Técnicas de Predição de Desvios

Jaudete Daltio - RA: 049240
Instituto de Computação - UNICAMP
Campinas, Brasil
jaudete@gmail.com

RESUMO

Um dos fatores mais importantes na determinação do desempenho de um processador é a vantagem tirada por sua implementação do nível de paralelismo das instruções, através de técnicas como *pipeline*, por exemplo. Um dos limites críticos para o paralelismo é a presença de desvios condicionais, que causam um retardo na definição de qual será a próxima instrução. Esse retardo é comumente conhecido como *conflito de controle (control hazard)*. Na tentativa de aumentar o paralelismo, muitos autores propõem técnicas para prever o fluxo de um desvio condicional. Em geral, esses previsores tiram vantagem de diferentes padrões observados no comportamento dos desvios. O objetivo deste trabalho é apresentar os principais métodos implementados para a predição de desvios em *software* e em *hardware* (estáticos e dinâmicos). Além de caracterizar as principais técnicas de predição, este trabalho ilustra algumas implementações em processadores reais e apresenta uma análise do desempenho de algumas técnicas em *benchmarks*.

1. INTRODUÇÃO

Vários processadores, na busca por altos níveis de desempenho, empregam combinações de técnicas para exploração de paralelismo. Nesse contexto encontram-se arquiteturas superescalares e o emprego de *pipeline* no desenvolvimento de processadores. O paralelismo temporal explorado por processadores com *pipeline* garante a execução parcial de até n instruções simultaneamente, onde n é o número de estágios do *pipeline*.

Nessas técnicas, as instruções de desvio representam um dos mais importantes problemas que afetam o desempenho do processador. Para suprir o *pipeline*, uma instrução deve ser buscada na memória a cada ciclo de *clock*. Porém, as instruções de desvio geram um retardo na definição de qual será a próxima instrução, causando o que é conhecido como *conflito de controle (control hazard)* [3]. É possível reduzir esse deslocando a execução dos desvios para mais cedo no *pipeline*, porém a penalidade, mesmo que de um ciclo apenas, provoca uma queda no desempenho de processadores com esse tipo de organização. Considerando que as instruções de desvio são responsáveis por uma grande parcela do tempo de processamento dos programas de usuário (cerca de 15 a 30% das primitivas interpretadas pelos processadores [7]), vários esforços são feitos para obter técnicas que reduzam as penalidades impostas pelas instruções de desvio.

Com esse intuito, foram desenvolvidos mecanismos que tentam antecipar se uma instrução é de desvio e determinar, antes da sua fase de execução, se o desvio será tomado ou não. Essa técnica é conhecida como *"predição de desvio"*, e faz parte da maioria dos processadores, influenciando significativamente o desempenho de arquiteturas *pipeline* e processadores superescalares.

Uma vez feita a predição, o processador pode especular a execução das instruções que dependem do resultado do desvio. A presença de instruções que não pertencem ao caminho especificado pelo resultado de um desvio nos estágios iniciais do *pipeline* somente pode ser detectada na fase de execução do desvio. Caso o resultado do desvio tenha sido previsto incorretamente, será necessário remover as instruções que foram introduzidas indevidamente no *pipeline* [3]. Conseqüentemente, ocorre uma perda de alguns ciclos de processador com a remoção, o redirecionamento do fluxo e o preenchimento dos estágios iniciais.

Existem várias técnicas propostas para especular o caminho de um fluxo de instrução depois de um desvio. O objetivo desse trabalho é apresentar as principais técnicas e métodos implementados para a predição de desvios, mostrando as principais características de cada uma dessas técnicas.

Considerando como critério o nível de implementação, podemos identificar dois tipos de técnicas de predição de desvios: técnicas implementadas em *software*, empregadas durante a compilação do programa de aplicação; e técnicas implementadas em *hardware*, que atuam durante a execução do programa de aplicação e são implementadas pela unidade de controle do processador. A apresentação das técnicas neste trabalho seguirá esse critério de classificação.

O restante deste documento está organizado como segue. A Seção 2 descreve técnicas de predição feitas em *software*, ilustrando a implementação dessas técnicas em processadores. Em seqüência, abordam-se as técnicas de predição de desvio em *hardware*, suas classificações, implementações em

MO401 - Arquitetura de Computadores UNICAMP, Instituto de Computação

processadores e análise do desempenho em *benchmarks*. Finalmente, a Seção 4 descreve as conclusões sobre os pontos mais importantes das técnicas apresentadas.

2. TÉCNICAS DE PREDIÇÃO EM SOFTWARE

As técnicas descritas nessa seção são empregadas durante a compilação do programa de aplicação, sendo que compete ao software de suporte a tarefa de reorganizar suas instruções indicando quais serão executadas em paralelo com o comando de desvio.

2.1 Delayed Branch

Também conhecida como "*desvio atrasado*", esaa técnica consiste em reorganizar as instruções do programa aplicação, preservando sua equivalência semântica, de forma a minimizar os retardos impostos pelas instruções de desvio. O ciclo de execução de um desvio com atraso de uma unidade pode ser descrito pelos seguintes passos:

instrução de desvio
sucessor seqüencial
destino de desvio

Dizemos que o sucessor seqüencial está no *slot* de atraso de desvio, pois como o atraso é de uma unidade essa instrução é executada quer o desvio seja ou não seguido. Embora seja possível ter um desvio com atraso mais longo que uma unidade, na prática quase todos os processadores que utilizam essa técnica têm um único ciclo de atraso; para *pipelines* com penalidades maiores são utilizadas outras técnicas mais adequadas.

Uma solução trivial seria substituir o *slot* de atraso por uma instrução do tipo *NOP(No Operation)*. Porém, essa estratégia é duplamente desvantajosa, por causar uma degradação no desempenho do processador e aumentar consideravelmente o tamanho do código objeto como conseqüência dos NOP´s introduzidos.

Outros meios são abordados para tornar a instrução do sucessor seqüencial válida e útil [7]: 1) o *slot* de atraso é escalonado com uma instrução independente que existe antes do desvio; 2) o *slot* de atraso é escalonado a partir do destino do desvio; normalmente, a instrução de destino precisará ser copiada, pois poderá ser alcançada por outro caminho; 3) o desvio é escalonado a partir do *fall-through* de não seguido. A Figura 1, extraída de [3] ilustra esses três meios pelos quais o atraso de desvio pode ser escalonado. No caso (1), a instrução DADD R1,R2,R3 pode ser escalonada com o *slot* de atraso, pois o registrador R1 não influencia na condição do desvio. No caso (2), o *slot* de atraso é escalonado com a instrução DSUB R4,R5,R6, que pertence ao fluxo correto caso o desvio seja seguido. Em (3), o *slot* de atraso é escalonado com a instrução OR R7,R8,R9 que pertence ao fluxo do desvio não-seguido.

A primeira opção é sempre a melhor escolha, as outras devem ser usadas apenas quando essa não é possível. Para tornar a otimização dos casos (2) e (3) válida deve ser correto executar a instrução deslocada quando o desvio segue na direção não esperada. Por correto, queremos dizer que

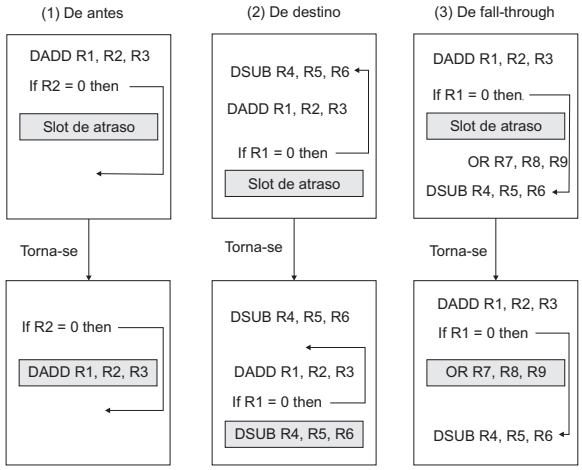

Figure 1: Escalonamento de execução do slot de atraso de desvio

o trabalho é desperdiçado, mas que o programa ainda será executado corretamente.

Apesar de bastante eficiente, essa técnica é limitada pelo estágio atual das técnicas de movimentação de código. Conforme relatado por [7], as técnicas de otimização de código para o processador MIPS conseguem preencher somente 70% dos *slots* de atraso adjacentes às instruções de desvio com retardos de um ciclo. No caso de desvios com retardo de dois ciclos a percentagem de instruções movimentadas decresce substancialmente, alcançando apenas 25%.

Vale salientar ainda que essa técnica de predição de desvio foi implementada em diversas arquiteturas do tipo RISC, como no IBM 801, no processador MIPS de Stanford, na arquitetura RISC I de Berkeley e no modelo i860 da Intel.

2.2 Branch Folding

Essa técnica, descrita em [2], consiste em fazer com que cada instrução inclua o endereço da sua sucessora. Em instruções de desvio incondicional, o endereço destino é armazenado na instrução que precede o comando de desvio, eliminando-se dessa forma a necessidade de executar instruções de transferência de controle incondicional.

Para as instruções de desvio condicional é necessário um tratamento diferenciado. Compete ao compilador especificar qual das duas instruções sucessoras do desvio terá seu endereço armazenado na instrução que antecede a transferência de controle. Empregando uma técnica de predição estática, o compilador seleciona o endereço com maior probabilidade de execução e armazena num dos bits do registrador *PSW (Program Status Word)*, na instrução antecedendo o comando de desvio.

Além do endereço armazenado na instrução precedente, o processador armazena o endereço da instrução alternativa na memória *cache*. Esse endereço será usado no caso da predição feita pelo compilador ser incorreta. Quando a instrução de desvio condicional for lida, o endereço da sucessora que foi especificado pelo compilador é transferido para o PC,

enquanto que o endereço alternativo é retido pelas instruções subseqüentes à medida que elas percorrem os estágios do *pipeline*. O endereço do contador de programa alternativo permanece retido nos diferentes estágios até que o conteúdo do código de condição especificado pelo comando de desvio esteja pronto para o teste. Caso a previsão esteja errada, as instruções em progresso são descartadas, o contador alternativo torna-se o corrente, e as instruções da outra direção são introduzidas no *pipeline*.

A técnica *Branch Folding* requer um *hardware* complexo e um compilador capaz de movimentar instruções que modificam os indicadores de condição, permitindo que a sucessora apropriada seja escolhida sempre que possível, eliminando-se a penalidade no tempo de processamento imposta pela escolha indevida da sucessora do desvio condicional. Na presença de múltiplos desvios, a movimentação das instruções que alteram os indicadores de condição torna-se uma tarefa de difícil realização, o que limita o alcance dessa técnica.

A efetividade da técnica *Branch Folding* depende principalmente das características do programa de aplicação. Essa técnica foi implementada no processador CRISP da AT&T [2] e é utilizada em outras arquiteturas como a PowerPC e IBM RISC System 6000.

2.3 In Line

A técnica *In-line* é utilizada no tratamento de instruções de desvio com retorno de subrotinas. Como um mesmo procedimento/função pode ser ativado de diferentes pontos do programa, as técnicas de predição de desvio precisam armazenar longos padrões de retornos para aumentar a taxa de acerto. Desta forma, as técnicas para desvios condicionais apresentam uma reduzida taxa de acerto quando tratam instruções de retorno de subrotinas.

A expansão *In-line* de código é uma técnica de otimização bastante eficiente [1]. Essa técnica substitui as chamadas de sub-rotinas dos programas de aplicação pelo código objeto correspondente. O tempo de execução de um programa após a aplicação da técnica *In-line* é mais eficiente pelas seguintes razões: elimina as instruções de passagem de parâmetros e descarta as instruções para chamada e retorno de procedimentos. A principal desvantagem dessa técnica é o aumento considerável do código objeto gerado pelo compilador.

3. TÉCNICAS DE PREDIÇÃO EM HARDWARE

Existem dois tipos de previsões de desvios implementados em *hardware*: as técnicas estáticas, que se baseiam em definições feitas em tempo de concepção de um novo processador; e as técnicas dinâmicas, que se baseiam nas informações coletadas em tempo de execução. As seções subseqüentes descrevem exemplos de técnicas desses dois tipos.

3.1 Técnicas Estáticas

Os previsores de desvios estáticos são utilizados em processadores nos quais a expectativa é que o comportamento do desvio seja altamente previsível em tempo de compilação. Essas técnicas também são utilizadas para auxiliar previsores dinâmicos.

3.1.1 Predição Fixada

A técnica de predição fixada possui três variações: o desvio sempre será tomado; o desvio nunca será tomado; e o código da operação determina a previsão. Nessas implementações, o resultado previsto para um desvio será sempre o mesmo.

A primeira técnica explora o fato de que a maioria dos desvios condicionais provoca uma transferência no fluxo de controle, prevendo que a sucessora de um comando de desvio é a instrução contida no endereço alvo [8]. Logo, é feita a busca da instrução do endereço alvo ao invés do trecho de código adjacente ao comando de desvio. Esse esquema tem uma taxa de média de previsões incorretas igual à freqüência de desvios não-seguidos. Em [5], o monitoramento do comportamento das instruções de desvio verificou que em média 67,6% dos desvios de um conjunto de programas de teste foram tomados. Essa técnica foi utilizada pelo IBM 360/91.

Na segunda técnica de previsão a execução segue a seqüência normal das instruções, ignorando-se o fato da instrução ser um desvio [7]. Assumindo que a transferência de controle nunca ocorrerá, a unidade de instruções continua buscando as instruções adjacentes ao comando de desvio. Essa técnica é implementada na maioria dos processadores que fazem busca antecipada de instruções *(look-ahead processors)* e foi implementada em vários processadores, como o i960CA, MC68020 e o VAX 11/780.

A terceira técnica considera o código de operação do comando de desvio para decidir se o fluxo de controle será transferido para o endereço alvo ou não, já que algumas operações de desvios têm maior tendência para um dos fluxos (como a instrução de chamada de procedimento, onde sempre ocorrerá o desvio). Essa técnica utiliza resultados de estudos sobre o comportamento dos diversos tipos de comandos de desvio [8], levando em consideração a probabilidade de o controle ser transferido.

3.2 Técnicas Dinâmicas

Esta seção se concentra no uso do *hardware* para prever dinamicamente o resultado de um desvio - a previsão dependerá do comportamento do desvio em tempo de execução e mudará se o desvio alterar seu comportamento. Usualmente, essas técnicas são mais eficientes do que as estáticas por armazenarem informações das instruções de desvio coletadas em tempo de execução.

3.2.1 Histórico do Desvio

O esquema mais simples de previsão dinâmica é um buffer de previsão de desvios [7] ou uma Tabela de Histórico de Desvios BHT *(Branch History Table)*. A idéia básica é verificar o que ocorreu com as execuções mais recentes de um desvio e com base nisso realizar uma predição do resultado que será produzido pela corrente execução do desvio.

Um buffer de previsão de desvios é uma memória pequena indexada pela porção mais baixa do endereço da instrução de desvio. No caso de uma BHT a indexação será a mesma, sendo que os históricos serão armazenados em uma tabela. Um esquema bastante simples de predição consiste em utilizar apenas o resultado da última execução da instrução de desvio. Nesse caso, um bit seria suficiente para informar se o desvio foi seguido recentemente ou não [3, 8].

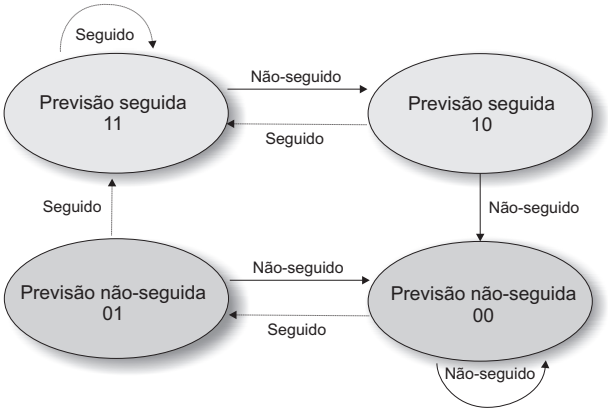

Figure 2: Máquina de estados para previsões de 2 bits

De fato, não sabemos se a previsão é correta - ela pode ter sido colocado ali por outro desvio que tenha os mesmos bits de endereço de baixa ordem utilizados da indexação. O ideal seria manter uma correspondência biunívoca entre cada entrada da BHT e cada instrução de desvio do código objeto. Contudo, limitações no espaço de armazenamento no interior do processador impedem a manutenção de uma entrada distinta para cada instrução de desvio. Apesar disso, a sugestão da previsão é considerada correta, e a busca da próxima instrução começa na direção prevista. Caso a sugestão se mostre incorreta, o bit de previsão é invertido e armazenado de volta. Uma deficiência dessa técnica refere-se a qual será o palpite adotado na primeira predição da instrução de desvio. Técnicas dinâmicas de predição são, em geral, afetadas pela predição inicial [4].

Esse esquema simples de previsão de 1 bit apresenta uma pequena perda de desempenho: ainda que um desvio seja quase sempre seguido, é provável que essa previsão seja incorreta duas vezes, em vez de uma, quando ela não for seguido. Para remediar isso, são usados com freqüência esquemas de previsão de 2 bits. Os bits que armazenam a história de um desvio definem uma máquina de estado finito que é utilizada na implementação da predição. A Figura 2, extraída de [3], apresenta a máquina de estados de um preditor de 2 bits. Nessa máquina de estado,somente se modifica a predição após dois erros consecutivos.

O esquema de 2 bits é uma especialização de um esquema mais geral que tem n bits para cada entrada no *buffer* de previsões. Com n bits, o contador pode assumir valores entre 0 e 2^{n-1}: quando o contador é maior ou igual a metade de seu valor máximo 2^{n-1} o deslocamento é previsto como seguido; caso contrário ele é previsto como não seguido. Com o esquema de 2 bits, o contador é incrementado em um desvio seguido e decrementado em um desvio não seguido. Estudos realizados com previsores de n bits mostraram que BHTs contendo de 1 a 3 bits de história apresentam taxas de acerto muito próximas do máximo valor atingido [8, 5], sendo portanto mais utilizados na prática.

3.2.2 Contadores Saturados

Esta técnica consiste em associar um contador de controle aos desvios. A idéia básica desse contador é indicar um palpite para o desvio com base em seu comportamento. Em um dado momento em que o contador for não negativo a técnica indicará que o desvio deve ser tomado. Caso contrário, a técnica indicará que o desvio não deve ser tomado e que a próxima instrução é a adjacente. Após a execução do comando de desvio, o contador de controle será incrementado caso o desvio tenha sido tomado e decrementado caso contrário. Essas operações apenas são realizadas enquanto o valor do contador não atinge um de seus valores extremos.

A taxa de acerto apresentada pelo uso de contadores de dois bits é considerada razoável sendo superada pelo contador de três bits em poucos casos, o que nos leva a concluir que contadores com limites de variação muito grandes não implicam necessariamente em percentagens mais altas de acerto. Isso ocorre porque contadores com maior capacidade normalmente apresentam uma inércia maior para neutralizar o efeito provocado por uma outra instrução de desvio mapeada na mesma entrada.

Nos experimentos descritos em [8] utilizou-se uma tabela hash com somente 16 entradas, fazendo com que diferentes desvios podem ser mapeados na mesma entrada. Desta forma, a substituição da técnica baseada em bits de história por contadores saturados aumentou as percentagens de acerto da predição.

3.2.3 Tabela de Alvos dos Desvios

Uma técnica alternativa para a predição é a que emprega uma tabela contendo os alvos das instruções de desvio. Denominada BTB *(Branch Target Buffer)*, essa tabela é uma evolução da tabela que contém a história dos desvios [7]. Como anteriormente, a BTB inclui campos para identificar a instrução de desvio e para armazenar a história das recentes execuções do comando de desvio (ou o contador saturado). Adicionalmente, a BTB inclui um campo contendo informações sobre a instrução sucessora do desvio. Geralmente o campo armazena o endereço efetivo da instrução sucessora. Em outras implementações, a própria instrução sucessora é armazenada.

A BTB torna o processador mais eficiente do que aqueles que usam simplesmente uma BHT, dado o potencial oferecido pelas informações sobre a instrução alvo do desvio. Em paralelo com a busca da próxima instrução, podemos detectar antecipadamente se ela é um desvio e se for, qual o endereço da instrução que deve suceder o comando de desvio que está sendo buscado. Através dessa antecipação, o estágio de busca pode ser prontamente redirecionado para o fluxo com maior probabilidade de execução, reduzindo a incidência de instruções introduzidas indevidamente nos estágios iniciais do *pipeline*.

O funcionamento é semelhante do mecanismo com BHT, com apenas algumas diferenças: no caso de falha, o endereço destino também é inserido na entrada juntamente com o endereço do desvio. Quando acontece uma predição incorreta, o endereço destino é atualizado para refletir o destino correto do desvio. A organização e o gerenciamento da BTB também é semelhante ao da tabela que armazena somente a história recente dos desvios. Por esse motivo, as taxas de

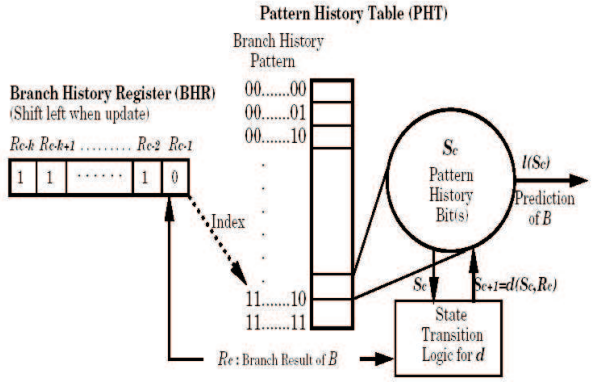

Figure 3: Estrutura de uma previsão de desvio com dois níveis de história

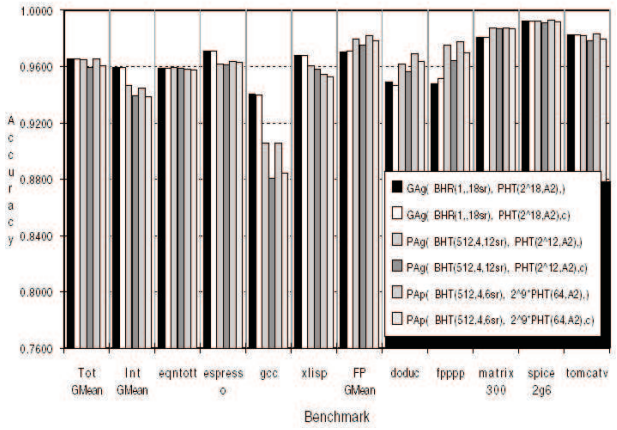

Figure 4: Desempenho dos Preditores da Família Dois Níveis

acerto das técnicas de predição baseadas na BHT e na BTB são muito próximas.

Essa técnica é implementada pelo microprocessador Pentium, lançado pela Intel no mercado em março de 1993. Nessa implementação, a predição de desvios ocorre da seguinte forma:

- 2 *buffers de prefetch* com 32 bytes cada, sendo que a cada momento, um *buffer* está processando instruções em endereços consecutivos até encontrar um desvio;
- Se BTB prevê que desvio não ocorre, *prefetch* continua seqüencialmente no mesmo *buffer de prefetch*;
- Se BTB prevê que desvio ocorre, segundo *buffer* começa *prefetch* de instruções a partir do novo endereço;
- Se a predição estava errada, os *pipelines* são esvaziados e instrução correta é buscada.

3.2.4 Dois Níveis de História

A abordagem de [9] considera que o comportamento de outros desvio deve influenciar a previsão de uma instruções de desvio específica. Essa técnica possui dois níveis de informação sobre os desvios: o primeiro nível é o histórico dos últimos k desvios encontrados e o segundo nível é o comportamento do desvio na última ocorrência s do padrão em questão.

Para manter os dois níveis de informação, essa técnica usa duas estruturas de dados, um registrador BHR *(Branch History Register)* para armazenar o primeiro nível de história; e uma tabela PHT *(Patttern History Table)* para cada padrão de bits no primeiro nível, armazenando o segundo nível, como ilustra a Figura 3 extraída de [9]. As informações armazenadas no primeiro nível são globais, podendo ser originadas por diferentes instruções de desvio. Já os históricos do segundo nível estão associados ao comportamento de uma instrução de desvio específica.

Se o desvio é seguido, então o valor 1 é gravado, caso contrário, o valor 0 é gravado. Após a execução de qualquer desvio, o registrador BHR é atualizado, deslocando-se seus bits uma posição para a esquerda (descarta-se o resultado de execução mais antigo) e o novo valor é gravado no bit mais a direita. Uma vez que existem k bits no para os registradores de histórico então 2^k padrões diferentes aparecem na BHR. Para cada um desses 2^k padrões existe uma entrada correspondente na tabela de padrões de histórico. Experimentos realizados para determinar o número k de bits do registrador BHR constataram que valores variando de 11 a 14 permitem atingir elevadas taxas de acerto a um custo aceitável pela tecnologia disponível atualmente.

Embora o esquema básico possua apenas um registrador BHR e uma tabela PHT, existem esquemas com mais de uma tabela. A família de preditores de dois níveis de histórico proposta por [9] é constituída por nove componentes que podem ser diferenciados pela forma como a informação é organizada nos dois níveis. Em [9] são utilizadas as seguintes siglas para identificar os componentes dessa família de preditores: GAg, GAs,GAp ,PAg ,PAs ,PAp ,SAg ,SAs e SAp. A Figura 4 extraída de [9] ilustra o desempenho de alguns dsses preditores no *benchmark* SPEC, que possuem média de acerto de 97% das predições.

Esta técnica de predição de desvio é implementada pelo processador Pentium Pro. Diferentemente da BTB do Pentium, a BTB do Pentium Pro é capaz de reconhecer procedimentos padrão. Cada entrada da BTB usa 4 bits para manter um histórico a respeito do procedimento tomado com aquela instrução nas suas últimas 4 execuções.

3.2.5 Preditores Híbridos

A busca por preditores mais eficientes motivou recentemente o início de investigações de mecanismos híbridos de predição. A idéia básica é considerar que diferentes técnicas de predição de desvio possuem vantagens diferentes, sendo que algumas apresentam desempenhos muito bons para casos específicos. Desta forma, uma boa técnica de predição seria obtida se pudessemos combinar as diferentes vantagens de outros métodos. Ao invés de uma única técnica de predição, preditores híbridos incluem diversas técnicas, operando em paralelo, sendo que somente a técnica com maior probabilidade de acerto fornece o resultado da predição para a

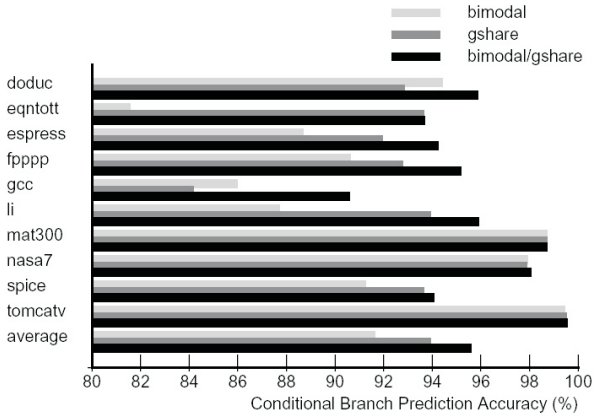

Figure 5: Desempenho do Preditor Híbrido no Benchmark SPEC 89

unidade de busca de instruções.

O preditor híbrido descrito em [6] possui dois preditores simples que são controlados por um mecanismo que seleciona dinamicamente qual das duas predições será considerada. Este preditor híbrido usa uma tabela contendo contadores saturados de dois bits. Quando chega um desvio, seu endereço indexa a tabela e dependendo do bit mais significativo do contador saturado, o mecanismo seleciona um dos preditores simples. Após a execução da instrução de desvio, o correspondente contador é incrementado ou decrementado se o desvio foi respectivamente tomado ou não.

A Figura 5, extraída de [6], ilustra o desempenho desse preditor no *benchmark* SPEC 89. Podemos observar nesse gráfico que a combinação das técnicas produz um aumento no desempenho em todos os casos. Atualmente, os preditores híbridos são os que apresentam maiores taxas de acerto, podendo atingir patamares próximos a 99%.

4. CONCLUSÕES

Ao se fazer considerações a respeito do custo e das penalidades impostas pela execução de instruções de desvio, fica clara a motivação pelo estudo de técnicas que possam minimizar os efeitos dessas instruções nos processadores modernos. Os desafios impostos por esse problema têm provocado o aparecimento de inúmeros projetos de pesquisa buscando métodos mais sofisticados de previsão de desvios para serem incorporados nas arquiteturas.

Diferentes de técnicas de predição de desvio foram descritas neste trabalho, embora existam outras não citadas aqui, baseadas nestes princípios. O desempenho alcançado pelas técnicas apresentadas possui uma relação direta com as características da aplicação em execução.

É importante ressaltar que a eficácia de um esquema de previsão de desvio depende não apenas da exatidão, mas também do custo de um desvio quando a previsão está correta e quando a previsão é incorreta. Essas penalidades de desvios dependem da estrutura do *pipeline*, do tipo de previsão e das estratégias utilizadas para recuperação de previsões erradas.

Existem basicamente duas tendências em torno dos tipos de técnicas de redução dos custos dos desvios: técnicas implementadas em *software* e técnicas implementadas em *hardware*. O argumento empregado para defesa do uso das técnicas implementadas em *software* é que o projeto do *hardware* pode ser bastante simplificado se o problema da redução de custo for transferido para um outro nível hierárquico do sistema. Por outro lado, os defensores da outra estratégia apontam que o decrescente custo do hardware justifica a implementação dessas técnicas diretamente na unidade de controle dos processadores.

Uma alternativa que tem atraído a atenção dos pesquisadores consiste em usar uma combinação desses dois tipos de técnicas para reduzir ainda mais o custo de execução dos comandos de desvio, o que tem apresentado ótimos resultados, se comparado às técnicas estáticas ou as técnicas dinâmicas em isolado. Preditores híbridos também apresentam ótimos resultados por combinarem diferentes vantagens de outros métodos de predição de desvio.

5. REFERÊNCIAS

[1] T. J. Campos. Técnicas de redução do custo de desvio por software, 1999.

[2] D. R. Ditzel and H. R. McLellan. Branch folding in the crisp microprocessor: reducing branch delay to zero. In *ISCA '87: Proceedings of the 14th annual international symposium on Computer architecture*, pages 2–8, New York, NY, USA, 1987. ACM Press.

[3] J. L. Hennessy and D. A. Patterson. *Arquitetura de Computadores - Uma Abordagem Quantitativa*. Rio de Janeiro, 2003.

[4] D. R. Kaeli and P. G. Emma. Branch history table prediction of moving target branches due to subroutine returns. In *ISCA '91: Proceedings of the 18th annual international symposium on Computer architecture*, pages 34–42, New York, NY, USA, 1991. ACM Press.

[5] J. K. F. Lee and A. J. Smith. Branch prediction strategies and branch target buffer design. pages 83–99, 1995.

[6] S. McFarling. Combining branch predictors. Technical Report TN-36, Digital Western Research Laboratory, 1993.

[7] S. McFarling and J. Hennesey. Reducing the cost of branches. In *ISCA '86: Proceedings of the 13th annual international symposium on Computer architecture*, pages 396–403, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[8] J. E. Smith. A study of branch prediction strategies. In *8 International Symposium on Computer Architecture - ISCA-81*, pages 135–148, New York, NY, USA, 1981. ACM Press.

[9] T.-Y. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *ISCA '98: 25 years of the international symposia on Computer architecture (selected papers)*, pages 451–461, New York, NY, USA, 1998. ACM Press.