Relatório - Trabalho Final

Amanda Edom Bandeira[1], **Maurício Kritli**[1], **Nicolau Pereira Alff** [1]

[1]Instituto de Informática – UFRGS

{aebandeira,mkritli,npalff}@inf.ufrgs.br

1. Introdução

O objetivo deste trabalho é apresentar a implementação e resultados de modificações de estruturas arquiteturais de um processador superescalar out-of-order para melhorar o desempenho (IPC) desse processador.

Será utilizado para essa implementação e medidas o simulador ChampSim [ChampSim]. As estruturas a serem alteradas são: Preditor de desvios, Prefetcher e Substituição de cache.

Na sessão 2 apresentamos os mecanismos implementados e suas principais modificações.

Para a Experimentação teremos 3 versões de referência que serviram de baseline para nossas modificações. E um conjunto de 5 aplicações de características variadas, a serem apresentadas na seção 3. Para a medida dos experimentos foram realizadas 50 milhões de instruções como warmup e 200 milhões para simulação.

2. Mecanismos Implementados

2.1. Versões de Referência

Na especificação do trabalho recebemos três configurações de arquiteturas de processadores para uso como baselines, sendo elas:

2.1.1. Configuração Baseline 1

- Branch Predictor: bimodal
- L1D Prefetcher: no
- L1I Prefetcher: no
- L2C Prefetcher: no
- LLC Prefetcher: no
- LLC Replacement: lru
- Cores: 1

2.1.2. Configuração Baseline 2

- Branch Predictor: bimodal
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: ip_stride
- LLC Prefetcher: no
- LLC Replacement: lru
- Cores: 1

2.1.3. Configuração Baseline 3

- Branch Predictor: hashed_perceptron
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: kpcp
- LLC Prefetcher: next_line
- LLC Replacement: drrip
- Cores: 1

2.2. Principais Modificações

Realizamos 2 grupos de testes. O primeiro deles mudando a política de substituição do último nível de cache (LLC), onde o algoritmo escolhido foi o FIFO. O segundo modificando o tamanho do Global History Buffer (GHB) do preditor de desvios, sendo escolhido o algoritmo gshared como preditor.

2.2.1. Cache replacement - FIFO

Como modificação realizada, optamos por modificar o padrão da política de LRU (Least Recently Used) para FIFO (First-In-First-Out). A substituição com LRU remove o item usado menos recentemente quando a cache está cheia e uma nova página é referenciada que não existe na cache, necessitando de espaço na mesma. Já a substituição com FIFO remove o item mais antigo presente na cache, independente de este item ser o menos usado ou o menos usado recentemente. Nós realizamos essa modificação no nível LLC de cache do ChampSim.

O algoritmo substituição de cache FIFO, foi escolhido baseado no conhecimento adquirido sobre nossas pesquisas sobre a anomalia de Belady [Belady et al. 1969], onde, através da literatura, descobrimos que a politica FIFO quando utilizada para gerenciamento de memória possuí o comportamento anômalo de aumentar a quantidade de Page Faults ocorridas, conforme o tamanho do buffer utilizado aumenta. Já para o caso do algoritmo LRU, o aumento do buffer utilizado reflete em diminuição da quantidade de Page Faults que ocorrem, por ser uma implementação baseada em pilha.

Ficando então com 3 novas configurações para comparação com os baselines:

Configuração FIFO 1

- Branch Predictor: bimodal
- L1D Prefetcher: no
- L1I Prefetcher: no
- L2C Prefetcher: no
- LLC Prefetcher: no
- LLC Replacement: fifo
- Cores: 1

Configuração FIFO 2

- Branch Predictor: bimodal
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: ip_stride
- LLC Prefetcher: no
- LLC Replacement: fifo
- Cores: 1

Configuração FIFO 3

- Branch Predictor: hashed_perceptron
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: kpcp
- LLC Prefetcher: next_line
- LLC Replacement: fifo
- Cores: 1

2.2.2. Diferentes Branch Predictors

Utilizamos a Configuração 2 como base para os diferentes resultados de Branch Predictor. Além do Baseline bimodal, utilizamos o gshare. Como um dos integrantes do grupo já havia trabalhado com o Global History em uma outra plataforma (BOOM) [Alff 2019], resolvemos fazer um experimento parecido para verificar se manteria o mesmo comportamento. Portanto, resolvemos rodar a mesma configuração com diferentes tamanhos de Global History. Rodamos então o gshare com 3 tamanhos diferentes, com o tamanho de buffer padrão do ChampSim: 14, e com outros dois tamanhos: 8 e 20. Estes tamanhos foram escolhidos sem um critério específico, apenas buscamos pegar um valor menor e outro maior suficientes para que consigamos avaliar o desempenho do gshare dados diferentes tamanho de Global History Buffer. As configurações ficaram da seguinte forma:

Configuração Branch 1

- Branch Predictor: bimodal
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: ip_stride
- LLC Prefetcher: no
- LLC Replacement: fifo
- Cores: 1

Configuração Branch 2

- Branch Predictor: gshare (tamanho 8)
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: ip_stride
- LLC Prefetcher: no
- LLC Replacement: fifo
- Cores: 1

Configuração Branch 3

- Branch Predictor: gshare (tamanho 14)
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: ip_stride
- LLC Prefetcher: no
- LLC Replacement: fifo
- Cores: 1

Configuração Branch 4

- Branch Predictor: gshare (tamanho 20)
- L1D Prefetcher: next_line
- L1I Prefetcher: next_line
- L2C Prefetcher: ip_stride
- LLC Prefetcher: no
- LLC Replacement: fifo
- Cores: 1

2.3. Hipóteses a serem testadas

Como estamos modificando a política de substituição da cache de LRU para FIFO, esperamos ter uma piora no desempenho das aplicações. Essa nossa hipótese é baseada no fato de que, a política LRU leva em conta o quão recentemente o item foi usado para removê-lo ou não da cache, e a nossa mudança para FIFO ignora esta informação de uso, removendo da cache, sempre que um novo espaço se faz necessário, o item mais antigo nela. Pelo fato de perdermos o benefício de exploração da localidade temporal dos dados, existente na LRU, acreditamos que a política FIFO irá fazer com que a quantidade de cache MISS aumente, e consequentemente, o custo em ciclos do processador para a execução completa do programa aumente.

Acreditamos também que, as configurações 1 e 2 devem ser as mais afetadas por essa mudança, visto que utilizam a política de substituição de cache LRU. Já a configuração 3 não deveria sofrer alterações, pois utiliza a política DRRIP (Dynamic Re-reference Interval Prediction).

Além, analisando as aplicações escolhidas (seção 3), podemos induzir que as aplicações que fazem mais uso da memória, como a 401.bzip2, 450.soplex e 453.povray, também serão as que terão uma maior discrepância nos resultados de acordo com a mudança realizada.

Com a utilização de diferentes Branch Predictors as hipóteses a serem testadas são que quanto maior for o global history buffer, melhor deve ser a predição de desvios, por conta de conseguir melhora na correlaçôes entre os branches e portanto, uma melhora na precisão do branch predictor e, por consequência, do IPC.

2.4. Custos Estimado em Hardware

Conforme mencionado no item 2.3, estimamos que a nossa modificação irá fazer com que o custo em processamento seja maior para a política FIFO de substituição de cache, logo muitos mais ciclos e acessos a memória serão necessários para o funcionamento correto da execução do programa.

Em relação ao branch predictor, quanto maior o tamanho do buffer utilizado, melhor será a precisão do branch predictor, logo, será necessário menos ciclos de processamento (maior IPC) para a execução do programa.

3. Aplicações Escolhidas

3.1. 401.bzip2

Este benchmark é baseado na versão 1.0.3 do benchamrk bzip2 criado por Julian Seward. É um benchmark de compressão de dados, desenvolvido na linguagem ANSI C. Resumidamente, seu funcionamento ocorre da seguinte maneira: Cada entrada é comprimida e descomprimida em três diferentes níveis de compressão, com o resultado de cada nível sendo comparado ao dado original. Todas as compressões e descompressões acontecem inteiramente na memória, com o intuito de ajudar a isolar que o trabalho seja feito somente na CPU e no sistema de memória. Por este motivo de trabalhar com a memoria de maneira isolada, optamos por essa aplicação.

3.2. 473.astar

Esta aplicação é derivada de uma biblioteca de busca de caminho 2D, utilizada em inteligência artificial de jogos. Ela foi desenvolvida por Lev Dymchenko na linguagem C++. Possui três implementações distintas de algoritmos de busca de caminho: O primeiro atua sobre mapas com tipos de terrenos atravessáveis e não-atravessáveis, o segundo atua sobre mapas com tipos de terrenos diferentes e velocidades de deslocamento distintas. O terceiro atua sobre grafos, que são formados por regiões de mapas com relação de vizinhança. Escolhemos este benchmark por se tratar de percurso de caminho e manipulação de grafos.

3.3. 458.sjeng

Este benchmark foi desenvolvido na linguagem ANSI C, por Gian-Carlo Pascutto e Vincent Diepeveen. Ele é baseado na versão 11.2 do Sjeng, que é um programa que joga

xadrez e suas diferentes variantes, como crazyhouse (similar ao Shogi) e antixadrez. O programa tenta então, encontrar o melhor movimento a ser feito, com base na combinação de alpha e beta ou árvores de busca com números de prioridade, ordenação avançada de movimentações, avaliação de posição e heurística de remoção direta. Resumidamente então, o programa irá explorar a árvore de variações resultante de uma determinada posição até uma profundidade especificada, estendendo variações interessantes, e descartando as duvidosas ou irrelevantes. A partir desta árvore, é determinada a linha de jogo ideal para ambos os jogadores ("variação de princípio"), bem como uma pontuação que reflete o equilíbrio de poder entre os dois. Pelo fato de tentar prever a melhor jogada, escolhemos esta aplicação por acreditar que ela poderá fazer um bom uso do prefetcher.

3.4. 450.soplex

Esta aplicação foi, desenvolvida por Roland Wunderling, Thorsten Koch e Tobias Achterberg, na linguagem ANSI C++, é baseada na versão 1.2.1 do SoPlex, que resolve problemas lineares usando o algoritmo Simplex. O programa linear é composto por uma matriz esparsa MxN (A), um vetor de dimensão M (B) e um vetor de coeficiente de função objetivo de dimensão N (C). Ele tem por objetivo encontrar um vetor (X), onde:

minimiza C'X , aplica AX <= B, com X >= 0.

A matriz A é bastante escassa na prática, e por isso, o programa usa algoritmos para álgebra linear esparsa, em particular uma fatoração LU esparsa e rotinas de solução apropriadas para os sistemas de equações triangulares resultantes. Por se tratar de manipulação da matrizes, e sua exploração de localidade espacial, optamos por escolher esse benchmark.

3.5. 453.povray

Este benchmark, desenvolvido em linguagem ISO C++ é um ray-Tracing. O ray-tracing é uma técnica de renderização que calcula a imagem de uma cena, simulando a maneira como os raios de luz viajam no mundo real, mas o faz de trás para frente. No mundo real, os raios de luz são emitidos a partir de uma fonte de luz e iluminam objetos. A luz reflete nos objetos ou passa através de objetos transparentes. Essa luz refletida atinge o olho humano ou a lente da câmera. Como a grande maioria dos raios nunca atinge um observador, levaria uma eternidade para traçar uma cena, por isso os ray-tracers, como o POV-Ray começam com sua câmera simulada e traçam raios de trás para a cena. O usuário especifica a localização da câmera, fontes de luz e objetos, bem como as texturas da superfície e seus interiores.

Para cada pixel, os raios são disparados da câmera para a cena para ver se ela se cruza com algum dos objetos na cena. Toda vez que um objeto é atingido, a cor da superfície nesse ponto é calculada. Com esse objetivo, os raios são enviados a cada fonte de luz para determinar a quantidade de luz que sai dela ou se o objeto está na sombra. Para objetos refletivos ou refrativos, mais raios são traçados para determinar a contribuição da luz refletida e refratada para a cor final da superfície.

O POV-Ray suporta 30 objetos geométricos diferentes, incluindo quadrados, esferas, cilindros, polígonos, malhas, isosuperfícies e geometria sólida construtiva. Interseções de raios com objetos de geometria são calculadas resolvendo equações matemáticas complexas diretamente ou por algoritmos de aproximação numérica.

4. Resultados

4.1. Cache replacement - FIFO

4.1.1. IPC

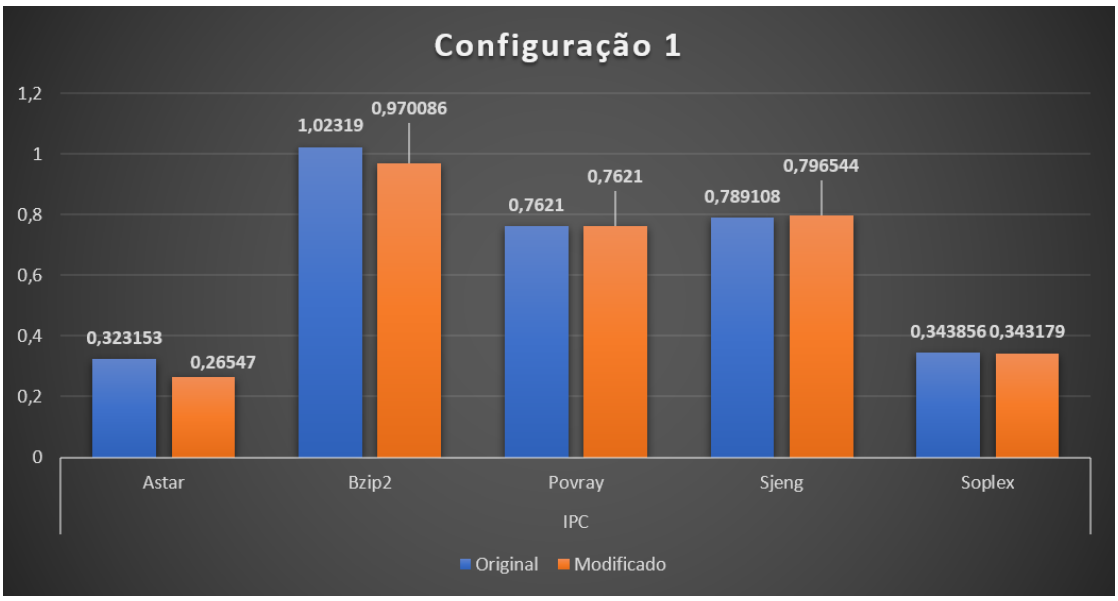

Figura 1. IPC - Configuração 1

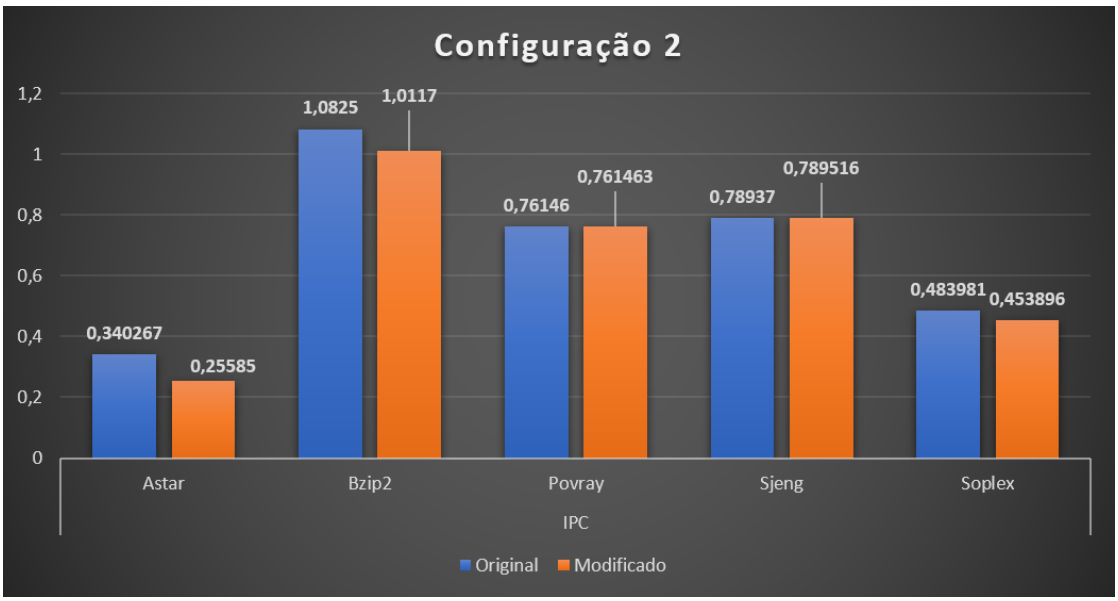

Figura 2. IPC - Configuração 2

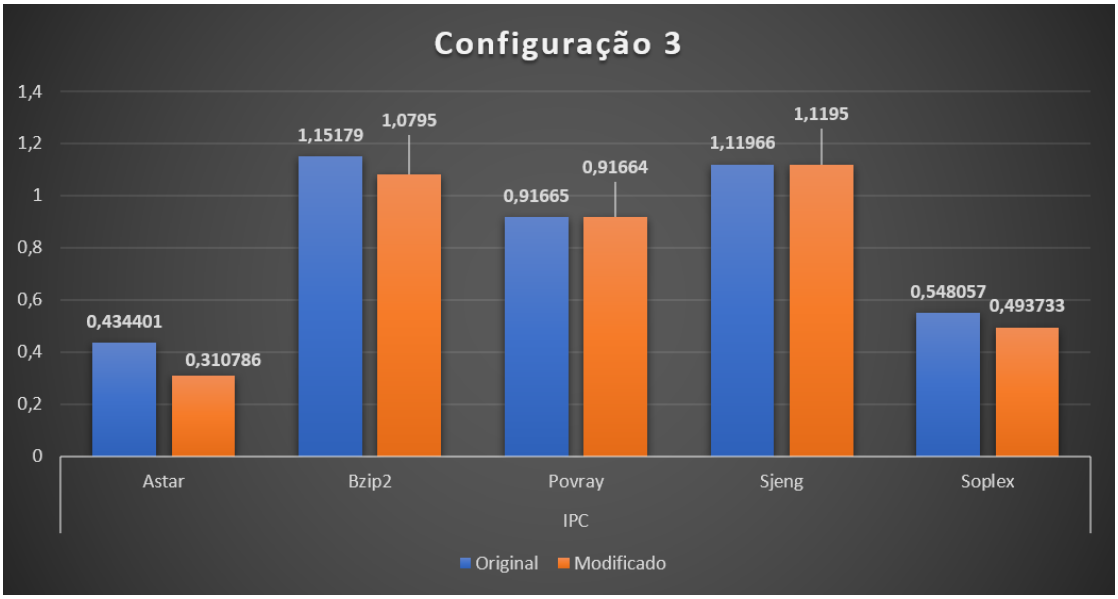

Figura 3. IPC - Configuração 3

4.1.2. Número de Ciclos

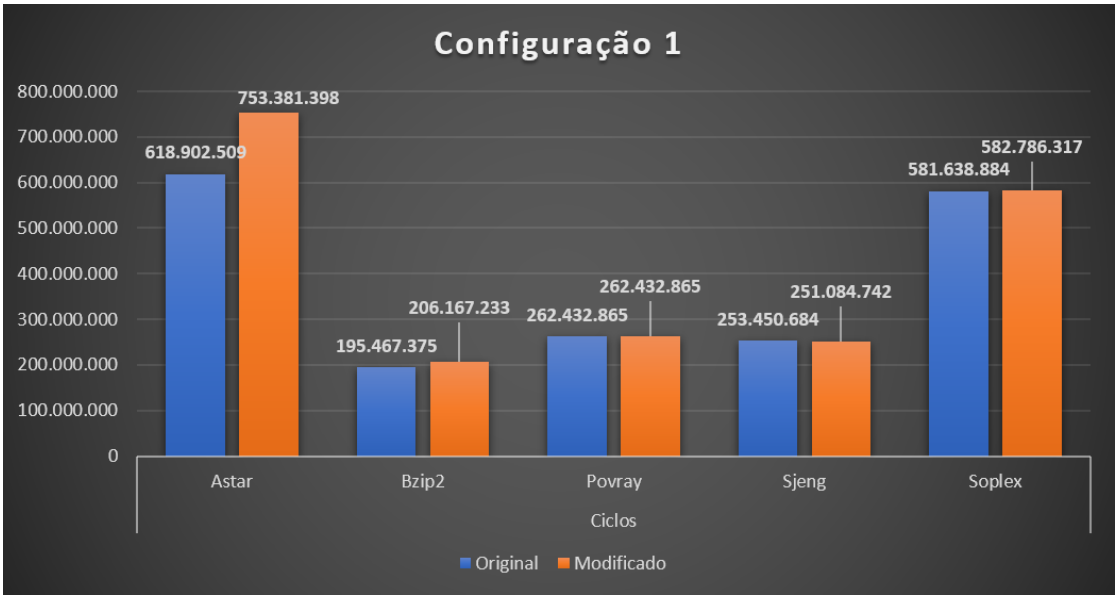

Figura 4. Ciclos - Configuração 1

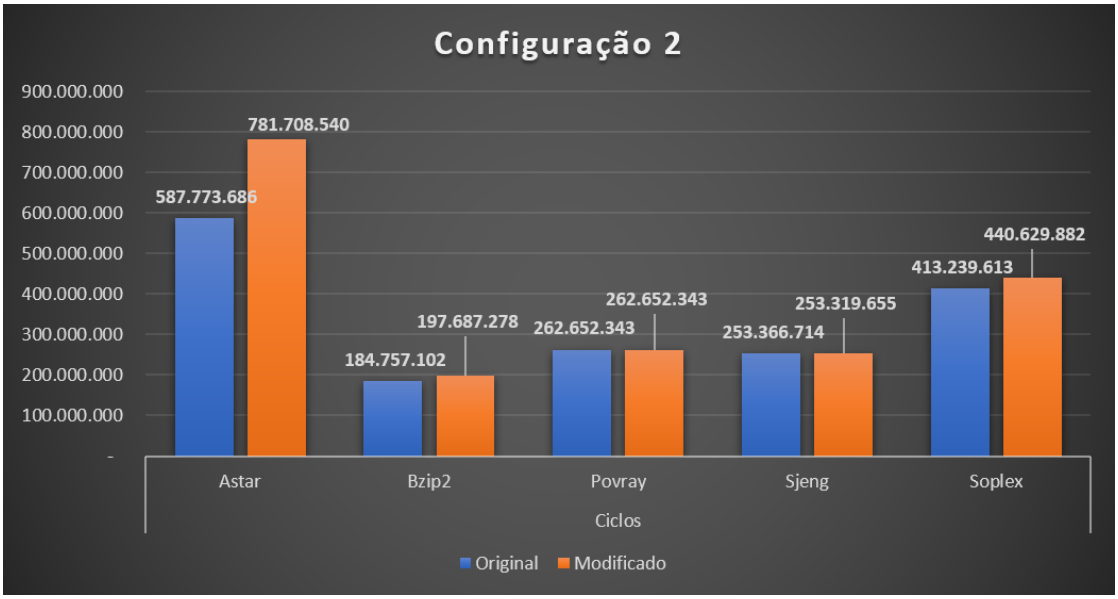

Figura 5. Ciclos - Configuração 2

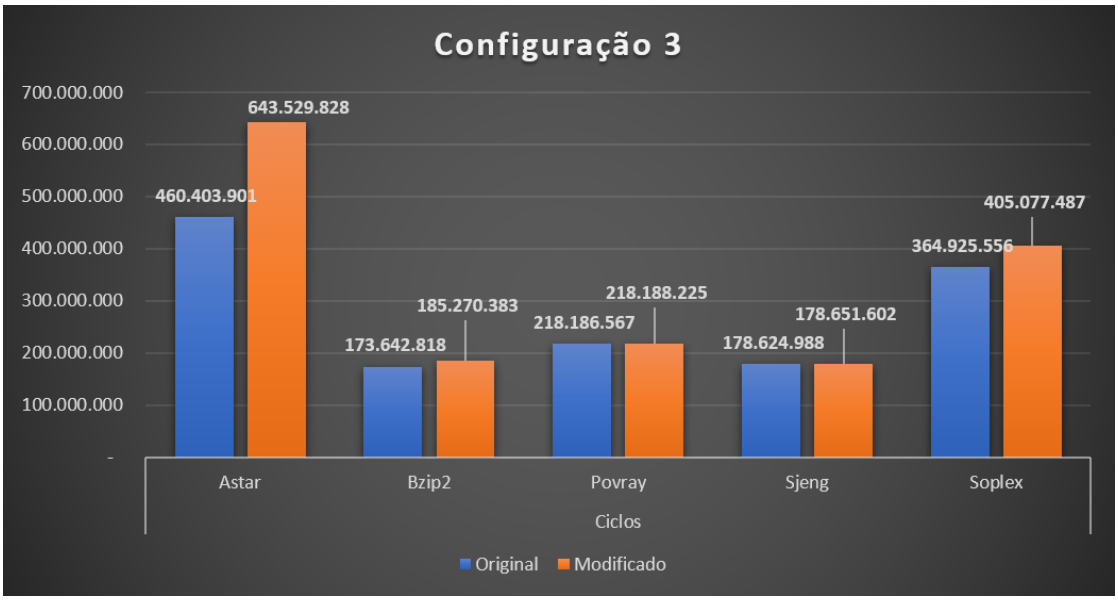

Figura 6. Ciclos - Configuração 3

4.1.3. LLC - Access, Hit e Miss

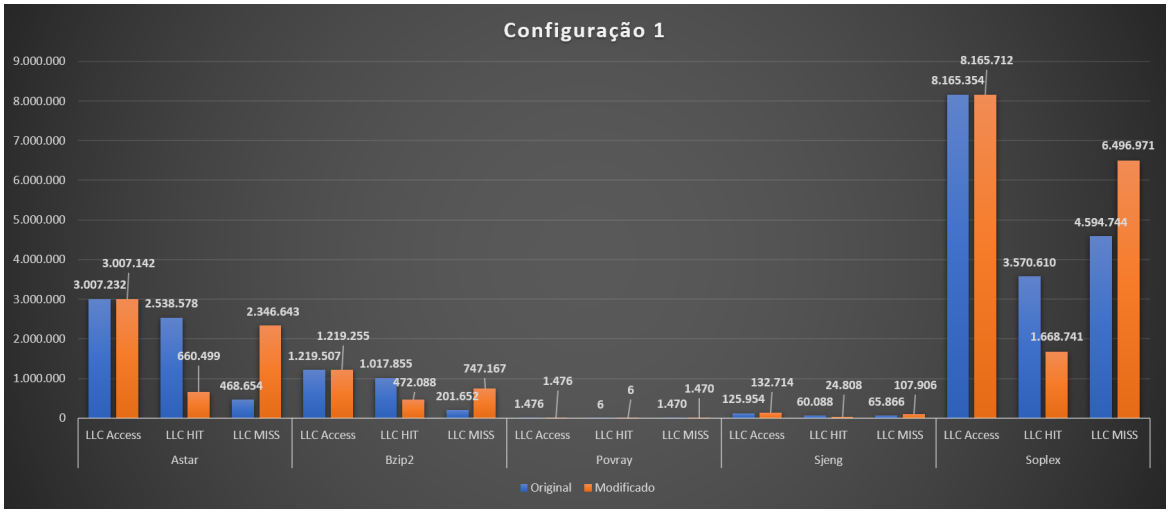

Figura 7. LLC - Configuração 1

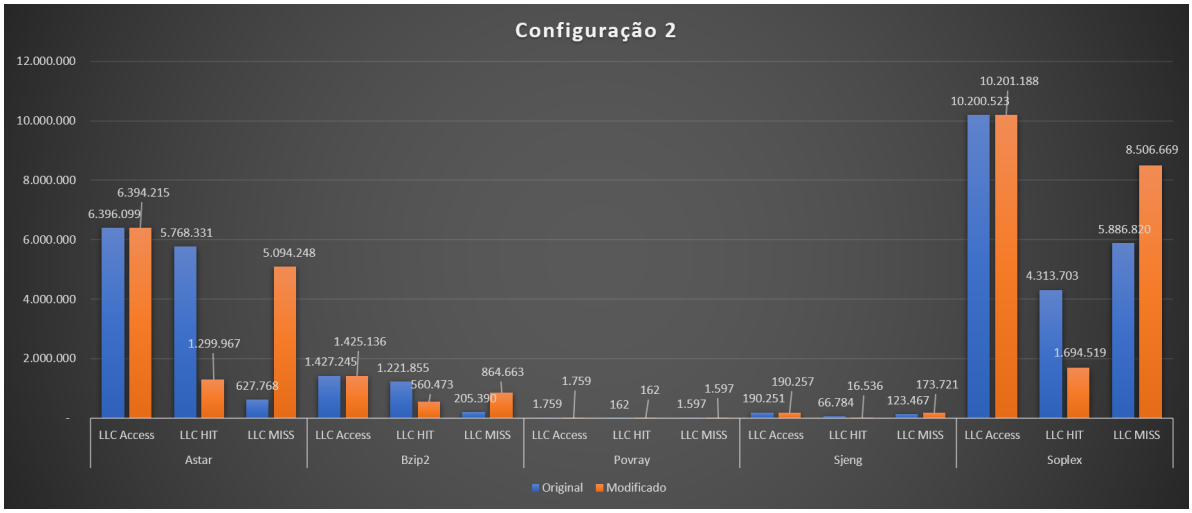

Figura 8. LLC - Configuração 2

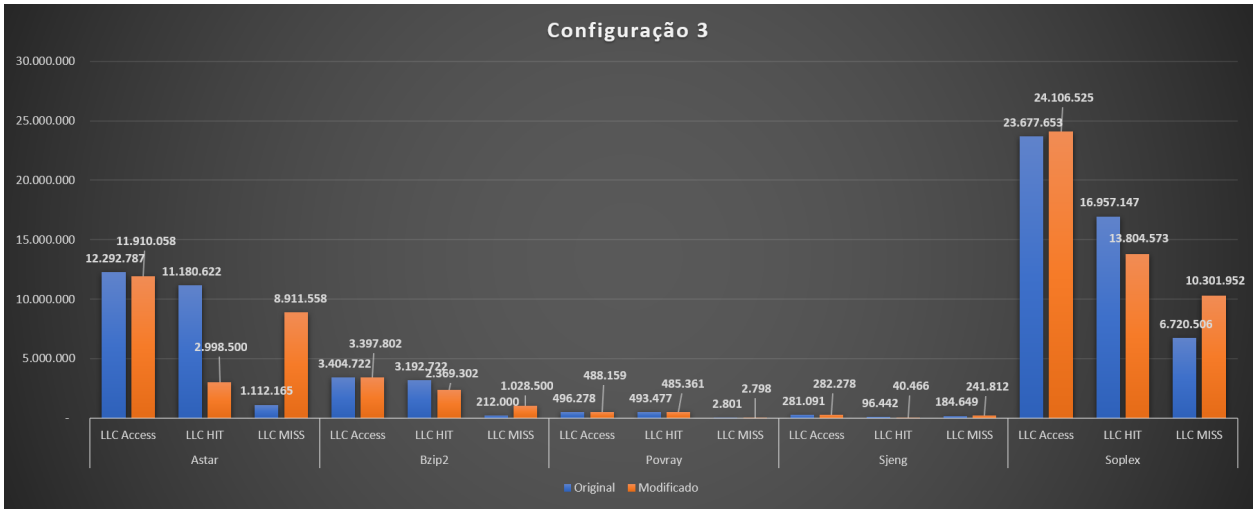

Figura 9. LLC - Configuração 3

4.2. Branch Predictors

4.2.1. IPC

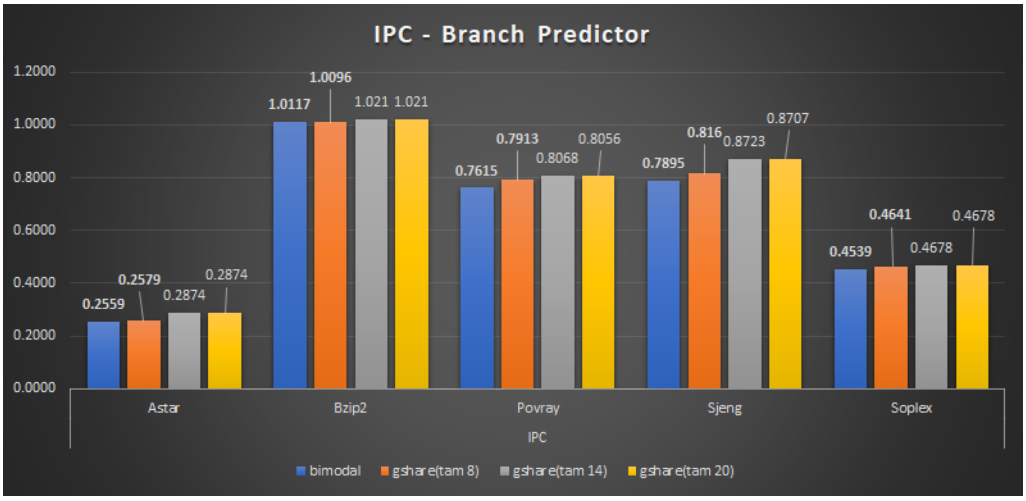

Figura 10. IPC - Branch Predictor

4.2.2. Precisão do Branch Predictor

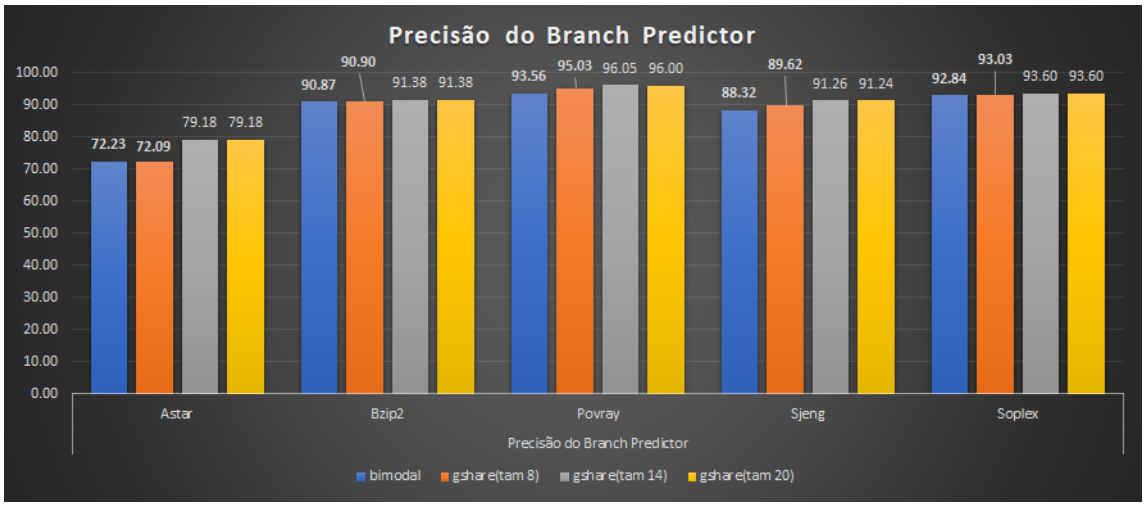

Figura 11. Precisão do Branch Predictor

5. Análise

5.1. Cache Replacement - FIFO

Analisando os resultados obtidos referentes a implementação do algoritmo FIFO para substituição de cache, verificamos que os resultados foram piores em relação aos resultados da baseline. Isto já era o esperado uma vez que a política de substituição FIFO não se aproveita em nada de localidade temporal, pois se uma certa posição de memória foi a primeira a ser ocupada ela obrigatoriamente será a primeira a ser retirada, a menos que este endereço de memória seja ocupado mais rápido que seu contator de idade aumente, assim, este endereço, por um intervalo de tempo, não chegará a possuir a última posição na fila FIFO.

Contudo, ficamos surpresos com alguns desempenhos extremamente negativos, pois apesar de esperarmos que fosse piorar, não achávamos que pioraria no nível que apresentou nos resultados.

5.1.1. Número de ciclos e IPC

A aplicação Astar se demonstrou uma aplicação bastante negativa nesse aspecto para nossa implementação FIFO. Uma vez que nas 3 configurações ela teve um aumento considerável no número de ciclos. Na configuração 1 (figura 4) tivemos um aumento de 21.73%, na figura 5 obtivemos um aumento de 32.995% no número de ciclos e por final o maior dos aumento na configuração 3 (figura 6) tivemos um aumento de 39.78%. O que demostra que a política de substituição FIFO é muito ruim para algoritmos (programas) com características semelhantes ao Astar. Através do gráfico de miss, hit e access para LLC, verificamos que esta diferença para ciclos de cpu esta relacionado com as taxas de erro observadas, o que comprova nossa premissa.

Como ciclos de cpu e IPC tem uma relação inversa, as mesmas razões que explicam a porcentagem de diferença entre a baseline e os cenários (arquitetura de processadores) utilizados explicam os resultados para IPC.

5.1.2. Miss rate

Tivemos também por conta da política de substituição de cache FIFO uma alta taxa de miss rate, o que indica que esta não é uma boa política de substituição para a cache. E como demostrado através da análise dos gráficos apresentados nas figuras 7, 8 e 9. Podemos ver que por se ter uma substituição ruim, aumenta a quantidade de vezes que é necessário obter dados da memória principal, introduzindo assim uma baixa eficiência de uso do hardware. Mais uma vez verificamos que os resultados que esperávamos não seria tão bons quanto a implementação LRU. Novamente a aplicação Astar - o que inclusive justifica os altos valores de ciclos de relógio encontrados nessa aplicação - chegou a atingir 79.7% de miss rate. Na maioria das aplicações obtivemos resultados condizentes com o dados obtidos em ciclos de cpu e IPC, porém, para a aplicação Povray, obtivemos as taxas de miss rates de 99.59% e 90.79% nas configurações 1 e 2, respectivamente, para apenas 0.57% na configuração 3. Atribuimos isso levemente ao Branch Predictor que mudou para esta configuração. Mas nossa hipótese é de que o que mais causou este impacto seja o LLC Prefetcher, uma vez que nas outras configurações estava em "no"e na 3ª configuração seja utilizado um prefetch "next_line". Isto serve para demonstrar o quanto a utilização de prefetch em caches é importante - inclusive no último nível - pois verificamos uma queda de 16.86% em relação a configuração 1 e 16.93% em relação a 2 do número de ciclos desta aplicação.

5.2. Branch Predictors

A escolha de utilizarmos os diferentes tamanhos de GHB era para verificar se realmente haveria um aumento na precisão da predição de desvios e como isso se correlacionaria com o IPC (ou ciclos de cpu).

Nós esperavamos que teria uma melhora no IPC, uma vez que com maior taxa de acerto na predição dos desvios, teríamos uma eficiência melhor do processador, visto que aproveitariamos melhor o pipeline do processador superescalar.

Através dos gráficos disponíveis nas figuras 10 e 11 verificamos que isso realmente acontece. Porém, este ganho não se mantém para todo o aumento de buffer, visto que após certo tamanho a taxa de acerto acaba sendo equivalente, conforme vemos na figura 11, isto é, do branch predictor gshare com tamanho 8 para o tamanho 14 temos melhora em todas as aplicações com destaque para a Astar que teve um acréscimo de 7.09% em sua precisão. Porém quando analisamos a modificação de 14 para 20 no tamanho do GHB, podemos notar que em algumas aplicações os valores se mantém e em outras chega até mesmo a diminuir levemente, como por exemplo na Povray que diminui 0.05%. Indicando assim que possivelmente possa haver, com um GHB grande, desvios que não tem correlação alguma, tendo uma correlação forçada por conta desse buffer maior. Outra hipótese é que o preditor tenha um limite de atuação, em que após certo ponto o maior ponto positivo do gshare, que é a correlação entre os desvios, não consiga melhorar ainda mais o preditor.

Na figura 10 podemos notar um comportamento muito parecido com o da figura 11, concluindo que há uma correlação entre os dois (IPC e Predição de desvios), causada justamente por conta do aproveitamento do pipeline, uma vez que um preditor que erre pouco mantém o pipeline preenchido a maior parte do tempo, evitando assim diversos flushes e bolhas no pipeline. O destaque em IPC cabe a aplicação Bzip2, que através da utilização dos brach predictors testados se manteve com IPC acima de 1 em todos os resultados.

6. Conclusão

Este trabalho teve suma importância para o esclarecimento e fixação dos conteúdos visto em aula, pois ajudou a ligar o conhecimento teórico com o conhecimento prático, contribuindo para o desenvolvimento de habilidades e raciocínio crítico muito úteis, tanto fora quanto dentro do meio acadêmico, visto que, não havíamos realizado ainda nenhum trabalho com esse tipo de experimento. Após a sua realização, pudemos perceber o quão importante é aprender a manipular e modificar estruturas arquiteturais de um processador superescalar out-of-order para melhorar o desempenho (IPC) desse processador. Começamos definindo o que iria ser modificado, e, com base em conhecimentos prévios, levantamos algumas hipóteses do que esperávamos de resultados. Após termos realizados os testes, foi possível perceber que de fato, nossas hipóteses se tornaram verdadeiras.

Com a mudança de cache replacement para FIFO, podemos concluir que ela, de fato, apresenta uma piora no desempenho, aumentando o número de ciclos dos programas, diminuindo seu IPC e aumentando a quantidade de miss rate da cache. Já através da mudança dos branch predictors, conseguimos concluir que o aumento do tamanho da GHB apresenta um aumento na precisão da predição de desvios, aumentando também o IPC. É importante ressaltar que essa relação entre tamanho de GHB e melhora da precisão acabou se estabilizando após um determinado tamanho, não mais refletindo melhora e dessa forma apenas aumentando a área do processador.

7. Anexo

Em anexo junto a este relatório consta:

- Uma pasta com os códigos fontes modificados em relação a base do código disponível no github do ChampSim [ChampSim].
- Uma pasta contendo todos os arquivos de resultados gerados durante o trabalho, esses arquivos estão com os resultados brutos gerados automaticamente pelo ChampSim.

Para a realização deste trabalho utilizamos para versionamento o github, disponível em https://github.com/npalff/ChampSim [ChampSim et al. 2020].

Referências

Alff, N. P. (2019). Exploração de espaço de projeto em processadores superescalar configuráveis. Apresentação de Trabalho e Pôster. Salão UFRGS 2019: XXXI SALÃO DE INICIAÇÃO CIENTÍFICA DA UFRGS.

Belady, L., Nelson, R., and Shedler, G. (1969). An anomaly in space-time characteristics of certain programs running in a paging machine. *Communications of the ACM*, 12:349–353.

ChampSim. Champsim. `https://github.com/ChampSim/ChampSim`. Acesso em: 17 de dezembro de 2019.

ChampSim, Alff, N. P., Bandeira, A. E., and Kritli, M. (2020). Champsim. `https://github.com/npalff/ChampSim`. Acesso em: 03 de janeiro de 2020.

Mcfarling, S. (1993). Combining branch predictors. *DEC WRL Technical Report*.

SPEC (2006). 458.sjeng. `https://www.spec.org/auto/cpu2006/Docs/458.sjeng.html`. Acesso em: 17 de dezembro de 2019.

SPEC (2008). 453.povray. `https://www.spec.org/auto/cpu2006/Docs/453.povray.html`. Acesso em: 17 de dezembro de 2019.

SPEC (2011a). 401.bzip2. `https://www.spec.org/auto/cpu2006/Docs/401.bzip2.html`. Acesso em: 17 de dezembro de 2019.

SPEC (2011b). 450.soplex. `https://www.spec.org/auto/cpu2006/Docs/450.soplex.html`. Acesso em: 17 de dezembro de 2019.

SPEC (2011c). 473.astar. `https://www.spec.org/auto/cpu2006/Docs/473.astar.html`. Acesso em: 17 de dezembro de 2019.