Previsão de Desvios Baseada nos Tipos de Desvios e nas Probabilidades de Transição de Históricos

Zenaide C. da Silva[α], Marcos A. Cavenaghi[β], João A. Martini, Ronaldo A. L. Gonçalves
Departamento de Informática - DIN - Universidade Estadual de Maringá - UEM
Avenida Colombo, 5790 - CEP 87020-900 - Maringá - PR - Brasil.
{zecarvas, jangelo, ronaldo}@din.uem.br
[β]*Departamento de Computação - DCo - Universidade Estadual Paulista - UNESP*
AV. Eng. Luis Edmundo Carrijo Coube, s/n - CEP 17033-360 - Bauru - SP - Brasil.
marcos@fc.unesp.br

Resumo

As arquiteturas superescalares possuem a habilidade de explorar o paralelismo em nível de instruções. Para isso, técnicas de previsão de desvios são necessárias para tratar as dependências de controle, agilizando a busca de instruções e aumentando o número de instruções úteis disponíveis para a execução paralela. Atualmente, a maioria dos previsores de desvios usa alguma forma de tabela contendo os históricos dos desvios e os endereços alvos a serem seguidos. Sabe-se que estes históricos geram diferentes padrões que se repetem com probabilidades que dependem do fluxo de execução dos programas. O previsor PPM (Prediction Partial Matching), o qual trabalha sobre as probabilidades dos padrões de desvios, foi analisado e serviu de base para o desenvolvimento de um modelo mais agressivo, denominado PPDT (Previsor com Probabilidade Dependente de Transição). Esse novo modelo foi simulado e avaliado sobre a plataforma SimpleScalar Tool Set. Os resultados obtidos sobre benchmarks *do SPEC 2000 alcançaram taxas médias de acerto acima de 95% em muitas situações, atingindo picos de 98% para tamanhos de históricos de 16 bits. O modelo PPDT se mostrou mais eficiente do que o PPM e apropriado para implementação real no futuro breve.*

1. Introdução

Atualmente, as arquiteturas superescalares são utilizadas na maioria dos processadores comerciais, permitindo a exploração do paralelismo em nível de instrução e possibilitando que instruções independentes possam ser executadas em paralelo. Elas aumentam o desempenho na execução das aplicações convencionais, de forma implícita tanto para o programador quanto para o compilador. Os processadores Pentium da Intel, Alpha 21264 da DEC, MIPS R10000, Athlon da AMD e o PowerPC da Motorola, entre outros, são alguns exemplos de processadores que empregam essas arquiteturas [1].

Entretanto, o desempenho desses processadores é limitado pelas instruções de desvio, que podem alterar o fluxo de execução de um programa e com isso dificultar a detecção de instruções paralelas visto que esta habilidade depende do conhecimento sobre qual caminho será seguido após a busca de cada instrução de desvio. Esta situação é conhecida como dependência de controle. Este problema requer investigação, pois é conhecido que o percentual de instruções de desvio pode alcançar 25% do total das instruções em diversas aplicações convencionais, ou seja, a cada 4 instruções uma pode ser de desvio [2].

Para minimizar esse problema e aumentar o despacho de instruções por ciclo, os processadores superescalares utilizam mecanismos de previsão de desvios para estimar o caminho que será seguido [3], permitindo buscar instruções com menos paradas, preencher o pipeline com instruções úteis e executar instruções antecipadamente e paralela mente.

Apesar da utilização destes mecanismos, o aumento do número de instruções úteis buscadas também aumenta o número de instruções de desvio que deve ser previsto e conseqüentemente os erros na previsão. Infelizmente, cada erro de previsão degrada muito o desempenho do processador, pois este fica com a cache em situação desfavorável (contendo instruções inúteis que substituíram as instruções úteis) e ainda

[α] Mestra pelo Programa de Pós Graduação em Ciência da Computação da UEM
Suporte Financeiro da CAPES e CNPq

deve perder tempo com o descarte das instruções já executadas especulativamente e o reinicio da nova busca de instruções no caminho correto. Estes problemas colaterais motivam cada vez mais as pesquisas na área, onde pequenos ganhos no acerto da previsão já justificam o desenvolvimento de novos previsores de desvios [4].

Levando-se em consideração que os desvios existentes nos programas geram diferentes padrões e que estes padrões possuem diferentes probabilidade de ocorrência, o previsor PPM [5] tem sido utilizado para aproveitar esta característica [6]. Neste presente trabalho, o previsor PPM foi previamente simulado e analisado sob diferentes condições, e um previsor mais agressivo e eficiente, chamado PPDT, foi proposto e avaliado. O previsor PPDT executa menos passos que o PPM e atua sobre os diferentes tipos de instruções de desvio, provendo melhor desempenho.

O texto está organizado da seguinte forma. A sessão 2 apresenta o estado da arte em previsão de desvios. A sessão 3 apresenta o previsor PPM e os resultados de simulações realizadas. A sessão 4 apresenta o previsor PPDT proposto. A sessão 5 analisa seu desempenho e mostra os benefícios sobre o previsor PPM. A sessão 6 apresenta as conclusões e os trabalhos futuros para a continuidade da pesquisa. As referencias bibliográficas aparecem na última sessão.

2. Previsão de desvios: o Estado da Arte

Os previsores de desvios mais utilizados são dinâmicos e são baseados em alguma tabela do tipo BTB – *Branch Target Buffer* (buffer de alvos de desvios) [7, 8, 9]. Esta tabela funciona como uma pequena cache associada ao estágio de busca do pipeline onde cada entrada apresenta basicamente três campos: o endereço de origem do desvio, a informação para a previsão e o endereço alvo do desvio.

A previsão é feita comparando o endereço da instrução de desvio a ser prevista com os endereços de origem contidos na BTB. Caso ocorra uma correspondência, a informação de previsão e o endereço alvo associados são obtidos da BTB e usados na previsão, de acordo com alguma heurística. Se o desvio é previsto como tomado, a busca de instruções é direcionada para o endereço alvo, caso contrário, a busca continua no caminho natural do código. Quando o desvio é resolvido, a previsão é conferida com o resultado real da execução. Caso esteja correta, o processador continua sua execução normal, senão, o processador atualiza os campos da BTB, esvazia o pipeline e redireciona a busca no caminho correto.

Vários trabalhos foram feitos no sentido de estudar e propor mecanismos para a previsão de desvios. Um dos trabalhos mais importante foi feito por Smith [10], que avaliou várias técnicas de previsão de desvios, onde a melhor técnica atingiu uma média de 92,55% de acerto. Dez anos depois de Smith, Yeh e Patt [11] propuseram uma técnica de previsão de desvios baseada em dois níveis (two-level adaptative training), cujo algoritmo de previsão se adapta de acordo com as informações em tempo de execução. Várias conFigurações desta técnica foram simuladas e comparadas com outros esquemas, sobre nove programas SPEC, atingindo 97% de acerto contra 93% das demais técnicas. Posteriormente Yeh e Patt [12, 13] propuseram alterações a que chamaram de forma abreviada de GAg, PAg e PAp e que estão demonstradas na Figura 1.

No previsor GAg há um registrador de histórico global e uma tabela global de padrão de histórico, onde todas as previsões são baseadas, e que são atualizadas depois que cada desvio diferente é resolvido. No previsor PAg, para coletar informações individuais do histórico do desvio, há um registrador de histórico associado a cada desvio distinto, estes históricos estão contidos numa tabela de histórico acessada pelo endereço do desvio e que indexam uma tabela global de padrão de histórico, onde todas as previsões são baseadas. Depois que cada desvio é resolvido ele atualiza sua própria entrada na tabela de histórico e a tabela global de histórico de padrões, podendo ocorrer interferência de vários desvios na mesma entrada da tabela de padrões. No previsor PAp, além da tabela de histórico de desvios por endereço, cada desvio tem sua própria tabela de padrões, para evitar a interferência na tabela de padrões. Estas tabelas são agrupadas em um conjunto chamado: tabelas de histórico de padrões por endereço.

Figura 1 – Variações da previsão em dois níveis

Estudos feitos em [14, 15, 16] mostraram as limitações impostas pelos desvios na obtenção do paralelismo em nível de instrução, provando que a taxa de acerto na previsão de desvios é um fator decisivo na ampliação da porção de paralelismo em nível de instrução que pode ser explorado. Assim, prever o

desvio é uma maneira de assegurar um fluxo mais constante de instruções úteis nos estágios do pipeline, possibilitando a execução de um maior número de instruções por ciclo.

Em 1999, Gonzalez [17] apresentou uma nova técnica chamada Previsão de Desvios Através da Previsão de Valores (Branch Prediction through Value Prediction – BPVP). Resultados das avaliações feitas com esta técnica mostraram uma redução significante na taxa de erro de previsão (misprediction) implicando em aumento no ipc. Sherwood e Calder [18] trabalharam com desvios que são difíceis de prever através dos métodos de dois níveis tradicionais: os desvios finalizadores de laços, os quais podem causar uma grande quantidade de erros de previsão em programas com laços altamente aninhados.

Fern et al. [19] propuseram um modelo de seleção dinâmica de características de desvios para que os previsores baseados em tabelas possam conter históricos mais relevantes dependendo de cada tipo de desvio, provendo economia do espaço de armazenamento se comparado com as abordagens baseadas em tabelas comumente utilizadas.

Aragon et al. [20] propuseram um avaliador de confidencia para ser usado em conjunto com qualquer previsor de desvios. O avaliador é capaz de informar se a previsão realizada possui baixa ou alta confidência. Para os desvios de baixa confidência eles propuseram uma Unidade de Inversão de Previsão de Desvios. O mecanismo proposto possui maior efeito para desvios com baixa correlação com históricos passados tais como os desvios finalizadores de laços. Os resultados de simulações mostraram em alguns casos uma redução da ordem de 15% de erro de previsão para benchmarks inteiros do SPEC2000 e aumento de desempenho da ordem de 9% em processadores superescalares. A principal conclusão é que utilizar o avaliador proposto é melhor do que simplesmente aumentar o previsor.

Em 2001, Gonçalves et al [4] investigaram o impacto da previsão de desvios em arquiteturas superescalares e SMT, mostrando que acertos acima dos 95% podem causar impactos da ordem de 18% de melhoria no desempenho dos processadores por eles detalhados. Recentemente, previsores baseados em redes neurais têm sido investigados em [21]. Outros trabalhos sobre a previsão de desvios foram desenvolvidos por McFarling [22], Young [23] e posteriormente Kesller [24]. McFarling desenvolveu um mecanismo de previsão de desvios que combinando outros mecanismos permite uma taxa de acerto de até 98.1%. Young analisou esquemas para previsão de desvios correlacionados. Kesller apresentou o mecanismo de previsão real do Alpha 21264.

3. Previsor PPM

Técnicas modernas de compressão de dados podem representar um número grande de símbolos binários, freqüentes em uma cadeia, através de um número pequeno de bits e assim reduzir o volume total dos dados. Estas mesmas técnicas têm sido utilizadas para a previsão de desvios condicionais [25, 26] pois a heurística por elas utilizadas geralmente define um modelo probabilístico que prevê o próximo símbolo da cadeia com uma alta precisão.

Neste contexto surge o previsor PPM (*Prediction Partial Matching*) [6], que codifica os resultados dos desvios como bits: 1 (tomado) ou 0 (não tomado) e tenta prever o valor do próximo bit a partir da seqüência anterior que já foi observada. O previsor PPM de ordem N é formado por $N+1$ previsores de Markov, de ordens N, N-1, ..1 e 0.

Um previsor de Markov de ordem 0 prevê o próximo desvio sem considerar nenhum histórico anterior, ou seja, é um previsor estático. Um previsor de Markov de ordem K prevê o próximo desvio baseando-se nos K desvios mais recentes (K últimos desvios resolvidos até o momento), os quais constituem um histórico atual dos desvios. O próximo desvio a ser resolvido entrará a direita desta seqüência e fará com que o bit mais a esquerda seja deslocado para fora do histórico atual. A cada passo, o histórico é atualizado no seu bit mais a direita e, sucessivamente, diferentes históricos vão surgindo com a execução do previsor Markoviano.

Durante a execução dos programas, os diferentes históricos se repetem e com isso formam diferentes padrões os quais possuem diferentes freqüências de ocorrência. Com isso, estatisticamente, cada padrão de desvios possui uma diferente probabilidade de ocorrência. Sabe-se que o histórico atual, após a solução do desvio corrente, poderá levar a 2 possíveis históricos atuais (históricos candidatos), dependendo do resultado do desvio ser *1* ou *0*. Pode-se então dizer que a transição do histórico atual para o próximo histórico atual ocorrerá com certa probabilidade.

O previsor Markoviano constrói a freqüência de transição registrando em uma tabela de contadores o número de vezes que ocorre 1 ou 0 a partir de cada histórico real que aparece durante a execução do programa. Esta tabela é indexada pelos diferentes tipos de históricos. Para prever um desvio o previsor Markoviano indexa a tabela com o histórico atual e usa a transição mais freqüente. As Figuras 2 3 exemplificam como o previsor de Markov trabalha.

Neste exemplo, a seqüência dos últimos desvios é *010101101*. Para um previsor de Markov de ordem 2, o próximo bit será previsto baseado no histórico *01* (final da seqüência). Pela tabela da Figura 2, o padrão *01* já ocorreu 3 vezes, sendo que o bit *0* seguiu este padrão duas vezes e o bit *1* uma vez. Nesse caso, o previsor Markoviano preverá o próximo bit como sendo 0 com uma probabilidade de 2/3. O Diagrama de Transição Markoviano de 4 estados (incompleta) é mostrado à esquerda da Figura 2. Os arcos que ligam os estados da cadeia são construídos quando um estado segue o outro. Cada arco contém um par de números, onde o primeiro número indica o próximo bit previsto e o outro mostra a freqüência daquela transição.

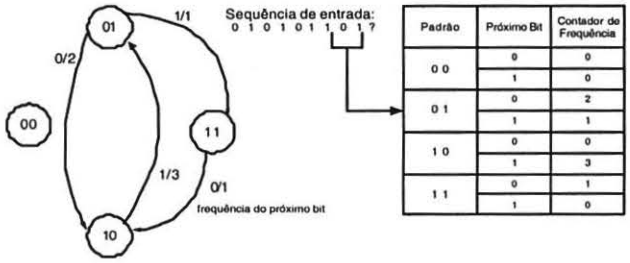

Figura 2 – Previsor de Markov de ordem 2

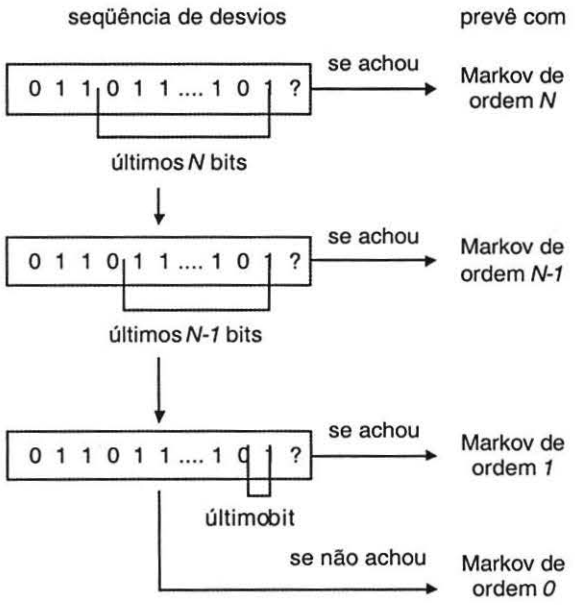

Figura 3 – Fluxo de Controle do PPM de ordem *N*

O previsor PPM de ordem *N* tenta primeiramente prever usando um previsor de Markov de ordem *N*. Caso ela não obtenha sucesso, ou seja, o histórico atual de *N* bits ainda não ocorreu nenhuma vez (contém *0* nos contadores dos campos *0* e *1*), o previsor PPM tenta então prever usando um previsor de Markov de ordem *N-1* sobre um histórico também de *N-1* bits. Caso ele ainda não tenha sucesso, o previsor PPM reduz novamente um bit do histórico e assim sucessivamente, até que a previsão correspondente possa ser feita, na pior das hipóteses pelo previsor de Markov de ordem *0*. Esta característica do previsor PPM é aqui chamada de multiplicidade de ordem e é representada na Figura 3.

3.1. Análise de desempenho do PPM

Para melhor entender o desempenho do previsor PPM, o mesmo foi experimentado com o simulador *sim-bpred* da plataforma *SimpleScalar Tool Set* [27] e avaliado sobre 13 benchmarks do SPEC 2000 [28], sendo 7 programas de inteiros (*gzip, vpr, cc1, mcf, parser, bzip2 e twolf*) e 6 programas de ponto flutuante (*mgrid, swim, applu, equake, apsi e art*). Os tamanhos dos históricos foram variados de 2 a 16.

O gráfico da Figura 4 mostra o desempenho do previsor PPM, em percentual de acerto, para os benchmarks de ponto flutuante. A curva média entre as taxas de acerto de todos os benchmarks para todos os tamanhos de históricos está acima de 83%, alcançando 97% para histórico de 16 bits. O benchmark *equake* foi o responsável pela pior taxa de acerto (60%), para históricos de 2 bits, mas apresentou um crescimento considerável quando o histórico passou de 2 para 16, alcançando 97%. Os programas *mgrid* e *apsi* destacaram-se alcançando resultados superiores a 98% para históricos de 16 bits.

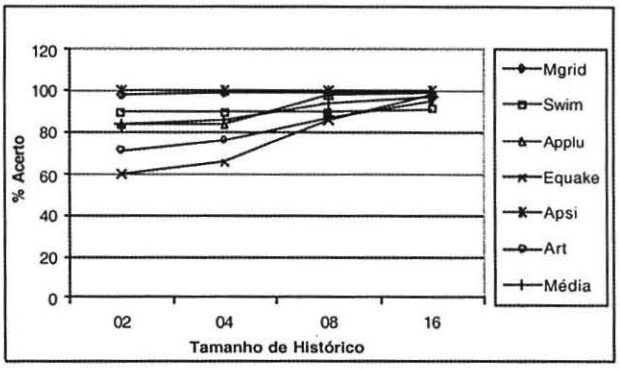

Figura 4 - Desempenho do PPM para ponto flutuante.

Apresenta-se no gráfico da Figura 5 o desempenho do previsor PPM para os benchmarks de inteiros, que de uma forma geral alcançaram desempenhos inferiores, em relação aos benchmarks de ponto flutuante. Neste caso, a curva média entre as taxas de acerto de todos os benchmarks para os diferentes tamanhos do histórico são todas superiores a 68%, alcançando 93% para histórico de 16 bits. O benchmark *cc1* foi o responsável

pela pior taxa de acerto; pouco mais de 60%. O programa mcf destacou-se alcançando resultados superiores a 98% para históricos de 16 bits.

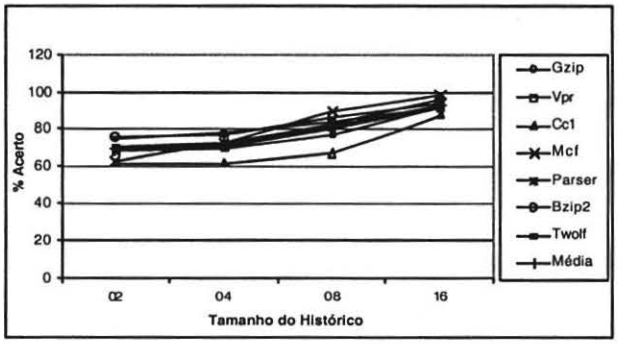

Figura 5 - Desempenho do PPM para inteiros.

O gráfico da Figura 6 apresenta a média geral entre todos os benchmarks (ponto flutuante e inteiros) para os diferentes tamanhos de histórico. O gráfico mostra uma taxa média sempre ascendente, conforme o tamanho do histórico aumenta. Para o tamanho de histórico de 16 bits a média de acerto atinge 95%.

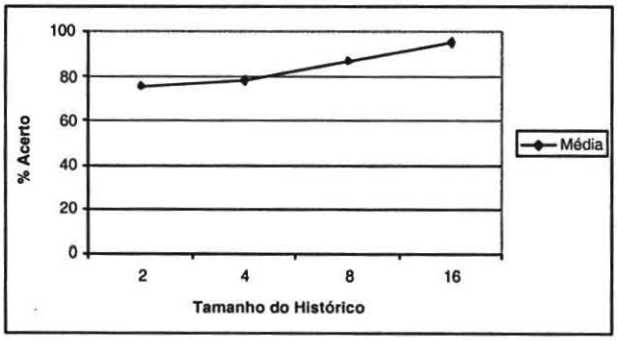

Figura 6 - Desempenho geral do PPM.

4. Previsor PPDT

Durante as simulações com o previsor PPM experimentou-se desativar o uso da multiplicidade de ordem e este fato não implicou em perda de desempenho. Notou-se então que o previsor PPM de ordem N precisa de um único previsor de Markov de ordem N para alcançar o mesmo desempenho. Ainda nos experimentos com o previsor PPM, foi observado que os diferentes tipos de desvios implicavam em diferentes taxas de acerto do previsor. Com base nesta análise, foi desenvolvido o previsor PPDT (Previsor com Probabilidade Dependente de Transição), que utiliza vários previsores de Markov, um para cada tipo de desvio.

Este modelo, representado na Figura 7, utiliza um registrador de histórico global, que contém N bits para guardar os resultados das direções dos últimos N desvios do programa, independentemente de endereço ou tipo do desvio, e várias tabelas de probabilidades, uma para cada tipo de desvio. Durante a previsão, o registrador de histórico global é analisado e o bit mais significativo é extraído, chamado de bit de transição, o qual será usado para indexar uma das colunas da tabela de probabilidades associada ao respectivo tipo da instrução de desvio. O bit de transição é necessário pois cada um dos 2 históricos candidatos pode ser gerado a partir de dois outros históricos. Por exemplo, o histórico candidato 1010 pode ser gerado a partir dos históricos 0101 ou 1101. Isto ocorre porque o bit mais significativo deste histórico é descartado. Isto quer dizer que não basta saber as probabilidades de ocorrência dos históricos candidatos, mas sim as probabilidades de ocorrências das transições que levam à geração dos históricos candidatos. Estas transições podem ser detectadas pelo bit descartado, ou seja, ocorre uma transição quando o bit descartado é 0 e outra quando o bit descartado é 1.

A tabela de probabilidades é analisada para ver quais destes dois possíveis históricos candidatos possuem a maior probabilidade de ocorrência. Caso o histórico candidato obtido com a suposta inserção de 0 seja o de maior probabilidade, o previsor então prevê 0 (desvio não-tomado), caso contrário o previsor prevê 1 (desvio tomado). Note que cada tipo de desvio tem sua própria tabela de probabilidades, mantendo cada uma 2^n entradas com duas colunas, a qual armazena as probabilidades de ocorrência dos possíveis históricos, uma para cada transição.

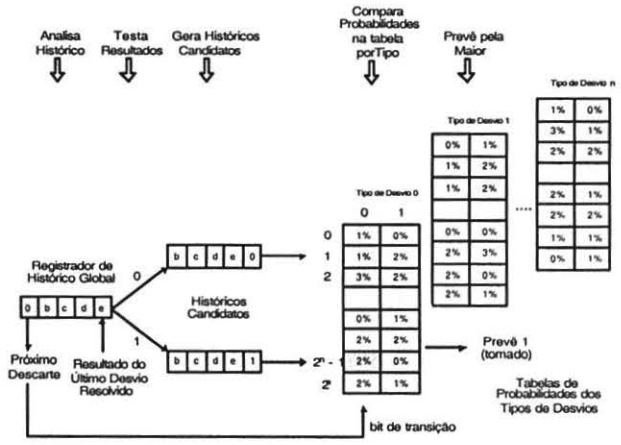

Figura 7 - Organização do modelo PPDT.

5. Simulação e avaliação de desempenho

O previsor PPDT foi simulado nas mesmas condições que o previsor PPM apresentado na sessão

anterior. Os gráficos das Figuras 8, 9 e 10 apresentam os desempenhos do previsor PPDT para os benchmarks de ponto flutuante, inteiros e ambos. O comportamento do previsor PPDT para todos os benchmarks é semelhante àquele obtido com o previsor PPM, embora apresente melhor desempenho.

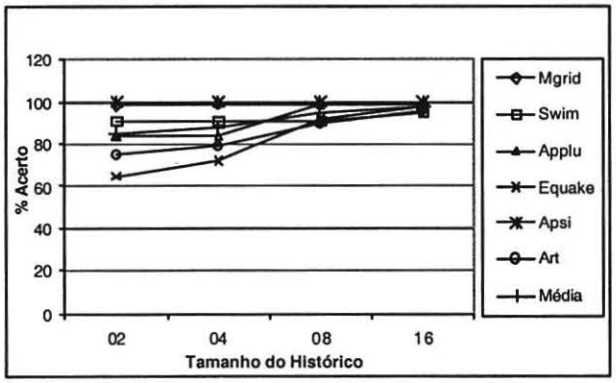

Figura 8 - Desempenho PPDT para Ponto Flutuante.

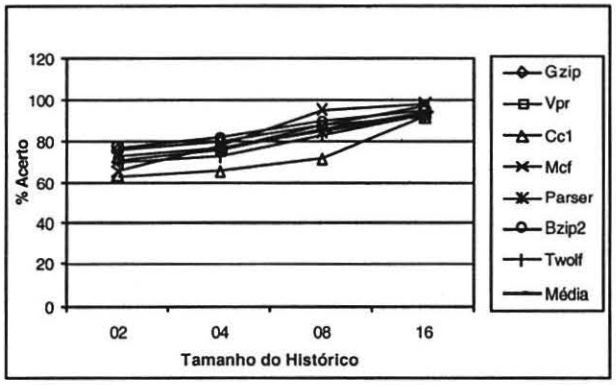

Figura 9 - Desempenho PPDT para Inteiros.

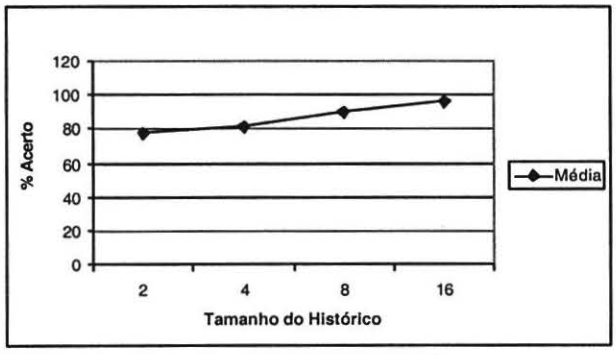

Figura 10 - Desempenho global do PPDT Tipificado

Na média dos benchmarks, o previsor PPDT se mostrou mais eficiente que o previsor PPM, para todos os tamanhos de históricos. A Figura 11 mostra o ganho (*speedup*) do previsor PPDT sobre o previsor PPM para os diferentes tamanhos de históricos. Pode-se observar um ganho acima de 4% para os tamanhos de histórico de 4 bits. É importante salientar que o PPDT substitui o esforço computacional antes utilizado pelo PPM para manter a sua característica de multiplicidade de ordem pelo gasto adicional de tabelas por tipo de desvio.

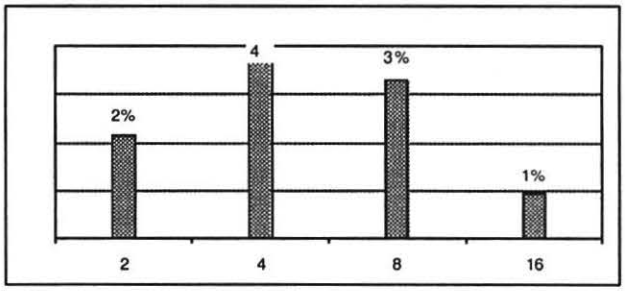

Figura 11 – Ganho do PPDT sobre o PPM.

Se os tamanhos das entradas das tabelas puderem ser reduzidos, então o gasto adicional poderá não ser significativo. O efeito seria o mesmo que substituir uma tabela larga usada pelo PPM por várias tabelas estreitas usadas pelo PPDT, com melhor desempenho. A próxima sessão investiga estas questões.

5.1. Explorando os limites de hardware

Nas simulações descritas anteriormente, os tamanhos das entradas das tabelas do modelo PPDT, que registram as freqüências das transições, não eram estipulados, operando sem limites, o que de fato é irreal. Para avaliar o impacto desta limitação no desempenho do previsor PPDT, novas simulações foram feitas para tamanhos de entradas que variam de 2 a 16 bits, pois com 16 bits os resultados dos desempenhos já não mais são alterados. Com esta variação, os valores das freqüências podem alcançar valores máximos de 3 a 65535, chamados de *Max*. Somente o histórico de 16 bits foi considerado pelo fato de prover os melhores desempenhos.

Adicionalmente, uma nova característica foi inserida no previsor PPDT: a reinicialização (*reset*) das freqüências. Esta característica funciona tal como segue. Após a solução do desvio, a entrada da tabela referente ao histórico ocorrido deve ser incrementada. Entretanto, este incremento somente será feito se o valor da entrada já não atingiu o valor *Max* (limite máximo da freqüência - todos os bits em 1). Após este incremento, ainda é verificado se o valor dessa entrada e o valor da entrada associada ao histórico candidato que não ocorreu são simultaneamente iguais ao valor *Max*. Caso sejam, os valores dessas entradas são "zerados". Esta versão do previsor é chamada de

previsor $PPDT_R$. Os resultados das simulações são apresentados nos gráficos a seguir.

O gráfico da Figura 12 mostra o desempenho do previsor $PPDT_R$ para os programas de ponto flutuante. Nota-se que para os tamanhos de *Max* de 7 a 1023 as taxas de acerto se mantêm altas. A partir do tamanho 2047 as taxas de acerto começam a diminuir, mostrando que com esses tamanhos o desempenho fica comprometido. O tamanho de *Max* que apresentou maior taxa de acerto foi 31, alcançando 98,45%. Para este caso, não se justifica o uso de mais do que 5 bits em cada entrada da tabela.

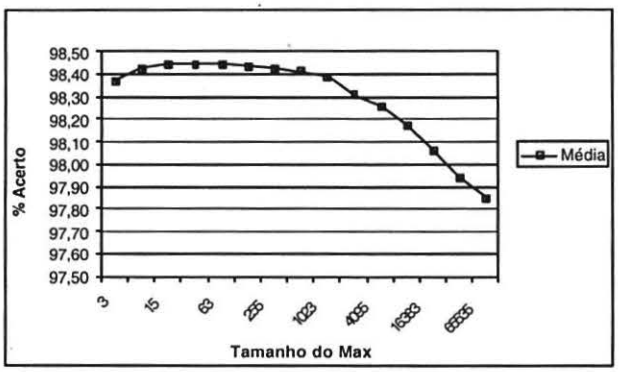

Figura 12 - Desempenho do $PPDT_R$ para pto flutuante

No gráfico da Figura 13 são mostrados os resultados das taxas de acerto do previsor $PPDT_R$ para os programas de inteiros. Observa-se que as taxas de acerto a partir do tamanho *Max* de 15, ou seja 4 bits em cada entrada da tabela, começam a diminuir. É justamente neste ponto que o previsor atinge a maior taxa de acerto, alcançando 95,18%.

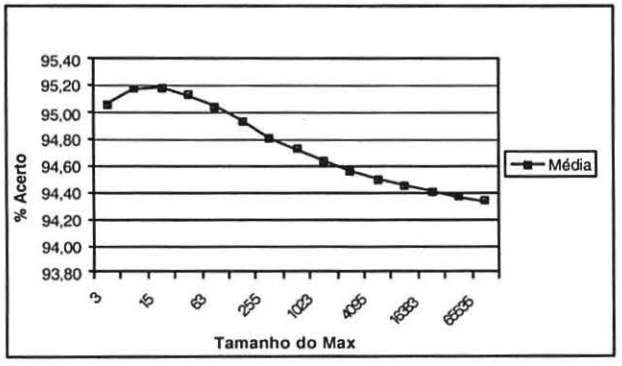

Figura 13 - Desempenho do $PPDT_R$ para inteiros.

O gráfico da Figura 14 apresenta o desempenho médio do previsor $PPDT_R$ para todos os programas. Pode-se observar que o melhor desempenho médio, 96,69%, é alcançado para *Max* de 15, isto é, com contadores de apenas 4 bits. As taxas de acerto começam a diminuir conforme o tamanho de *Max* aumenta.

O percentual de ganho obtido do previsor $PPDT_R$ com o valor *Max* de 15 sobre o mesmo previsor com tamanho ilimitado alcança 0,8%. Isto significa que mesmo que fosse possível dispor de tabelas enormes, de nada adiantaria pois o aumento descontrolado dos valores das freqüências prejudica o desempenho. Este fato ocorre porque o contador de freqüência preserva valores desatualizados, alcançados em partes anteriores do código dinâmico que não mais se aplicam a situação atual da execução. Neste sentido, a reinicialização após certo tempo melhora o desempenho pelo fato de acabar com o "vício" dos contadores de freqüência. O mais importante é que o uso de limites para os tamanhos das entradas das tabelas de probabilidades promove a redução do volume de hardware significativamente.

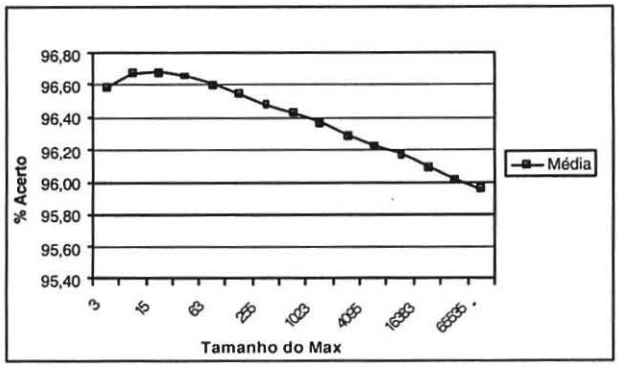

Figura 14 - Desempenho global do previsor $PPDT_R$.

6. Conclusões

Os processadores modernos sofrem penalidades em função das instruções de desvio existentes nos códigos das aplicações. Querer mudar os códigos existentes para remover estas instruções é absolutamente impossível devido a pelo menos 3 razões. Primeiro, porque não sabemos programar de forma incondicional e exata. Segundo, porque reescrever todas as aplicações existentes demandaria esforço inalcançável. Terceiro, alterar todos os compiladores seria igualmente inviável. Uma alternativa aceitável é prover mecanismos de hardware que permitam aos processadores tratar os desvios de forma inteligente e com alto desempenho, aproveitando todo o software já existente de forma implícita.

O previsor PPDT foi proposto e avaliado sobre benchmarks do SPEC 2000, apresentando taxas médias de acerto superiores 95% e melhores do que o predecessor PPM, atingindo picos de 98%. Pôde-se

observar que o aumento do tamanho do histórico causa um aumento proporcional no desempenho e que não adianta utilizar tabelas com entradas de tamanhos exagerados. Constatou-se que com contadores de freqüência de apenas 4 bits as taxas de acerto atingem o máximo esperado.

7. Referencias bibliográficas

[1] Gonçalves, R. A. L. e Navaux, P. O. A. Arquiteturas Multi-Tarefas Simultâneas: SEMPRE – Arquitetura SMT com Capacidade de Execução e Escalonamento de Processos - Porto Alegre: PPGC do II/UFRGS, Tese, 2000. 136p.

[2] Fernandes, E. S. T.; Santos, A. D. Arquiteturas Super Escalares: Detecção e Exploração do Paralelismo de Baixo Nível. In: ESCOLA DE COMPUTAÇÃO, 7., 1992, Gramado-RS. Anais... [S.l.:s.n.], 1992.

[3] Smith, J.E.; Sohi, G.S. The Microarchitecture of SuperScalar Processors. Proceedings of the IEEE, [S.l.], v.83, n.12, Dec. 1995.

[4] Gonçalves, R. A. L. et al. Evaluating the Effects of Branch Prediction Accuracy on the Performance of SMT Architectures, In: EUROMICRO WORKSHOP ON PARALLEL AND DISTRIBUTED PROCESSING, EUROMICRO PDP, 9., 2001, Mantova, Italy. Proceedings... [S.l.:s.n.], 2001.

[5] Moffat, A. Implementing the PPM data Compression Scheme. IEEE Transactions on Communications, November 1990, 38(11):1917-1921.

[6] Chen, I.C.K; Coffey, J.T.; Mudge, T.N. Analysis of Branch Prediction via Data Compression. In: Proceedings of ASPLOS, October 1995, pages 128-137.

[7] Lee, J. K. L.; Smith, A.J. Branch Prediction Strategies and Branch Target Buffer Design. Computer, 17(1), Jan, 1984.

[8] Bray, B. K.; Flynn, M. J. Strategies For Branch Target Buffers. In: Proceedings of Micro-24, Nov, 1991, p. 42-50.

[9] Perleberg, C. H.; Smith, A.. Branch Target Buffer Design and Optimization. IEEE Transaction on. Computers, April 1993, p. 396-412.

[10] Smith, E. A Study of Branch Prediction Strategies. In Proc. of 8th International Symposium in Computer Architecture, Minneapolis, MN, 1981, p. 135-148.

[11] Yeh, T.; Patt, Y. N. Two-level Adaptive Training Branch Prediction. In Proceedings of the 24th Annual International Symposium on Microarchitecture, Albuquerque, New Mexico, November 1991, p. 51-61.

[12] Yeh, T.; Patt, Y. N. Alternative Implementation of Two-Level Adaptative Branch prediction. The 19th Annual International Symposium on Computer Architecture, Gold Coast, Australia, May 1992, p. 124-134.

[13] Yeh, T.Y.; Patt, Y. A comparasion of Dynamic Branch Predictors that use Two Levels of Branch History. In Proceedings of the International Symposium on Computer Architecture, May1993, pages 257-266.

[14] Jouppi, N. P.; Wall, D. W. Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines, 3rd International Conference on Architecture Support for Programming Languages and Operating Systems, IEEE and ACM, April, 1989.

[15] Wall, D.W. Limits of Instruction-Level Parallelism. In Proceedings of ASPLOS IV, Santa Clara, 1991, p. 176-188.

[16] Lam, M.S.; Wilson, R. P. Limits of Control Flow on Parallelism. In Proceedings 20th International Symposium on Computer Architecture, May 1993, p.46-57.

[17] Gonzalez, J.; Gonzalez, A. Control–Flow Speculation through Value Prediction for Superscalar Processor. In: Proceedings of PACT, 1999, p. 57-65.

[18] Sherwood, T.; Calder, B. Loop Termination Prediction. Proceedings of the 3rd International Symposium on High Performance Computing (ISHPC2K), Oct. 2000.

[19] Fern, A.; et al. Dynamic Feature Selection for Hardware Prediction. Technical Report TR-ECE 00-12, School of Electrical and Computer Engineering, Purdue Univ, 2000.

[20] Aragon, J. L.; et al. Confidence Estimation for Branch Prediction Reversal. In Proceedings of the 8th International Conference on High Performance Computing, Dec. 2001.

[21] Ribas, V.; Figueiredo, M. e Gonçalves, R. *Analyzing Branch Prediction Under Speculative Execution Using Perceptron*, XXIII Congresso da SBC - ENIA - Encontro Nacional de Inteligência Artificial, Campinas, Agosto, 2003.

[22] McFarling, S. Combining Branch Predictors. Technical Report TN-36m, Digital Western Laboratory, Jun. 1993.

[23] Young, C.; Gloy, N.; Smith, M. D. A Comparative Analysis of Schemes for Correlated Branch Prediction. . In: Annual International Symposium on Computer Architecture, ISCA, 22., 1995, Santa Margherita Ligure, Italy. Proceedings... [S.l.:s.n.], 1995.

[24] Kessler, Richard E. The Alpha 21264 microprocessor. IEEE Micro, [S.l.], v.19, n.2, Mar./Apr. 1999.

[25] Federovsky, E.; Feder, M.; Weiss, S. Branch Prediction based on Universal Data Compression Algorithms. In Proceedings of the International Symposium on Archicteture, June 1998, pages 62-72.

[26] Kalamatianos, J.; Kaeli, D.R..Indirect Branch Prediction using Data Compression. Journal of Instruction Level Parallelism , Vol. 1, 1999.

[27] Burger, D.; Austin, T. M. The SimpleScalar Tool Set, Version 2.0. Technical Report #1342, Computer Sciences Department, University of Wisconsin-Madison, Jun. 1997.

[28] Henning, J.L. SPEC CPU2000: Measuring CPU Performance in the New Millennium. IEEE, 2000. p. 28-35.