UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Análise do Impacto de Diferentes Versionamentos de Dados das Memórias Transacionais sobre Memórias *Phase-Change*

Felipe Leivas Teixeira

Pelotas, 2016

Felipe Leivas Teixeira

Análise do Impacto de Diferentes Versionamentos de Dados das Memórias Transacionais sobre Memórias *Phase-Change*

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Computação

Orientador: Prof. Dr. Maurício Lima Pilla
Coorientador: Prof. Dr. André Rauber Du Bois

Pelotas, 2016

Dados Internacionais de Catalogação na Publicação (CIP)

T266a Teixeira, Felipe Leivas
Análise do impacto de diferentes versionamentos de dados das memórias transacionais sobre memórias *Phase-Change* / Felipe Leivas Teixeira ; Maurício Lima Pilla, orientador; André Rauber Du Bois, coorientador. - Pelotas, 2016.
116 f.

Dissertação (Mestrado) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2016.

1. Memórias transacionais. 2. Memórias *Phase-Change*. 3. Processamento paralelo. 4. Hierarquia de memória. I. Pilla, Maurício Lima, orient. II. Du Bois, André Rauber, coorient. III. Título.

CDD: 005

Catalogação na Fonte: Aline Herbstrith Batista CRB 10/ 1737
Biblioteca Campus Porto - UFPel

Dedico este trabalho a todos aqueles que me ajudaram em mais essa conquista.

“Do. Or do not. There is no try.”
— Yoda

RESUMO

TEIXEIRA, Felipe Leivas. **Análise do Impacto de Diferentes Versionamentos de Dados das Memórias Transacionais sobre Memórias *Phase-Change***. 2016. 116 f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2016.

Dois problemas dos grandes sistemas computacionais atualmente estão relacionados com o consumo de energia e a programação concorrente correta que aproveite os recursos disponibilizados. Das várias tecnologias para resolver esses problemas, destacam-se a *Phase-Change Memory* e as memórias transacionais.

A *Phase-Change Memory* (PCM) é uma nova tecnologia que está sendo estudada para substituir as DRAMs, como memória principal, em grandes *data centers*, devido a sua não volatilidade que reduz o consumo estático de potência. O principal problema da PCM está em suas escritas, que são lentas e degradam o seu material, diminuindo assim sua vida útil.

Memórias transacionais são um método de sincronização de *threads* desenvolvido para diminuir as dificuldades e limitações de métodos baseados em *locks*. Suas principais vantagens são relacionadas a ser um método de alto nível, mais fácil de programar e que permite a composição e reúso de código com mais facilidade. Outra vantagem das memórias transacionais em comparação com *locks* é a inexistência do problema de *deadlock*. Memórias transacionais são baseadas nas transações de banco de dados. As transações em sistemas de banco de dados satisfazem quatro propriedades: atomicidade, consistência, isolamento e durabilidade, ou ACID. As transações das memórias transacionais também devem garantir as propriedades ACID, exceto a durabilidade. Para garantir a atomicidade, as memórias transacionais implementam vários mecanismos de versionamento de dados para fazer o gerenciamento dos dados.

Desta forma, o objetivo deste trabalho é analisar o impacto em PCMs das diferentes implementações de versionamento de dados em STMs. Para tanto, foi implementado o *Phase-Change Memory - Multicore Simulator* (PCM-MS), um simulador de hierarquia de memória para arquiteturas de múltiplos núcleos onde a PCM é a memória principal. O PCM-MS faz a simulação dos acessos e determina os bits alterados na PCM para estimar o desgaste e o consumo de energia da PCM. Além do PCM-MS, a ferramenta Pintools foi utilizada para gerar arquivos de traço que são executados no simulador. Como biblioteca de STM foi utilizada a TinySTM, pois implementa diversos versionamentos e constitui parte do estado da arte de STM. Como *benchmarks*, foram utilizados o Eigenbench e o conjunto de *benchmarks* STAMP.

Os resultados mostram que o versionamento WBC apresentou o menor desgaste na PCM em 3 dos 7 *benchmarks* analisados. Esses resultados estão ligados ao número de *aborts* dos versionamentos, onde o WBC apresenta um número de *aborts* muito menor que os outros, sendo até 39 vezes menor no experimento com o *benchmark* Kmeans com 64 *threads*. Em trabalhos futuros, pretende-se continuar o desenvolvimento do simulador, além de fazer a análise do desgaste na PCM de outros sistemas transacionais.

Palavras-chave: Memórias Transacionais, *Phase-Change Memory*, Processamento Paralelo, Hierarquias de Memória.

ABSTRACT

TEIXEIRA, Felipe Leivas. **Impact Analysis of Different Version Management of Transactional Memory on Phase-Change Memories**. 2016. 116 f. Dissertação (Mestrado em Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2016.

Two of the major issues in current large computer systems are energy consumption and how to explore concurrent systems in a correct and efficient way. Phase-Change Memories and Transactional Memories are two technologies that intend to solve these issues.

Phase-Change Memory (PCM) is a new memory technology being studied to replace DRAMs as the main memory in large data centers, as its non-volatility reduces static power consumption. The main problem of PCMs consists in its write operations, which are slow and generate degradation in material, thus reducing its life.

Transactional memories are synchronization methods developed to reduce the difficulties and limitations of lock-based methods. Their main advantages are related to being high-level and allowing composition and reuse of code. Another advantage of transactional memories compared to locks is the absence of deadlocks. Transactional memories are based on database transactions. Transactions in database systems meet four properties: atomicity, consistency, isolation and durability, or ACID. Transactional memories must also implement the ACID properties, except for durability. Transactional memories implement version management of data to ensure atomicity.

The objective of this study is to analyze the impact on the PCM of different version management techniques implemented by STMs. To that end, the Phase-Change Memory - Multicore Simulator (PCM-MS) was implemented, a memory hierarchy simulator for multi-core systems where the PCM is the main memory. It determines changed bits in PCM to estimate the wear and energy consumption. In addition to the PCM-MS, Pintools was used to generate trace files that run in the simulator. As the STM library, TinySTM was chosen because it implement various version management and it represents the state-of-art in STM. As benchmarks, Eigenbench and the STAMP set of benchmarks were used.

The results showed that the WBC VM had the lowest wear on the PCM in 3 of 7 benchmarks analyzed. These results are related to the number of aborts of VMs, where the WBC presents a much smaller number of aborts than others VM, being up to 39 times lower in the experiment with the benchmark Kmeans with 64 threads. In future works, we intend to enhance the simulator and make the impact analysis in PCM of others transactional systems.

Keywords: Software Transactional Memory, Phase-Change Memory.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
DRAM	*Dynamic Random Access Memory*
HTM	*Hardware Transactional Memory*
PCM	*Phase-Change Memory*
PMD	*Personal Mobile Devices*
RAM	*Random Access Memory*
RSTM	*Rochester Software Transactional Memory*
SSCA2	*Scalable Synthetic Compact Applications 2*
STAMP	Stanford *Transactional Applications for Multi-Processing*
STM	*Software Transactional Memory*
TL2	*Transactional Locking* 2
TM	*Transactional Memory*
VM	*Vesion Management*
WBC	*Write Back Commit-time*
WBE	*Write Back Encounter-time*
WT	*Write Through*

SUMÁRIO

1 INTRODUÇÃO

A eficiência energética é crucial para o atual paradigma computacional, onde dispositivos pessoais móveis ou *personal mobile devices* (PMD) servem como clientes para acesso à computação em nuvem (PATTERSON; HENNESSY, 2013). Grandes *data centers* utilizam uma hierarquia onde a memória principal é implementada em tecnologia DRAM. Memórias tradicionais (DRAM) representam entre 20% e 40% do consumo total de um servidor (XIA et al., 2015). Uma alternativa atraente para grandes *data centers* é a tecnologia PCM que surgiu como uma alternativa para o consumo proibitivo de energia em memórias tradicionais (LEE et al., 2010). A não volatilidade deste tipo de memória reduz o consumo estático de potência. Entretanto, as escritas são lentas, pois o processo de armazenamento de um bit altera o estado do material da célula de memória em questão (LEE et al., 2010). Além disso, escritas degradam o material, limitando a vida útil deste tipo de memória.

A programação concorrente é uma área que vem ganhando espaço e importância na computação, devido aos computadores, que contam com múltiplos processadores. Isso favorece a programação concorrente, visto que ela pode explorar estes múltiplos processadores de forma a melhorar o desempenho de um programa. Porém, um dos problemas da programação concorrente é a condição de corrida (SILBERSCHATZ, 2010), que consiste na situação em que várias *threads* ou processos acessam e manipulam os mesmos dados concorrentemente e na qual o resultado da execução depende da ordem específica em que o acesso ocorre. A parte do programa que contém os dados manipulados concorrentemente é chamada de seção crítica (SILBERSCHATZ, 2010). Para que não ocorra a condição de corrida, podem ser utilizados diferentes métodos de controle de acesso às mesmas. A sincronização é feita para que ocorra a exclusão mútua, que consiste no princípio que se uma *thread* está manipulando os dados compartilhados dentro de uma seção crítica, nenhuma outra *thread* pode acessar a mesma seção crítica. Existem vários métodos que fazem a sincronização, entre eles as memórias transacionais.

As memórias transacionais (HERLIHY; ELIOT; MOSS, 1993) ou *transactional memories* (TM) são uma alternativa para sincronização de *threads*. Memórias transa-

cionais são baseadas em transações de banco de dados. A principal vantagem em relação às alternativas convencionais (baseadas em *locks*) é a maior simplicidade ao escrever o código. Outra vantagem das memórias transacionais em comparação com *locks* é a inexistência do problema de *deadlock*. A composição de componentes de software também é mais simples, aumentando a reusabilidade do código. Por essas e outras vantagens, memórias transacionais são uma alternativa promissora para a escrita de código portável e escalável em memória compartilhada (MORESHET; BAHAR; HERLIHY, 2006). Primeiramente foram pensadas para serem desenvolvidas em hardware (HTM), mas acabaram ganhando mais popularidade quando desenvolvidas em software (STM).

Como descrito anteriormente, memórias transacionais são baseadas nas transações de banco de dados. As transações em sistemas de banco de dados satisfazem quatro propriedades: atomicidade, consistência, isolamento e durabilidade. Essas propriedades podem ser resumidas na sigla ACID. As transações das memórias transacionais também devem garantir essas propriedades, exceto a durabilidade visto que não é necessário manter os dados por definitivo na memória. Para garantir a atomicidade, as memórias transacionais implementam versionamento de dados para fazer o gerenciamento das versões dos dados. Existem dois tipos de versionamento de dados: Versionamento Adiantado e Versionamento Atrasado. Esses dois tipos de versionamento se diferenciam na questão de onde estará o valor mais atualizado do dado durante a transação, esse valor poderá estar na memória (Versionamento Adiantado) ou em um *buffer* auxiliar (Versionamento Atrasado).

1.1 Motivação

Tanto memórias transacionais quanto as memórias PCM são alternativas promissoras em suas áreas, programação concorrente (memórias transacionais) e memória principal (memória PCM). As memórias transacionais são promissoras devido sua maior simplicidade ao escrever um código que seja portável e escalável em memória compartilhada. Já a memória PCM é uma alternativa promissora devido a sua não volatilidade, que reduz o consumo estático de potência, fazendo da memória PCM uma alternativa para o consumo proibitivo de energia em memórias tradicionais.

Então, como memórias transacionais e memórias PCM são alternativas promissoras, analisar o impacto das diferentes implementações de versionamento de dados das memórias transacionais (que são os responsáveis pelo gerenciamento das versões dos dados na memória) e ver qual prejudicaria menos a memória PCM será importante para prolongar a vida útil da PCM.

1.2 Objetivos

O objetivo principal deste trabalho é **analisar o impacto das diferentes implementações de versionamento de dados das memórias transacionais em software nas memórias PCM**.

Os objetivos específicos são:

1. Caracterizar os diferentes versionamentos de dados das STM;
2. Analisar o desgaste dos diferentes versionamentos de dados em uma memória PCM;
3. Analisar e caracterizar o consumo de energia dos diferentes versionamentos de dados em memórias PCM; e
4. Apontar qual é a melhor opção de versionamento de dados para ser executado em uma memória PCM.

1.3 Metodologia

Esta seção descreve os passos metodológicos que foram realizados para o desenvolvimento deste trabalho.

1.3.1 Estudo e caracterização dos versionamentos de dados implementados em STM

Primeiramente para o desenvolvimento deste trabalho foi necessário estudar, analisar e caracterizar os diferentes versionamentos de dados. Então foi feito um estudo sobre o comportamento de cada versionamento e uma caraterização dos mesmos por meio da execução de experimentos com a biblioteca de STM TinySTM e com o *benchmark* Eigenbench (HONG et al., 2010), que é um *microbenchmark* que faz uma avaliação ortogonal das características de aplicações que formam a base para o comportamento transacional. Ele foi útil para compreensão do comportamento dos diferentes versionamentos. O Eigenbench em seu site oficial não tem uma versão para a TinySTM, então foi feita a portabilidade do Eigenbench para a TinySTM.

1.3.2 Implementação do simulador

Para fazer a simulação foi implementado um simulador de hierarquia de memória, com a memória principal sendo a memória PCM. O simulador recebe arquivos de traço com os acessos à memória e faz a simulação. Uma característica deste simulador é que ele pode ser utilizado por programas *multithread*, onde cada *thread* terá um arquivo de traço. Para gerar os arquivos de traço foi utilizada a ferramenta Pintools (LUK et al., 2005). O simulador será descrito no Capítulo 3.

1.3.3 Comparação dos diferentes versionamentos de dados em relação ao impacto em uma memória PCM

Por fim para fazer a comparação dos diferentes versionamentos de dados, foram feitas execuções com o simulador, onde para cada conjunto de testes (*benchmark*-versionamento), foram feitas 30 execuções e calculada a média aritmética dos resultados. Para analisar o desgaste em uma memória PCM, foi analisado o número de bits alterados na memória principal (memória PCM). Também foi feito um cálculo do consumo de energia da memória PCM nos diferentes experimentos.

1.4 Estrutura do Texto

O trabalho que segue está organizado da seguinte forma: o Capítulo 2 apresenta os conceitos e o funcionamento da PCM, das memórias transacionais e da biblioteca de STM TinySTM. No Capítulo 3 é apresentado o ambiente de avaliação do trabalho. O Capítulo 4 apresenta a caracterização dos versionamento de dados. O Capítulo 5 discute os resultados da caracterização dos acessos à memória e o impacto na PCM. E por fim, o Capítulo 6 apresenta as considerações finais e os trabalhos futuros.

2 *BACKGROUND*

2.1 Phase-Change Memory

A *Phase-Change Memory* (PCM) é uma memória não-volátil que surgiu como uma alternativa para o consumo proibitivo de energia em memórias tradicionais, para substituir em grande parte as memórias DRAM (FERREIRA et al., 2010). Isto se dá pelo seu custo-benefício e a sua eficiência em relação ao consumo de energia. A PCM utiliza-se das fases de seu material para armazenar dados. A temperatura é a responsável por induzir a mudança de fase do material.

2.1.1 Escritas

As escritas, na PCM, são custosas, pois elas são aproximadamente 5 a 10 vezes mais lentas e consomem 10 vezes mais energia que as leituras (RAOUX et al., 2008). Elas são feitas da seguinte forma: um pulso de corrente passa pelo material, fazendo com que ele fique em um estado amorfo ou em um estado cristalino (LEE et al., 2010). Cada uma das estruturas tem resistências elétricas diferentes sendo possível, assim, armazenar com precisão o valor de um bit. Segue a descrição de como ocorre um *Reset* e um *Set* na memória PCM.

- **Reset**: para que ocorra o *Reset* é induzido um alto e curto pulso de corrente, que é interrompido abruptamente, fazendo com que a resistividade do material aumente. Ao extinguir rapidamente a geração de calor, o material torna-se amorfo.
- **Set**: para que ocorra o *Set*, é induzido um pulso moderado e longo de corrente, o que faz com que a resistividade do material seja reduzida, assim fazendo com que o material esfrie gradualmente e mude para um estado cristalino.

2.1.2 Leituras

No início de uma leitura, o *bitline* é carregado com a voltagem correspondente. Se a célula selecionada da PCM está em um estado cristalino, o *bitline* é descarregado com a corrente fluindo através do elemento de armazenamento e acessando o transistor. Se a célula está em um estado amorfo, a corrente do *bitline* é impedida ou

limitada (QURESHI; SRINIVASAN; RIVERS, 2009). Com isso é possível diferenciar os dois estados, e assim diferenciar o bit 0 do bit 1.

2.1.3 Limitações da PCM

O principal problema da PCM são suas escritas que, além de serem lentas, também desgastam o material (LEE et al., 2010). Como a escrita desgasta o material, o tempo de vida da PCM é ordens de grandeza menor que outras alternativas de armazenamento. A Figura 1 compara o *Reset*, o *Set* e a leitura (*Read*) em relação ao tempo e a corrente. Como pode ser visto o *Reset* requer um pulso de corrente mais alto, então é ele quem determina a potência da escrita. Já o tempo de escrita é dado pelo *Set* (WANG; WU, 2009), que tem uma corrente menor mas com um tempo mais prolongado. O tempo e a corrente das leituras são muito menores que o tempo e a corrente do *Reset* e do *Set*.

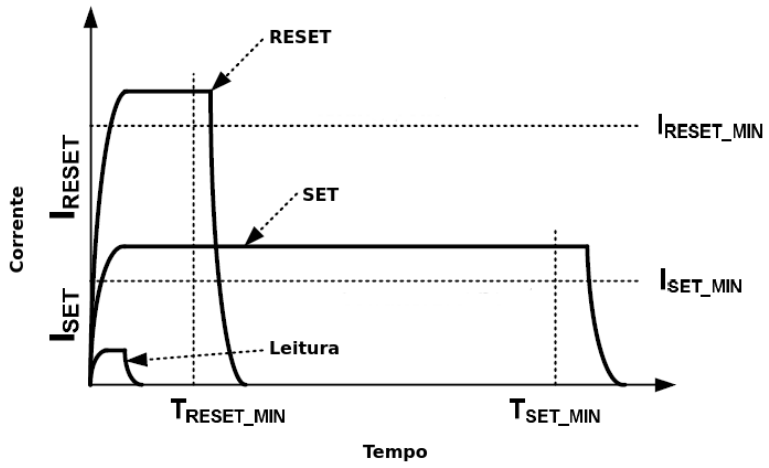


Figura 1: Comparação entre o *Reset*, o *Set* e a leitura (*Read*). Fonte: (WANG; WU, 2009).

2.1.4 Sistemas de Memória com PCM

Segundo Xia et al. (2015), existem três alternativas de sistema de memória onde a PCM é a memória principal. Essas alternativas podem ser vistas na Figura 2. A primeira alternativa é substituir a DRAM pela PCM, como mostra a Figura 2a. A segunda alternativa é utilizar a PCM e a DRAM em paralelo, em uma arquitetura híbrida, como mostra a Figura 2b. E por fim a terceira alternativa também é uma alternativa híbrida, com PCM e DRAM em paralelo, mas nesse caso a DRAM é utilizada como *cache* ou *buffer* da PCM, como mostra a Figura 2c. As alternativas de memória principal híbridas permitem que as aplicações explorem as vantagens tanto da PCM como da DRAM, sendo então as mais estudadas.

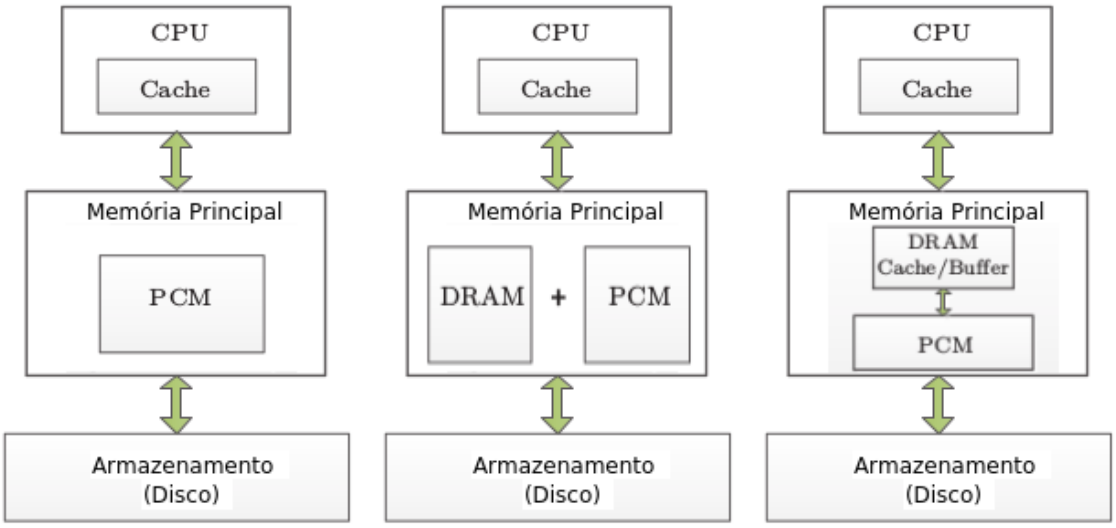


Figura 2: Alternativas de sistemas de memória com memória PCM. Fonte: (XIA et al., 2015)

2.2 Memórias Transacionais

Memórias transacionais (HERLIHY; ELIOT; MOSS, 1993) são uma alternativa para sincronização de *threads*. TMs são baseadas em transações de banco de dados e fornecem uma execução atômica e isolada de alterações em um conjunto de dados compartilhados. As vantagens das TMs são em relação a simplicidade de escrita de código e a inexistência de *deadlocks* (MORESHET; BAHAR; HERLIHY, 2006).

Primeiramente, TMs foram pensadas para serem desenvolvidas em *hardware* (HTM), mas acabaram ganhando mais popularidade quando desenvolvidas em *software* (STM). Grandes empresas como Intel e IBM acabaram investindo em memórias transacionais suportadas em *hardware* (HAMMARLUND et al., 2014; SHUM; BUSABA; JACOBI, 2013; LE et al., 2015; CLICK, 2009; DICE et al., 2009). TMs também podem ser implementadas em uma versão híbrida. Este trabalho aborda implementações de TMs em *software*.

Na programação utilizando STMs, todo o acesso à memória compartilhada é realizado dentro de transações e todas as transações são executadas atomicamente em relação a transações concorrentes.

2.2.1 Propriedades

Transação é uma sequência finita de escritas e leituras na memória, executada por uma *thread* (HERLIHY; ELIOT; MOSS, 1993), e que deve satisfazer três propriedades:

- **Atomicidade**: cada transação faz uma sequência de mudanças provisórias na memória compartilhada. Quando a transação é concluída, pode ocorrer um *com-*

mit, tornando suas mudanças visíveis a outras *threads* instantaneamente, ou pode ocorrer um *abort*, fazendo com que suas alterações sejam descartadas.

- **Consistência**: transações devem garantir que um sistema que era consistente deve ser mantido consistente. Essa é a propriedade relacionada com o conceito de invariância.

- **Isolamento**: transações não interferem na execução de outras transações, assim parecendo que elas são executadas serialmente. Uma transação não observa o estado intermediário de outra.

2.2.2 Versionamento de Dados

O versionamento de dados faz o gerenciamento das versões dos dados. Ele armazena tanto o valor do dado no início de uma transação como também o valor do dado modificado durante a transação, isso para garantir a propriedade de atomicidade (BALDASSIN, 2009).

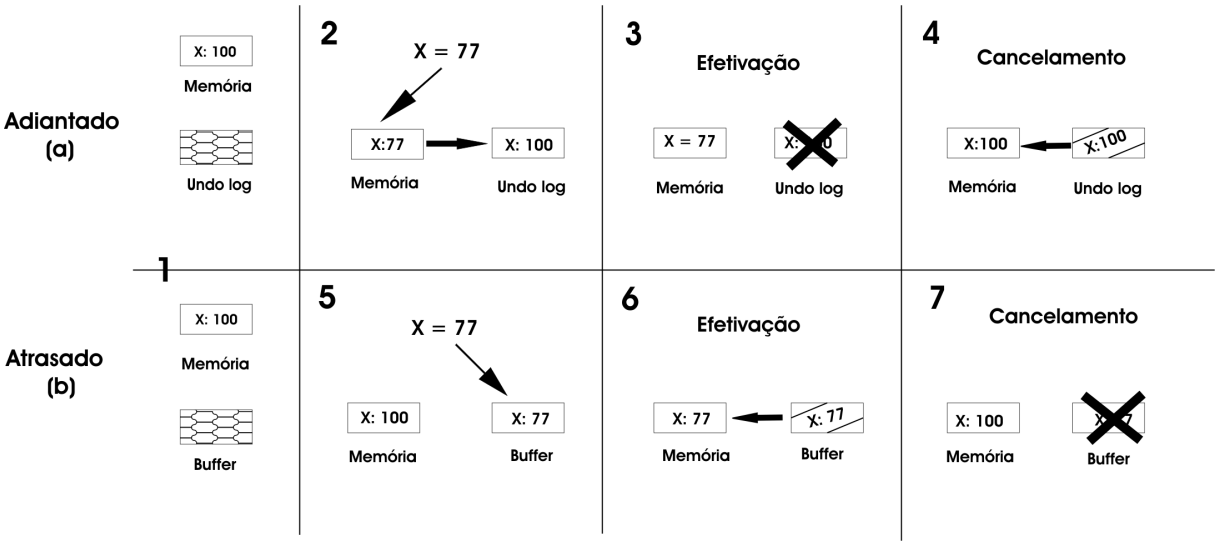


Figura 3: Exemplo de versionamento adiantado (a) e atrasado (b). Fonte: (BALDASSIN, 2009)

Existem dois tipos de versionamento de dados, são eles:

- **Versionamento Adiantado**: como pode ser visto na Figura 3 (a), o valor modificado durante a transação é armazenado direto na memória e o valor inicial é armazenado em um *undo log*, para que no caso de cancelamento na transação o valor inicial seja restaurado na memória.

- **Versionamento Atrasado**: como pode ser visto na Figura 3 (b) neste versionamento o valor modificado durante a transação é armazenado em um *buffer* e o valor inicial é mantido na memória até que aconteça um *commit* na transação,

onde o valor armazenado no *buffer* é escrito na memória. Caso aconteça o cancelamento na transação, o valor do *buffer* é descartado.

O versionamento de dados em memórias transacionais desenvolvidas em software, que é o foco deste trabalho, necessitam de dois conceitos, a aquisição de escrita e o versionamento. A aquisição de escrita pode ser de dois tipos, bloqueante ou não bloqueante (MARATHE; MOIR, 2007). A biblioteca utilizada neste trabalho, a TinySTM, implementa a aquisição de escrita bloqueante. No caso da *TinySTM*, o privilégio de escrita é a aquisição do *lock* referente ao endereço que será escrito. Na aquisição de escrita bloqueante, a transação pode adquirir o privilégio de escrita em dois momentos: em seguida que ocorre uma escrita ou no final da transação. Já o versionamento descreve onde será feita esta escrita, diretamente na memória (versionamento adiantado) ou primeiramente em um *buffer* (versionamento atrasado). Com isso, STM podem ter até três tipos de versionamentos:

- Com aquisição do privilégio no momento que ocorre uma escrita e versionamento adiantado;
- Com aquisição do privilégio no momento que ocorre uma escrita e versionamento atrasado; e
- Com aquisição do privilégio no final de uma transação e versionamento atrasado.

Não existe a possibilidade de ter a aquisição do privilégio no final de uma transação e versionamento adiantado, devido às transações terem que manter o controle das escritas para garantir a consistência do sistema.

2.2.3 Detecção de Conflito

Mecanismos de detecção de conflitos verificam a existência de operações conflitantes durante uma transação. Um conflito ocorre quando duas transações estão acessando um mesmo dado na memória e pelo menos uma das transações está fazendo uma operação de escrita (BALDASSIN, 2009).

Da mesma forma que o versionamento de dados, a detecção de conflito também pode ser de dois tipos:

- **Detecção de Conflitos Adiantada**: este tipo de detecção ocorre no momento que duas transações acessam um mesmo dado e uma delas faz uma operação de escrita. Essa operação de escrita é detectada e então uma transação é abortada. Neste tipo de detecção pode ocorrer um problema chamado de *livelock*, quando duas transações ficam se cancelando. Desta forma, a execução do programa não progride. A Figura 4 mostra como é feita a detecção de conflitos adiantada.

O Caso 1 mostra a execução sem conflitos, no qual as duas transações são executadas sem problemas. Já no Caso 2, mostra-se o que acontece quando ocorre um conflito, no caso T1 lê A e logo depois T2 escreve em A, então o conflito é detectado e T1 é abortada. Logo depois de ser efetivada T2, a transação T1 consegue ler A sem problema de conflito. Por fim, o Caso 3 mostra a situação de *livelock*, onde as duas transações tentam ler e escrever em A mas ambas acabam sempre abortando.

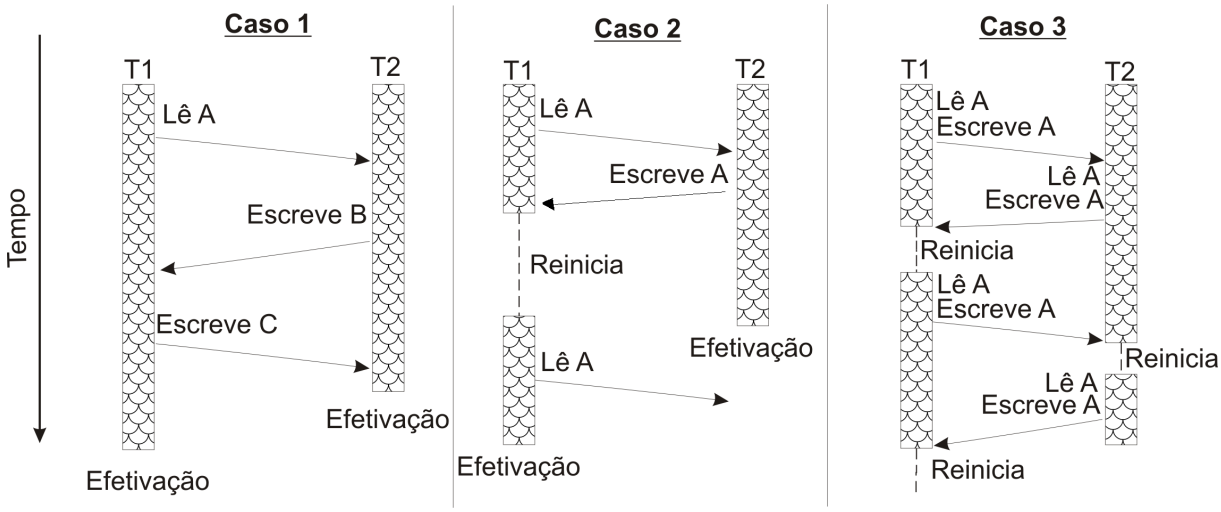


Figura 4: Detecção de conflitos em modo adiantado. Fonte: (RIGO; CENTODUCATTE; BALDASSIN, 2007)

- **Detecção de Conflitos Atrasada**: Este tipo de detecção de conflito ocorre no final da transação. Antes da transação ser completada, verifica-se se ocorreu um conflito. Caso tenha ocorrido, a transação é cancelada, senão é efetivada. Para transações muito grandes não é recomendado este tipo de detecção, pois uma transação grande pode ser abortada várias vezes por transações pequenas, assim gastando tempo de processamento desnecessário, este problema se chama *starvation*. A Figura 5 mostra como é feita a detecção de conflitos atrasada.

  No Caso 1 mostra-se as transações acessando dados diferentes, não ocasionando conflitos. No Caso 2, T2 lê A que é escrita por T1. A T2 só nota o conflito quando T1 é efetivado. Logo depois de notar o conflito T2 é abortada. No Caso 3 não ocorre nenhum conflito, pois T1 lê A antes de T2 escrever. O Caso 4 mostra a situação em que, após ser cancelada, T1 volta a executar.

Para solucionar o problema de qual transação continuará executando, quando ocorre um conflito, é utilizado um gerenciador de contenção (HARRIS; LARUS; RAJWAR, 2010). O gerenciador de contenção é o responsável por decidir quando e qual transação vai ser abortada, isso para garantir que a execução do programa prossiga sem problemas.

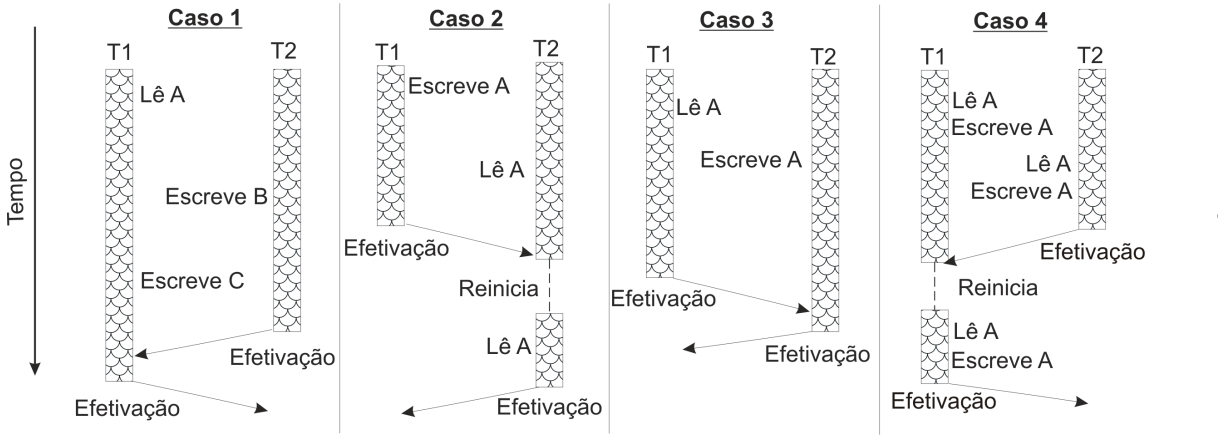


Figura 5: Detecção de conflitos em modo atrasado. Fonte: (RIGO; CENTODUCATTE; BALDASSIN, 2007)

2.3 TinySTM

A *TinySTM* (FELBER; FETZER; RIEGEL, 2008) é uma implementação de STM para as linguagens C e C++. Seu algoritmo é baseado em outros algoritmos de STM como o TL2 (DICE; SHALEV; SHAVIT, 2006). Ela é uma biblioteca utilizada para escrever aplicativos que usam memórias transacionais para sincronização, em substituição aos tradicionais *locks*.

2.3.1 Sincronização e Versionamento

Na *TinySTM* a sincronização é feita a partir de um *array* de *locks* compartilhado que gerencia o acesso concorrente à memória. Cada *lock* bloqueia vários endereços de memória, como pode ser visto na Figura 6. O mapeamento é feito por meio de uma função *hash*.

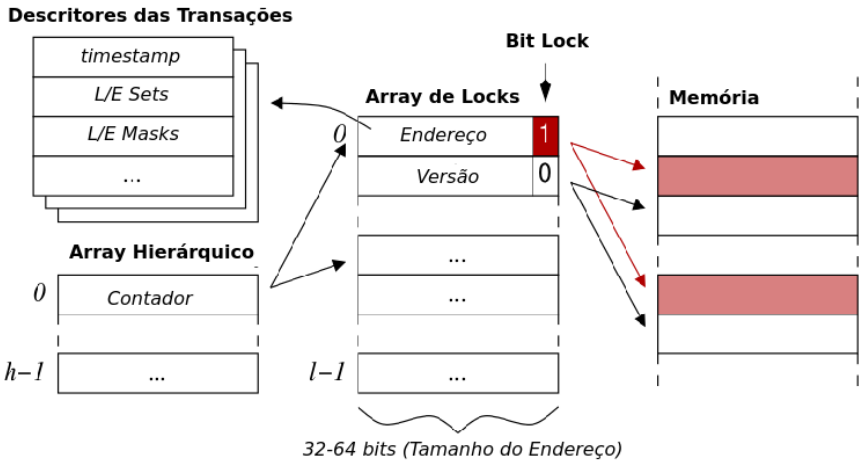


Figura 6: Como é feita a sincronização na *tinySTM*. Fonte: (FELBER; FETZER; RIEGEL, 2008)

A *TinySTM* apresenta três diferentes estratégias de versionamento, são elas:

- ***WRITE_BACK_ETL***: esta estratégia implementa o versionamento atrasado (*write-back*) com *encounter-time locking*, onde o *lock* é adquirido quando ocorre uma operação de escrita. O valor somente é escrito na memória no momento do *commit* da transação.
- ***WRITE_BACK_CTL***: esta estratégia implementa o versionamento atrasado (*write-back*) com *commit-time locking*, onde o *lock* é adquirido no momento da efetivação da transação (*commit*). Assim como o *WRITE_BACK_ETL*, o valor somente é escrito na memória no momento do *commit* da transação.
- ***WRITE_THROUGH***: esta estratégia implementa o versionamento adiantado (*write-through*) com *encounter-time locking*. A atualização é realizada diretamente na memória e um *undo log* é mantido para o caso de *abort* na transação.

A estratégia de versionamento padrão utilizada pela *TinySTM* é a *WRITE_BACK_ETL*.

2.3.2 Escritas

Nos versionamentos com aquisição do *lock* no momento que ocorre uma escrita (*WRITE_BACK_ETL* e *WRITE_THROUGH*), a transação primeiro identifica o *lock* correspondente ao endereço de memória e atomicamente lê o valor da memória. Se o *lock* está em uso a transação verifica se é a proprietária do *lock*. Caso positivo, então ela simplesmente escreve o novo valor, em um *buffer* (caso seja o versionamento *WRITE_BACK_ETL*) ou direto na memória (caso seja o versionamento *WRITE_THROUGH*), e retorna. Caso contrário, a transação pode esperar por algum tempo ou abortar imediatamente. A *TinySTM* utiliza a última opção em sua implementação.

Se o *lock* não está em uso, a transação tenta adquiri-lo para escrever o novo valor na entrada utilizando uma operação atômica *compare-and-swap*. A falha indica que outra transação adquiriu o *lock* nesse meio tempo, então a transação é reiniciada.

Para o versionamento com aquisição do *lock* no final da transação (*WRITE_BACK_CTL*), a transação primeiramente verifica se essa é a primeira escrita, que ela efetua, no endereço. Caso positivo, é alocada uma posição no *buffer* para se efetivada a escrita. Caso contrário, a posição do *buffer* correspondente ao endereço da escrita é atualizada.

2.3.3 Leituras

Quando ocorre uma leitura na memória, a transação deve verificar se o *lock* está em uso ou se o valor já foi atualizado concorrentemente por outra transação. Para esse

fim, a transação lê o *lock* correspondente ao endereço de memória. Se o *lock* não tem proprietário e o valor (número de versão) não foi modificado entre duas leituras, então o valor é consistente.

2.3.4 Gerenciamento de Memória

A *TinySTM* utiliza um gerenciador de memória que possibilita qualquer código transacional utilizar memória dinâmica. As transações mantém o endereço da memória alocada ou liberada. A alocação de memória é automaticamente desfeita quando a transação é abortada. Já a liberação não pode ser desfeita antes do *commit*. Contudo, uma transação pode somente liberar a memória depois de adquirir todos os *locks*. Um *free* é semanticamente equivalente a uma atualização.

2.3.5 Gerenciador de Contenção

A *TinySTM* implementa quatro estratégias de gerenciamento de contenção, são elas:

- ***CM_SUICIDE***: nesta estratégia a transação que detecta o conflito é abortada imediatamente.
- ***CM_DELAY***: parecido com a estratégia *CM_SUICIDE*, mas espera até a transação, que está em posse do *lock*, tenha liberado o *lock* antes de reiniciar a transação. Isso porque a transação que foi abortada irá provavelmente tentar adquirir o mesmo *lock* novamente e talvez falhe mais de uma vez caso o *lock* não tenha sido liberado. Além disso, essa estratégia aumenta as chances de que a transação tenha sucesso sem precisar abortar muitas vezes, o que melhora o tempo de execução do processador.
- ***CM_BACKOFF***: também parecida com a estratégia *CM_SUICIDE*, porém espera por um tempo, randômico, para reiniciar a transação. A duração deste atraso é escolhido uniformemente ao acaso em um intervalo cujo tamanho aumenta exponencialmente a cada reinicialização.
- ***CM_MODULAR***: esta estratégia implementa vários gerenciadores de contenção, que são alternados durante a execução. Os gerenciadores utilizados são:
  - *SUICIDE*: a transação que descobriu o conflito é abortada.
  - *AGGRESSIVE*: a transação que é abortada é a outra, e não a que descobriu o conflito.
  - *DELAY*: a mesma coisa que a *SUICIDE* mas espera pela resolução do conflito antes de reiniciar a transação.
  - *TIMESTAMP*: a transação mais nova é abortada.

O gerenciador de contenção utilizado neste trabalho foi *CM_SUICIDE*, que é a estratégia de gerenciamento de contenção padrão da *TinySTM*.

2.4 Trabalhos Relacionados

Esta Seção apresenta trabalhos que levam em consideração o aumento de tempo útil de uma memória PCM e a diminuição do seu desgaste.

2.4.1 *Analyzing the Impact of Useless Write-Backs on the Endurance and Energy Consumption of PCM Main Memory*

Bock et al. (2011) abordam a técnica *Useless Write-Backs* que consiste em não escrever na PCM dados que não serão mais utilizados na execução do programa, assim evitando fazer escritas desnecessárias na PCM. Eles apresentam o impacto que os *write-backs* inúteis tem sobre a durabilidade e consumo de energia dos sistemas baseados em memória principal PCM e implementa técnicas que evitam estes *write-backs*.

O trabalho apresenta um *framework* que mede o número de *write-backs* inúteis na PCM para três diferentes tipos de regiões de memória e apresenta um modelo energético para determinar o máximo de economia de energia que poderia ser alcançados através de um regime deste tipo. São algoritmos diferentes para cada região da memória, pois para cada uma delas, analisar se uma posição de memória não será mais usada é diferente.

O *framework* implementado funciona da seguinte forma, primeiramente o programa passa por uma instrumentação que gera um arquivo de traço com os endereços e tipos de cada referência de memória, assim como outras informações exigidas pelo tipo específico de região de memória que está sendo analisado, em seguida o arquivo de traço passa por um analisador que consiste de um simulador de *cache*, rotinas de análise e estruturas de dados que mantêm informação sobre zonas mortas de memória, por fim com a saída do analisador são feitos cálculos que mostram a economia de energia e o aumento da durabilidade da memória PCM.

Os testes feitos para este trabalho utilizaram o conjunto de *benchmarks* SPEC2006 (HENNING, 2006) e mostraram que, evitando *write-backs* inúteis se pode economizar até 19,8% de energia e melhorar a durabilidade de uma memória PCM em até 26,2%.

2.4.2 *Bit Mapping for Balanced PCM Cell Programming*

Du et al. (2013) propõem o *double XOR mapping (D-XOR)* para fazer uma distribuição dos bits modificados entre grupos de células da memória PCM de forma balanceada.

Primeiramente, foram analisados padrões de ciclos e agrupamentos de bits modificados nas escritas. Em seguida, os autores propuseram o D-XOR para fazer o balanceamento do desgaste das células de PCM e diminuir o tempo de escrita.

Nos resultados os autores mostraram que o D-XOR pode diminuir em média até 45% do tempo de escrita, isso fez com que o *throughput* fosse aumentado $1,8\times$.

2.4.3 Análise de desgaste de técnicas de correção de erros em *Phase-Change Memories*

Hoffman (2013) analisa, com uma modelagem matemática, como a probabilidade de *bit-flip* está relacionada com a durabilidade das memórias PCM, para os principais códigos de correção de erros (paridade, SECDED e BCH) e das principais técnicas de recuperação de falhas (ECP e SAFER).

Para desenvolver este trabalho o autor fez os experimentos de modelagem da probabilidade de *bit-flip*, para técnicas de correção de erros e, por fim, dos modelos analíticos.

Nos resultados, o autor apresentou que ocorreu uma visível degradação da durabilidade dos mecanismos de recuperação de falhas que usam códigos de correção de erros. Um destaque dos resultados foi a técnica ECP que foi a única que não mostrou degradação da PCM. Também foi feita uma análise de eficiência energética, relacionando a durabilidade da PCM e o consumo de energia onde novamente, a técnica ECP se destacou nos resultados, como também a técnica SAFER. Por fim, o trabalho apresenta modelos analíticos probabilísticos das técnicas ECP, SECDED e uma análise da técnica PAYG baseada no modelo analítico da ECP, que foram proposto pelo autor.

2.4.4 *Curling-PCM: Application-Specific Wear Leveling for Phase Change Memory based Embedded Systems*

Liu et al. (2013) centram-se na gestão da PCM em sistemas embarcados, e propõem uma técnica de nivelamento de desgaste simples, mas eficaz para estender o tempo de vida de sistemas embarcados baseados em PCM.

A ideia básica deste trabalho é mover periodicamente áreas que ocorreram muitas escritas, que são identificados pelos padrões de aplicação de escrita em todo o chip PCM, e distribuir as escritas uniformemente de modo que o nivelamento de desgaste pode ser melhorado. O trabalho apresenta duas implementações a *full curling* e a *partial curling*.

Os experimentos destes trabalho focaram em analisar o número de *bit-flips* na PCM e o tempo de resposta das implementações. Os resultados mostraram que o número total de *bit-flips* com as implementações aumentou, mas o número máximo de *bit-flips* em uma célula de PCM diminuiu. Em relação ao tempo de resposta o *partial curling*

reduziu em até 77% em relação ao *full curling*.

2.4.5 *A three-stage-write scheme with flip-bit for PCM main memory*

Li et al. (2015) apresentam um esquema de escrita de três fases, cujo o objetivo é reduzir o número de bits alterados e a latência de gravação para a PCM.

Este trabalho combina as ideias do Flip-N-Write (CHO; LEE, 2009) e da política de dois estágios de escrita (YUE; ZHU, 2013). O esquema de três fases compara os dados antigos e os novos e os novos dados são reorganizados de forma que minimize o número de bits alterados. Em seguida, todos os bits alterados são divididos em novos bits 0 e novos bits 1 e são atualizados separadamente para evitar o desperdício de tempo de escritas mistas.

Os resultados dos experimentos mostraram que o esquema proposto diminui 43,5% as mudanças de bit, 16,6% o tempo das escritas e 34,6% o consumo de energia das escritas em média.

2.4.6 *Profiling Patterns of Bit Flipping for Software Transactional Memories*

No trabalho, Teixeira *et al* (2014), é feita uma análise dos padrões de escrita e de bits alterados do conjunto de *benchmarks* STAMP e seu impacto em uma memória PCM.

Para o trabalho o autor implementou uma instrumentação, com a ferramenta Pin-Tools, que conta cada escrita feita à memória e também faz uma avaliação de quantos bits foram modificados em cada escrita. A implementação foi feita da seguinte forma: a cada escrita na memória é incrementado um contador, que contabiliza o número de escritas. Também é armazenado o endereço de memória que está sendo escrita e o valor que será escrito nela. Com o endereço de memória é obtido o valor atual que está armazenado em um vetor e é feita uma comparação com o valor que será escrito naquele endereço, de modo a verificar quantos bits foram modificados. Essa comparação é feita em tempo de execução. Logo depois de fazer essa comparação, o vetor é atualizado com o valor escrito.

Os resultados do trabalho mostraram alguns padrões nas escritas das STMs. Um destes foi que na maioria dos *benchmarks* quanto maior era o número de *threads* maior era o número de escritas, o número de bits alterados e o número de posições que foram escritas. Mas alguns *benchmarks* mostraram um comportamento diferente. Conforme aumentava o número de *threads* o número de escritas, o número de bits alterados e o número de posições que foram escritas variavam, o que pode ser justificado em parte por suas implementações, parâmetros escolhidos entre outras razões.

2.4.7 Análise de Consumo de Energia e Desempenho de Memórias Transacionais em Software em Ambiente de Computação Real

Rico (2013) analisa as bibliotecas de STM TL2, TinySTM, SwissTM e AdaptSTM, e faz uma análise do consumo de energia e desempenho das STMs em ambiente de computação real, utilizando-se o *benchmark* STAMP.

No trabalho, o autor analisa o consumo de energia por meio de um microcontrolador especializado embutido na placa-mãe, presente na maioria dos servidores, denominado *Baseboard Management Controller* (BMC). Com isso, ele coletou o consumo de energia nas execuções de cada biblioteca de STM. Para a análise do desempenho, cada *benchmark* apresentava em sua saída o tempo de execução, e esse tempo, foi utilizado para a análise do desempenho.

Os resultados obtidos no trabalho mostram que a biblioteca SwissTM foi a mais eficiente em termos de consumo de energia e desempenho, seguida pela AdaptSTM, TinySTM e TL2. O autor constatou que a escalabilidade das STMs utilizadas está relacionada diretamente à particularidade das estratégias de detecção e resolução de conflitos empregada por cada biblioteca.

2.4.8 Experimentos com Gerenciamento de Contenção em uma Memória Transacional com Suporte em Software

Uma comparação do desempenho de diferentes implementações de gerenciador de contenção pode ser visto no trabalho de Kronbauer e Rigo (2009). Este apresenta uma abordagem nova para gerenciar a contenção entre transações, que leva em consideração os padrões de acesso aos diferentes dados de um programa ao escolher o gerenciador de contenção usado para o acesso a estes dados. Como biblioteca de STM, a RSTM (MARATHE et al., 2006) foi utilizada.

Para implementar esta abordagem proposta, os autores testaram algumas técnicas. Primeiramente eles testaram inserir um campo adicional em cada objeto transacional para denotar o gerenciador de contenção a ele associado, e fazer com que a transação detectasse qual gerenciador de contenção estava associado ao primeiro objeto aberto pela transação para utilizar este gerenciador pelo restante de sua execução, mas esta técnica se mostrou inviável visto o trabalho adicional inserido nos métodos *clone* e *redo*. Outra técnica testada foi utilizar herança para introduzir o campo citado apenas em objetos transacionais específicos, mas então foi preciso utilizar conversão de tipos dinâmica ao consultar o objeto a respeito de qual gerenciador de contenção estava associado a ele, o que pareceu ser outra fonte significante de trabalho adicional. Por fim, a técnica utilizada por eles neste trabalho foi permitir que o programador associe uma estratégia de gerenciamento de contenção a cada transação.

Com os resultados os autores chegaram à conclusão que conhecer o padrão

de acesso às estruturas de dados e também as configurações do *hardware* não foi o suficiente para determinar os melhores gerenciadores de contenção para uso na aplicação, uma vez que os resultados mostraram alta variação nos diferentes sistemas de computação testados.

2.5 Considerações Finais do Capítulo

Este capítulo apresentou o *background* deste trabalho. As características e os conceitos da memória PCM, das memórias transacionais e da biblioteca TinySTM foram discutidos. A TinySTM é uma implementação de STM para as liguagens C e C++. Ela faz parte do estado da arte de memórias transacionais e é uma das bibliotecas de STM mais utilizada nas pesquisas. Um dos motivos de utilizar a TinySTM neste trabalho é que os três versionamentos possíveis são implementados.

A memória PCM é uma memória não-volátil que surgiu como alternativa às DRAMs como memória principal em uma hierarquia de memória.

A grande vantagem da PCM está em relação ao seu consumo de energia, já que para manter os dados na memória não precisa realizar *refresh*. Apesar das vantagens, a PCM têm problemas em relação a suas escritas, que além de serem lentas ainda desgastam o seu material, diminuindo assim sua vida útil.

As TMs garantem maior abstração na codificação de programas concorrentes, pois possibilita uma programação de mais alto nível, onde o programador não se preocupa com o modo como as sincronizações serão feitas.

TMs implementam versionamentos de dados para garantir a atomicidade. O versionamento de dados em STM, que é o foco deste trabalho, necessitam de dois conceitos, a aquisição de escrita e o versionamento. A aquisição de escrita pode ser de dois tipos, bloqueante ou não bloqueante. Na aquisição de escrita bloqueante, a transação pode adquirir o privilégio de escrita em seguida que ocorre uma escrita ou no final da transação. Já o versionamento descreve onde será feita esta escrita, diretamente na memória (versionamento adiantado) ou primeiramente em um *buffer* (versionamento atrasado).

Por fim, conseguir prolongar a vida útil da PCM é o grande desafio das pesquisas sobre PCM. Evitar a mudança de bits, além de diminuir o consumo de energia, evita a degradação do material da PCM. Desta forma, este trabalho terá uma abordagem de apresentar a opção de versionamento de dados das memórias transacionais que provoque o menor desgaste de uma memória PCM.

3 AMBIENTE DE AVALIAÇÃO

Neste Capítulo será abordado o simulador implementado para este trabalho, o PCM-MS, assim como as ferramentas e os *benchmarks* utilizados.

3.1 *Phase-Change Memory - Multicore Simulator* (PCM-MS)

O *Phase-Change Memory - Multicore Simulator* (PCM-MS) é um simulador de hierarquia de memória onde a PCM é a memória principal, e pode simular arquitetura com múltiplos núcleos de processamento. Ele faz a simulação dos acessos à memória e determina os bits alterados na PCM para estimar seu desgaste e seu consumo de energia da PCM. Ele foi desenvolvido devido aos simuladores disponíveis não apresentarem em sua maioria a possibilidade de análise dos bits, e também devido a não simularem arquiteturas com múltiplos núcleos de processamento. O PCM-MS foi desenvolvido originalmente para este trabalho.

O PCM-MS utiliza arquivos de traço para fazer a simulação dos acessos à memória. Na Figura 7 pode ser visto com é feita a geração destes arquivos. Para gerar os arquivos, é feita uma instrumentação com a ferramenta Pintools na execução dos *benchmarks*. Essa instrumentação armazena os dados de acesso à memória para assim poder gerar os arquivos de traço.

Figura 7: Geração dos Arquivos de Traço

Na Figura 8 pode ser visto que o simulador PCM-MS recebe de entrada os arquivos de traço e gera como saída os resultados da simulação. Partindo de um arquivo de traço, onde estão ordenados os acessos à memória, o PCM-MS faz a simulação. Na

execução com múltiplos núcleos de processamento, existirá um arquivo de traço para cada *thread*.

Figura 8: Entrada e Saída da Simulação com o PCM-MS

3.1.1 Hierarquia de Memória

A hierarquia de *cache* no PCM-MS pode ser simulada em até três níveis, sendo o nível L1 o mais próximo do núcleo de processamento e o L3 o mais próximo do controlador de memória. Na Figura 9 pode ser visto como os níveis de *cache* são compartilhados. As *caches* de nível L1 não são compartilhadas por mais que um núcleo de processamento, ou seja, elas são únicas para cada núcleo de processamento. As *caches* de nível L2 são compartilhadas por duas *caches* de nível L1, ou seja, são compartilhadas por até dois núcleos de processamento. Já a *cache* de nível L3 é compartilhada por até quatro *caches* de nível L2, sendo assim são compartilhadas por até oito núcleos de processamento. Para manter a consistência é implementada uma política de *write-back* na hierarquia de *cache*.

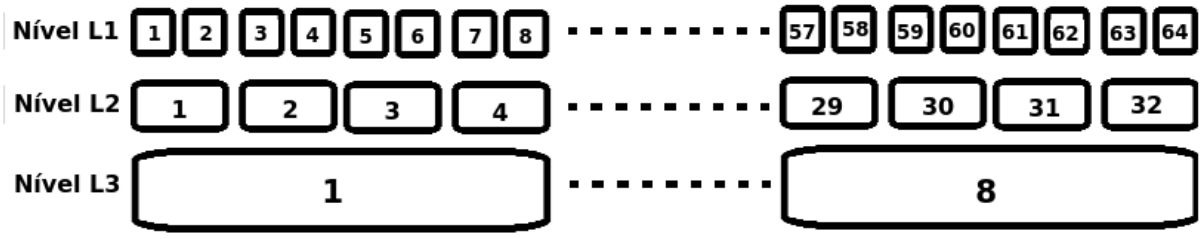


Figura 9: Níveis de *Cache*

A comunicação entre o nível de *cache* e a memória principal é feita por meio do controlador de memória ou *memory control* (MC), como pode ser visto na Figura 10. Na memória principal é feita a análise dos bits alterados nas escritas. A cada escrita na memória principal são analisados quantos bits foram alterados sendo possível, assim, calcular o desgaste que a PCM sofreu e a energia que foi consumida.

3.1.2 Limitações

Uma limitação do PCM-MS é que as *threads* são associadas a um determinado núcleo de processamento e não são escalonadas em núcleos diferentes. Com isso em

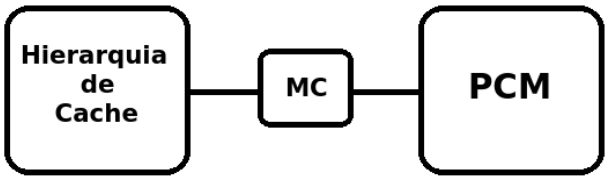


Figura 10: Comunicação entre a Hierarquia de Cache e a PCM

uma execução *multithread*, as *threads* são executadas somente em um único núcleo. O escalonamento das *threads* será providenciado para trabalhos futuros.

Outra limitação é que devido ao simulador utilizar arquivos de traço para fazer as simulações, não é possível calcular o tempo de execução da arquitetura simulada, visto que o tempo de execução é limitado ao acesso ao disco, para ler os arquivos de traço.

3.2 Pintools

Pin (LUK et al., 2005) é uma ferramenta de instrumentação dinâmica de programas. Pin foi projetado para fornecer uma funcionalidade onde um código escrito em C ou em C++, pode ser inserido em locais específicos de um executável. A versão do Pin utilizada foi a 2.12.

Diferente de outras ferramentas semelhantes que inserem o código de instrumentação estaticamente, modificando o executável antes da execução, o Pin insere o código dinamicamente, durante a execução. Outra característica do Pin é que ele salva e restaura os registradores que são substituídos pelo código inserido para que o aplicativo continue a executar. Pin também tem um acesso limitado aos símbolos e informações de depuração.

Uma limitação do uso de instrumentação no código é que esta pode alterar o comportamento dinâmico principalmente em termos de desempenho.

3.3 Eigenbench

Eigenbench (HONG et al., 2010) é um *microbenchmark* que faz uma avaliação ortogonal das características de aplicações que formam a base para o comportamento transacional. Ele também é útil para compreender plenamente o desempenho de um sistema transacional. O Eigenbench não possui, oficialmente em seu site, uma versão compatível com a TinySTM. Para utilizá-lo neste trabalho foi feita uma portabilidade que permitiu executar o Eigenbench com a TinySTM. As definições das características ortogonais podem ser vistas na Tabela 1. A versão utilizada foi a 0.8.0.

O Eigenbench faz uma simulação de cenários que podem ocorrer em um sistema transacional. Para isso, ele utiliza três *arrays* distintos:

Tabela 1: Definições das Características Ortogonais do EigenBench (HONG et al., 2010)

Características	**Definição**
Scalability	Número de *threads*
Contention	Probabilidade de conflitos de uma transação
Density	Proporção de leituras, em memória não compartilhada, dentro e fora de transações
Pollution	Proporção de escritas do total de acessos à memória
Predominance	Proporção de acessos compartilhados do total de acessos à memória
Temporal Locality	Probabilidade de repetição de um endereço por memória compartilhada
Transaction Length	Comprimento das Transações
Working-set Size	Tamanho da Memória Utilizada

- **Array1**: este *array* é chamado de "*hot array*", isso porque este *array* é compartilhado por todas as *threads*;
- **Array2**: este *array* é chamado de "*mild array*", devido a este *array* ser acessado por meio de transações. Cada *thread* acessa uma parte do **Array2**, dessa forma os acessos não causam conflitos;
- **Array3**: este *array* é chamado de "*cold array*", isso porque este *array* é dividido como o **Array2** mas não utiliza transações para realizar o acesso. Ele pode ser acessado dentro ou fora de transações.

3.4 STAMP *Benchmark*

STAMP (CAO MINH et al., 2008) é um conjunto de *benchmarks* criado para pesquisa de memórias transacionais, composto por oito *benchmarks*. Apesar de desenvolvido para a STM TL2, com algumas modificações disponíveis, pode ser usado no *TinySTM*. A versão do STAMP utilizada foi a 0.9.10. O conjunto de *benchmarks* STAMP foi escolhido devido ao fato de que implementa vários *benchmarks*, assim atingindo uma maior área de aplicações das STM além de ser o conjunto de *benchmark* mais utilizado na pesquisa de STM.

Os *benchmarks* implementados pelo STAMP serão descritos a seguir (CAO MINH et al., 2008).

3.4.1 Bayes

Esta aplicação implementa um algoritmo de aprendizado de redes Bayesianas, que é uma parte importante do aprendizado de máquina. Normalmente, nem as distribuições de probabilidades nem as dependências condicionais entre eles são conhecidas ou podem ser resolvidas por um ser humano, assim redes Bayesianas são frequentemente estudadas com os dados observados. O algoritmo específico implementa uma estratégia de *hill-climbing*, ou subida de encosta, que usa buscas locais e globais, semelhante à técnica descrita em Chickering *et al* (1997). Para eficientes estimativas de distribuição de probabilidade, utiliza-se uma *adtree*, ou árvore de decisão, a partir de Moore *et al* (1997).

3.4.2 Genome

Este *benchmark* implementa um programa de sequenciamento de genes que reconstrói a sequência de genes a partir de segmentos de um gene maior. O algoritmo usado para o sequenciamento de genes tem três fases:

1. Remove os segmentos duplicados utilizando uma *hash*;
2. Combina segmentos utilizando o algoritmo de pesquisa de sequência *Rabin-Karp* (KARP; RABIN, 1987);
3. Constrói a sequência.

3.4.3 Intruder

Este *benchmark* simula o Design 5 dos NIDS (*Network Intrusion Detection System*) descritos por Haagdorens *et al* (HAAGDORENS; VERMEIREN; GOOSSENS, 2005). Pacotes de rede são processados em paralelo e passam por três fases: captação, remontagem e detecção. A estrutura de dados principal na fase de captura é uma simples fila, e a fase de remontagem utiliza um dicionário (implementado por uma árvore autobalanceada), que contém a lista de pacotes pertencentes à mesma sessão. Ao avaliar seus cinco designs para um NIDS *multithread*.

3.4.4 Kmeans

Este *benchmark* foi extraído do *NU-MineBench 2.0* (PISHARATH et al., 2004). *K-means* é um método baseado em partição (BEZDEK, 1981) e é sem dúvida a técnica de agrupamento mais utilizada. Este algoritmo é comumente usado para partição de itens de dados em subconjuntos relacionados. Cada *thread* processa uma partição dos objetos iterativamente. A versão transacional adiciona uma transação para proteger o *update* do centro do *cluster* que ocorre durante cada iteração.

3.4.5 Labyrinth

Dado um labirinto, este *benchmark* encontra os caminhos de menor distância entre os pares de pontos inicial e final. O algoritmo de roteamento utilizado é o algoritmo de Lee (LEE, 1961).

Nesse algoritmo, o labirinto é representado como uma grade, em que cada ponto da grade pode conter ligações adjacentes, para os pontos da grade que não estão nas diagonais. O algoritmo busca um caminho mais curto entre os pontos de conexão, através da realização de uma busca em largura, e marca cada ponto da grade com a sua distância para o início. Esta fase de expansão acabará por chegar ao ponto final, se a conexão for possível. A segunda fase de rastreamento estabelece a ligação, seguindo todo o caminho, diminuindo a distância. Este algoritmo assegura o encontro do caminho mais curto entre um ponto inicial e final, no entanto, quando vários caminhos são feitos, um caminho pode bloquear outro.

3.4.6 SSCA2

Scalable Synthetic Compact Applications 2 (SSCA2) (BADER; MADDURI, 2005) é composta por quatro *kernels* que operam em um grande, dirigido e ponderado grafo. Estes quatro *kernels* são comumente usados em aplicações que vão desde a biologia computacional até a segurança. O SSCA2 incide sobre um *Kernel*, que constrói uma estrutura de dados eficiente utilizando matrizes de adjacência e matrizes auxiliares.

3.4.7 Vacation

Este *benchmark* implementa um sistema de reserva de viagens alimentado por um banco de dados não-distribuído. A carga de trabalho é composta por vários segmentos dos clientes que interagem com o banco de dados via gerenciador de transações do sistema.

O banco de dados é composto por quatro tabelas: carros, quartos, voos e clientes. Os três primeiros têm relações com os campos que representam um número único de identificação, quantidade reservada, a quantidade total disponível e o preço. A tabela de clientes acompanha as reservas feitas por cliente e o preço total das reservas que eles fizeram. As tabelas são implementadas como árvores rubro-negras.

3.4.8 Yada

Este *benchmark* implementa o algoritmo de Ruppert para refinamento de malha (RUPPERT, 1995). A versão transacional é similar, em design, ao apresentado em Kulkarni *et al.* (2006). A estrutura de dados básica utilizada é um grafo. Ele armazena todos os triângulos de malha (é um conjunto que contém os segmentos de contorno de malha), e uma fila de tarefas que contém os triângulos que precisam ser refinados. Em cada iteração do algoritmo, um pequeno triângulo é removido da fila e

uma retriangularização é realizada na malha. Quaisquer novos triângulos finos que resultem da retriangularização, são adicionados à fila.

3.5 Considerações Finais do Capítulo

Este capítulo apresentou o simulador PCM-MS, que foi implementado para este trabalho, e as demais ferramentas e *benchmarks* utilizados neste trabalho. O PCM-MS simula uma hierarquia de memória e faz a simulação dos acessos a essa hierarquia, onde a PCM é a memória principal. Uma característica importante dele é que ele faz simulações de arquiteturas com múltiplos núcleos de processamento. A ferramenta PinTools, auxilia o PCM-MS gerando arquivos de traço com os acessos à memória.

Os *benchmarks* utilizados estão entre os mais utilizados na pesquisa de STM. O Eigenbench devido ele simular diferentes aplicações e características que permitem analisar sistemas transacionais. O STAMP é um conjunto de *benchmarks*, sendo o mais utilizado na pesquisa de STM devido ele ser robusto e abranger uma ampla área de aplicações das STM.

4 CARACTERIZAÇÃO DOS VERSIONAMENTOS DE DADOS

Neste capítulo são discutidos os resultados da caracterização dos versionamentos de dados. A caracterização dos versionamentos de dados foi feita com o *benchmark* Eigenbench. Este benchmark foi utilizado por ser útil ao compreendimento pleno do desempenho de um sistema transacional em aplicações que formam a base do comportamento transacional. A caracterização é feita por meio da avaliação do desempenho, apresentado em forma de *speedup*, dos diferentes versionamentos em diferentes cenários e aplicações do *benchmark* Eigenbench. Os resultados desta caracterização serão úteis para o entendimento dos resultados do próximo capítulo, onde será analisado os padrões de acessos à memória. Isso porque entendendo o comportamento dos versionamentos em determinadas características de execução, facilita-se a compreensão do porque dos versionamentos terem certo comportamento ou apresentarem certo padrão.

Para a avaliação dos versionamentos de dados, foram utilizadas abreviações nas figuras, onde:

- **WBE**: para versionamento de dados **WRITE_BACK_ETL**;
- **WBC**: para versionamento de dados **WRITE_BACK_CTL**;
- **WT**: para versionamento de dados **WRITE_THROUGH**;
- **Unp**: Para execução sem nenhum tipo de proteção à memória compartilhada;

4.1 Configurações dos Experimentos

A princípio os experimentos foram executados na mesma máquina que os experimentos do Capítulo 5, mas devido ao alto desvio padrão, os resultados ficaram inconclusivos. Isso ocorreu devido a máquina apresentar uma arquitetura NUMA, e como o *benchmark* Eigenbench baseia-se em acessos a memória, os resultados apresentavam um desvio padrão muito alto e executar um número maior de vezes ficava inviável devido ao grande tempo de execução de alguns experimentos ser muito grande.

Com isso, os experimentos com o *benchmark* Eigenbench foram executados em uma máquina com um processador Intel Core i7-2600 com 3.4GHz e três níveis de *cache* (L3 com 8MB, L2 com 1MB e L1 com 256KB). O processador tem 4 *cores* e 8 *threads* com *Hyper-Threading*. Foi utilizado o sistema operacional GNU/Linux Ubuntu 12.04 LTS 64-bits. Todos os experimentos com o Eigenbench foram executados 30 vezes cada. A maioria dos experimentos foram executados com 8 *threads* e de forma sequencial. O experimento que mediu a escalabilidade foi executado com 1, 2, 4 e 8 *threads* e de forma sequencial.

Tabela 2: Parâmetros Utilizados com o EigenBench

	N	**A1**	**A2**	**A3**	**R1**	**W1**	**R2**	**W2**	**R3i**	**R3o**	**W3(i/o)**	**Ict**
Default	8	0	32k	0	0	0	90	10	0	0	0	0
Conflict	-	Var	Var	-	45	5	45	5	-	-	-	-
Density	-	512	31k	512	8	2	82	8	Var	Var	-	-
T. Len	-	-	-	-	-	-	Var	Var	-	-	-	-
Locality	-	-	-	-	-	-	-	-	-	-	-	Var
Pollution	-	-	-	-	-	-	-	Var	-	-	-	-
Predom	-	-	16k	16k	-	-	-	-	-	Var	-	-
Scale	Var	-	-	-	-	-	-	-	-	-	-	-
Workset	-	-	Var	-	-	-	-	-	-	-	-	-

Var = Significa que o parâmetro foi variado;
N = Número de *Threads*;
A<**número**> = *Array* <número>;
R/W<**número**> = Leituras/Escritas no *Array* <número>;
R/W3i = Leituras/Escritas no *Array* 3 dentro de transações;
R/W3o = Leituras/Escritas no *Array* 3 fora de transações;
Ict = Probabilidade de repetição do endereço.

Os parâmetros utilizados pelo Eigenbench para os experimentos podem ser vistos na Tabela 2. A Tabela 2 apresenta os valores utilizados como parâmetros para cada característica. A maioria das características são executadas somente utilizando o *array* 2, que é o *array* acessado por transações mas que não ocorre conflito, ou o *array* 3, que é o *array* que não é acessado por transações. Isso para que os conflitos não atrapalhem o desempenho dos resultados.

4.2 *Scalability*

Para o experimento *Scalability*, o número de *threads* foi variado de 1 a 8. Como mostra a Figura 11, todos os versionamentos apresentaram um comportamento semelhante, com o aumento do número de *threads*, o que aumentou também o desempenho. O versionamento WT teve os melhores resultados, pois escreve diretamente na memória, fazendo com que não tenha nenhum *overhead* no momento do *commit*. O versionamento WBC apresentou o pior desempenho para todos os cenários. Isso porque ele tem que adquirir todos os *locks* ao mesmo tempo (no momento da efetivação

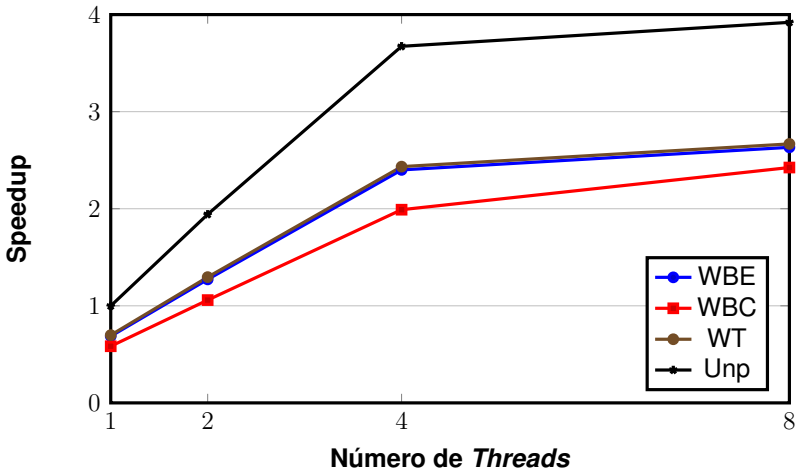


Figura 11: Resultados do Experimento *Scalability*

da transação, o *commit*). O *Speedup* não ultrapassou 4, mesmo com 8 *threads*, isso devido ao processador utilizar a tecnologia *hyper-threading*, e como o *benchmark* faz uma análise através de leituras e gravações na memória, o *hyper-threading* não é eficaz.

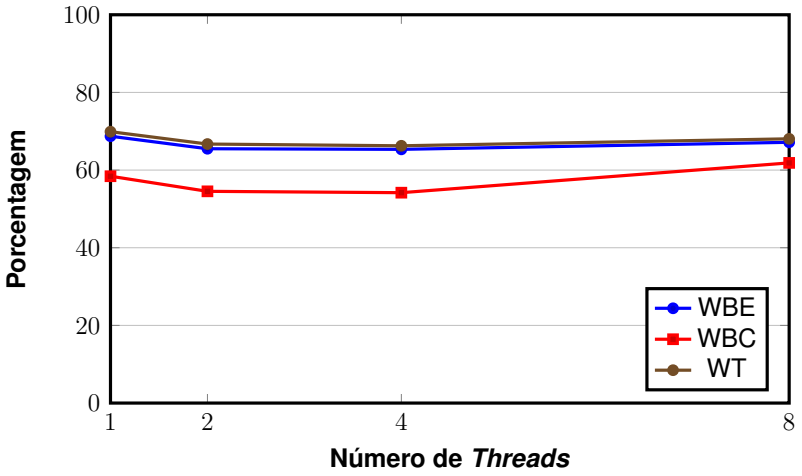


Figura 12: Resultados da Porcentagem Máxima de Escalabilidade (PME)

A Figura 12 apresenta a porcentagem máxima de escalabilidade (PME), que é a relação entre o tempo de execução dos versionamentos com o tempo de execução sem nenhum tipo de proteção. Os versionamentos tiveram as melhores PME com uma única *thread*. Isso é uma tendência esperada, já que *threads* adicionais inserem um *overhead* e no caso da execução sem nenhum tipo de proteção esse *overhead* não existe. Quando foi modificado o número de *threads* de 1-2 e de 2-4, ocorreu uma diminuição no PME, mas quando mudou o número de *threads* de 4-8 o PME aumentou. Esse aumento se deu devido ao uso de *hyper-threading*, como o *benchmark* faz uma análise através de leituras e escritas na memória, as execuções são limitadas pelo acesso à memória causando esse resultado. O versionamento WT mostrou as

melhores PME, isso porque ele escreve diretamente na memória.

4.3 *Contention*

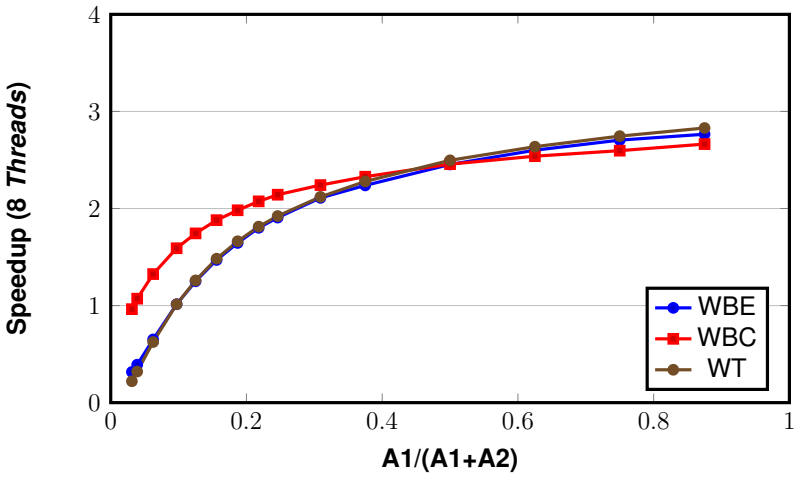


Figura 13: Resultados do Experimento *Contention*

O tamanho da memória compartilhada entre *threads* foi variada para a medição do experimento *contention*. A memória compartilhada foi aumentada de 1 KB de 32 KB, onde quanto menor for o tamanho da memória compartilhada maior é a contenção. A Figura 13 mostra que em cenários com alta contenção o versionamento WBC apresentou os melhores resultados. Isso porque os versionamentos com aquisição do *lock* adiantado (caso dos versionamentos WBE e WT) são abortados mais frequentemente (como pode ser visto na Figura 14). No entanto, com a diminuição da contenção, o versionamento WBC apresenta o pior desempenho. Em cenários com baixa contenção, o versionamento WT obteve os melhores resultados. Isso ocorreu porque em cenários com baixa contenção, onde há uma baixa taxa de *aborts*, ele tem vantagem, uma vez que ele escreve diretamente na memória e não tem o *overhead* de adquirir os *locks* no momento da efetivação das transações.

A Figura 14 apresenta o número de *aborts* do experimento *contention*. Como pode ser visto os versionamentos WT e WBE obtiveram um número semelhante de *aborts* em todos os cenários, com o versionamento WT tendo um número de *aborts* um pouco maior. Isso se deve ao seu menor *overhead* na efetivação da transação e a manipulação rápida da leitura-após-escrita e escrita-após-escrita (FELBER; FETZER; RIEGEL, 2008). Assim, ele detecta conflitos mais rapidamente que o versionamento WBE. O versionamento WBC mostrou o menor número de *aborts* em todos os cenários. Isso se deve à TinySTM ter um gerenciador de contenção não-agressivo, que quando uma transação detecta um conflito é imediatamente interrompida e com isso os versionamentos com aquisição do *lock* atrasado (caso do versionamento WBC) são abortados com menos frequência. Um fato interessante é que o versionamento

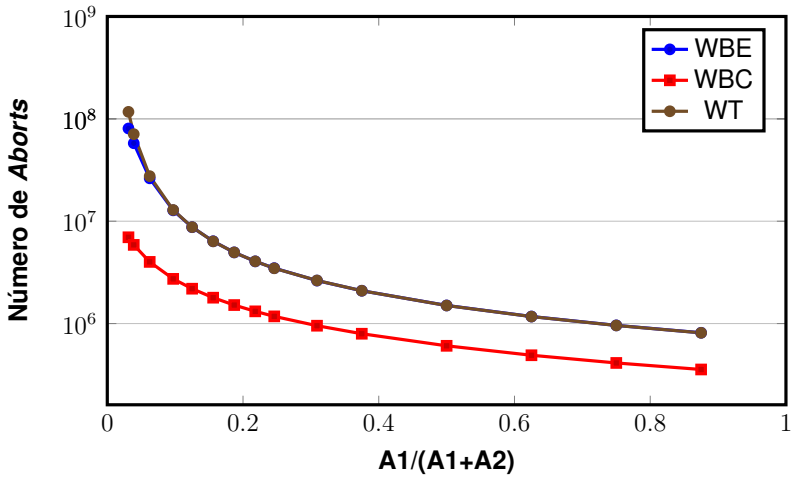


Figura 14: Número de Aborts dos Experimentos com a Aplicação *Contention*

WBC apresentou um número de *aborts* 2,5 vezes menor que os outros versionamentos, quando $A1/(A1 + A2)$ = 0,5, mas o desempenho foi o pior. Isso mostra que um *abort* com o versionamento WBC é muito custoso devido ao *abort* acontecer apenas no momento da efetivação da transação.

4.4 *Density*

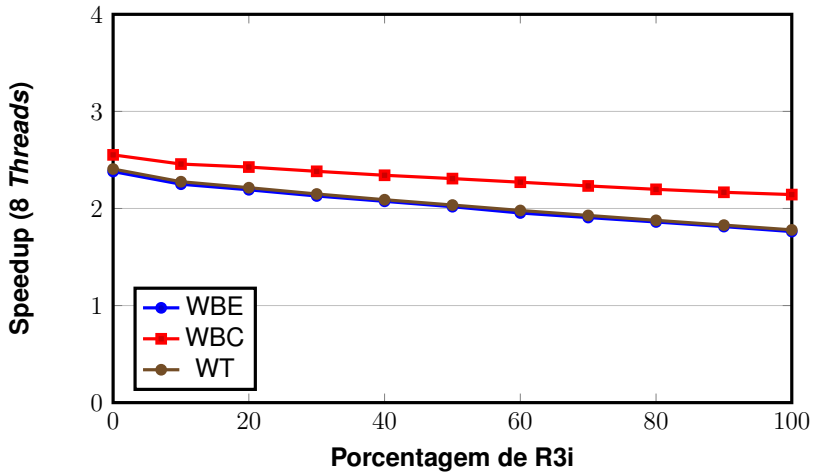


Figura 15: Resultados do Experimento *Density*

A proporção de leituras no *Array* 3 dentro e fora de transações foi variado, para realizar a medição do experimento *density*. Esta variação foi de 0% a 100% e de 100% a 0%, respectivamente. A Figura 15 mostra que todos os versionamentos tiveram o mesmo comportamento. Em todos os cenários, quando a porcentagem de leitura no *Array* 3 é aumentada dentro de transações, o desempenho teve uma pequena diminuição, isso porque com a adição de mais instruções em uma transação, aumenta o número de *aborts*. O versionamento WBC obteve os melhores resultados em todos

os cenários, isso ocorreu devido aos cenários apresentarem uma alta contenção o que favorece o versionamento WBC.

4.5 *Transaction Length*

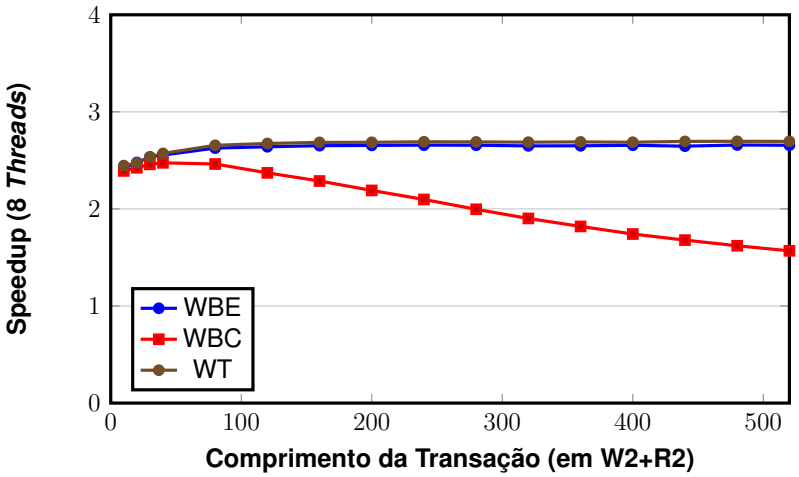


Figura 16: Resultados do Experimento *Transaction Length*

Outro experimento foi desenvolvido de forma a medir o comportamento dos versionamentos em relação ao comprimento de transação, esse experimento foi o *transaction length*. Para os quatro primeiros cenários, o comprimento das transações foi incrementado de 10 em 10, enquanto para os demais cenários, o incremento foi de 40 em 40 até 520. Em todos os cenários, o número de leituras foi de 90% do total de acessos à memória e o número de escritas foi de 10%. A Figura 16 mostra que os versionamentos WT e WBE tiveram o mesmo comportamento com o aumento do comprimento das transações. O versionamento WT obteve o melhor desempenho em todos os cenários, isso porque como não há contenção nesse experimento o versionamento WT se sai melhor (como pode ser visto na Figura 13). O versionamento WBC obteve o pior desempenho em todos os cenários e, diferente dos outros que aumentaram o desempenho de acordo com comprimento da transação, ele perdeu desempenho. Esse comportamento pode ser explicado devido à necessidade da aquisição de todos os *locks* no momento da efetivação da transação, causando assim um maior *overhead* em transações mais compridas.

4.6 *Temporal Locality*

Para o experimento *temporal locality*, o lct foi variado de 0 a 1 para medir localidade temporal. A $lct = 0$ representa que um endereço não é acessado mais de uma vez por transações, e $lct = 1$ representa que apenas um único endereço é acessado pelas transações (HONG et al., 2010). Os resultados deste experimento podem ser

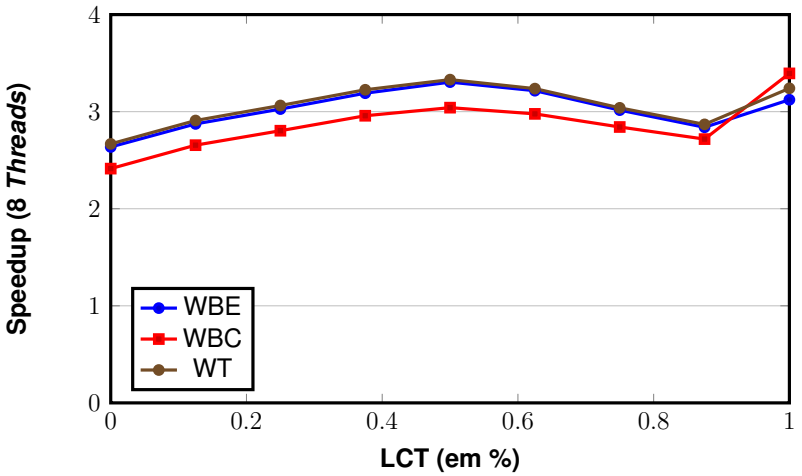


Figura 17: Resultados do Experimento *Temporal Locality*

vistos na Figura 17. Os resultados mostraram que todos os versionamentos tiveram um comportamento semelhante. O versionamento WT obteve o melhor desempenho até $lct = 0,875$. Quando $lct = 1$, o versionamento WBC obteve o melhor resultado. Isso ocorreu pois como este versionamento verifica e adquire os *locks* no momento da efetivação da transação e como as transações acessam apenas um único endereço, o versionamento verifica e adquire o *lock* apenas uma vez, enquanto os demais versionamentos devem verificar o *lock* em cada acesso à memória por meio de uma transação.

4.7 *Pollution*

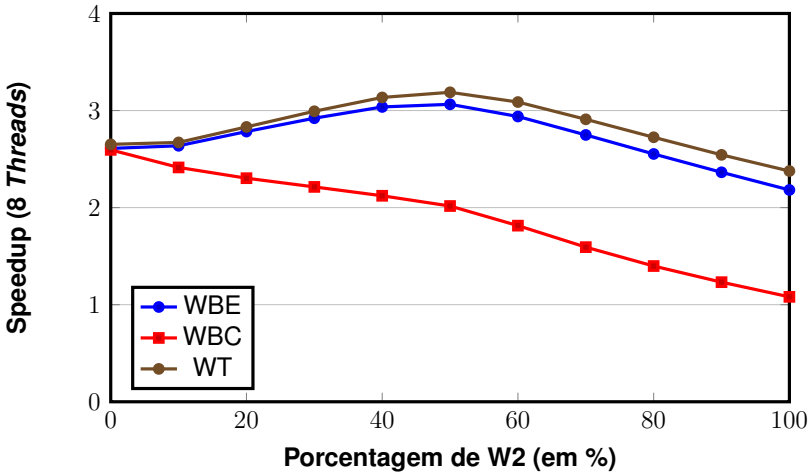


Figura 18: Resultados do Experimento *Pollution*

Para o experimento *Pollution*, a porcentagem de escritas foi aumentada de 0% a 100%. A Figura 18 mostra que os versionamentos WT e WBE obtiveram um comportamento semelhante. O versionamento WT mostrou o melhor desempenho em todos

os cenários. Isso ocorre porque ele escreve diretamente na memória, enquanto os outros versionamentos escrevem em um *buffer* antes de escrever na memória. O versionamento WBC obteve o pior desempenho em todos os cenários do experimento: enquanto os outros versionamentos ganharam desempenho até 50%, o WBC perdeu desempenho. Isso ocorre devido ao fato de ele ter que adquirir todos os *locks* no momento da efetivação da transação e, assim, com o aumento do número de escritas, aumenta também o *overhead*.

4.8 *Predominance*

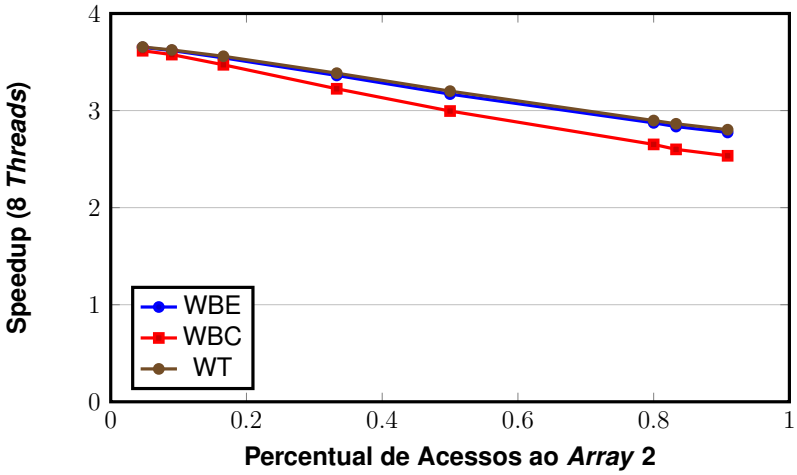


Figura 19: Resultados do Experimento *Predominance*

Para medir o experimento *Predominance*, é feita uma variação das leituras no *Array* 3 fora das transações. O parâmetro R3o foi variado de 10 a 2000. A Figura 19 apresenta os resultados deste experimento. Analisando os resultados pode ser visto que os versionamentos tiveram o mesmo comportamento. De acordo com o aumento do percentual de acessos ao *Array* 2 (*array* que é acessado por transações que não causam conflito) o desempenho diminui. Essa perda de desempenho é esperada, uma vez que TMs inserem uma sobrecarga em tempo de execução. O versionamento WT obteve o melhor desempenho em todos os cenários, isso ocorreu devido a sua escalabilidade ser melhor que a dos outros versionamentos.

4.9 *Working-set*

A fim de medir o experimento *Working-set* foi variado o tamanho do *Array* 2 de 8KB até 132MB. A Figura 20 mostra que ocorre uma queda drástica no desempenho de todos os versionamentos quando 1MB/*thread*. Este é um efeito da hierarquia de *cache*, porque oito *threads* executando concorrentemente no processador i7, que tem um tamanho de *cache* L3 de 8MB, acabam transbordando a *cache*. O versionamento

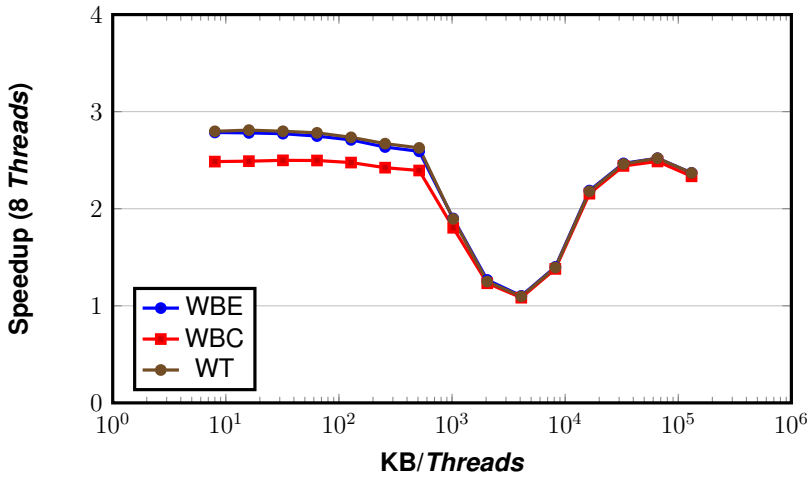


Figura 20: Resultados do Experimento *Working-Set*

WT obteve o melhor desempenho até a queda dramática. Isso aconteceu devido a sua escalabilidade ser melhor que a dos outros versionamentos. Após a perda de desempenho, todos os versionamentos tiveram um desempenho similar. Isso pode ser explicado devido ao acesso à memória fora do chip, ter se tornado um gargalo na execução.

4.10 Discussão

Os resultados da caracterização dos versionamentos de dados mostram que o versionamento WT foi o melhor na maioria das características analisadas. O versionamento WBE obteve um comportamento semelhante ao versionamento WT, mas com um desempenho um pouco pior para algumas características, devido a sobrecarga no momento da efetivação das transações. O versionamento WBC foi o melhor para cenários com alta contenção, isso devido aos versionamentos com a aquisição do *lock* adiantado serem abortados mais frequentemente (como pode ser visto na Figura 14), o que causa perda do desempenho. No entanto, os *aborts* com o versionamento WBC são mais custosos, pois acontecem somente no final da transação. Em geral, a caracterização dos versionamentos mostraram que o parâmetro com maior impacto no desempenho foi a aquisição do *lock* para a gravação e não tanto a questão de onde é armazenado o valor.

5 CARACTERIZAÇÃO DOS PADRÕES DE ACESSO À MEMÓRIA

Neste capítulo será feita a análise dos padrões de acesso à memória e seu impacto na PCM. Para fazer a análise foram executados experimentos com o conjunto de *benchmarks* STAMP no simulador PCM-MS. Foi escolhido o STAMP devido ele ser um conjunto de *benchmarks* robusto e que abrange grande área de aplicações das STM. Outro motivo deste trabalho utilizar o STAMP é para facilitar comparabilidade com outros trabalhos na área. Para auxiliar na análise dos resultados, as tabelas com número de transações, número de *aborts* e as taxas de *misses* e *hits* nos diferentes níveis de *cache*, de todos os experimentos, estão nos apêndices deste trabalho.

Com os experimentos, foi visto que o *benchmark* Yada apresentava um erro durante a execução, com mais de uma *thread*, com o versionamento WBC. O erro aparece em todos os experimentos feitos com o versionamento WBC com mais de uma *thread*. Com isso foi feito um estudo do motivo pelo qual ocorria esse erro. Com o estudo ficou constatado que o erro é devido à má implementação do *garbage collector* (coletor de lixo) específico do *benchmark* Yada, pois quando uma transação acessava um dado que havia sido excluído, o *garbage collector* simplesmente abortava a execução do programa. Executando mais alguns experimentos, também foi constatado que para execuções com um número de *threads* maior que o número de núcleos de processamento o mesmo erro pode ocorrer para os demais versionamentos. O motivo pelo qual ocorre esse erro nos outros versionamentos, também é devido à má implementação do *garbage collector*. No caso de ter mais *threads* do que *threads* de sistema, isso também ocorre devido às *threads* serem escalonadas nas *threads* de sistema, podendo fazer com que uma transação exclua dados que podem estar sendo acessados por outras transações, e o *garbage collector* acaba por abortar a execução. Constatado esse problema foi decidido não apresentar os resultados com o *benchmark* Yada neste trabalho e para trabalhos futuros pretende-se corrigir este erro.

Para a avaliação dos versionamentos de dados foram utilizadas abreviações nas figuras, onde:

- **WBE**: para versionamento de dados **WRITE_BACK_ETL**;
- **WBC**: para versionamento de dados **WRITE_BACK_CTL**;
- **WT**: para versionamento de dados **WRITE_THROUGH**;

5.1 Configurações dos Experimentos

Os experimentos com os *benchmarks* do STAMP foram executados em uma máquina com quatro nodos de processamento AMD Opteron 6276 com 2,3 GHz e três níveis de *cache* (L3 com 6 MB, L2 com 2 MB e L1 com 64 KB). Cada nodo de processamento tem 16 cores, formando um total de 64. Esta máquina tem 128 GB de memória principal, sendo 32GB por nodo separados em blocos de 16 GB. Foi utilizado o sistema operacional GNU/Linux Ubuntu Server 12.04 LTS 64-bits. Para cada combinação do *benchmark* STAMP e versionamento de dados foram feitas 30 execuções com 1, 2, 4, 8, 16, 32 e 64 *threads*. Os parâmetros utilizados pelo STAMP para os experimentos podem ser vistos na Tabela 3. Já as configurações do simulador podem ser vistas na Tabela 4.

Tabela 3: Parâmetros Utilizados com cada *Benchmark* do STAMP

Aplicações	Parâmetros
Bayes	-v32 -r1024 -n2 -p20 -i2 -e2 -s0
Genome	-g16384 -s64 -n16384
Intruder	-a10 -l32 -n32608 -s1
Kmeans	-m15 -n15 -t0.05 -i random-n65536-d32-c16.txt
Labyrinth	-i random-x128-y128-z5-n128.txt
SSCA2	-s17 -i1.0 -u1.0 -l3 -p3
Vacation	-n4 -q60 -u90 -r196608 -t16384

Tabela 4: Parâmetros Utilizados no Simulador PCM-MS

Nível	Número de Linhas	Associatividade	Tamanho de Bloco
L1	8 K	4	4
L2	64 K	4	4
L3	256 K	4	4

Os parâmetros do simulador, são para uma única L1, L2 e L3. No total como foram utilizadas até 64 *threads*, foram simuladas 64 *caches* de nível L1, 32 *caches* de nível L2 e 8 *caches* de nível L3. Como o simulador ainda não tem como definir o tamanho da PCM, atribui-se que a PCM tem tamanho suficiente para conter todo o programa.

5.2 Acessos à Memória

Para analisar o desgaste da PCM é preciso observar os acessos à memória (escritas e leituras). As Figuras desta seção apresentam a média de acessos à memória de cada *benchmark* do conjunto STAMP em cada cenário de execução, com o desvio padrão.

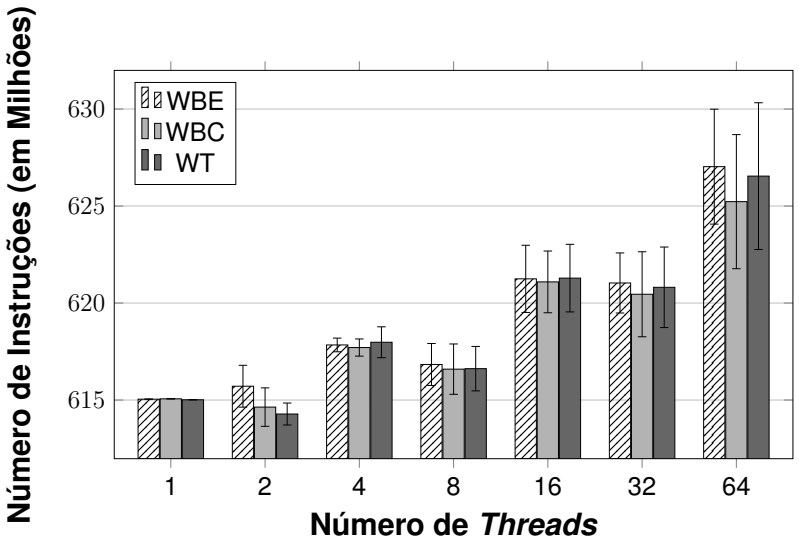


Figura 21: Total de Instruções de Acesso à Memória do *Benchmark* Bayes

Na Figura 21 pode ser vista a média de acessos à memória dos experimentos com o *benchmark* Bayes. Os resultados mostraram que todos os versionamentos apresentaram uma média de acessos à memória muito parecida, com poucos casos onde um versionamento apresentou uma média maior que outro (levando em consideração o desvio padrão). Os resultados são explicados devido à implementação do *benchmark* Bayes. O *benchmark* passa mais de 90% do seu tempo de execução em uma execução sequencial, fazendo com que os diferentes versionamentos não apresentem grandes diferenças. Já em relação ao aumento do número de *threads*, a cada aumento os versionamentos oscilaram entre aumento ou diminuição da média de acessos.

Os resultados dos experimentos com o *benchmark* Genome podem ser vistos na Figura 22. Em relação aos versionamentos, o versionamento WT apresentou a menor média de acessos em todos os cenários de execução. Isso ocorreu pois o versionamento WT escrever diretamente na memória, o que diminui o número de acessos no momento de uma efetivação da transação. O versionamento WBC apresentou a maior média de acessos em todos os cenários de execução. Essa maior média de acessos é explicada devido ao versionamento WBC detectar os conflitos no fim das transações, então um *abort* é muito custoso, ocasionando vários acessos à memória que não serão utilizados. Em relação ao número de *threads*, ocorreu uma diminuição da média de acessos quando aumentado o número de *threads* de 1-2 e 2-4. Isso se deve ao *benchmark* Genome ser separado em duas etapas (RUAN; LIU; SPEAR, 2014), onde

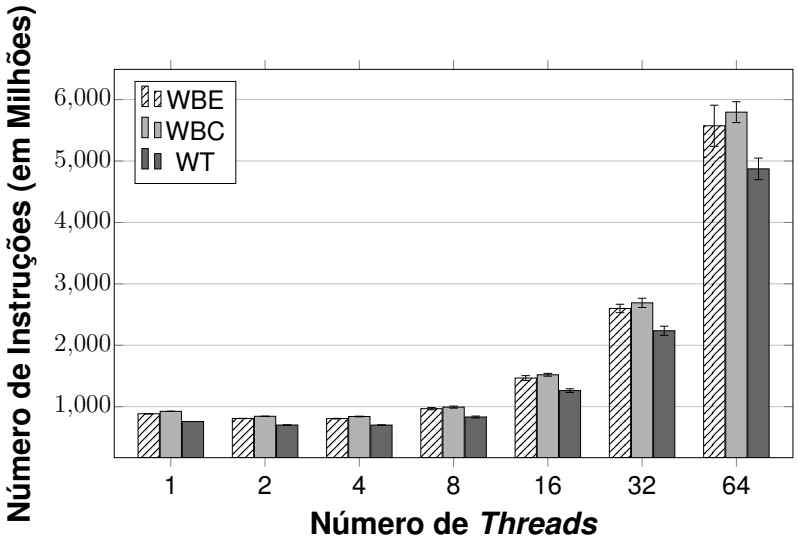


Figura 22: Total de Instruções de Acesso à Memória do *Benchmark* Genome

a primeira etapa ocorre somente leituras, e isso faz com que não ocorra um número elevado de *aborts* em cenários com baixa contenção (cenários com poucas *threads*), como nestes casos. Nos demais cenários, quando aumentado o número de *threads* a média de acessos também aumentou. Isso é esperado visto que com mais *threads* a tendência é que ocorra mais acessos à memória.

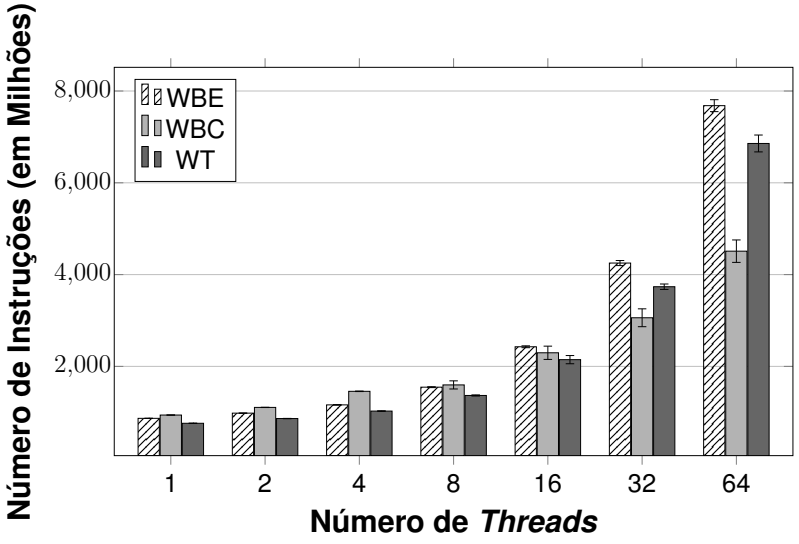


Figura 23: Total de Instruções de Acesso à Memória do *Benchmark* Intruder

A Figura 23 apresenta os resultados dos experimentos com o *benchmark* Intruder. Os resultados mostram que o versionamento WBC apresentou a maior média de acessos à memória, em cenários de execução com baixa contenção, mas nos cenários de alta contenção ele apresentou a menor média de acessos. Isso se deve ao versionamento WBC apresentar melhores resultados quando executado em ambientes de alta contenção, como apresentado na Figura 13, apresentando nestes ambientes um

número menor de acessos à memória. O versionamento WT apresentou a menor média de acessos até o cenário de execução com 16 *threads*, e nos cenários com 32 e 64 *threads* ele obteve uma média de acessos menor que o versionamento WBE e maior que o versionamento WBC. O motivo pelo qual isso ocorreu é que o versionamento WT escreve diretamente na memória, o que faz com que ele apresente uma média de acessos menor que os outros versionamentos. Já nos cenários de alta contenção, ele obteve uma média maior de acessos que o versionamento WBC devido ao versionamento WBC apresentar melhores resultados em cenários de alta contenção e ele apresentar resultados ruins (como pode ser visto na Figura 13). Em relação ao aumento do número de *threads* todos os versionamentos obtiveram um aumento da média de acessos conforme o aumento do número de *threads*, o que é esperado como já explicado.

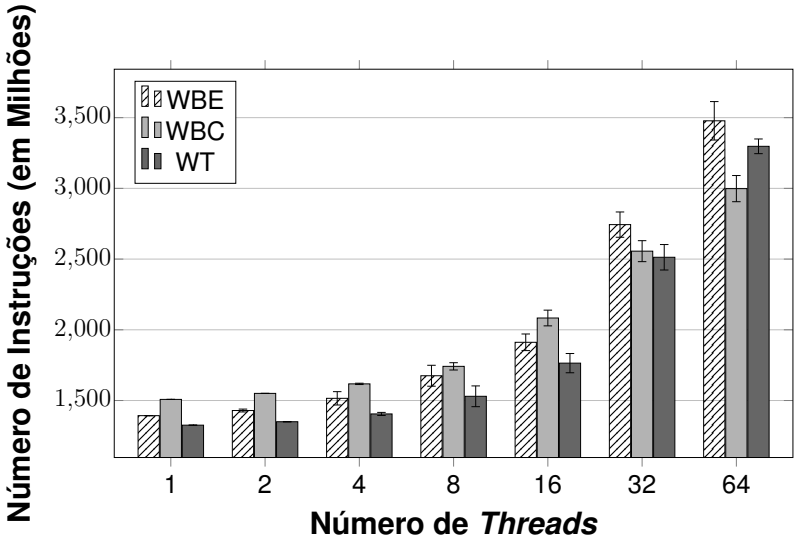


Figura 24: Total de Instruções de Acesso à Memória do *Benchmark* Kmeans

Na Figura 24 podem ser vistos os resultados do *benchmark* Kmeans. Em relação aos versionamentos de dados o versionamento WT apresentou a menor média de acessos à memória até o cenário de execução com 32 *threads*. Isso ocorreu devido ao versionamento WT escrever diretamente na memória, fazendo com que quando a transação é efetivada não ocorra mais acessos à memória, ao contrário do que acontece nos outros versionamentos. Já com 64 *threads*, o versionamento WBC obteve a menor média de acessos à memória. Isso ocorre devido ao versionamento WBC apresentar melhores resultados em cenários de alta contenção. Em relação ao número de *threads* o comportamento foi o mesmo do *benchmark* Intruder, conforme aumentou o número de *threads*, todos os versionamentos obtiveram um aumento na média de acessos à memória, o que é uma tendência.

Os resultados dos experimentos com o *benchmark* Labyrinth podem ser vistos na Figura 25. Nestes experimentos, os versionamentos apresentaram resultados

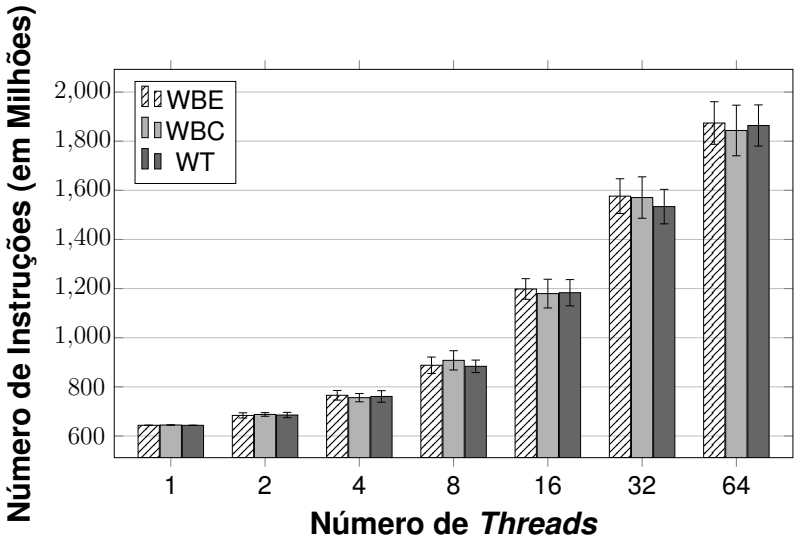


Figura 25: Total de Instruções de Acesso à Memória do *Benchmark* Labyrinth

muito parecidos, com poucos casos onde um versionamento apresentou uma média maior que outro (levando em consideração o desvio padrão). Isso se deve ao *benchmark* Labyrinth apresentar alta contenção e longas transações (CAO MINH et al., 2008), fazendo com que o versionamento que tem melhor desempenho com uma alta contenção (versionamento WBC) perca desempenho devido às transações longas e vice-versa. Outro motivo é que o *benchmark* apresenta um número pequeno de transações fazendo com que os versionamentos não apresentem grandes diferenças. Em relação ao número de *threads*, pode ser visto o mesmo comportamento dos *benchmarks* Intruder e Kmeans, conforme o aumento do número de *threads* ocorre também um aumento na média de acessos à memória.

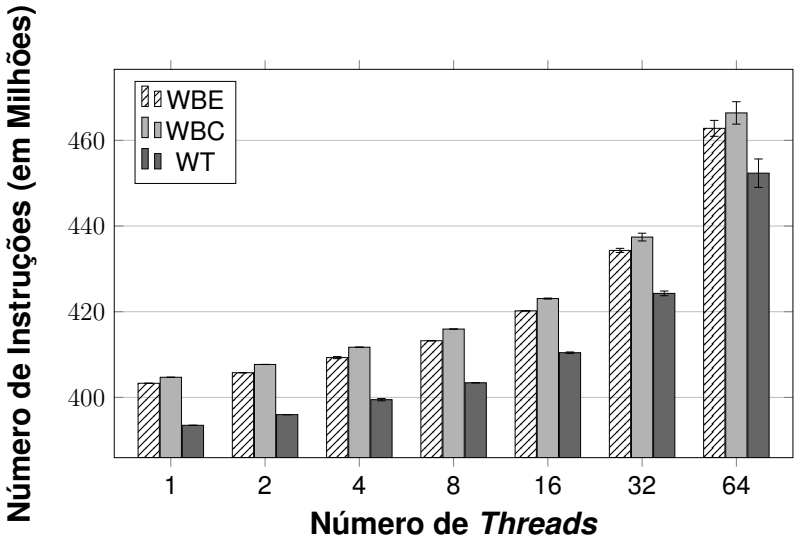


Figura 26: Total de Instruções de Acesso à Memória do *Benchmark* SSCA2

A Figura 26 apresenta os resultados dos experimentos com o *benchmark* SSCA2.

Em relação aos versionamentos em todos os cenários de execução, eles apresentaram o mesmo comportamento, onde o versionamento WBC apresentou a maior média de acessos à memória seguido do versionamento WBE, que apresentou a segunda maior média de acessos, e do versionamento WT, que apresentou a menor média de acessos. Esse comportamento ocorreu devido ao *benchmark* SSCA2 passar menos de 20% do tempo de execução dentro de transações e suas transações sejam pequenas (CAO MINH et al., 2008), fazendo com que os versionamentos não modifiquem esse padrão. Já em relação ao número de *threads* o *benchmark* SSCA2 obteve o mesmo comportamento dos *benchmarks* anteriores, conforme o aumento do número de *threads* ocorreu um aumento na média de acessos à memória.

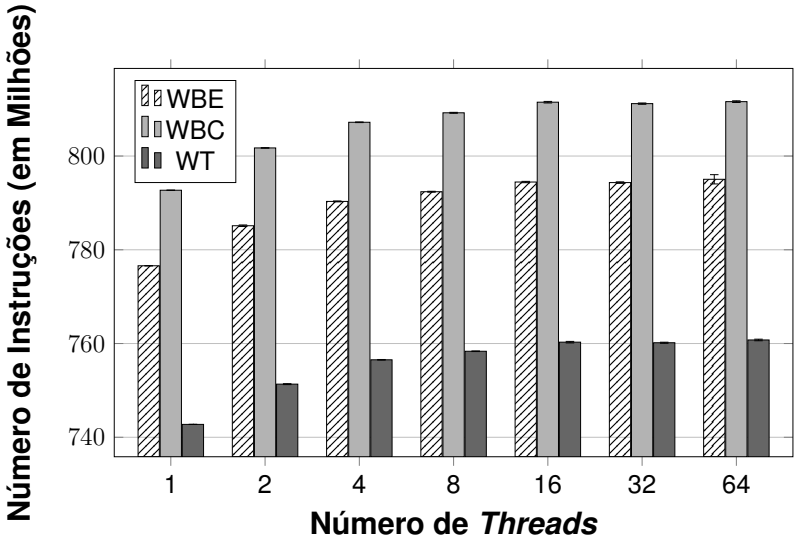


Figura 27: Total de Instruções de Acesso à Memória do *Benchmark* Vacation

Na Figura 27 pode-se observar os resultados dos experimentos com o *benchmark* Vacation. Os experimentos apresentaram o mesmo comportamento do *benchmark* SSCA2 em relação aos versionamentos, o versionamento WBC apresentou a maior média de acessos à memória seguido do versionamento WBE, que apresentou a segunda maior média de acessos, e do versionamento WT, que apresentou a menor média de acessos. O motivo pelo qual o versionamento WT apresentou a menor média de acessos, é devido a escrever diretamente na memória, diferente dos outros versionamentos que escrevem, primeiramente, em um *buffer* para, depois, caso efetivada a transação, escrever os valores na memória, o que faz com que eles tenham mais acessos à memória. Já na questão do versionamento WBC apresentar a maior média de acessos à memória, isso se deve a ele detectar os conflitos no fim das transações, então um *abort* é muito custoso, fazendo com que ocorra mais acessos à memória, diferente do versionamento WBE que detecta o conflito no momento em que acontece. Em relação ao número de *threads*, conforme o aumento do número de *threads*, a média de acessos à memória também aumentou, exceto no aumento de

16-32 *threads* que apresentou uma pequena diminuição do número de acessos.

5.3 Acessos à Memória Principal (PCM)

Depois de analisar os acessos à memória e o comportamento dos versionamentos, o próximo passo é analisar os acessos que ocorreram na PCM. Para a análise dos acessos à PCM, as leituras e escritas serão analisadas separadamente. As figuras desta seção apresentam a média de leituras e escritas, respectivamente, na PCM, do conjunto de *benchmarks* STAMP, em cada cenário de execução, apresentando também o desvio padrão.

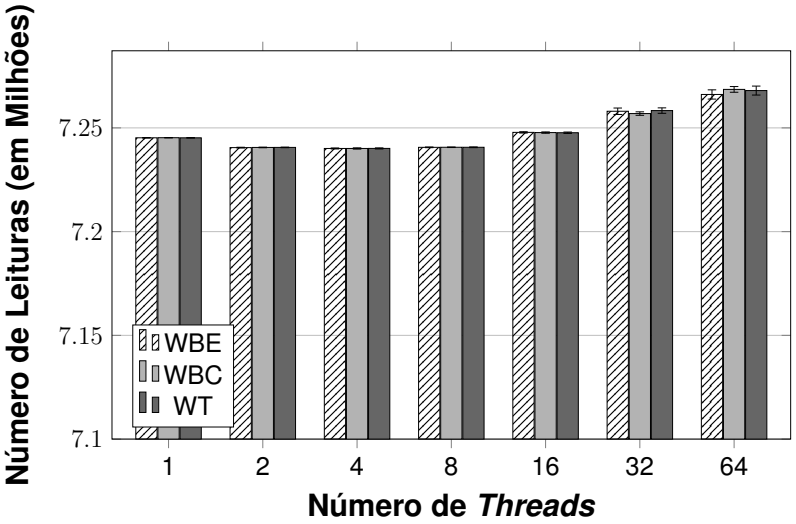


Figura 28: Número de Leituras na PCM dos Experimentos com o *Benchmark* Bayes

A Figura 28 apresenta a média de leituras na PCM dos experimentos com o *benchmark* Bayes. Como pode ser visto, em relação aos versionamentos, eles apresentaram resultados muito parecidos em todos os cenários de execução, apresentando pequenas diferenças somente nos cenários de alta contenção. Isso ocorreu devido ao *benchmark* Bayes ser pouco paralelizável, fazendo com que os versionamentos não apresentem grandes diferenças nas médias de leituras na PCM. Já em relação ao número de *threads*, conforme o aumento do número de *threads*, a média de leituras na PCM se manteve por volta de 7,2 milhões, apresentando pouca variação nos diferentes cenários de execução. Esse comportamento ocorre pelo mesmo motivo descrito anteriormente, o *benchmark* Bayes ser pouco paralelizável.

Os resultados dos experimentos com o *benchmark* Genome podem ser vistos na Figura 29. Em relação aos versionamentos, em todos os cenários eles não apresentaram uma diferença significativa na média de leituras na PCM. Estes resultados estão diretamente ligados a taxa de *misses* na hierarquia de *cache*, onde a taxa de *misses* fez com que os diferentes versionamentos apresentassem resultados muito parecidos em todos os cenários de execução. Já em relação ao número de *threads*, conforme o

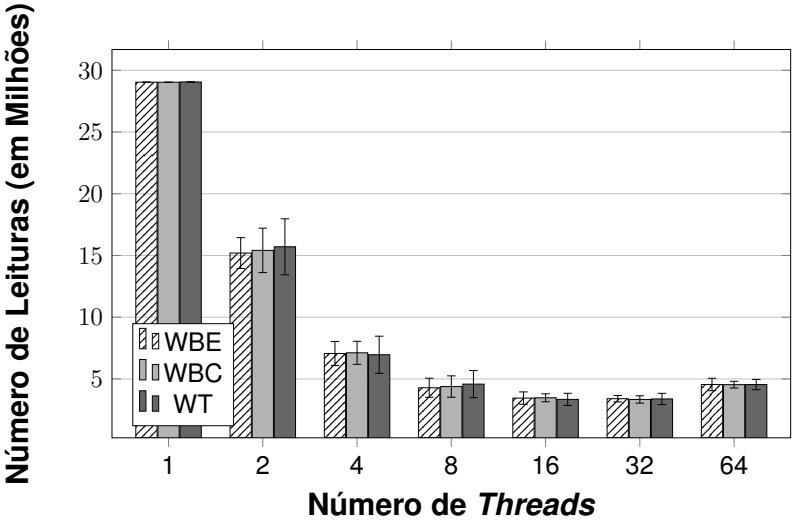


Figura 29: Número de Leituras na PCM dos Experimentos com o *Benchmark* Genome

aumento do número de *threads* ocorreu uma diminuição da média de leituras na PCM até o cenário de execução com 32 *threads*, quando ocorreu um pequeno aumento quando o número de *threads* passou de 32 para 64. Esse comportamento também está diretamente relacionado a taxa *misses* da hierarquia de *cache*, onde conforme o aumento do número de *threads*, ocorria uma diminuição dos *misses* na hierarquia de *cache*.

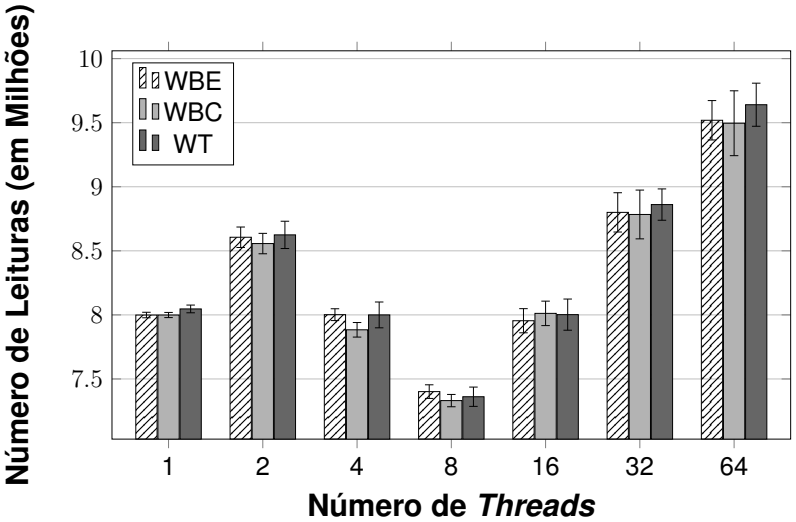


Figura 30: Número de Leituras na PCM dos Experimentos com o *Benchmark* Intruder

Na Figura 30 podem ser vistos os resultados dos experimentos com o *benchmark* Intruder. Em relação aos versionamentos, eles apresentaram uma média de leituras na PCM muito parecidas em todos os cenários de execução. Nos cenários com 2 e 4 *threads* o versionamento WBC apresentou uma média um pouco menor que os outros versionamentos. Isso está relacionado com a taxa de *misses* na *cache* L1, onde os versionamentos WBE e WT apresentaram uma maior taxa de *misses* que o

versionamento WBC. Conforme o aumento do número de *threads*, os versionamentos não apresentaram um padrão de diminuição ou aumento da média de leituras na PCM. Assim como os experimentos com o *benchmark* Genome, ele apresentou um comportamento relacionado diretamente com os *misses* da hierarquia de *cache*.

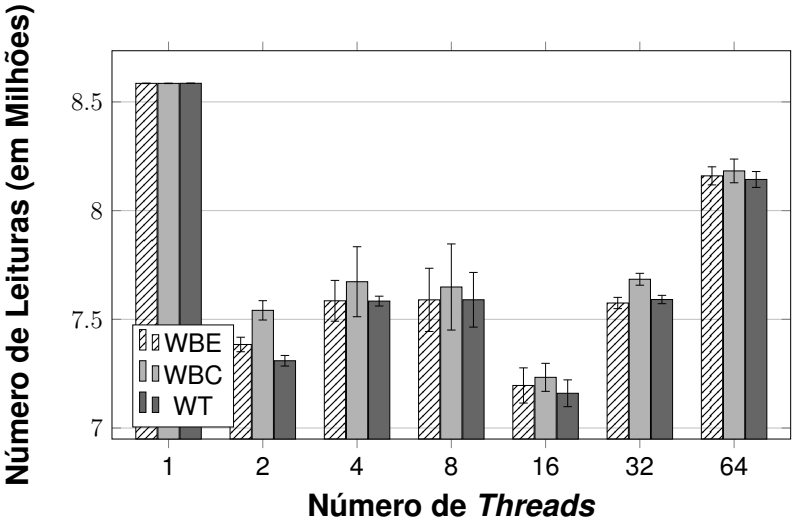


Figura 31: Número de Leituras na PCM dos Experimentos com o *Benchmark* Kmeans

A Figura 31 apresenta a média de leituras na PCM do *benchmark* Kmeans. Os versionamentos apresentaram resultados muito parecidos, sendo equivalentes em alguns cenários, levando em consideração o desvio padrão. Nos cenários com 2, 4, 32 e 64 *threads* o versionamento WBC apresentou uma média de leituras na PCM maior que os outros versionamentos. Isso está relacionado com a taxa de *misses* na hierarquia de *cache*, onde o versionamento WBC apresentou uma taxa maior que os outros versionamentos, e isso fez com que sua média de leituras na PCM fosse maior, até mesmo nos cenários onde ele apresentou as menores médias de acessos à memória, como nos cenários com 32 e 64 *threads*. O *benchmark* Kmeans não apresentou um padrão de comportamento da média de leituras na PCM conforme o aumento de *threads*, mas os resultados estão relacionados à taxa de *misses* na hierarquia de *cache*.

A média de leituras na PCM dos experimentos com o *benchmark* Labyrinth, pode ser vista na Figura 32. Os resultados mostraram que a média de leituras na PCM dos versionamentos foi muito parecida em todos os cenários de execução. Já em relação ao número de *threads*, conforme seu aumento, a média de leituras na PCM também aumentou. Os resultados estão relacionados diretamente à média de acessos à memória e ao acessos na hierarquia de *cache*.

Os resultados dos experimentos com o *benchmark* SSCA2 podem ser vistos na Figura 33. Entre os versionamentos, o versionamento WBE e o versionamento WBC apresentaram uma média de leituras na PCM equivalentes, já o versionamento WT apresentou a maior média de leituras na PCM em todos os cenários de execução, isso

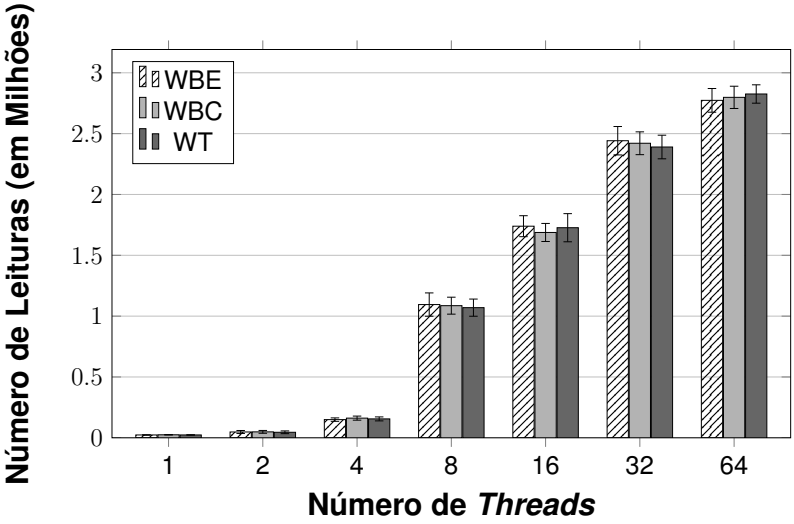


Figura 32: Número de Leituras na PCM dos Experimentos com o *Benchmark* Labyrinth

ocorreu mesmo ele tendo a menor média de acessos à memória. Esses resultados estão relacionados à taxa de *misses* da hierarquia de *cache*, onde o versionamento WT obteve uma taxa de *misses* maior que os demais versionamentos. Já em relação ao número de *threads*, conforme seu aumento, a média de leituras na PCM também aumentava, a única exceção foi no aumento de 2-4 *threads* que apresentou uma pequena queda na média de leituras na PCM.

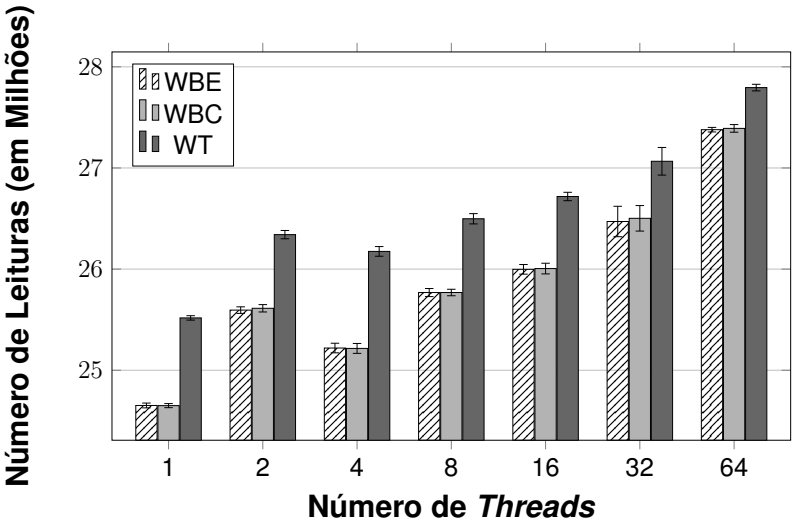


Figura 33: Número de Leituras na PCM dos Experimentos com o *Benchmark* SSCA2

A Figura 34 apresenta os resultados dos experimentos com o *benchmark* Vacation. Os versionamentos apresentaram uma média de leituras na PCM muito parecidas, com exceção do cenário de execução com 8 *threads* onde o versionamento WT apresentou uma média maior que os outros versionamentos. Esse comportamento está relacionado à taxa de *misses* no nível de *cache* L1, onde o versionamento WT obteve uma taxa mais alta que os outros versionamentos. Já conforme o aumento

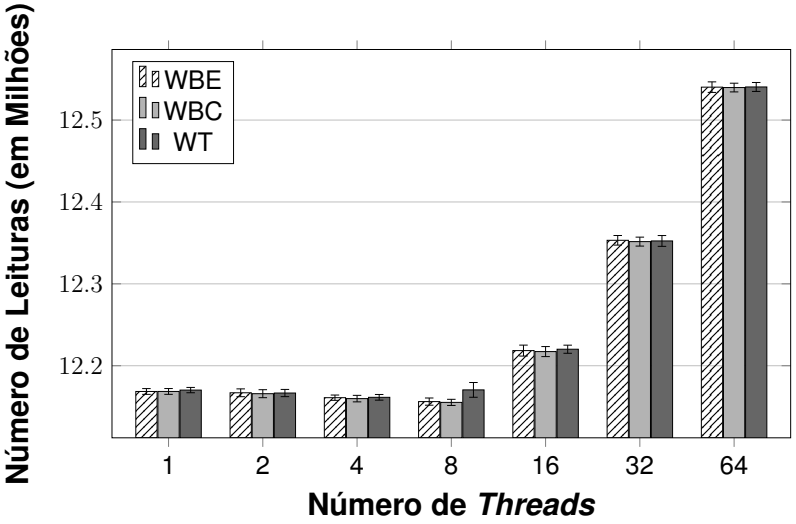


Figura 34: Número de Leituras na PCM dos Experimentos com o *Benchmark* Vacation

do número de *threads*, primeiramente houve uma diminuição da média de leituras na PCM, que ocorreu até o cenário de execução com 8 *threads*. A partir de 16 *threads*, a cada aumento do número de *threads* aumentava também a média de leituras na PCM. Esse comportamento está relacionado à taxa de *misses* na hierarquia de *cache*, que a partir do cenário de execução com 16 *threads*, a taxa de *misses*, nas *caches* L2 e L3, aumentou fazendo com que os acessos à PCM também aumentem.

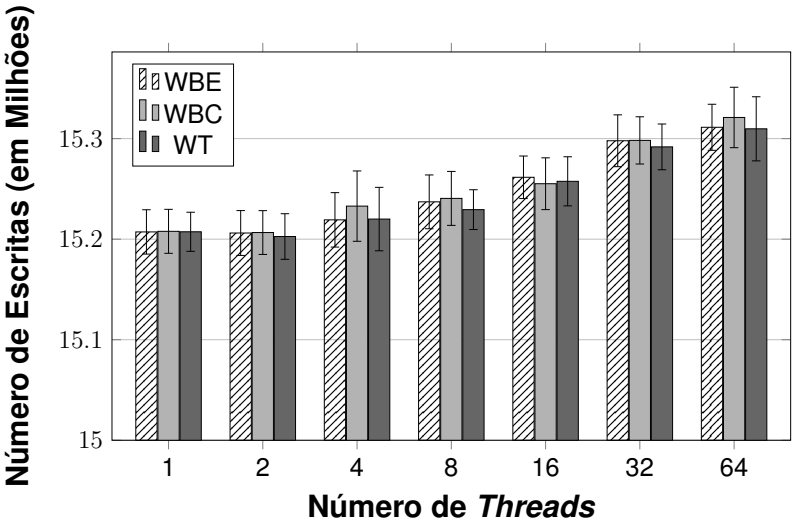


Figura 35: Número de Escritas na PCM dos Experientos com o *Benchmark* Bayes

A Figura 35 apresenta a média de escritas na PCM dos experimentos com o *benchmark* Bayes. Em relação aos diferentes versionamentos verifica-se que eles apresentaram estatísticas semelhantes. Isso ocorreu devido ao *benchmark* ser pouco paralelizável, fazendo com que os diferentes versionamentos não apresentem grandes diferenças. Em relação ao número de *threads* pode-se notar que, conforme o aumento do número de *threads*, ocorreu um pequeno aumento na média de escritas na PCM.

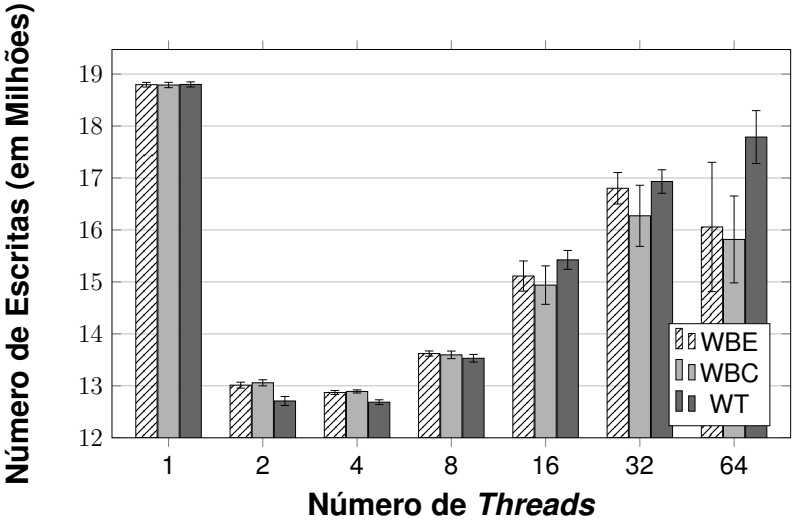


Figura 36: Número de Escritas na PCM dos Experientos com o *Benchmark* Genome

A Figura 36 apresenta a média de escritas na PCM dos experimentos com o *benchmark* Genome. Em relação aos versionamentos, o versionamento WT apresentou a menor média de escritas na PCM até o cenário de execução com 8 *threads*, mas nos cenários de alta contenção ele apresentou a maior média. Isso ocorreu devido ao número de *aborts* dos versionamentos, que como visto na Figura 14, em ambientes de alta contenção o versionamento WT não apresenta um bom desempenho devido ao alto número de *aborts*. O versionamento WBC e WBE apresentaram uma média de escritas com estatísticas semelhantes até 16 *threads*, mas a partir de 32 *threads* o versionamento WBC apresentou a menor média de escritas na PCM. Isso ocorreu devido ao versionamento WBC apresentar melhores resultados em cenários com alta contenção. Em relação ao número de *threads*, todas as execuções com mais de uma *thread* obtiveram uma média menor de escritas na PCM do que a execução com uma *thread*.

Na Figura 37 são apresentados os resultados dos experimentos com o *benchmark* Intruder. Em relação aos versionamentos, o versionamento WBC apresentou a média de escritas na PCM mais baixa a partir do cenário de execução com 8 *threads*. O versionamento WBE apresentou a média mais alta nos cenários de execução com mais de uma *thread* (nos cenários 16 e 32 *threads* o versionamento WT apresentou médias equivalentes ao versionamento WBE). Em relação ao número de *threads* todas as execuções com mais de uma *thread* obtiveram uma média de escritas na PCM menor que a execução com uma *thread*. Os resultados dos experimentos com o *benchmark* Intruder estão diretamente relacionados a taxa de *misses* da hierarquia de *cache*.

Os resultados dos experimentos com o *benchmark* Kmeans encontram-se na Figura 38. Os versionamentos apresentaram uma média de escritas na PCM muito similares, apenas nos cenários com 16 e 32 *threads* foi visto alguma diferença entre

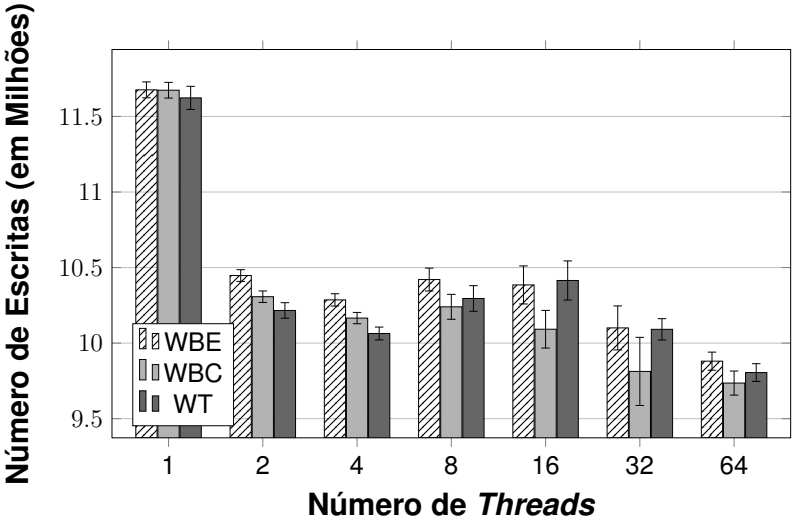


Figura 37: Número de Escritas na PCM dos Experientos com o *Benchmark* Intruder

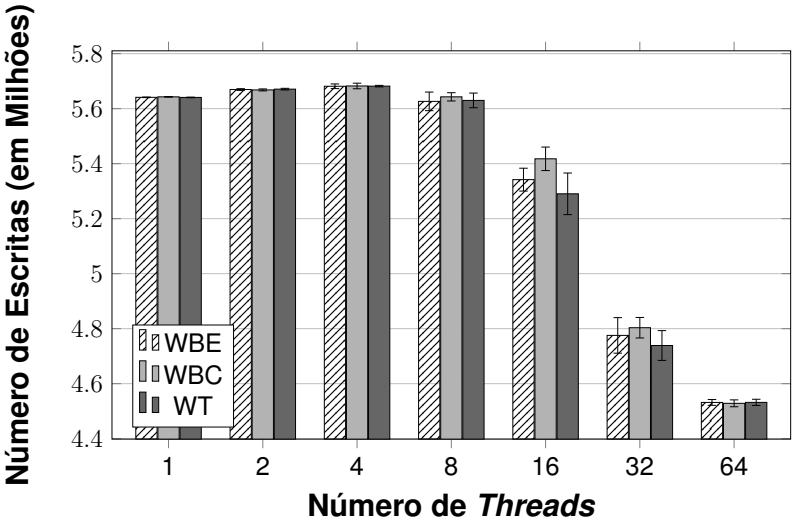


Figura 38: Número de Escritas na PCM dos Experientos com o *Benchmark* Kmeans

os versionamentos, onde o versionamento WBC apresentou resultados maiores que o versionamento WT nos cenários com 16 e 32 *threads* e maiores que o versionamento WBE no cenário com 16 *threads* (isso levando em consideração o desvio padrão dos resultados). Em relação ao número de *threads*, a média de escritas aumentou conforme o aumento do número de *threads* até o cenário com 4 *threads*, mas a partir do cenário com 8 *threads* a média de escritas na PCM diminuiu. Esses comportamentos, em relação aos versionamentos e do número de *threads*, ocorreram devido a taxa de *misses* da hierarquia de *cache* L1.

A Figura 39 apresenta os resultados dos experimentos com o *benchmark* Labyrinth. Os versionamentos apresentaram uma média de escritas na PCM muito parecida em todos os cenários de execução. Isso se deve ao pequeno número de transações que o *benchmark* executa, fazendo com que pouca diferença entre os

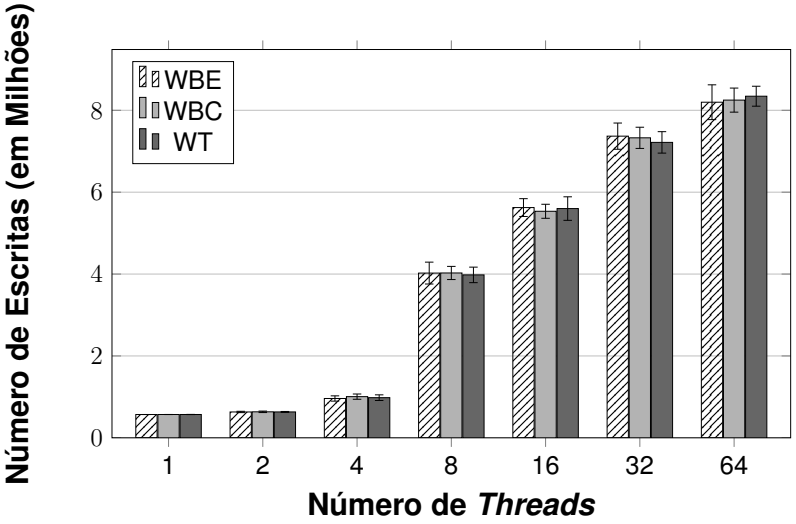


Figura 39: Número de Escritas na PCM dos Experientos com o *Benchmark* Labyrinth

versionamentos seja percebida. Já em relação ao número de *threads*, conforme o aumento do número de *threads*, a média de escritas na PCM também aumentou. Esse resultado está diretamente ligado ao número de acessos à memória, pois conforme o aumento dos acessos, aumenta também as escritas na PCM.

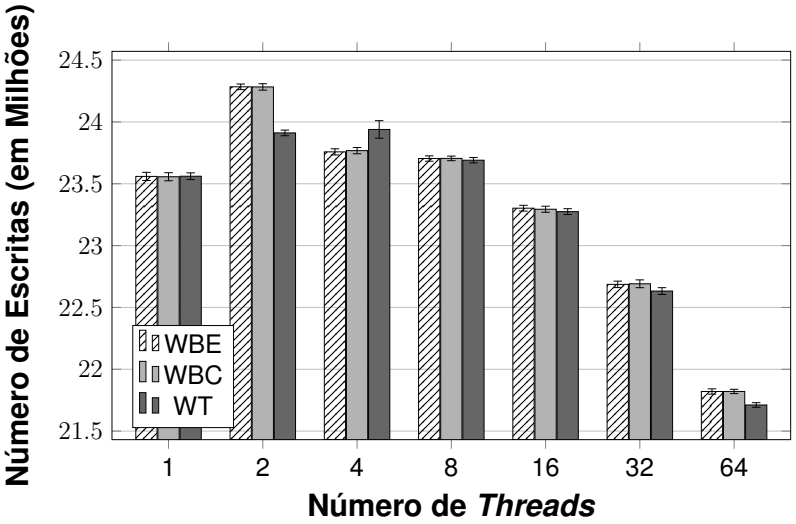


Figura 40: Número de Escritas na PCM dos Experientos com o *Benchmark* SSCA2

Na Figura 40 pode ser vista a média de escritas na PCM dos experimentos com o *benchmark* SSCA2. Em relação aos versionamentos eles apresentaram médias muito similares, somente o versionamento WT apresentou, em alguns cenários, médias diferentes dos outros versionamentos. Em relação ao número de *threads*, ocorreu um aumento da média de escritas quando foi aumentado o número de *threads* de 1-2, mas depois desse aumento, a cada novo aumento do número de *threads* ocorreu uma diminuição da média de escritas na PCM. Isso está relacionado com a taxa de *misses* da hierarquia de *cache*, que quanto maior a taxa de *misses* maior a média de escritas

na PCM.

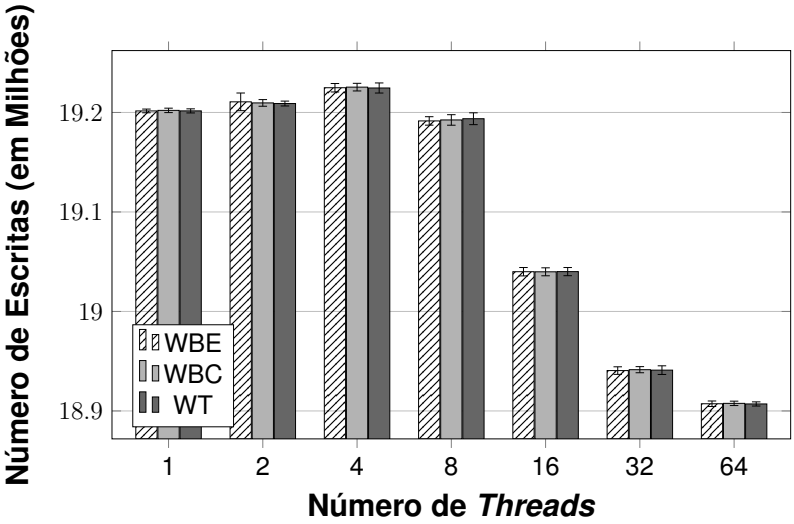


Figura 41: Número de Escritas na PCM dos Experientos com o *Benchmark* Vacation

A Figura 41 apresenta os resultados dos experimentos com o *benchmark* Vacation. Os versionamentos apresentaram uma média de escritas na PCM muito parecida em todos os cenários de execução. O comportamento dos versionamentos em relação ao número *threads* foi o seguinte, primeiramente, conforme o aumento do número de *threads* também ocorreu um aumento na média das escritas na PCM, isso ocorreu até o cenário com 4 *threads*. A partir do cenário de execução com 8 *threads*, os versionamentos apresentaram uma diminuição da média de escritas na PCM conforme o aumento do número de *threads*. Esse comportamento seguiu o mesmo comportamento da taxa de *misses* da hierarquia de *cache*.

5.4 Bits Alterados na PCM

Para analisar o desgaste da PCM é necessário analisar a quantidade de bits que foram alterados nas escritas. As Figuras apresentam o número de bits alterados na PCM, do conjunto de *benchmarks* STAMP, em cada cenário de execução apresentando também o desvio padrão.

A Figura 42 apresenta os resultados dos experimentos com o *benchmark* Bayes. Em relação aos versionamentos, eles apresentaram uma média de bits alterados muito parecida para todos os cenários de execução. Já em relação ao número de *threads*, os versionamentos se mantiveram por volta 12 milhões de bits alterados em todos os cenários de execução, variando um pouco para mais ou para menos. O *benchmark* Bayes como já foi dito passa mais de 90% do seu tempo de execução em uma execução sequencial e sendo assim não é possível ver muita diferença entre os versionamentos, visto que o *benchmark* é pouco paralelizável.

Na Figura 43 podem ser vistos os resultados dos experimentos com o *benchmark*

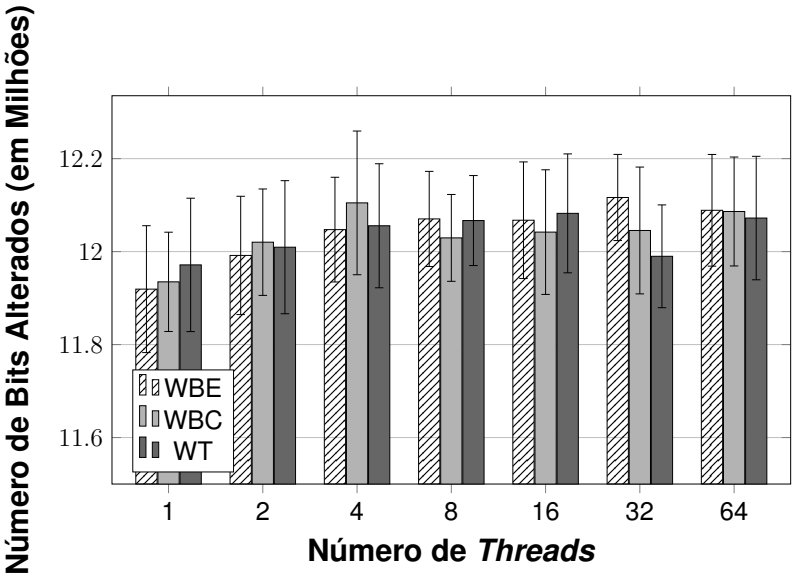


Figura 42: Bits Alterados na PCM do Experimento com o *Benchmark* Bayes

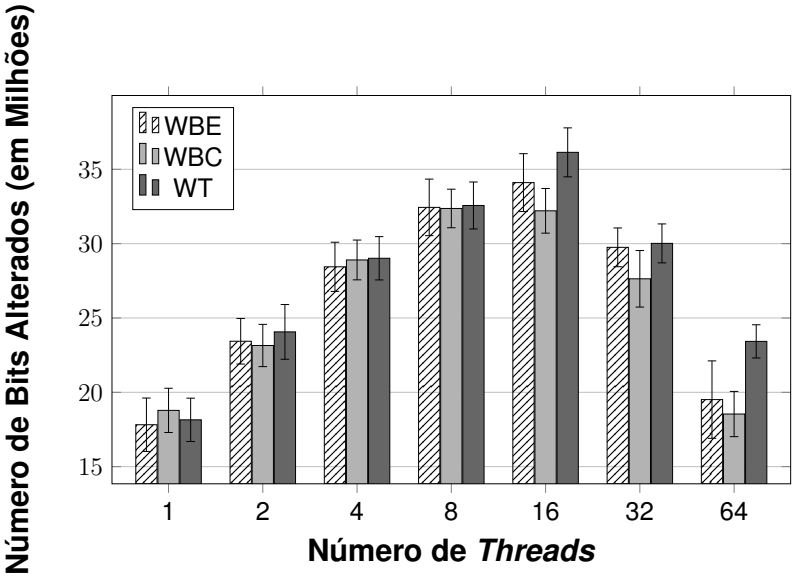


Figura 43: Bits Alterados na PCM do Experimento com o *Benchmark* Genome

Genome. Os resultados mostraram que nos cenários com 1, 2, 4 e 8 *threads*, os versionamentos obtiveram uma média de bits alterados muito parecida, sendo estatisticamente semelhantes. Mas, a partir do cenário com 16 *threads*, o versionamento WBC apresentou uma média menor de bits alterados do que os demais versionamentos enquanto que o versionamento WT apresentou as maiores médias de bits alterados. Esse comportamento tem relação com o número de *aborts* de cada versionamento, onde o versionamento WBC apresenta um número de *aborts* menor que os outros versionamentos. Já nos cenários de execução até 8 *threads*, o número de *aborts* foi pequeno, em relação ao número de transações, fazendo com que os versionamentos apresentem resultados muito próximos. Em relação ao número de *threads*, os versi-

onamentos apresentaram o comportamento de aumento da média de bits alterados até o cenário com 16 *threads*, mas nos cenários com 32 e 64 *threads* ocorreu uma diminuição da média de bits alterados.

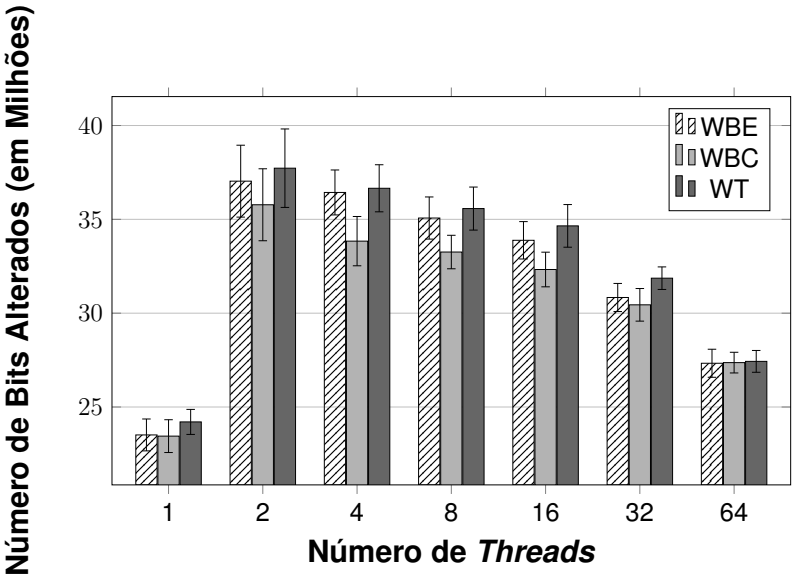


Figura 44: Bits Alterados na PCM do Experimento com o *Benchmark* Intruder

Nota-se, a partir da Figura 44, que a média de bits alterados dos experimentos com o *benchmark* Intruder. Os resultados mostraram que o versionamento WBC apresentou a menor média de bits alterados na maioria dos cenários de execução. Assim como o *benchmark* Genome, o motivo pelo qual ocorreu esse comportamento é o número de *aborts* dos versionamentos, onde o versionamento WBC apresentou um número menor de *aborts* que os outros versionamentos. Já os versionamentos WBE e WT apresentaram resultados muito similares em todos os cenários. Isso se dá devido aos versionamentos terem resultados muito parecidos como apresentado no Capítulo 4. O comportamento dos versionamentos em relação ao número de *threads* foi o seguinte, primeiramente, no aumento de 1-2 *threads* ocorreu um aumento na média de bits alterados, mas depois conforme o aumento do número de *threads* a média de bits alterados diminuía.

Os resultados dos experimentos com o *benchmark* Kmeans podem ser vistos na Figura 45. Os resultados mostraram que o versionamento WBC obteve a menor média de bits alterados na maioria dos cenários de execução. Assim como os *benchmarks* Genome e Intruder, isso está relacionado ao número de *aborts* do versionamento WBC que foi muito menor que o dos outros versionamentos. Os versionamentos WBE e WT apresentaram médias parecidas na maioria dos cenários, com o versionamento WT tendo uma média de bits alterados um pouco maior. Esse comportamento também está relacionado ao número de *aborts* dos versionamentos, onde os versionamentos apresentam números parecidos e como um *abort* no versionamento WT é mais cus-

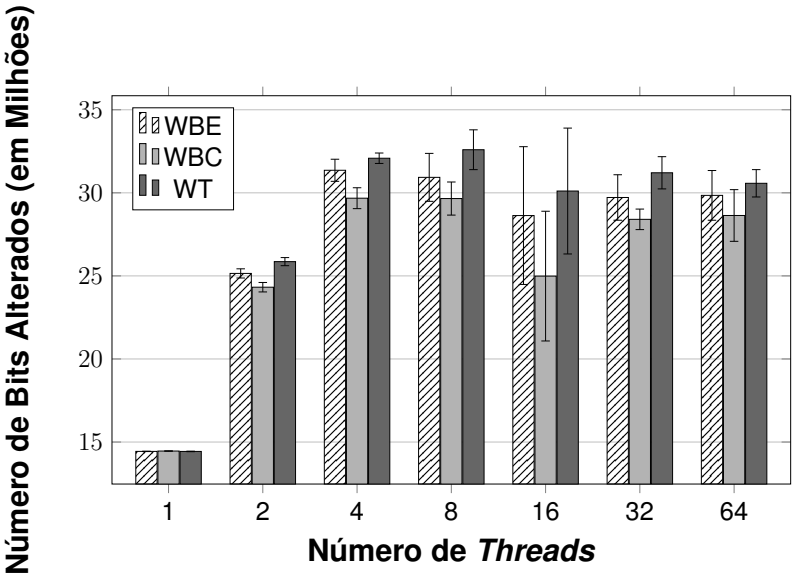


Figura 45: Bits Alterados na PCM do Experimento com o *Benchmark* Kmeans

toso, visto que ele tem que restaurar a memória, ele acaba tendo uma média um pouco maior que o versionamento WBE. Em relação aos diferentes cenários de *threads*, os versionamentos WBE e WBC apresentaram um aumento da média de bits alterados até o cenário de execução com 4 *threads*, enquanto o versionamento WT apresentou um aumento até o cenário com oito *threads*. No cenário com 16 *threads* os versionamentos apresentaram uma diminuição na média de bits alterados, mas também apresentou um alto desvio padrão. Já no cenário com 32 *threads* a média voltou a ter um aumento. E no cenário com 64 *threads* a média de bits alterados teve uma nova diminuição.

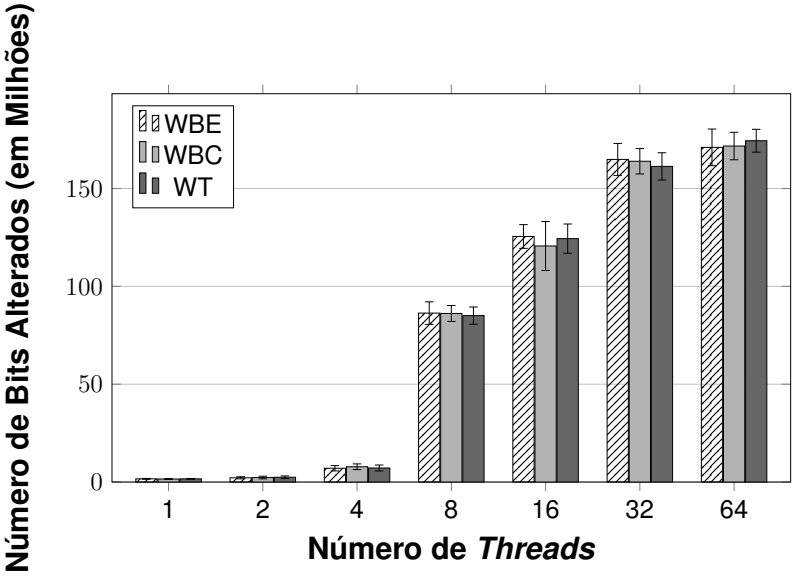


Figura 46: Bits Alterados na PCM do Experimento com o *Benchmark* Labyrinth

Já a Figura 46 apresenta as médias de bits alterados dos experimentos com o

benchmark Labyrinth. Em relação aos versionamentos, eles apresentaram médias de bits alterados muito parecidas em todos os cenários de execução. Já em relação ao número de *threads*, conforme o aumento do número de *threads* a média de bits alterados também aumentava. Esses resultados mostram o que já foi visto nas análises anteriores, o *benchmark* Labyrinth não apresentou grande diferença entre os versionamentos. Isso se deve ao fato do *benchmark* ter um número pequeno de transações (o experimento com maior número de transações, ocorreu com 64 *threads* que teve 384). Então como ele executa poucas transações, pouca também será a diferença dos versionamentos.

A Figura 47 apresenta os resultados dos experimentos com o *benchmark* SSCA2. Como pode ser visto, os versionamentos não apresentaram diferenças significativas em todos os cenários de execução. Isso ocorreu devido aos versionamentos apresentarem um número muito baixo de *aborts*, fazendo com que os resultados dos versionamentos sejam muito próximos. Já em relação ao número de *threads*, em seu aumento de 1-2 ocorreu um aumento na média de bits alterados, mas depois a cada novo aumento do número de *threads*, a média de bits alterados diminuía ou se mantinha com uma média estatisticamente equivalente.

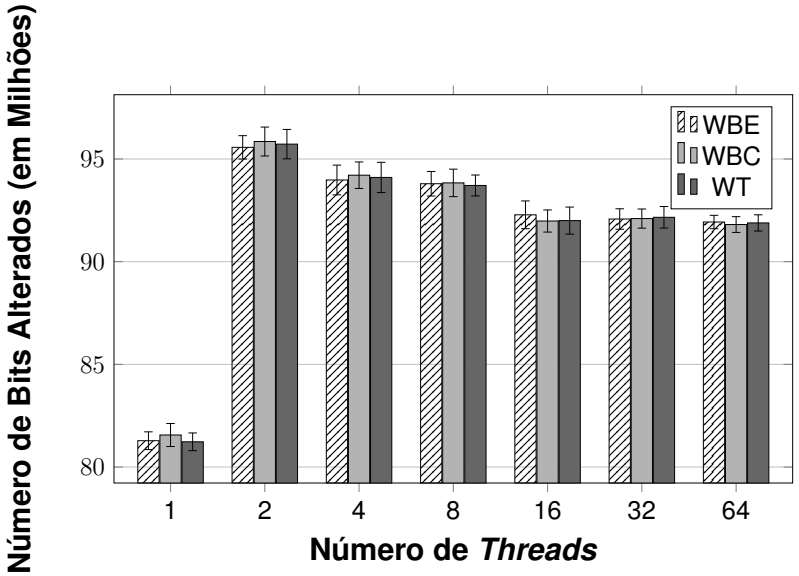


Figura 47: Bits Alterados na PCM do Experimento com o *Benchmark* SSCA2

Os resultados dos experimentos com o *benchmark* Vacation podem ser vistos na Figura 48. Em relação aos versionamentos, eles apresentaram resultados muito parecidos em todos os cenários de execução. Isso ocorreu devido ao número de *aborts* dos experimentos com este *benchmark* serem muito baixos em todos os cenários de execução, assim como o *benchmark* SSCA2, fazendo com que os resultados dos versionamentos sejam muito parecidos. Já em relação ao número de *threads*, primeiramente, ocorreu um pequeno aumento da média de bits alterados, conforme o aumento

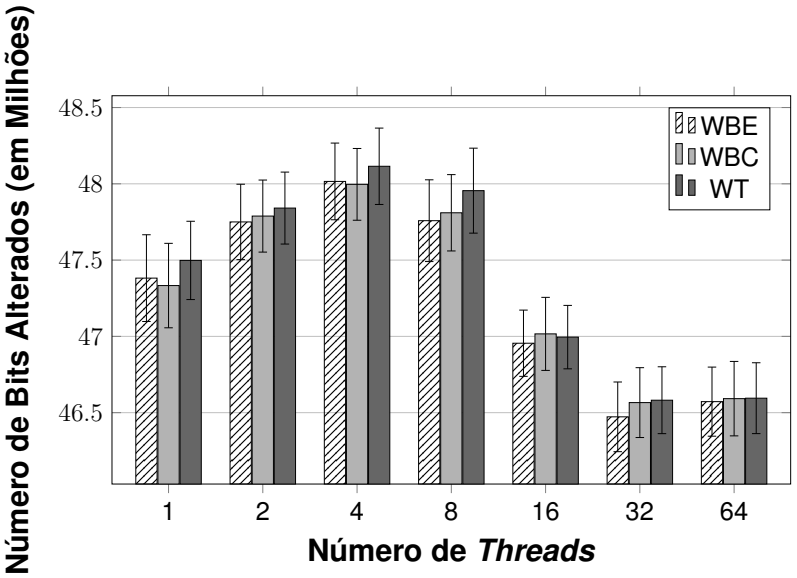


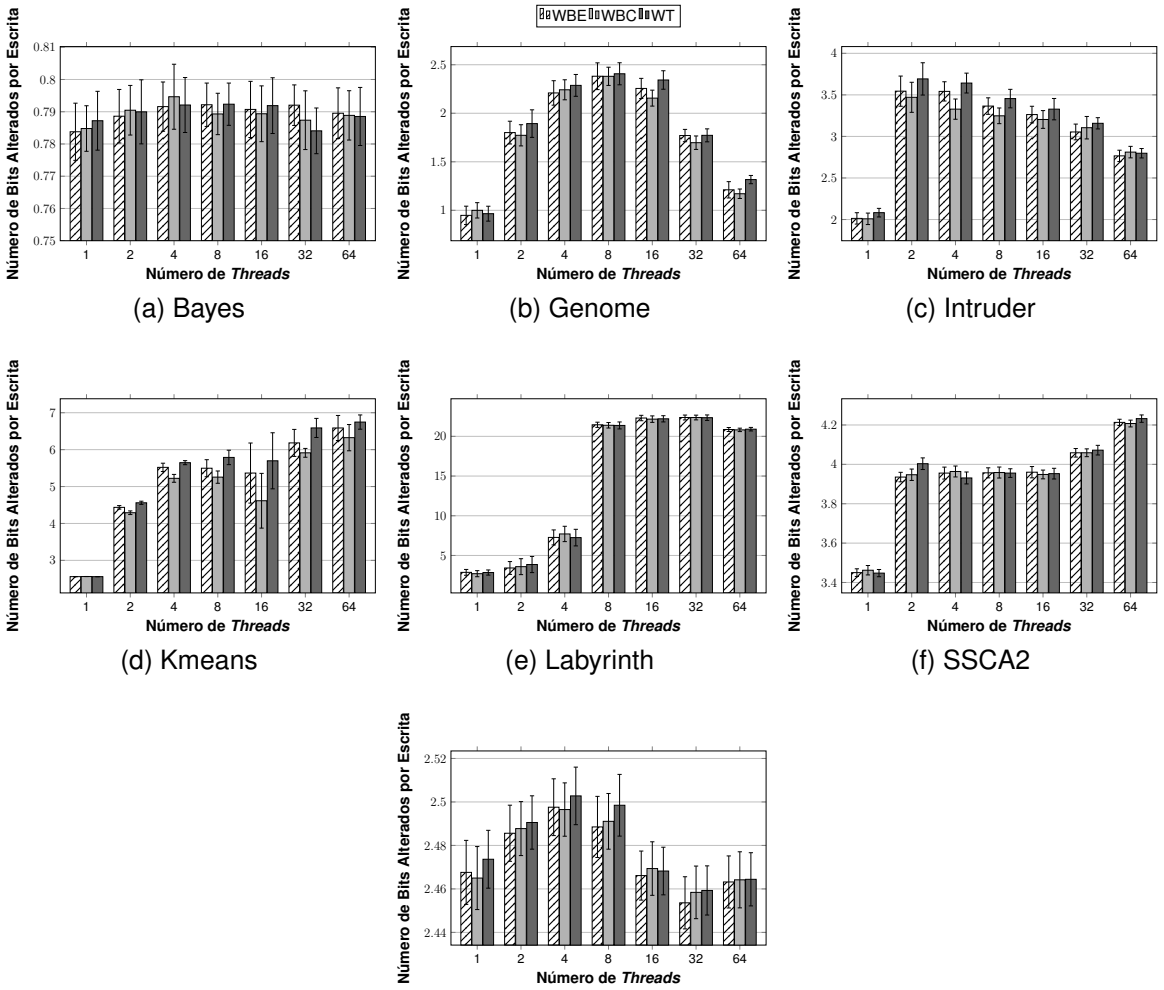
Figura 48: Bits Alterados na PCM do Experimento com o *Benchmark* Vacation

de *threads*. Após o cenário com 4 *threads*, houve uma diminuição na média de bits alterados a cada novo aumento do número de *threads*. Esse comportamento está relacionado com a média de escritas na PCM, pois obteve o mesmo comportamento conforme o aumento do número de *threads*.

Após analisar quantos bits foram alterados na PCM, foi feita uma análise da média de bits alterados por escrita na PCM. A Figura 49 apresenta os resultados deste experimento para cada *benchmark* do conjunto STAMP. Os resultados também apresentam o desvio padrão dos experimentos.

Analisando o experimento, podem ser vistos que os gráficos ficaram muito parecidos com os gráficos dos bits alterados respectivos de cada *benchmark*. Isto acontece devido a média de escritas na PCM dos diferentes versionamentos serem muito próximas, com exceção de alguns casos (como por exemplo a média de escritas do *benchmark* Genome com 32 e 64 *threads*). O *benchmark* Labyrinth foi quem apresentou a diferença mais significativa em relação ao gráfico que apresenta a média de bits alterados, como pode ser visto na Figura 49e. No cenário de execução com 64 *threads*, a média de bits aumentou, em relação a média com 32 *threads*, mas a média de bits alterados por escrita diminuiu. Isso aconteceu devido a porcentagem de aumento do número de escrita na PCM ter sido maior que a porcentagem de aumento da média de bits alterados, fazendo com que a média de bits alterados por escrita seja menor no cenário de execução com 64 *threads* em relação ao cenário com 32 *threads*.

Outra diferença que podemos ver da média de bits alterados e da média de bits alterados por escrita é no *benchmark* SSCA2. A média de bits alterados apresentou um comportamento que conforme o aumento do número de *threads*, a partir de 2 *threads*, ocorria uma diminuição da média de bits alterados, mas já a média de bits



(a) Bayes (b) Genome (c) Intruder

(d) Kmeans (e) Labyrinth (f) SSCA2

(g) Vacation

Figura 49: Bits Alterados por Escrita

alterados por escrita, para os mesmos cenários, apresentou um aumento, como pode ser visto na Figura 49f. Isso ocorreu devido a porcentagem de diminuição da média de escritas na PCM ter sido menor que a porcentagem de diminuição de bits alterados.

No *benchmark* Intruder no cenário de execução com 32 *threads*, os versionamentos mostraram ter muito pouca diferença entre eles em relação a média de bits alterados por escrita, ao contrário do que ocorreu com a média de bits alterados onde, o versionamento WT apresentou uma média maior de bits alterados. Isso ocorreu devido à média de escritas na PCM ter tido um comportamento muito similar ao de bits alterados, fazendo com que os resultados deste experimento sejam muito próximos.

Os experimentos com o *benchmark* Bayes, em todos os cenários de execução, e com o *benchmark* Genome, no cenário de execução com uma *thread*, apresentaram uma média de bits alterados por escrita menor que um, ou seja, nem toda a escrita modificou os bits. Isso ocorreu devido à ferramenta PinTools, utilizada para gerar os arquivos de traço, pegar o valor na memória antes que ocorra a escrita, ou seja, na primeira escrita de um endereço o valor pego foi zero e como o simulador inicia com

valor zero nos endereços da PCM, isso faz com que algumas escritas não alterem os bits, e nos casos citados o número de escritas que não modificaram bits foi maior que as que modificaram, fazendo com que a média de bits alterados por escrita ficasse abaixo de 1.

5.5 Consumo de Energia

Para estimar o consumo de energia foi utilizada a fórmula da Equação 1. O cálculo do consumo de energia foi feito da seguinte forma, foram utilizados o número de *sets*, de *resets* e de leituras que ocorreram na PCM e estes valores foram multiplicados pelos seus valores respectivos de consumo de energia em uma PCM (os valores do consumo de energia foram retirados do artigo Lee *et al* (2009)), por fim os resultados das multiplicações foram somados gerando o consumo de energia. A Figura 50 apresenta os resultados da estimativa do consumo de energia dos *benchmarks* do conjunto STAMP.

$$Consumo\ de\ Energia = (sets \times 13.5) + (resets \times 19.2) + (Leituras \times 2) \quad (1)$$

Assim como a Figura 49 que apresenta a média de bits alterados por escrita, a média de consumo de energia da PCM também apresentou um comportamento bem parecido com a média de bits alterados. Pode ser visto que apesar de o número de leituras na PCM ter tido um comportamento diferente, ele pouco interferiu nos resultados do consumo de energia. Isso ocorreu devido às leituras na PCM ter um consumo de energia muito menor que de um bit alterado. Para caracterizar isso, a Figura 51 apresenta a porcentagem respectiva do total do consumo de energia que tem os *Sets*, *Resets* e Leituras na PCM.

Na Figura 51 podemos ver que em todos *benchmarks* e experimentos, a porcentagem do consumo de energia de *Sets* e *Resets* foi muito maior que a porcentagem do consumo de energia das leituras. Isso confirma que o consumo de energia de uma memória PCM está diretamente ligada a suas escritas e que as leituras, em alguns casos, são até menores que 0,2%, como pode ser visto no *benchmark* Labyrinth (Figura 51e). O *benchmark* que apresentou a maior porcentagem de leituras foi o *benchmark* Genome, como pode ser visto na Figura 51b. No experimento com uma *thread*, em torno de 14% do consumo total veio das leituras, um valor maior até que a porcentagem de *Sets*, que ficou por volta de 5%.

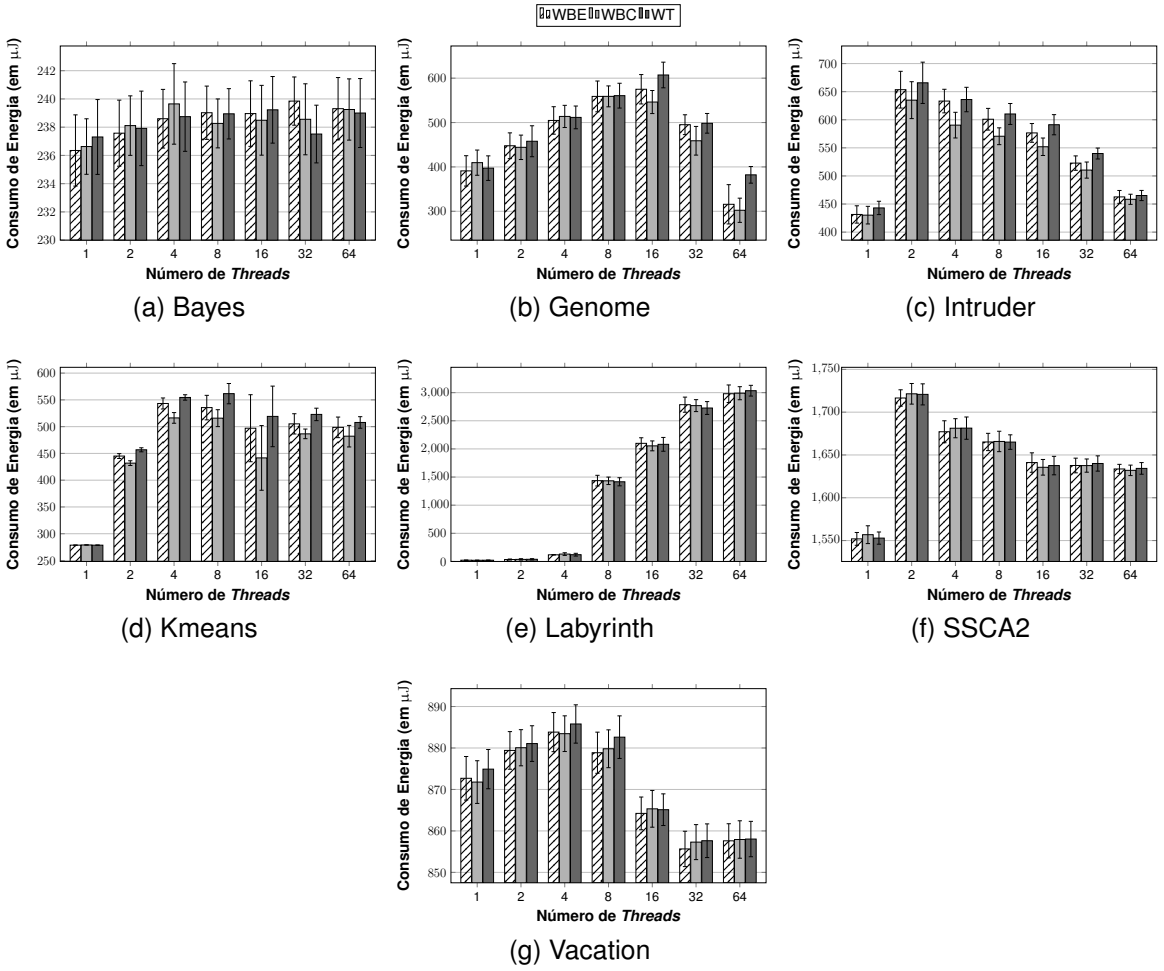


Figura 50: Estimativa do Consumo de Energia da PCM

5.6 Discussão

Com a análise dos resultados pode ser visto que alguns *benchmarks* do conjunto STAMP não apresentaram diferenças entre os versionamentos em relação ao desgaste da PCM. Os *benchmarks* que apresentaram isso foram o Bayes, Labyrinth, SSCA2 e Vacation. Esses *benchmarks*, ou apresentaram um número baixo de transações, que é o caso dos *benchmarks* Bayes e Labyrinth, ou apresentaram um número muito pequeno de *aborts* em relação ao número de transações, que foi o caso dos *benchmarks* SSCA2 e Vacation. Nesses *benchmarks* os versionamentos não apresentaram grandes diferenças, sendo considerados estatisticamente semelhantes.

Já nos outros *benchmarks*, Genome, Intruder e Kmeans, o versionamento WBC apresentou uma média menor de bits alterados na PCM, ou seja, apresentou o menor desgaste na PCM. Nos resultados, pode ser visto que na maioria dos experimentos com estes *benchmarks*, o versionamento WBC apresentou uma média menor de bits alterados que os outros versionamentos. Foi verificado que os resultados estão ligados ao número de *aborts* dos versionamentos. Esses *benchmarks* obtiveram um número

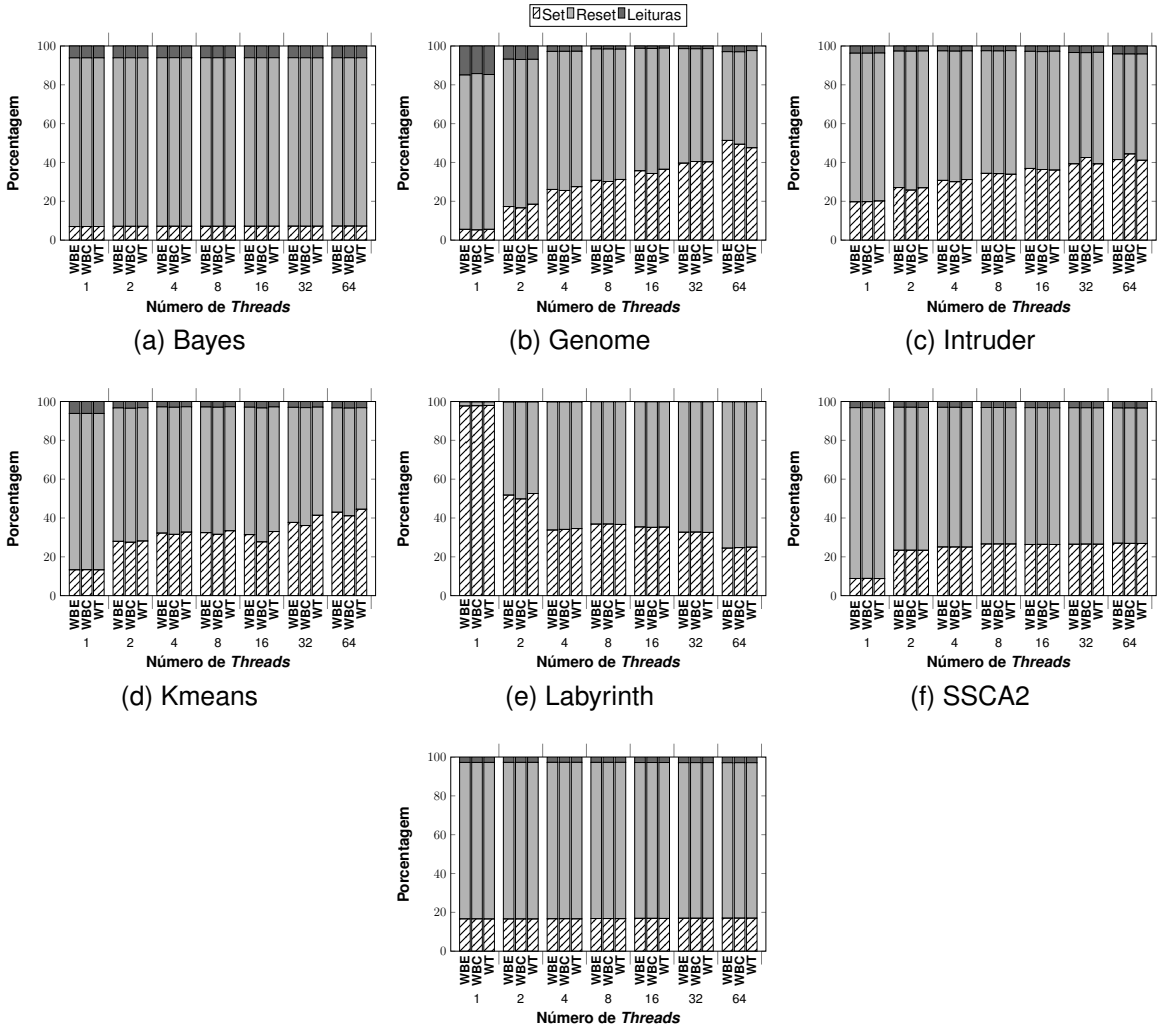


Figura 51: Porcentagem do consumo de energia de Sets, Resets e Leituras na PCM

de *aborts* grande, sendo até 39 vezes maior, fazendo assim com que o versionamento WBC desgastasse menos a PCM.

6 CONCLUSÃO

Neste trabalho, foi analisado o impacto das diferentes implementações de versionamentos de dados, de STM, em memórias PCM. O trabalho também apresentou um novo simulador de hierarquia de memória, implementado para este trabalho, chamado PCM-MS. O PCM-MS faz a simulação dos acessos a uma hierarquia de memória e implementa a possibilidade de simular arquiteturas com múltiplos núcleos de processamento. As principais contribuições deste trabalho foram apresentar a melhor opção para causar menos desgaste à memória PCM, apresentar uma caracterização dos versionamentos de dados e apresentar o simulador PCM-MS.

A ideia geral deste trabalho foi comparar os diferentes versionamentos de dados implementados em STM e seu impacto em uma memória PCM, ou seja, analisar o desgaste causado pelos diferentes versionamentos em uma memória PCM. Como dito, a biblioteca de STM utilizada neste trabalho foi a TinySTM, que faz parte do estado da arte de STM. Para a simulação foi utilizado o PCM-MS. Para gerar os arquivos de traço, para o simulador, foi utilizada a ferramenta Pintools (LUK et al., 2005). Como *benchmarks*, foram utilizados o Eigenbench, para fazer a caracterização dos versionamentos e entender o comportamento dos versionamentos nas diferentes características de aplicações que formam a base para o comportamento transacional, além do conjunto de *benchmarks* STAMP, que é um conjunto robusto de *benchmarks* que abrange uma ampla área de aplicações das STM. Os *benchmarks* do conjunto STAMP foram executados no simulador com os diferentes versionamentos como forma de analisar o desgaste na PCM, causados pelos versionamentos.

Analisando os resultados, em 3 dos 7 *benchmarks* utilizados para a análise do impacto, o versionamento WBC apresentou uma menor média de bits alterados na PCM, ou seja, apresentou o menor desgaste na PCM. Os *benchmarks*, Genome, Intruder e Kmeans, foram os *benchmarks* que apresentaram isso. Nos resultados pode ser visto que na maioria dos experimentos com estes *benchmarks*, o versionamento WBC apresentou uma média menor de bits alterados em relação aos outros versionamentos, ou seja, desgastou menos a memória PCM que os outros versionamentos. Foi verificado que os resultados estão ligados ao número de *aborts* dos versionamentos.

Nos demais *benchmarks*, Bayes, Labyrinth, SSCA2 e Vacation, os versionamentos não apresentaram diferenças significativas, sendo considerados em muitos cenários de execução estatisticamente semelhantes. Esses *benchmarks*, ou apresentaram um número baixo de transações (*benchmarks* Bayes e Labyrinth), o que faz com que não seja possível ver grandes diferenças entre os versionamentos, ou apresentaram um número muito pequeno de *aborts* em relação ao número de transações (*benchmarks* SSCA2 e Vacation), o que faz com que os versionamentos pouco interfiram na execução.

Os *benchmarks* em que o versionamento WBC apresentou os melhores resultados apresentaram uma alta contenção, obtendo um número de *aborts* grande. Isso fez com que o versionamento WBC, que apresenta um número menor de *aborts* em relação aos outros versionamentos, nestes ambientes (como pode ser visto na Figura 14), desgastasse menos a PCM.

6.1 Publicações

As publicações feitas durante o período do mestrado foram:

- TEIXEIRA, F. L.; PILLA, M. L.; DU BOIS, A. R. **Análise das escritas das Memórias Transacionais em relação ao tempo de execução**. 14° Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS), 2014
- TEIXEIRA, F. L.; PILLA, M. L.; DU BOIS, A. R. **Estimativa do Consumo de Energia das Escritas das STM em uma Memória PCM**. XVI Encontro de Pós-Graduação (ENPOS), 2014
- TEIXEIRA, F. L.; PILLA, M. L.; DU BOIS, A. R.; MOSSÉ, D. **Profiling Patterns of Bit Flipping for Software Transactional Memories**. IEEE 26th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2014
- TEIXEIRA, F. L.; PILLA, M. L.; DU BOIS, A. R. **Inversão de Bit uma Alternativa na Redução do Desgaste das Memórias PCM**. 15° Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS), 2015
- TEIXEIRA, F. L.; PILLA, M. L.; DU BOIS, A. R. **Análise Ortogonal dos Versionamentos Implementados pela TinySTM**. XVII Encontro de Pós-Graduação (ENPOS), 2015
- TEIXEIRA, F. L.; PILLA, M. L.; DU BOIS, A. R.; MOSSÉ, D. **Impact of Version Management on Transactional Memories' Performance**. 6th Workshop on Applications for Multi-Core Architectures (WAMCA), 2015

- TEIXEIRA, F. L.; PILLA, M. L.; DU BOIS, A. R. **Caracterização do Consumo de Energia dos Diferentes Versionamentos de Dados da biblioteca TinySTM em uma memória PCM**. 16° Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS), 2016

6.2 Trabalhos Futuros

Como trabalhos futuros pretende-se:

- Incrementar o simulador PCM-MS com diferentes escalonamentos entre as *threads* além implementar outros tipos de arquitetura de memória com a PCM sendo a memória principal, como, por exemplo, uma arquitetura com uma DRAM como *cache* da PCM;
- Analisar o desgaste na PCM dos diferentes gerenciadores de contenção implementados pela biblioteca TinySTM;
- Executar experimentos com os diferentes versionamentos e gerenciadores de contenção, no PCM-MS, com outras configurações de hierarquia de *cache*, para analisar o impacto das diferentes configurações no desgaste na PCM;
- Corrigir o *benchmark* Yada, e analisar o desgaste na PCM, dos diferentes versionamentos de dados e gerenciadores de contenção; e
- Analisar o desgaste na PCM de diferentes bibliotecas de STM.

REFERÊNCIAS

BADER, D. A.; MADDURI, K. Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors. In: HIGH PERFORMANCE COMPUTING, 12., 2005, Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2005. p.465–476. (HiPC'05).

BALDASSIN, A. J. **Explorando Memória Transacional em Software nos Contextos de Arquiteturas Assimétricas, Jogos Computacionais e Consumo de Energia**. 2009. Dissertação de Doutorado — Universidade Estadual de Campinas.

BEZDEK, J. C. **Pattern Recognition with Fuzzy Objective Function Algorithms**. Norwell, MA, USA: Kluwer Academic Publishers, 1981.

BOCK, S.; CHILDERS, B. R.; MELHEM, R. G.; MOSSÉ, D.; ZHANG, Y. Analyzing the impact of useless write-backs on the endurance and energy consumption of PCM main memory. In: ISPASS, 2011. **Anais...** IEEE Computer Society, 2011. p.56–65.

CAO MINH, C.; CHUNG, J.; KOZYRAKIS, C.; OLUKOTUN, K. STAMP: Stanford Transactional Applications for Multi-Processing. In: IISWC '08: PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON WORKLOAD CHARACTERIZATION, 2008. **Anais...** [S.l.: s.n.], 2008.

CHICKERING, D. M.; HECKERMAN, D.; MEEK, C. A Bayesian approach to learning Bayesian networks with local structure. In: IN PROCEEDINGS OF THIRTEENTH CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, 1997. **Anais...** Morgan Kaufmann, 1997.

CHO, S.; LEE, H. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In: MICROARCHITECTURE, 2009. MICRO-42. 42ND ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2009. **Anais...** [S.l.: s.n.], 2009. p.347–357.

CLICK, C. Azulś experiences with hardware transactional memory. In: TRANSACTIONAL MEMORY WORKSHOP, 2009. **Anais...** [S.l.: s.n.], 2009.

DICE, D.; LEV, Y.; MOIR, M.; NUSSBAUM, D.; OLSZEWSKI, M. **Early Experience with a Commercial Hardware Transactional Memory Implementation**. Mountain View, CA, USA: [s.n.], 2009.

DICE, D.; SHALEV, O.; SHAVIT, N. Transactional Locking II. In: DISC 2006, 2006. **Anais...** [S.l.: s.n.], 2006. p.194–208.

DU, Y.; ZHOU, M.; CHILDERS, B. R.; MOSSÉ, D.; MELHEM, R. Bit mapping for balanced PCM cell programming. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 40., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p.428–439. (ISCA '13).

FELBER, P.; FETZER, C.; RIEGEL, T. Dynamic Performance Tuning of Word-Based Software Transactional Memory. In: PPOPP '08: PROCEEDINGS OF THE 13TH ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.237–246.

FERREIRA, A. P.; ZHOU, M.; BOCK, S.; CHILDERS, B.; MELHEM, R.; MOSSÉ, D. Increasing PCM main memory lifetime. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2010, 3001 Leuven, Belgium, Belgium. **Proceedings...** European Design and Automation Association, 2010. p.914–919. (DATE '10).

HAAGDORENS, B.; VERMEIREN, T.; GOOSSENS, M. Improving the Performance of Signature-Based Network Intrusion Detection Sensors by Multi-threading. In: INFORMATION SECURITY APPLICATIONS, 2005. **Anais...** [S.l.: s.n.], 2005. p.188–203. (Lecture Notes in Computer Science (LNCS), v.3325).

HAMMARLUND, P.; MARTINEZ, A. J.; BAJWA, A. A.; HILL, D. L.; HALLNOR, E.; JIANG, H.; DIXON, M.; DERR, M.; HUNSAKER, M.; KUMAR, R.; OSBORNE, R. B.; RAJWAR, R.; SINGHAL, R.; D'SA, R.; CHAPPELL, R.; KAUSHIK, S.; CHENNUPATY, S.; JOURDAN, S.; GUNTHER, S.; PIAZZA, T.; BURTON, T. Haswell: The Fourth-Generation Intel Core Processor. **IEEE Micro**, Los Alamitos, CA, USA, v.34, n.2, p.6–20, 2014.

HARRIS, T.; LARUS, J.; RAJWAR, R. Transactional Memory, 2nd edition. **Synthesis Lectures on Computer Architecture**, [S.l.], v.5, n.1, p.1–263, 2010.

HENNING, J. L. SPEC CPU2006 Benchmark Descriptions. **SIGARCH Comput. Archit. News**, New York, NY, USA, v.34, n.4, p.1–17, Sept. 2006.

HERLIHY, M.; ELIOT, J.; MOSS, B. Transactional Memory: Architectural Support for Lock-Free Data Structures. In: PROCEEDINGS OF THE 20TH ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 1993. **Anais...** [S.l.: s.n.], 1993. p.289–300.

HOFFMAN, C. **Análise de desgaste de técnicas de correção de erros em Phase-Change Memories**. 2013. Master Thesis — UNICAMP.

HONG, S.; OGUNTEBI, T.; CASPER, J.; BRONSON, N.; KOZYRAKIS, C.; OLUKOTUN, K. Eigenbench: A simple exploration tool for orthogonal TM characteristics. In: WORKLOAD CHARACTERIZATION (IISWC), 2010 IEEE INTERNATIONAL SYMPOSIUM ON, 2010. **Anais...** [S.l.: s.n.], 2010. p.1–11.

KARP, R. M.; RABIN, M. O. Efficient Randomized Pattern-matching Algorithms. **IBM J. Res. Dev.**, Riverton, NJ, USA, v.31, n.2, p.249–260, Mar. 1987.

KRONBAUER, F.; RIGO, S. Experimentos com Gerenciamento de Contenção em uma Memória Transacional com Suporte em Software. **Anais do X Simpósio em Sistemas Computacionais WSCAD-SSC**, São Paulo, SP, Brasil, p.44–51, 2009.

KULKARNI, M.; CHEW, L. P.; PINGALI, K. Using Transactions in Delaunay Mesh Generation. In: WTW'06: PROCEEDINGS OF THE WORKSHOP ON TRANSACTIONAL MEMORY WORKLOADS, 2006, Ottawa, Canada. **Anais...** [S.l.: s.n.], 2006. p.23–31. (Held in conjunction with PLDI 2006).

LE, H.; GUTHRIE, G.; WILLIAMS, D.; MICHAEL, M.; FREY, B.; STARKE, W.; MAY, C.; ODAIRA, R.; NAKAIKE, T. Transactional memory support in the IBM POWER8 processor. **IBM Journal of Research and Development**, [S.l.], v.59, n.1, p.8:1–8:14, Jan 2015.

LEE, B. C.; IPEK, E.; MUTLU, O.; BURGER, D. Architecting phase change memory as a scalable dram alternative. In: COMPUTER ARCHITECTURE, 36., 2009, New York, NY, USA. **Proceedings...** ACM, 2009. p.2–13. (ISCA '09).

LEE, B. C.; ZHOU, P. Z. P.; YANG, J. Y. J.; ZHANG, Y. Z. Y.; ZHAO, B. Z. B.; IPEK, E.; MUTLU, O.; BURGER, D. Phase-Change Technology and the Future of Main Memory. **IEEE Micro**, [S.l.], v.30, n.1, p.131–141, 2010.

LEE, C. Y. An Algorithm for Path Connections and Its Applications. **Electronic Computers, IRE Transactions on**, [S.l.], v.EC-10, n.3, p.346–365, 1961.

LI, Y.; LI, X.; JU, L.; JIA, Z. A three-stage-write scheme with flip-bit for PCM main memory. In: DESIGN AUTOMATION CONFERENCE (ASP-DAC), 2015 20TH ASIA AND SOUTH PACIFIC, 2015. **Anais...** [S.l.: s.n.], 2015. p.328–333.

LIU, D.; WANG, T.; WANG, Y.; SHAO, Z.; ZHUGE, Q.; SHA, E. Curling-PCM: Application-specific wear leveling for phase change memory based embedded systems. In: DESIGN AUTOMATION CONFERENCE (ASP-DAC), 2013 18TH ASIA AND SOUTH PACIFIC, 2013. **Anais...** [S.l.: s.n.], 2013. p.279–284.

LUK, C.-K.; COHN, R.; MUTH, R.; PATIL, H.; KLAUSER, A.; LOWNEY, G.; WALLACE, S.; JANAPA, V.; HAZELWOOD, R. K. Pin: Building customized program analysis tools with dynamic instrumentation. In: IN PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, 2005. **Anais...** ACM Press, 2005. p.190–200.

MARATHE, V. J.; MOIR, M. Efficient Nonblocking Software Transactional Memory. In: ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING, 12., 2007, New York, NY, USA. **Proceedings...** ACM, 2007. p.136–137. (PPoPP '07).

MARATHE, V. J.; SPEAR, M. F.; HERIOT, C.; ACHARYA, A.; EISENSTAT, D.; III, W. N. S.; SCOTT, M. L. Lowering the overhead of nonblocking software transactional memory. In: FIRST ACM SIGPLAN WORKSHOP ON LANGUAGES, COMPILERS, AND HARDWARE SUPPORT FOR TRANSACTIONAL COMPUTING, 2006. **Anais...** [S.l.: s.n.], 2006.

MOORE, A.; LEE, M. S. Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. **Journal of Artificial Intelligence Research**, [S.l.], v.8, p.67–91, 1997.

MORESHET, T.; BAHAR, R. I.; HERLIHY, M. Energy-Aware Microprocessor Synchronization: Transactional Memory vs. Locks. In: WORKSHOP ON MEMORY PERFORMANCE ISSUES, 2006. **Proceedings...** [S.l.: s.n.], 2006.

PATTERSON, D. A.; HENNESSY, J. L. **Computer organization and design**: the hardware/software interface. [S.l.]: Newnes, 2013.

PISHARATH, J.; LIU, Y.; LIAO, W.-k.; MEMIK, G.; CHOUDHARY, A.; DUBEY, P. **NU-MineBench**: Understanding the Performance and Scalability Characteristics of Data Mining Algorithms. [S.l.]: Technical Report CUCIS-2004-05-001, Center for Ultra-Scale Computing and Information Security, Northwestern Univ, 2004.

QURESHI, M. K.; SRINIVASAN, V.; RIVERS, J. A. Scalable high performance main memory system using phase-change memory technology. In: COMPUTER ARCHITECTURE, 36., 2009, New York, NY, USA. **Proceedings...** ACM, 2009. p.24–33. (ISCA '09).

RAOUX, S.; BURR, G.; BREITWISCH, M.; RETTNER, C.; CHEN, Y.; SHELBY, R.; SALINGA, M.; KREBS, D.; CHEN, S.-H.; LUNG, H. L.; LAM, C. Phase-change random access memory: A scalable technology. **IBM Journal of Research and Development**, [S.l.], v.52, n.4.5, p.465–479, July 2008.

RICO, T. M. **Análise de Consumo de Energia e Desempenho de Memórias Transacionais em Software em Ambiente de Computação Real**. 2013. Master Thesis — Federal University of Pelotas.

RIGO, S.; CENTODUCATTE, P.; BALDASSIN, A. **Memorias Transacionais**: Uma Nova Alternativa para Programação Concorrente. [S.l.]: In Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing, 2007.

RUAN, W.; LIU, Y.; SPEAR, M. STAMP Need Not Be Considered Harmful. In: ACM SIGPLAN WORKSHOP ON TRANSACTIONAL COMPUTING, 9., 2014, New York. **Anais...** ACM, 2014. p.1–7.

RUPPERT, J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. **J. Algorithms**, Duluth, MN, USA, v.18, n.3, p.548–585, May 1995.

SHUM, C. K.; BUSABA, F.; JACOBI, C. IBM zEC12: The Third-Generation High-Frequency Mainframe Microprocessor. **IEEE Micro**, Los Alamitos, CA, USA, v.33, n.2, p.38–47, 2013.

SILBERSCHATZ, A. **Fundamentos de sistemas operacionais**. 8.ed. [S.l.]: LTD, 2010.

TEIXEIRA, F. L.; PILLA, M. L.; BOIS, A. R.; MOSSE, D. Profiling Patterns of Bit Flipping for Software Transactional Memories. In: COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING (SBAC-PAD), 2014 IEEE 26TH INTERNATIONAL SYMPOSIUM ON, 2014. **Anais...** [S.l.: s.n.], 2014. p.136–143.

WANG, F.; WU, X. Non-volatile Memory Devices Based on Chalcogenide Materials. **Information Technology: New Generations, Third International Conference on**, Los Alamitos, CA, USA, v.0, p.5–9, 2009.

XIA, F.; JIANG, D.-J.; XIONG, J.; SUN, N.-H. A Survey of Phase Change Memory Systems. **Journal of Computer Science and Technology**, [S.l.], v.30, n.1, p.121–144, 2015.

YUE, J.; ZHU, Y. Accelerating write by exploiting PCM asymmetries. In: HIGH PERFORMANCE COMPUTER ARCHITECTURE (HPCA2013), 2013 IEEE 19TH INTERNATIONAL SYMPOSIUM ON, 2013. **Anais...** [S.l.: s.n.], 2013. p.282–293.

APÊNDICE A RESULTADOS *SCALABILITY*

Tabela 5: Tabela com a média dos tempos de execução do experimento *Scalability*

Número de Threads	1	2	4	8
Sequencial	1417,7265333333	2831,7273333333	5646,9398333333	11275,7438666667
WBE	2065,4790333333	2226,0029333333	2352,9140333333	4283,2514
WBC	2429,7878666667	2673,7197	2837,5266666667	4650,683
WT	2031,0354666667	2185,6280666667	2320,3138333333	4226,7315
unp	1419,3260666667	1476,3956333333	1537,1565666667	2876,1956333333

Tabela 6: Tabela com o desvio padrão dos tempos de execução do experimento *Scalability*

Número de Threads	1	2	4	8
Sequencial	5,1754263841	7,2237574965	12,0045170466	17,9528796456
WBE	31,4701263713	34,8140201216	12,2818199047	20,8249933288
WBC	74,2675360696	3,8793134226	27,5438867356	7,0561171065
WT	26,3093468532	13,6840830414	21,4607554967	4,8267565732
unp	4,5928455302	102,6847487996	6,2123503114	4,5323565651

APÊNDICE B RESULTADOS *CONTENTION*

Tabela 7: Tabela com a média dos tempos de execução do experimento *Contention*

A1/(A1+A2)	0.0312	0.039	0.0625	0.0971	0.125	0.156	0.187	0.218
Sequencial	6491.8360666667	6485.4937666667	6489.7753666667	6486.334	6488.9055333333	6494.5051333333	6488.2614333333	6489.2919333333
WBE	20506.4310333333	16592.1758666667	9967.0228333333	6396.9781333333	5191.0369666667	4419.0425	3944.1774666667	3603.4120666667
WBC	6739.4225333333	6054.7467666667	4899.8155333333	4077.7705	3721.4102	3454.3725333333	3273.8007	3129.4386
WT	29434.6536	20253.4971333333	10388.2139666667	6400.2592333333	5154.5633666667	4379.3751333333	3903.1088666667	3578.2382333333
A1/(A1+A2)	0.246	0.309	0.375	0.5	0.625	0.75	0.875	
Sequencial	6488.5029666667	6485.2528333333	6485.3603666667	6487.7947333333	6488.0256	6486.2644666667	6490.6010666667	
WBE	3403.5621	3074.4277333333	2897.954	2644.2446333333	2494.8631	2398.2820333333	2347.4445	
WBC	3029.0961666667	2893.5640333333	2786.2576666667	2640.5976666667	2555.3928333333	2498.7119333333	2436.8134	
WT	3375.1901666667	3059.2526	2845.9657666667	2598.8528333333	2460.5705666667	2362.4437	2294.0970333333	

Tabela 8: Tabela com o desvio padrão dos tempos de execução do experimento *Contention*

A1/(A1+A2)	0.0312	0.039	0.0625	0.0971	0.125	0.156	0.187	0.218
Sequencial	9.3268861675	12.1055633169	6.1249211028	10.3773725696	10.1569435388	10.603051476	6.6447120209	22.1263615469
WBE	630.7262963904	594.7185330254	77.5654972595	32.914393646	37.0679156056	51.8337992532	56.4424503114	5.587937507
WBC	39.1317484948	41.0782116814	26.8144357772	12.9523812609	24.2233034585	22.6716321602	39.6671540361	21.9810265685
WT	1512.0199075104	715.4712212761	68.4039884699	25.0517163342	51.844512407	37.0835703241	38.4102468259	39.423088032
A1/(A1+A2)	0.246	0.309	0.375	0.5	0.625	0.75	0.875	
Sequencial	7.3714603779	5.1373553178	9.3666572129	7.1305896643	9.4766252126	3.6859839722	32.1621873065	
WBE	36.890427141	3.9606214327	80.7242621457	50.2460849149	8.0813291897	10.2387889622	85.225168994	
WBC	11.7547253597	68.1221743642	86.0503812131	31.7429802974	28.83605328	35.7004647926	3.9483314555	
WT	64.2283417789	57.4723027731	48.132935924	5.7112676477	11.8689799541	6.1983719395	18.6505450757	

APÊNDICE C RESULTADOS *DENSITY*

Tabela 9: Tabela com a média dos tempos de execução do experimento *Density*

Porcentagem de R3i	0	10	20	30	40	50
Sequencial	10995.2823333333	11014.4981	11085.4640666667	11067.9535666667	11078.6052666667	11075.2741
WBE	4618.9372666667	4895.5049	5055.7524333333	5199.5532333333	5342.4639666667	5487.6871666667
WBC	4308.1379666667	4481.6526666667	4568.0634333333	4644.8730333333	4730.1473333333	4798.8205333333
WT	4569.2917666667	4839.8763666667	5006.685	5149.6342	5300.0494666667	5441.2763
Porcentagem de R3i	60	70	80	90	100	
Sequencial	11084.3327	11090.6076333333	11084.9615666667	11083.3464	11088.8382666667	
WBE	5674.4507333333	5816.6107666667	5956.6633333333	6111.8011333333	6293.3524333333	
WBC	4881.5079333333	4968.4192666667	5044.9182666667	5114.9730666667	5173.8871	
WT	5599.4621	5751.4314666667	5902.9500666667	6061.0569333333	6230.8195333333	

Tabela 10: Tabela com o desvio padrão dos tempos de execução do experimento *Density*

Porcentagem de R3i	0	10	20	30	40	50
Sequencial	71.8532787203	47.0179205633	27.2435159966	21.661552768	16.0492583563	30.3898890971
WBE	32.5477366584	28.8270057296	26.9276070638	26.7237972816	28.1602921009	20.3092115521
WBC	13.5211434032	38.4766024293	34.5871730476	47.8827772784	49.3209088691	37.9510575054
WT	21.4093992615	25.8851219169	18.1995934797	25.6062815346	38.0309827611	24.7710968395
Porcentagem de R3i	60	70	80	90	100	
Sequencial	17.1328149571	24.1704086472	18.3103286448	19.0770638321	17.3597197398	
WBE	87.228597467	59.9105637681	37.2267260579	21.8634230603	61.1108445236	
WBC	44.2877708418	39.5701599824	17.818637426	11.6473551709	23.060352211	
WT	30.5150721858	27.2167500856	15.5819012671	28.8298202082	41.5370799413	

APÊNDICE D RESULTADOS *TRANSACTION LENGHT*

Tabela 11: Tabela com a média dos tempos de execução do experimento *Transaction Lenght*

Comprimento das Transações	10	20	30	40	80	120	160	200
Sequencial	4186.8688	8758.7875666667	13310.2416333333	17853.6314333333	9029.1693333333	13549.0735333333	18090.3729	22632.6070333333
WBE	1732.2064333333	3533.3835333333	5250.6443666667	6987.7272	3437.6265333333	5131.3376	6823.0345333333	8526.5190666667
WBC	1750.6353333333	3613.0277666667	5411.8272666667	7213.8730333333	3666.0821333333	5713.2891666667	7908.4278666667	10332.1581
WT	1711.2739333333	3539.3237	5250.482	6941.7166333333	3400.3524333333	5066.285	6736.6764	8424.9852666667
Comprimento da Transação	240	280	320	360	400	440	480	520
Sequencial	27192.3156333333	31729.7764666667	9067.2756333333	10206.2167333333	11340.7644	12482.7564333333	13611.1449666667	14749.8418333333
WBE	10232.6775	11945.0485	3422.4515	3850.5742666667	4270.0813333333	4716.4638333333	5119.8723666667	5553.0109
WBC	12964.2602333333	15895.7783666667	4767.5467333333	5608.2756666667	6511.9575666667	7435.2093333333	8398.8551	9403.6953
WT	10100.9822	11794.0167	3373.4981666667	3793.4515666667	4256.2950666667	4630.7641666667	5047.5609666667	5471.6051

Tabela 12: Tabela com o desvio padrão dos tempos de execução do experimento *Transaction Lenght*

Comprimento das Transações	10	20	30	40	80	120	160	200
Sequencial	8.7204185537	14.4946262258	15.1920432883	28.2266350868	67.9828269189	16.3858754835	20.437592009	28.8779444434
WBE	3.8661183392	5.059110258	6.3190932429	67.7965326147	14.6275842084	23.1941137756	6.9274214625	32.3451185678
WBC	21.4135205811	35.4426727788	30.6790349238	38.3432317972	5.3276578326	73.6406448538	55.4666771406	67.3575633985
WT	9.1300233067	4.7093143959	23.7145791299	28.1011927818	18.4106076558	25.4613193458	27.6427120751	61.7223973046
Comprimento da Transação	240	280	320	360	400	440	480	520
Sequencial	51.3327870768	36.713201271	12.2750214846	14.2781918611	19.6373788942	23.8417870261	22.2490741682	25.2666293988
WBE	53.2569778098	11.7060106834	24.2254823172	32.8707883595	7.347527979	46.9762189786	5.6136741886	29.4707910971
WBC	26.7356260597	37.4625825513	39.1746511231	19.5183321174	66.8985579736	111.6357337592	16.2289505203	21.0660122262
WT	77.0016158157	17.3037992622	19.3693016805	21.443398511	200.6847178768	16.8563164035	8.3296049312	27.39640683

APÊNDICE E RESULTADOS *TEMPORAL LOCALITY*

Tabela 13: Tabela com a média dos tempos de execução do experimento *Temporal Locality*

LCT	0	0.125	0.25	0.375	0.5
Sequencial	11286.9088333333	15041.3578666667	16422.9587666667	17888.6629	18884.6083
WBE	4281.2875333333	5234.0420333333	5426.0722333333	5607.3913	5715.0844666667
WBC	4676.9077666667	5665.6454333333	5857.5467	6047.7161333333	6209.4328666667
WT	4230.6253666667	5171.1114333333	5360.8445	5546.0767333333	5671.1642
LCT	0.625	0.75	0.875	1	
Sequencial	18264.3363333333	16959.2464666667	15679.4591	14300.7607333333	
WBE	5679.2795666667	5623.2724666667	5522.7975	4579.3282333333	
WBC	6134.7249666667	5967.2765666667	5769.9952666667	4213.2499	
WT	5641.6533666667	5579.6582	5465.0866	4413.1580666667	

Tabela 14: Tabela com o desvio padrão dos tempos de execução do experimento *Temporal Locality*

LCT	0	0.125	0.25	0.375	0.5
Sequencial	18.8934348862	97.4302023441	91.7188011996	84.8533647525	102.5262179304
WBE	5.7829978309	19.1189889602	15.0172793823	15.6674585161	14.8481154005
WBC	74.6141114876	52.6914766684	19.3798701682	7.8125734589	111.6450232866
WT	16.5837486476	19.6491844977	15.4041814134	4.7079619936	9.0287920008
LCT	0.625	0.75	0.875	1	
Sequencial	97.5151706653	129.0022525547	177.5489835272	167.1909746157	
WBE	7.0619471462	10.1648736062	15.716288124	13.6748348327	
WBC	87.9030588281	8.071846525	62.8922360298	40.8347825928	
WT	15.866418481	7.0652444056	5.6612512057	7.6589178984	

APÊNDICE F RESULTADOS *POLLUTION*

Tabela 15: Tabela com a média dos tempos de execução do experimento *Pollution*

Porcentagem de W2	0	10	20	30	40	50
Sequencial	10234.5842333333	11288.7852333333	12511.1050666667	13803.6357666667	15076.4382333333	15783.8051666667
WBE	3921.4647666667	4280.8632333333	4494.679	4725.5471	4964.1191666667	5151.0631333333
WBC	3943.8979	4676.4043333333	5432.1809333333	6236.4528333333	7104.7728333333	7827.2649666667
WT	3859.5229666667	4225.4026	4418.7348333333	4612.5269	4809.0783	4950.9363
Porcentagem de W2	60	70	80	90	100	
Sequencial	15221.6883	14067.5443	12942.6109	11890.7545666667	10872.4100333333	
WBE	5179.7979666667	5117.9955666667	5070.1943	5030.1207666667	4982.7481	
WBC	8389.3696	8831.1965333333	9256.2232666667	9651.4379333333	10058.6937666667	
WT	4929.9518333333	4834.7286333333	4750.0774333333	4674.5342	4573.8054	

Tabela 16: Tabela com o desvio padrão dos tempos de execução do experimento *Pollution*

Porcentagem de W2	0	10	20	30	40	50
Sequencial	21.9407733002	19.8183270905	23.071766983	26.490516192	16.7932597597	25.3596956811
WBE	18.9706083332	13.9936087275	8.7857909306	24.7022425777	26.3816766677	22.2431699331
WBC	29.8102154596	70.6978947189	87.5978890286	88.8394745645	96.6398850665	16.443753692
WT	4.3565860605	3.4073354804	32.2194972275	30.9726672589	7.9404962633	5.6687108195
Porcentagem de W2	60	70	80	90	100	
Sequencial	36.643381718	34.484301686	60.6675699384	18.7960151604	5.4271617621	
WBE	22.4418350308	11.9137526549	32.5757206035	6.4871439903	34.0136535398	
WBC	20.6565368006	8.8368461792	51.6291121725	83.7014348198	90.0258720482	
WT	16.7130757298	7.6141560969	9.0779861837	13.0720626302	7.6749274088	

APÊNDICE G RESULTADOS *PREDOMINANCE*

Tabela 17: Tabela com a média dos tempos de execução do experimento *Predominance*

Percentual de Acessos ao *Array* 2	0.047	0.09	0.166	0.333
Sequencial	205671.973766667	108524.018433333	59931.8348	30745.5920333333
WBE	56341.8388666667	29993.0474333333	16915.5165666667	9146.0819666667
WBC	56873.4900333333	30341.0660666667	17263.2589333333	9535.1363666667
WT	56269.2266	29940.4689	16837.4529	9082.0806666667
Percentual de Acessos ao *Array* 2	0.5	0.8	0.833	0.909
Sequencial	21005.3437333333	13693.8334333333	13181.9427	12190.3321666667
WBE	6627.2905333333	4765.9552333333	4649.5143	4394.1647666667
WBC	7013.0428	5164.7388	5069.4566	4810.2584333333
WT	6563.5621666667	4726.3778333333	4602.8868	4349.0336333333

Tabela 18: Tabela com o desvio padrão dos tempos de execução do experimento *Predominance*

Percentual de Acessos ao *Array* 2	0.047	0.09	0.166	0.333
Sequencial	22.0972354484	60.4302688272	30.2143619965	51.0831137282
WBE	8.8844446656	35.9108113331	55.5670801934	34.5755801731
WBC	1326.034065447	168.4670684592	58.4777949983	10.2378184788
WT	65.4178911606	188.8991518849	66.5617062983	47.9650432253
Percentual de Acessos ao *Array* 2	0.5	0.8	0.833	0.909
Sequencial	21.1842372271	22.6191036008	21.0025597802	18.0904833871
WBE	10.1610419902	8.3722481869	25.0435863868	20.3541079742
WBC	28.661937789	58.8588764538	100.5150035826	89.3147733985
WT	15.8507248134	29.5613022811	23.2610834737	9.1247286081

APÊNDICE H RESULTADOS *WORKING-SET SIZE*

Tabela 19: Tabela com a média dos tempos de execução do experimento *Working-set Size*

KB/*Threads*	8	16	32	64	128
Sequencial	11098.1827333333	11095.6641	11110.0255333333	11167.5286333333	11199.8237333333
WBE	3983.8755	3989.8432666667	4006.2682333333	4063.2561666667	4133.9186333333
WBC	4464.9620333333	4455.9202	4446.7726666667	4472.4510666667	4523.9693333333
WT	3967.0402333333	3948.5975333333	3982.7823	4014.5909666667	4093.9575333333
KB/*Threads*	256	512	1024	2048	4096
Sequencial	11287.4628333333	11490.3544333333	11554.2424	11593.7846666667	12188.9073
WBE	4282.558	4433.9913	6081.8135666667	9152.4691	11055.5256666667
WBC	4658.4026333333	4799.8108	6404.9951	9407.8791333333	11247.5078666667
WT	4226.245	4374.1121666667	6098.1878666667	9273.1161	11111.2418666667
KB/*Threads*	8192	16384	32768	65536	131072
Sequencial	16419.1769	3346.5885	3912.2416333333	4210.5375666667	4437.3163666667
WBE	11719.1982333333	1530.0247333333	1624.8921	1730.6701666667	1872.9561666667
WBC	11895.4757666667	1559.1432666667	1602.622	1692.9299	1901.1000333333
WT	11885.5369666667	1562.9948333333	1589.3438	1768.8678	1872.8823

Tabela 20: Tabela com o desvio padrão dos tempos de execução do experimento *Working-set Size*

KB/*Threads*	8	16	32	64	128
Sequencial	9.2743129941	6.7497112191	5.4895362767	13.1852697342	10.0452275199
WBE	19.2825679159	36.9298569546	27.5181397084	46.08499955	8.0091917259
WBC	72.9769458861	71.3610469129	18.0981452133	14.4997707733	7.7215624037
WT	31.6332164517	17.3583201808	71.1440089741	12.7295731529	6.8817236951
KB/*Threads*	256	512	1024	2048	4096
Sequencial	14.6014847287	25.1022513246	86.7384979952	49.3972101413	115.4195673622
WBE	20.1485582645	26.5703437614	45.4421328889	13.7569904221	9.9777766245
WBC	47.5353573368	44.8016732318	49.3316837947	27.1263707853	8.5206997435
WT	4.9631454987	17.9700988121	32.4145598179	19.385092926	27.0167774907
KB/*Threads*	8192	16384	32768	65536	131072
Sequencial	216.7448351098	7.1770049403	10.5910743895	13.2564165873	16.2829803433
WBE	35.43065153	5.5061198174	215.0586065899	220.5865361668	3.8961456558
WBC	49.2363920151	36.0117379886	8.5424267932	31.3758143231	73.3797899549
WT	629.3206480426	107.6371334294	6.553759839	377.5416181727	4.4335433961

APÊNDICE I RESULTADOS MÉDIA DE ACESSOS À MEMÓRIA

Tabela 21: Tabela com a média de acessos dos experimentos com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	615044279,833333	615711612,733333	617839464,607143	616833673,75	621246496,5	621033909,05	627030070,411765
WBC	615061664	614636011,833333	617706115,809524	616592078,695652	621089551,185185	620452400,15	625225875,733333
WT	615013306,4	614279415,366667	617978642,259259	616617236,103448	621284146,678572	620810221,6	626542375,08

Tabela 22: Tabela com o desvio padrão do número de acessos dos experimentos com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	12,8467984294	1078483,04580724	352594,88064458	1081246,83297929	1731379,71711038	1547886,58981143	2962592,30332759
WBC	51,619964061	991480,640930004	442089,54685964	1297620,75063204	1591268,37634772	2190583,93285192	3455863,50482064
WT	96,707022888	565939,494945473	795893,49998262	1145899,02629898	1741866,64772879	2076043,22300404	3781514,58234839

Tabela 23: Tabela com a média de acessos dos experimentos com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	884272837,3	807850128,766667	805603273,433333	967121150,6	1466211921,23333	2599164048,63333	5573709903,83333
WBC	923935373,3	843326882,266667	840135224,033333	992748072,566667	1519399605,8	2690671136,96667	5795817530,13333
WT	758300225,666667	701156859,966667	700279951,166667	831788381,666667	1261619255,53333	2236086270,13333	4872630608,16667

Tabela 24: Tabela com o desvio padrão do número de acessos dos experimentos com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	14,2154602579	2764247,57349939	4356472,09441574	18006341,8699988	38910805,479273	68493181,3144056	335056472,54113
WBC	10,8981175217	3402491,09895974	5134784,45545881	16815469,2438973	22723087,8945299	75136642,7763473	169653236,607503
WT	11,8127147551	3411438,20150836	3912254,71958578	15266994,5902661	29811533,6898922	74206191,6924061	174809640,439385

Tabela 25: Tabela com a média de acessos dos experimentos com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	868018401,266667	981324717,666667	1161353091,23333	1547292587,36667	2427466657,83333	4251550588,9	7680826782,13333
WBC	939842757,766667	1106372818,43333	1456781118,36667	1596369457,76667	2296424234,3	3059448817,8	4510445704,93333
WT	760393366,366667	863654851,566667	1026440871,96667	1366607496,9	2146943467,63333	3735823656,14815	6857178518,03448

Tabela 26: Tabela com o desvio padrão do número de acessos dos experimentos com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	12,7791137066	2457239,41551111	5077180,0321853	9735029,68469457	20294456,4607282	55152490,7796125	129996164,295288
WBC	10,6047592872	2940719,05927129	5326050,97519384	88505196,4261661	144531686,148926	194719177,566081	244431880,532879
WT	12,9547843202	3479993,0222642	5595162,91515468	12219704,3842192	89419704,7889236	59224535,,0679057	183774480,551684

Tabela 27: Tabela com a média de acessos dos experimentos com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	1392920398,1	1429209961,26667	1515476579,43333	1675146194,6	1911613550,66667	2743712369,4	3477742594,6
WBC	1507870557,76667	1550380559,3	1617660953,46667	1741541446,9	2083389596,2	2556022368,43333	2998249954,8
WT	1325737133,7	1350009854	1405112176,33333	1529820704	1764263235,7	2512821070,7	3297328533,3

Tabela 28: Tabela com o desvio padrão do número de acessos dos experimentos com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	11,7747538227	10111607,2012235	47200012,8630365	73967975,6245668	58763193,8203117	89451102,0451041	135947467,964505
WBC	12,1248770583	318032,541974728	4695857,54734858	25799735,9917529	55574518,5711296	74182682,0292831	92556087,4264739
WT	9,2443831968	1690635,15502129	10887226,7037883	73428078,5538141	68041421,4141764	90036892,7898438	51935532,8810219

Tabela 29: Tabela com a média de acessos dos experimentos com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	643471592,966667	683634101,233333	765620923,5	887844663,566667	1198350273,96296	1576273910,2	1873979190,57692
WBC	644565458,4	687615165,266667	755915055,766667	907807442,666667	1179577312,34483	1570632828,19231	1843636554,89655
WT	643336252,566667	685174521,8	760840024,8	883485520,8	1183135427,72	1533735463,86207	1863871814,10345

Tabela 30: Tabela com o desvio padrão do número de acessos dos experimentos com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	13,9962638693	10712547,2430383	19581931,4837921	33361180,3243285	42032035,6396902	70821548,7001072	86701368,8645201
WBC	12,153047014	7414762,57127269	16828125,0282943	39148149,5233415	58440764,6049687	84345410,0723371	103020126,161703
WT	13,6626430859	10888671,5153405	23565086,1817543	25437271,0279046	53496230,0873244	70237077,6733608	83882425,8414726

Tabela 31: Tabela com a média de acessos dos experimentos com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	403300097,266667	405765816,066667	409290353,2	413222755,8	420207377,433333	434311425,533333	462803468,533333
WBC	404701561,666667	407689839,4	411717318,633333	415954707,366667	423072360,966667	437422601,033333	466401683
WT	393489876,7	395980013,966667	399473717,833333	403416102,9	410434323,533333	424280641,266667	452340850,6

Tabela 32: Tabela com o desvio padrão do número de acessos dos experimentos com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	447,658962969	44910,9610228439	244130,838404525	26255,7406785509	61591,2344601323	481083,680267196	1857931,96608966
WBC	12,2455715664	18120,7724993283	34114,2968567921	29302,779380336	116553,863542964	906611,418092725	2618960,34159817
WT	22,0722168941	23642,8666875603	262737,878634947	21310,3819936151	171686,62704529	549119,391509397	3314419,00224583

Tabela 33: Tabela com a média de acessos dos experimentos com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	776589445,466667	785129287,566667	790318720,4	792386068,2	794456077,633333	794335232,966667	795038396,166667
WBC	792703989,4	801728693,833333	807203633,6	809224407,533333	811487154,733333	811181572,1	811604472,066667
WT	742749520,1	751337224,7	756530158,233333	758350804,833333	760291272,866667	760167252,1	760760865,733333

Tabela 34: Tabela com o desvio padrão do número de acessos dos experimentos com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	11,162539982	166303,292358743	96397,6262733934	95103,958535547	142133,105303465	161277,3395074	988713,249838537
WBC	14,4308099517	52232,3604384415	68247,3978094808	98929,4020026519	165790,789971294	139100,936339928	181409,905227975
WT	10,8575414757	99664,7499778547	65776,7581742153	63866,8843115188	162296,612201184	115093,639721016	167170,300871015

APÊNDICE J RESULTADOS MÉDIA DE LEITURAS NA PCM

Tabela 35: Tabela com a média de leituras na PCM com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	7245221,43333333	7240491,5	7240083,31034483	7240628,64285714	7247837,64285714	7258017,5	7266113,52941177
WBC	7245265,66666667	7240576,7	7240084,52380952	7240662	7247738,2962963	7256904,1	7268551,53333333
WT	7245210,86666667	7240593,9	7240109,48148148	7240655,75862069	7247667,07142857	7258343,24	7267989,64

Tabela 36: Tabela com o desvio padrão do número de leituras na PCM com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	121,2838458508	180,6886730034	241,6925590568	161,7586712509	318,624034141	1555,103871975	2231,4097314715
WBC	94,8826256154	145,9414502345	301,2465135147	126,80299681	320,9271202691	840,1630104989	1377,3850414394
WT	110,7574963084	138,6741031936	255,4188671982	173,7273428542	337,8329554974	1312,4567383346	2159,9889637064

Tabela 37: Tabela com a média de leituras na PCM com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	29034829,1333333	15196599,6333333	7059919,2	4284360,43333333	3449003,06666667	3399549,93333333	4551791,16666667
WBC	29031081,4666667	15413377,2666667	7117280,03333333	4383120,63333333	3473155,36666667	3340003,2	4536419,43333333
WT	29052677,0666667	15703670,8	6959970,7	4582567,63333333	3346332,5	3379717,7	4545701,63333333

Tabela 38: Tabela com o desvio padrão do número de leituras na PCM com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	8181,2545692233	1249757,00774802	976180,899222786	773561,463044244	501774,413280265	257744,253798744	503737,462638822
WBC	9894,6468903674	1799039,6125459	933606,039230884	866707,999111481	321639,347116106	298254,483762973	271866,422939735
WT	8985,7274567192	2266513,50839271	1502975,95691528	1098413,18672797	492953,053169873	456420,180612887	417547,922546538

Tabela 39: Tabela com a média de leituras na PCM com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	7998993,83333333	8605956,66666667	8001756,73333333	7401277,5	7954505,03333333	8800092,76666667	9518839,23333333
WBC	7999106,63333333	8556729	7883685,9	7331164,4	8011931,83333333	8783784,56666667	9495773,7
WT	8046593,73333333	8624513,36666667	7999918,5	7361368,76666667	8002082,93103448	8860947,59259259	9640242,67857143

Tabela 40: Tabela com o desvio padrão do número de leituras na PCM com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	21505,97716287	79847,6893545728	46147,9510943043	53217,003599608	94224,2868696935	153285,865119027	153755,478436892
WBC	20118,2692055365	79659,2831042915	56801,6826039936	48056,7890654667	95428,5888219954	190327,954585777	253119,031238767
WT	30138,1703983656	106488,821889495	100733,435257666	75317,9683817253	121646,552395786	122273,394476118	168192,882958173

Tabela 41: Tabela com a média de leituras na PCM com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	8585354,2	7384316,53333333	7585206,37037037	7589533,2	7195788,9	7575050	8159893,5
WBC	8585279,6	7541294,66666667	7672965,4	7648710,73333333	7233181,53333333	7684290,66666667	8182639,5
WT	8585711,4	7309458,56666667	7583823,93333333	7589948	7159844,26666667	7591114,96666667	8143362,56666667

Tabela 42: Tabela com o desvio padrão do número de leituras na PCM com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	1090,0832742628	33384,2508922513	93872,1889507005	145367,868342496	80932,3939787251	25912,7435088191	41614,788521713
WBC	951,007871899	44466,7628051314	160978,626195851	198059,943127803	64403,1537564327	27339,9007759132	54513,9475216806
WT	870,6795839636	24359,587205159	22639,5940546848	125772,250350687	61668,6112864187	19199,8595549521	36530,4109775001

Tabela 43: Tabela com a média de leituras na PCM com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	23444,3333333333	47979,1333333333	149515,1	1095125,36666667	1739204,81481482	2441686,1	2773418,77777778
WBC	24835,5666666667	48888,3	161742,533333333	1086026,8	1687707,24137931	2420876,5	2798173,51724138
WT	23557,3666666667	45432,0333333333	155488,433333333	1069740,4	1726765,48	2390084,89655172	2825751,17241379

Tabela 44: Tabela com o desvio padrão do número de leituras na PCM com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	676,6517636092	12170,7154317049	14173,5495258622	95788,2831023111	86198,0288334489	116781,008169388	97392,0000435723
WBC	978,9448465791	11634,902320619	16763,558655882	69681,7687052335	74173,9436766497	93767,7989509192	91453,078160966
WT	975,3180023701	11248,484015814	16310,5414838197	70551,0198589987	115672,078916406	97003,1433346897	75491,9964542453

Tabela 45: Tabela com a média de leituras na PCM com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	24651636,7666667	25594549,3333333	25219636,0666667	25768253,9333333	25997795,5333333	26470741,1	27378690,5172414
WBC	24650166,8333333	25612577,8	25215838,9666667	25768693,5333333	26005322,4	26502203,3333333	27391357,4
WT	25517424,1666667	26340887,3	26175043,2666667	26497822,6666667	26718020,1666667	27066199,7333333	27794729,2666667

Tabela 46: Tabela com o desvio padrão do número de leituras na PCM com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	23780,015335518	32486,7491282144	47471,7411911501	39971,7106955211	47566,8485656173	150433,458066865	22469,4447919529
WBC	20253,0006820332	36361,9065937285	48995,5664746441	32061,9703760608	52773,3455380911	126119,006548777	37660,2029620769
WT	21855,5515950293	41069,8780055365	47933,9718304636	50889,9632102464	41477,8547030143	136608,868411048	32606,7239849373

Tabela 47: Tabela com a média de leituras na PCM com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	12168570,9333333	12166965,1	12160976,5333333	12156203,2333333	12218492,3333333	12353188,1333333	12540224,8666667
WBC	12168598,3666667	12165855,4333333	12159849,5666667	12155241,9	12217283,5333333	12351585,0666667	12539732,2
WT	12170260,4666667	12166664,7333333	12161422,1666667	12170505,3	12220163,8	12352399,8	12540454

Tabela 48: Tabela com o desvio padrão do número de leituras na PCM com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	3524,8786424043	4776,0284203943	3236,8007697399	4396,2803923892	6692,33404124	5862,1413689782	6453,5742608441
WBC	3550,4384040453	4878,405503238	3980,754049894	3747,8628056994	612,0825217219	5466,0699424808	5313,3113021075
WT	3247,0184916408	4405,9993420675	3472,4099857585	9014,4074647965	4958,2613012058	6595,4306774228	5460,6934446599

APÊNDICE K RESULTADOS NÚMERO DE ESCRITAS NA PCM

Tabela 49: Tabela com a média de escritas na PCM com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	15207209,1333333	15206126,3666667	15219207,2758621	15237130,3928571	15261546,25	15297880,6	15311278,8235294
WBC	15207797,5	15206594,8333333	15232873,4285714	15240557,5217391	15255149,2222222	15298234,2	15321043
WT	15207348,3333333	15202642,9666667	15219998,3333333	15229366,137931	15257531,6071429	15291787,52	15309755,76

Tabela 50: Tabela com o desvio padrão do número de escritas na PCM com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	22051,3052795823	22308,2196997265	27072,1339043576	26748,8056186651	21152,1845095909	25767,3126131784	22902,7796845582
WBC	21863,5657639323	21779,2750268553	34977,5884911631	26832,9471272463	25755,7588203154	23488,4852990163	30047,6507638879
WT	19436,1003970364	22674,3329833587	31545,5026356922	19795,3744628171	24433,4704745875	22758,3447060633	31883,8278093038

Tabela 51: Tabela com a média de escritas na PCM com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	18795483,4666667	13013208,2333333	12872183,6333333	13620533,8666667	15113069,3333333	16802327,8333333	16056862,0666667
WBC	18789917,5666667	13058058,8	12892261,4	13595928,8666667	14938318	16272420,3333333	15817424
WT	18800716,9	12708259,8	12684954,3	13530150,4	15424244,4333333	16932640,1666667	17787690,6

Tabela 52: Tabela com o desvio padrão do número de escritas na PCM com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	44766,2829075737	57254,6052944777	38427,5500825082	50261,4488475757	290190,607442684	303262,433453167	1244191,12173849
WBC	51360,5019336329	58908,4269367763	30209,3452731846	73063,0079856602	369975,014533647	588045,433479675	836172,096756898
WT	49139,0336279236	86286,4399365517	46112,1986240233	74049,8239666352	181853,317315733	225995,649831353	509179,92594222

Tabela 53: Tabela com a média de escritas na PCM com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	11676278,9	10447321,1333333	10286104,8	10420717,5666667	10385178,4666667	10100674,6333333	9880684,7
WBC	11673808,6666667	10307197,2666667	10165757,6333333	10240271,5	10091931,9	9812832,76666667	9735641,26666667
WT	11622930,7333333	10216108,1333333	10063798	10295468,3333333	10414586,6896552	10091310,4814815	9805252,35714286

Tabela 54: Tabela com o desvio padrão do número de escritas na PCM com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	52664,0533475126	38815,753538225	40766,2379075018	75741,4657542277	126021,372643755	145604,655997335	59807,8733140978
WBC	51952,5777219381	38021,8996078379	37380,7991422882	82529,6138464123	124677,92225608	225419,268752877	79708,3274306299
WT	76604,8760656258	51493,6665551401	42489,8771579367	84802,8560556507	129528,573386642	70484,7617259713	58679,7084558224

Tabela 55: Tabela com a média de escritas na PCM com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	5641211,9	5669564,13333333	5681445,40740741	5626510,5	5341898,76666667	4775543,25925926	4531774,13333333
WBC	5642973,83333333	5668173,1	5682498,06666667	5642991,8	5417633,83333333	4803489,73333333	4528569,96666667
WT	5640860,6	5670972,86666667	5681906,7	5630006,3	5290343,4	4738786,86666667	4531999,9

Tabela 56: Tabela com o desvio padrão do número de escritas na PCM com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	1031,5424456809	3140,7709874173	8465,2435116924	33848,9407930343	41569,9459761607	64661,9870555233	10547,9810087485
WBC	957,5707440628	3593,6242329418	10155,8647976677	15024,4402073697	42733,7027399665	37264,0884488865	12530,9936130334
WT	1062,5534501769	3009,167999726	2770,2350223962	26687,8331598683	75647,3140747974	54189,4714807306	11348,484466188

Tabela 57: Tabela com a média de escritas na PCM com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	569411,333333333	631726,2	960869,533333333	4024124,9	5624150,66666667	7368522,26666667	8197321,33333333
WBC	569777,033333333	635263,3	1006169,1	4026634,2	5533561,79310345	7327689,5	8248882,06896552
WT	570108,6	632486,633333333	981128,033333333	3979687,76666667	5599760,6	7216208,55172414	8344076,44827586

Tabela 58: Tabela com o desvio padrão do número de escritas na PCM com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	764,7330538679	16341,0834917015	64598,0963719378	267125,826962701	217436,228894996	320348,403171312	424972,824408084
WBC	1084,0668793255	18066,3716986885	64426,8137664211	160749,32780326	171268,977276634	258747,559978795	294038,874497906
WT	2219,3361946374	11576,4645710324	69009,372863893	189564,743919924	287417,277987946	261895,689680696	243662,361652142

Tabela 59: Tabela com a média de escritas na PCM com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	23559491,3333333	24284102,2	23758009,2333333	23704065,7333333	23302539,7666667	22686475,0666667	21820121,0333333
WBC	23556878,4	24283122,5333333	23767940,6666667	23705274,4	23294021,4666667	22691147,0333333	21820582,0333333
WT	23560923,1666667	23911228,6	23939148,8333333	23690537,8666667	23274673,5666667	22632449,3333333	21711077

Tabela 60: Tabela com o desvio padrão do número de escritas na PCM com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	33092,2336209418	22729,4052464021	24718,4049070844	21614,7595836149	23502,595564193	26044,6601382997	21240,7648045359
WBC	32747,2012961571	25980,9759654933	25268,1076053199	18656,5129990242	24543,0478065945	31624,2103485805	17402,9800430317
WT	27186,7030506527	22831,8668193201	71113,3124910802	21636,357896284	23626,6983212714	27289,7395286256	18724,9025576397

Tabela 61: Tabela com a média de escritas na PCM com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	19201553,4333333	19210688,9	19224866,9	19191508,0666667	19039951,4	18940572,0666667	18907229,1333333
WBC	19202109,0333333	19209606,2333333	19225469,3333333	19192491	19039856,0666667	18941471,3333333	18907630,8666667
WT	19201596,9333333	19208994,7666667	19224613,6666667	19193717,0333333	19040054,3333333	18941023,9666667	18907035,8666667

Tabela 62: Tabela com o desvio padrão do número de escritas na PCM com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	1875,7962480952	8878,2727442161	4250,1660523342	4248,2056247179	4216,0254170535	3804,4469255327	2782,2919568826
WBC	2221,8086680611	3340,7360915423	3838,5002758651	5290,4849526752	4008,1455505038	3076,0424791172	2211,4501363918
WT	2056,8595736577	2444,9286948679	5072,1992600186	5945,3578963837	4104,6484526132	4350,8885773648	2094,9114166662

APÊNDICE L RESULTADOS NÚMERO DE BITS ALTERADOS NA PCM

Tabela 63: Tabela com a média de bits alterados na PCM com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	11919265,6	11991779,8333333	12047296,5172414	12070318,1785714	12067500,6071429	12116424,05	12088850,4705882
WBC	11934946,7	12020271,3	12104805,4285714	12029510,5652174	12041966,7037037	12045462,25	12086288,8
WT	11971412,2333333	12009481,8333333	12055596,5555556	12066803,3448276	12082285,7857143	11989912,88	12072183,04

Tabela 64: Tabela com o desvio padrão do número de bits alterados na PCM com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	136466,57001817	127253,926120608	112630,862593766	102282,786903852	125444,752574173	92673,559013104	120156,707674352
WBC	106801,611007529	114450,572970696	154573,66958495	93419,8541749831	134050,829320252	136399,74059702	117162,179673672
WT	143448,497758431	143125,77467222	133413,610124631	96803,4975169787	127910,202743481	110555,043483371	132793,604959124

Tabela 65: Tabela com a média de bits alterados na PCM com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	17818721,2666667	23434857,8666667	28441523,4333333	32439218,1666667	34105175,8333333	29751532,1333333	19510696,3333333
WBC	18785154,1333333	23149233,1666667	28903549,6666667	32367021,4	32208391,0666667	27635483,1	18538909,4333333
WT	18149069,8333333	24064763,7666667	29015503,8	32566995,3333333	36141873,9666667	30017638,0666667	23426072,7666667

Tabela 66: Tabela com o desvio padrão do número de bits alterados na PCM com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	1796235,4547808	1533742,24967522	1654522,668943	1902524,08693339	1946517,13213965	1305171,25454102	2604261,10842849
WBC	1487856,80357833	1421382,48945084	1338447,65584065	1296481,26637767	1502590,46831138	1901239,49444861	1521213,34216513
WT	1457070,31192702	1840428,77756171	1456599,75454709	1579880,84816797	1646755,62903828	1309417,07496894	1117328,91454174

Tabela 67: Tabela com a média de bits alterados na PCM com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	23512011,2333333	37034488,9666667	36432839,5666667	35069997,6	33883159,5333333	30835821,4666667	27331277,6
WBC	23448703,5666667	35776972,8	33838583,7	33258020,0333333	32327280,1	30444820,9333333	27365755,6666667
WT	24206084,3	37725983,9666667	36655197,8	35573892,3666667	34651521,4827586	3186491,8518519	27431895,5

Tabela 68: Tabela com o desvio padrão do número de bits alterados na PCM com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	849636,886266439	1915520,09847054	1197575,33440303	1123027,50959061	996218,078494301	748509,671508093	747262,31120102
WBC	874080,361283106	1917459,06231127	1313244,94520498	892264,967211456	922878,116758891	868827,067386272	550904,051282062
WT	666694,247269893	2090586,51045153	1254244,80564605	1146492,3356293	1136602,62753576	600959,711482949	581243,22529707

Tabela 69: Tabela com a média de bits alterados na PCM com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	14436663,9	25149750,2333333	31357931	30931950,2333333	28631999,8666667	29720100,2	29847427,2
WBC	14458003,3666667	24320088,3666667	29677279,8666667	29656276,4666667	24987829,2666667	28404819,3666667	28636959,1
WT	14434449,8333333	25859172,1666667	32083285,9666667	32594806,7333333	30108831,0666667	31206982,4	30574322,5333333

Tabela 70: Tabela com o desvio padrão do número de bits alterados na PCM com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	10501,8949959946	276857,913264583	665281,637780398	1442746,9099629	4146508,60820642	1368928,90206501	1495967,66604289
WBC	8923,5990129363	284669,692590587	629027,117638502	996142,798315678	3904376,17898625	617690,466685349	1555361,29048339
WT	14899,4877789377	243921,866648556	313852,766450535	1198133,06825298	3788451,62839882	969694,501067559	823143,596411688

Tabela 71: Tabela com a média de bits alterados na PCM com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	1635868,16666667	2169962,2	7040014,2	86337252,6	125495620	164885982,166667	171031473,666667
WBC	1541519,23333333	2293011,8	7812085,1	86161467,4333333	120638126,733333	163941284,346154	171725512,586207
WT	1625347,9	2447537,96666667	7175717,76666667	85061734,8	124401976,28	161313025,241379	174426241,517241

Tabela 72: Tabela com o desvio padrão do número de bits alterados na PCM com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	208700,47112203	538098,826913562	1359170,34555117	5774425,3610561	6053921,96675027	8164513,89404806	9379370,23120525
WBC	217724,871083808	683817,775872793	1456114,53219574	4049775,33342046	12489080,0101036	6493085,48176885	7029982,60891397
WT	185852,436090154	671833,942504241	1521788,15469925	4391439,71065959	7472881,00934731	6973287,8763985	5843338,08713036

Tabela 73: Tabela com a média de bits alterados na PCM com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	81282138,5	95567337,7666667	93978013,3333333	93795884,3	92282071,1333333	92077500,5333333	91931383,7333333
WBC	81560714,8333333	95851302,6666667	94211272,5333333	93836565,6666667	91980358,5	92099752,2666667	91808747,3
WT	81229251,3666667	95724535,0666667	94102526,4	93711766,4666667	92000871	92161084,6666667	91887318,4333333

Tabela 74: Tabela com o desvio padrão do número de bits alterados na PCM com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	431855,808782064	568952,96620757	724705,253285458	597443,067441412	675987,142006758	498885,827648712	326184,407821704
WBC	560961,991380268	703638,468774802	645206,954413091	669445,250220032	538990,233599687	466579,103706448	384121,515617322
WT	431080,451143922	715621,549829909	737856,767010845	509770,757612014	658716,422238343	525080,022035087	395295,9965555

Tabela 75: Tabela com a média de bits alterados na PCM com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	47381887,3666667	47750189,1666667	48015719,0666667	47758379,1666667	46955100,0333333	46472612,7	46572259,9666667
WBC	47333056	47788584,3666667	47996492,7333333	47810150,7666667	47016387,6666667	46565952,2333333	46591866,1333333
WT	47497965,7666667	47841022,8666667	48114966,9666667	47955207,3	46995197,6333333	46581612,4666667	46594888,8333333

Tabela 76: Tabela com o desvio padrão do número de bits alterados na PCM com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	284268,206895201	247138,113198224	251361,895963587	267933,649301475	217148,585261175	228372,2723098	226419,523838852
WBC	276776,131119694	236146,742500401	235205,14629397	250175,63793445	239404,571195855	228935,017188764	243985,400035886
WT	256577,340887047	235850,432972157	250318,195725097	278469,43398416	207885,436411747	219154,603187155	232259,408617574

APÊNDICE M RESULTADOS DO CONSUMO DE ENERGIA

Tabela 77: Tabela com a média do consumo de energia com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	236350115,276667	237575365,3	238599174,455172	239023076,896429	238962563,821429	239847780,94	239310542,888235
WBC	236628416,833333	238111386,56	239648002,804762	238266709,486957	238489607,648148	238558196,48	239251800,386667
WT	237305960,683333	237915424,56	238745893,82963	238946536,575862	239228933,575	237513848,944	239001543,932

Tabela 78: Tabela com o desvio padrão do consumo de energia com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	2521993,97459283	2340113,74557954	2077357,1741079	1886768,09925072	2320475,86878253	1707965,56288213	2204138,98275723
WBC	1965845,11478892	2106832,45592362	2852065,31853083	1732502,81511934	2472309,86611401	2514088,81167962	2168766,46506911
WT	2650347,5363133	2638646,98613849	2455935,27982977	1781554,05970521	2358750,99765774	2043419,04845579	2442366,13040016

Tabela 79: Tabela com a média do consumo de energia com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	391060539,296667	447702739,476667	504719190,48	558912048,206667	575006937,923333	495161908,846667	315751876,893333
WBC	409601767,633333	444174819,143333	513783044,336667	559005007,566667	546228946,933333	458926439,21	302307372,716667
WT	397250516,233333	457830728,06	511591591,95	560591436,566667	607179080,52	498271018,02	382134502,466666

Tabela 80: Tabela com o desvio padrão do consumo de energia com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	34214614,6601087	29041237,3958371	30968918,2926813	34685834,7873185	33118812,9913623	22358503,1010483	44299099,9770726
WBC	28408034,2606822	27515503,1480607	24996399,0912667	23559623,2733732	25984947,5342648	32308421,1376017	27275668,7881982
WT	27677256,220987	34901342,1974412	25636247,3726035	27958602,576249	28896938,3911165	22183986,835815	18743174,5178433

Tabela 81: Tabela com a média do consumo de energia com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	431441107,286667	653522536,203333	633256151,536667	600915506,89	576588287,996667	522843878,443333	462692745,176667
WBC	430208801,306667	634882941,15	590422872,41	570657811,36	551935538,616667	510423344,153333	458426486,75
WT	443063237,516667	665761588,963333	636000626,64	610281339,583333	591162929,748276	539899500,840741	465168114,103571

Tabela 82: Tabela com o desvio padrão do consumo de energia com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	15423901,4208797	32694960,8702287	20993047,0532809	19318420,6471392	16689921,9852708	12822186,7092412	11440473,2603737
WBC	15681098,959553	32836764,4724766	22848458,0158993	14973535,5854086	15557046,9166299	14341114,7528688	9019826,8854203
WT	11873920,4195912	36744414,9299012	21758491,8460784	18644557,5992209	17943734,3286106	9731247,2588151	9057047,50583125

Tabela 83: Tabela com a média do consumo de energia com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	278666474,6	445060248,846667	543256852,662963	535583693,19	497067419,13	505196739,31	498665528,32
WBC	278996888,21	431846763,723333	516309664,43	515851249,696667	441616885,016667	486514172,083333	482184940,41
WT	278643056,67	456771474,673333	554528988,606667	561635803,98	519132011,513333	522846838,063333	507816301,003333

Tabela 84: Tabela com o desvio padrão do consumo de energia com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	170299,775013366	4532842,90164059	10306972,6285153	22891050,9814498	62589782,277017	18924521,9508942	19241955,5587398
WBC	151397,598598705	4436147,44227543	9957444,74063045	15805991,0075966	60406885,6287997	9009163,34790091	20126356,7985235
WT	241258,365778019	3767222,62144071	5055896,09966636	18952832,6025054	56574992,3737594	11625170,5049634	10784605,042168

Tabela 85: Tabela com a média do consumo de energia com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	22273233,4766667	34469822,2766667	118405388,72	1436238215,37333	2099089580,27407	2786090574,07	2980602479,68889
WBC	20981380,6433333	36904477,54	131267920,946667	1433112752,6	2054138594,81034	2769301322,3	2989900052,51724
WT	22113774,1133333	38711394,8366667	120336291,216667	1415954198,32	2081462705,576	2726820768,04138	3034506159,87241

Tabela 86: Tabela com o desvio padrão do consumo de energia com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	2875169,49349921	9155592,22344571	22092314,7093301	94800501,4282777	99058538,0499019	134429067,715966	155862524,275804
WBC	2968707,33579742	12137932,4160636	23692365,323559	66223038,7794101	87920555,9635219	106138042,791021	116103575,004235
WT	2557528,19882057	10990114,954685	24591641,5347593	72064474,2099681	122622763,106817	113974206,514998	95528441,9814435

Tabela 87: Tabela com a média do consumo de energia com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	1551948169,04333	1716551741,32667	1677120174,00333	1665164250,11667	1641123534,94667	1637689651,25	1633630000,58
WBC	1556850987,29667	1721447291,35	1681230977,43333	1665784489,82667	1635588509,4	1637582755,08667	1632023995,16
WT	1552866701,35333	1720787238,61	1681344299,78333	1665048815,70333	1637598619,69333	1640030910,33667	1634293104,82333

Tabela 88: Tabela com o desvio padrão do consumo de energia com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	7484026,09111447	9595844,20206357	12823514,4616281	10103352,2372586	11364280,7400395	8572019,89062419	5393830,70454792
WBC	10412793,8609133	12048708,2320236	11145654,2425914	11928635,2369022	9150032,76817907	7720868,42383842	6115016,87952133
WT	7137702,30121698	12310545,5199453	13015751,0980637	8481999,62324821	10791072,0692456	8928944,80973824	6813086,12909818

Tabela 89: Tabela com a média do consumo de energia com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	872692057,466667	879414453,47	883829202,956667	878856708,136667	864229184,296667	855657967,906667	857612984,163333
WBC	871780407,763333	880060248,706667	883450935,303334	879812822,74	865326967,896667	857299660,073333	857930461,2
WT	874901078,853333	881063311,026666	885789841,903333	882611102,61	865121404,61	857636863,89	858041163,5

Tabela 90: Tabela com o desvio padrão do consumo de energia com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	5255128,42944153	4537439,90025404	4717217,50387919	4961920,84920384	3946178,7676221	4264973,34304474	4120998,53673862
WBC	5146850,41605966	4358515,88494463	4303713,42443852	4563023,27572937	4428875,87644393	4221106,97343337	4516744,38209683
WT	4735626,19512269	4299440,6438478	4619913,03620368	5135417,42815663	3817456,07626433	4060881,64737012	4267307,22577981

APÊNDICE N RESULTADOS NÚMERO DE TRANSAÇÕES

Tabela 91: Tabela com a média de transações com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	479	487	524,4137931034	519,8	561	572,7272727273	631
WBC	479	486,1	523,7692307692	524,9615384615	565,16	564,4615384615	627
WT	479	499	524,6	523,9	568,9285714286	573,68	638,1363636364

Tabela 92: Tabela com o desvio padrão do número de transações com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	0	15,7370636528	4,4442357641	10,733126292	20,4886728288	15,6879748251	25,6216819947
WBC	0	14,5088557975	6,1142959844	11,0832513974	21,6153494844	23,0533828654	21,6333076528
WT	0	20,0550965231	4,4844559121	11,9779970693	18,8835193115	17,351080658	27,5496010766

Tabela 93: Tabela com a média de transações com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	689678	689676	689678	689679	689682,966666667	689686,766666667	689714,333333333
WBC	689678	689676	689678	689679	689682,7	689686,9	689715,533333333
WT	689678	689676	689678	689679	689682,633333333	689686,3	689716,033333333

Tabela 94: Tabela com o desvio padrão do número de transações com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	0	0	0	0	1,7116907015	1,104327954	2,9282610272
WBC	0	0	0	0	1,417866294	1,2958820721	3,4912040541
WT	0	0	0	0	1,5196036991	1,5346570997	3,3577838888

Tabela 95: Tabela com a média de transações com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	796012	796013	796015	796019	796027	796043	796075
WBC	796012	796013	796015	796019	796027	796043	796075
WT	796012	796013	796015	796019	796027	796043	796075

Tabela 96: Tabela com o desvio padrão do número de transações com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	0	0	0	0	0	0	0
WBC	0	0	0	0	0	0	0
WT	0	0	0	0	0	0	0

Tabela 97: Tabela com a média de transações com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	262146	262149	262155	262167	262191	262239	262335
WBC	262146	262149	262155	262167	262191	262239	262335
WT	262146	262149	262155	262167	262191	262239	262335

Tabela 98: Tabela com o desvio padrão do número de transações com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	0	0	0	0	0	0	0
WBC	0	0	0	0	0	0	0
WT	0	0	0	0	0	0	0

Tabela 99: Tabela com a média de transações com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	258	260	264	272	288	320	384
WBC	258	260	264	272	288	320	384
WT	258	260	264	272	288	320	384

Tabela 100: Tabela com o desvio padrão do número de transações com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	0	0	0	0	0	0	0
WBC	0	0	0	0	0	0	0
WT	0	0	0	0	0	0	0

Tabela 101: Tabela com a média de transações com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	700777	700782,533333333	700786,8	700796,466666667	700817,666666667	700854,933333333	700937,066666667
WBC	700777	700782,466666667	700786,933333333	700796,6	700818	700852,533333333	700937,4
WT	700777	700782,666666667	700787,066666667	700796,733333333	700817,6	700854,466666667	700936,533333333

Tabela 102: Tabela com o desvio padrão do número de transações com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	0	0,8603661343	0,6102571533	1,9605300714	1,6045911142	4,5632212861	4,4094986292
WBC	0	0,8995528902	0,3651483717	1,9930915165	1,3645764784	3,4713937709	4,9102285922
WT	0	0,7580980436	0,6396838299	2,016027732	1,8307714598	4,9808829942	4.6589427048

Tabela 103: Tabela com a média de transações com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	16384	16384	16384	16384	16384	16384	16384
WBC	16384	16384	16384	16384	16384	16384	16384
WT	16384	16384	16384	16384	16384	16384	16384

Tabela 104: Tabela com o desvio padrão do número de transações com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	0	0	0	0	0	0	0
WBC	0	0	0	0	0	0	0
WT	0	0	0	0	0	0	0

APÊNDICE O RESULTADOS NÚMERO DE *ABORTS*

Tabela 105: Tabela com a média de *aborts* com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	0	8,5	408,2068965517	2186,1	4795,9310344828	10218,5	48004,9444444444
WBC	0	6,5333333333	27,7307692308	93,0384615385	187,12	414,1538461538	573,1111111111
WT	0	10,2666666667	477,8333333333	2296,5	4886,75	9724,88	47752,2727272727

Tabela 106: Tabela com o desvio padrão do número de *aborts* com o *benchmark* Bayes

Número de Threads	1	2	4	8	16	32	64
WBE	0	4,1667816076	269,8971205201	448,5425709239	1559,5582234135	4556,1694724741	18486,1011015819
WBC	0	1,5477087255	4,0055730407	10,3864556774	32,5606101499	117,734172544	281,1789663384
WT	0	9,5878275319	276,3275272188	409,5604666738	1840,9719981977	5549,5843562078	14407,3771023631

Tabela 107: Tabela com a média de *aborts* com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	0	4999,9333333333	11549,8	18184,3666666667	46984,6333333333	153841,666666667	1479621,53333333
WBC	0	1054,8666666667	1991,6	5941,8666666667	18759,1	53337,0333333333	148826,666666667
WT	0	5089,4333333333	8684,1	16732,6	51020,5	169394,5	1866721,3

Tabela 108: Tabela com o desvio padrão do número de *aborts* com o *benchmark* Genome

Número de Threads	1	2	4	8	16	32	64
WBE	0	3514,8211210023	13239,6091368496	9785,6097672088	10186,2606907991	37687,4029428231	196955,830651712
WBC	0	48,1124640338	120,44933117	415,5070631653	823,1105573707	1969,9648777428	4398,8833091838
WT	0	1442,6701461439	13784,1309316685	7314,9021423204	4837,8758786519	27127,0061756126	344291,414235408

Tabela 109: Tabela com a média de *aborts* com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	0	359758,3	913050,933333333	2314907,4	4559643,53333333	13926280,2666667	29549372,5666667
WBC	0	29731,4	93701,3666666667	217744,9	987640,966666667	2837890,4	2953330,5
WT	0	332230,6	824782,833333333	1924189,68965517	3919809,46666667	12293653,4642857	26994859,64

Tabela 110: Tabela com o desvio padrão do número de *aborts* com o *benchmark* Intruder

Número de Threads	1	2	4	8	16	32	64
WBE	0	3535,1067500831	44739,861825148	126353,135106663	433559,615837095	556404,102766669	670860,048628659
WBC	0	249,138045102	3370,0041387812	7894,6643061562	100242,252436538	94468,2761188589	241488,423372526
WT	0	18390,6019019836	29262,8229942354	87827,1690871597	249733,191670594	678343,360854712	672245,273346026

Tabela 111: Tabela com a média de *aborts* com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	0	264841,766666667	989551,433333333	2364953,96666667	5087114,13793104	13282911,1666667	19888312,8666667
WBC	0	12513,9333333333	36228,9666666667	83546,1	224574,733333333	338085,6	506031,2
WT	0	256533,133333333	739693,166666667	2346523,63333333	4460001,5	11931451,9666667	18484003,5333333

Tabela 112: Tabela com o desvio padrão do número de *aborts* com o *benchmark* Kmeans

Número de Threads	1	2	4	8	16	32	64
WBE	0	16371,3704093948	475864,54648904	481957,268127722	637558,531714076	1293636,68304622	810429,671405906
WBC	0	100,8245089708	1203,4846956417	6248,8824233922	15940,0953549983	41831,0817097394	41737,9051581785
WT	0	42416,1213363856	335339,343241222	780644,437240752	569598,368843603	1103465,52769363	1111416,20351649

Tabela 113: Tabela com a média de *aborts* com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	0	11,5	49,6333333333	343,0666666667	751,3793103448	4052,5384615385	27596,375
WBC	0	6,7	23,9666666667	53,6206896552	108,7857142857	264,4615384615	850,6551724138
WT	0	17,7	49,1666666667	110,3	659,4827586207	3936,7037037037	25659,7037037037

Tabela 114: Tabela com o desvio padrão do número de *aborts* com o *benchmark* Labyrinth

Número de Threads	1	2	4	8	16	32	64
WBE	0	16,6168796606	51,3181310524	393,7264877125	704,4242032526	3252,0994970114	13640,636216223
WBC	0	1,3933338833	2,7478309105	6,0202859039	11,895710845	63,1241511748	402,1075971732
WT	0	31,6490997856	56,6727346312	105,9463592318	577,7146861736	3280,6635915832	16498,8567338533

Tabela 115: Tabela com a média de *aborts* com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	0	33,4	85,2333333333	282,6333333333	660,3	4284,2666666667	24243,4333333333
WBC	0	12,1666666667	39,2	97,9	217,9333333333	464,7333333333	905,9333333333
WT	0	62,9666666667	81,9333333333	262,1666666667	1256,2666666667	3741,0333333333	11872,6666666667

Tabela 116: Tabela com o desvio padrão do número de *aborts* com o *benchmark* SSCA2

Número de Threads	1	2	4	8	16	32	64
WBE	0	23,1778729202	30,2069680627	142,0974612924	262,7299846374	4398,1993332173	20538,9661974725
WBC	0	2,6663074471	7,2796362168	11,7278036866	16,6379983323	50,6202449165	99,1376379806
WT	0	40,4760323575	24,6323775342	182,5135013113	1474,2376155581	3922.0237341536	15986,7593152052

Tabela 117: Tabela com a média de *aborts* com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	0	3,1666666667	13,8666666667	56,7666666667	104,2413793103	174,3928571429	364,3
WBC	0	2,4	8,4	25,2	42,6	74,5666666667	140,2666666667
WT	0	4,5666666667	14,6666666667	59,7666666667	114	164,7	389,5

Tabela 118: Tabela com o desvio padrão do número de *aborts* com o *benchmark* Vacation

Número de Threads	1	2	4	8	16	32	64
WBE	0	2,0014362659	5,3930564128	19,7792704676	23,8352307856	33,780708154	44,1534549747
WBC	0	1,5222487902	2,5677039254	5,5236170059	6,2400265251	9,6977897707	9,5337271842
WT	0	7,2904440683	6,1044547517	17,0853503155	22,3560529488	30,9907105103	88,8628350227

APÊNDICE P RESULTADOS TAXAS DE HIT E MISS NOS NÍVEIS DE CACHE

Tabela 119: Tabela com a porcentagem de Hit e Miss no nível de cache L1 com o *benchmark* Bayes

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	96,51	3,49	96,51	3,49	96,52	3,48	96,51	3,49	96,53	3,47	96,52	3,48	96,55	3,45
WBC	96,51	3,49	96,50	3,50	96,52	3,48	96,51	3,49	96,53	3,47	96,52	3,48	96,54	3,46
WT	96,51	3,49	96,51	3,49	96,52	3,48	96,51	3,49	96,53	3,47	96,52	3,48	96,55	3,45

Tabela 120: Tabela com a porcentagem de Hit e Miss no nível de cache L2 com o *benchmark* Bayes

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	5,30	94,70	5,36	94,64	5,37	94,63	5,41	94,59	5,43	94,57	5,46	94,54	5,44	94,56
WBC	5,29	94,71	5,35	94,65	5,37	94,63	5,40	94,60	5,44	94,56	5,44	94,56	5,42	94,58
WT	5,30	94,70	5,31	94,69	5,38	94,62	5,41	94,59	5,44	94,56	5,45	94,55	5,42	94,58

Tabela 121: Tabela com a porcentagem de Hit e Miss no nível de cache L3 com o *benchmark* Bayes

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	0,04	99,96	0,04	99,96	0,12	99,88	0,25	99,75	0,33	99,67	0,40	99,60	0,44	99,56
WBC	0,04	99,96	0,04	99,96	0,12	99,88	0,25	99,75	0,33	99,67	0,40	99,60	0,45	99,55
WT	0,04	99,96	0,04	99,96	0,12	99,88	0,25	99,75	0,33	99,67	0,40	99,60	0,45	99,55

Tabela 122: Tabela com a porcentagem de Hit e Miss no nível de cache L1 com o *benchmark* Genome

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	92,09	7,91	91,54	8,46	91,96	8,04	92,16	7,84	92,96	7,04	94,61	5,39	96,65	3,35
WBC	92,43	7,57	91.88	8,12	92,30	7,70	92,51	7,49	93,30	6,70	94,90	5,10	96,87	3,13
WT	90,78	9,22	90,32	9,68	90,82	9,18	90,95	9,05	91,69	8,31	93,59	6,41	95,89	4,11

Tabela 123: Tabela com a porcentagem de Hit e Miss no nível de cache L2 com o *benchmark* Genome

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	12,28	87,72	33,76	66,24	48,51	51,49	62,12	37,88	76,34	23,66	86,31	13,69	91,52	8,48
WBC	12,29	87,71	33,29	66,71	48,38	51,62	61,61	38,39	76.57	23.43	86,26	13,74	91,54	8,46
WT	12,31	87,69	32,87	67,13	48,61	51,39	61,50	38,50	76,12	23,88	86,61	13,39	91,78	8.22

Tabela 124: Tabela com a porcentagem de Hit e Miss no nível de cache L3 com o *benchmark* Genome

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	41,26	58,74	56,39	43,61	68,14	31,86	74,20	25,80	72,86	27,14	65,34	34,66	48,26	51,74
WBC	41,27	58,73	56,39	43,61	68,02	31,98	73,62	26,38	72,61	27,39	64,76	35,24	47,72	52,28
WT	41,25	58,75	55,29	44,71	67,81	32,19	72,89	27,11	73,40	26,60	64,98	35,02	50,63	49,37

Tabela 125: Tabela com a porcentagem de Hit e Miss no nível de cache L1 com o *benchmark* Intruder

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	97,29	2,71	97,54	2,46	97,89	2,11	98,38	1,62	98,93	1,07	99,37	0,63	99,66	0,34
WBC	97,50	2,50	97,83	2,17	98,34	1,66	98,45	1,55	98,88	1,12	99,12	0,88	99,40	0,60
WT	96,91	3,09	97,20	2,80	97,62	2,38	98,16	1,84	98,79	1,21	99,24	0,76	99,61	0,39

Tabela 126: Tabela com a porcentagem de Hit e Miss no nível de cache L2 com o *benchmark* Intruder

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	26,40	73,60	24,24	75,76	28,10	71,90	30,62	69,38	29,28	70,72	26,71	73,29	24,07	75,93
WBC	26,44	73,56	24,87	75,13	28,37	71,63	30,80	69,20	28,98	71,02	25,85	74,15	23,35	76,65
WT	26,13	73,87	24,07	75,93	27,80	72,20	30,46	69,54	29,14	70,86	26,46	73,54	24,53	75,47

Tabela 127: Tabela com a porcentagem de Hit e Miss no nível de cache L3 com o *benchmark* Intruder

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	32,95	67,05	29,96	70,04	30,20	69,80	31,43	68,57	28,75	71,25	25,89	74,11	17,96	82,04
WBC	32,95	67,05	29,55	70,45	30,22	69,78	30,94	69,06	28,24	71,76	25,47	74,53	21,00	79,00
WT	32,60	67,40	30,08	69,92	30,14	69,86	31,46	68,54	28,83	71,17	26,44	73,56	18,85	81,15

Tabela 128: Tabela com a porcentagem de Hit e Miss no nível de cache L1 com o *benchmark* Kmeans

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	98,22	1,78	98,27	1,73	98,36	1,64	98,52	1,48	98,70	1,30	99,10	0,90	99,29	0,71
WBC	98,36	1,64	98,40	1,60	98,47	1,53	98,58	1,42	98,81	1,19	99,03	0,97	99,17	0,83
WT	98,13	1,87	98,17	1,83	98,24	1,76	98,38	1,62	98,59	1,41	99,01	0,99	99,25	0,75

Tabela 129: Tabela com a porcentagem de Hit e Miss no nível de cache L2 com o *benchmark* Kmeans

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	39,37	60,63	37,89	62,11	38,37	61,63	39,67	60,33	40,64	59,36	40,33	59,67	40,08	59,92
WBC	39,39	60,61	37,81	62,19	38,44	61,56	39,60	60,40	40,54	59,46	40.02	59,98	40.07	59,93
WT	39,37	60,63	37,76	62,24	38,20	61,80	39,52	60,48	40,56	59,44	40,21	59,79	40.05	59,95

Tabela 130: Tabela com a porcentagem de Hit e Miss no nível de cache L3 com o *benchmark* Kmeans

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	12,52	87,48	24,26	75,74	22,17	77,83	19,81	80,19	21,69	78,31	18,06	81,94	12.82	87,18
WBC	12,52	87,48	23,09	76,91	20,91	79,09	19,52	80,48	21,79	78,21	17,95	82,05	12,94	87,06
WT	12,52	87,48	25,01	74,99	21,82	78,18	19,92	80,08	21,98	78,02	17,78	82,22	12,82	87,18

Tabela 131: Tabela com a porcentagem de Hit e Miss no nível de cache L1 com o *benchmark* Labyrinth

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	96,70	3,30	96,73	3,27	96,78	3,22	96,85	3,15	96,97	3,03	97,06	2,94	97,05	2,95
WBC	96,70	3,30	96,74	3,26	96,78	3,22	96,87	3,13	96,96	3,04	97,07	2,93	97,04	2,96
WT	96,70	3,30	96,73	3,27	96,78	3,22	96.85	3,15	96,97	3,03	97,06	2,94	97,04	2,96

Tabela 132: Tabela com a porcentagem de Hit e Miss no nível de cache L2 com o *benchmark* Labyrinth

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	83,00	17,00	69,96	30,04	68,54	31,46	66,74	33,26	64,91	35,09	64,11	35,89	64,43	35,57
WBC	83,16	16,84	70,04	29,96	68,41	31,59	66,82	33,18	64,88	35,12	64,00	36,00	63,90	36,10
WT	83,00	17,00	69,71	30,29	68,47	31,53	66,69	33,31	64,84	35,16	63,80	36,20	64,16	35,84

Tabela 133: Tabela com a porcentagem de Hit e Miss no nível de cache L3 com o *benchmark* Labyrinth

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	92,99	7,01	94,41	5,59	91,59	8,41	76,26	23,74	71,50	28,50	66,53	33,47	60,72	39,28
WBC	92,92	7,08	94,40	5,60	91,35	8,65	76,80	23,20	71,63	28,37	66,61	33,39	60,61	39,39
WT	92,99	7,01	94,47	5,53	91,49	8,51	76,57	23,43	71,38	28,62	66,30	33,70	60,54	39,46

Tabela 134: Tabela com a porcentagem de Hit e Miss no nível de cache L1 com o *benchmark* SSCA2

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	87,87	12,13	87,73	12,27	87,43	12,57	86,75	13,25	85,40	14,60	82,82	17,18	78,15	21,85
WBC	87,91	12,09	87,79	12,21	87,50	12,50	86,83	13,17	85,49	14,51	82,94	17,06	78,31	21,69
WT	87,56	12,44	87,43	12,57	87,12	12,88	86,43	13,57	85,05	14,95	82,41	17,59	77,64	22,36

Tabela 135: Tabela com a porcentagem de Hit e Miss no nível de cache L2 com o *benchmark* SSCA2

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	5,71	94,29	4,45	95,55	6,48	93,52	9,15	90,85	13,25	86,75	21,99	78,01	31,28	68,72
WBC	5,72	94,28	4,45	95,55	6,49	93,51	9,20	90,80	13,26	86,74	21,93	78,07	31,25	68,75
WT	5,49	94,51	4,28	95,72	6,24	93,76	8,97	91,03	13,00	87,00	21,72	78,28	31,01	68,99

Tabela 136: Tabela com a porcentagem de Hit e Miss no nível de cache L3 com o *benchmark* SSCA2

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	17,94	82,06	13,17	86,83	16,81	83,19	19,98	80,02	27,62	72,38	36,20	63,80	48,11	51,89
WBC	17,93	82,07	13,15	86,85	16,81	83,19	19,94	80,06	27,60	72,40	36,18	63,82	48,19	51,81
WT	17,10	82,90	12,64	87,36	15,70	84,30	19,35	80,65	26,83	73,17	35,44	64,56	47,54	52,46

Tabela 137: Tabela com a porcentagem de Hit e Miss no nível de cache L1 com o *benchmark* Vacation

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	94,58	5,42	94,64	5,36	94,67	5,33	94,68	5,32	94,69	5,31	94,68	5,32	94,68	5,32
WBC	94,69	5,31	94,75	5,25	94,78	5,22	94,79	5,21	94,80	5,20	94,79	5,21	94,78	5,22
WT	94,34	5,66	94,40	5,60	94,43	5,57	94,44	5,56	94,45	5,55	94,44	5,56	94,44	5,56

Tabela 138: Tabela com a porcentagem de Hit e Miss no nível de cache L2 com o *benchmark* Vacation

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	34,88	65,12	34,97	65,03	34,96	65,04	34,91	65,09	34,70	65,30	34,23	65,77	33,50	66,50
WBC	34,88	65,12	34,96	65,04	34,97	65,03	34,90	65,10	34,71	65,29	34,22	65,78	33,51	66,49
WT	34,88	65,12	34,96	65,04	34,96	65,04	34,86	65,14	34,68	65,32	34,22	65,78	33,50	66,50

Tabela 139: Tabela com a porcentagem de Hit e Miss no nível de cache L3 com o *benchmark* Vacation

Threads	1		2		4		8		16		32		64	
	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss	Hit	Miss
WBE	30,47	69,53	30,49	69,51	30,59	69,41	30,68	69,32	30,47	69,53	30,07	69,93	29,57	70,43
WBC	30,48	69,52	30,50	69,50	30,59	69,41	30,69	69,31	30,47	69,53	30,07	69,93	29,57	70,43
WT	30,46	69,54	30,51	69,49	30,59	69,41	30,65	69,35	30,46	69,54	30,07	69,93	29,57	70,43