Projeto 4 - Aprimoramento da reprodução feita do artigo escolhido

Alceu Emanuel Bissoto[1]

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

Abstract. *This work improves the reproduction of the solution described in the article* Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement *[A. Jain], that was done on Project 3. The main focus was to add the Hawkeye predictor, which uses the instruction PC that accesses the cache, and the OPTgen algorithm in order to decide the priority of eviction of each of the ways of a cache line. The x86 architecture simulador Sniper [Eeckhout] was used, and some changes were necessary to be able to accesss the PC information. The tests have been made using the benchamark SPEC2006, in order to compare the cache misses obtained between the proposed implementation and the one suggested on Project 3. The results showed that despite the changes proposed actually brings benefits to the cache, the results were worse when comparing the ones obtained on Project 3, where only the OPTgen algorithm was used to take the eviction decisions.*

Resumo. *Neste trabalho buscou-se aprimorar a reprodução da solução descrita pelo artigo* Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement *[A. Jain] que foi feita no Projeto 3. O foco principal foi adicionar o preditor Hawkeye, de modo a utilizar o PC das instruções que acessam a cache, e o OPTgen para decidir a prioridade das diferentes vias de cada linha de cache de serem substituídas. O simulador x86 Sniper [Eeckhout] continuou sendo usado, com algumas mudanças, de modo a ter acesso ao PC. Os testes foram feitos sobre o mesmo conjunto de programas do benchmark* SPEC2006*, de modo a comparar a quantidade de* misses *na cache obtidas pela solução proposta, com a já apresentada no Projeto 3. O resultado mostrou que de apesar da implementação do algoritmo trazer benefícios à cache, o uso do PC piorou a performance da cache em comparação com a implementação utilizada no Projeto 3, onde apenas o OPTgen foi utilizado para tomar as decisões da política de substituição da cache.*

1. Introdução

A cache é um componente essencial dos computadores modernos. Ela é responsável por tornar mais rápidos os acessos a dados que são reusados. Tal acesso é mais rápido pela proximidade deste componente ao processador, e também ao pequeno tamanho destas estruturas. Quando se tem apenas poucos *kilobytes* para armazenar os dados, como no caso de caches L1, a pergunta "Qual linha de de cache deve ser descartada?"precisa ser respondida da melhor forma possível para que os dados que estejam na cache possam ser aproveitados ao máximo e potencializar os benéficios que este componente nos trás. As técnicas mais tradicionais, como LRU (*Least Recently Used*) e MRU (*Most Recently*

Used), que retiram da cache as linhas menos recentemente utilizadas, e as mais recentemente utilizadas, repectivamente, 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 decide 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 acessos e verifica se caso a política OPT estivesse sendo usada, haveria um *hit* ou um *miss*. Tal módulo é chamado pelos autores de OPTgen. A partir desta informação, é feito um treinamento do PC (*Program Counter*) que indica os PCs mais propensos a carregarem dados que serão utilizados novamente, e portanto devem permanecer na cache. Quando não é possível decidir qual dado eliminar da cache apenas com essa informação, é utilizado LRU.
Para este trabalho buscou-se aprimorar a reprodução da solução descrita pelo artigo, feita no Trabalho 3, utilizando o mesmo gráfico deste último trabalho citado para mostrar os resultados obtidos. Tal gráfico mostra a redução de MPKI (*Miss Per Kilo Instruction*) utilizando o *Hawkeye*, em comparação com a política LRU.

2. Metodologia

Neste trabalho procurou-se implementar o preditor *Hawkeye*, descrito no artigo *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement*, que deve ser utilizado em conjunto com o OPTgen. As figuras 1 e 2 ilustram o relacionamento entre o Hawkeye e o OPTgen.

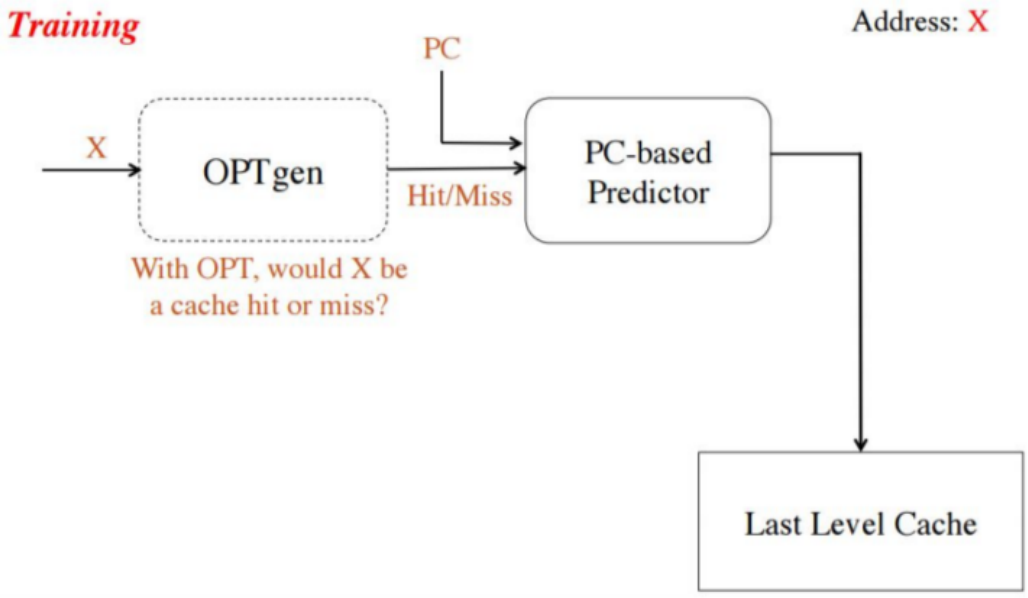

Figura 1. Fase de treinamento.

Para tal, foi continuado o código criado durante o Projeto 3, utilizando novamente o Sniper [Eeckhout] como um simulador de arquitetura x86, aproveitando sua

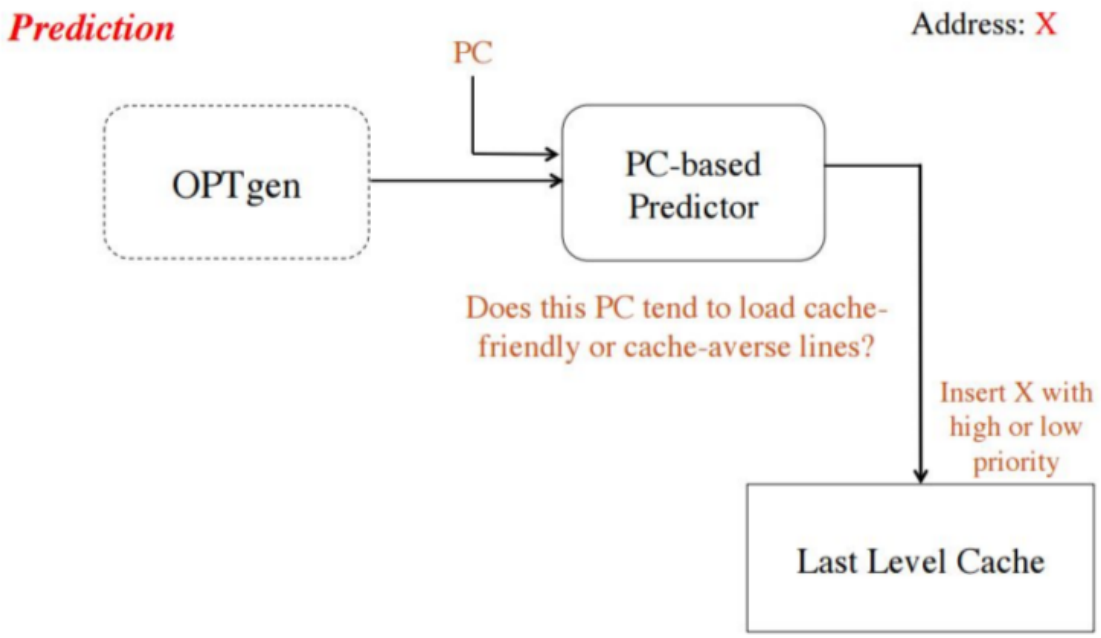

Figura 2. Fase de predição.

flexibilidade que permite a implementação de novas políticas de substituição da cache, e também a interação com *pinballs*, que foram utilizados de modo a agilizar os testes com o benchmark *SPEC2006*, que foi também utilizado no artigo original para medir a performance da nova política de substituição criada. Assim como no projeto 3, 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 foi replicado do artigo original, mantendo 2MB para esta, além de associatividade de 16 vias.

3. Implementação

Para melhor representar a solução descrita pelo artigo *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement* foi implementado o preditor *Hawkeye*, que prevê de acordo com o PC da instrução de load/store que está acessando a cache; e de acordo com o algoritmo OPTgen, que por sua vez busca responder se uma dada linha de cache resultaria num miss ou hit caso o algoritmo OPT estivesse sendo utilizado; a prioridade de cada um das diferentes vias de uma linha de cache de ser retirada do último nível de cache. Para tal, era necessário ter acesso ao componente essencial do preditor, o PC. O código de alguns componentes do Sniper precisaram ser modificados de modo a passar o PC como parâmetro até que este fosse acessível dentro do arquivo que implementa a política de substituição em questão. Em posse do PC das instruções que estão modificando a cache, foi possível implementar alguns pontos descritos no artigo original. Em resumo, as seguintes modificações foram implementadas:

- Adição de uma lista, onde cada entrada armazena um PC diferente que acessou esta linha de cache. No máximo, são armazenados 50 PCs diferentes por linha de cache.

- Adição de uma lista que indica o score de cada PC. Quanto maior esse score, mais um PC é mais propenso a fazer o *load/store* de linhas de cache que devem ser mantidas na cache.
- Adição de uma lista que armazena o último PC responsável pelo acesso às diferentes vias de cache.
- Se o OPTgen indicar que haveria um hit sob a política do OPT, então o PC que gerou este acesso à cache é treinado positivamente, incrementando seu score em uma unidade, até um score máximo de 7. Caso contrário, e o OPTgen indique que haveria um miss, então o PC é treinado negativamente, decrescendo uma unidade de seu score, até um score mínimo de 0.
- Caso o dado que foi retirado da cache tenha baixa prioridade de ser retirado, então o último PC que acessou esta via de cache é treinado negativamente, decrescendo uma unidade de seu score.

4. Resultados

Os resultados obtidos neste trabalho, ao comparar os resultados do projeto 3 e 4, encontram-se no gráfico da figura 3.

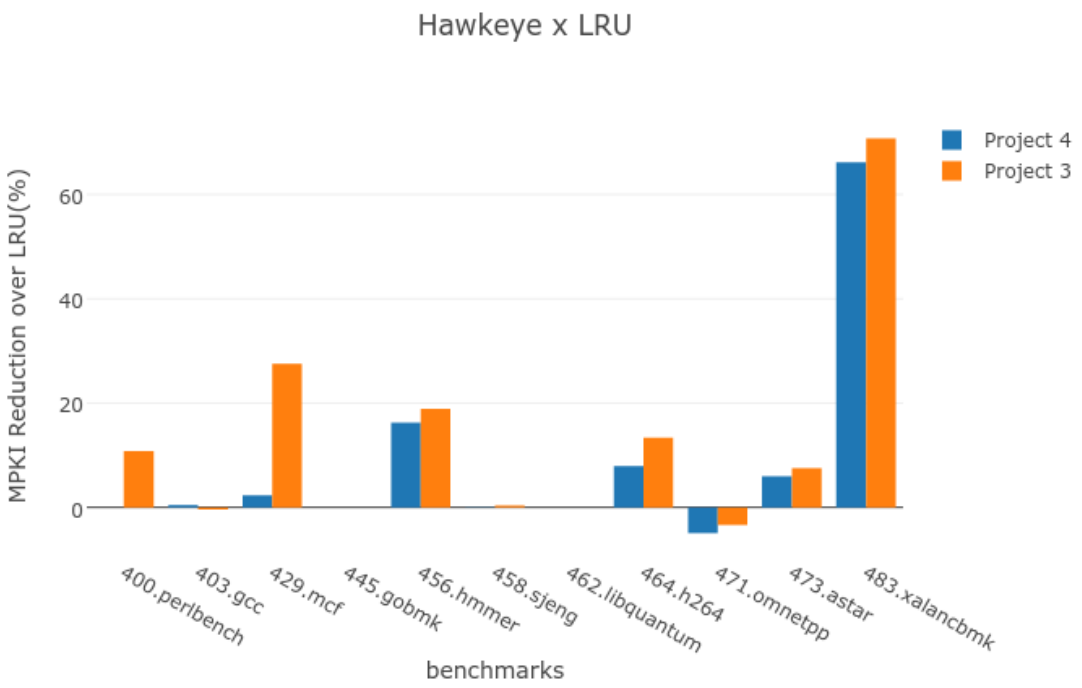

Figura 3. Comparação do resultado obtido através da simulação com o Sniper para diferentes programas do SPEC2006 entre o projeto 3 e 4.

Quando comparamos com os resultados apresentados no artigo original e na figura 4, vemos que há uma piora.

Nota-se que apesar de manter uma melhora em relação ao uso de LRU, quando levados em consideração os resultados obtidos no trabalho 3, vemos que houve um resultado pior.
Porém, é verificável que a solução proposta se saiu bem em diversos programas do

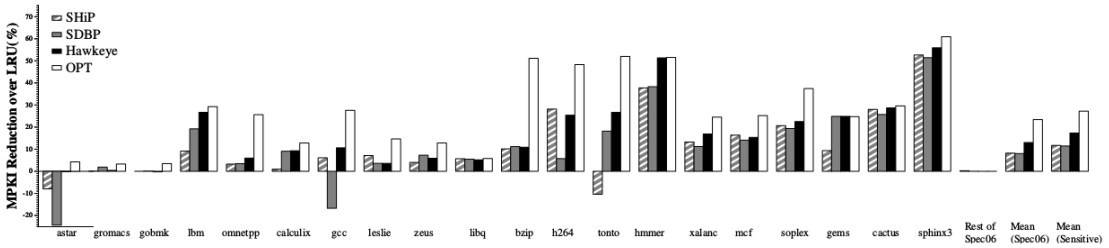

Figure 8: Miss rate reduction for all SPEC CPU 2006 benchmarks.[4]

Figura 4. Resultado apresentado no artigo *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement*.

SPEC2006. Por utilizar LRU como técnica de substituição quando o algoritmo OPTgen não consegue decidir por si só qual dado deverá ser retirado da cache, o Hawkeye apresenta comportamento bastante similar ao LRU em alguns casos. A mesma observação feita no projeto 3 continua valendo, já que o padrão da presença ou não de melhoras se manteve. Deste modo, para programas onde o OPTgen é bastante explorado, ou seja, quando este consegue determinar múltiplas ocorrências de *misses*, vemos que a performance aumenta.

5. Conclusão

Foi feita a implementação do preditor Hawkeye, que utiliza o PC da instrução que provocou o acesso a cache; e a informação dada pelo OPTgen, que indica se haveria miss ou hit utilizando a política OPT; de modo a prever a prioridade de uma via determinada via de uma linha de cache de ser substituída. Apesar de buscar se aproximar da implementação proposta no artigo *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement*, os resultados obtidos mostraram uma piora em relação aos obtidos no Projeto 3, onde utilizou-se puramente o OPTgen para fazer tais previsões, sem considerar o PC das instruções que causaram os acessos à cache. Apesar das mudanças, o maior problema detectado ainda continua, sendo este o fato de que o algoritmo proposto não consegue prever misses, e portanto acaba funcionando da mesma forma que o LRU.

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.

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).