Previsão de Desvios

Daniel Nicácio
Universidade Estadual de Campinas
E-mail: dnicacios@yahoo.com.br
RA: 057612

Resumo

Pipelines profundos e altas velocidades do clock exigem o desenvolvimento de previsores de desvios altamente precisos. Este artigo detalha a necessidade de previsores de desvios, apresenta seus princípios básicos e então reúne as principais técnicas desenvolvidas nos últimos anos. Estas técnicas englobam tanto esquemas estáticos quanto dinâmicos. Em cada uma delas é feita uma comparação de seu desempenho com as demais. Ao final, concluímos em que estágio de desenvolvimento os previsores de desvio estão e quais são os possíveis caminhos para sua evolução.

1. Introducão

Os *pipelines* dos processadores estão cada vez mais profundos e as taxas de *clock* cada vez maiores. Portanto, as penalidades devido a erros de previsão de desvios se tornam cada vez mais altas e mais prejudiciais ao sistema, se tornando o principal limitador para a evolução dos sistemas computacionais.

Desenvolver um processador é um exercício de compromisso entre desempenho, custo e uso de energia. Uma técnica que não requer este compromisso, ou seja, que provê um ganho para todas as três métricas é muito desejada. Uma dessas técnicas é previsão de desvios.

Com esses fatos em mente, é certo que para o desenvolvimento de um bom processador, e de uma arquitetura como um todo, a escolha de um bom método para a previsão de desvios é fundamental.

Este artigo começa explicando os principais fundamentos previsores de desvios e quais foram as primeiras técnicas utilizadas e como elas evoluíram até chegarem a previsores de desvios com taxas de erros menores do que 5%.

Em seguida é mostrado diversas pesquisas atuais e o que tem sido feito para que os previsores de desvios deixem d ser um fator limitante para a evolução da computação.

Este artigo está organizado da seguinte forma: seção 2 apresenta a motivação para o estudo de previsores. As seções 3,4 e 5 apresentam os tipos de técnicas existentes pra prever os resultados dos desvios. A Seção 6 contém diversos tópicos de pesquisa atuais e por fim, a seção 7 apresenta as conclusões e por fim, a seção 8 contém as referências bibliográficas.

2. Necessidade da previsão de desvios

Segundo [1], existem três tipos de perigos: estrutural, dados e de controle. Este último é gerado devido às dependências de controle que são freqüentemente encontradas nos programas. Em média, a cada cinco instruções, uma é operação de desvio. Dessa forma, quando queremos explorar o paralelismo do código, ou seja, emitir e executar múltiplas instruções por ciclo de *clock,* as operações de desvio rapidamente se tornam um fator limitante. De forma geral, não seria possível emitir mais de cinco instruções por ciclo, pois teríamos um desvio impedindo a emissão da sexta instrução. Portanto, eliminar as dependências de controle é crucial para emissão múltipla de instruções.

Com o intuito de evitar paradas devido às dependências de controle, existem diversas técnicas para a previsão de desvios, ou seja, técnicas para definir prematuramente o resultado de um desvio, evitando uma parada na execução do programa. A eficácia de um esquema de previsão é dada pela exatidão da previsão, o custo quando a previsão é correta e quando é incorreta. As penalidades de desvio dependem da estrutura do *pipeline,* do tipo de previsão e da estratégia usada para recuperar uma previsão incorreta.

As previsões podem ser estáticas ou dinâmicas. As previsões dinâmicas são realizadas pelo hardware durante a execução do programa, enquanto que a previsão estática de desvios permite otimizar o escalonamento das instruções. Portanto, os desvios devem ser previstos estaticamente quando o programa for compilado.

Na próxima seção serão apresentadas algumas técnicas básicas de previsão e a evolução das mesmas.

3. Técnicas de previsão dinâmica

3.1 Previsão de desvio básico e *buffers* de previsão de desvios.

O esquema mais simples é utilizar um buffer de previsão de desvios ou uma tabela de histórico de desvios. O buffer é uma pequena memória indexada pela porção mais baixa do endereço da instrução de desvio. Esta memória contém um bit informando se o desvio deve ser tomado ou não. O programa toma a direção indicada pelo bit, se mais tarde for detectado que a decisão estava errada o bit na memória é invertido.

Esta técnica possui o inconveniente de que dois desvios podem acessar o mesmo bit de previsão, ou seja, o resultado de um desvio pode influenciar na previsão de outro desvio, sendo que estes dois desvios deveriam ser independentes um do outro.

Visando evitar esta perda de exatidão, a memória passa a usar dois bits para predizer o resultado de um desvio. Dessa forma são necessárias duas previsões incorretas para que o resultado da previsão seja invertido.

Estudos mostram que utilizar mais de dois bits para a previsão não resulta em uma exatidão maior do que a apresentada com 2 bits.

3.2 Previsores de desvio com correlacionamento

Existem instruções de desvio com comportamento correlacionado a outras instruções de desvio. Dessa forma, é recomendado que um previsor de desvio analise as relações entre os desvios para um resultado mais exato. Esses previsores são chamados de previsores com correlacionamento.

De forma geral, é utilizado um previsor (m,n), ou seja, o previsor utiliza o comportamento dos últimos m desvios para escolher entre 2^m previsores de desvio, cada um dos quais é um previsor com n bits para um único desvio. Como já foi dito, n geralmente possui o valor igual a 2.

3.3 Previsores por torneio – combinação adaptativa de previsores locais e globais

A previsão por torneio considera o resultado dado por diferentes técnicas e determina qual destes é o mais apropriado para o desvio em questão. Isto é útil, pois alguns desvios dependem de informações locais, enquanto que outros são influenciados por informações globais.

3.4 Exemplo – Alpha 21264

Esta arquitetura utiliza 2 bits (indexados pelo endereço da instrução de desvio em questão) para selecionar entre o previsor global e o previsor local. O previsor global é indexado pelos últimos 12 desvios, e cada entrada deste previsor é um previsor padrão de 2 bits. O previsor local consiste em um previsor de dois níveis. O nível superior é composto por uma tabela de histórico local com 1024 entradas de 10 bits. Cada entrada corresponde ao resultado dos 10 desvios mais recentes. Esse método possibilita identificar padrões de até 10 desvios. A entrada selecionada na tabela de histórico local é utilizada para indexar um segundo nível formado por contadores de saturação de 3 bits, os quais fornecem a previsão local. No *benchmark* SPECfp95 a taxa de previsão incorreta chegou a ser menor do que 0,1%. Já no SPECint95, esta taxa ficou menor que 1,2%, ambas nos melhores casos.

4 Técnicas de previsão estática

Existem diversas técnicas para prever estaticamente o comportamento de um desvio. A mais simples delas é prever um desvio como seguido. Porém, a taxa de previsões incorretas deste método varia muito, entre 9% e 59%. Uma maneira de melhorar esta técnica é prever os desvios com base na orientação do desvio, onde os desvios com sentido inverso são definidos como seguidos e os com sentido direto como não-seguidos. Mas esta técnica ainda apresenta alta taxa de previsões incorretas, em torno de 30% a 40%.

Uma estratégia mais precisa é prever desvios de acordo com informações de perfil coletadas de execuções anteriores. Esta abordagem é vantajosa porque um desvio individual tem forte tendência a assumir um mesmo valor: seguido ou não-seguido. Com este tipo de previsão é possível manter a taxa de erros entre 5% e 22%

5 Uma técnica alternativa - instruções predicadas

Outra forma de evitar o perigo de controle causado pelos desvios é fazer com que a dependência de desvios não exista mais. Para isso, todas as instruções são executadas normalmente, inclusive os desvios, independentemente de quaisquer desvios. As instruções de desvios armazenam seus resultados em uma memória específica e as demais instruções armazenam seus resultados em registradores provisórios, os quais dependem de um predicado. Este predicado aponta para o endereço de memória no qual a instrução de desvio guarda seu resultado, quando este resultado estiver pronto, o registrador provisório saberá se seu resultado deve passar para o

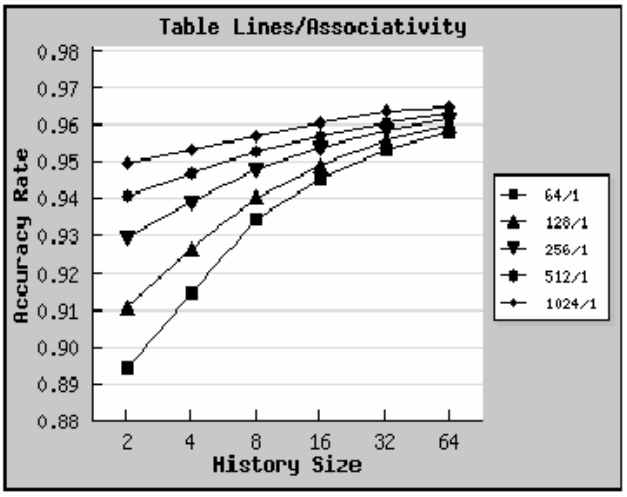

Figura 1 - TLPT Global: tabela X tamanho do histórico de desvios

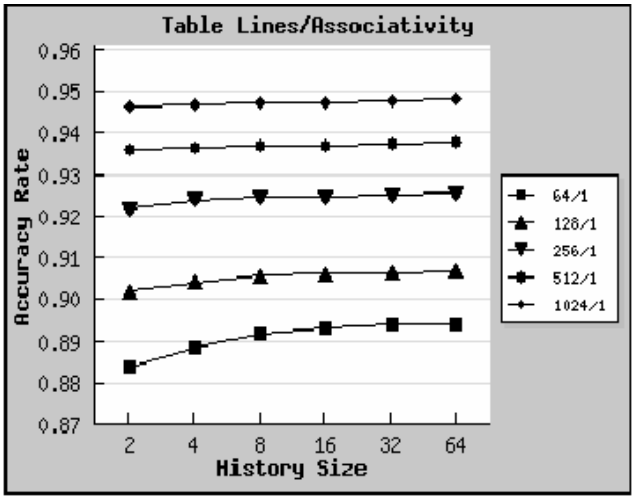

Figura 2 - TLPT Local: tabela X tamanho do histórico de desvios

registrador definitivo ou não.

Dessa forma, a dependência de controle passa a ser uma dependência de dados, a qual é mais fácil de ser evitada, tanto pelo hardware quanto pelo compilador.

6 Pesquisas recentes

6.1 O previsor 2bc-gskew

[9] relata que em uma tabela de histórico de desvios, uma interferência destrutiva ocorre quando dois desvios com resultados distintos apontam para a mesma entrada da tabela de histórico. Isto resulta no aumento da taxa de erros de previsão. Dessa forma, uma maneira de melhorar a previsão de desvios é diminuir as interferências destrutivas.

Em 1997, [2] propôs o uso de 3 bancos de historicos de desvios destintos uns dos outros, ou seja, cada um com a sua própria função de hash para indexar suas entradas. Um desse bancos é um previsor bimodal, sua função é alocar os desvio seguidos em uma tabela e os desvios não seguidos em outra tabela. Dessa forma, as interferências destrutivas diminuem drasticamente. Quando um desvio está sendo previsto, os três bancos são verificados e um "juiz" determina qual dos resultados deve ser utilizado. Esta técnica foi batizada de e-gskew.

Em 1999, [8] aprimorou seu trabalho e propôs o previsor hibrido 2bc-gskew, a idéia central deste previsor é combinar a o e-gskew com um previsor de dois bits correlacionados. O 2bc-gskew consiste de quatro bancos de contadores de 2 bits cada, sendo que um deles é o previsor bimodal.

6.2 Métodos neurais para previsão de desvios

Em Novembro de 2002, [3] apresentou um método bastante preciso para a previsão de desvios. A idéia central é utilizar o mais simples dos métodos neurais, o perceptron, como uma alternativa ao comumente usado contador de dois bits. O segredo da precisão deste método é o uso de longos históricos de desvios, isso é possível porque a quantidade de hardware necessário cresce de forma linear, e não exponencialmente, em relação ao tamanho do histórico.

Um perceptron é um dispositivo capaz de aprender e dado a ele um conjunto de valores e um conjunto de pesos (adquiridos através do aprendizado) ele é capaz de produzir um valor de saída. Neste método, cada peso representa o grau de correlacionamento entre o comportamento de um desvio passado e do desvio que está sendo previsto. Um peso positivo representa uma correlação positiva e um peso negativo apresenta uma correlação negativa.

A previsão do desvio acontece da seguinte forma: cada peso contribui na proporção de sua magnitude da seguinte maneira: se o desvio correspondente foi tomado, o peso é adicionado, senão, o peso é subtraído. Se o resultado da soma for positivo, o desvio é tomado, se for negativo, ele não é tomado. Para fazer esta solução funcionar, o histórico dos desvios usa 1 para desvios tomados e -1 para desvios não tomados. O perceptron é treinado por um algorítmo que incrementa o peso quando o resultado de um desvio concorda com o correlacionamento do peso, e decrementa o peso caso contrário.

Esse método apresentou uma taxa de erros de apenas 4,6%, o que significa uma melhora de 26%

em relação ao gshare e de 14% em relação ao previsor híbrido de McFarling (utilizado no Alpha 21264). A melhora obtida no IPC foi de 15,8% em relação ao gshare e de 5,7% em relação ao McFarling.

6.3 O uso de perceptrons em dois níveis

[4] propõem um modelo de predição de desvios utilizando um conjunto de perceptrons organizados em dois níveis. Este modelo, embasado em conceitos de redes neurais, é chamado de TLPT (Two-Level Perceptron Table). A primeira tabela do modelo possui a história dos desvios e é indexada pelo endereço do desvio. Nesta tabela, é possível identificar alguns padrões de desvios do programa. Cada entrada desta primeira tabela endereça uma entrada da segunda tabela, a qual possui os pesos sinápticos para o perceptron associado àquela historia de desvios. O modelo pode considerar a história global de desvios ou a história local.

As figuras 1 e 2 ilustram os resultados dos testes realizados.

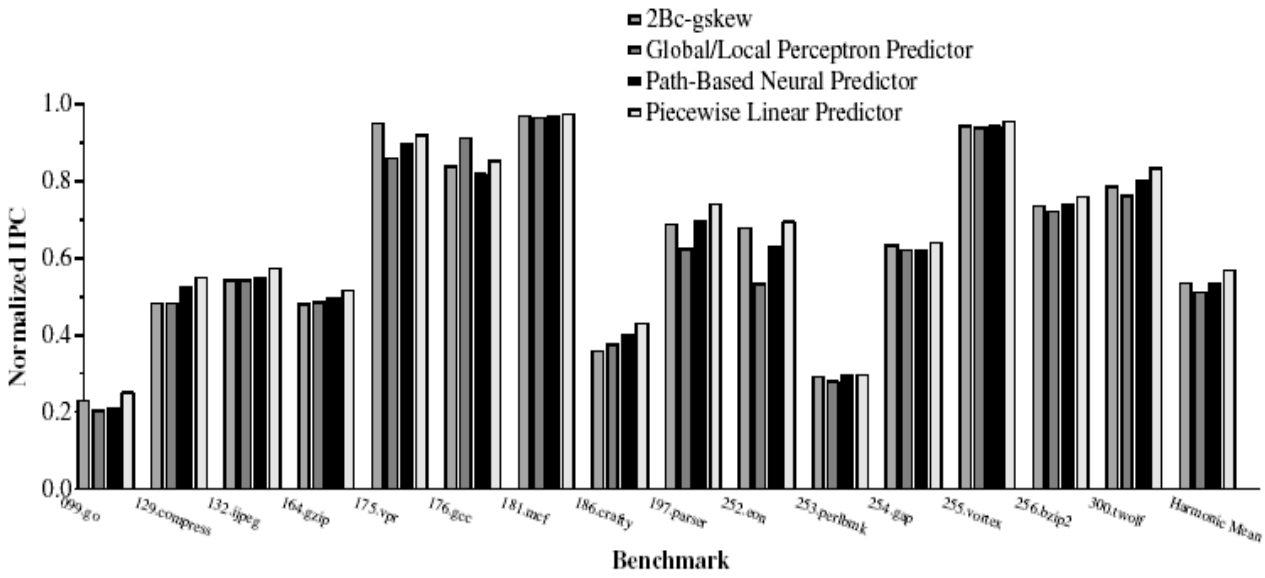

Figura 4 - IPC utilizando 256K

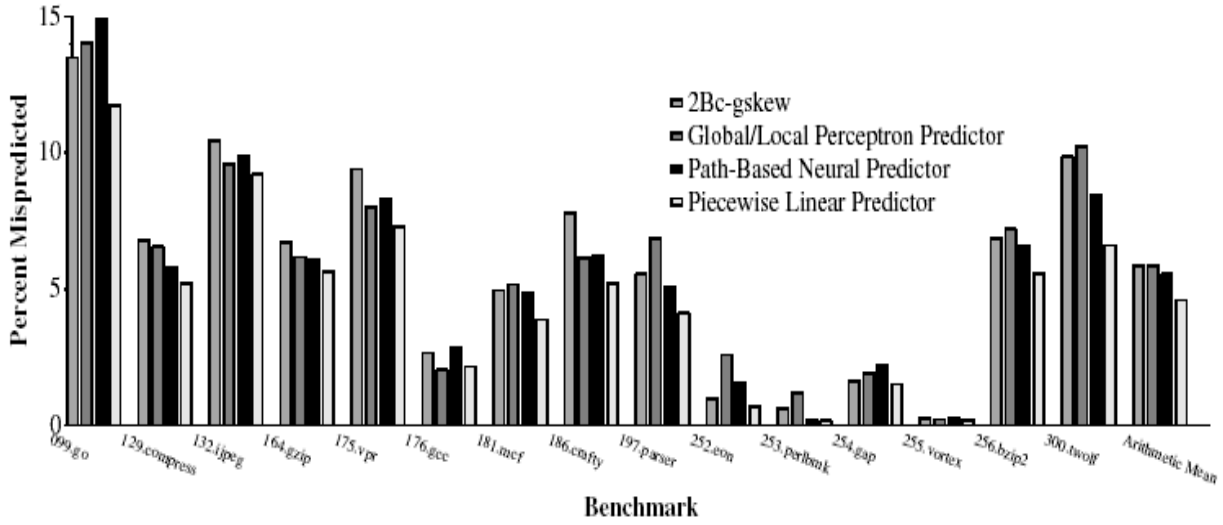

Figura 5 - Média da taxa de erros utilizando 256K

6.4 O previsor Piecewise

[5] apresentou um previsor de desvios que se baseia no aprendizado de um conjunto de funções lineares para cada desvio. Juntas, estas funções formam uma superfície plana com padrões de histórias de desvios bem definidas. Esta superfície permite identificar padrões de desvios que outras técnicas não conseguem. Por exemplo a função XOR, cujos resultados de ambas as técnicas podem ser vistos na figura 3.

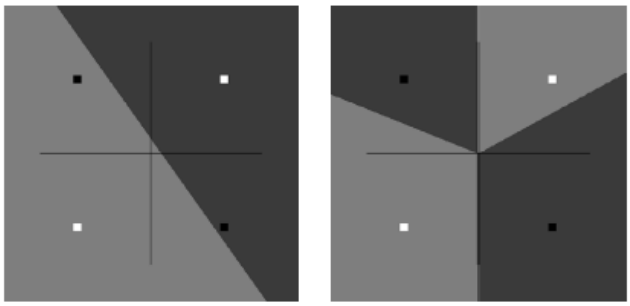

Figura 3 - A função XOR não é aprendida por um perceptron (a), mas pode ser aprendida utilizando uma superfície de decisão linear (b)

Para formar a superfície é preciso analisar todos os caminhos do programa que terminam no desvio B. Cada análise é uma função linear, e juntas elas formam a superfície. Portanto, para analisar um desvio especifico, basta ver qual o caminho utilizado para chegar neste desvio e em qual área da superfície a função deste caminho está.

Comparações entre este previsor, o 2Bc-gskew, Global/Local Perceptron e Path-Based Neural mostraram que o previsor piecewise possui a menor taxa de erros e também o maior numero de instruções executadas por ciclo. Estes resultados são vistos nas figuras 4 e 5. Trabalhos futuros visam diminuir a quantidade de hardware extra necessário para a implementação deste previsor.

6.5 Identicando desvios correlacionados através de um longo histórico global

Segundo [7], os previsores atuais possuem alguns estágios para chegarem a um resultado promissor: em um primeiro estagio, é feita a predição em apenas um ciclo, em seguida um previsor global apresenta um resultado mais eficaz e por fim um previsor ainda mais acurado corrige seletivamente o resultado dos últimos previsores. A estrutura do *pipeline* de um previsor corretivo é mostrada na figura 6.

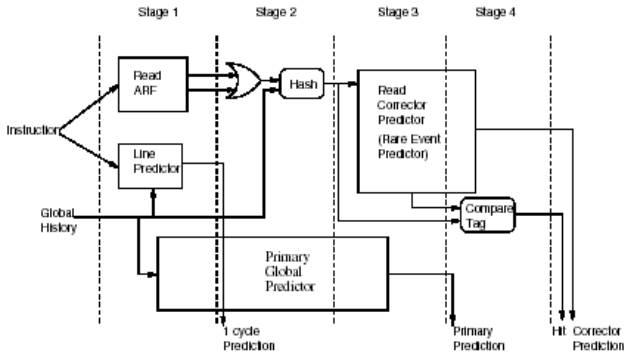

Figura 6 - Tempo de pipeline de um previsor corretor

Portanto, o último previsor deve possuir uma taxa de erros extremamente baixa. Baseando-se neste fato, foi proposta uma técnica baseada em longos históricos globais de desvios e em identificando desvios correlacionados neste histórico. Cada desvio que afeta o resultado de outro desvio é um *affector*.

Os *affectors* de cada desvio são identificados em tempo de execução, rastreando o fluxo de dados do programa no *front-end* do *pipeline*. O hardware necessário para identificar os *affectors* de 64 desvios requer apenas 312 bytes.

Dois esquemas são propostos:

Zeroing – Neste esquema, os desvios que não são *affectors* são mascarados, se tornam zeros, e portanto, não importam para a decisão do desvio. Isto pode ser feito utilizando a função AND tendo como operandos o *bitmap* dos *affectors* do desvio e o histórico do desvio. O resultado é então padronizado através de uma função de *hash* para que o previsor de desvios possa indexá-lo.

Packing – Neste segundo esquema os zeros que não afetam o resultado do previsor são retirados do resultado antes que a função de *hash* seja aplicada. Em outra palavras, os *affectors* são agrupados, e esse grupo é então formatado de forma que também possa ser indexado. A figura 7 ilustra os dois métodos.

Testes comparativos mostram que estas técnicas conseguem proporcionar uma boa melhora em previsores primários existentes (*Perceptron* e YAGS).

A figura 8 mostra o desempenho obtido por vários esquemas de previsores corretivos, e mostra também a taxa de erros obtida por eles. É importante ressaltar que com a adição de apenas 8K a um *perceptron* de 16K (total de 24K) é possível ter um desempenho comparável apenas a um *perceptron* de 256K o qual leva o dobro de ciclos para alcançar o resultado.

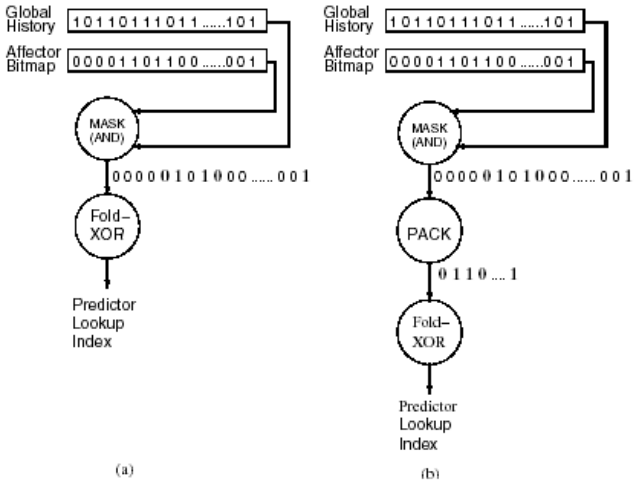

Figura 7 - Ilustração dos esquemas para utilizar informação dos "affectors" na previsão de desvios: (a) Zeroing; (b) Packing.

6.6 O previsor Profeta/Crítico (*Prophet/Critic*)

[6] introduziu uma técnica híbrida para melhorar a predição de desvios chamada de profeta/crítico (*prophet/critic*). Este esquema híbrido é composto por dois componentes, os quais desempenham o papel de profeta ou crítico. O profeta é um previsor tradicional que utiliza o histórico do desvio para predizer seu resultado. Desvios posteriores a este são considerados o futuro do desvio. O componente crítico utiliza tanto o histórico do desvio quanto o futuro do desvio para criticar se a decisão do profeta foi correta ou não.

Sendo assim, durante a execução do programa, o profeta provê um resultado para o desvio em questão; o programa segue sua execução normalmente, gerando outros resultados para novos desvios. Baseados nestes novos resultados (chamados de futuro do primeiro desvio) o crítico determina se o resultado dado ao primeiro desvio foi correto ou não. Se sim, o programa continua normalmente e o crítico continua criticando os resultados seguintes do profeta; se não, são executadas as devidas instruções para que o programa siga pelo caminho correto.

Resultados mostraram que um previsor 8K + 8K byte profeta/crítico possui taxa de erros 39% menor do que um previsor 2bc-gskew de 16K.

6 Adição de código para aumentar a precisão dos previsores de desvio

Como já foi dito, uma maneira de melhorar a previsão de desvios é diminuir as interferências destrutivas em uma tabela de histórico de desvios. Uma forma de alcançar este objetivo é particionar a tabela de histórico em duas regiões: uma com desvios que normalmente são seguidos e outra com desvios que normalmente não são seguidos. [9] propõem uma técnica em software (estática) para implementar este esquema.

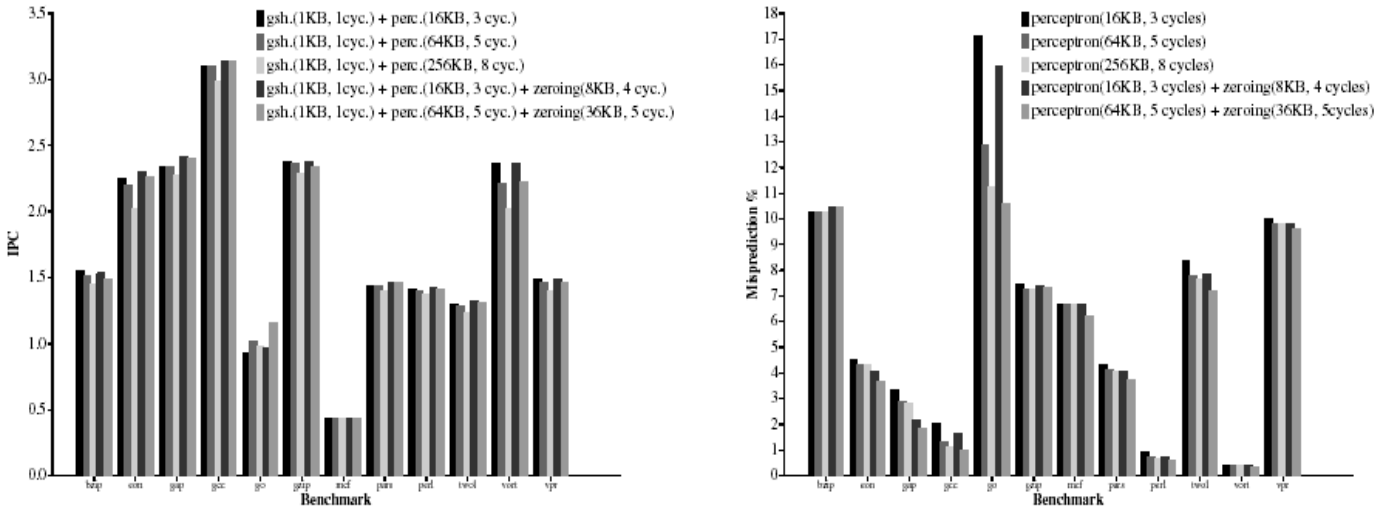

Figura 8

(i) performance de diversos previsores com correção; (ii) taxa de erros de diversos previsores com correção.

A idéia central é de que todo desvio utiliza, pelo menos, o seu bit menos significativo para indexar as entradas da tabela de histórico. Portanto, pode-se definir que o valor deste bit será sempre 0 para desvios não seguidos e 1 para desvios seguidos; dessa forma, a tabela fica particionada em duas regiões bem definidas: desvios seguidos e desvios não seguidos. Para que esses bits tenham o valor desejado, é preciso adicionar instruções não operacionais (no-ops) no código fonte, de maneira que os desvios adquiram endereços conforme a técnica enunciada anteriormente.

Para que não sejam incluídos muitos no-ops, e inevitavelmente diminuir a desempenho da máquina, é utilizada uma heurística que determina em quais regiões do código a inclusão de no-ops não afetará o fluxo de instruções da máquina.

Como geralmente é utilizado mais de um bit (pelo menos dois) para indexar a tabela de histórico de desvios, uma otimização natural desta técnica é dividir a tabela em quatro regiões ao invés de duas. Estas regiões são: desvios fortemente seguidos, desvios fortemente não seguidos, desvios fracamente seguidos e desvios fracamente não seguidos. O funcionamento da técnica é análogo ao uso de apenas dois bits.

Resultados mostram que esta técnica apresenta melhoria em relação a técnicas semelhantes. O particionamento da tabela de histórico em quatro regiões obteve, em média, uma melhora de 4,5% no *speedup* e uma taxa de erros 3,5% menor. Todos estes resultados são comparados ao uso da tabela de histórico de desvios simples.

7 Conclusões

Com o aumento da profundidade dos *pipelines* e da velocidade do clock das máquinas, fica clara a necessidade de se evitar ao máximo a penalidade gerada por previsões de desvios incorretas. Portanto, nos últimos dez anos muitas pesquisas se voltaram para esta área e muitos progressos foram obtidos. Hoje, existem previsores de desvios com taxas de erro menores do que 5%.

Diversas técnicas, tanto estáticas quanto dinâmicas, foram propostas e a maioria delas ainda apresentam boa perspectiva para sua evolução. È certo que, em um futuro próximo, teremos acesso a técnicas capazes de praticamente anular as penalidades causadas por desvios.

8 Referências

[1] Hennessy, Patterson. Arquitetura de Computadores – Uma abordagem quantitativa. 3ª Edição. Editora Campus. 2003

[2] P. Michaud, A. Seznec, e R. Uhlig. Trading conict and capacity aliasing in conditional branch predictors. In Proceedings of the 24th Annual International Symposium on Computer Architecture, June 1997. pages 292-303.

[3] D. A. Jiménez e C. Lin. Neural methods for dynamic branch prediction. ACM Transactions on Computer Systems, Volume 20 Issue 4. 2002. pages 369-397

[4] L. Ribas e R. Gonçalves. Evaluating branch prediction using two-level perceptron table. 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. Page(s):4 pp.

[5] D. A. Jiménez. Piecewise Linear Branch Prediction. Proceedings of the 32nd International Symposium on Computer Architecture. 2005. pages 12.

[6] A. Falcón, J. Stark, A. Ramirez, K. Lai e M. Valero. Prophet/Critic hybrid branch prediction. In Proceedings of the 31st annual international symposium on computer architecture. 2004. page 250.

[7] R. Thomas, M. Franklin, C. Wilkerson e J. Stark. Improving Branch Prediction by Dynamic Dataflow-based Identification of Correlated Branches from a Large Global History. Proceedings of the 30th annual international symposium on Computer architecture. Volume 31 Issue 2. 2003.

[8] A. Seznec e P. Michaud. Dealised hybrid branch predictors. Technical Report PI-1229, IRISA. 1999.

[9] D. A. Jiménez. Code Placement for Improving Dynamic Branch Prediction Accuracy. Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation PLDI '05, Volume 40 Issue 6. 2005.