Análise de técnicas de predição de desvio sob a arquitetura RISC-V*

Lucas Arruk Mendes[1] **, Maurício Figueiredo**[1] **, Ricardo Menotti**[1]

[1] Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Rod. Washington Luís, Km 235 (SP-310) CEP 13565-905 – São Carlos – SP

lucasarruk@estudante.ufscar.br, {mauricio, menotti}@ufscar.br

Abstract. *Branch prediction is a critical technique for enhancing the performance of pipelined processors by minimizing the penalties associated with control hazards. This paper presents a comparative analysis of branch prediction models implemented in a RISC-V processor architecture, focusing on Bimodal, Gselect, and Gshare predictors. Each model was designed, implemented, and tested on an FPGA platform to evaluate its accuracy, resource utilization, and impact on overall processor performance.*

Resumo. *A previsão de desvio é uma técnica crítica para melhorar o desempenho de processadores em pipeline minimizando as penalidades associadas aos hazards de controle. Este artigo apresenta uma análise comparativa de modelos de previsão de desvio implementados em uma arquitetura de processador RISC-V, com foco em preditores bimodais, Gselect e Gshare. Cada modelo foi projetado, implementado e testado em uma plataforma FPGA para avaliar sua precisão, utilização de recursos e impacto no desempenho geral do processador.*

1. Introdução

A previsão de desvios em processadores é um dos principais desafios na busca por desempenho otimizado em arquiteturas *pipeline*. Quando uma instrução de desvio é encontrada, a incerteza sobre o caminho que será seguido pode causar penalidades significativas se o *pipeline* precisar ser corrigido. Isso acontece porque, até que o desvio seja resolvido, o processador pode continuar a buscar e decodificar instruções que, eventualmente, precisarão ser descartadas, resultando em perda de ciclos e redução do desempenho.

Neste contexto, as técnicas de predição de desvios tornam-se essenciais, pois permitem que o processador antecipe o resultado do desvio com base em padrões de execução anteriores, minimizando as penalidades de controle. Diversas abordagens de predição foram desenvolvidas ao longo dos anos, variando desde métodos mais simples e estáticos, até modelos dinâmicos e adaptativos, que ajustam suas previsões conforme a execução do programa avança.

Este trabalho foca na análise e comparação de três técnicas de predição de desvios em uma arquitetura RISC-V: Bimodal, Gselect e Gshare. Cada uma dessas técnicas apresenta características distintas em termos de complexidade, utilização de recursos de

*O presente trabalho foi realizado com apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), Brasil. Processo nº 2024/01738-8

hardware e precisão das predições. Para realizar essa análise, as técnicas foram implementadas em uma plataforma FPGA, onde sua eficácia foi medida em termos de acurácia, utilização de recursos e impacto no desempenho geral do processador.

O objetivo principal deste estudo é entender as vantagens e limitações de cada técnica de predição no contexto de uma arquitetura RISC-V, proporcionando *insights* que possam guiar o desenvolvimento de processadores mais eficientes e de alto desempenho. A seguir, são apresentados os detalhes da implementação do processador, as métodos adotados para a análise e os resultados obtidos a partir dos testes realizados.

2. Visão geral do processador

Os mecanismos de previsão de desvios descritos neste artigo foram incorporados a um processador RISC-V, com conjunto de instruções RV32I (Waterman, Lee, Patterson, and Asanovic, 2014, chap. 2), ou seja, abrange apenas instruções capazes de realizar cálculos com inteiros. A descrição do hardware foi feita em Verilog, baseada na implementação de Lévy (2020). A CPU é escalar com execução em ordem e possui 5 estágios de *pipeline*. Tanto a memória de instruções quanto a de dados são armazenadas em BRAM com latência de acesso de um único ciclo, mas nenhuma outra memória está disponível e, portanto, nenhuma *cache* foi usada.

Os cinco estágios de *pipeline* do processador em questão são: busca, decodificação, execução, acesso à memória e escrita. Naturalmente, quando uma instrução de desvio condicional entra no *pipeline* os dados referentes ao desvio (valor lógico da decisão e o endereço) serão produzidos no estágio de execução, já que é onde a ULA é requisitada. Porém, existe no estágio de decodificação um módulo de soma, para que os endereços de instruções de desvios incondicionais possam estar disponíveis antes de uma nova busca, evitando o hazard de controle. Assim, todas as técnicas que serão utilizadas compartilham deste módulo, com suas devidas particularidades.

3. Técnicas de predição de desvios

Um dos fatores que mais causa atrasos durante execuções de programas em processadores *pipeline* é o desvio condicional. Isso se dá pelo fato de que o resultado da decisão de desvio torna-se disponível apenas no estágio de execução. Logo, caso o desvio seja tomado, é preciso adicionar bolhas nos estágios anteriores, evitando que instruções indevidas sejam executadas. É possível considerar esta dinâmica nativa do *pipeline* como uma estratégia de predição, ou seja, o *pipeline* é alimentado como se o desvio nunca fosse tomado. Entretanto, na maioria dos cenários, os desvios são tomados em maior frequência, logo parece ser mais vantajoso utilizar uma estratégia de predição que sempre desvia.

Porém, detendo-se de forma mais aprofundada na análise da dinâmica de execução das estruturas de repetição (*loop*), observa-se que a maioria dos desvios tomados nestes casos transferem a execução para um endereço anterior. Portanto, sob este ponto de vista, é mais interessante sempre desviar quando o endereço de desvio é para uma posição anterior de memória. Esta estratégia é chamada de BTFNT (*backwards taken, forward not taken*) e pode ser aplicada facilmente através do *bit* mais significativo do valor imediato presente na instrução de desvio, que, sendo 1, indica um valor negativo a ser somado ao endereço atual, de maneira que o resultado referencia um endereço anterior.

As estratégias descritas anteriormente são estáticas, ou seja, não mudam de acordo com a execução do código, o que impossibilita a correção de erros de predições feitas durante a execução. Ainda assim, existem outras estratégias, dinâmicas, que se adaptam de acordo com o código em diversas maneira possíveis: desde as mais simples, estruturadas em níveis com tabelas de um ou dois *bits*, até as mais complexas, tais como: Gselect, Gshare, *agree* e *hybrid*. Porém, entre estas duas categorias, estáticas e dinâmicas, existem diversos *trade-offs* a serem analisados, principalmente entre o custo do hardware e o desempenho obtido.

O modelo mais básico de predição adaptativa é o Bimodal (Smith, 2000), feito através de tabelas de predição que contêm informações sobre a última decisão de desvio de um dado endereço correspondente à instrução. Desta forma, é possível armazenar um histórico de desvios, associados a um endereço, de maneira que o sistema se adapta ao longo da execução do código. Estas informações do histórico das decisões de desvio podem ser representadas através de um *bit*: caso o desvio seja tomado, associa-se 1 ao endereço da instrução, caso contrário, 0. Assim, a tabela de predições consegue prever dois padrões: desviar ou não desviar, adaptando-se entre as duas opções. Algumas estruturas não permitem que o histórico de decisão possa ser representado com um único *bit*. No caso do *loop*, por exemplo, após o padrão de repetição ter sido registrado, em algum momento o *loop* é encerrado, promovendo uma alteração no padrão que deve ser considerado no histórico. Assim, para aperfeiçoar a estratégia de decisão se faz necessário expandir cada entrada da tabela para dois *bits* (valor ideal analisado por Smith (2000)) de modo que um único desvio dissonante do padrão não prejudica a previsão indicada na tabela. Para tal, a predição é associada ao *bit* mais significativo, e os valores da tabela de predição são atribuidos a um somador saturado. Se o devio for tomado, incrementa-se o valor da tabela em 1, mantendo-se fixo se o valor presente na tabela é 0b11, caso contrário, decrementa-se em 1, mantendo-se fixo se o valor é 0b00. Desta maneira, um único desvio dissonante do padrão não prejudica a previsão indicada na tabela.

A estratégia Gselect(Pan, So, and Rahmeh, 1992) permite superar as dificuldades da estratégia Bimodal. Neste modelo a forma de indexação da tabela de predições é alterada. Nela o histórico de desvios (com X *bits*) também é considerado, de forma que o índice é formado através da concatenação de $N \leq X$ *bits* do histórico de desvio com M *bits* do contador do programa, i.e. o endereço da instrução de desvio. Desta maneira, cada histórico possui uma entrada diferente na tabela de predição, possibilitando tomadas de decisão de forma mais específica. Os valores de N e M dependem de diversos fatores, principalmente do tamanho da tabela de desvios, já que $N+M$ deve ser igual à quantidade de entradas na tabela.

O modelo acima traz uma solução concatenando dois valores, resultando em perda de informação, já que o índice da tabela de desvio é maior tanto do contador de programa, quanto do histórico de desvio. Assim, de maneira a otimizar as informações disponíveis, é proposto por McFarling (1993) o uso de um *Hash* entre as duas entradas anteriores, modelo convencionado como Gshare. Uma porta lógica XOR é usada para a indexação da tabela de desvio, com o mesmo número de *bits* do histórico e do contador de programas, proporcionando um aumento da quantidade de informação global e, consequentemente, diminuindo redundâncias causadas pelo truncamento das informações.

4. Materiais e métodos

A implementação das técnicas de predição, no processador *pipeline* em questão, foi feita no estágio de decodificação. Desta forma, no segundo estágio são necessários os devidos cálculos de endereço do desvio, montado a partir da soma do contador de programas (presente no registrador de *pipeline*) com o imediato obtido a partir da instrução de desvio. Assim, quando a decisão de desvio é tomada, o endereço de desvio calculado é selecionado para a busca da próxima instrução. Porém, independentemente da técnica utilizada, é preciso corrigir o desvio caso ocorra um erro de predição, então o valor lógico da predição é passado adiante nos registradores de *pipeline*. Logo, no estágio de execução são feitos os devidos cálculos para então compará-los com a predição feita. No caso de acerto da predição (predição igual a decisão) o *pipeline* continua a sua execução, caso contrário é necessário desviar o contador de programas para o endereço correto e esvaziar as duas instruções presentes nos estágios de busca e decodificação.

De maneira mais específica, todos os modelos dinâmicos implementados necessitam da tabela de desvio (de tamanho variado e referenciado pelo número de *bits* do endereçamento, e. g. 5 *bits* equivalem a $2^5 = 32$ posições), as diferenças residem apenas na indexação da tabela. Porém, o modelo Bimodal/1*bit* utiliza uma tabela com 1 *bit* por posição e nos outros modelos cada posição possui 2 *bits*. No modelo Bimodal, o índice de busca na tabela é montado a partir dos 30 *bits* mais significativos do contador de programas, já que, qualquer variação nos dois *bits* menos significativos referenciam a mesma instrução.

No modelo Gselect, existe uma grande diversidade de implementações, pois quaisquer valores para N e M podem ser adotados, desde que produzam um índice de (N+M) *bits* igual à quantidade de *bits* de endereçamento da tabela em questão. Por outro lado, no modelo Gshare a indexação é mais direta, ou seja, é resultante de uma operação XOR *bit* a *bit* entre os N *bits* do contador de programas (já desconsiderando os dois primeiro *bits*) e os N *bits* do histórico de desvios.

Desta forma, foram implementadas quatro variações do processador, cada um deles possui como única diferença o tratamento das predições de desvio, e serão referenciados de acordo com o modelo implementado: BTFNT, Bimodal/1*bit*, Bimodal/2*bits*, Gselect e Gshare. A simulação foi realizada através do software Verilator, e para tal foram utilizados três *benchmarks*: (i) Dhrystones; (ii) CoreMark e (iii) Raystones.

4.1. Resultados

Foram analisados as seguintes informações de desempenho: ciclos por execução, instruções por execução e porcentagem de acertos das previsões de desvio. No que se refere à apresentação de resultados, apenas os melhores desempenhos são inidicados dentre todos obtidos a partir das variações de N, M e dimensão da tabela.

Na Figura 1 são apresentados os resultados de comparação com respeito ao críterio de porcentagem de acertos para todas as estratégias e *benchmarks*. A porcentagem de acertos é calculada por meio da divisão entre o número de acertos pela quantidade de instruções de desvio. Constata-se que as estratégias mais robustas refletem em mais acertos nos *benchmarks* Raystones e Dhrystones. Para o Coremark nota-se um resultado que não correspondente à expectativa.

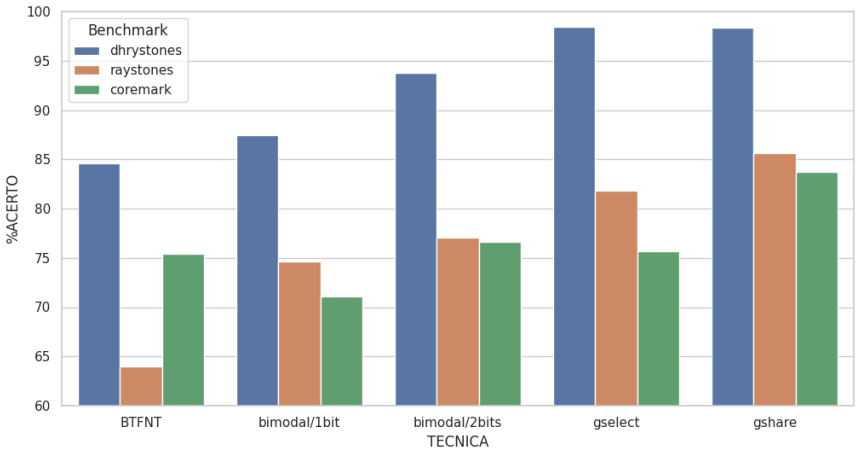

Figura 1. Porcentagem de acertos por modelo em cada *benchmark*

Os benchmarks Dhrystones, CoreMark e Raystones apresentam características distintas que podem influenciar significativamente a taxa de acerto das técnicas de predição de salto. O Dhrystones é um benchmark que mede o desempenho de operações comuns em linguagens de programação, como chamadas de funções, *loops* e acessos a variáveis. Devido à sua natureza focada em operações de controle de fluxo, como *loops* e funções, a predição de desvio tem um impacto direto, especialmente para técnicas que se beneficiam de padrões repetitivos, como o preditor Bimodal ou o de dois níveis.

O CoreMark, por outro lado, é mais focado em medir o desempenho de sistemas embarcados, testando operações que incluem manipulação de listas, operações de matrizes e algoritmos de busca. Esse benchmark pode apresentar uma taxa de acerto de predição de salto mais alta em técnicas que conseguem detectar padrões específicos ou comportamentos previsíveis em *loops* curtos e recursivos. No entanto, sua diversidade de operações pode introduzir variações que afetam negativamente técnicas de predição mais simples.

Por fim, o Raystones, sendo um benchmark que simula traçados de raios luminosos (computação gráfica), envolve muitas operações de aritmética de ponto flutuante e decisões baseadas em condições complexas. Esse tipo de aplicação pode introduzir saltos menos previsíveis, dificultando a tarefa de preditores simples ou até mesmo de alguns preditores mais sofisticados que dependem de padrões facilmente identificáveis. Técnicas mais avançadas tendem a ter melhor desempenho neste cenário, pois podem capturar as complexidades e as interações das operações aritméticas através das decisões de controle.

Na Figura 2 são apresentados os resultados de comparação com respeito ao críterio de CPI para todas as estratégias e *benchmarks*. O CPI é calculado por meio da divisão entre o número de ciclos pela quantidade de instruções. Constata-se que as estratégias mais robustas refletem em mais acertos nos *benchmarks* Raystones e Dhrystones. Para o Coremark nota-se um resultado que não correspondente à expectativa, inclusive refletindo o mesmo resultado do critério anterior.

Na Figura 3 o principal objetivo é apresentar o comportamento da variação de desempenho de cada estratégia com respeito à variação da dimensão da tabela com respeito

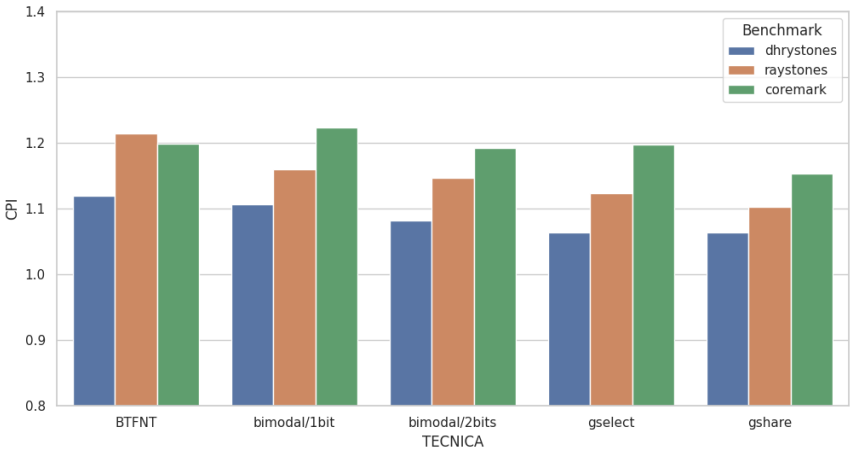

Figura 2. CPI por modelo em cada *benckmark*

do critério de porcentagem de acertos. Constata-se que nos três benchmarks apresentados há um ponto de saturação a partir do qual o aumento no tamanho da tabela não resulta em melhorias significativas na taxa de acertos, ou seja, uma vez atingido um tamanho ótimo da tabela, a precisão da predição não se beneficia de tabelas maiores, sugerindo que há um limite para o ganho de desempenho.

A diferença no ponto de saturação do tamanho da tabela de predição entre os diferentes métodos de predição de saltos — Bimodal, Gselect e Gshare — está relacionada à complexidade e à sofisticação de cada técnica.

No caso do preditor Bimodal, que é uma técnica mais simples, o ponto de saturação ocorre mais cedo. Isso se deve ao fato de que o Bimodal utiliza uma tabela de predição que baseia suas decisões principalmente em uma pequena quantidade de informações, geralmente apenas parte do endereço do salto. Como resultado, uma vez que a tabela atinge um certo tamanho, adicionar mais entradas não contribui significativamente para melhorar a precisão da predição, pois as informações adicionais disponíveis para o Bimodal não são suficientes para distinguir efetivamente entre padrões complexos de saltos.

Por outro lado, o Gselect, que é um método intermediário, utiliza tanto o endereço do salto quanto parte do histórico global dos saltos recentes para fazer a predição. Isso permite que ele capture padrões de comportamento mais complexos em comparação com o Bimodal. Assim, o ponto de saturação para o Gselect ocorre em um tamanho de tabela maior do que o do Bimodal, pois o método se beneficia de uma tabela maior que pode armazenar mais combinações de histórico e endereços, melhorando a precisão até que um limite seja atingido.

Finalmente, o Gshare, que é a técnica mais sofisticada das três, usa uma abordagem que combina o histórico global de saltos com o endereço do salto de uma forma ainda mais eficiente, utilizando a operação de XOR. Isso permite que o Gshare aproveite melhor as informações disponíveis e, portanto, ele continua a se beneficiar de tabelas de predição maiores por mais tempo, atingindo seu ponto de saturação em um tamanho de tabela maior do que o Bimodal e o Gselect. O Gshare consegue capturar padrões de

predição mais complexos e variados, o que explica por que seu limite de saturação é o mais alto entre os três métodos.

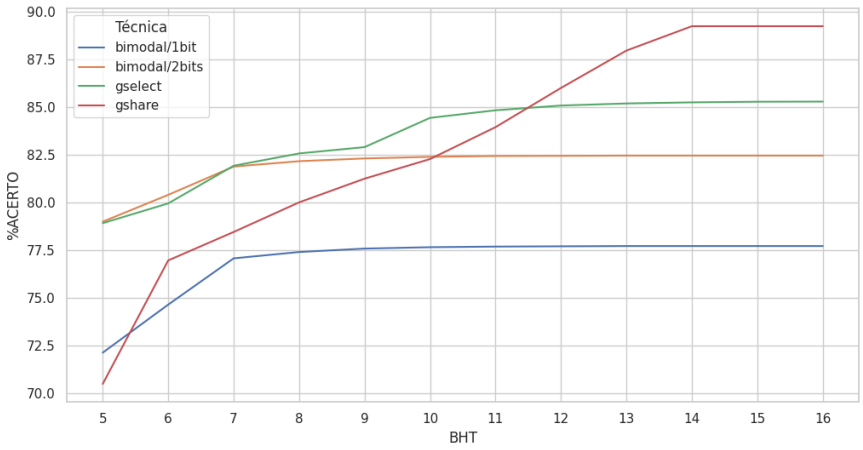

Figura 3. Porcentagem de acertos por modelo em tabelas com 5 a 16 *bits* de endereço (média dos três *benchmarks*)

Essa saturação na taxa de acertos de predição tem um impacto direto no desempenho global do sistema, medido em termos de CPI. A Figura 4 demonstra que, à medida que o tamanho da tabela de predição aumenta, o CPI inicialmente diminui, indicando um melhor desempenho do processador, uma vez que menos ciclos são necessários para completar cada instrução. No entanto, semelhante à taxa de acertos, o CPI também tende a saturar a partir de um certo ponto. Isso ocorre porque, apesar de uma tabela maior potencialmente oferecer uma maior taxa de acertos, outras limitações no *pipeline* do processador e a natureza das instruções processadas limitam o impacto do aumento da tabela sobre o CPI.

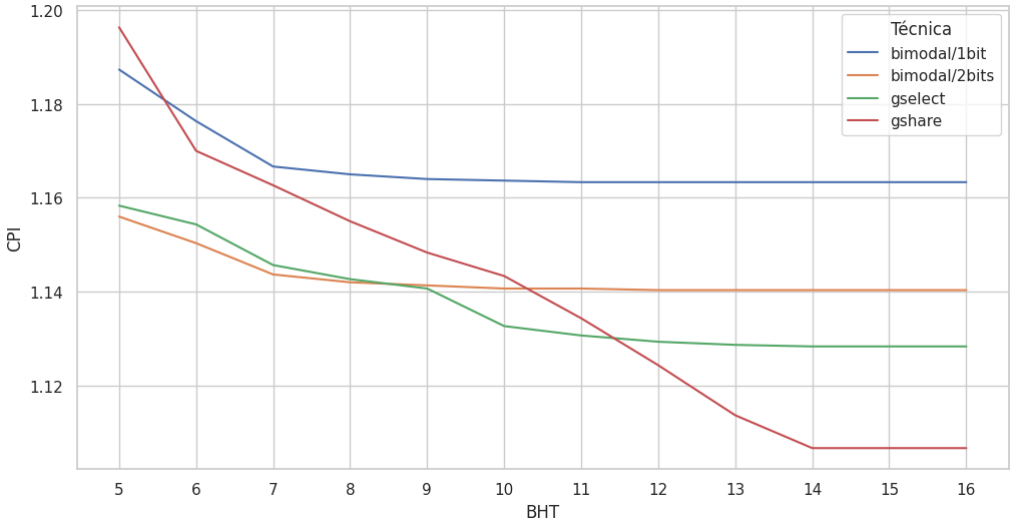

Figura 4. CPI de acordo com a variação da Tabela de Desvios

A síntese das implementações foi feita para o kit DE10-Standard da Terasic, o

qual possui um FPGA da Altera Cyclone V (5CSXFC6D6F31C6) Terasic Technologies (2017) para a obtenção de informações relevantes sobre uso do hardware. Na Tabela 1 são listados os valores de ALMs (Adaptative Logic Modules), *bits* de memória e blocos de RAM para cada um dos modelos considerando o valor máximo de cada tabela de histórico de desvio (2^{16} *bits*).

Tabela 1. Recursos por Implementação

Técnica	ALM	mem *bits*	RAM blk
BTFNT	729	1050624	130
Bimodal/1*bit*	731	1116160	138
Bimodal/2*bits*	739	1247232	154
Gselect	744	1247232	154
Gshare	4308	1058816	131

5. Conclusões

Neste trabalho, analisamos o impacto de diferentes técnicas de predição de salto – Bimodal, Gselect e Gshare – focando em como o tamanho das tabelas de predição afeta tanto a taxa de acertos quanto o desempenho geral do sistema. Observamos que o preditor Bimodal, devido à sua simplicidade, atinge rapidamente um ponto de saturação, onde o aumento do tamanho da tabela não traz mais benefícios significativos na taxa de acertos. Por outro lado, os preditores Gselect e Gshare demonstraram melhorias mais sustentadas, com o Gshare destacando-se como o mais eficiente, especialmente em cenários com padrões de saltos mais complexos, devido à sua capacidade de capturar e explorar essas complexidades de maneira mais eficaz. Esses resultados enfatizam a importância de selecionar a técnica de predição de salto e o dimensionamento da tabela de predição de forma cuidadosa para maximizar o desempenho dos processadores. Além disso, nossos achados sugerem que pesquisas futuras devem explorar abordagens híbridas e adaptativas, que possam combinar o melhor de cada técnica, para alcançar ainda maiores otimizações.

Referências

Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. The RISC-V instruction set manual, volume I: User-level ISA, version 2.0. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54*, 2014.

Bruno Lévy. Learn FPGA. `https://github.com/BrunoLevy/learn-fpga/`, 2020.

James E Smith. A study of branch prediction strategies. *Readings in Computer Architecture*, 2000.

Shien-Tai Pan, Kimming So, and Joseph T Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pages 76–84, 1992.

Scott McFarling. Combining branch predictors. Technical report, Citeseer, 1993.

Terasic Technologies. DE10-Standard User Manual. `https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=1081`, 2017.