# Melhorando a Afinidade de Memória de Aplicações Científicas em Ambientes NUMA

**Rafael Gauna Trindade, João V. Ferreira Lima, Andrea Schwertner Charão**

[1]Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

{rtrindade,jvlima,andrea}@inf.ufsm.br

***Resumo.*** *Esse trabalho apresenta estratégias encontradas na literatura sobre como identificar e corrigir problemas de desempenho de aplicações em ambientes NUMA e propõe a otimização de 4 aplicações científicas conhecidas: CoMD, LBM, LULESH e Ondes3D. Os resultados mostram que é possível recuperar um pouco do desempenho perdido melhorando a afinidade de memória dessas aplicação quando aplicado um mapeamento de dados eficiente. Entretanto, os resultados evidenciam também que nem todo tipo de aplicação pode obter melhorias a partir desse tipo de abordagem.*

## 1. Introdução

Uma plataforma com memória de acesso não-uniforme (NUMA) é um sistema multiprocessado de memória compartilhada, onde processadores podem acessar um quantia relativamente grande desse tipo de memória em bancos de memória localizados em seu nó NUMA local ou remoto [Diener et al. 2015b]. Em resumo, o desempenho de um aplicação em um ambiente NUMA está relacionado à afinidade entre *threads* e dados. Esse tipo de sistema permite que aplicações paralelas aproveitem uma maior largura de banda de memória se forem projetadas de maneira sensível ao compartilhamento de dados, uma vez que o mapeamento de páginas de memória para nós NUMA pode influenciar o desempenho [Diener et al. 2015b]. Para alcançar baixas latências e maiores larguras de banda de memória, os programadores devem priorizar a afinidade de memória das aplicações, aproximando as tarefas computacionais com os dados que precisam ser acessados, um aspecto não tão importante nas gerações anteriores de microprocessadores, mas crucial em sistemas NUMA [Diener et al. 2015b].

Este trabalho propõe um estudo de caso o qual avalia o desempenho e busca otimizar quatro aplicações científicas bem estabelecidas para um ambiente NUMA. Os objetivos específicos do trabalho são: Extrair estatísticas de desempenho das aplicações com duas políticas de alocação de memória diferentes: *first-touch* (primeiro toque) e *interleaved* (intercalado), usando ferramentas como likwid[1] e numalize[2]; Avaliar métricas como exclusividade de página, tráfego de dados entre nós e tempos de execução; Melhorar a afinidade de memória das aplicações a partir das estatísticas coletadas; Avaliar os resultados pós-otimização.

Este artigo está estruturado da seguinte forma: a Seção 2 detalha o hardware experimental e a metodologia utilizada. A Seção 3 lista trabalhos relacionados à otimização

[1]https://hpc.fau.de/research/tools/likwid
[2]https://github.com/matthiasdiener/numalize

e avaliação de aplicativos em ambientes NUMA. Os resultados experimentais são apresentados na Seção 4. Finalmente, a Seção 5 discute os resultados obtidos e finaliza o trabalho.

## 2. Metodologia

Para otimizar uma aplicação para ambientes NUMA, se faz necessário possuir métricas que avaliem o quão mal a mesma foi projetada e quão bom foi o ganho de desempenho pós-otimização. Alguns trabalhos anteriores criaram ferramentas como a numalize [Diener et al. 2015a] – que ajudam a extrair algumas informações relevantes – e também criaram métricas [Diener et al. 2015b] que ajudam nesse processo, como exclusividade de página. Essa métrica será aplicada e avaliada nas aplicações, em prol de mostrar o estado original do desempenho das mesmas em ambientes NUMA.

### 2.1. Aplicações Científicas

Foram selecionadas 4 aplicações bem conhecidas de dinâmica molecular e de fluídos, com implementações relativamente bem otimizadas para arquiteturas paralelas de memória compartilhada: CoMD[3], LBM, LULESH e Ondes3D[4]. Parâmetros de entrada para tamanho de malha foram escolhidos de forma a terem um tempo de execução próximo a 300 segundos usando 100 iterações (passos) nas implementações originais de cada aplicações no ambiente `blade01`. Os parâmetros escolhidos foram 150x150x150 (CoMD), 8000x8000x9 (LBM), 175x175x175 (LULESH) e 500x500x250 (Ondes3D).

### 2.2. Políticas de Alocação de Memória

O kernel do Linux vem com quatro tipos diferentes de políticas de alocação de memória para ambientes NUMA: *fist-touch*, *interleaved*, *local* e *preferred*. Para este trabalho a política *first-touch* fora escolhida para todas as aplicações. A principal razão é que os programas escritos em plataformas de memória compartilhada não-NUMA não têm nenhuma preocupação em relação às políticas de alocação de memória, e acabam usando essa política, uma vez que é a padrão. Outra razão é que o uso desta política requer poucas modificações no código-fonte quando comparada às outras políticas que exigem chamadas de funções da biblioteca **libnuma**, e embora esta última dê mais flexibilidade na escolha de políticas, suas rotinas de alocação de memória são relativamente lentas.

### 2.3. Hardware

Duas máquinas foram usadas para análise de desempenho: uma máquina com 2 CPUs Intel® Xeon® E5-2420 de 12 *threads* cada, um por nó NUMA (2 nós) e 15.6 GB de memória (`lsc-srv1`), e outra máquina com 8 CPUs Intel® Xeon® E5-4617 de 6 *threads* cada (`blade01`), um por nó NUMA (8 nós) e 488 GB de memória. Em ambos os sistemas há uma conexão ponto-a-ponto entre cada par de nós, e cada nó está um passo de distância de qualquer outro nó.

[3]https://github.com/ECP-copa/CoMD
[4]https://bitbucket.org/fdupros/ondes3d

## 3. Trabalhos Relacionados

Diversos trabalhos focam na otimização e/ou avaliação de aplicações e *benchmarks* para arquiteturas NUMA. No entanto, *benchmarks* não representam a carga de trabalho de aplicações reais, embora sejam comprovadamente úteis para avaliações de desempenho, já que os programas contidos nos mesmos geralmente possuem geralmente um kernel computacional, diferentemente de diversas aplicações científicas.

[Mariano et al. 2016] buscou otimizar uma aplicação grande e irregular para ambientes NUMA: HashSieve, um algoritmo de criptografia baseado em malha. Por ser relacionado ao uso de memória, o desempenho do HashSieve é comprometido em ambientes NUMA. Melhorou-se o tempo de execução final em até 2 vezes usando diversas técnicas, incluindo mapeamento de páginas de memória e – especialmente – mudanças relativamente grandes no código-fonte. No entanto, acreditamos que seria ideal usar abordagens menos intrusivas o possível em relação ao código-fonte, uma vez que certas modificações podem remover algumas das características como portabilidade e manutenibilidade.

[H. M. Cruz et al. 2018] menciona em seu trabalho um esforço para otimizar a aplicação Ondes3D para ambientes NUMA. Usando mapeamento de dados e *threads*, um ganho de desempenho de mais de 200% fora obtido após as modificações relacionadas à afinidade de memória. Neste trabalho tentaremos reproduzir parcialmente o desempenho obtido usando somente mapeamento de dados.

## 4. Resultados Experimentais

As aplicações foram executadas nos dois ambientes previamente descritos na Seção 2, bem como o uso de todas as *threads* disponíveis e a política de alocação *first-touch*. Uma análise da exclusividade de página das versões originais pode ser conferida na Figura 1.

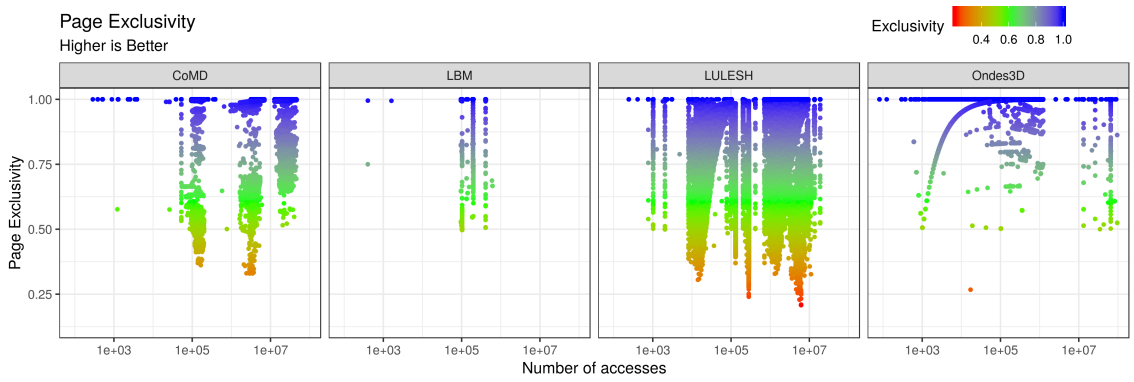


**Figura 1. Exclusividade de página no ambiente `blade01` com 48 *threads*. Cada ponto nos gráficos representa uma página de memoria.**

Com a Figura 1 podemos inferir que as aplicações LBM e Ondes3D são mais passivas de melhoria através do mapeamento de dados, ao contrário das duas demais, que possuem um grande número de páginas com exclusividade baixa. Ao analisar também a localidade dos dados, percebe-se que as duas aplicações podem se beneficiar de um mapeamento de dados mais adequado ao padrão de acesso feito pelo núcleo da aplicação, pois a maior parte de seus dados fica concentrada em um nó NUMA somente.

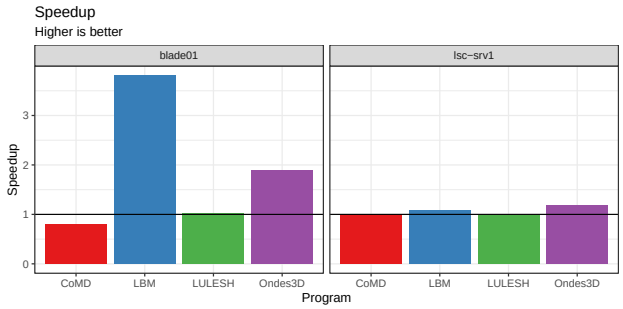


**Figura 2. Melhoria de desempenho obtida após o mapeamento de dados**

A Figura 2 exibe a aceleração obtida pelas aplicações após alterações via código na forma como os *buffers* utilizados são inicializados. A nova inicialização é feita de forma paralela com o auxílio da biblioteca OpenMP. Nem todas as aplicações conseguiram melhorias no desempenho, entretanto, tendo destaque a aplicação LBM no ambiente `blade01`, com aceleração de $\approx 3.8x$ quando comparado a versão paralela original.

## 5. Discussão

Este trabalho exibe uma análise de afinidade de memória em aplicações científicas diferentes e os efeitos de um conciente mapeamento de dados. Constatou-se que é possível ganhar desempenho somente com o mapeamento, tomando cuidado com a forma como os dados são inicializados em tempo de execução, uma vez que hábitos de programação para ambientes não-NUMA tendem a fazer a aplicação acumular seus dados em um nó NUMA, degradando o desempenho. Para a aplicação se beneficiar de um mapeamento, é necessário que ela tenha um padrão regular de acessos à memória entre suas *threads*. Aplicações com padrão de acesso irregular tendem a não obter melhorias apenas com este tipo de mapeamento.

## Agradecimentos

Este trabalho foi financiado pelo CNPq.

## Referências

Diener, M., Cruz, E. H., Pilla, L. L., Dupros, F., and Navaux, P. O. (2015a). Characterizing communication and page usage of parallel applications for thread and data mapping. *Performance Evaluation*, 88-89:18 – 36.

Diener, M., Cruz, E. H. M., and Navaux, P. O. A. (2015b). Locality vs. balance: Exploring data mapping policies on numa systems. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 9–16.

H. M. Cruz, E., Diener, M., and O. A. Navaux, P. (2018). *Thread and Data Mapping for Multicore Systems: Improving Communication and Memory Accesses*. Springer International Publishing.

Mariano, A., Diener, M., Bischof, C., and Navaux, P. O. A. (2016). Analyzing and improving memory access patterns of large irregular applications on numa machines. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 382–387.