Otimizando Algoritmos de Machine Learning com Mapeamento de Threads e Dados*

Matheus S. Serpa, Arthur M. Krause, Eduardo H. M. Cruz, Philippe O. A. Navaux

[1] Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970, Porto Alegre – RS – Brasil

{msserpa, amkrause, ehmcruz, navaux}@inf.ufrgs.br

Resumo. *Impulsionada pelo desenvolvimento de novas tecnologias, como carros autônomos, o aprendizado de máquina tornou-se rapidamente um dos campos mais ativos da ciência da computação. Neste artigo, nos concentramos no mapeamento de threads e dados para o Intel Xeon Phi Knights Landing. Estudamos o impacto das estratégias de mapeamento, revelando que, com políticas de mapeamento inteligentes, pode-se reduzir o tempo de execução em até 18,5%.*

1. Introdução

O aprendizado de máquina é um dos tópicos da Ciência da Computação com maior crescimento, ajudando diversos campos do conhecimento devido a sua abordagem para a compreensão de grandes conjuntos de dados. Normalmente, algoritmos de aprendizado de máquina são acelerados através de *Graphics processing units* (GPUs). No entanto, os novos processadores de muitos núcleos podem ser uma escolha melhor para essas cargas de trabalho paralelas, que anteriormente eram dominadas por GPUs [Witten et al. 2016].

Nesses processadores, *threads* costumam compartilhar dados, os quais são enviados através das interconexões *intra* e *inter-chip*, dependendo da localização das *threads* e dos dados. Como consequência, mapear *threads* que se comunicam mais entre si próximas umas das outras na hierarquia de memória é uma forma de reduzir a latência de comunicação. Além disso, os dados podem estar alocados em bancos locais ou remotos, sendo que ler dados de um banco remoto resulta em latências mais altas.

Este artigo tem como objetivo mostrar que algoritmos de aprendizado de máquina podem ser acelerados utilizando um mapeamento de *threads* e dados inteligente. Para tanto, avaliamos diferentes políticas de mapeamento no Xeon Phi "Knights Landing".

2. Metodologia

A análise dos trabalhos relacionados mostra diversos trabalhos utilizando mapeamento de *threads* e dados para melhorar o desempenho de aplicações paralelas. Cruz et al. [Cruz et al. 2016] apresenta um método que usa o tempo que cada entrada fica na TLB para fazer o mapeamento. Esse método não leva em consideração a memória rápida presente no Xeon Phi. You et al. [You et al. 2017] refatora algoritmos de *machine learning* com objetivo de melhorar a comunicação. Esses trabalhos são limitados a algoritmos e arquiteturas específicas.

*Este trabalho foi parcialmente financiado por recursos do projeto HPC4E (www.hpc4e.eu), financiamento nº 689772 do acordo internacional entre o programa H2020-EU e o MCTI/RNP-Brasil. Também foi financiado pela Intel sobre o projeto Modern Code e Petrobras.

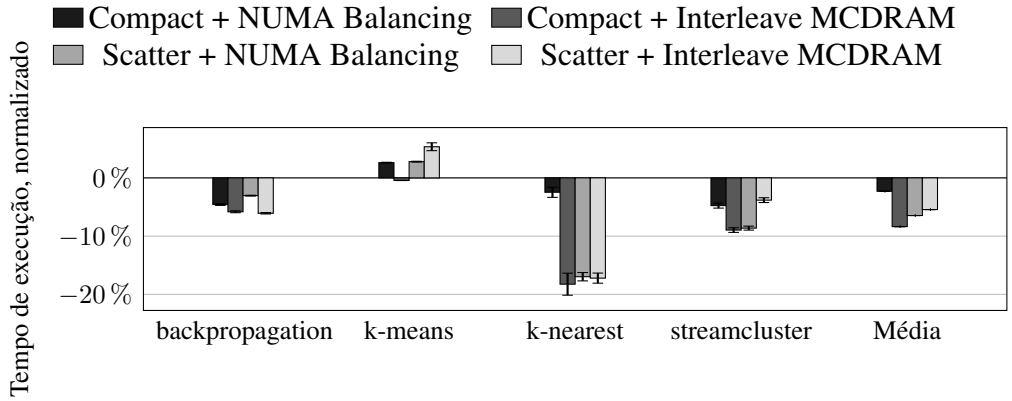

Figura 1. Mapeamento de *Threads* e Dados no Knights Landing.

O objetivo das políticas de mapeamento é melhorar o uso dos recursos, sendo que cada política foca em um aspecto. A política *compact thread* foca em agrupar *threads* vizinhas em *cores* próximos na hierarquia. Enquanto a *scatter thread* foca no inverso. O mapeamento de dados *NUMA Balancing* migra páginas para o nó da última *thread* que realizou o acesso. A política *Interleave* aloca páginas consecutivas em nós consecutivos.

Para os experimentos, uma máquina Knights Landing, com um processador Intel Xeon Phi 7250 de 68 *cores* físicos, que permite execução de 272 *threads*, foi utilizada. Esse processador também tem uma memória rápida (MCDRAM) de 16 GB. Os experimentos são a média de 30 execuções aleatórias. O desvio padrão foi calculado pela distribuição *t-student* com intervalo de confiança de 95%. Os algoritmos foram retirados do *benchmark* Rodinia em linguagem C paralelizados com OpenMP.

3. Resultados Preliminares e Conclusão

A Figura 1 apresenta os resultados sendo que o maior ganho de desempenho foi de 18.5% para um mapeamento de *threads scatter* com um mapeamento de dados *interleave MCDRAM* no algoritmo *k-nearest*. Nesse mapeamento, existem quatro nós NUMA para os quais as páginas são distribuídas. O mapeamento foi ineficaz para o algoritmo *k-means*, pois a maior parte do tempo de execução desse algoritmo é devido a entrada e saída.

Os resultados mostraram que, para algoritmos de *machine learning*, políticas que focam no balanceamento de carga são melhores. Outro ponto é que utilizar a memória rápida (MCDRAM) do Xeon Phi apresenta resultados melhores.

Referências

[Cruz et al. 2016] Cruz, E. H., Diener, M., Alves, M. A., Pilla, L. L., and Navaux, P. O. (2016). Lapt: A locality-aware page table for thread and data mapping. *Parallel Computing*, 54:59–71.

[Witten et al. 2016] Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[You et al. 2017] You, Y., Buluc, A., and Demmel, J. (2017). Scaling deep learning on gpu and knights landing clusters. *arXiv preprint arXiv:1708.02983*.