Aprendendo Hierarquia de Memória e a Exploração das Localidades Espacial e Temporal com o Simulador Amnesia

Danielle D. Vieira, Thiago de Campos R. Nolasco, João Augusto S. Silva, Cecília C. Bouchardet, Henrique C. Freitas
Computer Architecture and Parallel Processing Team (CArT), Departamento de Ciência da Computação (DCC)
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte, Brasil
{ddvieira, tcrnolasco, joao.silva.452811, cbouchardet}@sga.pucminas.br, cota@pucminas.br

***Resumo*—A evolução das tecnologias exige cada vez mais poder de processamento dos computadores (e.g., *smartphones*, tablets, notebooks, desktops e servidores), sendo demandado desempenho e eficiência. Um dos principais componentes associados ao desempenho de um computador é a memória e, portanto, o quão rápido um dado ou instrução pode ser acessado pelo processador. Durante a graduação em Ciência da Computação, os estudantes estão imersos a diversas disciplinas capazes de trazer compreensão da importância da memória. No entanto, uma delas por objetivo, trata a hierarquia de memória com destaque: Arquitetura de Computadores. Em linhas gerais, aprender os efeitos positivos e negativos de uma exploração de memória está no entendimento de como explorar a localidade espacial e temporal de forma a reduzir os impactos em *cache miss* e *page fault*. Por este motivo, este artigo apresenta como objetivo principal um estudo e abordagem de aprendizado de hierarquia de memória usando o simulador Amnesia. Esta abordagem tem sido utilizada na disciplina de Arquitetura de Computadores III. Entre os resultados alcançados está este artigo, que em sua versão inicial, foi elaborado como um dos produtos do trabalho da disciplina. Como contribuição, está a abordagem de aprendizado e o amadurecimento dos estudantes envolvidos no trabalho.**

***Palavras-chave*—Arquitetura de Computadores, Hierarquia de Memória, Simulador, Amnesia.**

I. Introdução

Com o aumento da quantidade de dados e a complexidade dos novos sistemas, a melhoria dos tipos de memória se torna cada vez mais importante, visando a garantia de um bom desempenho dos sistemas computacionais. Pensando nisso, arquitetos e engenheiros da área buscam soluções para que os projetos arquiteturais acompanhem as demandas tecnológicas, ou seja, desempenho, escalabilidade e eficiência no armazenamento e processamento dos dados. Nesse contexto, simuladores [1]–[5] são de vital importância para o estudo, desenvolvimento e teste de novas arquiteturas.

Hierarquia de memória é um conceito crucial difundido atualmente para viabilizar a eficiência e desempenho dos sistemas computacionais modernos. Assim, como dito por Patterson e Hennessy [6], ela foi desenvolvida com o objetivo de explorar tamanho e velocidade. Como o nome indica, o conceito define que as memórias devem estar dispostas em níveis e em hierarquia - em formato de pirâmide -, que o tipo de memória que está na base - geralmente chamada de Memória Secundária - é a maior em armazenamento e a mais barata, tendo como *trade-off* ser lenta. Por outro lado, a que se encontra no topo é a mais rápida, porém é mais cara e menor. Conforme Welch [7], o autor desenvolve uma maneira de calcular o tamanho e velocidade das memórias em sistemas com o intuito de minimizar o custo e taxa média de acesso, i.e., balancear o *trade-off* para reduzir o tempo de latência de um dado.

Outra maneira de otimizar um sistema é apresentada por Denning [8], onde é dito que toda computação em execução gera referência para algum objeto (e.g., páginas de memória, disco). Esses objetos podem ser acessados mais ou menos vezes a dependender do programa em execução. Deste modo, pesquisadores da área criaram os conceitos de localidade espacial e temporal para reduzir acessos às memórias que se encontram em níveis mais baixos da hierarquia, assim, diminuindo a latência. O primeiro tipo de localidade é uma presunção de que dados próximos no espaço terão maior probabilidade de serem acessados em sequência. Já a segunda, é a inferência de que um dado acessado recentemente terá maior probabilidade de ser acessado em um futuro breve.

Em ambas alternativas de exploração de localidade, o intuito final é reduzir a quantidade de vezes que um dado não é encontrado na memória, ou seja, reduzir a quantidade de *cache miss* e *page fault*. O primeiro conceito está associado aos níveis de memória cache, e ele acontece quando o processador solicita um dado, porém ele não é encontrado em nenhum bloco nos diferentes níveis de cache, tendo então que fazer acesso à memória primária. O segundo está associado à memória virtual [9], e é dado quando uma página solicitada não é encontrada na memória principal, forçando então, buscar pela página solicitada na memória secundária e inseri-la na memória primária, o que requer um tempo de transferência muito alto.

Os quatro primeiros autores são estudantes de graduação em Ciência da Computação e executaram este trabalho na disciplina de Arquitetura de Computadores III, sob orientação do quinto autor e professor da disciplina. A atividade (estudo, configurações, simulações, avaliações e 1ª versão deste artigo) foi realizada com um deadline de aproximadamente 5 semanas. O artigo original foi atualizado para esta submissão sem comprometer a contribuição dos estudantes envolvidos.

Neste trabalho serão apresentadas diferentes arquiteturas com o intuito de analisar o impacto que alterações podem ter na redução do *cache miss* e *page fault*. Como explicitado por Hill e Smith [10] e Tao, Schloissnig e Karl [11], e aprendido no presente trabalho, uma das formas mais práticas de redução da latência de execução de um código está diretamente associado à exploração das localidades. Tendo isso em mente, quando relacionados às caches, as principais alterações estudadas estão associadas à exploração da localidade espacial e temporal, e ao aumento de níveis de cache. Além disso, quando está em nível de memória primária, as alterações analisadas incluem aumento da tabela de páginas, aumento da memória e a inclusão da TLB (*Translation Lookaside Buffer*).

De acordo com Hill e Smith [10], há 3 parâmetros essenciais para análise de memória cache, sendo eles: i) tamanho geral da cache; ii) tamanho do bloco por conjunto; iii) níveis de associatividade. Uma das análises do artigo está em torno do terceiro parâmetro, apresentando o impacto que a associatividade na cache tem sobre a CPU. Mostrando, por meio de gráficos, como o aumento das vias diminuem o *miss ratio* quando deseja-se acessar um dado. No presente trabalho, serão utilizados conceitos e inspirações sobre memória cache e associatividade, se atentando também aos limites encontrados no trabalho referenciado.

Tao, Schloissnig e Karl [11] propuseram e desenvolveram uma ferramenta de análise de otimização que observa diretamente os acessos à memória de uma aplicação em seu tempo de execução. A primeira otimização proposta é a identificação de grupos e padrões de memórias acessadas recorrentemente, com isso, identifica-se grupos de endereços que podem estar no mesmo conjunto, explorando a localidade espacial. Ademais, é proposto o cálculo da distância de reutilização, de distância de retorno e *hit/miss*. Caso um acesso à cache retorne como *miss*, será informado ao usuário quantos "passos para trás" aquele acesso deve dar para evitar o *miss* encontrado, incentivando a exploração da localidade temporal. No presente artigo, é utilizado como inspiração alguns conceitos apresentados para a análise dos resultados de simulação.

Em linhas gerais, serão discutidos os conceitos de: i) hierarquia de memória; ii) impacto do aumento de tamanho de cache, de blocos por conjunto, níveis de associatividade e níveis de cache para a diminuição do *cache miss*; iii) impacto do aumento de tamanho da tabela de páginas e inclusão da TLB para a diminuição do *page fault* no nível de memória virtual. Resultados indicaram que, embora a depender do programa em execução, a exploração das localidades propostas neste trabalho apresentou diferenças em tempos de execução, podendo ser prejudiciais, aumentando o tempo decorrido para a execução dos mesmos, mas também podendo ser benéficas para a diminuição do tempo de execução.

O problema motivador deste artigo está na relativa dificuldade de aprender conceitos fundamentais que norteiam um projeto que envolve hierarquia e organização de memória somente com aulas conceituais e expositivas. Portanto, o objetivo deste artigo é apresentar um estudo com auxílio do simulador Amnesia, no qual os estudantes exercitam variações arquiteturais capazes de reforçar os conceitos de hierarquia de memória discutidos em sala de aula. A principal contribuição[1] deste artigo está na abordagem focada na exploração de localidade espacial e temporal, visando ajudar no aprendizado de importantes conceitos na formação de recursos humanos em Ciência da Computação.

O restante deste trabalho está organizado da seguinte maneira. Na Seção II, é feita uma revisão sobre outros trabalhos que fizeram análises sobre o impacto da exploração das localidades e aumento de níveis de cache na melhoria de desempenho. Já na Seção IV, é descrita a metodologia empregada no trabalho mostrando as arquiteturas e os *traces* analisados. Na Seção V, são discutidos os resultados obtidos associando-os aos conceitos previamente citados e explicando o porquê das arquiteturas apresentarem os respectivos valores para os *traces* executados. Por fim, na Seção VI é feita a conclusão do trabalho e a proposição de trabalhos futuros.

II. Trabalhos Relacionados

Historicamente, o grupo de pesquisa CArT da PUC Minas tem trabalhado em uma linha de pesquisa voltada para educação em computação. Alguns simuladores já foram avaliados, em boa parte, simuladores completos [12]–[14], o que possibilita configuração de sistema operacional, execução de programas reais, etc. No entanto, estes tipos de simuladores demandam uma experiência maior no conhecimento de arquitetura de computadores, o que dificulta estudantes iniciantes na área. Esta seção dá ênfase em artigos voltados para educação e outros sem este objetivo, mas que contribuíram com este trabalho de hierarquia e organização de memória. O presente trabalho fez parte do aprendizado de quatro estudantes (primeiros autores) de Ciência da Computação e o estudo da literatura escolhida por eles ajudou a entender a evolução da área.

Conforme Tiosso et al. [3], os autores apresentam o simulador e objeto de aprendizagem Amnesia que mostra, de maneira visual, o funcionamento simulado das memórias cache e virtual nos computadores na arquitetura de von Neumann. Nesse trabalho, o Amnesia foi utilizado para o desenvolvimento e simulação das arquiteturas explorando localidade temporal e espacial. Ademais, apresenta como o Amnesia ajudou os alunos a aprenderem melhor os conceitos relacionados a disciplina de Organização e Arquitetura de Computadores. No presente trabalho, o Amnesia também ajudou neste trabalho a simular e analisar o desempenho na busca de instruções utilizando memória cache e virtual.

O trabalho apresentado por Coutinho et al. [15] descreve o projeto de um simulador de hierarquia de memória (cache e memória virtual) com suporte a caches separadas. O simulador foi desenvolvido como pesquisa de graduação, e objetiva ser uma ferramenta didática que facilita o aprendizado de arquitetura de computadores em nível de graduação e pós-graduação. Dos mesmos autores, o Web-MHE [16] é um

[1] Arquivos de simulação disponíveis em: https://github.com/cart-pucminas/amnesia

ambiente didático que pode ser acessado via navegador *web*, que oferece conteúdo e o simulador Web-MHSim para estudo de hierarquia de memória.

Moreno et al. [17] apresentam o simulador MNEME (nome inspirado na mitologia grega). O objetivo deste simulador é oferecer uma visão completa de hierarquia de memória. Como forma de validação, os autores fizeram uso do simulador durante dois anos acadêmicos como ferramenta de auxílio ao aprendizado para os estudantes.

Já em Freitas et al. [18], os autores apresentam um simulador baseado em realidade virtual para auxiliar no ensino e aprendizado de gerência de memória. Embora o foco esteja mais voltado para a disciplina de sistemas operacionais, a proposta possui fundamental alcance para explicitar conceitos de hierarquia de memória. Oitenta alunos fizeram parte do estudo de validação do simulador.

A principal diferença deste trabalho para os correlatos está na abordagem de uso do simulador Amnesia com o objetivo de aprendizado de hierarquia e organização de memória. Embora este seja uma das metas do simulador, entende-se que a abordagem adotada neste artigo se diferencia dos demais, uma vez que é focada a atender demanda típica de aprendizado em disciplinas de Arquitetura de Computadores.

III. O Simulador Amnesia

Amnesia é um simulador flexível, que permite os usuários montarem uma arquitetura e *trace* com as instruções que serão executadas da maneira que desejarem. A arquitetura é construída em um arquivo com extensão XML (*Extensible Markup Language*) e o *trace* em um arquivo com extensão txt (*Text*). O Amnesia é uma ferramenta importante para a exploração da hierarquia de memória em sistemas computacionais pois, permite simular o comportamento de memórias cache, memória principal e memória virtual paginada. Desde 2007, alunos de graduação e pós-graduação do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP) desenvolvem o Amnesia, orientados pelos professores Paulo Sérgio Lopes de Souza e Sarita Mazzini Bruschi. Além disso, o simulador está disponível gratuitamente para uso acadêmico e de pesquisa, permitindo que estudantes e pesquisadores explorem e experimentem diferentes configurações de hierarquia de memória. Na PUC Minas, o simulador Amnesia é utilizado na disciplina de Arquitetura de Computadores III desde 2017.

Um arquivo de rastro, que também é chamado de *trace*, contém os acessos realizados por um programa à memória, acessos de leitura, escrita de dados ou busca de instruções. Cada linha deste arquivo é composta de uma dupla: rótulo (decimal) e endereço (hexadecimal). Qualquer outra informação é vista como um comentário. Os *traces* utilizados neste artigo possuem rótulo igual a 2, i.e., busca de instruções.

IV. Abordagem de Ensino e Aprendizado

Este artigo é resultado de um trabalho prático desenvolvido na disciplina de Arquitetura de Computadores III. Em linhas gerais, o enunciado do trabalho especifica que o simulador Amnesia deve ser utilizado para estudo de localidades espacial e temporal. É obrigatório simulações que envolvam no mínimo 3 cenários com variações em caches e em memória virtual. Duas aulas foram realizadas em laboratório para acompanhamento e auxílio na solução de dúvidas. Uma aula teórica foi utilizada para explicação de como o artigo deveria ser escrito. Os trabalhos foram apresentados e discutidos em sala de aula e o artigo enviado para avaliação uma semana depois.

Neste trabalho em específico, foi definido que a exploração das localidades espacial e temporal seria realizada com *traces* para executar instruções. Também foi utilizada a interface de desenvolvimento *Visual Studio Code* para a construção dos arquivos de arquitetura e *trace*.

Foram criados três tipos de *traces* com tamanho de trinta linhas cada. O *Trace* 1, foi criado de forma aleatória, não tendo como objetivo explorar nenhuma localidade. Nele há buscas de instruções aleatórias para verificar como será o desempenho nesta situação. Já o *Trace* 2, foi criado a fim de explorar a localidade espacial, buscando instruções em posições sequenciais de um bloco de palavras na memória cache ou em uma página com blocos, se for em uma memória virtual. Por fim, o *Trace* 3, foi criado a fim de explorar a localidade temporal, onde uma mesma instrução é buscada várias vezes, sendo que terá pouca substituição dos dados na memória onde está a instrução.

Trace 1 10 primeiras linhas do *Trace* Aleatório.

```
2 5    // busca de instrução no endereço 5
2 c    // busca de instrução no endereço c
2 b    // busca de instrução no endereço b
2 a    // busca de instrução no endereço a
2 f    // busca de instrução no endereço f
2 8    // busca de instrução no endereço 8
2 0    // busca de instrução no endereço 0
2 9    // busca de instrução no endereço 9
2 4    // busca de instrução no endereço 4
2 1    // busca de instrução no endereço 1
```

Trace 2 10 primeiras linhas do *Trace* Espacial.

```
2 0    // busca de instrução no endereço 0
2 1    // busca de instrução no endereço 1
2 2    // busca de instrução no endereço 2
2 3    // busca de instrução no endereço 3
2 4    // busca de instrução no endereço 4
2 3    // busca de instrução no endereço 3
2 5    // busca de instrução no endereço 5
2 6    // busca de instrução no endereço 6
2 7    // busca de instrução no endereço 7
2 8    // busca de instrução no endereço 8
```

A. Memória Cache

Para a execução das simulações voltadas à memória cache, foram idealizados 3 cenários distintos com algumas alterações na arquitetura base, visando explorar a localidade espacial e

Trace 3 10 primeiras linhas do *Trace* Temporal.

2 1	// busca de instrução no endereço 1
2 5	// busca de instrução no endereço 5
2 1	// busca de instrução no endereço 1
2 2	// busca de instrução no endereço 2
2 7	// busca de instrução no endereço 7
2 a	// busca de instrução no endereço a
2 1	// busca de instrução no endereço 1
2 2	// busca de instrução no endereço 2
2 6	// busca de instrução no endereço 6
2 4	// busca de instrução no endereço 4

temporal, assim como diminuir o acesso à memória principal. A arquitetura base, conforme Tabela I, é utilizada como referência de desempenho para as alterações que serão feitas. O intuito é verificar se houve uma melhora ou piora nos resultados, ao buscar as instruções, quando feito a exploração das localidades.

No cenário 1, representado na Tabela I, a arquitetura construída tem um número de vias maior que a arquitetura base, aumentando assim a associatividade por conjunto até ser completamente associativo. O acréscimo de vias tem como objetivo aumentar a exploração da localidade temporal, tendo em vista que caberá mais blocos em apenas um conjunto. Deste modo, haverá uma diminuição de substituição de dados na memória cache, fazendo com que os dados permaneçam por mais tempo em cache, aumentando o número de *hits*, visto que um dado acessado anteriormente pode ser acessado novamente sem necessidade de ir à memória principal. Neste cenário há variações apenas do número de vias com relação a arquitetura base. As quantidades de vias simuladas são de 2, 4 e 8.

No cenário 2, conforme Tabela I, existe a variação do número de palavras por bloco em relação a arquitetura base, visando explorar a localidade espacial. O intuito é de que, ao buscar um dado, i.e., instrução, os dados próximos também sejam carregados à memória, e então, ao buscar uma das instruções próximas à anterior ela já esteja carregada na cache, assim, terá aumento de número de *hits*, e consequentemente, melhora no desempenho. O número de palavras por bloco simulados neste cenário foram 2, 4 e 8.

Por fim, no cenário 3, conforme Tabela I, é criado o segundo nível na memória cache, onde o tamanho das caches L1 e L2 é incrementado a fim de analisar seu desempenho na busca de instruções, visando a diminuição de acessos na memória principal. As variações dos tamanhos nas memórias caches nos níveis são:

- Nível 1 com tamanho de dados de 8 *Bytes* e nível 2 com 16 *Bytes*.
- Nível 1 com tamanho de dados de 8 *Bytes* e nível 2 com 32 *Bytes*.
- Nível 1 com tamanho de dados de 16 *Bytes* e nível 2 com 16 *Bytes*.
- Nível 1 com tamanho de dados de 16 *Bytes* e nível 2 com 32 *Bytes*.
- Nível 1 com tamanho de dados de 32 *Bytes* e nível 2 com 32 *Bytes*.

B. Memória Virtual

Para a execução das simulações voltadas para a memória virtual foram também idealizados 3 cenários distintos contendo alterações em relação a arquitetura base. Cada cenário construído tem como objetivo explorar a localidade espacial e temporal, assim como diminuir o número de acessos na memória secundária. Nas arquiteturas com memória virtual não há memória cache, apenas memória principal e secundária. A arquitetura base, apresentada na Tabela II, também é utilizada como referência de desempenho para as alterações que serão feitas para verificar se o desempenho da busca de instruções irá melhorar ou piorar, com a exploração das localidades.

No cenário 1, que pode ser visualizado na Tabela II, as arquiteturas possuem o número de blocos por páginas variando em 2, 4 e 8. O objetivo desta variação é explorar a localidade espacial, no qual ao buscar um dado, i.e., instrução, há grande possibilidade de buscar o seu vizinho e assim aumentar o número de *hits* na TLB (*Translation Lookaside Buffer*) ou na tabela de páginas e, consequentemente, na memória principal.

No cenário 2, apresentado na Tabela II, as arquiteturas possuem o tamanho de dados na memória principal e secundária variando em 32, 64 e 128 *Bytes*. Neste cenário, as alterações foram feitas a fim de explorar a localidade temporal, isto é, ao aumentar o tamanho da memória há maior probabilidade de ter mais dados em memória RAM, logo os dados ficarão por mais tempo na memória principal, necessitando de menos acessos à memória secundária e aumentando o desempenho ao buscar as instruções.

Por fim, no cenário 3, conforme Tabela II, as arquiteturas não possuem a TLB (*Translation Lookaside Buffer*) e é simulado como seria o desempenho com os tamanhos das memórias do cenário 2. A TLB é uma memória cache da tabela de páginas que se encontra no processador. Seu papel é armazenar os mapeamentos de endereços virtuais para endereços físicos de memória das páginas do *working set*. Com a TLB na arquitetura, caso encontre o endereço real nela, não é necessário ir à memória principal duas vezes; o primeiro acesso para consultar e buscar na tabela de páginas pelo endereço real, e o segundo para buscar o dado que está no endereço real. Sem ela o desempenho em tese deve ser pior, tendo em vista que sua latência é de 1 ms e o da memória principal de 10 ms, ou seja, 10 vezes mais tempo é gasto na memória principal do que no acesso à TLB.

V. RESULTADOS

Após a execução das simulações propostas, foram obtidos resultados que objetivam validar conceitos e a diferença no desempenho ao buscar instruções com a exploração das localidades temporal e espacial de acordo com as arquiteturas e arquivos de *trace* elaborados. Conforme os resultados, o desempenho das arquiteturas é influenciado diretamente pelo

Memória Principal	**Cenário Base**	**Cenário 1**	**Cenário 2**	**Cenário 3**
Palavras por Bloco	1	1	**2, 4 e 8**	1
Tamanho de dados (*Bytes*)	64	64	64	64
Ciclos por acesso para leitura	10	10	10	10
Ciclos por acesso para escrita	20	20	20	20
Tempo de ciclo (ms)	1	1	1	1
Memória Cache	Unificada	Unificada	Unificada	**Multinível**
Palavras por Bloco	1	1	**2, 4 e 8**	1
Tamanho de dados (*Bytes*)	32	32	32	***L1 = 8 e L2 = 16, L1 = 8 e L2 = 32, L1 = 16 e L2 = 16, L1 = 16 e L2 = 32, L1 = 32 e L2 = 32***
Ciclos por acesso para leitura	1	1	1	1
Ciclos por acesso para escrita	2	2	2	2
Tempo de ciclo (ms)	1	1	1	1
Nº de vias	1	**2, 4 e 8**	1	1
Política de Escrita	*Write Through*	*Write Through*	*Write Through*	*Write Through*
Algoritmo de substituição	FIFO	FIFO	FIFO	FIFO

TABELA I
ARQUITETURAS COM MEMÓRIA CACHE.

Memória Principal	**Cenário Base**	**Cenário 1**	**Cenário 2**	**Cenário 3**
Palavras por Bloco	1	1	1	1
Tamanho de dados (*Bytes*)	32	32	**64 e 128**	**32, 64 e 128**
Ciclos por acesso para leitura	10	10	10	10
Ciclos por acesso para escrita	20	20	20	20
Tempo de ciclo (ms)	1	1	1	1
Blocos por página na Tabela de Páginas	2	**4 e 8**	2	2
Memória Secundária	-			
Palavras por Bloco	1	1	1	1
Tamanho de dados (*Bytes*)	64	64	**64 e 128**	**64 e 128**
Ciclos por acesso para leitura	100	100	100	100
Ciclos por acesso para escrita	200	200	200	200
Tempo de ciclo (ms)	1	1	1	1
Algoritmo de substituição	FIFO	FIFO	FIFO	FIFO
TLB	Sim	Sim	Sim	**Não**
Tamanho de dados (*Bytes*)	16	16	16	-
Ciclos por acesso para leitura	1	1	1	-
Ciclos por acesso para escrita	2	2	2	-
Tempo de ciclo (ms)	1	1	1	-
Algoritmo de substituição	FIFO	FIFO	FIFO	-

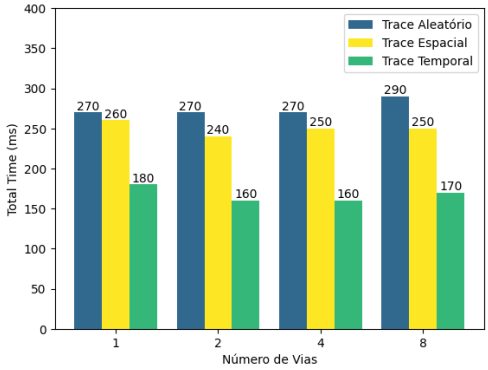
TABELA II
ARQUITETURAS COM MEMÓRIA VIRTUAL.

tipo de *trace*. Os resultados estão divididos nas subseções seguintes, em conformidade com a Seção IV.

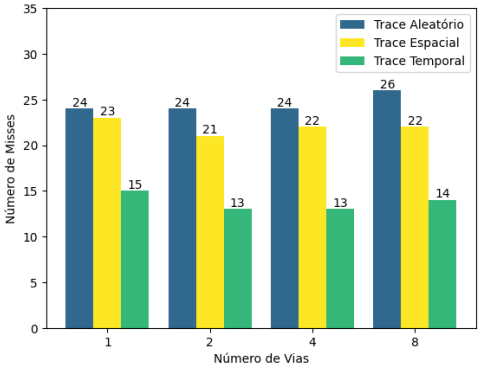
A. Memória Cache

A fim de facilitar o entendimento dos resultados, será realizada a divisão entre cenários para a descrição dos valores obtidos através das simulações utilizando a memória cache.

1) Cenário 1: O aumento de vias no mapeamento da memória cache no cenário 1, conforme arquitetura descrita na Seção IV-A, tem como objetivo diminuir o *cache miss* ao buscar as instruções na memória. É possível analisar na Figura 1(a), que o tempo total para a execução do *Trace* 1 se mantém constante até 4 vias no mapeamento, mas com 8 vias no mapeamento o tempo aumenta devido aos endereços aleatórios do *Trace* 1, pois os blocos são substituídos (Figura 1(b)) na memória cache quando não são encontrados, aumentando o tempo total ao forçar a busca das instruções na memória principal. Não há diminuição no tempo total com o aumento do número de vias, pois os endereços no *Trace* 1 são diferentes e distantes ao ponto de não buscar uma instrução vizinha da outra ou buscar a mesma instrução em um curto intervalo de tempo. Assim, o *Trace* 1 não explora nenhuma das localidades, e constantemente há *miss* e busca de instruções na memória principal. Há 8 *cache misses* compulsórios para preencher a cache em seu primeiro acesso, há *cache misses* por conflitos em que blocos serão substituídos pelo algoritmo de substituição FIFO e *cache miss* de capacidade, tendo em vista que a cache comporta 8 palavras e o *Trace* 1 busca 16 palavras. Há um aumento no tempo para quantidade maior de vias o que sugere um questionamento, afinal, a quantidade de ciclos é a mesma para todas as variações de mapeamento, e mais vias deveriam melhorar a exploração de localidade temporal. A explicação será feita para o *Trace* 3 (temporal), que em teoria seria o que melhor se comportaria em uma arquitetura com mais vias.



(a) Cenário 1: Aumento da Associatividade.

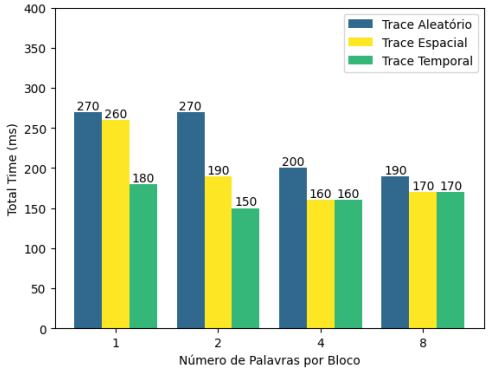


(b) *Misses* cenário 1

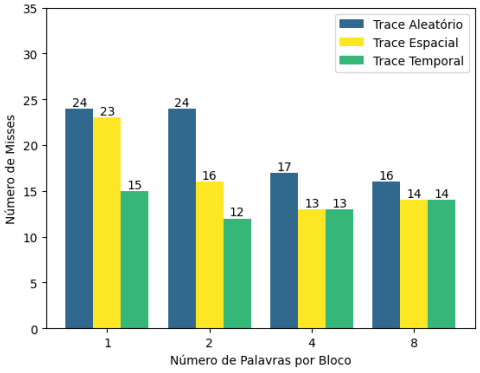
Fig. 1. Memória Cache: Cenário 1.

Já na execução do *Trace* 2 os endereços foram colocados sequencialmente de forma a explorar a localidade espacial ao buscar instruções que são vizinhas e possivelmente estariam no mesmo bloco de palavras. No cenário 1, conforme Figura 1(a), com o *Trace* 2, houve uma pequena melhora no tempo total, porém não foi tão significativa pois, mesmo diminuindo o número de conflitos ao aumentar o número de vias, ainda há muito *miss* (Figura 1(b)) na memória cache fazendo com que tenha muitos acessos a memória principal para buscar instruções. O *miss* se deve aos endereços no *Trace* 2 não se repetirem muito, assim, os blocos são, na sua maioria, substituídos na memória cache. É importante lembrar que somente o *trace* explora localidade espacial. Há variação na associatividade, mas o tamanho do bloco é igual a 1.

Por fim, na execução do *Trace* 3 no cenário 1, conforme Figura 1(a), há o melhor tempo total do cenário, pois tanto o *Trace* 3 quanto a arquitetura foram feitos para explorar a localidade temporal. No *Trace* 3, os endereços estão se repetindo em um curto espaço de tempo, de forma que não sejam muito substituídos na memória cache e se repitam para serem encontrados, obtendo cache *hit*. No mapeamento completamente associativo, na arquitetura com 8 vias, o tempo total aumentou. Inicialmente, isso despertou uma surpresa, porque este mapeamento oferece mais opções para alocar um bloco, o que reduziria os *misses*. Como a simulação não considerou mais ciclos gastos com aumento de vias, foi necessário entender com mais detalhes o comportamento do *trace* temporal nos mapeamentos. A instrução do endereço 7 é mapeada na cache e, no caso de duas vias, permanece no conjunto 3, via 2, até o último passo de simulação (30 no total). Para o mapeamento com 8 vias, completamente associativo, o endereço 7 é substituído no passo 21. O *trace* possui uma sequência de endereços que, depois de uma certa quantidade de passos de simulação, no caso de duas vias, não realiza mais mapeamentos no conjunto 3. Dessa forma, o endereço 7 não é substituído, e este é acessado novamente ao final do *trace*. No caso do mapeamento com 8 vias, o endereço 7, assim como qualquer outro endereço mapeado, está disponível para ser substituído. De acordo com o algoritmo FIFO, no passo 21, é isso que ocorre. O endereço 7 é substituído pelo endereço 2. Outras simulações foram feitas com a política LRU. Houve uma melhoria na redução dos *misses* e no tempo total para executar o *trace*, mas a característica do próprio *trace* resultou novamente em um desempenho pior no mapeamento completamente associativo.



(a) Cenário 2: Aumento no número de palavras por bloco.

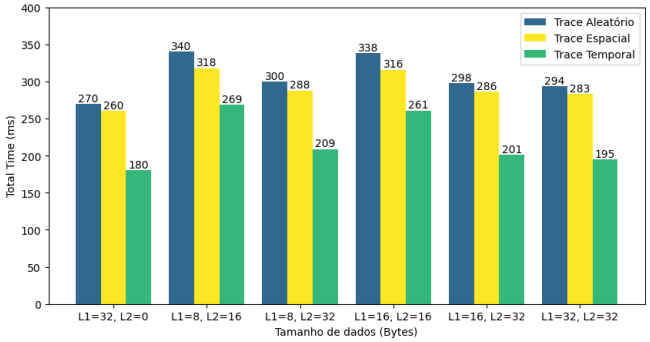


(b) *Misses* cenário 2

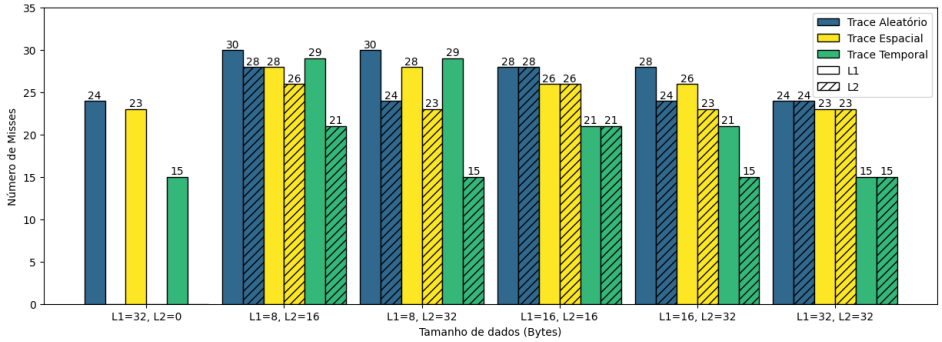
Fig. 2. Memória Cache: Cenário 2.

2) Cenário 2: Já no cenário 2, conforme Figura 2(a), o *Trace* 1 teve seu tempo total reduzido a medida que o número de palavras por bloco aumentou. Este resultado é justificado, uma vez que a arquitetura explora localidade espacial. À medida que mais palavras podiam ser colocadas no mesmo bloco foi possível ter mais *hits* e menos *misses* (Figura 2(b)) na memória cache, pois uma palavra buscada na memória cache tem uma maior chance de ser encontrada na mesma.

Já na execução do *Trace* 2, conforme Figura 2(a), houve uma melhoria no tempo total, pois o *Trace* 2 foi construído



(a) Cenário 3: Caches Multiníveis.



(b) *Misses* cenário 3

Fig. 3. Memória Cache: Cenário 3.

para explorar a localidade espacial e o cenário também. Com a arquitetura armazenando blocos de palavras vizinhas na memória cache e o os endereços do *Trace* 2 buscando palavras vizinhas que possivelmente estejam no mesmo bloco, esse fato faz com que tenha muito *hit* na memória cache, aumentando assim o desempenho na busca de instruções. Para o *Trace* 2, o melhor resultado foi com 4 palavras por bloco, pois com 8 palavras a memória cache obteve apenas um bloco, e quando há *miss* neste bloco, é necessário ir a memória principal e buscar as 8 palavras e substituir na memória cache. Então toda vez que há um *miss*, o tempo de buscar e substituir 8 palavras para apenas uma busca de instrução será grande.

Por fim, na execução do *Trace* 3 utilizando as arquiteturas propostas no cenário 2, conforme mostra a Figura 2(a), há também um bom tempo total quando há duas palavras por bloco. Apesar de diminuir o número de linhas na memória cache pela metade, neste *Trace* 3 foi possível buscar as instruções sem muitas substituições de blocos na cache. Porém, quando o número de palavras por bloco aumenta para 8, há mais *misses*, de capacidade, e consequentemente os blocos são substituídos mais vezes, aumentando o tempo total na busca de instruções.

3) Cenário 3: O cenário 3 tem como objetivo utilizar as caches com níveis diferentes para otimizar o tempo de busca de instruções, pois quanto mais próximo do processador, mais rápido será a busca. Porém, como há variações de tamanho de memória em que alguns tamanhos são pequenos com relação ao tamanho da memória principal, essa otimização não ocorrerá. Isso se dá pelo fato de que, quanto menor a memória cache, menos dados estarão nela e mais dados terão que ser buscados na memória principal, mesmo com a adição de uma nova cache, nível 2, esse problema acontecerá caso ela também seja pequena. É importante ressaltar que o tempo de acesso para buscar uma instrução é maior em uma arquitetura multinível, caso ocorra *miss* em cache, tendo em vista que será necessário analisar se o dado está em todos os níveis de cache antes de acessar a memória principal. No entanto, o que se espera de caches multiníveis é justamente não precisar ir à memória principal. Encontrando em qualquer um dos níveis, o desempenho é muito superior, uma vez que o maior tempo de busca é justamente na memória principal.

No cenário 3, conforme Figura 3(a), o *Trace* 1 teve o pior tempo total dos três cenários, apresentando também a maior quantidade de *misses* (Figura 2(b)). O número de *hits* nas arquiteturas com multiníveis de cache, como por exemplo, em L1 com tamanho de dados de 16 *Bytes* e L2 com 16 *Bytes*, teve um total de zero *hits*. Isso se deve ao mapeamento direto na cache, que causa conflitos de endereços quando os mesmos não são consecutivos ou repetidos em um espaço curto de tempo, causando uma substituição frequente de blocos na memória cache, fazendo com que tenha muitos acessos na memória principal.

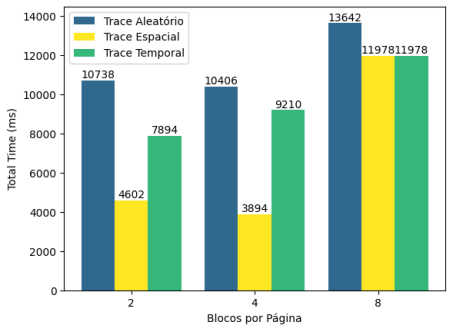
Na execução do *Trace* 2 no cenário 3, conforme mostrado na Figura 3(a), é possível visualizar que o menor tempo total se deu quando a cache de nível 1 armazenava 8 palavras, justamente pela memória cache ser capaz de armazenar mais palavras, aumentando a exploração de localidade espacial. Nas arquiteturas com cache de nível 2 que armazena 8 palavras o tempo foi um pouco superior de quando a cache de nível 1 armazenava 8 palavras, isso porque quando a instrução não

é encontrada na L1 de 2 ou 4 palavras, a mesma é buscada em L2 e a cada vez que isso acontecia o tempo é somado em 1 ms e caso não fosse encontrado em L2 é necessário buscar a instrução na memória principal, somando mais 10 ms.

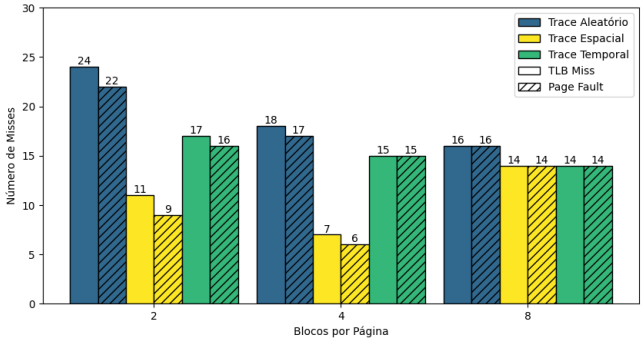
Na execução do *Trace* 3, no cenário 3, representada na Figura 3(a), houve comportamento semelhante em termos de desempenho, uma vez que os piores tempos totais ocorrem nesta simulação em comparação com os três cenários da memória cache. Isso porque, além do mapeamento ser direto e haver muitas colisões e substituições de dados na memória cache, os níveis em cache tem tamanhos muito pequenos e faz com que poucos dados sejam armazenados, portanto, mais acessos à memória principal precisam ser feitos.

B. Memória Virtual

A fim de facilitar o entendimento dos resultados, também será realizada a divisão entre cenários para a descrição dos valores obtidos através das simulações utilizando a memória virtual. Em linhas gerais, a memória virtual é uma técnica, portanto, seu uso objetiva aumentar aquilo que seria a memória de trabalho, i.e., a memória principal, fazendo uso da memória secundária como extensão da memória principal.



(a) Cenário 1: Aumento no número de blocos por página.



(b) *Misses* virtual cenário 1

Fig. 4. Memória Virtual: Cenário 1.

1) Cenário 1: O aumento no número de blocos por página da memória virtual no cenário 1, descrito na Seção IV-B, tem como objetivo explorar a localidade espacial ao mover blocos vizinhos de uma única vez para a memória principal.

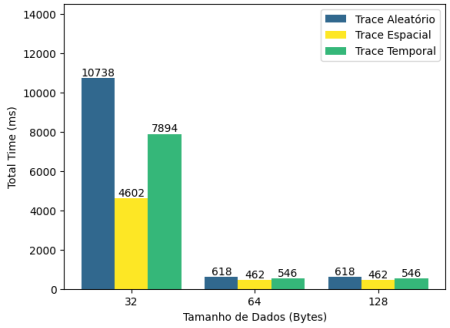
Na execução do *Trace* 1 no cenário 1, conforme é mostrado na Figura 4(a), o melhor tempo total é com 4 blocos por página, o motivo é a exploração da localidade espacial, mesmo os endereços sendo aleatórios, foi possível explorar esta localidade, e também porque com 2 blocos por página ocorreu o maior número de *misses* na TLB e *page fault* na tabela de páginas, conforme Figura 4(b). Mesmo que o número de *misses* na TLB e *page fault* na tabela de páginas serem menores na arquitetura com 8 blocos por página, o tempo para preencher estes 8 blocos da memória secundária para a memória principal fez com que o tempo neste *trace* 1 e arquitetura fique maior.

Já na execução do *Trace* 2 no cenário 1, conforme representado na Figura 4(a), o melhor tempo total neste cenário é na arquitetura com 4 blocos por página. Esse resultado se dá pelo fato de tanto o *trace* como a arquitetura estarem explorando a localidade espacial. Mesmo que com 8 blocos por página também explore a localidade espacial, o número de blocos por página sendo muito grande faz com que haja uma perda de endereços de mapeamento, logo há mais conflitos, tendo *misses* e *page fault*. É exatamente o que ocorre neste cenário com 8 blocos por página, o número de *misses* na TLB e *page fault* na tabela de páginas é maior do que com 4 palavras por bloco, conforme Figura 4(b), aumentando assim muito o tempo total de busca de instruções.

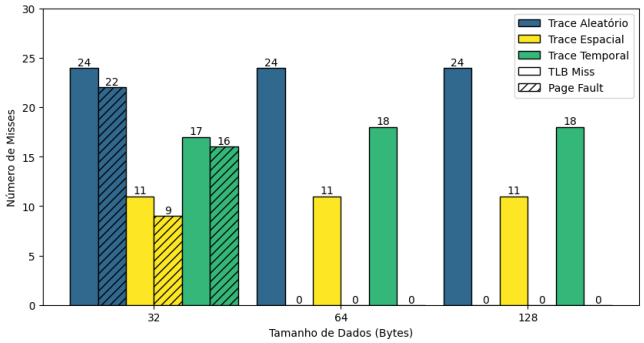
Por fim, na execução do *Trace* 3 no cenário 1, representada na Figura 4(a), o tempo total aumenta com mais blocos por página. O motivo é que como o *trace* não explora a localidade espacial, e há endereços acessados que estão em disco e precisam ir para memória principal, quanto mais blocos por página, mais tempo se leva para ler e escrever os blocos que estavam na RAM em disco e atualizar os endereços na tabela de páginas e TLB. Apesar do *Trace* 3 ter endereços que se repetem e que já estão na memória principal quando buscados novamente, ao ocorrer uma troca de dados com o disco - *page fault* - a penalidade de ir em disco no tempo total é grande, pois a cada acesso ao disco são 100 ms, ou 100 ciclos, sendo que o acesso para escrita utiliza 200 ciclos. É importante considerar que o experimento pode ser elaborado para ter uma proporção pequena de ida à memória secundária, i.e., disco, o que reduziria o impacto dos *page faults*. No entanto, os *traces* são os mesmos ao longo de todos os experimentos apresentados neste artigo.

2) Cenário 2: Já o aumento do tamanho de armazenamento de dados em *Bytes* da memória principal e secundária no cenário 2, descrito na Seção IV-B, tem como objetivo explorar a localidade temporal aumentando a capacidade de armazenamento, e também, possivelmente, o tempo em que o dado ficará na memória principal.

Na execução do *Trace* 1 no cenário 2, representada na Figura 5(a), o tempo total é o maior em todos os tamanhos de memória, pois não explora nenhuma localidade. Na memória de tamanho 32 *Bytes* é o maior tempo total, pois há endereços que precisam ser buscados em disco e possui o maior número de *misses* na TLB e *page fault* na memória principal, conforme Figura 5(b). Com a memória de tamanho 64 *Bytes* o tempo total diminui bastante, e não há nenhum *page fault*, pois todos os endereços buscados estão na memória principal, ou seja, não é necessário ir até o disco buscar dados. Como o mesmo

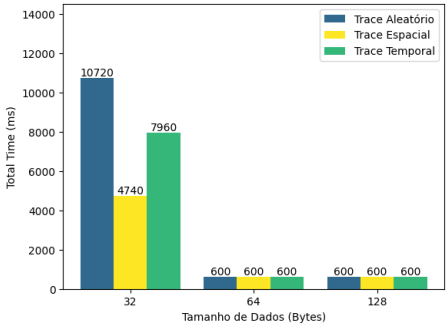


(a) Cenário 2: Aumento do tamanho de dados das memórias.

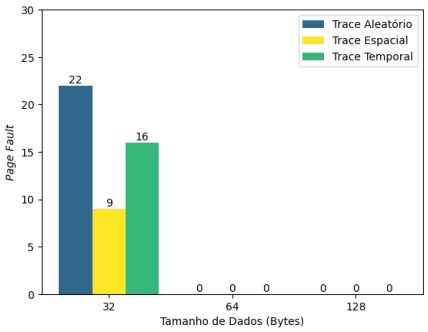


(b) *Misses* virtual cenário 2

Fig. 5. Memória Virtual: Cenário 2.



(a) Cenário 3: Sem TLB.



(b) *Misses* virtual cenário 3

Fig. 6. Memória Virtual: Cenário 3.

Trace 1 foi utilizado em todas as variações e com 64 *Bytes* já foi o suficiente, com 128 *Bytes* a memória está com espaço livre, mantendo assim o mesmo tempo de busca de instruções da arquitetura com memória de 64 *Bytes*.

Já na execução do *Trace* 2, que busca explorar a localidade espacial, no cenário 2, o tempo total é maior na memória que tem tamanho de dados de 32 *Bytes*, pois alguns endereços buscados estão em disco causando *page fault*. Já na memória com tamanho de dados 64 e 128 *Bytes* não é necessário ir a disco buscar nenhum endereço, portanto os dois tempos totais são iguais e há espaço livre na memória principal para armazenar mais dados. É o menor tempo total, visto que possui o menor número de *misses* na TLB e *page fault* na memória principal, conforme Figura 5(b).

Por fim, na execução do *Trace* 3, que tem como objetivo explorar a localidade temporal, assim como o cenário 2, o tempo total é também maior na memória com tamanho de dados de 32 *Bytes*, pois, assim como no *Trace* 2, há endereços que são buscados em disco, ou seja, endereços que causaram *page fault*. Nas memórias de tamanhos de dados 64 e 128 *Bytes* todos os endereços buscados já estão na memória principal, que também possui espaço livre para guardar mais dados. Apesar deste cenário 2 e *Trace* 3 explorar a localidade temporal e o esperado é que o melhor tempo fosse do *Trace* 3, isso não acontece, pois no *Trace* 2 há mais *hits* na TLB, então há menos acessos na memória principal. Como o parâmetro alterado na arquitetura não é relacionado à TLB, apenas a memória principal e secundária, e o *Trace* 2 acessa endereços contínuos que já estão na TLB, seu resultado de tempo total foi melhor por 84 ms.

3) Cenário 3: Por fim, o cenário 3, descrito na Seção IV-B busca simular o aumento do tamanho da memória virtual sem a utilização da TLB, desta forma é possível analisar o desempenho na busca de instruções sem a cache de mapeamentos de endereços virtuais para endereços físicos de memória.

Conforme esperado, na Figura 6(a), a maior parte dos tempos totais sem TLB são maiores que os tempos totais com TLB na Figura 5(a), pois ao buscar os endereços informados pelos *traces*, a memória principal é acessada mais vezes sem TLB. Como o tempo de ciclo da memória principal é 10 ms, e é maior que o tempo de ciclo da TLB, que é 1 ms, logo o tempo total de acesso se torna maior, prejudicando o desempenho na busca de instruções.

O tempo total no cenário 3 com o *Trace* 1, é menor do que no cenário 2, conforme Figura 5(a) e 6(a), pois o número de *misses* na TLB é muito grande, conforme Figura 5(b), fazendo com que o tempo total com TLB seja maior do que sem TLB. Assim como no cenário 2, a memória com tamanho de 64 *Bytes* é suficiente, tendo espaço livre e tempo igual com a memória de 128 *Bytes*.

Tanto na execução do *Trace* 2, quanto na execução do *Trace* 3, representadas na Figura 6(a), o tempo total é menor do que o tempo necessário para a execução do *Trace* 1, pois ambos possuem menos *page fault* na memória principal, conforme Figura 6(b). Porém, o *Trace* 2 faz menos acesso ao disco - gerando um menor *page fault* do que o *Trace* 3, por isso seu tempo total de execução é menor. Todos os tempos totais são iguais a partir da memória de tamanho de dados de 64 *Bytes*, pois todos os endereços acessados estão na memória principal. É possível concluir que não é apenas ter uma memória maior

que irá diminuir o tempo total de busca de instruções, há outras alterações, principalmente ligados a cache, que fazem a diferença para melhorar o desempenho.

VI. Conclusão

Utilizando o objeto de aprendizagem Amnesia, foram feitas três análises em seis cenários distintos, três associados à memória cache, e três para memória virtual, totalizando em dezoito resultados encontrados. Após as análises, os resultados foram satisfatórios e, de modo geral, dentro do esperado pelos autores no início do planejamento. Portanto, o objetivo de aprender sobre hierarquia de memória foi alcançado.

Após a análise dos resultados obtidos com a execução dos *traces* propostos dentro dos cenários da memória cache, quando há aumento na associatividade (Figura 1(a)) e também no tamanho do bloco (Figura 2(a)), há redução nos tempos de execução e *misses*, sendo o *trace* 3, o de melhor desempenho.

Já nos resultados dos cenários na memória virtual, aumentar o tamanho de dados da memória pode ajudar a melhorar o desempenho, porém há outras alterações na arquitetura que em conjunto com o tamanho de dados fará o desempenho ser melhor ainda, como por exemplo, ter uma TLB para mapeamento de endereços virtuais para endereços físicos de memória. Sendo assim, conforme apresentado na Seção V-B, o cenário 2 é o de melhor desempenho na busca de instruções no contexto de memória virtual.

Para trabalhos futuros, é desejável a construção de novas arquiteturas e arquivos de *trace* maiores, com valores diferentes dos propostos no presente trabalho e com instruções que remetam à aplicações reais, proporcionando novos cenários e exploração de outros conceitos, tais como políticas de substituição e coerência de memória. Uma outra análise que pode ser realizada é na relação aumento da associatividade e/ou memória com consequente aumento da latência de acesso, pois neste trabalho a latência foi mantida igual, para não ser incluída mais uma nova variável na análise realizada. Também é desejável a exploração de diferentes simuladores para avaliação do potencial de aprendizagem.

Agradecimentos

Os autores agradecem à FAPEMIG, ao CNPq e a PUC Minas pelo suporte parcial na execução desta pesquisa. Ao professor Carlos Augusto Martins, por ter indicado o uso deste simulador para a disciplina de Arquitetura de Computadores III da PUC Minas e ao grupo de pesquisa do ICMC-USP pelo desenvolvimento do simulador Amnesia.

Referências

[1] Lioris, T., Dimitroulakos, G. and Masselos, K., "XMSIM: EXtensible Memory SIMulator for Early Memory Hierarchy Evaluation," 2010 IEEE Computer Society Annual Symposium on VLSI, Lixouri, Greece, pp. 375-380, 2010. doi: 10.1109/ISVLSI.2010.106.

[2] Binkert, N., et al., "The gem5 simulator." SIGARCH Comput. Archit. News 39, 2 (May), 1–7, 2011. doi: 10.1145/2024716.2024718

[3] Tiosso, F., Bruschi, S. M., Souza, P. S. L., Barbosa, E. F., "Amnesia: um Objeto de Aprendizagem para o Ensino de Hierarquia de Memória," Proceedings of the 25o. Simpósio Brasileiro de Informática na Educação (SBIE 2014), Dourados, Sociedade Brasileira de Computação, 2014. v. 1. p. 1-10.

[4] Agrawal, R., Bandara, S., Ehret, A., Isakov, M., Mark, M. and Kinsy, M. A., "The BRISC-V Platform: A Practical Teaching Approach for Computer Architecture." In Proceedings of the Workshop on Computer Architecture Education (WCAE'19). Association for Computing Machinery, New York, NY, USA, Article 1, 1–8. 2019. doi: 10.1145/3338698.3338891

[5] Petersen, M. B., "Ripes: A Visual Computer Architecture Simulator," ACM/IEEE Workshop on Computer Architecture Education (WCAE), Raleigh, NC, USA, pp. 1-8, 2021. doi: 10.1109/WCAE53984.2021.9707149.

[6] Patterson, D. A., Hennessy, J. L., "Organização e Projeto de Computadores: A Interface Hardware/Software," Elsevier Brasil, 2017.

[7] Welch, T., "Memory Hierarchy Configuration Analysis," in IEEE Transactions on Computers, vol. 27, no. 05, pp. 408-413, 1978. doi: 10.1109/TC.1978.1675120.

[8] Denning, P. J. "The locality principle," Commun. ACM 48, 7 (July 2005), 19–24, doi: 10.1145/1070838.1070856.

[9] Denning, P. J., "Virtual Memory," ACM Comput. Surv. 2, 3 (Sept. 1970), 153–189. doi: 10.1145/356571.356573.

[10] Hill, M. D., and Smith, A. J., "Evaluating associativity in CPU caches," in IEEE Transactions on Computers, vol. 38, no. 12, pp. 1612-1630, Dec. 1989, doi: 10.1109/12.40842.

[11] Tao, J., Schloissnig, S., Karl, W., "Analysis of the Spatial and Temporal Locality in Data Accesses," In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds) Computational Science – ICCS 2006. ICCS 2006. Lecture Notes in Computer Science, vol 3992. Springer, Berlin, Heidelberg. doi: 10.1007/11758525_68.

[12] Vasconcelos, L. B. A., Machado, M. V., Freitas, H. C., "Ambiente para Estudo de Computação Paralela Baseado no Simulador Completo GEM5 e em Algoritmos de Ordenação Escritos com OpenMP." International Journal of Computer Architecture Education, v. 3, p. 1-4, 2014.

[13] Penna, P. H. M. M., Freitas, H. C., "Análise e Avaliação de Simuladores de Sistemas Completos para o Ensino de Arquitetura de Computadores." International Journal of Computer Architecture Education, vol. 2, no 1, p.13-16, 2013.

[14] Alves, M. A. Z., Freitas, H. C., Navaux, P. O. A., "Ensino de arquiteturas de processadores many-core e memórias cache utilizando o simulador Simics." In: Carlos Augusto Paiva da Silva Martins; Philippe Olivier Alexandre Navaux; Rodolfo Jardim de Azevedo; Sérgio Takeo Kofuji. (Org.). Arquitetura de Computadores: educação, ensino e aprendizado. 1ed. Porto Alegre: Sociedade Brasileira de Computação, v. 1, p. 74-110, 2012.

[15] Coutinho, L. M. N., Mendes, J. L. D. and Martins, C. A. P. S., "MSC-Sim -Multilevel and Split Cache Simulator," Proceedings. Frontiers in Education. 36th Annual Conference, San Diego, CA, USA, pp. 7-12, 2006. doi: 10.1109/FIE.2006.322536.

[16] Mendes, J. L. D., Coutinho, L. M. N. and Martins, C. A. P. S., "Web memory hierarchy learning and research environment." In Proceedings of the 2006 workshop on Computer architecture education: held in conjunction with the 33rd International Symposium on Computer Architecture (WCAE '06). Association for Computing Machinery, New York, NY, USA, 5–es. doi: 10.1145/1275620.1275629

[17] Moreno, L., González, E.J., Popescu, B., Toledo, J., Torres, J. and Gonzalez, C., "MNEME: A memory hierarchy simulator for an engineering computer architecture course." Comput. Appl. Eng. Educ., 19: 358-364, 2011. doi: 10.1002/cae.20317

[18] Freitas, L. F. S., Ancioto, A. S. R., de Fátima Rodrigues Guimarães, R., Martins, V. F., Dias, D. R. C., de Paiva Guimarães, M., "A Virtual Reality Simulator to Assist in Memory Management Lectures." Computational Science and Its Applications. ICCSA 2020. Lecture Notes in Computer Science, vol 12255. 2020. Springer, Cham. doi: 10.1007/978-3-030-58820-5_58