Avaliação de Desempenho de Compressão de Páginas Utilizando o ZRAM

Guilherme Eduardo Gonçalves da Silva
Universidade Federal do Paraná
Curitiba, Paraná, BR
guilherme.eduardo1@ufpr.br

Felipe Duarte Silva
Universidade Federal do Paraná
Curitiba, Paraná, BR
felipeduarte@ufpr.br

Gustavo Benitez Frehse
Universidade Federal do Paraná
Curitiba, Paraná, BR
gustavo.frehse@ufpr.br

Rodrigo Machniewicz Sokulski
Universidade Federal do Paraná
Curitiba, Paraná, BR
rodrigosokulski@ufpr.br

Eduardo Todt
Universidade Federal do Paraná
Curitiba, Paraná, BR
todt@ufpr.br

Marco Antonio Zanata Alves
Universidade Federal do Paraná
Curitiba, Paraná, BR
mazalves@ufpr.br

ABSTRACT

With computers processing an ever-increasing amount of data, expanding DRAM capacity is not always feasible due to economic and structural issues. In this context, memory compression emerges as an alternative provided by the ZRAM module, which expands the DRAM's data capacity by using a reserved area to store compressed pages directly into the main memory. Therefore, reducing secondary storage accesses caused by page swaps and potentially reducing the program's execution time. This paper evaluates the ZRAM performance, comparing the speedup obtained when running NAS Parallel Benchmarks under different disksize values for ZRAM, in systems with 8 GB, 4 GB, and 1 GB of DRAM. Also, we compared our baseline system with an older one, to study the effect of the systems improvement over the ZRAM usage impacts. Our results indicate that the impact of ZRAM on the application's execution time increases with the reduction of the available main memory space. We show no variations in speedup with 8 GB of DRAM, and up to 10% improvement in scenarios with 4 GB of memory. In the 1 GB scenario, compression allowed certain benchmarks to achieve speed increases of up to 39%, even with a low memory footprint application, although most others did not benefit. We conclude that ZRAM has a more pronounced impact in memory-constrained environments, and can have a significant impact even in applications with a low memory footprint. The code is available at https://github.com/Guilherme-Eduardo/ZRAM.

KEYWORDS

ZRAM, Compressão de Memória, NPB, *Speedup*, DRAM

1 INTRODUÇÃO

Com a continua demanda por processamento de volumes cada vez maiores de informações, os sistemas computacionais seguiram a mesma tendência de aumento no tamanho da capacidade de suas memórias DRAM (*Dynamic Random Access Memory*). Essa ampliação no armazenamento da memória principal tornou-se um requisito fundamental, visto que programas passaram a apresentar um *memory footprint* crescente [1, 2]. Pode-se citar, como exemplo, softwares desenvolvidos para analisar a previsão climática por meio de cálculos complexos [3], que exigem recursos disponíveis apenas na computação de alto desempenho (HPC). Como a adição de memória principal nem sempre é uma solução viável, seja por limitações físicas ou por fatores econômicos [4], pesquisas foram desenvolvidas para expandir a capacidade da DRAM de maneira alternativa [5, 6].

Nos computadores modernos, o componente chamado MMU (*Memory Management Unit*), que possui implementação em *hardware* e *software*, realiza o mapeamento da memória e a tradução de endereços virtuais para endereços físicos. A MMU também realiza uma etapa de verificação do endereço, evitando acessos não autorizados e garantindo maior segurança e proteção durante a execução de programas. Além disso, ela implementa a técnica denominada Memória Virtual, a qual cria uma extensão transparente do espaço de memória usando a memória secundária. Esse mecanismo permite que os softwares acessem mais memória do que o hardware permite fisicamente, utilizando dispositivos de armazenamento secundário como SSDs (*Solid State Drives*) e discos rígidos. Atualmente, a memória DRAM costuma situar-se no terceiro nível na hierarquia de memória e oferece tempo de acesso mais rápido que os dispositivos de armazenamento secundário [7, 8].

Para a implementação da MMU e da Memória Virtual, duas soluções principais foram propostas: a segmentação e a paginação. A segmentação visa dividir programas em seções de tamanho arbitrário para que o sistema operacional possa realocá-las mais facilmente na memória. Já a paginação é um mecanismo que utiliza apenas um tamanho de seção, chamada página, para tradução de endereços virtuais em endereços reais e para o gerenciamento do espaço de memória. Esse mecanismo de tradução é implementado usando uma tabela de páginas, na qual cada processo tem um espaço para armazenamento de informações, que são gerenciadas pelo *kernel*. Quando o espaço de armazenamento necessário para os processos é maior que o tamanho da memória, parte das seções é temporariamente instalada no armazenamento secundário. Essa operação de troca entre as memórias principal e secundária é denominada *swap*.

A memória principal possui tempo de acesso na ordem de nanosegundos (*ns*), enquanto a memória secundária apresenta latências na ordem de milissegundos *(ms)*. Por esse motivo, essas operações de *swap* se tornam custosas, pois a memória secundária exige mais energia e tempo de acesso que memória principal.

Para mitigar esse problema, foi disponibilizado na versão 3.14 do *kernel Linux* um módulo de *swap* denominado ZRAM, anteriormente chamado *Compcache*. Esse módulo é responsável por

reservar um bloco de armazenamento na DRAM para ser utilizado como memória virtual local. Ele funciona de forma semelhante à memória virtual tradicional: quando a memória principal está prestes a ficar cheia, o *kernel* envia páginas inativas para a área de *swap*. No caso do ZRAM, essas páginas são comprimidas e enviadas para um espaço determinado dentro da DRAM utilizando o módulo *kswapd*. Consequentemente levando a um aumento do desempenho do *swap* em comparação ao tradicional uso de dispositivos de armazenamento secundário [9].

Apesar das vantagens do ZRAM, não foram encontrados estudos que definam os parâmetros mais adequados para obter o melhor desempenho durante a execução de programas. Dentre esses parâmetros, o tamanho do bloco de armazenamento comprimido é essencial. De acordo com ele, o espaço de armazenamento total da DRAM pode ser expandido ou reduzido, assim como o número de operações de *swap*.

Com isso, este artigo visa avaliar o módulo ZRAM no contexto da execução de programas de computação de alto desempenho, com foco em aplicações que simulam *Computational Fluid Dynamics* (CFD), área do conhecimento que trata da simulação numérica de escoamentos, transferência térmica e fenômenos relacionados. A finalidade dos testes realizados é avaliar o impacto do tamanho do bloco de armazenamento disponível no ZRAM, bem como buscar identificar o impacto da porcentagem da DRAM utilizada pelo ZRAM no desempenho do sistema. Para isso, foram realizados testes utilizando 9 dos 12 programas de código aberto presentes no *NAS Parallel Benchmarks* (NPB) [10]. Esses *benchmarks* são divididos de acordo com a complexidade e tamanho de memória definido para a sua execução. Para os experimentos, foi utilizada a classe C por conta da memória requisitada pelos *benchmarks* dessa categoria ser entre 7 MB e 5 GB de DRAM, abrangendo um panorama geral dos tamanhos de memória utilizados nos sistemas considerados. Por fim, os resultados foram submetidos a uma comparação estatística a fim de determinar o desempenho do ZRAM.

2 COMPRESSORES DE MEMÓRIA

A compressão de memória é uma técnica realizada pelo *kernel*, cujo objetivo é reservar um espaço na memória principal, o qual armazena páginas comprimidas, denominadas de *zpage*. Essa área emula um armazenamento secundário que, em caso de *swap*, guarda páginas menos prováveis de serem utilizadas por um determinado programa [2]. Quando essa página for novamente solicitada, ela será descomprimida do bloco de armazenamento e transferida para a área descomprimida da memória principal.

De forma geral, a compressão da memória DRAM busca reduzir o custo de *page swaps* entre a memória principal e a secundária por meio do aumento da capacidade efetiva da DRAM. Além disso, tende a diminuir o consumo energético e a largura de banda necessária para acesso ao disco, reduzindo também o número de operações de entrada e saída (E/S), proporcionando melhor desempenho e contribuindo para o aumento da vida útil da memória secundária [11].

O *kernel Linux* fornece dois recursos de *swap* para a compressão da memória: ZRAM e o Zswap [12]. Ambos os módulos são semelhantes em relação à reserva de um espaço na memória principal para armazenamento de páginas comprimidas.

O ZRAM cria um dispositivo de bloco de armazenamento que funciona como uma área de *swap* compactada diretamente na memória principal, equivalente à troca de páginas convencional.

Enquanto isso, o Zswap age como uma cache de páginas comprimidas entre a memória principal e a área de *swap* do armazenamento secundário. Além disso, consegue transferir as páginas menos usadas para o disco quando o espaço da cache estiver cheio. Diferentemente, o ZRAM funciona como um armazenamento virtual que comprime as páginas, podendo ser utilizado não só como um disco de *swap*, mas também para armazenamento de arquivos temporários (diretório /tmp).

Por outro lado, o ZRAM não possui a capacidade de transferir suas *zpages* para a área de *swap* no armazenamento secundário. Porém, possui uma funcionalidade opcional chamada de *writeback*, que permite gravar páginas ociosas e incompressíveis com o seu algoritmo para algum armazenamento de apoio, ao invés de mantê-las na memória [9, 13, 14].

3 ZRAM

Ao realizar a compressão de páginas, unidade padrão de gerenciamento de memória pelo *kernel*, esse mecanismo torna possível armazenar mais dados do que o tamanho da própria DRAM [15]. Esse armazenamento adicional permite que o sistema recupere páginas de forma mais rápida que técnicas de *swap* tradicionais, pela redução da latência do acesso quando comparado com o disco.

Por outro lado, o *kernel* utiliza os ciclos do processador que ficariam ociosos aguardando o processo de *swap* para serem usados na compressão e descompressão dos dados na memória. Esse processo de compressão/descompressão é custoso e proporcional ao tamanho de cada página. Assim, sistemas que utilizam *huge pages* apresentam *overheads* maiores pelo uso do ZRAM.

O *kernel* utiliza o algoritmo de LRU (*Least Recently Used)* para selecionar as páginas com baixa probabilidade de serem utilizadas em um futuro próximo. Essas páginas são enviadas para o bloco de armazenamento do ZRAM.

Esse processo pode introduzir variações de tempo no acesso às páginas, especialmente conforme o grau de compressibilidade e o algoritmo utilizado. Assim, as diferenças no tempo de descompressão em ZRAM podem ser amplificadas dependendo dos valores e do layout de dados comprimidos, conforme identificado por Schwarzl et al. [16].

O ZRAM também permite a configuração do tamanho lógico do dispositivo de bloco, denominado *disksize*, o qual está relacionado com a quantidade máxima de dados não comprimidos que podem ser armazenados na DRAM. Por exemplo, de acordo com SegmentFault [17], alocar um espaço de 30 GB para o ZRAM com uma taxa de compressão de 2:1 (dois para um) ocuparia apenas 15 GB de memória física, sendo assim, proporcionando um ganho de 15 GB a mais na DRAM. Por outro lado, a medida que alguns aplicativos aumentam seu *memory footprint*, acessando essa memória, a frequência de compressões e descompressões pode se intensificar, exigindo maior uso de CPU. Caso o uso total de memória chegue a ultrapassar dado limiar, processos são selecionados para eliminação por módulos do *kernel* como o OOM (out of memory). [18].

No entanto, ainda não se sabe qual é a melhor configuração do ZRAM para alcançar o melhor desempenho, pois nenhum estudo explorou diferentes configurações para esse propósito.

Porém, no que diz respeito ao tamanho do bloco reservado na memória principal pelo ZRAM, espera-se que ele apresente uma melhora no desempenho em cenários com pouca memória disponível.

A Figura 1 descreve o funcionamento do ZRAM. Primeiramente, o *kernel Linux* possui um serviço, denominado *kswapd*, que monitora a memória e identifica páginas inativas para serem enviadas para a área de *swap*. Em seguida, ao serem transferidas para o ZRAM, essas páginas são comprimidas, fazendo com que sejam armazenadas ocupando um menor espaço. Posteriormente, se uma dessas páginas for necessária novamente, o *kernel* a descomprime e a retorna para a memória principal. No entanto, por padrão, caso o espaço de armazenamento do ZRAM esteja cheio, o programa é interrompido. Por fim, embora o ZRAM suporte o *writeback*, neste trabalho foi utilizada a configuração padrão do mecanismo.

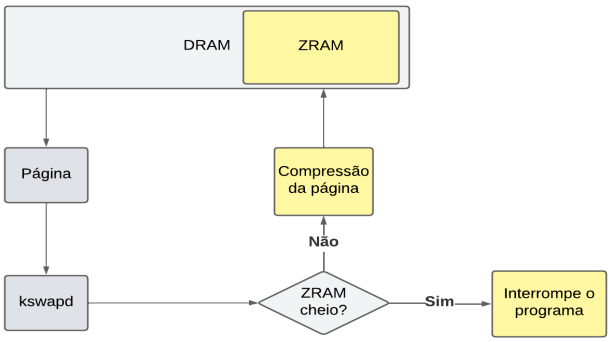

Figura 1: Funcionamento da compressão de páginas com o ZRAM. Fonte: autores.

3.1 Algoritmos de compressão

Atualmente, o *kernel* fornece uma variedade de algoritmos de compressão, incluindo *lzo-rle*, *lz4*, *lz4hc* e *zstd* [13]. Cada um desses algoritmos possui características específicas que influenciam tanto o tempo quanto a taxa de compressão alcançada. Geralmente, existe um equilíbrio entre velocidade e eficiência: os algoritmos que oferecem taxas de compressão mais altas tendem a ser mais lentos, enquanto aqueles que comprimem os dados mais rapidamente normalmente resultam em uma compressão menos eficiente [19]. O *lzo-rle* é utilizado como padrão por equilibrar bem esses dois aspectos, oferecendo uma boa relação entre velocidade e eficiência de compressão.

O *lzo-rle* funciona combinando o algoritmo LZO com a técnica de codificação por comprimento de execução RLE (*Run-Length Encoding*). O LZO é conhecido por sua alta velocidade de compressão e descompressão, sendo ideal para aplicações que exigem desempenho em tempo real. Por sua vez, a codificação RLE é eficaz na compressão de sequências de dados repetitivos, substituindo longas sequências de bytes idênticos por um par que indica o valor do byte e o número de repetições [4]. Ao integrar essas duas abordagens, o *lzo-rle* consegue reduzir significativamente o tamanho dos dados, mantendo uma alta velocidade de processamento. Isso é particularmente útil em sistemas onde a memória é limitada e a eficiência é crucial.

Como a taxa de compressão é o quociente entre o tamanho da página comprimida e seu tamanho original, não é possível prever exatamente seu valor, pois depende do conteúdo dos dados e do algoritmo utilizado. No entanto, observa-se que, em média, a compressão de páginas pode reduzir seu tamanho pela metade, apresentando uma taxa de compressão em torno de 50% [4].

4 TRABALHOS CORRELATOS

Nesta seção, são apresentados diversos estudos que exploram a compressão de memória DRAM. Para a seleção dos artigos, foi realizado um levantamento bibliográfico nas fontes *IEEE Xplore*, *Google Scholar* e *ACM Digital Library*. Nos próximos parágrafos, serão apresentados trabalhos resultantes da pesquisa.

Lee [20] propõe um esquema de troca híbrida utilizando o ZRAM e a memória secundária para dispositivos móveis. Nesse esquema, páginas menos usadas são encaminhadas para a memória secundária, enquanto páginas frequentemente acessadas são transferidas para o bloco do ZRAM. Dessa forma, evita-se o encerramento de processos para liberação de memória e melhora-se a eficiência do sistema.

Schwarzl *et al.* [16] exploraram ataques baseados em algoritmos de compressão utilizados pelo ZRAM para demonstrar vazamento de dados.

Wentong Li *et al.* [21] apresentaram uma interação entre ZRAM e NAND Flash para melhorar o desempenho durante o uso de aplicativos como YouTube e TikTok.

Srividya Desireddy *et al.* [12] propuseram um mecanismo para evitar o envio de páginas duplicadas tanto para o ZRAM quanto para o Zswap. A solução proposta inclui uma verificação da página antes do armazenamento, resultando em uma redução no consumo de memória.

Youngho Choi *et al.* [22] propõem a junção de uma biblioteca denominada DUMA com o ZRAM, para auxiliar na detecção de erros de memória, bem como reduzir o consumo dessa memória.

Chi Gao *et al.* [19] abordam a integração do ZRAM com o acelerador QAT da Intel, que oferece uma taxa de compressão de alto desempenho. Essa integração reduz o consumo de CPU e aumenta o espaço na memória.

Apesar dos estudos apresentados mostrarem resultados importantes relacionados ao ZRAM, obtendo inclusive melhorias no desempenho do sistema, não foi encontrado nenhum trabalho explorando o impacto isolado do tamanho da região de memória reservada ao ZRAM, em ambientes de testes controlados, assim como realizado no presente estudo.

5 METODOLOGIA

Nesta seção, são apresentados os programas utilizados para avaliar a compressão do módulo ZRAM (Seção 5.1) e o ambiente em que os testes foram executados (Seção 5.2).

Nome	Footprint (GB)
BT: "Block Tri-diagonal"	0,675
CG: "Conjugate Gradient"	0,869
EP: "Embarrassingly Parallel"	0,007
FT: "Fourier Transform"	5,014
IS: "Integer Sort"	1,032
LU: "Lower-Upper Gauss-Seidel"	0,547
MG: "Multi-Grid"	3,331
SP: "Scalar Penta"	0,707
UA: "Unstructured Adaptive"	0,471

Tabela 1: Média geométrica de 10 execuções da classe C de aplicações do NPB. Cada aplicação executada com quatro *threads*, assim como nos demais experimentos.

5.1 NAS Parallel Benchmarks

Para avaliar a eficiência do módulo ZRAM, foram utilizados códigos escritos em C e em Fortran agrupados no *NAS Parallel Benchmarks* (NPB). Esse conjunto de *benchmarks*, derivados de aplicações de *Computational Fluid Dynamics* (CFD), é disponibilizado em NASA [10] e fornecido pela *National Aeronautics and Space Administration* (NASA) para avaliar o processamento paralelo de supercomputadores.

O NPB divide seus *benchmarks* em 8 classes, de acordo com sua exigência de processamento e *memory footprint*. A classificação está descrita abaixo:

- S: testes pequenos e rápidos.
- W: *workstation size* (máquina referência dos anos 90).
- A, B, C: testes padrões de tamanho médio.
- D, E, F: testes para servidores de grande porte.

Para os experimentos deste artigo, adotamos a classe C, que é considerada um problema de teste padrão que exige quatro vezes mais memória que a classe anterior (B). Sendo que não foram possíveis testes com a classe D, pois a exigência de recursos de memória dessa classe ultrapassaria o limite definido para os experimentos.

Em sua primeira versão, o NPB era formado por oito *benchmarks*. No entanto ele foi adaptado ao longo de suas atualizações para abranger outros modelos. A sua versão mais atualizada (versão 3.4.3) é formada por doze *benchmarks*. Contudo, este trabalho utiliza apenas 9 deles, sendo os oito da primeira versão e um de computação não estruturada. As suas informações podem ser vistas na Tabela 1, na qual é possível visualizar o nome dos *benchmarks* utilizados e o tamanho do *memory footprint* respectivo.

Os *benchmarks* são disponibilizados em dois modelos de programação: MPI *(Message Passing Interface)* e *OpenMP (Open Multi-Processing)*. O primeiro é utilizado para paralelismo baseado em processos, onde cada processo tem seu próprio espaço e é executado independente dos demais. Já o segundo, refere-se ao paralelismo baseado em *threads*, que significa que as *threads* compartilham os mesmos recursos e acessam a mesma memória [23, 24].

Visto que os experimentos deste trabalho consideram uma única máquina com diversos núcleos físicos, optamos pelo uso da versão com *OpenMP*.

5.2 Ambiente de testes

Os testes foram executados em duas máquinas apresentadas a seguir. A configuração da primeira máquina está dada abaixo, e será referenciada ao longo do texto como **DDR4**.

- **Sistema operacional**: Ubuntu 24.04.1 LTS
- **Kernel**: 6.8.0-49-generic
- **Memória**: 8 GB, DDR4
- **CPU**: Intel(R) Core(TM) i5-7400, 4 threads
- **Frequência da CPU**: 3.00 GHz
- **Compiladores**: GCC e gfortran, 13.2.0.

A segunda máquina possui as configurações descritas abaixo e será referenciada ao longo do texto como **DDR3**.

- **Sistema operacional**: Ubuntu 24.04.1 LTS
- **Kernel**: 6.8.0-49-generic
- **Memória**: 4 GB, DDR3
- **CPU**: Intel(R) Core(TM) i5-2400, 4 threads
- **Frequência da CPU**: 3.10 GHz
- **Compiladores**: GCC e gfortran, 13.2.0.

Para retirar possíveis impurezas dos testes, as opções de *Turbo Boost*, *CPU Frequency Scaling* e *swap* para a memória secundária foram desabilitadas.

Esses ajustes visam reduzir o impacto da CPU ao alterar a frequência do *clock* em diferentes situações dos *benchmarks* e garantir que o *swap* ocorra apenas na região comprimida pelo ZRAM.

5.3 Experimentos

Nesta seção, são apresentados os testes executados a fim de realizar a comparação entre diferentes valores para o parâmetro *disksize* do ZRAM nos ambientes apresentados na seção anterior. O *benchmark NAS*, na versão 3.4.3 no modelo *OpenMP*, foi empregado para avaliar o desempenho.

Durante a execução dos experimentos, o módulo ZRAM foi mantido com suas configurações iniciais e padrões, com exceção do *disksize*, que foi alterado conforme as necessidades do estudo. Além disso, o algoritmo de compressão adotado foi o "lzo-rle", definido como padrão pelo ZRAM. O sistema gráfico foi desativado, não interferindo no consumo de memória.

Os experimentos foram conduzidos com cinco diferentes tamanhos de *disksize* do módulo ZRAM, correspondendo a 0%, 25%, 50%, 75% e 100% do tamanho da DRAM utilizada, o que denominamos de dimensão do *disksize* do ZRAM, ou mais sucintamente, dimensão do ZRAM. Por exemplo, um experimento com dimensão 25% indica que o parâmetro *disksize* do ZRAM corresponde a 25% do tamanho da DRAM. Para garantir consistência na coleta e análise dos dados, as variações foram controladas por um *script* automatizado, responsável por reiniciar o ZRAM e zerar suas estatísticas, conforme demonstrado no Algoritmo 1.

Com base no Algoritmo 1, o experimento consiste em repetir N vezes cada problema. Para os experimentos com 8 GB, utilizamos $N = 100$. Devido a limitações de tempo, para os experimentos com 4 GB DDR4 utilizamos $N = 4$, para o com 1 GB DDR4 utilizamos $N = 2$ e para o com 4GB DDR3 utilizamos $N = 1$.

Nos testes, é feita a execução dos *benchmarks* citados na Subseção 5.1. A cada iteração do ZRAM no Algoritmo 1 é feita a mudança do tamanho de disco do módulo ZRAM considerando a seguinte

Algoritmo 1 Pseudocódigo do *script* para os experimentos. B representa o conjunto de *benchmarks* selecionados, Z o conjunto das dimensões do ZRAM a serem avaliadas e N o número de repetições.

```
1:  Initialization:
2:  for each benchmark in B do
3:      compile(benchmark)
4:      for each zram in Z do
5:          for i = 1 to N do
6:              configure_zram(zram)
7:              run_benchmark(benchmark, zram)
8:              save_results(benchmark, zram)
9:          end for
10:     end foreach
11: end foreach
```

operação:

$$disksize = \frac{\text{Dimensão do ZRAM } \times \text{DRAM disponível}}{100}$$

Portanto, na iteração de 0%, o módulo de ZRAM é totalmente desabilitado a fim de testar o *benchmark* sem nenhum tipo de *swap* ou compressão de memória. Por outro lado, no teste em 100%, o objetivo é avaliar o ZRAM com uma capacidade igual à originalmente fornecida pela memória principal.

Com tudo isto, são adquiridas informações necessárias para a comparação do uso do ZRAM em diferentes situações, além dos benefícios e problemáticas das diferentes dimensões do ZRAM.

6 RESULTADOS

Nesta seção, são apresentados os resultados obtidos em quatro diferentes contextos:

(1) Ambiente com 8 GB de DRAM DDR4.
(2) Ambiente com 4 GB de DRAM DDR4.
(3) Ambiente com 1 GB de DRAM DDR4.
(4) Ambiente com 4 GB de DRAM DDR3.

A análise considera apenas o tempo de execução dos *benchmarks*. O *speedup* de todos os experimentos é calculado em relação ao caso sem ZRAM (*baseline*), exceto no teste de 1 GB, para o qual o *baseline* foi estabelecido como tendo um limite máximo de dados comprimidos correspondendo a 25% da dimensão do ZRAM. Isso foi definido devido à impossibilidade de executar os programas sem compressão de memória, visto que seu *memory footprint* somado à memória necessária para o restante do sistema ultrapassaria 1 GB. A partir disso, temos que o valor do *speedup* representa o quão mais rapidamente (ou não) o sistema executa ao utilizar diferentes configurações de *disksize*.

6.1 8 GB de DRAM DDR4

Como apresentado na Tabela 1, o *memory footprint* dos *benchmarks* da classe C do NPB variam entre aproximadamente 7 MB e 5 GB. Assim, devido à maior disponibilidade de memória em um sistema com 8 GB, não se esperava um impacto do ZRAM.

A Figura 2 apresenta o *speedup* obtido para diferentes dimensões do ZRAM, em relação ao *baseline*. Foram observados incrementos, inferiores a 0,01, em alguns casos, possivelmente por conta de variações do sistema operacional entre execuções.

O *benchmark* EP, que consome menos memória, apresentou *speedup* próximo a 1,03 quando a dimensão do ZRAM foi de 25%. Devido a esse ganho ser mínimo, não é possível afirmar que o ZRAM proporcionou um ganho de desempenho.

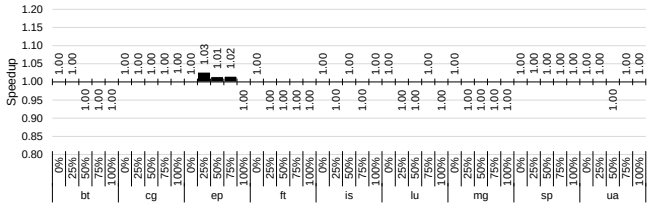

Figura 2: *Speedup* do ZRAM em *benchmarks* da classe C no sistema DDR4 (8 GB de DRAM) em comparação ao *baseline* (sem ZRAM).

6.2 4 GB de DRAM DDR4

Neste teste, foi utilizada a limitação de memória por meio do comando `'mem=nn[KMG]'`. Com apenas 4 GB de DRAM, esperava-se que o ZRAM pudesse exercer maior influência no desempenho, uma vez que a pressão sobre a memória — quando o *hardware* precisa lidar com demandas acima de sua capacidade — tende a ser mais intensa. Os mesmos *benchmarks* foram executados nesse ambiente, tornando-se mais evidente o efeito da memória limitada.

A Figura 3 apresenta o *speedup* obtido. Diferentemente do caso anterior, alguns benchmarks não puderam ser rodados devido ao menor tamanho da memória. Observamos que mesmo nesse ambiente mais restrito, o uso do ZRAM não prejudica o desempenho de nenhuma aplicação.

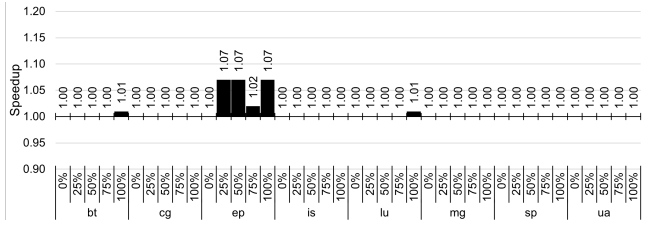

Figura 3: *Speedup* do ZRAM em *benchmarks* da classe C no sistema DDR4 (4 GB de DRAM) em comparação ao *baseline* (sem ZRAM).

O *benchmark* MG demanda aproximadamente 3,3 GB de memória, um valor inferior à capacidade total disponível. Dessa forma, mesmo com a compressão do ZRAM, não há uma redução efetiva da pressão de memória que justifique ganhos de desempenho. Como resultado, o *speedup* permaneceu em 1,0 em relação ao *baseline*, indicando que a compressão não trouxe benefícios para a execução nesse cenário.

Da mesma maneira, o *benchmark* IS também apresentou um *speedup* médio em torno de 1,0, indicando ausência de ganho ou perda de desempenho. Nesse caso, o ZRAM não trouxe benefícios suficientes, visto que o custo da compressão resultou em um *speedup* de execução igual ao *baseline*.

O *benchmark* EP novamente mostrou uma melhora na performance, similar ao observado na máquina com 8 GB de DRAM. No entanto, por ser um ganho menor que 0,08, não é possível descartar a hipótese de que se trate apenas de variação entre execuções.

Um ponto importante é o *benchmark* FT, que necessita de aproximadamente 5 GB. Mesmo com o uso de diferentes dimensões no ZRAM, não foi possível executá-lo completamente. Embora o ZRAM aumente a capacidade de memória, nesse caso, o fator de compressão e o *overhead* do processo não proporcionaram memória suficiente para executar o FT.

6.3 1 GB de DRAM DDR4

Com a intenção de avaliar o ZRAM em um cenário de memória ainda mais limitada, foi definida a configuração de memória DRAM em 1 GB utilizando o parâmetro `'mem=nn[KMG]'`.

O objetivo desse teste é explorar o desempenho do ZRAM e o seu comportamento com *benchmarks* cujos *memory footprints* são próximos ou superiores a esse limite, forçando o uso de compressão. Além disso, é importante destacar que o sistema operacional consome aproximadamente 400 MB, reduzindo ainda mais a memória disponível para as aplicações. Os *benchmarks* FT, CG, IS e MG não puderam ser executados devido ao alto consumo de memória exigido.

A Figura 4 mostra o *speedup* em relação ao *baseline*, que, neste caso, refere-se à compressão de 25% da memória por questões de padronização, pois sem compressão não foi possível executar todos os *benchmarks*. Observa-se que a maior parte dos resultados permanece próxima de 1,0 o que significa que não houve melhora no desempenho. Em alguns casos, os testes foram abortados por falta de memória. Isso ocorreu quando nem mesmo a compressão foi suficiente para acomodar o *memory footprint* do *benchmark*.

Um caso de destaque é o EP, que em certas configurações de compressão conseguiu ser executado e apresentou *speedup* de até 1,39. Apesar do ganho significativo e incremental, para sistemas de 8 GB, 4 GB e 1 GB, esses resultados não podem ser diretamente explicados pelo aumento da capacidade da memória fornecida pelo ZRAM. Isso porque o EP apresenta um *memory footprint* muito menor do que a capacidade da DRAM. Assim, a devida avaliação deste resultado exige estudos mais aprofundados para compreender de maneira consistente sua correlação com o ZRAM.

Além disso, muitos dos resultados não puderam ser coletados devido à interrupção dos programas pela falta de memória.

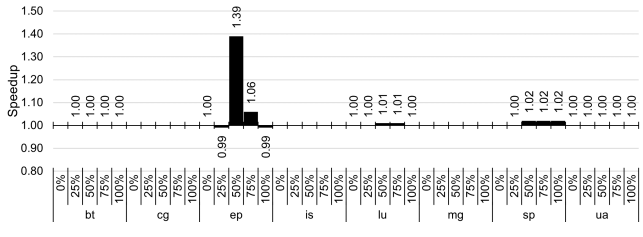

Figura 4: *Speedup* do ZRAM em *benchmarks* da classe C no sistema DDR4 (1 GB de DRAM) em comparação ao *baseline* (25% de compressão).

6.4 4 GB de DRAM DDR3

Para o teste abaixo, o ambiente foi alterado para a máquina **DDR3**, a fim de avaliar a *performance* do ZRAM em um sistema mais antigo, com uma geração anterior de memórias. A Figura 5 apresenta o *speedup* obtido. Ao contrário do caso anterior, notam-se diferenças mais perceptíveis, principalmente em *benchmarks* que demandam mais memória (MG e IS).

O *benchmark* MG, por exemplo, requer cerca de 3,3 GB. Estando próximo do limite físico da DRAM, a compressão do ZRAM ajudou a aliviar a pressão de memória. Como resultado, o MG apresentou um *speedup* médio de 1,10 em relação ao *baseline*. Esse resultado indica que a compressão foi benéfica ao desempenho da execução. Por outro lado, o *benchmark* IS apresentou um *speedup* médio em torno de 0,9, indicando uma queda no desempenho. Nesse caso, o ZRAM não trouxe benefícios, devido ao seu custo de compressão não ter sido mitigado pelo aproveitamento dos dados comprimidos presentes na DRAM, levando a um tempo de execução pior que o do *baseline*.

Acreditamos que isso não acontece no sistema DDR4 devido a seu maior desempenho, que oculta esse custo de compressão, evitando perdas. Além disso, o *benchmark* EP novamente mostrou uma melhora na performance, similar ao observado na máquina DDR4. No entanto, por ser um ganho pequeno, não é possível descartar a hipótese de que se trate apenas de variação entre execuções.

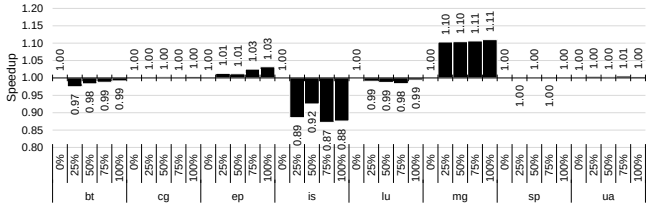

Figura 5: *Speedup* do ZRAM em *benchmarks* da classe C no sistema DDR3 (4 GB de DRAM) em comparação ao baseline (sem ZRAM).

7 Consolidação dos Resultados

A Figura 6 consolida os valores de *speedup* obtidos para cada *benchmark*, em cenários que variam tanto a quantidade de memória DRAM (1 GB, 4 GB e 8 GB) quanto a dimensão de *disksize* do ZRAM (0%, 25%, 50%, 75% e 100%), apenas para o sistema DDR4. Células mais escuras (valor acima de 1,0) indicam ganhos de desempenho em comparação ao caso de referência, enquanto as mais claras apontam desempenho equivalente ou pior. O símbolo "X" representa execuções interrompidas por falta de memória.

No contexto de menor disponibilidade de memória, como 1 GB ou 4 GB, a compressão tende a reduzir a pressão sobre a DRAM, o que gera variações no desempenho de alguns *benchmarks*, levando a ganhos para as aplicações EP e MG e perdas para a IS, que apresenta acessos aleatórios à memória. Entretanto, quando há mais memória disponível, como no caso de 8 GB, o uso do ZRAM não gera impactos sobre as aplicações.

Esses resultados revelam um panorama geral das análises anteriores, que mostram que, quanto maior o tamanho da memória

principal, menos o ZRAM apresenta impactos, positivos ou negativos, para as aplicações. Além disso, em casos específicos, mesmo aplicações com pequeno *memory footprint* como o EP podem ser afetadas por essa técnica, devido a fatores não aprofundados neste artigo.

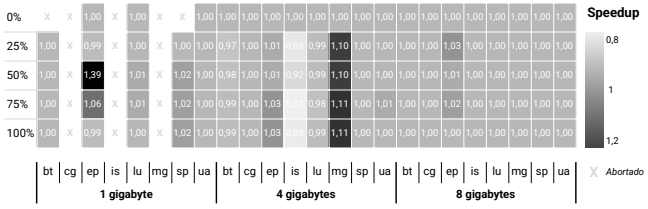

Figura 6: Sumário consolidado dos *speedups* obtidos para cada *benchmark*, no sistema DD4, em diferentes configurações de memória e dimensão do ZRAM, comparados ao *baseline*.

8 CONCLUSÃO

O presente artigo investigou o desempenho do uso do módulo ZRAM sobre os *NAS Parallel Benchmarks* (NPB), geralmente utilizados em ambientes de teste de computação de alto desempenho. Foram analisados cenários que variam tanto a capacidade de memória DRAM quanto valores para as dimensões do *disksize* aplicadas no ZRAM, buscando um melhor entendimento sobre em quais condições o seu uso se torna atrativo. Em vista disso, foi possível observar diferentes comportamentos de acordo com a natureza dos *benchmarks*.

Dentre os resultados obtidos, é possível destacar as seguintes conclusões:

(I) Em ambientes com 8 GB de DRAM (sistema DDR4), o uso do ZRAM não proporciona variações de *speedup* o que sugere que a compressão não foi necessária;

(II) Em cenários de 4 GB de DRAM (sistema DDR4), o uso da compressão de memória mostra-se mais impactante, ocorrendo pequenas variações no desempenho em casos com maior valor de *disksize*, porém sem levar a perdas de desempenho nos *benchmarks* avaliados;

(III) No contexto com apenas 1 GB de DRAM (sistema DDR4), a compressão permitiu a execução de certos *benchmarks*, alcançando *speedup* de até 39% na aplicação *EP*. Por outro lado, as demais aplicações não exibiram variações significativas de desempenho, com algumas sendo interrompidas devido à insuficiência de memória;

(IV) Mesmo em aplicações com baixo *memory footprint*, como o *benchmark EP* [25] podem apresentar ganhos de desempenho com o ZRAM, o que deve ser aprofundado em trabalhos futuros;

(V) Quando consideramos um sistema mais antigo (DDR3), os ganhos e perdas relacionados ao ZRAM se tornaram mais significativos, indicando que essas variações pode estar sendo ocultadas pela execução mais rápida dos componentes presentes em sistemas mais novos (DDR4).

Por fim, pesquisas futuras podem aprofundar os experimentos, testando outras taxas de compressão e analisando diferentes *benchmarks*. Além disso, seria interessante investigar o impacto de outros algoritmos de compressão oferecidos pelo *kernel*, bem como a utilização do módulo *Zswap*, também disponibilizado pelo sistema operacional, para fins de comparação com o ZRAM, de modo a aperfeiçoar as descobertas apresentadas neste artigo.

AGRADECIMENTOS

Este projeto foi parcialmente financiado pelo Ministério da Saúde através de uma TED para PD&I entre SAPS/MS e C3SL/UFPR.

REFERÊNCIAS

[1] Geraldo F. Oliveira, Saugata Ghose, Juan Gómez-Luna, Amirali Boroumand, Alexis Savery, Sonny Rao, Salman Qazi, Gwendal Grignou, Rahul Thakur, Eric Shiu, and Onur Mutlu. Extending memory capacity in modern consumer systems with emerging non-volatile memory: Experimental analysis and characterization using the intel optane ssd. *IEEE Access*, 11:105843–105871, 2023. doi: 10.1109/ACCESS.2023.3317884.

[2] Yuhui Deng, Liangshan Song, and Xinyu Huang. Evaluating memory compression and deduplication. In *2013 IEEE Eighth International Conference on Networking, Architecture and Storage*, pages 282–286, 2013. doi: 10.1109/NAS.2013.45.

[3] Takemasa Miyoshi, Masaru Kunii, Juan Ruiz, Guo-Yuan Lien, Shinsuke Satoh, Tomoo Ushio, Kotaro Bessho, Hiromu Seko, Hirofumi Tomita, and Yutaka Ishikawa. "big data assimilation" revolutionizing severe weather prediction. *Bulletin of the American Meteorological Society*, 97(8):1347–1354, 2016.

[4] LWN.net. In-kernel memory compression. *LWN.net*, 2021. URL https://lwn.net/Articles/545244/.

[5] Raghavendra Kanakagiri, Biswabandan Panda, and Mutyam. Multiblock data compression. *ACM Trans. Archit. Code Optim.*, 14(4), December 2017. ISSN 1544-3566. doi: 10.1145/3151033. URL https://doi.org/10.1145/3151033.

[6] M. Kim, C. Park, M. Han, Y. Han, and S. W. Kim. Epsim: a scalable and parallel marssx86 simulator with exploiting epoch-based execution. *IEEE Access*, 7:4782–4794, 2019. doi: 10.1109/access.2018.2886630.

[7] Andrew S. Tanenbaum and Albert S. Woodhull. *Sistemas Operacionais: Projeto e Implementação*. Pearson Prentice Hall, São Paulo, 3 edition, 2009. ISBN 978-85-7605-373-2.

[8] Narjes Jomaa, David Nowak, Gilles Grimaud, and Samuel Hym. Formal proof of dynamic memory isolation based on mmu. In *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 73–80, 2016. doi: 10.1109/TASE.2016.28.

[9] S. Srividya, S. R. K. Sinha, and S. K. Nandy. Enhanced compressed swap scheme for mobile devices. In *2015 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6, 2015. doi: 10.1109/ANTS.2015.7413650. URL https://ieeexplore.ieee.org/document/7413650.

[10] NASA. Nas parallel benchmarks (npb), 2024. URL https://www.nas.nasa.gov/software/npb.html. Accessed: 2024-11-20.

[11] Jongseok Kim, Cheolgi Kim, and Euiseong Seo. $ezswap$: Enhanced compressed swap scheme for mobile devices. *IEEE Access*, 7:139678–139691, 2019. doi: 10.1109/ACCESS.2019.2942362.

[12] Srividya Desireddy and Dinakar Reddy Pathireddy. Optimize in-kernel swap memory by avoiding duplicate swap out pages. In *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)*, pages 1–4. IEEE, 2016.

[13] Kernel.org. Zram documentation, 2024. URL https://docs.kernel.org/admin-guide/blockdev/zram.html. Accessed: 2024-11-20.

[14] LWN.net. Improving swap performance with zswap, 2013. URL https://lwn.net/Articles/537422/. Accessed: 2024-11-20.

[15] Junyeong Han, Sungeun Kim, Sungyoung Lee, Jaehwan Lee, and Sung Jo Kim. A hybrid swapping scheme based on per-process reclaim for performance improvement of android smartphones (august 2018). *IEEE Access*, 6:56099–56108, 2018. doi: 10.1109/ACCESS.2018.2872794.

[16] Martin Schwarzl, Pietro Borrello, Gururaj Saileshwar, Hanna Müller, Michael Schwarz, and Daniel Gruss. Practical timing side channel attacks on memory compression. In *Proceedings of Conference*. IEEE, 2021.

[17] SegmentFault. Introduction to zram: What is it and how it works. *Segmentfault.com*, 2024. URL https://segmentfault.com/a/1190000041578292/en?decode__1660=n4jxyD2DuDRDgmD0DBw%2BQfYGKexWwF7ye%2BD. Accessed: 2024-11-20.

[18] Linhan Li, Qianying Zhang, Shijun Zhao, Zhiping Shi, and Yong Guan. Design and implementation of oom module based on rust. In *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, pages 774–775, 2022. doi: 10.1109/QRS-C57518.2022.00129.

[19] Chi Gao, Xiaofei Xu, Zhizou Yang, Liwei Lin, and Jian Li. Qzram: A transparent kernel memory compression system design for memory-intensive applications with qat accelerator integration. *Applied Sciences*, 13(18), 2023. ISSN 2076-3417. doi: 10.3390/app131810526. URL https://www.mdpi.com/2076-3417/13/18/10526.

[20] S. Lee, J. Lee, J. Jeong, S. Kim, and J. Huh. A hybrid swapping scheme based on per-process reclaim for performance improvement of android smartphones.

IEEE Transactions on Consumer Electronics, 61(4):502–509, 2015. doi: 10.1109/TCE.2015.7389790. URL https://ieeexplore.ieee.org/document/7389790.

[21] Wentong Li, Dingcui Yu, Yunpeng Song, Longfei Luo, and Liang Shi. Elasticzram: Revisiting zram for swapping on mobile devices. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706011. doi: 10.1145/3649329.3655943. URL https://doi.org/10.1145/3649329.3655943.

[22] Youngho Choi, Dong Hyun Kang, Jaeook Kwon, and Young Ik Eonf. Optimizing memory swapping scheme on the memory debugging platform of ce devices. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–3, 2018. doi: 10.1109/ICCE.2018.8326222.

[23] L. Pan, F. Wang, Z. Zhang, J. Cui, L. Hai, Z. Duan, and X. Ji. Three-dimensional discrete element analysis on tunnel face instability in cobbles using ellipsoidal particles. *Materials*, 12:3347, 2019. doi: 10.3390/ma12203347.

[24] A. Qawasmeh, A. M. Malik, and B. Chapman. Openmp task scheduling analysis via openmp runtime api and tool visualization. *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1049–1058, 2014. doi: 10.1109/ipdpsw.2014.116.

[25] David H. Bailey, Eric Barszcz, John T. Barton, David S. Browning, Robert L. Carter, Leonardo Dagum, Ramesh A. Fatoohi, Paul O. Frederickson, T. Andrew Lasinski, Robert S. Schreiber, Horst D. Simon, V. Venkatakrishnan, and Susan K. Weeratunga. The nas parallel benchmarks (npb), 1994. URL https://www.nas.nasa.gov/assets/nas/pdf/techreports/1994/rnr-94-007.pdf. Accessed: 2024-12-14.