Análise de desempenho do cálculo matricial em sistemas paralelos utilizando AVX-512

André Libório[1], **Alexandro Baldassin**[1], **João Paulo Papa**[1],

[1]Universidade Paulista Júlio de Mesquita Filho (UNESP) - Brasil

{andre.lb.ferraz, alexandro.baldassin, joao.papa}@unesp.br

Abstract. *Due to software optimization processes arising from more recent technologies, this study seeks to analyze the advantage of hardware-based vectorization implementations, i.e., AVX2 and AVX-512, in a matrix multiplication scenario. The results show that vectorization brings very expressive gains, highlighting the AVX-512 advantages.*

Resumo. *Em virtude dos processos de otimização em software advindos de tecnologias mais recentes, este estudo busca analisar a vantagem trazida por implementações de vetorização baseadas em hardware, neste caso, AVX2 e AVX-512, em um cenário de multiplicação matricial. Os resultados apresentam que a vetorização proporciona ganhos bastante expressivos, destacando-se o AVX-512.*

1. Introdução

Com a popularização e avanço dos códigos baseados em aprendizado de máquina nas últimas décadas, surgiu a necessidade de criar soluções distintas que melhorem seu desempenho, permitindo a execução de algoritmos mais complexos, assim como permitir a utilização de tais tecnologias em sistemas computacionais portáteis como *smartphones* e *tablets*. Com o advento da popularização de sistemas computacionais com processadores de múltiplos núcleos, a paralelização passa a ser essencial para otimização de *softwares* com foco em desempenho como, por exemplo, OpenMP, tornando o processo bastante acessível ao desenvolvedor [M. Müller 2009].

Este trabalho tem o intuito de apresentar os benefícios de um outro mecanismo de otimização, conhecido como *vetorização*. Em particular, utiliza-se a extensão proposta pela Intel conhecida como *Advanced Vector Extensions* (AVX) em duas versões: o AVX2 e o AVX-512 [Cornea 2015]. Será apresentado um estudo inicial que investiga o ganho da vetorização aliada ao *multithreading* em uma aplicação de multiplicação de matrizes. Essa investigação insere-se no contexto de um projeto posterior no qual será investigado o ganho proporcionado pela vetorização em algoritmos para aprendizado de máquina [Capra et al. 2020].

Foi utilizada a multiplicação de matrizes em precisão dupla para medir o desempenho das operações e comparou-se os diversos casos de uso, abrangendo as variações das caracterísiticas de vetorização e paralelização. Este trabalho é uma extensão de um trabalho anterior no qual investigou-se o impacto das diversas abordagens para a paralelização da multiplicação de matrizes [Libório 2021].

Este trabalho se inicia apresentando as estratégias de vetorização e paralelização básica na Seção 2, seguida pela avaliação experimental na Seção 3 e a conclusão na Seção 4.

2. Vetorização

A vetorização é uma maneira encontrada para otimizar processamento de vetores, de maneira que, com um conjunto de registradores especializados em cada um dos núcleos de um processador, uma única instrução possa ser aplicada a todo o vetor de dados. O AVX foi uma tecnologia popularizada pela Intel em 2011 com a arquitetura Sandy Bridge, com a finalidade de contribuir com a inovação tecnológica iniciada pela criação do conjunto de instruções MMX em 1997. Diferentemente do MMX, que contém registradores de 64-bits, o AVX2 tem suporte para registradores de até 256-bits e, em 2016, houve um outro incremento nessa tecnologia com a arquitetura Skylake da Intel, a criação do AVX-512, com registradores de 512-bits (zmm), que elevaram o potencial da tecnologia. A utilização de tais registradores, por sua vez, não é tão trivial quanto sua lógica básica. As instruções devem ser específicas para cada tecnologia e programadas sob a linguagem C/C++. Neste caso, por meio de *intrinsics* ou até mesmo utilizando a linguagem de montagem, que permite um maior nível de otimização para o código. Aqui, nos atemos à utilização da linguagem de programação C para todos os testes realizados.

A vetorização também se beneficia do alinhamento dos dados em blocos de 64 bytes (tamanho da linha de cache). Portanto o alinhamento, quando realizado adequadamente, pode aumentar a eficiência do carregamento e armazenamento de dados. O algoritmo apresentado no Pseudo-código 1 utiliza a estrutura tradicional de multiplicação matricial, ou seja, a operação é composta por três laços da estrutura de repetição *for*, de maneira que a multiplicação das matrizes quadradas A e B resulte na matriz C, de mesmas dimensões. No caso da vetorização, é utilizado o suporte de funções intrínsicas fornecidas pelo compilador para realização da multiplicação vetorial. Esta operação é mostrada de forma simplificada entre as linhas 8 e 11 do pseudo-código.

Para realizar a multiplicação vetorial, a quantidade de dados que deve ser carregada depende do tipo do dado e também da largura do vetor. No exemplo, temos dados de 64 bits (*double*) e a largura do vetor de 256 (AVX2) ou 512 (AVX-512). Assim, o número de colunas da matriz B (linha 5) é divido pela largura do vetor (LV) correspondente: 4 (256/64) para AVX2 e 8 para AVX-512 (512/64).

3. Experimento

3.1. Metodologia

O sistema utilizado para a execução dos experimentos realizados é equipado com dois processadores Intel Xeon Gold 5220, totalizando 36 núcleos físicos e 72 lógicos, com o sistema operacional CentOS 7.7. Foi utilizado também o compilador GCC 7.3.1 e as variáveis de ambiente CFLAGS com o switch de otimização -O3 para fins de otimização de compilação. Foi empregada a política de alocação de *threads close*, que busca utilizar *threads* fisicamente mais próximas, ou seja, no mesmo *socket*, aumentando a localidade dos dados e possibilitando uma menor latência no acesso à memória compartilhada.

Para assegurar a consistência dos testes, o tempo de execução das operações é calculado por meio da função *gettimeofday*, da biblioteca time.h [Rathore and Kumar 2014]. Os valores apresentados neste estudo são baseados na média de 30 execuções realizadas para cada configuração apresentada, com valores de ponto flutuante (*float*), com intervalos de confiança de 95%. O *speedup* foi obtido dividindo-se o tempo de execução sequencial pelos tempos com diferentes números de *threads*.

Pseudo-código 1. Multiplicação de matrizes com AVX

```
1  inicio
2    // matrizes: A, B e C
3    // vetores AVX: Aavx, Bavx, Cavx, aux
4    para linha de 0 ate numero de linhas de A faca
5      para coluna de 0 ate numero de colunas de B/LV faca
6        Cavx = 0;
7        para i de 0 ate numero de colunas de A faca
8          Aavx = conteudo da matriz em A[linha][i];
9          Bavx = conteudo da matriz em B[i][coluna];
10         aux = multiplicacao dos vetores Aavx e Bavx;
11         Cavx = soma elementos de aux e Cavx;
12       fimpara
13       armazena dados do vetor Cavx em C[i][j];
14     fimpara
15   fimpara
16 fim
```

Para os experimentos, foram utilizadas matrizes quadradas com tamanhos de 4000x4000 elementos, escolhidas pelo tempo razoável de execução, adequadas a produção dos experimentos.

3.2. Resultados

Os resultados da Figura 1 apresentam o *speedup* do código paralelizado, seguindo as configurações OpenMP, OpenMP+AVX2, OpenMP+AVX-512, para as matrizes de 4000x4000 elementos. O número de *threads* varia de 1 até 72.

Pela Figura 1 é possível observar que, mesmo no caso com apenas uma *thread*, existe uma diferença considerável de desempenho entre as três configurações. Utilizando a configuração sem vetorização como base, observa-se um *speedup* de 3,3 vezes em relação ao AVX2 e 3,9 vezes em relação ao AVX-512. Isso mostra que, mesmo obtendo um melhor desempenho, os benefícios de uso do AVX-512 não são completamente explorados nesse cenário (uma thread).

Os resultados com múltiplas *threads* evidenciam que, em geral, as versões do código que também utilizam vetorização tem desempenho maior do que a versão apenas com OpenMP. Há uma pequena queda de desempenho entre 36 e 42 *threads* devido ao fator NUMA, já que na configuração com 42 *threads* os cores do outro socket começam a ser utilizados. Outro comportamento anômalo se refere às configurações de 30 e 36 threads. Nesses casos, o desempenho do AVX2 é ligeiramente superior ao AVX-512. Observamos, também, que a partir de 48 *threads,* o desempenho do AVX-512 é bastante superior ao do AVX2. Atualmente estamos investigando o motivo para tal comportamento.

4. Conclusão

Neste estudo foi realizada uma análise desempenho de aplicações vetoriais paralelas com AVX2 e AVX-512 em um caso de multiplicação matricial. Por meio dos resultados obtidos, pode-se afirmar que a vetorização é uma estratégia viável em termos de desempenho,

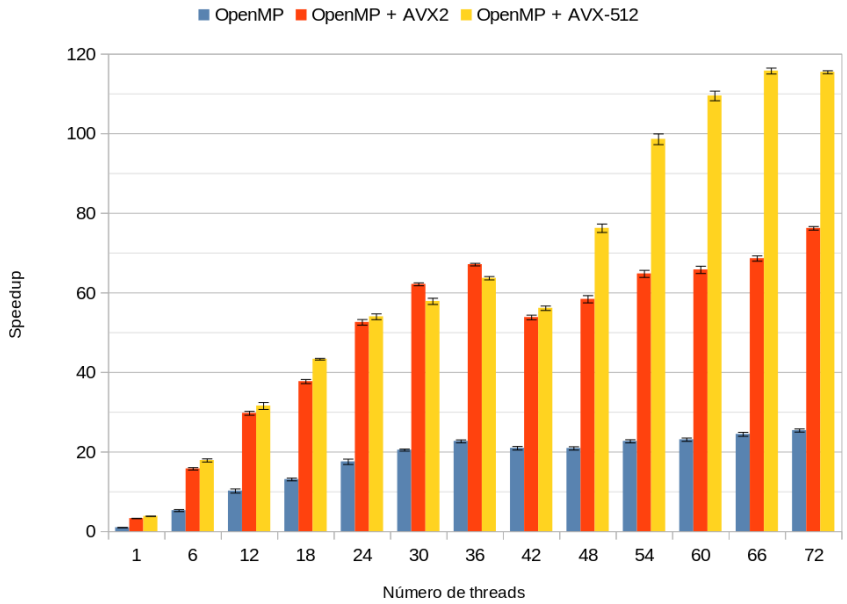

Figura 1. Gráfico de *speedup* das condições de vetorização com matrizes quadradas de tamanho 4000

principalmente quando considerado sua variável AVX-512. Em geral, a vetorização, junto com a paralelização, se mostraram benéficas para a aplicação estudada. Como próximo passo pretendemos aplicar a vetorização para otimizar a implementação de um algoritmo de aprendizado de máquina.

Referências

Capra, M., Bussolino, B., Marchisio, A., Masera, G., Martina, M., and Shafique, M. (2020). Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access*, 8:225134–225180.

Cornea, M. (2015). Intel avx-512 instructions and their use in the implementation of math functions. *Intel Corporation*, pages 1–20.

Libório, André e Baldassin, A. (2021). Análise de desempenho do cálculo matricial em sistemas paralelos utilizando openmp. In *Anais da XII Escola Regional de Alto Desempenho de São Paulo*, pages 13–16. SBC.

M. Müller, B. Supinski, B. C. (2009). *Evolving OpenMP in an Age of Extreme Parallelism*. Springer.

Rathore, Y. and Kumar, D. (2014). Performance evaluation of matrix multiplication using openmp for single dual and multi-core machines. *IOSR Journal of Engineering (IOSRJEN)*, 4:56–59.