Extensão da GSParLib para Suporte a GPUs NVIDIA, AMD e Intel

Gabriell Araujo[1], **Gabriel Rustick Fim**[1], **Dalvan Griebler**[1], **Luiz G. Fernandes**[1]

[1] Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil
{gabriell.araujo, gabriel.fim}@edu.pucrs.br
{dalvan.griebler, luiz.fernandes}@pucrs.br

Resumo. *A programação portável para GPUs de diferentes fabricantes ainda é um desafio em computação de alto desempenho, devido à predominância de código legado em CUDA e às limitações práticas do OpenCL. Este trabalho apresenta uma extensão da GSParLib, originalmente projetada para GPUs NVIDIA, para suportar também GPUs AMD e Intel. Resultados preliminares com benchmarks da suíte NAS Parallel Benchmarks mostram que a GSParLib alcança desempenho similar ou superior ao OpenCL na maioria dos casos, além de maior robustez. Esses resultados indicam que a GSParLib é uma alternativa promissora para programação portável e eficiente em ambientes heterogêneos.*

1. Contexto

Unidades de Processamento Gráfico (GPUs) tornaram-se aceleradores fundamentais para viabilizar aplicações modernas de alto desempenho. Com milhares de núcleos de processamento e elevada capacidade de paralelismo, essas arquiteturas têm sido amplamente empregadas em domínios como aprendizado profundo, veículos autônomos, biologia computacional, previsão do tempo, ciência de dados, imagens médicas e dinâmica de fluidos. Embora a NVIDIA tenha se consolidado como a principal referência nesse ecossistema, outros fabricantes, como AMD e Intel, passaram a oferecer alternativas de alto poder computacional, inclusive em sistemas de grande escala. Nesse cenário, surge um desafio importante para programadores e pesquisadores: desenvolver aplicações portáveis entre GPUs de diferentes fabricantes.

Esse desafio é agravado pelo fato de que grande parte do código legado para GPUs foi desenvolvida em CUDA, tecnologia proprietária da NVIDIA e incompatível com dispositivos de outros fabricantes. Como consequência, portar aplicações para GPUs AMD e Intel frequentemente exige reescrita de código ou adaptações significativas. O OpenCL é a principal referência em portabilidade de código para aceleradores, mas apresenta limitações práticas relevantes. Trata-se de uma API verbosa, de baixo nível, que expõe explicitamente detalhes de gerenciamento de dispositivo, memória e execução, aumentando a complexidade de desenvolvimento. Além disso, fabricantes tendem a priorizar suas próprias APIs, oferecendo ao OpenCL suporte mais limitado, o que pode resultar em instabilidade, diferenças entre versões suportadas e perda de desempenho quando comparado ao uso das APIs nativas de cada fabricante.

Diante desse contexto, este projeto investiga uma metodologia para construção de APIs voltadas à computação de alto desempenho que facilitem a programação portável de GPUs sem comprometer o desempenho. Como base para essa investigação, utiliza-se

a GSParLib [Rockenbach et al. 2025], uma API de programação para GPUs desenvolvida em C++ com orientação a objetos. Sua proposta consiste em abstrair mecanismos de baixo nível por meio de classes que representam elementos centrais da programação em GPU, como dispositivo, memória e *kernel*. A biblioteca também fornece abstrações para diferenças sintáticas e funcionais entre *backends*, incluindo macros expandidas para diferentes linguagens de *kernel* e mecanismos auxiliares, como operações atômicas.

Entretanto, apesar de sua proposta de abstração, a implementação original da GSParLib foi concebida essencialmente para GPUs NVIDIA. Mesmo seu suporte a OpenCL foi desenvolvido com dependências específicas dessa fabricante, utilizando atributos e recursos incompatíveis com implementações de AMD e Intel. Assim, embora a GSParLib represente uma base promissora para programação portável em GPUs, sua versão original ainda não atendia adequadamente ao cenário *multi-vendor*, o que motiva as extensões propostas neste trabalho.

2. Metodologia

O primeiro passo para estender a GSParLib ao suporte de múltiplos fabricantes consistiu em revisar sua implementação baseada em OpenCL, de modo a remover dependências específicas da NVIDIA e adaptar mecanismos internos conforme o fabricante da GPU. Entretanto, como o OpenCL apresenta limitações práticas em termos de estabilidade, desempenho e uniformidade de suporte entre fabricantes, a metodologia proposta neste trabalho prioriza o uso das APIs de referência de cada fabricante por meio de uma interface unificada na GSParLib. Assim, além do suporte a CUDA e OpenCL, a biblioteca passou a incorporar uma camada baseada em HIP, permitindo acesso às infraestruturas nativas de execução de GPUs AMD, NVIDIA e Intel.

Para isso, foram introduzidas implementações específicas do *backend* HIP para cada fabricante. No caso das GPUs AMD, a GSParLib utiliza HIP como API de referência. Para GPUs NVIDIA, o HIP pode atuar como camada compatível sobre CUDA. Já para GPUs Intel, a biblioteca acessa os mecanismos de execução por meio de HIP sobre a infraestrutura Level Zero, que é a API nativa dessa fabricante. Embora um *backend* nativo baseado diretamente em Level Zero ainda não tenha sido implementado, essa abordagem já permite integrar GPUs Intel à biblioteca dentro de uma estratégia comum. Dessa forma, a nova versão da GSParLib passa a suportar CUDA, HIP e OpenCL, ampliando a portabilidade entre fabricantes e aumentando o potencial de desempenho e estabilidade quando comparado ao uso exclusivo de OpenCL.

Além da ampliação dos *backends*, a metodologia também trata de uma limitação importante do HIP: a necessidade de compilação direcionada a fabricantes ou arquiteturas específicas. Em ambientes heterogêneos, como nodos ou *clusters* com GPUs de diferentes fabricantes, isso normalmente exige que o programador mantenha manualmente múltiplos binários e caminhos de execução. Para contornar esse problema, a GSParLib foi reestruturada com uma arquitetura baseada em *plugins* e em uma ABI (*Application Binary Interface*) comum. Em vez de vincular a biblioteca diretamente a um *backend* específico em tempo de compilação, define-se uma interface binária estável que padroniza operações como descoberta de dispositivos, alocação de memória, criação de *kernels*, configuração de parâmetros e lançamento da execução (*kernels*).

Cada *backend*, como CUDA, HIP para AMD, HIP para NVIDIA, HIP para Intel

e OpenCL, é compilado separadamente como uma biblioteca compartilhada (`.so`) que implementa essa ABI. Em tempo de execução, a GSParLib carrega dinamicamente os *plugins*, identifica os *backends* e dispositivos disponíveis e pode utilizar mais de um deles simultaneamente, mesmo que sejam GPUs de fabricantes distintos. Com isso, um mesmo programa pode distribuir sua execução entre GPUs NVIDIA, AMD e Intel sem modificar sua lógica principal. Cabe ao usuário apenas selecionar o identificador do dispositivo e o *backend* desejado (CUDA, HIP, ou OpenCL), enquanto o *runtime* da GSParLib escolhe o *plugin* apropriado e encaminha as operações por meio da interface unificada.

3. Resultados preliminares

Como avaliação preliminar, utilizaram-se a GSParLib e OpenCL para paralelizar os *benchmarks* CG, EP, FT, IS e MG da suíte NAS *Parallel Benchmarks* (NPB), tendo a versão C++ do NPB como base para o código paralelo [Löff et al. 2021]. As estratégias de paralelismo com GPU seguiram as estratégias da versão para GPUs do NPB [Araujo et al. 2025]. Cada versão foi executada dez vezes com o *workload* da classe C, e o *speedup* foi calculado em relação à execução sequencial na CPU. A Figura 1 apresenta os resultados, incluindo barras de erro correspondentes ao desvio padrão. Os experimentos foram realizados em um sistema com processador Ryzen 7 7700 (8 núcleos e 16 *threads*), GPU NVIDIA RTX 5060 Ti (4608 núcleos e 16 GB de VRAM), GPU AMD RX 7800 XT (3840 núcleos e 16 GB de VRAM) e GPU Intel B580 (2560 núcleos e 12 GB de VRAM).

De forma geral, a GSParLib apresentou desempenho similar ou superior ao OpenCL na maior parte dos casos. Na GPU NVIDIA, a GSParLib obteve resultados iguais ou superiores ao OpenCL em todos os *benchmarks*, com destaque para o FT, no qual alcançou aproximadamente o dobro do *speedup*. Na GPU AMD, a GSParLib também manteve desempenho competitivo em todos os casos, superando o OpenCL de forma mais expressiva em FT (cerca de 28%) e IS (cerca de 66%). Além disso, a versão OpenCL do MG não conseguiu executar o *workload* da classe C nessa GPU, devido a limitações da implementação OpenCL da AMD, relacionadas à disponibilidade de recursos do dispositivo e ao suporte restrito ao OpenCL 2.0.

Na GPU Intel, a GSParLib também apresentou resultados competitivos, embora com perdas de desempenho em alguns casos específicos. A principal diferença ocorreu no *benchmark* CG, no qual o OpenCL obteve aproximadamente o dobro do *speedup* em relação à GSParLib. Perdas menores também foram observadas em FT e IS. Nesses casos, a GSParLib utilizou o acesso à infraestrutura Level Zero por meio do HIP, cuja implementação ainda é experimental, incluindo o suporte à compilação em tempo de execução, o que ajuda a explicar a redução de desempenho observada.

Em síntese, os resultados preliminares indicam que a GSParLib possui potencial para oferecer desempenho comparável ou superior ao OpenCL, além de maior robustez em cenários nos quais o OpenCL apresenta limitações de suporte ou execução. Os ganhos observados nas GPUs NVIDIA, AMD e Intel, bem como a incapacidade do OpenCL de executar o MG na AMD, reforçam o potencial da GSParLib como alternativa para programação portável e eficiente em ambientes com GPUs de múltiplos fabricantes.

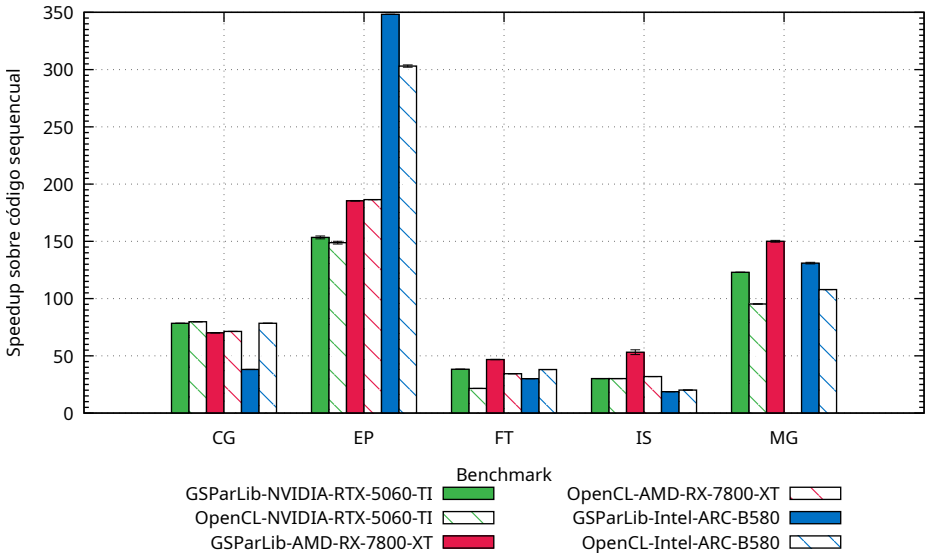

Figura 1. GSParLib vs OpenCL em GPUs NVIDIA, AMD, e Intel.

4. Conclusão

Este trabalho apresentou a extensão da GSParLib para suporte a GPUs NVIDIA, AMD e Intel, com foco em portabilidade de código, aproveitamento das APIs de referência de cada fabricante e suporte a ambientes heterogêneos com múltiplos dispositivos. A metodologia proposta envolveu tanto a adaptação do *backend* OpenCL para remover dependências específicas da NVIDIA quanto a incorporação de *backends* baseados em HIP, além da introdução de uma arquitetura baseada em *plugins* e de uma ABI *Application Binary Interface* comum para seleção dinâmica de *backends* em tempo de execução. Os resultados preliminares com os *benchmarks* CG, EP, FT, IS e MG da suíte NAS *Parallel Benchmarks* indicam que a GSParLib apresenta desempenho competitivo e, em diversos casos, superior ao OpenCL, além de maior robustez em cenários onde a implementação em OpenCL mostrou limitações de desempenho ou compatibilidade. Como trabalhos futuros, pretende-se amadurecer o suporte à Intel, incluindo um *backend* nativo para Level Zero, e expandir a avaliação experimental para cenários com múltiplas GPUs e aplicações mais amplas de computação de alto desempenho.

Referências

Araujo, G., Griebler, D., and Fernandes, L. G. (2025). Performance, Portability, and Productivity of HIP on GPUs with NAS Parallel Benchmarks. In *2025 IEEE/SBC 37th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 204–214.

Löff, J., Griebler, D., Mencagli, G., Araujo, G., Torquati, M., Danelutto, M., and Fernandes, L. G. (2021). The NAS parallel benchmarks for evaluating C++ parallel programming frameworks on shared-memory architectures. *Future Generation Computer Systems*, 125:743–757.

Rockenbach, D. A., Araujo, G., Griebler, D., and Fernandes, L. G. (2025). GSParLib: A multi-level programming interface unifying OpenCL and CUDA for expressing stream and data parallelism. *Computer Standards & Interfaces*, 92:103922.