Estudo de Algoritmos de Previsão de Desvio com o Simulador Gem5

Leonardo Vendrame[1]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
Maringá – PR – Brasil

leonvendrame@gmail.com

Abstract. *This paper presents a study about branch prediction. Branch prediction aims to improve the performance of applications execution, but, there's several algorithms to do it, that may have different performances. The objetivo is to look for the best-case branch prediction algorithm to the largest applications domain. Inside the study's scope, the LTAGE algorithm was the one who demonstrated the best results.*

Resumo. *Este paper apresenta um estudo sobre previsão de desvio. Previsão de desvio visa melhorar o desempenho de execução de aplicações, porém, existem diversos algoritmos para isso, que podem ter desempenhos diferentes. O objetivo é buscar o algoritmo de previsão de desvio que se sai melhor para o maior domínio de aplicações. Dentro do escopo do estudo o algoritmo LTAGE foi o que demonstrou os melhores resultados.*

1. Introdução

O desempenho da execução real de um programa pode ser medido de diversas formas, visto que é possível tomar como base vários aspectos. Tomando como foco o tempo de execução, é possível realizar a análise detalhada de processamento, procurando eliminar desperdícios de tempo.

Para que se obtenha um menor tempo de execução, é importante que o *pipeline* dos processadores esteja sempre completo, procurando impedir que existam ciclos de *clock* em que o processador fique ocioso. Porém, existem determinados problemas que podem impedir que o *pipeline* continue completo, um destes problemas é o de saltos.

Saltos são, resumidamente, instruções que permitem saltar para determinados trechos de código do programa. Existem saltos incondicionais e saltos condicionais. O primeiro não gera problema dentro de um *pipeline*, visto que deverá ser tomado, e portanto as próximas instruções devem ser aquelas imediatamente depois do salto. Porém, saltos condicionais podem gerar problemas. Um salto condicional, deverá ser tomado ou não somente após a execução de sua condição, o que acontece somente alguns estágios após a etapa de busca das próximas instruções. Com isso, pode ocorrer a busca de operações que não deveriam ser executadas, ocasionando numa perda de ciclos de *clock* úteis para processamento.

Visando diminuir ao máximo essa perda, foram desenvolvidos os algoritmos de previsão de desvio, ou, a téncica de execução especulativa [Pedronette 2006], isto é, algoritmos que tentam prever o comportamento do salto, especulando se este deve ou não ser tomado antes mesmo da execução de sua condição, procurando assim, buscar pelas próximas instruções que realmente deveriam ser executadas.

2. O Simulador Gem5

O Simulador Gem5 é uma união da infraestrutura de dois simuladores conhecidos como *M5* e *GEMS*. Com isso o Gem5 fornece um ambiente de simulação com uma ampla capacidade de configuração, oferencendo diversas arquiteturas de conjuntos de instruções, bem como diversos modelos de processadores, aliados à um sistema de memória com suporte a múltiplas caches [Binkert et al. 2011].

Simuladores como este dão base à pesquisas, testes e até mesmo projeções de produtos que podem vir a serem desenvolvidos. Fornecendo uma gama de recursos e permitindo a customização de uma arquitetura, o Gem5 permite um estudo sobre o comportamento dos algoritmos de previsão de desvio. Além da escolha do modelo de processador é possível também definir suas propriedades, entre elas, o aspecto aqui desejado, algoritmo de previsão de desvio. A partir disso o simulador se torna uma ferramenta capaz de prover informações específicas sobre tais algoritmos, fornecendo os dados resultantes de suas simulações, dando a possibilidade de que um algoritmo específico seja testado para diversos programas de entrada.

O simulador possui dois modos de execução, o *SE (System-call Emulation)* que captura e emula todas as chamadas ao sistema, e o *FS (Full-System)* que simula um sistema completo, provendo um ambiente de simulação baseado em um sistema operacional.

O Gem5 pode simular as seguintes arquiteturas: *ARM*, *ALPHA*, *MIPS*, *Power*, *SPARC*, e *x86*. Sendo que as arquiteturas *ARM*, *ALPHA*, e *x86* incluem suporte ao *boot* ao *Linux*.

3. Algoritmos de Previsão de Desvios

Algoritmos de previsão de desvio, são algoritmos que visam prever o comportamentos de instruções de saltos dentro de uma aplicação. O objetivo dessa especulação é ganhar desempenho, buscando amenizar que o processador faça trabalho que não será usado, isto é, processe informações que serão descartadas após seu processamento ao invés de serem efetivadas.

A execução especulativa de instruções permite aos processadores ganharem tempo eliminando instruções inúteis, porém, isso só acontece caso a especulação seja tomada corretamente. Nos casos de erro de especulação, o processador já terá buscado outras instruções, pois o mesmo só saberá que a especulação foi incorreta durante a execução da condição do salto. As instruções que foram buscadas especulativamente serão executadas antes que o processador possa buscar as instruções corretas, e neste caso os resultados das mesmas não serão efetivados, portanto, descartados. Neste tempo o processador gastou ciclos de clock com processamento desnecessário.

Algoritmos de previsão de desvio com alta precisão são essenciais em *pipelines* muito profundos e são um fator de extrema importância em processadores superescalares [Evers et al. 1996], visto que, deseja-se obter resultados no menor tempo possível e sempre minimizar o desperdício de ciclos de *clock*. Em média, a cada cinco instruções, uma é operação de desvio [Nicácio 2006], portanto, mostram-se necessários algoritmos com taxas de acerto cada vez maiores.

Existem algoritmos de previsão estática e algoritmos de previsão dinâmica. Algorimos de previsão estática tomam sempre um determinado comportamento, como por

exemplo tomar todos os saltos, ou não tomar nenhum salto. Já algoritmos dinâmicos possuem comportamentos variados e tentam analisar das mais diversas maneiras se um salto deve ou não ser tomado, isso acontece durante a execução da aplicação. Destacam-se alguns algoritmos de previsão dinâmica:

LTAGE: É um preditor de desvio baseado em histórico global. Apresenta um preditor bimodal indexado por *PC* (*Program Counter)* e N tabelas parcialmente demarcadas, indexadas com um *hash* de *PC* e do histórico global de predição. Todas as suas tabelas são acessadas paralelamente e aquela usando o maior histórico que combina é a que dá a predição (com algumas exceções). Além disso ainda conta com um preditor de *loops* que guarda o contador de interações de *loops* e os prevê de acordo.

BiMode: É um preditor de dois níveis que possui três vetores de histórico separados: um de saltos tomados, um de saltos não tomados e um de escolha. Visa eliminar o *aliasing* destrutivo que ocorre quando duas predições diferentes compartilham o mesmo padrão de histórico global, e faz isso por separar as predições em vetores separados de saltos tomados e não tomados e usando o desvio do *PC* (*Program Counter*) para escolher entre eles dois.

Tournament: Também conhecido como híbrido, o algoritmo Tournament geralmente combina o funcionamento dois algoritmos de previsão, tentando se adequar às diferentes aplicações que possam vir a serem executadas.

Local (2 Bits): Consiste de utilizar 2 bits para prever o desvio, um bit diz se deve ou não ser tomado e o outro diz se o preditor acertou ou não a ultima previsão. Geralmente é implementado através de um somador saturado.

4. Experimentos e Discussão

Serão apresentados aqui informações sobre os experimentos e resultados obtidos, bem como questões observadas na realização do projeto.

4.1. Ambiente de Testes

Hardware: Para a simulação foi utillizado um dispositivo com as seguintes características:
Processador: Intel®Xeon®CPU @ 2.20GHz.
Memória RAM: 8GB.
Sistema Operacional: Debian 9.4 (Stretch) 64 Bits.
Disco Rígido: SSD 20GB.

Software: A execução se deu através do Gem5 com uma arquitetura simulada com as seguintes características:
Gem5: Modo *SE (System-call Emulation)*. Versão *gem5.fast 2.0*.
Processador: O3CPU @ 2GHz.
Conjunto de Instruções: x86.
Cache: Multi-nível de 2 níveis.
Cache L1: Dividida entre dados e instruções. L1D: 16KB e L1I: 64KB.
Cache L2: Unificada. L2: 256KB.
Memória RAM: 3GB
Compilador: *GCC 6.3.0*.

Algoritmos e Aplicações: Algoritmos e aplicações simulados:

Algoritmos de previsão: *LTAGE*, *BiMode*, *Local (2 Bits)* e *Tournament*. Aplicações: *nbench (LLVM test-suite)*, *dijkstra (Polybench)*, *floyd-warshall (CBench)*, *flops-8 (LLVM test-suite)* e *mpeg2enc (MediaBench II)*.

nbench: Aplicação de Benchmark.
dijkstra: Determina o caminho mínimo, partindo de um vértice de início v para todos os outros vértices do grafo.
floyd-warshall: Determina a distância entre todos os pares de vértices de um grafo.
flops-8: Procura estimar a taxa de MFlops do sistema para determinadas operações baseadas em um mix de instruções específicas.
mpeg2enc: Codifica diversos arquivos *.ppm (Netpbm color image format)* em um arquivo *.mpg*.

4.2. Metodologia e Experimentos

Todos os algoritmos foram compilados utilizando a flag de otimização *-O3*.

Após compiladas, cada aplicação é executada uma vez para cada algoritmo de previsão de desvio. Desta forma, é possível obter o desempenho de cada algoritmo em situações e domínios variados.

Uma vez executadas as aplicações, o ambiente Gem5 gera um arquivo contendo todas as informações daquela execução, permitindo obter detalhes dos mais diversos aspectos. Aqui serão analisados os aspectos referentes a previsão de desvio, com foco na quantidade de erros e acertos, a fim de expor situações em que os algotimos podem ser mais eficientes.

Analisando as estatísticas presentes no final das simulações é possível obter o número de predições totais e incorretas de cada algoritmo para cada aplicação, resultando nas taxas de erros e acertos que serão avaliadas e apresentadas, com o objetivo de mostrar dados palpáveis da eficácia de tais algoritmos.

4.3. Questões

Destacam-se algumas questões a serem respondidas com a execução dos experimentos:
Qual é a taxa de acerto de cada algoritmo? Qual algoritmo se sai melhor no geral?
Qual é a taxa de erro de cada algoritmo? Qual algoritmo se sai pior no geral?
O domínio de aplicação e a aplicação em si influenciam nos resultados? De que maneira?

4.4. Resultados

Visando responder as questões apresentadas acima, temos os resultados a seguir.

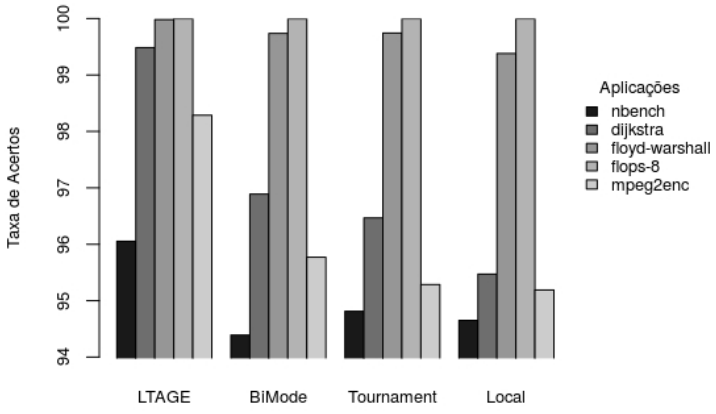

Figura 1. Resultado: Taxa de Acertos por Algoritmo

Através da Figura 1 pode-se observar a taxa de acerto de cada algoritmo de previsão de desvio. A média da taxa de acerto geral do escopo do projeto foi de 97.57%. Isso pode ser considerado uma boa taxa de acertos, porém, fora do escopo do projeto os processadores precisam lidar com um número expressivamente maior de domínios de aplicações, o que faz com que na prática, nem sempre os algoritmos de previsão de desvio alcancem tais taxas.

Fica exposto que em todos os casos, salvo à exceção de empate de performance na execução da aplicação *flops-8*, o algoritmo *LTAGE* foi o que obteve a maior taxa de acerto. Além disso, pode-se também notar um padrão quando se olha para todos os algoritmos, de maneira geral, as barras se comportam de maneiras semelhantes. De maneira complementar à Figura 1 é possível se obter também as taxas de erros dos algoritmos em questão, que variam entre 0,004% e 5,608%, com média de 2,42%. Para terminar de responder às primeiras questões, o algoritmo que se saiu pior no geral foi o *Local (2 Bits)*, que conseguiu as menores taxas de acertos em 80% dos casos.

Procurando dar resposta à última questão, vamos organizar os dados de uma forma diferente, verificando os resultados desta vez por aplicação.

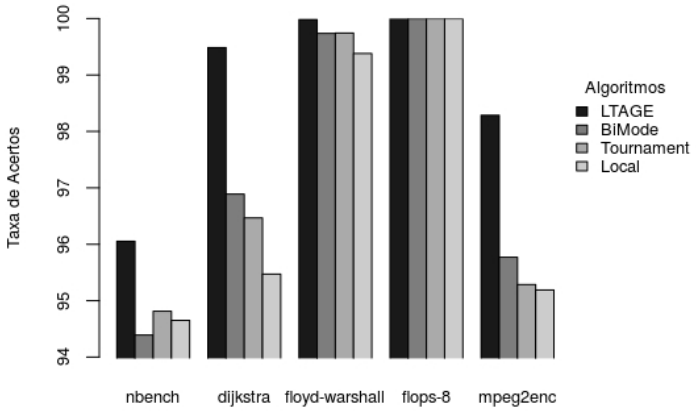

Figura 2. Resultado: Taxa de Acertos por Aplicação

Com a nova organização da Figura 2, evidenciamos que as aplicações e seus domínios influenciam no resultado dos algoritmos de previsão de desvio, isto é, existem certas aplicações que vão obter melhores resultados que outras. Um exemplo disso é a aplicação *flops-8*, que garantiu uma taxa de acertos maior que quaisquer outras.

Os preditores se mostraram no geral eficazes e precisos nos testes realizados, com taxas de acertos acima de 94%. Entretando ainda houve certa diferença. Analisando a aplicação *flops-8*, ela basicamente não lida com instruções condicionais do tipo *if*, mas sim com *loops* do tipo *for*. Isto pode explicar as predições próximas de 100%, visto que até mesmo em um algoritmo de previsão local com um histórico de 2 bits seriam poucas as vezes em que o algoritmo erraria a predição, basicamente na entrada e na saída do *loop*. Diferentemente da aplicação *nbench* que demonstra a menor média de acerto, com cerca de 95% e possui diversos casos condicionais tanto de instruções do tipo *if*, quanto de instruções do tipo *case* dentro de *switches*.

Portanto, quanto maior for o índice de condicionais maiores são as chances da especulação ser errada, o que expõe que os algoritmos de previsão de desvio se saem melhor com casos de *loops* e aplicações simplesmente iterativas ao invés de aplicações em que a entrada é imprevista e pode dividir o fluxo de instruções em muitas partes, com muitas instruções condicionais.

5. Conclusão

Algoritmos de previsão de desvio possuem grande importância no desempenho da execução de programas dentro de um processador, eles ditam o desempenho e o número de ciclos úteis quando instruções são executadas especulativamente. Portanto, quanto melhor o algoritmo, isto é, quanto menor for a taxa de erro do mesmo, menos tempo é desperdiçado com instruções que não deveriam ser executadas.

Apesar da grande eficácia atual de tais algorítmos eles ainda não são perfeitos e falham em conseguir prever todas as especulações. Desta forma, ainda existem instruções que são processadas sem que seus resultados sejam efetivamente úteis para as aplicações.

Contudo, a taxa de acertos, como mostrada durante os testes realizados, pode se mostrar muito próxima de 100% em algoritmos com poucas instruções condicionais. Além disso existem algoritmos que se saem melhor para determinadas aplicações.

Com os testes demonstra-se a importância de um estudo mais aprofundado e detalhado antes da implementação de algoritmos de previsão de desvio em *hardware*, visto que depois de implementados os mesmos não podem ser alterados tão facilmente quanto via *software* e portanto, deve-se procurar encontrar o que ofereça o melhor desempenho para os mais amplos de domínios de aplicações.

Referências

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., et al. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7.

Evers, M., Chang, P.-Y., and Patt, Y. N. (1996). Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *ACM SIGARCH Computer Architecture News*, volume 24, pages 1–11. ACM.

Nicácio, D. (2006). Previsão de desvios.

Pedronette, D. C. G. (2006). Execução especulativa - conceitos gerais e técnicas.