Projeto 3 - Reprodução de um resultado de um artigo escolhido

Alceu Emanuel Bissoto[1]

[1]Instituto de Computação - Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brasil

Abstract. *This work reproduces a result obtained in the article* Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement *[A. Jain], which introduce a new cache replacement policy. Differnet tools have been used, such as the x86 architecture simulador Sniper [Eeckhout], and some adaptations were necessary. The tests have been made using the benchamark SPEC2006, in order to compare the cache misses obtained using LRU and the one proposed by the original paper. The results showed that in fact the changes proposed bring benefits to the cache, resulting in improvements for different programs and workflows. The differences noticed between both the evaluations can be explained by the utilization of different tools, and subsequent adaptations that were necessary to be done in the implementation of the solution.*

Resumo. *Neste trabalho buscou-se reproduzir um resultado obtido no artigo* Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement *[A. Jain], que introduz uma nova política de substituição para a cache. Diferentes ferramentas foram utilizadas, como o simulador x86 Sniper [Eeckhout], e algumas adaptações foram necessárias. Os testes foram feitos sobre o benchmark* SPEC2006*, de modo a comparar a quantidade de* misses *na cache obtidas pela solução proposta com a técnica LRU, que já é bem estabelecida. O resultado mostrou que de fato a implementação do algoritmo traz benefícios à cache, chegando até a melhoras significativas para diferentes programas, e cargas. As diferenças obtidas entre esta reprodução e o trabalho original podem ser explicadas pelo fato da utilização de diferentes ferramentas, e por consequentes adaptações que foram necessárias na implementação.*

1. Introdução

Novas políticas de substituição para a cache são amplamente estudadas pelo meio científico, uma vez que estas estão diretamente ligadas a performance da cache, e portanto, do computador de modo geral. Vários estudos oferecem diferentes modos de chegar à resposta para a mesma pergunta. "Qual linha de de cache deve ser descartada?". As técnicas mais tradicionais, como LRU (*Least Recently Used*) e MRU (*Most Recently Used*) funcionam bem para diferentes algoritmos, e ainda são amplamente utilizadas, e comparadas com as novas técnicas que são criadas.

Através do artigo *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement* [A. Jain], busca-se introduzir um novo modo de decisão para a pergunta feita acima, utilizando de um algoritmo criado nos anos 60 por Belady [Belady], que decidia a linha a ser removida da cache olhando para o futuro, eliminando a linha que seria

utilizada no futuro mais distante possível. Partindo da premissa de que não temos conhecimento sobre o futuro, e de que o passado tende a prever o futuro, foi criado o *Hawkeye*, que armazena os últimos oito acessos e verifica se, caso a política OPT estivesse sendo usada, haveria um hit ou um miss. Quando não é possível decidir qual dado eliminar da cache apenas com essa informação, é utilizado LRU.
Para este trabalho buscou-se reproduzir a solução descrita pelo artigo, comparando os resultados obtidos através da montagem de um gráfico que mostra a redução de MPKI (*Miss Per Kilo Instruction*) utilizando o *Hawkeye*, em comparação com a política LRU.

2. Metodologia

Os autores do artigo utilizaram um *framework* desenvolvida para o *First JILP Cache Replacement Championship* [A. R. Alameldeen and Emer], mas por se tratar de uma versão muito antiga, incompatível com o *kernel* dos sistemas operacionais mais modernos, buscou-se uma alternativa. Sendo assim, para a reprodução da solução proposta, foi utilizado o Sniper [Eeckhout] que é um simulador de arquitetura x86, e que oferece a possibilidade da implementação de políticas de substituição da cache, além de facilitar a interação com *pinballs*, que foram utilizados de modo a agilizar os testes com o *SPEC2006*, que foi o *benchmark* utilizado também no artigo original. Foi utilizado neste trabalho apenas o conjunto com dados inteiros do *SPEC2006*.
O tamanho do último nível da cache, que receberá a política de substituição em questão, também foi replicado do artigo original, mantendo 2MB para esta, além de associatividade de 16 vias.

3. Implementação

O *Hawkeye* se utiliza de um vetor de ocupação para poder manter controle de quantos dados na cache simultâneos pode-se ter na cache a cada instante de tempo. Este vetor é utilizado de modo a saber se um dado acesso a uma linha da cache sobre a política de substituição OPT, haveria *hit* ou *miss*. Quando a cache não é mapeada diretamente, ou seja, quando o nível de associatividade é maior que 1, o *Hawkeye* mantém um vetor de ocupação para cada linha de cache, onde seu tamanho é de 8 vezes o nível de associatividade da cache. Como estavam sendo utilizadas 16 vias de cache, então o vetor tinha tamanho fixo de 128. Essa técnica recebe o nome de OPTgen pelos autores do artigo original.
O Sniper possui dois métodos que podem ser sobrescritos para a implementação de novas políticas de substituição de cache. São eles *updateReplacementIndex()*, e *getReplacementIndex*. De modo geral, o *getReplacementIndex* tem a função de retornar em qual via para uma determinada linha de cache está o dado que será removido da cache caso necessário. Esta função é chamada pelo sniper sempre que há a possibilidade de remoção de um dado da cache. A *updateReplacementIndex()* é chamada em toda leitura e escrita de dados na cache, e é utilizada de forma a manter o controle das estruturas responsáveis por determinar qual o dado a ser retirado.
Foram criados três vetores de modo a garantir o funcionamento do algoritmo como descrito no documento original. São três vetores: *m_occupancy, m_last_occurrence e m_priority_eviction*.
O vetor *m_occupancy* é um vetor de ocupação que implementa o OPTgen, informando a

cada novo acesso a cache, se este dado seria um *hit* ou *miss* sobre a lógica OPT. Para manter controle sobre a última aparição de um dado de uma via no *m_occupancy*, é utilizado o *m_last_occurrence*. Este tem o mesmo tamanho do nível de associatividade da cache (16), cada entrada simboliza uma via da linha de cache em questão. Finalmente, temos o *m_priority_eviction*, que também tem o mesmo tamanho do nível de associatividade da cache, e é utilizado para indicar qual dos níveis deve ser retornado por *getReplacementIndex()* para uma possível retirada de dados da cache. Os campos com maiores valores indicam que são mais prioritários para uma possível substituição.

4. Resultados

Os resultados obtidos neste trabalho, ao replicar o trabalho proposto *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement* encontra-se no gráfico da figura 1.

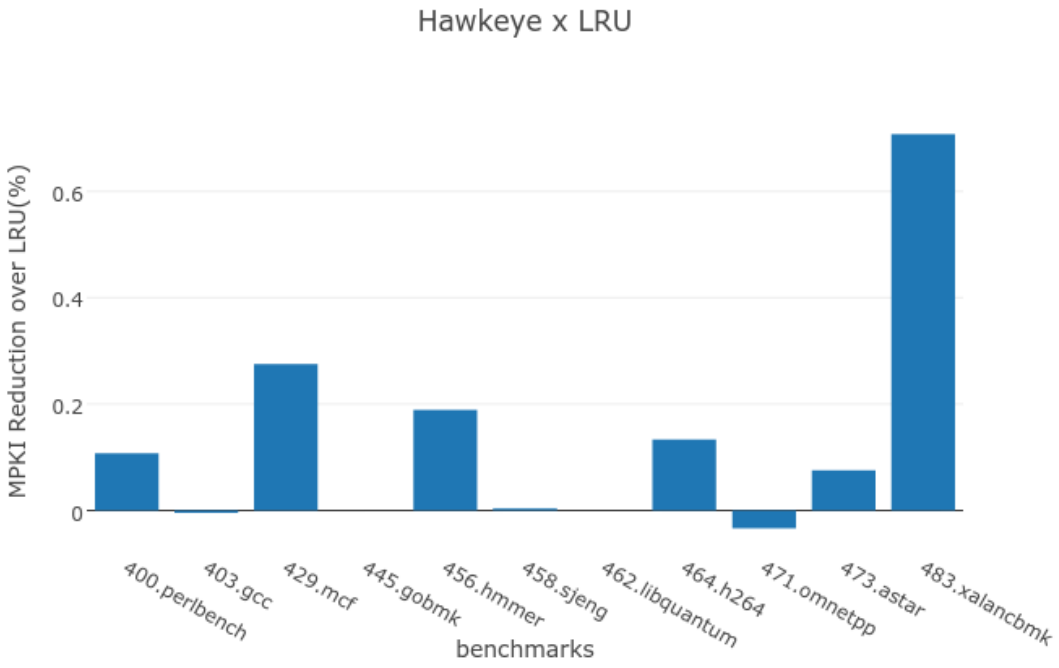

Figura 1. Resultado obtido através da simulação com o Sniper para diferentes programas do SPEC2006.

Quando comparamos com os resultados apresentados no artigo original e na figura 2, vemos que há algumas diferenças consideráveis.

Além de terem sido usadas diferentes ferramentas para a simulação da solução, como o sniper, e a utilização de pinballs do SPEC2006, ainda podemos citar algumas incertezas quanto a implementação que surgiram durante o trabalho. Alguns passos do algoritmo não são explorados em detalhes pelo artigo original, fazendo com que algumas adaptações fossem necessárias.
Porém, é verificável que a solução proposta se saiu bem em diversos programas do SPEC2006. Por utilizar LRU como técnica de substituição quando o algoritmo OPTgen

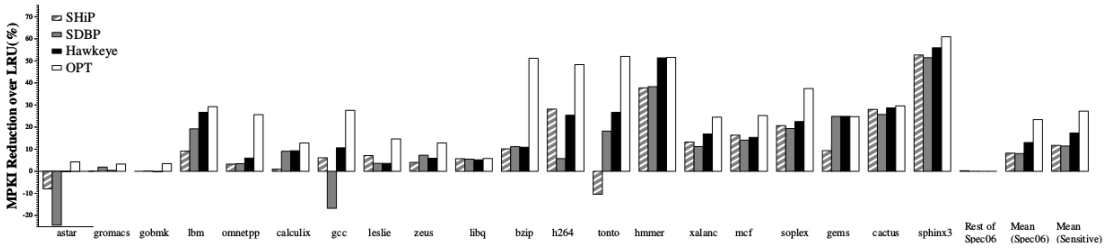

Figure 8: Miss rate reduction for all SPEC CPU 2006 benchmarks.[4]

Figura 2. Resultado apresentado no artigo *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement*.

não consegue decidir por si só qual dado deverá ser retirado da cache, o Hawkeye apresenta comportamento bastante similar ao LRU em alguns casos. Para programas onde o OPTgen é bastante explorado, ou seja, quando este consegue determinar múltiplas ocorrências de *misses*, vemos que a performance aumenta consideravelmente, como é o caso no programa *429.mcf* que apresentou melhora acima dos 25% e para o *483.xalancbmk* que mostrou uma melhora de aproximadamente 70%.

5. Conclusão

Durante a reprodução do artigo escolhido, diferentes ferramentas se mostraram necessárias, entre elas o simulador x86 Sniper [Eeckhout], e a utilização de *pinballs* para o benchmark SPEC2006. O ambiente de testes e a forma de implementação foram discutidos neste documento, de forma a mostrar possíveis alternativas. Testes foram realizados de forma a medir a quantidade de *misses* gerados comparando à política de substituição LRU. O resultado mostrou que com a implementação do algoritmo, houve ganhos significativos para diferentes programas e cargas. Quando a solução apresentada era explorada pelo programa, o ganho chegou a aproximadamente 70% no programa *483.xalancbmk*, e se manteve pelo menos igual ou pouco abaixo ao LRU em todos os casos estudados.

Referências

A. Jain, C. L. Back to the future: Leveraging belady's algorithm for improved cache replacement. Proceedings of The 43rd International Symposium on Computer Architecture, 2016.

A. R. Alameldeen, A. Jaleel, M. Q. and Emer, J. 1st jilp workshop on computer architecture competitions (jwac-1) cache replacement championship. 2010.

Belady, L. A. A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal, pages 78–101, 1966.

Eeckhout, T. E. C. W. H. L. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. International Conference for High Performance Computing, Networking, Storage and Analysis (SC).