Análise de Desempenho entre Arquiteturas TPU e GPU utilizando PyTorch e JAX para IA e HPC

Jose Augusto M. de Lima[1], **Calebe P. Bianchini**[1,2]

[1]Programa de Pós-Graduação em Computação Aplicada – PPGCA/FCI
Universidade Presbiteriana Mackenzie – São Paulo, SP — Brasil

[2]CESAR – Centro de Estudos e Sistemas Avançados do Recife
Recife, PE – Brasil

joseaugustomarcellino.lima@mackenzista.com.br

calebe.bianchini@mackenzie.br, cpb@cesar.org.br

Resumo. *Este trabalho apresenta uma análise técnica da arquitetura Tensor Processing Unit (TPU), comparando-a com aceleradores GPU tradicionais em tarefas de computação de alto desempenho. Além de microbenchmarks de multiplicação de matrizes densas via JAX e PyTorch sobre o compilador XLA, o estudo introduz uma avaliação da largura de banda de memória (HBM - High Bandwidth Memory). Os resultados demonstram que a TPU supera a GPU de referência não apenas no tempo de execução, mas também na vazão de dados, consolidando se como uma infraestrutura eficiente para aplicações científicas intensivas em dados.*

1. Introdução

O avanço da computação científica e da Inteligência Artificial (IA) tem sido impulsionado pelo desenvolvimento de aceleradores de *hardware* especializados. Enquanto as Unidades de Processamento Gráfico (GPU – *Graphics Processing Unit*) consolidaram-se como o padrão de mercado devido à sua flexibilidade e ecossistema de *software* maduro [Silva et al. 2022], as Unidades de Processamento de Tensor (TPUs – *Tensor Processing Unit*) surgiram como dispositivos dedicados a otimizar operações tensoriais. Para contextualizar, dados tensoriais são estruturas matemáticas multidimensionais fundamentais que representam os parâmetros (pesos) e as ativações de modelos matemáticos.

Compreender o comportamento dessas arquiteturas em operações fundamentais, como a multiplicação de matrizes, e na vazão de memória, é crucial para otimizar *workloads* científicos. Embora comparações entre arquiteturas de aceleradores já existam na literatura em trabalhos como os de *Jouppi et al.* (2017) e *Kumar e Kasivajhula* (2022), a principal contribuição deste trabalho reside na avaliação empírica da TPU v6e (Trillium), recentemente disponibilizada na região da América do Sul pelo Google Cloud, contrapondo-a diretamente à GPU NVIDIA A100.

Este artigo investiga o desempenho entre ambas arquiteturas, através do uso de microbenchmarks em operações fundamentais de álgebra linear. O estudo concentra seus esforços no uso dos *frameworks* PyTorch e JAX, além do compilador XLA como ponte de eficiência entre o *software* e o *hardware* especializado. A investigação usando

PyTorch, JAX e XLA apresenta-se como uma combinação promissora, oferecendo suporte prático a aceleradores com otimização via compilação JIT (*Just-In-Time*) para operações tensoriais [Wang et al. 2019].

2. Arquiteturas de *Hardware*: GPU vs TPU

A diferenciação entre GPUs e TPUs reside em suas filosofias de projeto [Nikolić et al. 2022]. A GPU utiliza uma arquitetura SIMT (*Single Instruction, Multiple Threads*), projetada para paralelismo generalista. Sua força reside na flexibilidade para executar *kernels* customizados, mas enfrenta gargalos de transferência de dados entre a memória e as unidades de cálculo (Gargalo de *Von Neumann*) em operações matriciais densas [Silva et al. 2022].

Em contrapartida, a TPU utiliza a arquitetura de *Systolic Array* (Arranjo Sistólico) em suas Unidades de Multiplicação de Matrizes (MXU). Conforme detalhado por [Jouppi et al. 2017], essa arquitetura foi desenhada especificamente para a matemática de tensores, permitindo que os pesos das matrizes permaneçam estáticos no chip enquanto os dados fluem através das células de processamento, reduzindo drasticamente os acessos à memória e o consumo de energia por operação. Essa especialização permite que a TPU atinja uma utilização real de *hardware* (MFU – *Model FLOPs Utilization*) superior quando comparado à GPU [Wang et al. 2019].

A utilização de memória de alta largura de banda (HBM - *High Bandwidth Memory*) é essencial para manter essas unidades saturadas de dados. Uma forma de visualizar as diferenças entre essas arquiteturas pode ser vista na Figura 1, que evidencia essas distinções. A arquitetura da GPU foca no gerenciamento de uma hierarquia de memória complexa (Caches L1/L2 e SMs) para alimentar threads independentes. A TPU, por sua vez, aloca a maior área do silício para *buffers* de ativação e registradores diretos de acumulação acoplados à MXU, o que reduz drasticamente os acessos à memória principal e maximiza a utilização real do *hardware*.

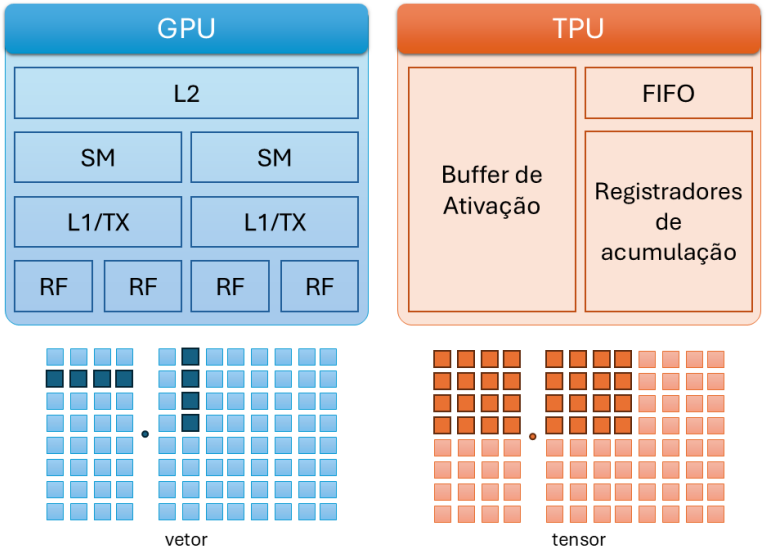

Figura 1. Hierarquia de memória e sistema de computação da GPU e da TPU, adaptado de [Nikolić et al. 2022].

3. Frameworks e Compilação: PyTorch, JAX e XLA

O desempenho desses aceleradores é orquestrado por *frameworks* que visam abstrair a complexidade do *hardware* especializado. O PyTorch adota um modelo de execução "preguiçosa" (*Lazy Tensor*) [Suhan et al. 2021] para interagir com compiladores de domínio específico. Essa abordagem adia a execução das operações, registrando-as em um grafo de computação construído sob demanda.

O JAX, por sua vez, baseia-se em princípios de programação funcional e utiliza a compilação JIT para gerar um grafo completo antes da execução. O XLA (*Accelerated Linear Algebra*) é o compilador intermediário comum a ambos. Sua principal técnica é o *kernel fusion*, que agrupa múltiplas operações em um único passo de computação para reduzir a largura de banda de memória necessária e otimizar o fluxo de dados no *hardware*.

4. Metodologia Experimental

Os experimentos foram executados em ambiente de nuvem do *Google Cloud Platform*. Foram elaborados *microbenchmarks* a partir de receitas conhecidas[1], aplicando três rotinas em sequência: adição de tensores, multiplicação de matrizes e simulação "neural". Para garantir a equivalência e a precisão da comparação, todos os testes operaram com o tipo de dado *FP32*.

Cada bateria de testes contemplou 20 execuções sucessivas para consolidar uma medição mitigada de ruídos sistêmicos, precedidas por uma fase de *warm-up* para estabilização do compilador. Foram estipuladas duas dimensões de matrizes quadradas para induzir diferentes taxas de ocupação de memória: 16k x 16k (carga regular) e 32k x 32k (carga extrema). A carga de 32k é particularmente relevante, pois representa ordens de grandeza reais frequentemente encontradas em projeções de matrizes de atenção em Modelos de Linguagem de Larga Escala (LLMs) modernos.

Durante a avaliação da memória de alta largura de banda (HBM), utilizou-se um script focado em multiplicação de matrizes com 16.777.216 elementos cada (32MB), exclusivamente para medir a vazão real da memória principal em GBps, isolando o barramento de memória de operações de processamento.

No ambiente TPU, utilizou-se uma instância com 1 *chip* TPU v6e-1 (32 GB HBM). Para a GPU, utilizou-se uma instância com 1 *chip* NVIDIA A100 (40 GB HBM). A escolha destes modelos como equivalentes, baseia-se na paridade arquitetural de capacidade de memória e largura de banda teórica, onde ambas fornecem aproximadamente 1,6 TB/s. Os *frameworks* utilizados foram PyTorch 2.5.1 e JAX 0.4.30, ambos operando sobre o compilador XLA.

5. Resultados e Discussão

A Tabela 1 consolida os tempos totais de execução (incluindo *warm-ups*) obtidos nas baterias de testes para a operação de multiplicação de matrizes.

Os dados revelam que a TPU superou a GPU em todos os cenários testados. Para matrizes de 16k, a TPU foi aproximadamente 10 vezes mais rápida com PyTorch

[1]Veja mais sobre essas receitas em `https://github.com/joseaugustolima/ai-infra/`

Table 1. Comparativo de Tempo Total de Execução (em segundos)

Framework	*Hardware*	Matriz 16k	Matriz 32k
PyTorch	TPU (v6e-1)	2,93 s	16,45 s
JAX	TPU (v6e-1)	5,15 s	17,06 s
PyTorch	GPU (A100)	29,85 s	238,76 s
JAX	GPU (A100)	18,25 s	159,83 s

em comparação à GPU. Na carga extrema de 32k, a vantagem da TPU acentuou-se significativamente, completando a tarefa com PyTorch em 16,45s, enquanto a GPU demandou 238,76s.

Na avaliação da largura de banda de memória (HBM), a GPU atingiu uma mediana de 365,08 GBps, ao passo que a TPU v6e-1 alcançou 1.453,77 GBps. Essa superioridade de aproximadamente 4 vezes na vazão, confirma a capacidade da TPU de manter suas Unidades de Multiplicação de Matriz (MXU) operando sem ociosidade, minimizando a necessidade de tráfego de dados intermediários.

A diferença de desempenho observada nos *microbenchmarks*, especialmente na multiplicação de matrizes de carga extrema, encontra explicação na filosofia de *design* das Arquiteturas de Domínio Específico (DSAs). Conforme analisado por [Jouppi et al. 2020], as TPUs são projetadas para maximizar o rendimento em álgebra linear densa ao priorizar Unidades de Multiplicação de Matrizes (MXUs) em detrimento de recursos de computação de propósito geral.

Isso significa, por exemplo, que enquanto as GPUs precisam gerenciar caches complexos e paralelismo generalista, a arquitetura da TPU permite que os dados fluam através de arranjos sistólicos, mantendo as MXUs ocupadas com uma utilização real de *hardware* muito próxima do pico teórico. Esse foco arquitetural em reduzir o custo de movimentação de dados e sua concepção voltada a um conjunto restrito de operações é o que permite à TPU processar volumes massivos de tensores de forma significativamente mais eficiente que aceleradores convencionais [Jouppi et al. 2017].

6. Considerações finais

Este estudo preliminar demonstra que a arquitetura TPU (v6e) oferece um bom desempenho para processamento matricial, superando aceleradores GPU de referência ao realizar experimentos de *microbenchmarks*. A especialização do *hardware* sistólico presente nas TPUs se mostra ideal para aplicações que operam com grandes volumes de dados tensoriais densos, como esperado e descrito na literatura [Jouppi et al. 2017]. Esses resultados repetiram consistentemente alguns experimentos de comparação conhecidos [Kumar and Kasivajhula 2022].

As conclusões preliminares deste estudo reforçam que a especialização do *hardware* é uma resposta eficaz para a desaceleração da *Lei de Moore* e das melhorias de desempenho em processadores convencionais [Jouppi et al. 2020]. Os testes demonstram que, para o domínio de HPC voltado à IA e operações matriciais densas, a TPU não atua apenas como um acelerador alternativo, mas como uma evolução

importante que reconfigura os princípios de desenvolvimento de código especializado, para alcançar novos patamares de performance.

A transição de *workloads* de GPUs para TPUs é tecnicamente justificada pela superioridade das DSAs em lidar com a intensidade aritmética exigida por modelos de IA contemporâneos, sugerindo que futuros investimentos em infraestrutura de HPC podem priorizar a especialização para maximizar a eficiência por *watt* e o desempenho bruto.

Como próximos passos, pretende-se investigar a construção de um agente de IA para auxiliar na migração de códigos legados de GPU para infraestruturas de TPU, focando na refatoração automática de grafos computacionais para otimizar a portabilidade entre arquiteturas heterogêneas.

Agradecimentos

Os autores agradecem o apoio da MackCloud, Laboratório Multidisciplinar de Computação Científica e Nuvem[2]; e do projeto SPRACE – Processo nº 2018/25225-9, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP). Este trabalho foi financiado em parte pelo Fundo Mackenzie de Pesquisa e Inovação (MackPesquisa) – Projetos nº 231009 e 251005.

References

Jouppi, N. P. et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12. ACM.

Jouppi, N. P., Yoon, D. H., Kurian, G., Li, S., Patil, N., Laudon, J., Young, C., and Patterson, D. (2020). A domain-specific supercomputer for training deep neural networks. *Commun. ACM*, 63(7):67–78.

Kumar, N. and Kasivajhula, V. (2022). Cloud tpu v4 records fastest training times on five mlperf 2.0 benchmarks.

Nikolić, G. S., Dimitrijević, B. R., Nikolić, T. R., and Stojcev, M. K. (2022). A survey of three types of processing units: Cpu, gpu and tpu. In *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, pages 1–6.

Silva, G. P., Bianchini, C. P., and Costa, E. B. (2022). *Programação Paralela e Distribuída com MPI, OpenMP e OpenACC para computação de alto desempenho*. CasaDoCodigo.

Suhan, A., Libenzi, D., Zhang, A., Schuh, P., Saeta, B., Sohn, J. Y., and Shabalin, D. (2021). Lazytensor: combining eager execution with domain-specific compilers.

Wang, Y. E., Wei, G.-Y., and Brooks, D. (2019). Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*.

[2]`https://mackcloud.mackenzie.br`