Avaliação de preditores de desvios por meio de simuladores como parte do processo de ensino e aprendizagem de Arquitetura de Computadores

Liana Duenha
Faculdade de Computação - FACOM
Universidade Federal de Mato Grosso do Sul - UFMS
Campo Grande, MS
Email: lianaduenha@facom.ufms.br

Felippi Crominski Magalhães
Faculdade de Computação - FACOM
Universidade Federal de Mato Grosso do Sul - UFMS
Campo Grande, MS
Email: felipppi@gmail.com

Mateus Tostes dos Santos
Faculdade de Computação - FACOM
Universidade Federal de Mato Grosso do Sul - UFMS
Campo Grande, MS
Email: mateustostesdossantos@gmail.com

Ricardo Ribeiro dos Santos
Faculdade de Computação - FACOM
Universidade Federal de Mato Grosso do Sul - UFMS
Campo Grande, MS
Email: ricardo@facom.ufms.br

***Resumo*—Este artigo mostra os resultados da aplicação de simuladores funcionais de processadores como recurso didático adicional para ensino de arquitetura de computadores. Especificamente, o artigo mostra o uso de simuladores para avaliação de preditores de desvios como parte de uma metodologia auxiliar durante o ensino deste tópico na disciplina Arquitetura de Computadores. A metodologia consiste em utilizar traços de execução de um conjunto de aplicações gerados por simuladores de quatro modelos de processadores e, a partir destes, caracterizar e avaliar o desempenho de oito preditores de desvios.**

I. Introdução

Grande parte das disciplinas que compõem a área de sistemas de computação e, especificamente, a disciplina Arquitetura de Computadores pode ser ministrada utilizando-se uma abordagem teórica, em que o conteúdo é ministrado com o objetivo de dar uma formação conceitual profunda sobre tópicos relevantes da área. Contudo, torna-se cada vez mais comum o uso de ferramentas para auxiliar no processo de ensino e aprendizagem, tornando essencial a aplicação de metodologias que possibilitem a conexão entre a teoria e a prática, fundamentando o aprendizado com mecanismos de experimentação e avaliação de resultados [1].

A maioria dos tópicos presentes no programa da disciplina Arquitetura de Computadores está intrinsecamente ligada à avaliação de desempenho de componentes isolados e no seu impacto no desempenho do sistema como um todo. Embora os livros adotados contenham muitos gráficos e análises comparativas para colaborar com premissas de desempenho previamente ensinadas, consideramos que demandas como estas são muito melhor exploradas quando se utiliza uma metodologia baseada em implementação e/ou experimentação pelo uso de simuladores de módulos isolados de hardware ou de sistemas completos. Dado um determinado contexto, a análise dos resultados experimentais de simulações permite que os alunos possam elaborar suas próprias conclusões que corroboram ou não com a teoria apresentada previamente e, desta forma, sintam-se mais ativos e partícipes do seu próprio processo de aprendizado [2].

Em um trabalho prévio [3], apresentamos uma metodologia de ensino complementar às disciplinas de Arquitetura de Computadores e correlatas, principalmente àquelas que utilizam projetos de arquiteturas RISC.

Especificamente, descrevemos os resultados obtidos em uma disciplina de Tópicos em Arquitetura de Computadores em que utilizou-se ferramentas de simulação de código aberto para apoio pedagógico.

Dando continuidade ao trabalho, aplicamos a metodologia equivalente na primeira metade da disciplina de Arquitetura de Computadores ministrada na pós-graduação. A partir dos experimentos, os alunos realizaram avaliação de plataformas multicore do ponto de vista de desempenho, uso de memória, políticas de cache, consumo energético, DVFS e mecanismos de interconexão. Obtivemos resultados favoráveis, cumprindo com o objetivo de nivelar a turma que tinha características bastante heterogêneas.

Na segunda metade da disciplina, nós nos deparamos com tópicos mais avançados como, por exemplo, execução especulativa e predição de desvios condicionais. Infelizmente, estes são recursos não suportados pelos simuladores que tínhamos disponíveis até então. Foram, então, desenvolvidos módulos de simulação para prever o comportamento de preditores de desvios que foram adicionados a simuladores funcionais de processadores ou de plataformas completas [4], [5].

O objetivo deste artigo é mostrar os resultados obtidos com a implementação de oito preditores de desvios utilizando simuladores funcionais de processadores RISC. Especificamente, a

abordagem aqui apresentada visa motivar estudantes e professores para utilizar uma infraestrutura para implementação dos preditores possibilitando assim novas formas de aprendizagem por meio de experimentações e avaliações empíricas.

Este artigo está organizado da seguinte forma: na Seção II são apresentados alguns dos trabalhos que utilizaram ferramentas de simulação para ensino de Arquitetura de Computadores; a Seção III contém a fundamentação teórica sobre os tópicos de Arquitetura de Computadores que serão avaliados durante a experimentação; a Seção IV apresenta a metodologia aplicada para instrumentação dos simuladores e posterior trabalho de experimentação; na Seção V são descritos e avaliados os resultados dos experimentos e, a Seção VI conclui este trabalho.

II. Trabalhos Relacionados

Esta seção mostra o uso de simuladores funcionais durante o processo de ensino de Arquitetura de Computadores e disciplinas afins.

Em 2012, Zeferino et al. [6] apresentou os resultados do uso da família de processadores BIP (do inglês *Basic Instruction-set Processor*), a qual foi concebida para auxiliar na aprendizagem de conceitos de arquitetura de computadores em distintas disciplinas da grade do curso de Ciência da Computação (Algoritmos e programação, Circuitos digitais, Compiladores, entre outras), permitindo que fosse dado um enfoque interdisciplinar aos conceitos de arquitetura e organização de computadores.

Em muitas universidades, é escolhida a arquitetura MIPS como exemplo para o estudo de conjunto de instruções e projeto do processador. Um dos primeiros simuladores MIPS desenvolvidos para fins educacionais foi o MIPSim [7], desenvolvido a partir das especificações do livro comumente adotado para ensino de arquitetura de computadores [8]. Nessa versão do simulador, é explorado o comportamento dinâmico do processador no nível organizacional como, por exemplo, valor dos registradores, entradas e saídas dos multiplexadores e comparadores.

O DIMIPSS [9] é um software multiplataforma de simulação do caminho de dados e de sinais de controle para execução de instruções do processador MIPS Monociclo. O simulador considera como entrada um programa em linguagem de montagem para MIPS, converte-o para linguagem de máquina e representa graficamente o comportamento das instruções durante a execução.

O WebMIPS [10] foi proposto por Branovic et al. é um simulador MIPS cuja maior vantagem é seu acesso via web sem necessidade de instalações ou configurações prévias. Esse simulador inclui visão dos estágios de pipeline, memórias, registradores e unidades de forwarding.

ArchC é uma linguagem de descrição de arquitetura que permite a geração de instâncias de processadores, compiladores e montadores (*cross-compilers*) e diversos outros componentes de um sistema computacional. Atualmente, o ArchC disponibiliza quatro modelos de processadores que serão utilizados neste trabalho e estão descritos na Seção IV. O mesmo grupo de pesquisa que faz a gestão do ArchC, disponibiza também o MPSoCBench [5], uma ferramenta para descrição e simulação de plataformas *multicore* (MPSoCs). Ambas as ferramentas foram concebidas inicialmente para o apoio à pesquisa, porém há trabalhos recentes que mostram o seu uso também como ferramenta de apoio ao ensino [3].

III. Fundamentação teórica

Desde a década de 1980, os processadores exploram paralelismo em nível de instrução (ILP) por meio de *pipelining*, permitindo que várias instruções estejam em execução durante um mesmo ciclo de *clock*. Em sequência, o suporte a pipeline foi aprimorado e aos poucos foram implementadas técnicas dinâmicas ou estáticas mais sofisticadas como, por exemplo, despacho múltiplo, especulação ou escalonamento de código.

Existem duas abordagens para exploração de paralelismo em nível de instrução, uma estática e outra dinâmica. Na primeira delas, conta-se com a tecnologia de software para encontrar e explorar paralelismo e, na segunda, o paralelismo é explorado dinamicamente (em tempo de execução) pelo hardware presente no processador.

Os processadores com exploração dinâmica de ILP dominam o mercado de computadores pessoais (*desktops*, *notebooks*) e servidores, pois alcançam melhor desempenho. Entretanto, técnicas dinâmicas de exploração de ILP tendem a aumentar consideravelmente o consumo energético, podendo comprometer seu uso em dispositivos de computação móvel, onde a eficiência energética é fundamental.

Atualmente, busca-se aproveitar o que há de melhor nas duas técnicas, aliando a eficiência das técnicas de exploração de paralelismo por hardware (dinâmicas) com o baixo custo, baixo consumo e a praticidade de desenvolvimento de técnicas baseadas em compilador (estáticas) [11], [12], [13].

Nas disciplinas de Arquitetura de Computadores em nível de graduação ou pós-graduação são abordados tópicos de projeto de processadores com exploração de paralelismo em nível de instrução baseadas em técnicas dinâmicas e estáticas. São ensinadas técnicas baseadas em compilador, desde as mais simples como escalonamento estático e desdobramento de laços, até mais avançadas como processadores VLIW, e técnicas dinâmicas como escalonamento dinâmico e despacho múltiplo e especulação. Todas estas técnicas têm em comum o uso de predição de desvios como ponto essencial para redução dos atrasos causados por conflitos de controle.

Dizemos que uma instrução j é dependente de uma instrução i se a execução de j depende do resultado da execução de i. Se a instrução i é uma instrução de desvio, dizemos que esta é uma *dependência de controle* que determina a ordem de execução de j com relação a uma instrução de desvio i.

A priori, o despacho da instrução j só poderia ser feito após a instrução de desvio i ter sido completada. Porém, em processadores que exploram paralelismo, essa condição de ordem pode trazer prejuízos consideráveis no desempenho.

Com a intenção de contornar essa limitação, é comum que se permita o despacho de outras instruções dependentes de

um desvio antes de que o mesmo tenha sido completado, desde que não afetem a exatidão do programa. Esta técnica é chamada de especulação. Se uma instrução é executada de maneira especulativa, ela não pode realizar a escrita do resultado produzido em registrador ou memória antes de que a instrução de desvio na qual ela é dependente tenha sido finalizada.

Para se aplicar a especulação, deve-se realizar a predição do resultado dos desvios, isto é, deve-se decidir entre despachar a instrução seguinte ao desvio no código, supondo que o desvio não será tomado, ou a instrução resultante do cálculo de endereço de desvio, supondo que o desvio será tomado.

No caso de predição correta do resultado do desvio, não houve atraso causado por dependências de controle. Caso contrário, se houver erro de predição, as instruções executadas de forma especulativa serão ignoradas e as instruções corretas serão despachadas, o que causa vários ciclos de atraso. Um ciclo de *stall* para cada desvio pode causar perda de desempenho de 10-30%, dependendo da frequência dos desvios [11].

Com o objetivo de diminuir os atrasos causados por erros de predição, os processadores contam com técnicas sofisticadas de predição de desvios. Em sala de aula, é comum que o docente descreva diversas técnicas de predição, algumas muito simples baseadas em software e outras mais sofisticadas implementadas em hardware. Nesse ponto, acredita-se que uma avaliação mais detalhada dos preditores, considerando não somente a taxa de acertos, mas também o seu custo em hardware, traz benefícios consideráveis para a fundamentação dos conceitos e, consequentemente, para o aprendizado do aluno.

Neste trabalho foram avaliados diversos preditores de desvios, alguns estáticos e outros dinâmicos, que serão descritos a seguir:

1) **Preditor de desvio não tomado ou *not taken*(NT)**: assume-se que o desvio não será tomado e despacha-se a instrução que se encontra imediatamente após o desvio no código;
2) **Preditor de desvio tomado ou *taken*(T)**: assume-se que o desvio será tomado e despacha-se a instrução que se encontra no endereço calculado a partir da instrução de desvio;
3) **Preditor global de 2 bits (2BG)**: há um único preditor para todos os desvios do programa que segue as regras descritas pelo diagrama de estados da Figura 1. Cada estado representa o resultado da predição que pode ser T (*tomado*) ou NT (*não-tomado*). As transições entre estados ocorrem a partir do resultado de cada desvio, ou seja, do que de fato foi decidido pelo último desvio analisado. Se consideramos o sistema binário em que o valor 0 significa "tomado"e o valor 1 significa "não-tomado", cada estado pode ser descrito por um valor binário com 2 bits, em que o bit à esquerda representa a predição e o bit à direita representa o resultado do último desvio.
4) **Preditor local de 2 bits (2B)**: há um preditor de 2 bits para cada desvio do programa e o comportamento de cada preditor obedece as regras descritas na Figura 1. A quantidade de preditores implementados no hardware pode ser consideravelmente menor do que a quantidade de desvios do programa; assim, pode ocorrer que vários desvios compartilham um mesmo preditor local.

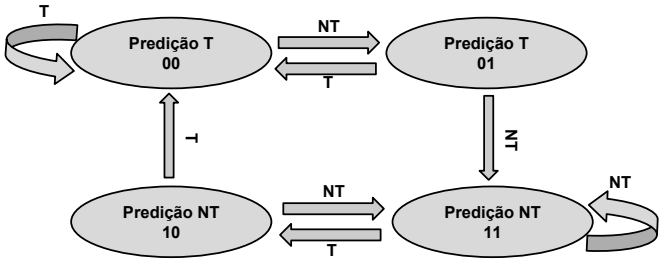


Figura 1. Diagrama de estados para o preditor de 2 bits

5) **Preditor de correlação** (m,n): Esquema em que se utiliza o comportamento de outros desvios para fazer a predição do desvio sendo avaliado no momento. Para predizer o comportamento de um determinado desvio, o preditor de correlação (m,n) utiliza o comportamento dos últimos m desvios para escolher entre 2^m preditores, cada qual sendo um preditor de n bits [11]. A Figura 2 ilustra um preditor de correção (2,3) em que um histórico global de 2 bits guarda o resultado dos últimos 2 desvios executados e utiliza esse valor para selecionar entre quatro preditores de desvios, cada qual sendo um preditor de 3 bits.

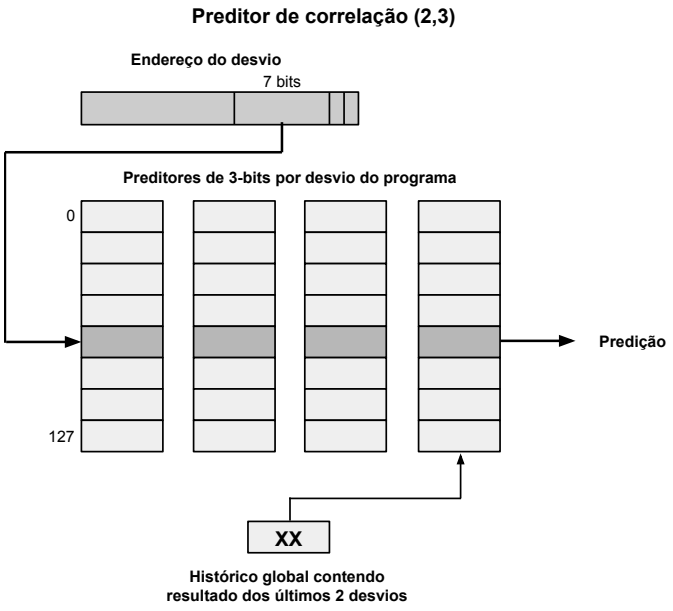


Figura 2. Esboço de um preditor de correção (2,3) com 128 entradas

6) **Preditor de desvio baseado em direção (Direção):** a predição é feita a partir da comparação entre o endereço da instrução atual (a própria instrução de desvio) e o endereço de destino do desvio (endereço alvo). Neste

trabalho, assumimos a seguinte regra: se o endereço de destino do desvio é maior do que o endereço da instrução de desvio, dizemos que é um desvio *para frente* e, assume-se predição *not taken*; caso contrário, dizemos que é um desvio *para trás* e assume-se predição *taken*;

7) **Preditor de Torneio:** O preditor de torneio utiliza um contador de saturação de 2 bits por desvio para escolher entre 2 preditores diferentes. Para este estudo foram utilizados dois preditores: 1) um preditor local de 2 bits que usa o comportamento do desvio atual; 2) um preditor global de 2 bits que utiliza o comportamento dos desvios anteriores.
A Figura 3 representa o diagrama de estados do preditor de torneio. Para um dado desvio sendo avaliado, os estados representam a escolha de qual preditor deve ser utilizado e a notação A/B que identifica as transições representando acerto ou erro de cada preditor. Por exemplo: 0/0 representa a situação em que ambos os preditores erraram; 0/1 representa acerto pelo preditor 1 e erro pelo preditor 2; 1/0 representa acerto pelo preditor 1 e erro pelo preditor 2 e, 1/1 representa a situação em que ambos os preditores acertaram. No caso em que, por 2 vezes, o preditor atual erra e o outro acerta, troca-se de preditor.

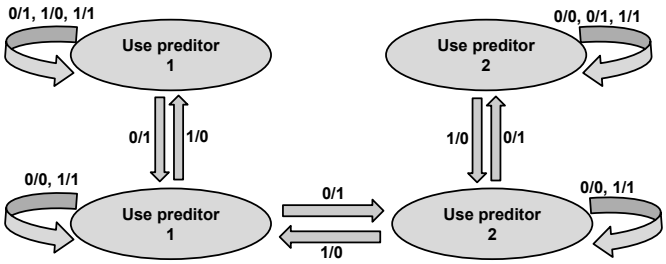


Figura 3. Diagrama de estados de um preditor de torneio para escolher entre 2 preditores.

8) **Preditor de *loop* e 2B (*loop*)**: há um preditor de 2 bits para cada desvio do programa e o comportamento de cada preditor obedece as regras descritas na Figura 1. Em paralelo, há um preditor de *loop* que entra em ação após um *branch* efetuar a mesma ação 3 vezes consecutivas. Quando um *loop* alcança sua última iteração predita, a predição é invertida. Para predizer o número de iterações de um *loop*, são utilizados dados de execuções anteriores do mesmo *loop* por meio de um contador com 8 bits. Há ganhos substanciais de desempenho principalmente em *loop*s aninhados ou quando os mesmos estão em funções ou métodos chamados com frequência significativa.

IV. Metodologia

O objetivo desta seção é descrever a metodologia para a avaliação dos preditores de desvios descritos na Seção III. A Figura IV ilustra as 6 etapas realizadas durante o processo de implementação e experimentação.

As etapas 1 e 2 correspondem à geração e instrumentação dos simuladores para a finalidade deste trabalho. Foram gerados os simuladores dos processadores ARM, MIPS, PowerPC e SPARC utilizando o *framework* ArchC [4].

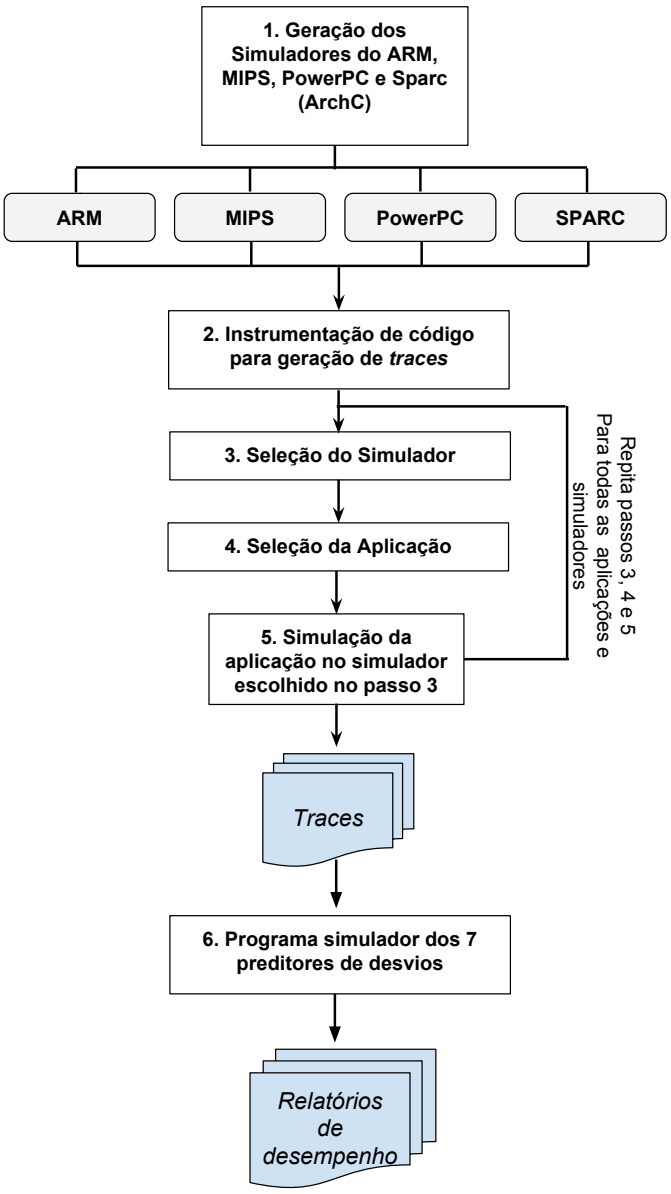


Figura 4. Ilustração da metodologia adotada

O ArchC[4], [5] é ferramenta que provê uma linguagem descritora de arquitetura (ADL) que permite a geração e simulação de diferentes instâncias de processadores e diversos outros componentes de um sistema computacional.

Para a definição de um processador, o ArchC requer dois tipos de informações: a descrição dos recursos arquiteturais que definem o processador e seu conjunto de instruções, ambos definidos pelas respectivas macros `AC_ARCH` e `AC_ISA`.

`AC_ARCH` requer informações sobre quantidade de registradores, memórias associativas e estrutura de *pipeline*; já

AC_ISA requer a descrição do formato, tipo e codificação (ou decodificação) de instruções e a descrição do comportamento de cada instrução.

O código a seguir mostra como definir a arquitetura do processador MIPS. A descrição do modelo inicia pela declaração da memória principal, cache de dados, caches de instruções, seguindo pela definição do banco de registradores de propósito geral e registradores de propósito específico (PC, Hi e Lo), definição da palavra do computador e ao final, indica onde se encontra a definição do conjunto de instruções, define *endianess* e realiza as ligações entre os módulos de memória.

```
AC_ARCH(mips){

  ac_mem    MEM:512M;

  ac_icache   IC("2w", 64, 8, "wt", "random");
  ac_dcache   DC("2w", 64, 8, "wt", "random");

  ac_regbank RB:32;
  ac_reg npc;
  ac_reg hi, lo;

  ac_wordsize 32;
  ac_fetchsize 32;

  ARCH_CTOR(mips) {
    ac_isa("mips_isa.ac");
    set_endian("big");
    IC.bindTo(MEM);
    DC.bindTo(MEM);
  };
};
```

O arquivo mips_isa.ac contém informações relativas ao conjunto de instruções, como formato e demais informações para decodificação. Segue abaixo trecho de código deste arquivo que define o formato, definições relativas ao tipo das instruções e os mnemônicos que as identificam do modelo MIPS.

```
AC_ISA(mips) {

 ac_format Type_R = "%op:6 %rs:5 %rt:5 %rd:5 0x00:5
     %func:6";
 ac_format Type_I = "%op:6 %rs:5 %rt:5 %imm:16";
 ...
 ac_instr<Type_R> add, mul;
 ac_instr<Type_I> lw, bne, addi;
 ...
 ISA_CTOR(mips) {

    add.set_asm("add %reg, %reg, %reg", rd,rs,rt);
    add.set_decoder(op=0x00, func=0x20);
    ...
    lw.set_asm("lw %reg, %imm(%reg)", rt,imm,rs);
    lw.set_decoder(op=0x23);
    ...
 };
};
```

O comportamento de cada instrução é definido por meio de métodos em um arquivo separado (mips_isa.cpp). A seguir, o código original da instrução beq (*branch on equal*) que realiza o desvio condicional para uma instrução cujo endereço é calculado a partir dos bits armazenados no campo imm da instrução. Nesse caso, o endereço de destino é salvo em npc que, nesse contexto, representa o registrador PC. O desvio é realizado quando os dois registradores fornecidos na instrução têm o mesmo valor. Caso contrário, o PC não é atualizado e a execução segue para a próxima instrução após o desvio no código do programa.

```
//!Instruction beq behavior method.
void ac_behavior( beq )
{
  dbg_printf("beq r%d, r%d, %d\n", rt, rs, imm & 0
      xFFFF);
  if( RB[rs] == RB[rt] ){
#ifndef NO_NEED_PC_UPDATE
    npc = ac_pc + (imm<<2);
#endif
    dbg_printf("Taken to %#x\n", ac_pc + (imm<<2));
  }
};
```

Foram realizadas modificações no arquivo de descrição do comportamento das instruções para que os resultados dos desvios condicionais fossem gravados em arquivos de saída. Para cada desvio executado, foi armazenado o seu endereço, o tipo do desvio, o endereço de destino e o resultado do desvio (tomado ou não-tomado).

As etapas 3, 4 e 5 são relacionadas à geração de *traces* das instruções de desvios executadas e, posteriormente, a partir dos arquivos de saída gerados nessas etapas, o comportamento dos preditores de desvios serão avaliados.

Para a geração dos *traces*, foram executadas dez aplicações [1] do benchmark Mibench [14], disponíveis em http://www.archc.org/downloads.html. Todas foram executadas nos quatro simuladores gerados nas etapas 1 e 2. Segue a descrição das aplicações:

- **Bitcnts**: aplicação para avaliar habilidade do processador em contar a quantidade de bits de um vetor de inteiros;
- **Qsort**: ordena um vetor de *strings* em ordem ascendente utilizando o algoritmo *quick sort*.
- **Susan Smoothing:** faz parte de um conjunto de funcionalidades para reconhecimento de imagens utilizadas sobre dados de ressonâncias magnéticas do cérebro. Esta aplicação realiza ajustes de brilho e suavização da imagem de entrada. Os experimentos foram executados sobre versões "small" e "large" desta aplicação que se diferenciam pelo tamanho dos arquivos de entrada;
- **Dijkstra:** calcula caminhos mínimos entre todos os pares em um gráfico representado por uma matriz de adjacência. Os experimentos foram executados sobre versões "small" e "large" desta aplicação que se diferenciam pelo tamanho dos grafos de entrada;
- **Patricia**: uma árvore patricia é uma estrutura de dados bastante utilizada para representar tabelas de roteamento em aplicativos de rede. Os dados de entrada para esta aplicação são uma lista de tráfego IP de um servidor web por um período de 2 horas.

[1] São oito aplicações distintas, porém *dijkstra* e *susan* são utilizadas em duas versões (*small* e *large*)

- **SHA:** implementa o algoritmo Secure Hash, útil para gerar assinaturas digitais usadas na troca segura de chaves criptográficas.
- **FFT**: é um algoritmo eficiente para realizar a Transformada Rápida de Fourier que é de grande importância para aplicações de processamento digital de sinais, resolução de equações diferenciais e algoritmos para multiplicação de grandes inteiros.
- **Basicmath:** realizar cálculos matemáticos como resolução de funções cúbicas, conversões angulares de graus em radianos e raízes quadradas inteiras;

A etapa 6 corresponde à simulação do comportamento de preditores de desvios para futura avaliação de desempenho. Cada aluno implementou sua própria versão dos preditores de desvios para computar a taxa de acertos para cada tipo de desvio condicional presente na arquitetura. Foram implementados oito preditores de desvios que foram avaliados utilizando os 40 arquivos de *traces* gerados anteriormente.

V. Resultados Experimentais

Os preditores de desvios foram simulados tendo como entrada os 40 arquivos de *traces* resultantes da execução das dez aplicações nos quatro simuladores de desempenho dos processadores (MIPS, PowerPC, SPARC e ARM). Embora experimentos similares tenham sido realizados por todos os alunos da turma, selecionamos dados de execução de apenas dois dos alunos para compor os gráficos e análises deste artigo. Os testes foram realizados em um sistema com processador Intel i3 M370 2.4GHz, com 4GB de memória RAM DDR3, HD SSD 254GB, Sistema operacional Ubuntu 14 x64.

A Figura 5 a 8 mostram as taxas de acerto dos preditores quando utilizamos dados dos simuladores MIPS, PowerPC, SPARC e ARM, respectivamente. Para mostrar os resultados para todas as aplicações, foi necessário dividi-las em dois grupos, resultando em dois gráficos por Figura.

Utilizando como métrica de desempenho as taxas de acertos alcançadas por cada preditor, nota-se pela Figura 5 que no processador MIPS, os preditores 2B, CR23, Torneio e Loop obtiveram taxa acima de 95% em pelo menos uma das aplicações. Os preditores NT, T e de direção obtiveram taxas abaixo dos 70% em pelo menos metade das aplicações. Ainda, o preditor NT obteve taxa de acertos abaixo de 60% em todas as aplicações, com o pior caso sendo taxas próximas a 10% para as aplicações susan_lg e sha.

A Figura 6 mostra o comportamento dos preditores de desvios em programas executados no processador PowerPC e nota-se comportamento similar ao avaliado para MIPS. Apenas os preditores 2B, CR23, Torneio e Loop obtiveram taxa acima de 95% em pelo menos uma das aplicações. Os preditores NT, T e de direção obtiveram taxas abaixo dos 60% em pelo menos metade das aplicações. Mais uma vez, o preditor NT obteve taxa de acertos abaixo de 60% em todas as aplicações, com o pior caso sendo taxas próximas a 10% na execução do sha.

A Figura 7 mostra o comportamento dos preditores de desvios programas executados no processador SPARC. Nota-se que os preditores 2B, CR23, Torneio e Loop alcançaram

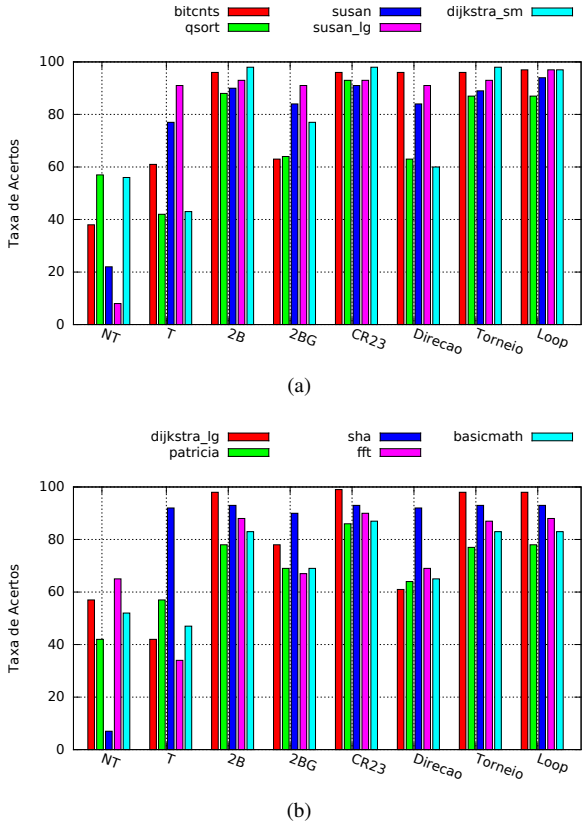


Figura 5. Taxa de acertos de oito preditores de desvios no MIPS

taxas de acerto acima dos 95% em 3 aplicações. O preditor Direção obteve taxas de acerto mais baixas no SPARC do que nos demais processadores. O preditor NT obteve taxa de acerto abaixo dos 50% para todas as aplicações.

A Figura 8 mostra o comportamento dos preditores de desvios no ARM com dados similares aos obtidos para os demais processadores.

Pelos dados analisados até aqui, nota-se que a execução em processadores diferentes não causa grandes impactos nas taxas de acertos dos preditores e para ratificar essa conclusão.

A Figura 9 mostra a taxa de acertos médio de cada um dos oito preditores durante a execução das dez aplicações nos quatro processadores. O preditor que apresentou menor diferença de comportamento foi o CR23 (menos de 1%) e o preditor que apresentou maior diferença de comportamento foi o 2BG (pouco mais de 9%). Os demais preditores apresentam diferenças em torno de 3%.

Avaliou-se também o comportamento dos preditores com relação a diferentes classes de desvios na Figura 10, utilizando a taxa média de acertos nas predições para as dez aplicações. Primeiramente, diferenciamos o comportamento dos desvios com relação à condição: **beq** (*branch on equal*) e **bne** (*branch on not equal*) (Figuras 10(a) e 10(b), respectivamente), independentemente da direção do desvio.

Nota-se que o predidor de correlação CR23 obteve o melhor desempenho, seguido de Loop e 2B. O preditor T obteve o pior desempenho dentre todos.

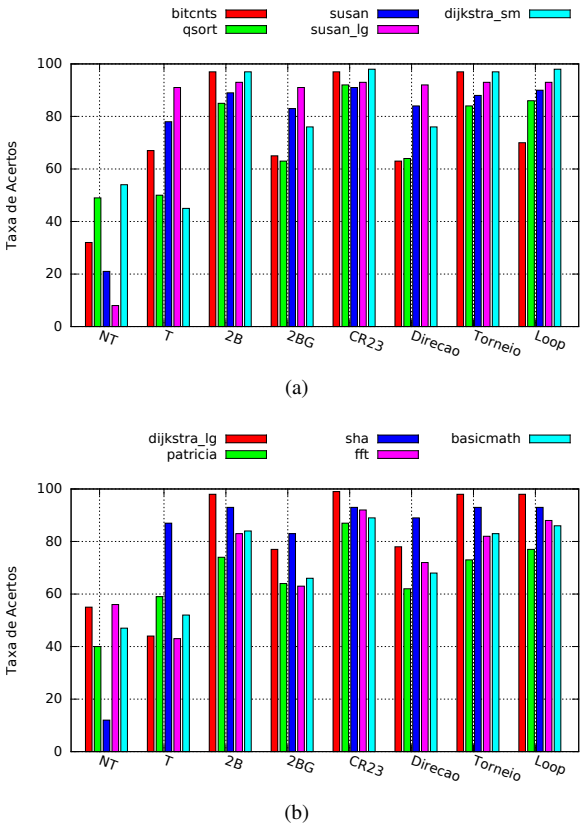


Figura 6. Taxa de acertos de oito preditores de desvios no PowerPC

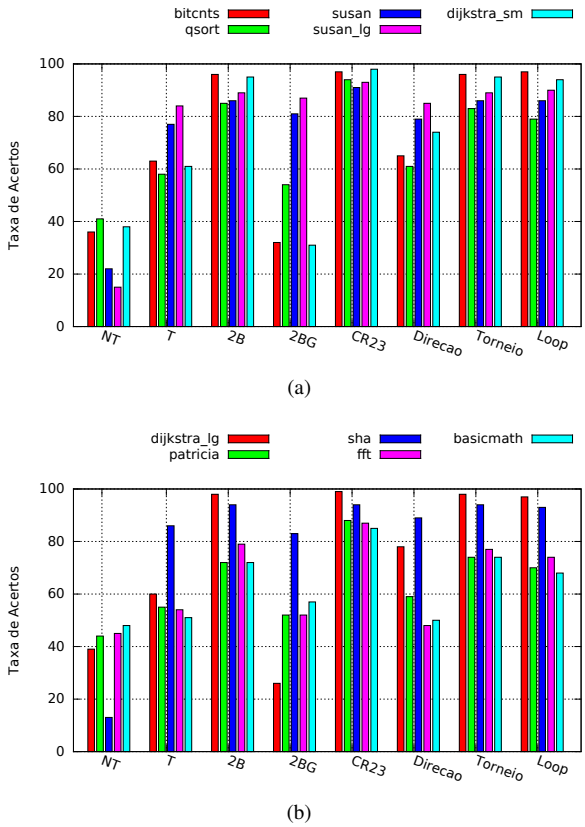


Figura 7. Taxa de acertos de oito preditores de desvios no SPARC

Verifica-se também que que a taxa de acertos baseadas no preditor T é muito baixa para desvios do tipo **beq**; isso reflete o uso comum deste tipo de desvios em laços de repetição em que o desvio é utilizado para saída do laço, quando a variável de controle atinge um determinado valor (ex: laços **for**).

Os quatro preditores com melhor desempenho para predizer o comportamento das instruções **beq** também tiveram o melhor desempenho para predizer **bne**. Entretanto, o preditor global NT foi o que obteve pior desempenho (Figura 10(b)).

Em seguida, utilizamos direção do desvio como critério de avaliação. Os resultados são mostrados nas Figuras 10(c) e 10(d). Chamamos de desvio para frente quando o endereço de destino do desvio é maior que o PC atual e de desvio para trás quando o endereço de destino do desvio é menor que o PC.

Os desvios com endereço-alvo maior do que o endereço do PC (desvios para frente) foram preditos mais eficientemente pelo preditor de correlação CR23, enquanto o pior desempenho foi alcançado pela predição global T. Com relação dos desvios para trás, o preditor de correlação CR23 teve desempenho mais baixo (menos que 80%) e os preditores Torneio, Loop, 2B, 2BG e direção alcançaram taxas próximas a 90%. Desvios para trás são bastante utilizados para comparação e desvio no final do laço (ex: laços **do-while**) e, nesses casos, espera-se que a predição NT tenha desempenho pior que a predição T, como é possível verificar pelo gráfico.

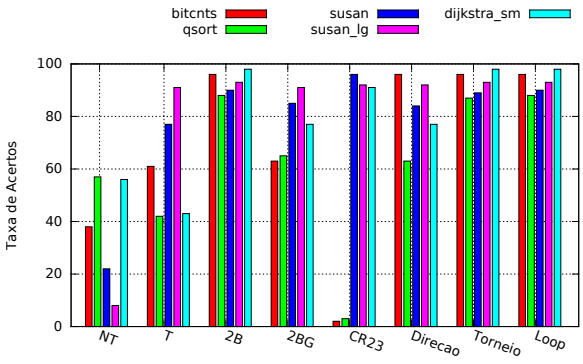
Uma forma de comparar os preditores com relação ao custo de hardware é avaliar a quantidade de bits necessários para armazenamento dos dados. Consideramos que os preditores estáticos T, NT e Direção não necessitam de hardware adicional. Também consideramos desprezível o hardware adicional para o preditor global de 2 bits (2BG) que também é utilizado como parte do preditor Loop.

Para os demais, mostramos na Tabela I a quantidade total de bits para a computação da predição, considerando que todos os preditores têm 128 linhas. Contabilizamos a quantidade de preditores por linha e a quantidade de bits por preditor local somando bits gastos para armazenamento da predição, histórico dos últimos desvios, contadores e quantidade de colunas (no caso de CR23 e Torneio).

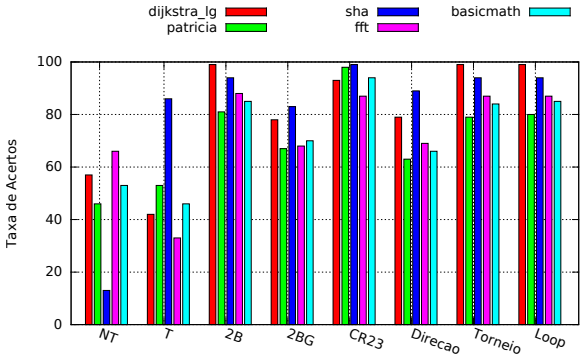
O preditor 2B necessita menor quantidade de bits para sua implementação, quando comparado aos outros três preditores avaliados. O preditor CR23 requer mais bits para sua implementação; entretanto, como foi desprezada a quantidade de bits necessários para implementação de máquina de estados necessárias para implementação dos preditores 2B, Torneio e Loop, podemos concluir que CR23, Torneio e Loop não apresentam diferenças significativas com relação a quantidade de bits necessárias para sua implementação.

VI. Conclusão

Este artigo mostrou os resultados do uso de simuladores de processadores para apoio ao ensino de Arquitetura de Computadores. Especificamente, avaliou-se o desempenho de oito



(a)



(b)

Figura 8. Taxa de acertos de oito preditores de desvios no ARM

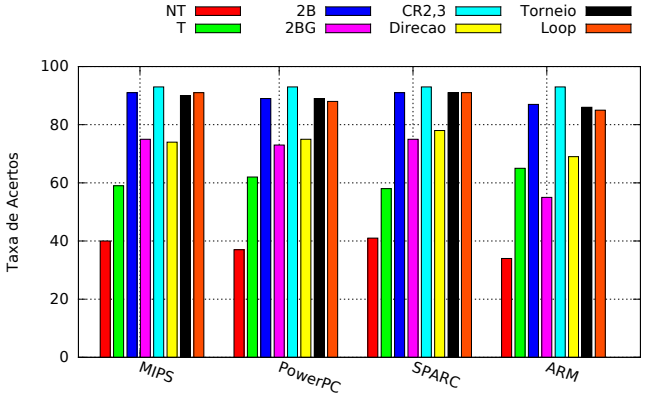
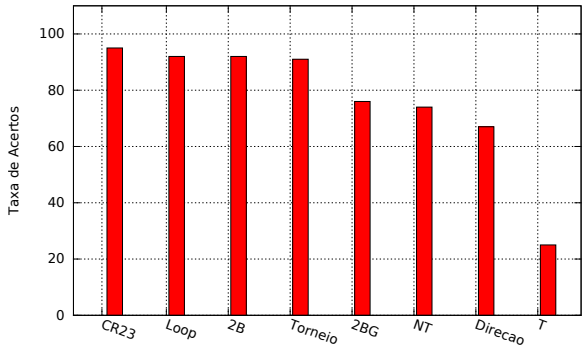


Figura 9. Taxa de acerto médio de cada um dos oito preditores de desvios durante a execução de dez aplicações nos quatro processadores

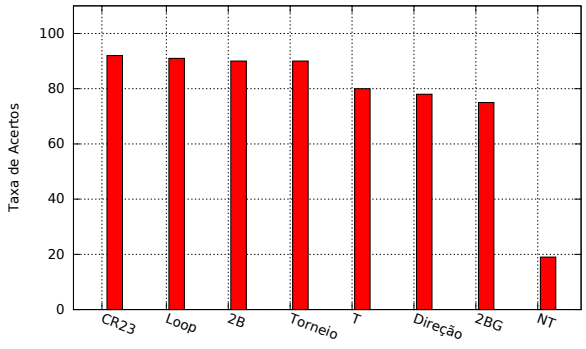
preditores de desvios sobre *traces* de execução de programas do *suite benchmark* Mibench sobre simuladores funcionais dos processadores MIPS, PowerPC, SPARC e ARM, como parte das atividades avaliativas da disciplina.

No tocante ao uso desta abordagem como apoio ao ensino, podemos citar os principais impactos observados:

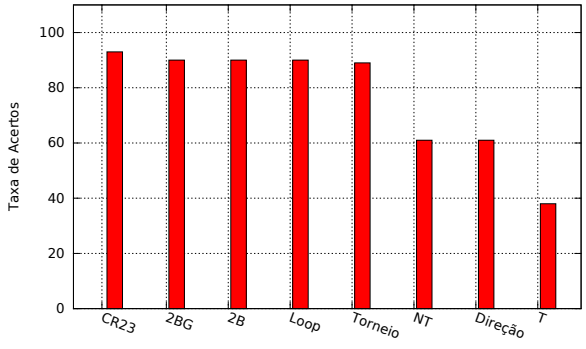
- A redução de problemas de aprendizagem devido à possibilidade de aprofundar empiricamente o aprendizado de tópicos teóricos;
- A facilidade com que se pode variar as configurações e



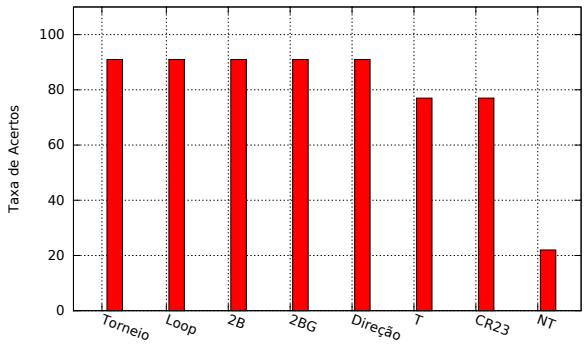
(a)



(b)



(c)



(d)

Figura 10. Taxa de acertos de oito preditores de desvios para desvios condicionais (a) "branch on equal" (b) "branch on not equal" (c) para frente (d) para trás

Tabela I
QUANTIDADE DE BITS POR PREDITOR

	2B	CR23	Torneio	Loop
Bits por preditor	2	3	2	2
Preditores por linha	1	4	4	1
Bits por linha	2	12	8	11 [2]
Total de linhas	128	128	128	128
Bits adicionais	0	2(histórico)	0	0
Total	256	1538	1024	1408

parâmetros arquiteturais que impactam no desempenho do sistema e a possibilidade de mensurar o impacto empiricamente;

- A possibilidade de realizar comparação entre os resultados esperados e os resultados obtidos experimentalmente de forma a corroborar (ou não) com a teoria presente na literatura;
- A mudança de postura do aluno com relação ao seu papel no seu processo de formação; nota-se que o aluno torna-se mais maduro e auto-confiante quando é capaz de fornecer conclusões a respeito dos tópicos que estão sendo avaliados;

REFERÊNCIAS

[1] Diana Morandi, Maicon Carlos Pereira, André Luis Alice Raabe, and Cesar Albenes Zeferino. Um processador básico para o ensino de conceitos de arquitetura e organização de computadores. *HÍFEN*, 30(58), 2006.

[2] Angela Carbone and Jens J Kaasbøll. A survey of methods used to evaluate computer science teaching. In *ACM SIGCSE Bulletin*, volume 30, pages 41–45. ACM, 1998.

[3] L Duenha and R Azevedo. Utilização dos simuladores do mpsocbench para o ensino e aprendizagem de arquitetura de computadores. *International Journal of Computer Architecture Education (IJCAE)*, 5(1):26–31, 2016.

[4] Rodolfo Azevedo, Sandro Rigo, Marcus Bartholomeu, Guido Araujo, Cristiano Araujo, and Edna Barros. The ArchC Architecture Description Language and Tools. In *International Journal of Parallel Programming. Vol. 33, No. 5*, pages 453–484. October 2005.

[5] Liana Duenha, Marcelo Guedes, Henrique Almeida, Matheus Boy, and Rodolfo Azevedo. Mpsocbench: A toolset for mpsoc system level evaluation. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, INSPEC Number: 14564763, pages 164–171. IEEE, July 2014.

[6] Cesar Albenes Zeferino, André Luis Alice Raabe, Paulo Viniccius Vieira, and Maicon Carlos Pereira. Um enfoque interdisciplinar no ensino de arquitetura de computadores. *C. Martins, P. Navaux, R. Azevedo, S. Kofuji. Arquitetura de Computadores: educação, ensino e aprendizado*, 2012.

[7] Herbert Grunbacher and H Khosravipour. Windlx and mipsim pipeline simulators for teaching computer architecture. In *Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on*, pages 412–417. IEEE, 1996.

[8] David A Patterson and John L Hennessy. *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[9] Ariane Felix, Christiane Pousa, and Milene Carvalho. Dimipss: Um simulador didático e interativo do mips. In *Workshop sobre Educação em Arquitetura de Computadores*, pages 49–52, 2006.

[10] Irina Branovic, Roberto Giorgi, and Enrico Martinelli. Webmips: a new web-based mips simulation environment for computer architecture education. In *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*, page 19. ACM, 2004.

[11] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[12] William Stallings. Arquitetura e organização de computadores 8a edição, 2010.

[13] Linda Null and Julia Lobur. *Princípios básicos de arquitetura e organização de computadores*. Bookman Editora, 2009.

[14] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of IEEE 4th Annual Workshop on Workload Characterization, held in conjunction with The 34th Annual IEEE/ACM*, pages 03–14. December 2001.