Otimização de Fábricas de IA Sustentáveis por Compartilhamento de GPU

Matheus M. Costa[1], **Sandro Rigo**[2], **Carla Osthoff**[3], **Silvio Rizzi**[4], **Philippe O. A. Navaux**[1], **Arthur F. Lorenzon**[1]

[1]Instituto de Informática – UFRGS – Porto Alegre – RS – Brasil

[2]Instituto de Computação – UNICAMP – Campinas – SP – Brasil

[3]Laboratório Nacional de Computação Científica – Petrópolis – RJ – Brasil

[4]Leadership Computing Facility – Argonne National Laboratory – IL – EUA

sandro@ic.unicamp.br, osthoff@lncc.br, srizzi@anl.gov

{mmcosta,navaux,aflorenzon}@inf.ufrgs.br

Resumo. *O uso de Inteligência Artificial (IA) em larga escala tem crescido rapidamente, aumentando a pressão sobre infraestruturas de computação de alto desempenho (HPC). Em particular, com o surgimento de AI Factories, é necessária a adoção de estratégias eficientes de alocação de recursos para maximizar o throughput e reduzir o consumo energético. Nesse cenário, o compartilhamento de GPUs surge como uma alternativa promissora para melhorar a utilização dos aceleradores e equilibrar desempenho e eficiência energética. Este artigo investiga o impacto do compartilhamento da GPU na eficiência energética durante a inferência de modelos de IA usando o acelerador Intel Data Center GPU Max 1550 no supercomputador de classe exascale Aurora. Avaliamos quatro modelos distintos de IA de diferentes domínios de aplicação, como visão computacional, processamento de linguagem natural e geração de texto. Os resultados mostram que a co-localização de aplicações com perfis de uso de recursos complementares pode reduzir o tempo de execução em até 50% e o consumo de energia em até 43%. Por fim, demonstramos que a alocação de tarefas concorrentes em GPU, baseada na caracterização prévia das aplicações, é uma estratégia promissora para maximizar a eficiência em ambientes HPC.*

1. Introdução

A evolução dos modelos de Inteligência Artificial (IA) em larga escala tem promovido transformações no uso de infraestruturas de computação de alto desempenho (HPC). A popularização de modelos como BERT [Devlin et al. 2019], ResNet [He et al. 2016] e, mais recentemente, *Large Language Models* (LLMs) ampliou de forma expressiva a demanda por recursos computacionais, em especial aceleradores. Essa tendência não apenas impulsionou avanços científicos e aplicações em múltiplos domínios, mas também trouxe novos desafios para a operação de supercomputadores: o crescimento contínuo do consumo energético, as pressões de escalabilidade e a necessidade de gerenciar recursos de forma eficiente em ambientes cada vez mais complexos [Navaux et al. 2023].

Nesse cenário, as Unidades de Processamento Gráfico (GPUs) se consolidaram como a principal plataforma de execução de tarefas de IA, devido à sua alta capaci-

dade de paralelismo [Silvano et al. 2025]. Entretanto, o modelo tradicional de uso exclusivo de GPU por aplicação tende a levar à subutilização dos recursos disponíveis [Hestness et al. 2015]. Essa subutilização torna-se ainda mais crítica quando observamos o ecossistema atual de *Machine Learning Operations* (MLOps), que envolve múltiplos estágios de treinamento, inferência, validação e monitoramento contínuo, muitas vezes executados em paralelo. Além disso, aplicações multiagentes e cenários operacionais com restrições de SLA (*Service Level Agreement*) demandam garantias rígidas de tempo de resposta e eficiência energética. Nesse ambiente, maximizar o aproveitamento do hardware é fundamental não apenas para reduzir custos operacionais, mas também para assegurar a previsibilidade e confiabilidade da execução.

Como resposta a esses desafios, fabricantes e iniciativas de pesquisa têm desenvolvido soluções que possibilitam o particionamento e compartilhamento de GPUs entre múltiplas aplicações, alinhando-se ao conceito emergente de *AI Factory*, onde fluxos contínuos de produção de modelos e serviços de IA dependem de utilização eficiente dos aceleradores. Ferramentas como o NVIDIA MPS [NVIDIA 2023] e MIG [NVIDIA 2020] demonstram essa viabilidade por meio de estratégias distintas, baseadas em isolamento de contexto ou particionamento físico. Já no ecossistema Intel, a arquitetura Intel Data Center GPU Max introduz múltiplos *Compute Command Streamers* (CCSs) [Intel Corporation 2025], permitindo a execução paralela e isolada de diferentes tarefas dentro de uma mesma GPU. Essas abordagens abrem caminho para estratégias avançadas de compartilhamento de GPUs em supercomputadores como o Aurora, possibilitando um equilíbrio mais eficiente entre desempenho e consumo energético.

Dado esse contexto, este trabalho investiga a viabilidade e o impacto do compartilhamento de GPUs Intel por múltiplos modelos de IA em execução concorrente, analisando o equilíbrio entre desempenho e eficiência energética em um supercomputador de classe *exascale* como o Aurora, situado no Laboratório Nacional de Argonne (*Leadership Computing Facility - Argonne National Laboratory*). Esse cenário reflete práticas cada vez mais comuns em *pipelines* de MLOps, nos quais diferentes etapas de treinamento, ajuste fino, inferência e monitoramento são executadas de forma contínua e paralela. Da mesma forma, em arquiteturas inspiradas no conceito de *AI Factory*, diversos modelos e serviços convivem simultaneamente, exigindo estratégias de alocação que maximizem o *throughput* e mantenham a previsibilidade sob restrições de SLA. Exemplos típicos incluem desde a execução paralela de múltiplos LLMs especializados (e.g., tradução, sumarização e geração de código) até *pipelines* multiagentes, nos quais diferentes componentes de IA interagem em tempo real para compor soluções mais complexas.

Para realizar esta análise, consideramos os seguintes modelos de IA para inferência da suíte *MLPerf* [Reddi et al. 2020]: *3D U-Net*, usado em segmentação volumétrica de imagens médicas; *BERT-Base*, usado em processamento de linguagem; *Llama3.1-8B*, usado em geração de texto; e *ResNet-50*, para a tarefa de classificação de imagens. Os modelos foram executados em um nó do supercomputador Aurora, composto por dois processadores Intel Xeon Max, seis GPUs da série Intel Max e arquitetura de memória unificada, fornecendo poder computacional de até 130 teraFLOPs.

Por meio do conjunto de experimentos realizados, foi possível demonstrar que o uso do compartilhamento de GPU pode proporcionar ganhos médios de cerca de 10% no tempo de execução e 15% no consumo de energia. No entanto, esses ganhos podem che-

gar a 50% e 43%, respectivamente, a depender dos modelos executados. Para identificar essas oportunidades de co-execuções eficientes, foram gerados perfis de uso de recursos para cada aplicação, permitindo analisar seu comportamento individual e identificar combinações complementares que maximizam a utilização do *hardware*.

2. Fundamentação Teórica

Esta seção apresenta os conceitos fundamentais que sustentam este trabalho. Inicialmente, são abordadas a prática de *Machine Learning Operations* (MLOps) e sua relação com o conceito de *AI Factories*. Em seguida, são detalhadas as principais estratégias de compartilhamento de GPU. Por fim, é apresentada uma revisão dos trabalhos relacionados.

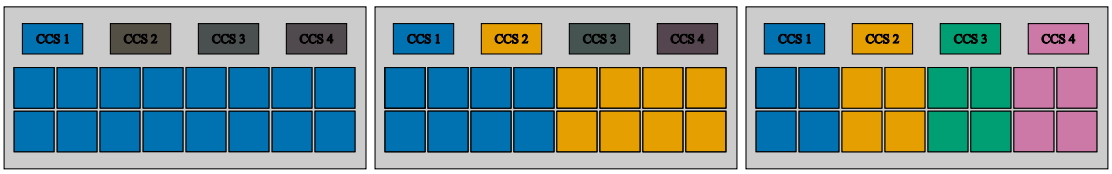
2.1. *Machine Learning Operations* (MLOps)

Machine Learning Operations (MLOps) [Kreuzberger et al. 2023] surgiu como prática essencial para operacionalizar modelos de aprendizado de máquina em escala, combinando princípios de *DevOps*, largamente conhecidos em Engenharia de Software, com as particularidades do ciclo de vida de modelos de IA. Esse paradigma sustenta a visão de *AI Factories*, ambientes computacionais onde dados, modelos e processos são continuamente integrados e atualizados para suportar aplicações dinâmicas e em larga escala. Entre essas aplicações, estão em muita evidência os sistemas multiagentes de IA, nos quais diferentes agentes especializados interagem, cada um apoiado em modelos distintos, para compor soluções complexas e adaptáveis. Esse arranjo leva naturalmente ao cenário em que há compartilhamento de recursos entre modelos diferentes em uma mesma infraestrutura.

2.2. Compartilhamento de GPUs

Existem duas abordagens principais para o compartilhamento de GPU: *compartilhamento temporal* e *compartilhamento espacial*. O compartilhamento temporal permite que vários processos utilizem a GPU ao longo do tempo, dividido em pequenos intervalos. Seu funcionamento é baseado em conceder acesso exclusivo a um processo por meio de preempção e troca de contexto. No entanto, para grandes cargas de trabalho, essa troca pode gerar uma sobrecarga, reduzindo a eficiência geral [Pratheek et al. 2021]. Por outro lado, o compartilhamento espacial divide os recursos físicos da GPU entre múltiplos processos, permitindo a execução simultânea em partições isoladas, objetivando melhorar a utilização dos recursos, permitindo paralelismo real, especialmente quando as cargas de trabalho individuais não utilizam completamente a GPU.

As GPUs Intel utilizam múltiplos CCSs [Intel Corporation 2025] para o particionamento espacial. Cada CCS atua como um mecanismo independente dentro da arquitetura de GPU, gerenciando e despachando *kernels* para um grupo de unidades computacionais. A microarquitetura Intel utiliza o núcleo X^e como unidade computacional, em que



(a) 1-CCS (b) 2-CCS (c) 4-CCS

Figura 1. Representação da distribuição entre núcleos X^e de acordo com o modo dos CCSs.

cada núcleo contém 8 unidades vetoriais e 8 matriciais, além de uma memória cache L1 compartilhada de 512 KB. Os CCSs podem ser configurados em três modos diferentes: ***(i)*** **1-CCS**: um único CCS ativo controla todos os núcleos X^e; ***(ii)*** **2-CCS**: os núcleos são distribuídos igualmente entre dois CCSs ativos; ***(iii)*** **4-CCS**: quatro CCSs dividem os núcleos igualmente. A Figura 1 ilustra os três modos de configuração possíveis. Esse compartilhamento espacial permite que as aplicações utilizem 1-CCS para *kernels* maiores ou múltiplos CCSs para a execução paralela de *kernels* menores em partições isoladas.

Outros fabricantes de GPUs também implementam mecanismos para suportar o compartilhamento de recursos. A NVIDIA, por exemplo, oferece uma combinação de estratégias de compartilhamento temporal e espacial. O compartilhamento temporal (*time-slicing*) permite que múltiplos processos compartilhem a GPU de forma serializada, enquanto o *Multi-Process Service* (MPS) [NVIDIA 2023] permite a execução concorrente de kernels através de um particionamento lógico da GPU. Além disso, o *Multi-Instance GPU* (MIG) [NVIDIA 2020] oferece particionamento espacial em nível de *hardware*, dividindo uma GPU física em múltiplas instâncias isoladas. Já a AMD utiliza motores de computação assíncronos (ACEs) e Unidades de Computação (CUs), que são atribuídos a filas de comandos independentes [Otterness and Anderson 2021].

2.3. Trabalhos Relacionados

A subutilização de recursos é um problema comum em GPUs modernas, frequentemente causada por desequilíbrios durante a execução. Gargalos como latência de memória, interrupções no *pipeline* e sobrecargas de sincronização comprometem o desempenho e deixam parte da GPU ociosa, reduzindo a eficiência geral [Sethia and Mahlke 2014, Hestness et al. 2015]. A avaliação de *benchmarks* em IA é essencial para compreender o desempenho e a eficiência de diferentes aceleradores. [Ferdaus et al. 2025] avalia GPUs e outros aceleradores tanto para treinamento quanto para inferência, destacando que as GPUs ainda mantêm uma forte competitividade dependendo da carga de trabalho realizada. O *benchmark* CARAML [John et al. 2024], por sua vez, é voltado à avaliação da eficiência energética em cenários distribuídos de IA, considerando tanto o treinamento quanto a inferência. Já o LLM-Inference-Bench [Chitty-Venkata et al. 2024] avalia um conjunto de *benchmarks* projetados para medir o desempenho de diferentes *frameworks* de inferência de LLMs em diversos aceleradores.

Diversos estudos têm investigado o compartilhamento de GPUs para melhorar a sua utilização. Ferramentas como ParvaGPU [Lee et al. 2024] e MIGER [Zhang et al. 2024] utilizam técnicas disponíveis em GPUs da NVIDIA para aumentar o desempenho de tarefas de aprendizado profundo. [Costa et al. 2025b] mostram que MPS e MIG melhoram a eficiência energética e reduzem a quantidade de nós para visualização *in-transit*, sem impactar o desempenho. [Adufu et al. 2024] demonstram que a combinação de MPS com MIG melhora tanto o desempenho como a largura de banda da memória quando comparada ao uso exclusivo do MIG. [Tramm, John et al. 2024] concluem que a estratégia de aumentar a densidade de ranks do *Message Passing Interface* (MPI) por GPU através do compartilhamento resulta em melhorias de desempenho e na simplificação da escalabilidade em arquiteturas da AMD, Intel e NVIDIA. De forma semelhante, [Costa et al. 2025a] mostram que o uso de múltiplos CCS em simulações científicas no supercomputador Aurora pode melhorar a eficiência energética dos aceleradores Intel.

Enquanto trabalhos anteriores já validaram a relevância do compartilhamento de

GPUs para mitigar a subutilização de recursos, a maioria concentra-se em arquiteturas da NVIDIA. Em contraste, ainda são escassos os estudos que exploram as capacidades específicas de compartilhamento espacial oferecidas pelos aceleradores Intel em cenários de inferência de IA. Neste contexto, nossa contribuição consiste em uma análise detalhada do desempenho e da eficiência energética, além de realizar uma caracterização detalhada dos perfis de uso dos recursos, alcançados pela co-execução de diferentes cargas de trabalho de IA, aproveitando o particionamento de recursos via múltiplos CCS.

3. Metodologia

3.1. Ambiente de Execução

Os experimentos foram realizados no supercomputador *exascale* Aurora[1], do *Argonne Leadership Computing Facility* (ALCF). Ele é baseado em HPE Cray-Ex com 10.624 nós, onde cada nó está equipado com dois processadores Intel® Xeon® CPU Max 9470C Series processors e seis aceleradores Intel® Data Center GPU Max 1550 Series. Cada GPU possui dois *compute tiles* independentes e 128 GB de memória HBM2e (64 GB por *tile*), totalizando 12 *tiles*, cada um representando um dispositivo visível para o nó. Os experimentos foram conduzidos utilizando um único *tile* de uma GPU em um nó do sistema. A escolha de utilizar apenas um *tile* em um nó foi feita para garantir um controle mais preciso sobre o desempenho e evitar qualquer interferência que possa surgir da comunicação entre múltiplos nós ou GPUs.

3.2. Conjunto de Aplicações

A seleção de aplicações utilizadas nos experimentos foi baseada nos *benchmarks* presentes no *MLPerf*[2], especificamente no conjunto de inferência. Os *benchmarks* foram escolhidos por sua relevância e suas aplicações em diferentes domínios. Para conduzir os experimentos, foi necessária uma adaptação utilizando o *framework PyTorch*, com suporte para otimizações via *Intel® Extension for PyTorch* (IPEX), possibilitando assim a execução em GPUs da Intel presentes no supercomputador Aurora.

Quatro modelos para inferência foram utilizados nos experimentos: ***3D U-Net*** (segmentação volumétrica), ***BERT-Base*** (processamento de linguagem natural), ***Llama3.1-8B*** (modelo generativo) e ***ResNet-50*** (classificação de imagens). Cada modelo é carregado com pesos pré-treinados e transferido para a GPU. A fim de isolar as partes de maior carga computacional dos modelos analisados, foi considerada exclusivamente a etapa de processamento. Esta abordagem visa capturar o cenário com a maior demanda de GPU [Kamath et al. 2025]. As entradas sintéticas foram geradas de acordo com o formato de dados esperado por cada modelo: volumes de tamanho 64^3 com 4 canais para o *3D U-Net*, sequências de comprimento fixo de 128 *tokens* para *BERT-Base* e *Llama3.1-8B*, e imagens RGB com resolução 224×224 *pixels* para o *ResNet-50*. Em todos os experimentos, o tamanho do *batch* foi variado, com valores de 2 a 64. O termo *batch size* refere-se à quantidade de amostras processadas simultaneamente em uma única etapa. Considerando que cada modelo possui diferentes métricas e entradas, e visando equilibrar o tempo de execução, foram estabelecidas as seguintes quantidades de entradas por modelo: 1.200 volumes para o *3D U-Net*, $1.000.000$ de *tokens* para o *BERT-Base*, 100.000 de *tokens* para o *Llama3.1-8B* e 15.000 imagens para o *ResNet-50*.

[1] `https://www.anl.gov/aurora`

[2] `https://mlcommons.org/benchmarks/inference-datacenter/`

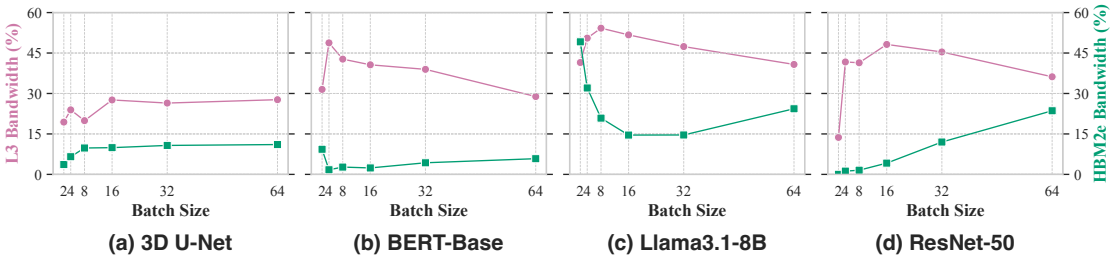


Figura 2. Largura de banda da memória L3 e HBM2e no modo de execução individual.

3.3. Modos de Execução

A metodologia é baseada no uso da variável de ambiente `ZEX_NUMBER_OF_CCS` para controlar o número de CCSs disponíveis na GPU. Foram definidos os seguintes cenários de execução: **Individual**: uma única aplicação é executada de forma exclusiva na GPU, utilizando o modo 1-CCS, em que apenas um CCS é responsável por controlar e coordenar todas as operações da GPU, garantindo o acesso total aos recursos. **Em pares**: execução simultânea de duas aplicações na mesma GPU. É empregado o modo 2-CCS, onde os recursos da GPU são particionados igualmente entre as duas aplicações.

Todas as execuções foram precedidas por uma fase de aquecimento (*warm-up*) de 10 inferências, realizada antes da coleta de métricas para estabilizar a execução e garantir a alocação dos recursos computacionais, evitando vieses de inicialização. O consumo de energia foi avaliado com o *Global Extensible Open Power Manager* (GEOPM) [Eastep et al. 2017], por meio de duas leituras do comando `geopmread GPU_ENERGY gpu 0`, antes e após a execução, sendo o valor consumido obtido pela diferença entre elas. Além disso, utilizou-se a ferramenta *unitrace*[3] para coletar métricas de perfilagem, como largura de banda da memória e estados dos X^e *Vector Engines* (XVEs).

4. Resultados

4.1. Execução Individual e Perfil das Aplicações

Para avaliar o potencial de compartilhamento de GPU, começamos caracterizando o uso individual dos recursos por aplicação. Para isso, analisamos a largura de banda da *cache* L3 e da memória *High Bandwidth Memory* (HBM2e), além dos estados de atividade dos XVEs. Os valores máximos de largura de banda para cada tipo de memória foram obtidos utilizando a ferramenta Intel® Advisor[4]. A Figura 2 ilustra a porcentagem da largura de banda em uso em função do *batch size*, revelando um comportamento comum entre os modelos, i.e., crescimento inicial até um pico, seguido de declínio, refletindo três regimes do uso de memória: subutilização em *batches* pequenos; máxima eficiência em um ponto ótimo; e saturação com consequente contenção em *batches* grandes. Portanto, o ponto de máximo varia entre as aplicações, ocorrendo em *batch* 4 para o *BERT-Base*, 8 para o *Llama3.1-8B* e 16 para o *3D U-Net* e *ResNet-50*.

[3] `https://github.com/intel/pti-gpu`

[4] `https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html`

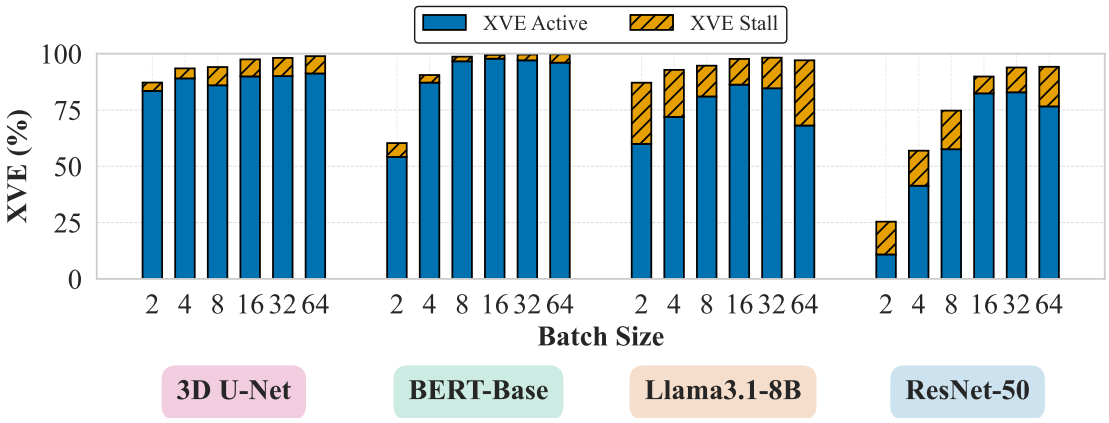


Figura 3. Distribuição do percentual de tempo dos estados dos XVEs no modo de execução individual.

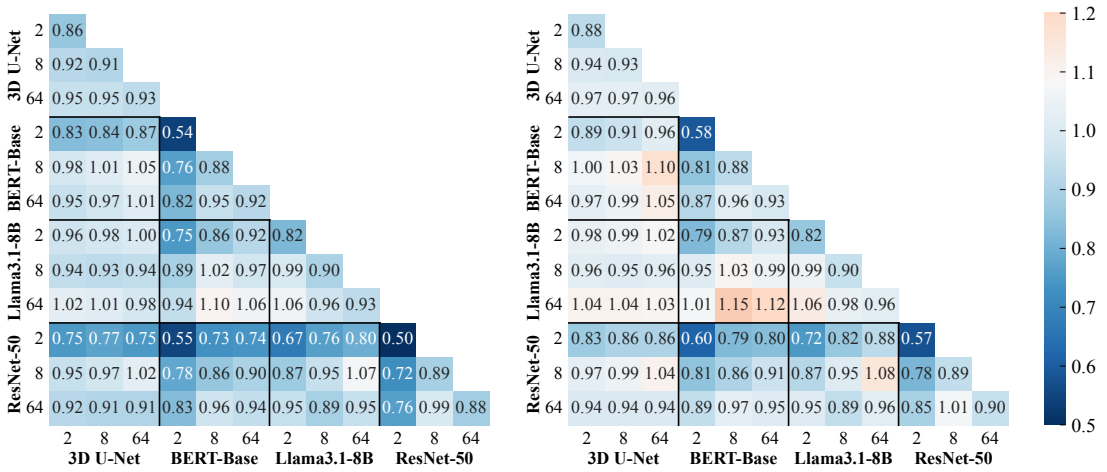
Tabela 1. Tempo total (s) e energia total (J) por modelo e batch size no modo individual. Valores em negrito indicam o melhor desempenho por modelo.

	Batch Size											
	2		4		8		16		32		64	
Modelo	*T*	*E*	*T*	*E*	*T*	*E*	*T*	*E*	*T*	*E*	*T*	*E*
3D U-Net	10.80	4396.71	**10.34**	**4227.74**	10.64	4378.07	10.64	4397.68	10.67	4450.94	10.57	4427.38
BERT-Base	28.06	10713.25	15.73	6444.92	13.54	5529.90	12.61	5155.01	12.14	4961.83	**11.45**	**4707.96**
Llama3.1-8B	18.96	8187.36	15.05	6400.55	13.64	5720.87	12.67	5325.14	**12.35**	**5196.41**	**12.36**	5242.06
ResNet-50	39.01	14224.35	19.44	7925.09	13.63	5559.16	11.49	4707.65	11.11	**4603.38**	**10.97**	4628.21

T = Tempo (s), *E* = Energia (J)

A análise dos estados de execução dos XVEs, apresentada na Figura 3, evidencia diferenças no perfil de utilização de cada modelo. Os estados *Active* e *Stall* representam situações de execução distintas: *execução efetiva de instruções* e *espera por recursos* (e.g., memória). Há ainda o estado *Idle*, associado à ociosidade, o qual corresponde ao complemento dos anteriores e, portanto, foi omitido da figura. O *3D U-Net* apresenta um padrão de computação intensiva, com predominância do estado *Active*, e um baixo uso de memória consistente observado na Figura 2a. Embora o *BERT-Base* tenha um perfil similar, ele se diferencia por uma maior largura de banda do cache L3, particularmente com *batch sizes* menores. O modelo também apresentou uma baixa utilização dos XVEs no menor valor para o *batch size*. Já o *Llama3.1-8B* mostra maior dependência de memória: picos de utilização da HBM (Figura 2c) aumentam o tempo em *Stall*, agravado pela redução da largura de banda da L3. O *ResNet-50*, por sua vez, revela forte sensibilidade ao *batch size*; para valores pequenos (2 e 4), há subutilização dos XVEs, resultando em menos de 50% do tempo em *Active* e em baixa largura de banda de memória (Figura 2d).

Esse comportamento se reflete diretamente nos resultados apresentados na Tabela 1. Em execução individual (1-CCS), configurações que mantêm os XVEs por mais tempo em *Active* apresentam menor tempo de execução e consumo de energia. Para o modelo *3D U-Net*, o desempenho ótimo é alcançado com *batch size* 4. Nesta configuração, observa-se uma maior proporção de XVEs ativos e um leve pico na utilização da *cache* L3. O *BERT-Base* apresenta crescimento monotônico de desempenho com o aumento do *batch size*, enquanto o *Llama3.1-8B* atinge seu ponto ótimo em *batch* 32, a partir do qual



(a) Tempo de execução (b) Consumo de energia

Figura 4. Razão entre execução em pares e execução individual para diferentes modelos. Valores menores que 1 indicam ganhos de eficiência quando duas instâncias são executadas simultaneamente.

a pressão sobre a HBM e o aumento de *Stall* reduzem o desempenho energético. Já o *ResNet-50* registra os maiores ganhos relativos, passando de 39,01 para 10,97 segundos com o aumento do *batch*, reflexo da maior ocupação dos XVEs e do aproveitamento mais eficiente da largura de banda disponível.

4.2. Avaliação da Execução em Pares

A Figura 4 apresenta dois gráficos em formato de *heatmap* que ilustram o desempenho e consumo energético normalizados para todas as combinações possíveis de modelos e valores de *batch size* no modo de execução em pares, cenário em que duas aplicações são executadas simultaneamente na mesma GPU.

Os valores exibidos correspondem à razão entre os resultados obtidos na execução em pares e os resultados das execuções individuais correspondentes a cada uma das duas aplicações. Valores inferiores a 1,0 indicam ganhos com o compartilhamento de recursos, enquanto valores superiores apontam para perdas. Cada célula do gráfico corresponde à execução simultânea de um par de modelos, sendo que o modelo no eixo Y é executado em conjunto com o modelo indicado no eixo X. Os *heatmaps* mostram que os maiores ganhos, tanto em tempo de execução quanto em consumo de energia, ocorrem na combinação dos modelos *ResNet-50* e *BERT-Base*, ambos configurados com *batch size* igual a 2. O maior benefício, no entanto, é observado na execução concorrente de duas instâncias do *ResNet-50* com *batch size* 2, em que o tempo de execução e o consumo energético correspondem a apenas 50% e 57%, respectivamente, dos valores das execuções individuais.

Os resultados reforçam as métricas da Figura 3, que mostram menor uso dos XVEs nos modelos com maiores ganhos de desempenho e eficiência energética na execução em pares. Em contrapartida, também revelam combinações com contenção de recursos e, consequentemente, perdas de desempenho. Os piores casos de desempenho ocorrem nas execuções com *Llama3.1-8B* e *BERT-Base*, com *batch sizes* 64 e 8, respectivamente, devido à disputa pelo uso dos XVEs. Especificamente, quando os dois modelos são executados em pares, observa-se um aumento no tempo de execução em aproximadamente

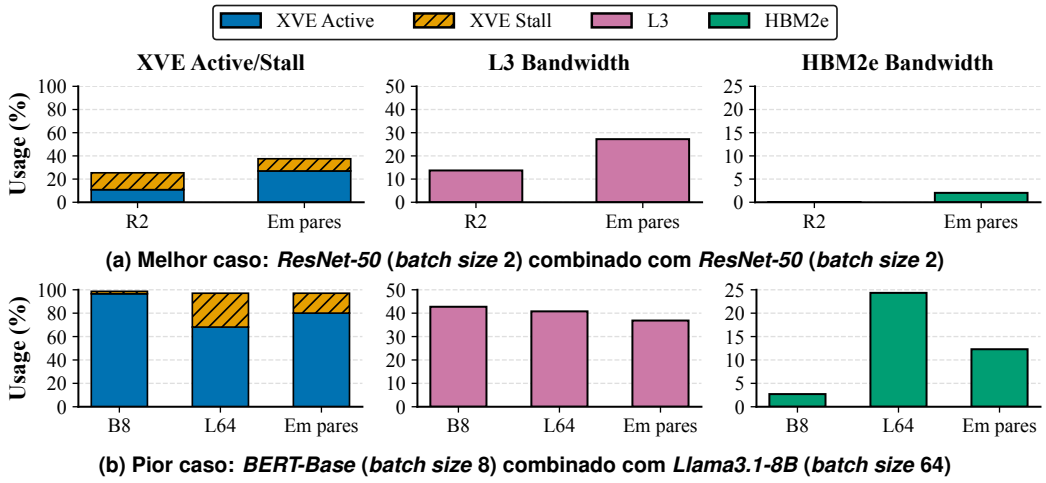


Figura 5. Análise dos casos de melhor e pior desempenho no modo de execução em pares.

10%, acompanhado de um aumento de até 15% no consumo energético.

Para analisar os casos de melhor e pior desempenho na execução em pares, a Figura 5 ilustra o impacto na utilização dos XVEs, bem como na largura de banda das memórias. No melhor caso (Figura 5a), que combina duas instâncias do *ResNet-50* com *batch size* 2, a execução individual (indicada como R2) apresenta baixa utilização dos XVEs, com o tempo em estado ativo em 10%. Ao executar as duas instâncias em pares, o tempo dos XVEs ativos alcança 27%, enquanto a largura de banda da *cache* L3 aumenta cerca de 13% para aproximadamente 27%. A demanda sobre a memória HBM2e permanece mínima, atingindo apenas 2%.

Ao observar o pior caso (Figura 5b), onde o *BERT-Base* (*batch size* 8) e o *Llama3.1-8B* (*batch size* 64) executam em pares, observa-se uma contenção de recursos. Individualmente, ambos os modelos exibem alta utilização dos XVEs e da *cache* L3. O *BERT-Base* (B8) praticamente satura os XVEs, 96% em *Active* e 2% em *Stall*, com 43% da largura de banda da L3 e somente 3% da HBM2e. Em contraste, o *Llama3.1-8B* (L64) apresenta menor atividade dos XVEs (68%), alta taxa de *Stall* (29%), e maior demanda por largura de banda tanto da L3 (41%) quanto da HBM2e (24%).

Na execução em pares, a atividade dos XVEs cai para 80%, abaixo do valor atingido isoladamente pelo *BERT-Base*, evidenciando limitação de desempenho. Além disso, a largura de banda da L3 se reduz para menos de 37%, inferior ao observado em ambas as execuções individuais. Para a HBM2e, a execução concorrente soma apenas 12%, bem abaixo dos 24% demandados pelo *Llama3.1-8B* em execução individual.

4.3. Discussão

A partir dos resultados apresentados na Figura 4, permite identificar cenários práticos onde a co-execução de modelos distintos oferece vantagens em termos de eficiência computacional e energética. Esta análise é fundamental para ambientes de HPC e *AI Factory*, onde a otimização de recursos computacionais e energéticos determina a viabilidade econômica e operacional de infraestruturas de larga escala.

Na área médica, a co-execução do *3D U-Net*, para segmentação de imagens volu-

métricas de tumores, e *BERT-Base*, empregado na interpretação de textos clínicos, representa um cenário em que o compartilhamento de GPU pode ser explorado para maximizar o uso do *hardware*. Nesse contexto, é possível alcançar ganhos de até 17% no tempo de execução e 11% no consumo de energia. Em um fluxo de trabalho clínico, essa otimização pode significar diagnósticos mais rápidos, maior capacidade de processamento de exames e a viabilização de análises integradas de dados de imagem e texto.

A co-execução do *ResNet-50*, para classificação de imagens, com o *Llama3.1-8B*, para geração de linguagem natural, apresenta um potencial de otimização maior. Essa combinação é útil em aplicações como o monitoramento urbano, onde imagens são classificadas e descritas automaticamente para auxiliar na tomada de decisões. Neste cenário, a redução de até 33% no tempo de execução e 28% no consumo energético demonstra que a assimetria nos perfis de uso de recursos entre um modelo de visão computacional e um modelo de geração textual pode ser explorada para maximizar a eficiência do sistema.

5. Conclusão

Este artigo investigou o impacto do compartilhamento espacial de GPU no tempo de execução e consumo de energia para cargas de trabalho de inferência de IA, utilizando a arquitetura Intel® Data Center GPU Max 1550 no supercomputador de classe *exascale* Aurora. Por meio de uma análise da co-execução de quatro modelos de IA distintos (*3D U-Net*, *BERT-Base*, *Llama3.1-8B* e *ResNet-50*), demonstramos que a alocação de múltiplos modelos em uma única GPU é uma estratégia viável e promissora para otimizar o uso de recursos em ambientes de *AI Factory* e HPC.

Os resultados revelam que a eficácia do compartilhamento de GPU depende da complementaridade dos perfis de uso de recursos das aplicações co-localizadas, alcançando reduções de até 50% no tempo de execução e 43% no consumo energético. Particularmente, a análise dos estados dos XVEs e do uso de largura de banda de memória revelou-se fundamental para identificar combinações eficientes. Modelos que apresentam baixa utilização dos XVEs, como o *ResNet-50* com *batch size* pequeno, demonstraram maior potencial de ganhos quando executados em pares.

Os ganhos observados têm implicações diretas para a sustentabilidade e viabilidade econômica de infraestruturas de IA em larga escala. A redução no consumo energético, combinada com melhorias no tempo de execução, representa uma contribuição relevante para a operação eficiente de *AI Factories*, onde múltiplos modelos e serviços coexistem. Como trabalhos futuros, investigaremos algoritmos de escalonamento dinâmico que utilizem as métricas de caracterização propostas para otimização em tempo real, além da extensão da análise para cenários com múltiplos nós e GPUs.

Agradecimentos

Este trabalho foi parcialmente apoiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior- Brasil (CAPES)- Código de Financiamento 001, FAPERGS - PqG 24/2551-0001388-1, e CNPq. *This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.*

Referências

Adufu, T., Ha, J., and Kim, Y. (2024). Exploring the Diversity of Multiple Job Deployments over GPUs for Efficient Resource Sharing. In *ICOIN*, pages 777–782. IEEE.

Chitty-Venkata, K. T., Raskar, S., Kale, B., Ferdaus, F., Tanikanti, A., Raffenetti, K., Taylor, V., Emani, M., and Vishwanath, V. (2024). LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators. In *SC24-W*, pages 1362–1379.

Costa, M. M., Navaux, P. O. A., Rizzi, S., and Lorenzon, A. F. (2025a). Optimizing QMCPACK Energy-Efficiency on Aurora via GPU Sharing. In *12th Latin American High Performance Computing Conference, CARLA 2025*.

Costa, M. M., Rizzi, S., Navaux, P. O. A., and Lorenzon, A. F. (2025b). One GPU, Many Ranks: Enabling Performance and Energy-Efficient In-Transit Visualization via Resource Sharing. In *Proceedings of the 54th International Conference on Parallel Processing*, New York, NY, USA. Association for Computing Machinery.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Eastep, J., Sylvester, S., Cantalupo, C., Geltz, B., Ardanaz, F., Al-Rawi, A., Livingston, K., Keceli, F., Maiterth, M., and Jana, S. (2017). Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions. In Kunkel, J. M., Yokota, R., Balaji, P., and Keyes, D., editors, *High Performance Computing*, pages 394–412, Cham. Springer International Publishing.

Ferdaus, F., Wu, X., Taylor, V., Lan, Z., Shanmugavelu, S., Vishwanath, V., and Papka, M. E. (2025). Evaluating Energy Efficiency of Ai Accelerators Using Two Mlperf Benchmarks. In *2025 IEEE 25th CCGrid*, pages 549–558.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778.

Hestness, J., Keckler, S. W., and Wood, D. A. (2015). GPU Computing Pipeline Inefficiencies and Optimization Opportunities in Heterogeneous CPU-GPU Processors. In *IISWC*, pages 87–97. Institute of Electrical and Electronics Engineers Inc.

Intel Corporation (2025). Advanced Topics — Intel® oneAPI GPU Optimization Guide. `https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2025-0/advanced-topics.html`.

John, C. M., Nassyr, S., Penke, C., and Herten, A. (2024). Performance and Power: Systematic Evaluation of AI Workloads on Accelerators with CARAML. In *SC24-W*, pages 1164–1176.

Kamath, A. K., Prabhu, R., Mohan, J., Peter, S., Ramjee, R., and Panwar, A. (2025). POD-Attention: Unlocking Full Prefill-Decode Overlap for Faster LLM Inference. In *ASPLOS*, ASPLOS '25, page 897–912, New York, NY, USA. ACM.

Kreuzberger, D., Kühl, N., and Hirschl, S. (2023). Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *IEEE Access*, 11:31866–31879.

Lee, M., Seong, S., Kang, M., Lee, J., Na, G.-J., Chun, I.-G., Nikolopoulos, D., and Hong, C.-H. (2024). ParvaGPU: Efficient Spatial GPU Sharing for Large-Scale DNN Inference in Cloud Environments . In *SC24*, pages 1–14, Los Alamitos, CA, USA. IEEE Computer Society.

Navaux, P. O. A., Lorenzon, A. F., and Serpa, M. D. S. (2023). Challenges in High-Performance Computing. *Journal of the Brazilian Computer Society*, 29(1):51–62.

NVIDIA (2020). NVIDIA A100 Tensor Core GPU Architecture. `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf`.

NVIDIA (2023). NVIDIA Multi-Process Service Overview. `https://docs.nvidia.com/deploy/mps/index.html`.

Otterness, N. and Anderson, J. H. (2021). Exploring AMD GPU Scheduling Details by Experimenting With "Worst Practices". In *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, RTNS '21, page 24–34, New York, NY, USA. ACM.

Pratheek, B., Jawalkar, N., and Basu, A. (2021). Improving GPU Multi-tenancy with Page Walk Stealing. In *HPCA*, pages 626–639.

Reddi, V. J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C.-J., Anderson, B., Breughe, M., Charlebois, M., Chou, W., Chukka, R., Coleman, C., Davis, S., Deng, P., Diamos, G., Duke, J., Fick, D., Gardner, J. S., Hubara, I., Idgunji, S., Jablin, T. B., Jiao, J., John, T. S., Kanwar, P., Lee, D., Liao, J., Lokhmotov, A., Massa, F., Meng, P., Micikevicius, P., Osborne, C., Pekhimenko, G., Rajan, A. T. R., Sequeira, D., Sirasao, A., Sun, F., Tang, H., Thomson, M., Wei, F., Wu, E., Xu, L., Yamada, K., Yu, B., Yuan, G., Zhong, A., Zhang, P., and Zhou, Y. (2020). MLPerf Inference Benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459.

Sethia, A. and Mahlke, S. (2014). Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution. In *IEEE/ACM Int. Symposium on Microarchitecture*, pages 647–658.

Silvano, C., Ielmini, D., Ferrandi, F., Fiorin, L., Curzel, S., Benini, L., Conti, F., Garofalo, A., Zambelli, C., Calore, E., Schifano, S., Palesi, M., Ascia, G., Patti, D., Petra, N., De Caro, D., Lavagno, L., Urso, T., Cardellini, V., Cardarilli, G. C., Birke, R., and Perri, S. (2025). A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms. *ACM Comput. Surv.*, 57(11).

Tramm, John, Romano, Paul, Shriwise, Patrick, Lund, Amanda, Doerfert, Johannes, Steinbrecher, Patrick, Siegel, Andrew, and Ridley, Gavin (2024). Performance Portable Monte Carlo Particle Transport on Intel, NVIDIA, and AMD GPUs. *EPJ Web Conf.*, 302:04010.

Zhang, B., Li, S., and Li, Z. (2024). MIGER: Integrating Multi-Instance GPU and Multi-Process Service for Deep Learning Clusters. In *Proceedings of the 53rd International Conference on Parallel Processing*, page 504–513, New York, NY, USA. ACM.