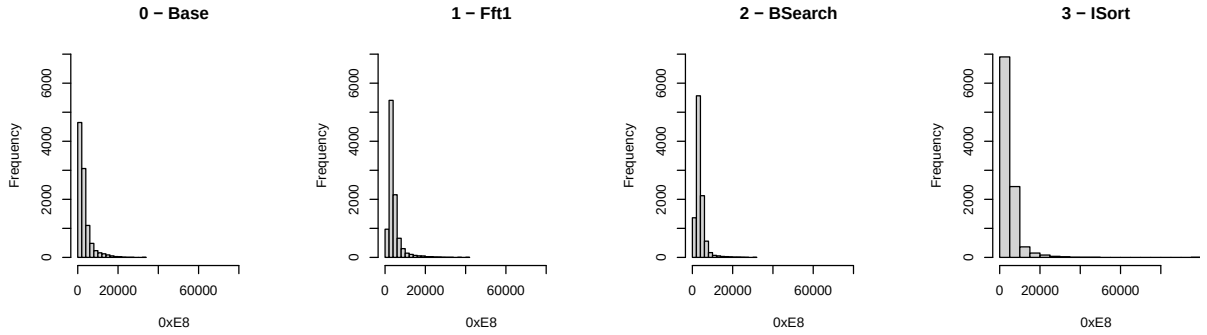
Os eventos `PipeBlockSB`, `PipeWaitLMiss`, `PipeWaitStore`, que estão relacionados a



(a) `DPUEmptyIMiss`



(b) `PipeWaitLMiss`



(c) `PipeWaitStore`

Figura 6.2 Histogramas dos valores observados dos eventos `DPUEmptyIMiss` (DPU vazia e *instruction cache miss* sendo processada), `PipeWaitLMiss` (bloqueio no *pipeline* devido um *load miss*) e `PipeWaitStore` (bloqueio no pipeline devido operações de armazenamento), considerando os quatro cenários analisados.

Tabela 6.3 Valores das estatísticas Z_0 e F_0 para eventos de hardware diretamente relacionados ao impacto no desempenho do sistema, considerando os cenários descritos na Seção 6.2.1.

		Cenários de Execução		
Eventos		1 - Fft1	2 - BSearch	3 - ISort
`PipeBlockSB`	Z_0	$3{,}10 \times 10^{-2}$	$1{,}25 \times 10^{-1}$	$1{,}30 \times 10^{-1}$
	F_0	$1{,}49 \times 10^{0}$	$7{,}28 \times 10^{-1}$	$2{,}33 \times 10^{00}$
`DPUEmpty`	Z_0	$5{,}17 \times 10^{0}$	$5{,}17 \times 10^{0}$	$3{,}20 \times 10^{0}$
	F_0	$3{,}13 \times 10^{0}$	$2{,}23 \times 10^{0}$	$1{,}56 \times 10^{1}$
`DPUEmptyIMiss`	Z_0	$1{,}00 \times 10^{1}$	$1{,}13 \times 10^{1}$	$3{,}64 \times 10^{0}$
	F_0	$2{,}06 \times 10^{2}$	$1{,}25 \times 10^{2}$	$2{,}30 \times 10^{3}$
`PipeWaitLMiss`	Z_0	$1{,}32 \times 10^{0}$	$1{,}12 \times 10^{0}$	$2{,}03 \times 10^{0}$
	F_0	$2{,}00 \times 10^{0}$	$1{,}23 \times 10^{0}$	$1{,}33 \times 10^{0}$
`PipeWaitStore`	Z_0	$0{,}23 \times 10^{0}$	$0{,}13 \times 10^{0}$	$0{,}35 \times 10^{0}$
	F_0	$1{,}07 \times 10^{0}$	$0{,}53 \times 10^{0}$	$1{,}51 \times 10^{0}$

bloqueios no *pipeline*, apresentam valores de Z_0 e F_0 relativamente próximos entre os cenários concorrentes. Os eventos `PipeBlockSB` (bloqueio no *pipeline* devido a um *store buffer*) e `PipeWaitStore` (bloqueio no *pipeline* causado por operações de armazenamento) exibem valores de Z_0 e F_0 ligeiramente mais elevados no cenário 3. O que sugere que o volume de operações de armazenamento em *buffer* ou em memória é levemente superior nesse cenário, mas segue o padrão encontrado nos outros dois cenários. Essa hipótese pode ser confirmada por meio da análise da Figura 6.2(c), que apresenta a distribuição das ocorrências do evento `PipeWaitStore` nos quatro cenários de execução. As distribuições são razoavelmente semelhantes entre os cenários, porém, o cenário 3 apresenta os maiores valores observados.

Em relação ao evento `PipeWaitLMiss` (bloqueio no *pipeline* devido a um *load miss*), os valores observados em todos os cenários, incluindo o cenário 0, situam-se no intervalo de $[200.000, 350.000]$. No entanto, conforme Figura 6.2(b), os valores máximos observados para esse evento ultrapassam 500.000 no cenário 3.

6.3.3 Conclusões e estimativas pWCET

As análises estatísticas realizadas sobre os eventos relacionados à memória e aos bloqueios na CPU permitiram chegar às seguintes conclusões:

- Em média, os eventos de hardware se manifestaram de forma razoavelmente semelhante entre os cenários, diferenciando-se pela variância observada;
- Quando o MSort é executado concorrentemente com o ISort no núcleo 1, é observado uma maior variância na ocorrência de *miss* na cache L1 de instruções, acessos à cache L2 e memória externa. Consequentemente, a probabilidade de observar, em uma única execução, um número maior de ocorrências de eventos relacionados a

bloqueios na CPU, devido à cache *miss* também aumenta. Assim, o cenário 3 se configura como o mais crítico para a execução do *MSort* entre os cenários analisados;

- O comportamento dos eventos de hardware é razoavelmente semelhante ao comparar os dados observados nos cenários 1 e 2. No entanto, com base nos valores de F_0, a execução do Fft1 no mesmo núcleo que o MSort apresenta uma variância ligeiramente maior na ocorrência de eventos relacionados à memória nos diferentes níveis e ao bloqueio na CPU, em comparação à execução do BSearch. Portanto, entre os cenários 1 e 2, o cenário 1 impõe uma complexidade levemente superior à execução do MSort.

Agora, as estimativas de pWCET utilizando DBL-MBPTA são realizadas considerando os quatro cenários de execução. Como o DBL-MBPTA é uma abordagem MBPTA *multipath*, as medições do programa MSort foram realizadas utilizando vetores de entrada de tamanhos variando entre 1.000 e 10.000, com elementos gerados aleatoriamente. A Figura 6.3 apresenta o gráfico de dispersão e as linhas de referências para a execução do MSort nos cenários 1 a 4. Variabilidades semelhantes às encontradas nas ocorrências dos eventos de hardware também foram observadas no tempo de execução para o cenário 3.

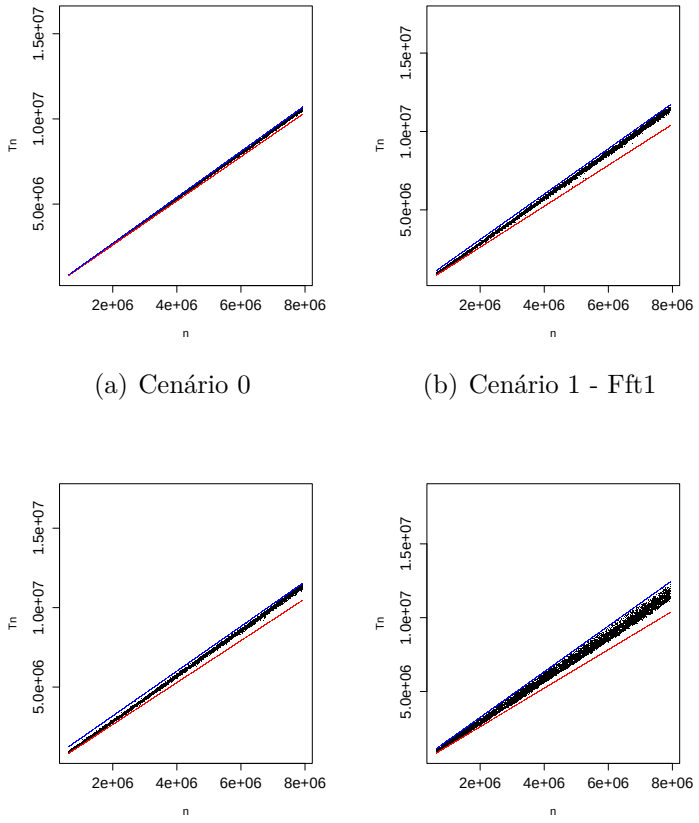
A Tabela 6.4 apresenta as estimativas de pWCET e os respectivos intervalos de confiança para a execução do programa MSort nos quatro cenários analisados. As estimativas foram obtidas considerando a probabilidade de excedência $p = 10^{-4}$ e o número de instruções de interesse $n^* = 6.000.000$.

Tabela 6.4 Estimativas de pWCET e intervalo de confiança, com probabilidade de excedência $p = 10^{-4}$ e instruções de interesse $n^* = 6.000.000$, para o programa MSort utilizando a DBL-MBPTA, considerando os quatro cenários de execução estudados.

	DBL-MBPTA	
Cenários	**pWCET**	**Inter. Confiança**
0	8.157.273	[8.068.883 , 8.245.663]
1	8.974.954	[8.874.627 , 9.075.280]
2	8.865.633	[8.716.629 , 9.014.638]
3	9.458.741	[9.023.173 , 9.894.310]

A execução do MSort no mesmo núcleo que o ISort resulta em uma maior competição por recursos de memória, iniciando na cache L1 de instruções. Essa competição se propaga para a cache L2 e, posteriormente, para a memória externa, aumentando a probabilidade de bloqueios no pipeline e períodos de inatividade na DPU devido à espera no carregamento de dados. Consequentemente, maiores tempos de execução são observados nesse cenário, resultando em estimativas de pWCET mais elevadas.

A análise dos eventos de hardware possibilitou compreender o impacto da concorrência de recursos na execução do MSort nos diferentes cenários considerados. Embora a simples execução do programa em contextos distintos permita até identificar o cenário mais



(a) Cenário 0 (b) Cenário 1 - Fft1

(c) Cenário 2 - BSearch (d) Cenário 3 - ISort

Figura 6.3 Gráficos de dispersão e linhas de referência para a execução do MSort nos cenários 1 a 4.

crítico, a observação dos eventos possibilita ao analista tomar decisões — como possíveis alterações no protocolo de medição ou em parâmetros do sistema —, fundamentadas em informações do comportamento do sistema.

6.4 CONSIDERAÇÕES FINAIS

Este capítulo investigou a relação entre a variação no tempo de execução do programa MSort e os eventos de hardware observados durante sua execução, considerando diferentes cenários. Através da análise dos dados observados, buscou-se identificar quais eventos eram os mais recorrentes entre os cenários, e como eles se distribuíram estatisticamente entre as múltiplas execuções. A utilização das estatísticas Z_0 e F_0, combinada com histogramas, permitiu analisar os eventos de acordo a sua média e variância, e compreender os seus comportamentos entre os diferentes cenários.

Os resultados apresentados mostraram que há variações significativas no tempo de execução do Msort, especialmente no cenário que considera o ISort como programa concorrente. Essa variação está fortemente associada a eventos relacionados a acessos à memória, como cache *miss* e memória externa, e à ociosidade e bloqueios de DPU e pipeline. A identificação de gargalos fornece subsídios para estratégias de melhorias no protocolo de medição ou modificações nos parâmetros de sistemas, com objetivo de gerar amostras mais alinhadas ao comportamento real do ambiente de execução. Para replicar ou estender este estudo, recomenda-se seguir as seguintes diretrizes:

- Utilizar plataformas com suporte a PMU, capazes de registrar eventos de hardware a ponto de poder distinguir padrões durante a execução dos programas, e com bibliotecas ou ferramentas adequadas para leitura dos contadores de eventos;
- Projetar experimentos considerando diferentes cenários de execução — por exemplo, com e sem interferência induzida, com e sem interface gráfica do sistema operacional, com e sem serviços ativos como Wi-Fi e Bluetooth. Importante manter fixa a entrada dos programas para assegurar que as variações observadas nos eventos registrados sejam atribuídas exclusivamente ao ambiente de execução;
- Aplicar abordagens estatísticas para caracterização dos eventos observados, como as estatísticas Z_0 e F_0 utilizadas neste experimento, além de análises gráficas que ajudem a visualizar o comportamento dos eventos e possíveis padrões de ocorrência;
- Executar os programas repetidamente sob cada cenário experimental, de forma a capturar a variabilidade do sistema e permitir a obtenção de distribuições estatísticas dos eventos de hardware.

Capítulo 7

CONCLUSÕES E DIREÇÕES FUTURAS

RTS são sistemas computacionais projetados para executar suas tarefas em limites de tempo previamente definidos, desempenhando um papel importante em aplicações onde a precisão, segurança e confiabilidade são fundamentais. No entanto, a análise temporal de RTS em arquiteturas complexas é um desafio. Essas arquiteturas, frequentemente, incorporam componentes de hardware que podem melhorar o desempenho, mas acabam introduzindo incertezas quando o objetivo é determinar o WCET das tarefas, tornando inviável a utilização das metodologias tradicionais de análise temporal. Nesse contexto, MBPTA vem sendo empregado como uma possível solução a essas dificuldades, estabelecendo limites superiores para o tempo de execução com base em observações. No entanto, permanecem em aberto questionamentos relacionados à amostragem dos dados, como: qual protocolo de medição adequado? Como replicar o comportamento do ambiente real de produção? Como garantir a representatividade das amostras? Entre outras questões. Neste trabalho, foram apresentados dois estudos com o propósito de aprimorar a precisão e a confiabilidade do processo de análise temporal por meio de MBPTA.

O primeiro estudo, apresentado no Capítulo 4, introduziu uma metodologia para avaliar como o tempo de execução dos programas pode ser influenciado por eventos relacionados ao hardware. Dado que existem arquiteturas nas quais nem todos os eventos de hardware podem ser monitorados, foi apresentado uma abordagem para a seleção dos eventos mais relevantes. O objetivo deste estudo é fornecer informações adicionais no nível de hardware que possam ser adaptadas às estratégias atuais de aplicação de MBPTA.

O Capítulo 5 apresenta uma nova abordagem de MBPTA *multipath*, denominada DBL-MBPTA, que se diferencia da abordagem convencional de MBPTA por considerar não somente o tempo de execução das tarefas, mas também a quantidade de instruções executadas. Essa inclusão permite obter detalhes importantes, como a possibilidade de verificar como os caminhos de execução estão sendo cobertos, e qual é o impacto das interferências nas execuções. A abordagem utiliza ferramentas de ML e estratégias de amostragem proporcional para aprimorar as amostras, sugerindo caminhos de execução

previamente não observados e minimizando potenciais vieses de medição. Após a validação da amostra, a variável de interesse, para a qual se estima o pWCET, é obtida a partir das distâncias relativas das medições em relação a linhas de referência. Essas linhas estabelecem os limites inferior e superior dos dados em relação ao tempo de execução observado. A consistência da DBL-MBPTA é confirmada mediante dados sintéticos, e sua aplicabilidade é evidenciada por dados medidos em uma plataforma real, reforçando a relevância da abordagem em contextos práticos.

Apesar das abordagens apresentadas possuírem limitações, as contribuições deste trabalho avançam o estado da arte ao propor metodologias para a análise temporal em arquiteturas complexas, que possibilitam uma melhor compreensão do comportamento do ambiente de execução e um aprimoramento nos dados medidos. O uso do número de instruções na modelagem, associado ao tempo de execução, representa uma inovação no estado da arte, tornando a DBL-MBPTA uma das principais contribuições deste trabalho.

A hipótese que orientou os estudos apresentados nesta tese, descrita no Capítulo 1, foi, de modo geral, confirmada pelos resultados obtidos. De fato, técnicas de estatísticas e inteligência computacional, aliadas à análise de eventos de hardware, possibilitam melhorar os resultados obtidos com MBPTA. No entanto, alguns questionamentos presentes no estado da arte não puderam ser respondidos. Por exemplo, ainda não é possível garantir a representatividade da amostra, nem definir o protocolo de medição ideal.

Com base nos resultados alcançados, algumas questões podem ser exploradas como possíveis direções para trabalhos futuros:

- **Identificação de Eventos durante monitoramento**. Como poderia ser acelerado o processo de identificação dos eventos de hardware mais relevantes? A aceleração desse processo poderia contribuir para ajustes mais eficientes no protocolo de medição, permitindo, por exemplo, que o analista se concentre em cenários mais críticos e, assim, reduza a coleta de dados irrelevantes.

- **Amostragem com linhas de referências**. As linhas de referências $U(n)$ e $L(n)$ poderiam ser utilizadas para avaliar a amostra inicial S_0? Se sim, estimar $U(n)$ e $L(n)$ durante a amostragem poderia ser útil como um mecanismo para decidir quando parar de medir. Por exemplo, considere A_i o subconjunto das observações total A. Esses subconjuntos são coletados de forma crescente, ou seja $A_1 \subseteq A_2 \subseteq A_3 \subseteq ....$ Se as linhas $U(n)$ e $L(n)$ não apresentam variações significativas entre os subconjuntos, isso poderia indicar que a amostragem atingiu um ponto de estabilidade, sugerindo que a coleta de dados poderia ser interrompida.

- **Automatização da configuração do modelo DNN**. Considerando que a identificação dos elementos do vetor de entrada I é de responsabilidade do analista e que esses elementos variam conforme o programa em análise, influenciando diretamente o processo de aprendizado da DNN, como poderia ser otimizada essa etapa? O objetivo é diminuir a carga de responsabilidade do analista e aprimorar a seleção dos elementos que compõem I.

- **Verificação de homogeneidade**. Considerando que, para utilizar a DBL-MBPTA, é necessário que as amostras sejam compostas por caminhos de execução homogê-

neos, como poderia ser automatizado o processo de identificação de intervalos de n que violam essa suposição? A automação desse processo poderia agilizar a tomada de decisão pelo analista.

- **Gerenciamento de recursos**. Como utilizar as informações obtidas tanto no nível de eventos de hardware quanto na projeção do pWCET para otimizar o agendamento das tarefas? Integrar essas informações poderia permitir uma alocação mais inteligente dos recursos, reduzindo as latências e melhorando o desempenho geral do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

ABELLA, J. et al. 68. extreme value theory in computer sciences: The case of embedded safety-critical systems. In: *Current Topics on Risk Analysis: ICRA6 and RISK 2015 Conference*. [S.l.: s.n.], 2015. p. 579.

ABELLA, J. et al. Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Trans. Des. Autom. Electron. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 22, n. 4, jun 2017. ISSN 1084-4309. Disponível em: <https://doi.org/10.1145/3065924>.

ABELLA, J. et al. Heart of gold: Making the improbable happen to increase confidence in mbpta. In: *2014 26th Euromicro Conference on Real-Time Systems*. [S.l.: s.n.], 2014. p. 255–265.

AKESSON, B. et al. A comprehensive survey of industry practice in real-time systems. *Real Time Syst.*, v. 58, n. 3, p. 358–398, 2022.

ALTMEYER, S.; CUCU-GROSJEAN, L.; DAVIS, R. I. Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Syst.*, Kluwer Academic Publishers, USA, v. 51, n. 1, p. 77–123, jan. 2015. ISSN 0922-6443. Disponível em: <https://doi.org/10.1007/s11241-014-9218-4>.

ALZUBAIDI, L. et al. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, v. 8, n. 1, p. 53, Mar 2021. ISSN 2196-1115.

AMALOU, A. N.; PUAUT, I.; MULLER, G. We-hml: Hybrid wcet estimation using machine learning for architectures with caches. In: *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. [S.l.: s.n.], 2021. p. 31–40.

ANDRADE, T. N. C. et al. On the selection of relevant hardware events for explaining execution time behavior. In: *2021 XI Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2021. p. 1–8.

ARCARO, L. F.; SILVA, K. P.; OLIVEIRA, R. S. D. On the reliability and tightness of gp and exponential models for probabilistic wcet estimation. *ACM Trans. Des. Autom. Electron. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 23, n. 3, mar 2018. ISSN 1084-4309. Disponível em: <https://doi.org/10.1145/3185154>.

ARCHER, N. P.; WANG, S. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, v. 24, n. 1, p. 60–75, 1993. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-5915.1993.tb00462.x>.

ARM, H. *ARM Cortex-A53 MPCore Processor Technical Reference Manual*. [S.l.], 2016. Disponível em: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0500e/index.html>.

AROCA, R. V. Análise de sistemas operacionais de tempo real para aplicações de robótica e automação. In: . [S.l.: s.n.], 2008.

AWAD, M.; KHANNA, R. *Support Vector Regression*. Berkeley, CA: Apress, 2015. 67–80 p. ISBN 978-1-4302-5990-9.

AZIMI, R. et al. Enhancing operating system support for multicore processors by using hardware performance monitoring. Association for Computing Machinery, New York, NY, USA, v. 43, n. 2, 2009.

BATE, I.; REUTEMANN, R. Worst-case execution time analysis for dynamic branch predictors. In: *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.* [S.l.: s.n.], 2004. p. 215–222.

BECHTEL, H. Y. M. G. Denial-of-service attacks on shared cache in multicore: Analysis and prevention. 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2019.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. Hardcover. [S.l.]: Springer, 2011. ISBN 0387310738; 9780387310732.

BONENFANT, A. et al. Early wcet prediction using machine learning. In: *Worst-Case Execution Time Analysis*. [S.l.: s.n.], 2017.

BOZHKO, S. et al. What really is pwcet? a rigorous axiomatic proposal. In: *2023 IEEE Real-Time Systems Symposium (RTSS)*. [S.l.: s.n.], 2023. p. 13–26.

BRADLEY, J. V. Distribution-free statistical tests. In: . [s.n.], 1968. Disponível em: <https://api.semanticscholar.org/CorpusID:123472665>.

BREIMAN, L. 2001 forests. *Machine Learning 45, 5–32*, Machine Learning, 2001. ISSN 0360-0300.

BROOCK, W. A. et al. A test for independence based on the correlation dimension. *Econometric Reviews*, Taylor & Francis, v. 15, n. 3, p. 197–235, 1996.

BU, L. et al. Systematically ensuring the confidence of real-time home automation iot systems. *ACM Trans. Cyber-Phys. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 2, n. 3, jun 2018. ISSN 2378-962X. Disponível em: <https://doi.org/10.1145/3185501>.

BURGUIèRE, C.; ROCHANGE, C. A contribution to branch prediction modeling in wcet analysis. In: . [S.l.: s.n.], 2005. p. 612–617.

BURNS, A.; EDGAR, S. Predicting computation time for advanced processor architectures. In: *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*. [S.l.: s.n.], 2000. p. 89–96.

CANDEL, A.; LEDELL, E. *Deep Learning with H2O*. Mountain View, California, 2023. Disponível em: <https://h2o.ai/resources/>.

CANNON, A. J. *Multi-Layer Perceptron Neural Network with Optional Monotonicity Constraints*. Victoria, Canada, 2022. Disponível em: <https://CRAN.R-project.org/package=monmlp>.

CASTILLO, J. D.; DAOUDI, J.; LOCKHART, R. Methods to distinguish between polynomial and exponential tails. *Scandinavian Journal of Statistics*, v. 41, n. 2, p. 382–393, 2014.

CAZORLA, F. J. et al. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 1, fev. 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3301283>.

CAZORLA, F. J. et al. Proartis: Probabilistically analysable real-time systems. In: . [S.l.: s.n.], 2012.

CAZORLA, F. J. et al. Upper-bounding Program Execution Time with Extreme Value Theory. In: MAIZA, C. (Ed.). *13th International Workshop on Worst-Case Execution Time Analysis*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013. (OpenAccess Series in Informatics (OASIcs), v. 30), p. 64–76. ISBN 978-3-939897-54-5. ISSN 2190-6807.

COLES, S. *An introduction to statistical modeling of extreme values*. London: Springer-Verlag, 2001. (Springer Series in Statistics). ISBN 1-85233-459-2.

COLIN, A.; PUAUT, I. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, v. 18, p. 249–274, 05 2000.

CONWAY, D.; WHITE, J. M. *Machine learning for hackers*. Sebastopol, CA: O'Reilly Media, 2012. ISBN 9781449303716 1449303714.

CUCU-GROSJEAN, L. et al. Measurement-based probabilistic timing analysis for multi-path programs. In: *2012 24th Euromicro Conference on Real-Time Systems*. [S.l.: s.n.], 2012. p. 91–101.

DANIELSSON, J. et al. Modelling application cache behavior using regression models. In: *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. [S.l.: s.n.], 2021. p. 1879–1886.

DAVIS, R. Improvements to static probabilistic timing analysis for systems with random cache replacement policies. In: . [S.l.]: Proceedings of the Real-Time Scheduling Open Problems Seminar, 2013.

DAVIS, R. I.; CUCU-GROSJEAN, L. A survey of probabilistic timing analysis techniques for real-time systems. *LITES: Leibniz Transactions on Embedded Systems*, v. 6, n. 1, p. 1–60, 5 2019. ISSN 2199-2002.

DEVORE, J. L. *Probability and Statistics for Engineering and the Sciences*. 8th. ed. Boston, USA: Brooks/Cole, 2011. ISBN-13: 978-0-538-73352-6.

DIEBOLT, J.; GARRIDO, M.; GIRARD, S. *A Goodness-of-fit Test for the Distribution Tail*. [S.l.: s.n.], 2007. 95-109 p.

DIETRICH, L. D. H. D.; HüSLER, J. Testing extreme value condition. In: . [S.l.: s.n.], 2002. p. 71–85.

DONGARRA, J. et al. Experiences and lessons learned with a portable interface to hardware performance counters. In: *Proceedings International Parallel and Distributed Processing Symposium*. [S.l.: s.n.], 2003.

ELSAYED, N. A. et al. 7. diabetes technology: Standards of care in diabetes—2023. *Diabetes Care*, v. 46, p. S111–S127, 12 2022. ISSN 0149-5992.

FALK, H. et al. TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research. In: SCHOEBERL, M. (Ed.). *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. (Open Access Series in Informatics (OASIcs), v. 55), p. 2:1–2:10.

FAUDZI, A. M. et al. Real-time hand gestures system for mobile robots control. *Procedia Engineering*, v. 41, p. 798–804, 2012. ISSN 1877-7058. International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012). Disponível em: <https://www.sciencedirect.com/science/article/pii/S187770581202646X>.

FELLER, W. *An Introduction to Probability Theory and Its Applications*. [s.n.], 1968. Hardcover. ISBN 0471257087. Disponível em: <https://www.amazon.com/Introduction-Probability-Theory-Applications-Vol/dp/0471257087>.

FERRANDEZ, I. R. et al. Worst case execution time and power estimation of multicore and gpu software: A pedestrian detection use case. *Ada Lett.*, Association for Computing Machinery, New York, NY, USA, v. 43, n. 1, p. 111–117, out. 2023. ISSN 1094-3641. Disponível em: <https://doi.org/10.1145/3631483.3631502>.

FOUNDATION, R. P. *Raspberry Pi OS - Raspberry documentation*. [S.l.], 2018. Disponível em: <https://www.raspberrypi.org/documentation/raspbian/>.

GIL, S. J. Constraint-based testing and tail tests for measurement-based probabilistic timing analysis. PhD thesis. 2020. Disponível em: <https://etheses.whiterose.ac.uk/28767/>.

GIL, S. J. et al. Open challenges for probabilistic measurement-based worst-case execution time. *IEEE Embedded Systems Letters*, 2017.

GRIFFIN, D.; BURNS, A. Realism in statistical analysis of worst case execution times. In: *10th Intl. Workshop on Worst-Case Execution Time Analysis*. [S.l.: s.n.], 2010. p. 49–57.

GRIFFIN, D. et al. Forecast-based interference: Modelling multicore interference from observable factors. In: . New York, NY, USA: Association for Computing Machinery, 2017. (RTNS '17), p. 198–207. ISBN 9781450352864.

GUET, F.; SANTINELLI, L.; MORIO, J. On the Reliability of the Probabilistic Worst-Case Execution Time Estimates. In: *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*. TOULOUSE, France: [s.n.], 2016.

GUET, F.; SANTINELLI, L.; MORIO, J. On the Representativity of Execution Time Measurements: Studying Dependence and Multi-Mode Tasks. In: REINEKE, J. (Ed.). *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. (OpenAccess Series in Informatics (OASIcs), v. 57), p. 3:1–3:13. ISBN 978-3-95977-057-6. ISSN 2190-6807.

GUPTA, A. et al. *How to Incorporate Monotonicity in Deep Networks While Preserving Flexibility?* 2019.

GUSTAFSSON, J. et al. The mälardalen wcet benchmarks: Past, present and future. In: LISPER, B. (Ed.). *WCET*. Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. (OASICS, v. 15), p. 136–146. ISBN 978-3-939897-21-7.

HANSEN, J.; HISSAM, S.; MORENO, G. Statistical-based wcet estimation and validation. In: . [S.l.: s.n.], 2009. v. 10.

HAYKIN, S. *Redes neurais: Principios e prática*. 2nd. ed. [S.l.]: Bookman, 2005. ISBN 9788573077186; 8573077182.

HOLDING, A. *ARM v7-M Architecture Application Level Reference Manual*. [S.l.], 2021. <https://developer.arm.com/documentation/ddi0403/latest/>.

HURST, H. E. Long term storage capacity of reservoirs. *ASCE Transactions*, v. 116, n. 776, p. 770–808, 1951. Disponível em: <https://cir.nii.ac.jp/crid/1571980074853982464>.

JANIESCH, C.; ZSCHECH, P.; HEINRICH, K. Machine learning and deep learning. *Electronic Markets*, v. 31, n. 3, p. 685–695, September 2021. Disponível em: <https://ideas.repec.org/a/spr/elmark/v31y2021i3d10.1007_s12525-021-00475-2.html>.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. An introduction to reinforcement learning. In: STEELS, L. (Ed.). *The Biology and Technology of Intelligent Autonomous Agents*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. p. 90–127.

KERAS: R Interface to 'Keras'. [S.l.], 2024. R package version 2.15.0. Disponível em: <https://cran.r-project.org/web/packages/keras/index.html>.

KERRISK, M. *perf event open — Linux manual page*. [S.l.], 2019. Disponível em: <http://www.man7.org/linux/man-pages/man2>.

KHAZEN, M. W. el et al. Work in progress: Kdbench - towards open source benchmarks for measurement-based multicore WCET estimators. In: *28th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2022, Milano, Italy, May 4-6, 2022*. Milano, Italy: IEEE, 2022. p. 309–312.

Chapter seven - architecture of neural processing unit for deep neural networks. In: KIM, S.; DEKA, G. C. (Ed.). *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*. [S.l.]: Elsevier, 2021, (Advances in Computers, v. 122). p. 217–245.

KOPETZ, W. S. H. *Real-Time Systems: Design Principles for Distributed Embedded Applications 25*. 3. ed. [S.l.]: Springer, 2022. (Real-Time Systems Series). ISBN 9783031119910; 3031119916; 9783031119927; 3031119924.

KOSMIDIS, L. et al. A cache design for probabilistically analysable real-time systems. In: *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. [S.l.: s.n.], 2013. p. 513–518.

KOSMIDIS, L. et al. Pub: Path upper-bounding for measurement-based probabilistic timing analysis. In: *2014 26th Euromicro Conference on Real-Time Systems*. [S.l.: s.n.], 2014. p. 276–287.

KOSMIDIS, L. et al. Probabilistic timing analysis on conventional cache designs. In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. [S.l.: s.n.], 2013. p. 603–606.

KOSMIDIS, L. et al. Applying Measurement-Based Probabilistic Timing Analysis to Buffer Resources. In: MAIZA, C. (Ed.). *13th International Workshop on Worst-Case Execution Time Analysis*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013. (OpenAccess Series in Informatics (OASIcs), v. 30), p. 97–108. ISBN 978-3-939897-54-5. ISSN 2190-6807. Disponível em: <http://drops.dagstuhl.de/opus/volltexte/2013/4126>.

KOTZ, S.; NADARAJAH, S. *Extreme value distributions: Theory and applications*. [S.l.]: Imperial College Press, 2000. (G - Reference,Information and Interdisciplinary Subjects Series). Section 1.7.9. ISBN 9781860942242.

KUMAR, V. Deep neural network approach to estimate early worst-case execution time. In: *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. [S.l.: s.n.], 2021. p. 1–8.

KUMAR, V. An integrated approach of genetic algorithm and machine learning for generation of worst-case data for real-time systems. In: *2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. [S.l.: s.n.], 2022. p. 87–95.

KUTNER, M. et al. *Applied Linear Statistical Models 5th Edition*. [S.l.]: McGraw-Hill Irwin, 2004. (McGraw-Hill Irwin Series Operations and Decision Sciences). ISBN 0072386886; 9780072386882.

KWIATKOWSKI, D. et al. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, v. 54, n. 1, p. 159–178, 1992. ISSN 0304-4076.

LAPLANTE, P. A. *Real-Time Systems Design and Analysis*. 3rd ed. ed. [S.l.]: Wiley, 2004. ISBN 9780471228554; 0471228559.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, p. 436–444, 2015.

LESAGE, B. et al. Exploring and understanding multicore interference from observable factors. In: DENCKER, P. et al. (Ed.). *Automotive - Safety and Security 2017 - Sicherheit und Zuverlässigkeit für automobile Informationstechnik*. Germany: Gesellschaft für Informatik, Bonn, 2017. p. 75–88. Disponível em: <https://dl.gi.de/20.500.12116/148>.

LESAGE, B. et al. A framework for the evaluation of measurement-based timing analyses. In: *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. New York, NY, USA: Association for Computing Machinery, 2015. (RTNS '15), p. 35–44. ISBN 9781450335911. Disponível em: <https://doi.org/10.1145/2834848.2834858>.

LIAW, A.; WIENER, M. Classification and regression by randomforest. *R News*, v. 2, n. 3, p. 18–22, 2002. Disponível em: <https://CRAN.R-project.org/doc/Rnews/>.

LIMA, G.; BATE, I. Valid application of evt in timing analysis by randomising execution time measurements. In: *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. [S.l.: s.n.], 2017. p. 187–198.

LIMA, G.; DIAS, D.; BARROS, E. Extreme value theory for estimating task execution time bounds: A careful look. *ECRTS*, p. 200–211, 2016.

LIU, W. et al. A survey of deep neural network architectures and their applications. *Neurocomputing*, v. 234, p. 11–26, 2017. ISSN 0925-2312.

LIU, X. et al. *Certified Monotonic Neural Networks*. 2022.

LU, Y. et al. A new way about using statistical analysis of worst-case execution times. *SIGBED Rev.*, Association for Computing Machinery, New York, NY, USA, v. 8, n. 3, p. 11–14, sep 2011. Disponível em: <https://doi.org/10.1145/2038617.2038619>.

MALONE, C.; ZAHRAN, M.; KARRI, R. Are hardware performance counters a cost effective way for integrity checking of programs. In: *Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing*. New York, NY, USA: Association for Computing Machinery, 2011. (STC '11), p. 71–76. ISBN 9781450310017.

MAXIM, C. et al. Reproducibility and representativity: mandatory properties for the compositionality of measurement-based wcet estimation approaches. *SIGBED Rev.*, Association for Computing Machinery, New York, NY, USA, v. 14, n. 3, 2017.

MEIER, L. *PX4 Development Guide*. [S.l.], 2023. <https://dev.px4.io/en/>.

MEYER, D. et al. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. [S.l.], 2023. R package version 1.7-13. Disponível em: <https://CRAN.R-project.org/package=e1071>.

MOLKA, D. et al. Detecting memory-boundedness with hardware performance counters. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, 2017. (ICPE '17), p. 27–38. ISBN 9781450344043. Disponível em: <https://doi.org/10.1145/3030207.3030223>.

MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G. *Introduction to Linear Regression Analysis (4th ed.)*. New Jersey: Wiley & Sons, 2006.

NUTTX, G. *NuttX Operating System, User's Manual*. [S.l.], 2023. <http://www.nuttx.org/doku.php?id=documentation:userguide>.

OLIVEIRA, R. de. *Fundamentos Dos Sistemas de Tempo Real*. Independently Published, 2018. ISBN 9781728694047. Disponível em: <https://books.google.com.br/books?id=I2umyQEACAAJ>.

PINTO, M. L.; WEHRMEISTER, M. A.; OLIVEIRA, A. S. de. Real-time performance evaluation for robotics. *Journal of Intelligent & Robotic Systems*, v. 101, n. 2, p. 37, Feb 2021. ISSN 1573-0409. Disponível em: <https://doi.org/10.1007/s10846-020-01301-1>.

PISNER, D. A.; SCHNYER, D. M. Chapter 6 - support vector machine. In: MECHELLI, A.; VIEIRA, S. (Ed.). *Machine Learning*. [S.l.]: Academic Press, 2020. p. 101–121. ISBN 978-0-12-815739-8.

POLIKAR, R.; ZHANG, C.; MA, Y. *Ensemble Machine Learning: Methods and Applications*. 1. ed. New York, NY: Springer, 2012. ISBN 9781441993250.

PUJOL, R. et al. Event monitor validation in high-integrity systems. In: *2024 27th Euromicro Conference on Digital System Design (DSD)*. [S.l.: s.n.], 2024. p. 394–402.

QIAN, B.; RASHEED, K. M. Hurst exponent and financial market predictability. In: . [S.l.: s.n.], 2005.

QUINONES, E. et al. Using randomized caches in probabilistic real-time systems. In: *2009 21st Euromicro Conference on Real-Time Systems*. [S.l.: s.n.], 2009. p. 129–138.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2020. Disponível em: <https://www.R-project.org/>.

RANJBAR, B. et al. Improving the timing behaviour of mixed-criticality systems using chebyshev's theorem. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. [S.l.: s.n.], 2021. p. 264–269.

RASPBERRY, F. *Raspberry Pi 3 Model B v.1.2 (Made in UK and Made in PRC versions)*. [S.l.], 2018. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.

REGHENZANI, F. et al. Probabilistic-wcet reliability: On the experimental validation of evt hypotheses. In: *Proceedings of the International Conference on Omni-Layer Intelligent Systems*. New York, NY, USA: Association for Computing Machinery, 2019. (COINS '19), p. 229–234. ISBN 9781450366403.

ROCHANGE, C.; UHRIG, S.; SAINRAT, P. Timing analysis of real-time systems. In: _____. *Time-Predictable Architectures*. [S.l.]: John Wiley & Sons, Ltd, 2013. cap. 2, p. 19–36. ISBN 9781118790229.

ROSS, S. M. *Introduction to probability models*. Tenth. [S.l.]: Academic Pres, 2009.

RUNJE, D.; SHARATH, S. *Constrained Monotonic Neural Networks*. 2023.

SANDELL, D. et al. Static timing analysis of real-time operating system code. In: MARGARIA, T.; STEFFEN, B. (Ed.). *Leveraging Applications of Formal Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 146–160. ISBN 978-3-540-48929-0.

SANTINELLI, L.; GUET, F.; MORIO, J. Revising measurement-based probabilistic timing analysis. In: *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. [S.l.: s.n.], 2017. p. 199–208.

SARKER, I. H. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, v. 2, n. 6, p. 420, Aug 2021. ISSN 2661-8907.

SERHANI, M. A. et al. Ecg monitoring systems: Review, architecture, processes, and key challenges. *Sensors*, v. 20, n. 6, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/6/1796>.

SILL, J.; ABU-MOSTAFA, Y. Monotonicity hints. In: MOZER, M.; JORDAN, M.; PETSCHE, T. (Ed.). *Advances in Neural Information Processing Systems*. MIT Press, 1996. v. 9. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/1996/file/c850371fda6892fbfd1c5a5b457e5777-Paper.pdf>.

THEOBALD, O. *Machine Learning for Absolute Beginners: A Plain English Introduction (Third Edition)*. 6. ed. London, UK: Scatterplot Press, 2020.

TINETTI, F. G.; MéNDEZ, M. An automated approach to hardware performance monitoring counters. In: *2014 International Conference on Computational Science and Computational Intelligence*. [S.l.: s.n.], 2014. v. 1, p. 71–76.

VASCONCELOS, J. de B.; LIMA, G. Possible risks with evt-based timing analysis: an experimental study on a multi-core platform. In: *2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2022. p. 1–8.

VERA, X.; LISPER, B.; XUE, J. Data cache locking for higher program predictability. In: *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. [s.n.], 2003. p. 272–282. Disponível em: <http://www.es.mdh.se/publications/438->.

V.G. Baranoski, G.; G. Rokne, J.; XU, G. Applying the exponential chebyshev inequality to the nondeterministic computation of form factors. *Journal of Quantitative Spectroscopy and Radiative Transfer*, v. 69, n. 4, p. 447–467, 2001. ISSN 0022-4073.

VIEIRA, S. T. et al. Q-meter: Quality monitoring system for telecommunication services based on sentiment analysis using deep learning. *Sensors*, v. 21, n. 5, 2021. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/21/5/1880>.

VILARDELL, S. et al. Using markov's inequality with power-of-k function for probabilistic wcet estimation. In: MAGGIO, M. (Ed.). *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. (Leibniz International Proceedings in Informatics (LIPIcs), v. 231), p. 20:1–20:24. ISBN 978-3-95977-239-6. ISSN 1868-8969.

WEAVER, V.; MCKEE, S. Can hardware performance counters be trusted? In: . [S.l.: s.n.], 2008. p. 141 – 150.

WILHELM, R. et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 7, n. 3, may 2008. ISSN 1539-9087. Disponível em: <https://doi.org/10.1145/1347375.1347389>.

XU, Y.; GOODACRE, R. On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. In: STEELS, L. (Ed.). Berlin, Heidelberg: J Anal Test, 2018.

YILMAZ, C. Using hardware performance counters for fault localization. In: *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*. [S.l.: s.n.], 2010. p. 87–92.

ZAPARANUKS, D.; JOVIC, M.; HAUSWIRTH, M. Accuracy of performance counter measurements. In: *IEEE International Symposium on Performance Analysis of Systems and Software*. [S.l.: s.n.], 2009. p. 23–32.

ZHANG, F.; O'DONNELL, L. J. Chapter 7 - support vector regression. In: MECHELLI, A.; VIEIRA, S. (Ed.). *Machine Learning*. [S.l.]: Academic Press, 2020. p. 123–140. ISBN 978-0-12-815739-8.

ZICCARDI, M. et al. Epc: Extended path coverage for measurement-based probabilistic timing analysis. In: *2015 IEEE Real-Time Systems Symposium*. [S.l.: s.n.], 2015. p. 338–349.

ZOLDA, M.; BÜNTE, S.; KIRNER, R. Towards adaptable control flow segmentation for measurement-based execution time analysis. In: *Procs 17th Int Conf on Real-Time and Network Systems*. [S.l.: s.n.], 2009. p. 35–44. Int Conf on Real-Time and Network Systems ; Conference date: 26-10-2009 Through 27-10-2009.

Apêndice

A

MODELANDO O TEMPO DE EXECUÇÃO EM UMA ARQUITETURA MONONÚCLEO

A seguir, é apresentada a modelagem do tempo de execução de um programa com base em eventos de hardware, utilizando um microcontrolador mononúcleo ARMv7 Cortex-M4. O programa selecionado para monitoramento é parte integrante de um conjunto de programas de controle de voo de piloto automático, provenientes do *benchmark* KD-Bench (KHAZEN et al., 2022). Esse estudo foi conduzido na UFBA em parceria com o INRIA (*Institut National de Recherche en Informatique et en Automatique - France*), no âmbito do projeto Kepler.

A.1 PLATAFORMA E *BENCHMARK*

O processador ARMv7 Cortex-M4 utilizado no experimento possui somente um núcleo e uma frequência de 180MHz, com 256KB de SRAM, integrado na placa Pixhawk. A Pixhawk é a placa mais comum para drones com piloto automático PX4. O programa analisado é baseado no piloto automático PX4-RT, que foi obtido por meio de modificações no piloto automático PX4 original. Este último é um piloto automático de controle de voo de código aberto (MEIER, 2023) desenvolvido na ETH Zurich.

Tabela A.1 Eventos disponiveis para monitoramento no ARMv7-M.

Endereço	**Nome**	**Descrição**
$0xE0001004$	CYCCNT	Total de ciclos gastos
$0xE0001008$	CPICNT	Total de ciclos para executar instruções multi-ciclo
$0xE000100C$	EXCCNT	Total de ciclos para processar as exceções
$0xE0001010$	SLEEPCNT	Total de ciclos gastos quando a CPU está em repouso
$0xE0001014$	LSUCNT	Total de ciclos para executar as instruções de *load* e *store*
$0xE0001018$	FOLDCNT	Incrementado quando a instrução leva 0 ciclos

Na Tabela A.1, estão listados os seis eventos de hardware disponíveis para monitoramento, os quais todos podem ser observados simultaneamente. Para detalhes adicionais

sobre esses eventos, é possível consultar o manual de referência da arquitetura (HOLDING, 2021). O ARMv7-M possui um componente, denominado de *Data Watchpoint and Trace Unit* (DWT) que, uma vez configurado, atua como a PMU, permitindo a leitura de cada evento a partir do respectivo endereço de memória ao término da execução.

O sistema operacional utilizado é o NuttX, uma implementação baseada no Unix criada por Gregory Nutt (NUTTX, 2023). Os programas são agendados seguindo um esquema de prioridade fixa. Se houver vários programas com a mesma prioridade, a política de agendamento FIFO é aplicada. O monitoramento dos eventos é efetuado mediante modificações no código-fonte original do programa, escrito em linguagem `C++`. O PX4-RT contém nove programas fundamentais para o controle do drone. Dado que o monitoramento isolado de tarefas não é possível, o programa relacionado aos sensores do drone (*Sensor*) foi escolhido para ser observado, uma vez que possui a prioridade mais alta.

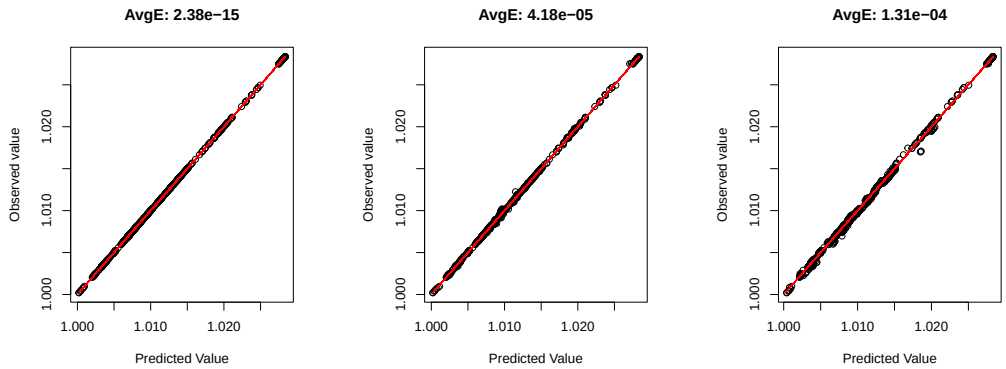
A.2 RESULTADOS OBTIDOS

Conforme a Tabela A.1, a arquitetura ARMv7 Cortex-M4 possui apenas seis eventos relacionados ao hardware. Durante o procedimento, foi observado que os valores do evento `SLEEPCNT` eram consistentemente zero, o que permite excluí-lo da modelagem. Portanto, o evento `CYCCNT` é o que está sendo modelado, enquanto `LSUCNT`, `FOLDCNT`, `EXCCNT` e `CPICNT` representam as variáveis independentes usadas para modelar `CYCCNT`. Todos os eventos são normalizados pelo número de instruções, conforme explicado na Seção 4.2.

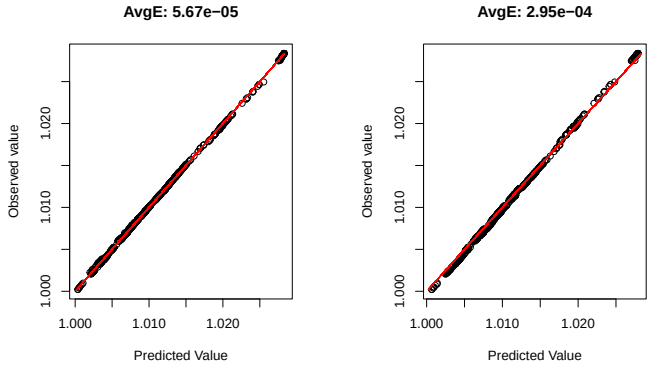
O programa *Sensor* é executado 7.659 vezes, sendo que 75% dos dados são utilizados na etapa de treinamento (selecionados aleatoriamente) e 25% são reservados para avaliar a qualidade dos modelos. A Figura A.1 ilustra a relação entre os valores observados e os valores previstos para o *execution pace*. Uma previsão perfeita seria representada por pontos seguindo a linha vermelha. Como pode ser visto, todos os modelos ML conseguem prever com precisão o comportamento do programa, mesmo com apenas quatro eventos disponíveis. A MLR alcançou um AvgE de $2{,}38 \times 10^{-15}$, sendo o valor mais baixo entre todas as ferramentas utilizadas.

A.3 CONSIDERAÇÕES FINAIS

Os resultados obtidos para a plataforma ARMv7 Cortex-M4 indicam que, se for possível medir todos os eventos associados ao hardware e as instruções executadas, é possível prever o *execution pace* com um alto grau de precisão. Contudo, na prática, plataformas mais complexas podem apresentar desafios para a modelagem, como foi observado no Capítulo 5 com a plataforma ARM Cortex-A53.



(a) MLR (b) RF (c) MNN



(d) DNN (e) SVR

Figura A.1 Relação entre os valores observados e os valores previstos para as estimativas do *execution pace*. Modelos baseados em ML são desenvolvidos para o programa *Sensor*, executado na plataforma ARMv7 Cortex-M4.: (a) MLR; (b) RF; (c) MNN; (d) DNN; (e) SVR.

Apêndice

B

INTERVALOS DE CONFIANÇA DAS ESTIMATIVAS DE PWCET APRESENTADAS NO CAPÍTULO 5

A seguir, são apresentadas as estimativas de pWCET e os respectivos intervalos de confiança (IC) associados aos dados das Tabelas 5.1, 5.2 e 5.4, presentes no Capítulo 5.

Tabela B.1 Estimativas de pWCET e respectivos intervalos de confiança (95%) para os dados da Tabela 5.1, considerando os casos (a) e (b) e diferentes níveis de probabilidade.

	Estimativas	**Intervalos de Confiança (95%)**
Caso (a):		
10^{-2}	174.782	[147.735, 190505]
10^{-4}	188.798	[110.489, 280.780]
Caso (b):		
10^{-2}	49.996	[44.444, 66.146]
10^{-4}	75.442	[53.362, 171.262]

Tabela B.2 Estimativas de pWCET e respectivos intervalos de confiança (95%) para os dados ajustados às distribuições GP e GEV, conforme a Tabela 5.2, considerando diferentes níveis de probabilidade.

	Estimativas	**Intervalos de Confiança (95%)**
GEV:		
10^{-2}	142.653	[130.715, 154.548]
10^{-4}	187.545	[116.325, 246.459]
GP:		
10^{-2}	145.615	[133.780, 157.447]
10^{-4}	174.734	[132.403, 216.848]

Tabela B.3 Estimativas de pWCET e intervalos de confiança (95%) para os dados apresentados na Tabela 5.4.

Programas	**pWCET$_A$**		**pWCET$_B$**	
	Estimativas	**Intervalo (95%)**	**Estimativas**	**Intervalo (95%)**
1 Recursion	368.824	[130.930, 606.731]	218.773	[133.820, 428.640]
2 Fft1	490.444	[198.372, 782.517]	448.310	[240.350, 651.798]
3 Cnt	2.548.313	[2.252.220, 2.844.405]	2.659.193	[2.215.099, 3.103.287]
4 Cover	245.566	[191.062, 300.069]	240.261	[152.577, 327.945]
5 BSearch	95.168	[41.267, 148.832]	112.182	[91.530, 133.677]
6 Sqrt	106.089	[47.407, 165.806]	92.134	[47.760, 137.033]
7 FibCall	1.161.595	[1.129.627, 1.193.686]	1.134.217	[1.093.483, 1.174.943]
8 Matmult	7.691.013	[6.696.537, 8.685.489]	7.557.081	[4.722.961, 10.391.200]
9 Insert Sort	3.056.232	[2.995.980, 3.159.285]	3.025.526	[2.976.004, 3.216.275]
10 Quick Sort	1.054.955	[736.474, 1.373.428]	1.073.299	[685.859, 1.460.739]
11 Edn	426.861	[345.317, 508.406]	499.105	[244.286, 706.430]
12 ADPCM Dec.	18.386.052	[18.305.673, 18.466.431]	18.550.155	[18.496.820, 18.603.493]
13 JPEG Transc.	8.825.930	[8.655.412, 8.996.446]	8.808.049	[8.650.688, 8.965.411]
14 Dijkstra	73.771.601	[67.292.404, 80.250.619]	91.518.800	[64.401.415, 118.636.210]
15 Huffman Enc.	4.374.557	[3.525.455, 5.223.644]	4.548.237	[3.531.022, 5.565.475]
16 Statemate	1.018.895	[900.731, 1.137.059]	964.522	[906.961, 1.022.086]