Provendo melhorias na GSParLib

Gabriell Araujo, Dalvan Griebler, Luiz G. Fernandes

[1] Escola Politécnica, Grupo de Modelagem de Aplicações Paralelas (GMAP),
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Brasil

gabriell.araujo@edu.pucrs.br,{dalvan.griebler,luiz.fernandes}@pucrs.br

Resumo. *Neste trabalho são apresentados resultados parciais do estudo que está sendo conduzido para prover melhorias de programabilidade e desempenho no framework de programação para GPUs GSParLib.*

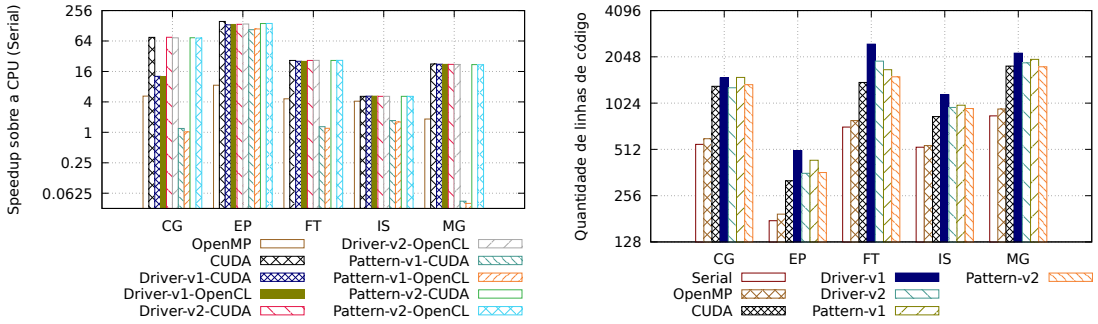
1. Contexto

Unidades de Processamento Gráfico (GPUs) estão presentes em quase todos os computadores modernos, porém, programá-las é um desafio para programadores, pois requer o uso de programação paralela. Neste sentido, a comunidade científica está trabalhando em pesquisas como o desenvolvimento de abstrações para programação paralela. Dado esse contexto, GSParLib [Rockenbach 2020] é um *framework* de programação paralela para C++ que oferece abstrações para programação de GPUs. A GSParLib visa fornecer uma interface de programação unificada para CUDA e OpenCL, e é divida em duas APIs: 1) Driver API, uma camada de *software* acima de CUDA e OpenCL; 2) Pattern API, um conjunto de esqueletos algorítmicos acima da Driver API; Contudo, a GSParLib é uma solução recente e ainda não foi avaliada de forma abrangente. Neste documento são apresentados os resultados parciais do estudo de avaliação, e implementação de melhorias.

2. Limitações e melhorias

Para avaliar a GSParLib foram paralelizados cinco *benchmarks* do NPB [Bailey et al. 1994] seguindo as estratégias do trabalho prévio com CUDA [Araujo et al. 2021]. Os experimentos foram conduzidos em uma máquina equipada com um processador Intel Xeon E5-2620 (6 núcleos e 12 *threads*), e uma GPU NVIDIA Titan X Pascal (3584 núcleos) com 12 Gigabytes de memória. Cada teste foi executado dez vezes e o desvio padrão permaneceu abaixo de 1%. Figura 1 apresenta os resultados deste trabalho. Figura 1(a) apresenta o *speedup* das versões paralelas em relação ao código sequencial na CPU. Figura 1(b) apresenta a quantidade de linhas de código das versões. No eixo x são listados os *benchmarks*. No eixo y são apresentadas as métricas. As versões são nomeadas da seguinte forma. Versão Sequencial como `Serial`. Versão OpenMP como `OpenMP`. Versão CUDA como `CUDA`. Versões com a GSParLib original como `Driver-v1-CUDA`, `Driver-v1-OpenCL`, `Pattern-v1-CUDA`, e `Pattern-v1-OpenCL`. Versões com a GSParLib modificada como `Driver-v2-CUDA`, `Driver-v2-OpenCL`, `Pattern-v2-CUDA`, e `Pattern-v2-OpenCL`.

Foram detectadas várias limitações de programabilidade na GSParLib. A principal limitação é a necessidade do programador prover rotinas escritas em CUDA e OpenCL para definir *kernels* de GPU, enquanto uma interface unificada deveria ser provida. Outras limitações são os recursos da GPU que não são reconhecidos, ou não conseguem ser habilitados ou utilizados. Quanto ao desempenho, foram observadas limitações principalmente na Pattern API, incluindo excesso de transferências de memória, e recompilação de



(a) *Speedup* com a Classe C.

(b) Quantidade de linhas de código.

Figura 1. Comparação de desempenho e linhas de código.

kernels de GPU. Para resolver as limitações da GSParLib, foi desenvolvido um conjunto de abstrações. Foram criadas palavras-chave e funções que podem ser utilizadas pelo usuário. Durante a compilação do *kernel* de GPU, através das abstrações com interface unificada, é gerado código CUDA e OpenCL. Quanto aos recursos da GPU e limitação de desempenho, os mecanismos internos foram modificados seguindo boas práticas de programação para GPUs. Figura 1(a) demonstra que a `Driver-v1` possui limitações de desempenho apenas nos *benchmarks* CG e EP. Em contrapartida, `Pattern-v1` teve perdas de desempenho relevantes em todos os *benchmarks*, chegando a ser mais lento que o código sequencial em alguns casos. Porém, ao utilizar a nova versão da GSParLib, tanto a `Driver-v2` quanto a `Pattern-v2` atingiram desempenho similar a CUDA em todos os *benchmarks*. Na Figura 1(b) verificou-se que a nova versão da GSParLib (`Driver-v2` e `Pattern-v2`) requer uma quantidade de linhas de código similar a CUDA. A versão anterior exigia códigos maiores pelo motivo de que era necessário escrever uma versão dos *kernels* de GPU para CUDA e outra para OpenCL. No entanto, a principal vantagem da interface unificada é facilitar a programação de GPUs para não especialistas em CUDA e OpenCL, bem como permitir código portável entre GPUs de diferentes fabricantes.

3. Conclusões

Os resultados parciais deste trabalho demonstram melhorias na programabilidade e desempenho do *framework* GSParLib. Os próximos passos do trabalho visam prover novos mecanismos e esqueletos algorítmicos para o *framework*.

Referências

Araujo, G., Griebler, D., Rockenbach, D. A., Danelutto, M., and Fernandes, L. G. (2021). NAS Parallel Benchmarks with CUDA and beyond. *Software: Practice and Experience*.

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K. (1994). The NAS Parallel Benchmarks RNR-94-007. Technical report, NASA Advanced Supercomputing Division, California, U.S.

Rockenbach, D. A. (2020). High-Level Programming Abstractions for Stream Parallelism on GPUs. Master's thesis, School of Technology - PPGCC - PUCRS, Porto Alegre, Brazil.