Projeto de Pesquisa para Iniciação Científica

Comparando a utilização de soluções livres e proprietárias para programação geral de GPUs

Isabella Basso do Amaral

Orientador: Alfredo Goldman Vel Lejbman
Instituto de Matemática e Estatísica
Universidade de São Paulo
São Paulo, Brasil
gold@ime.usp

Resumo

A adoção, por parte da comunidade científica, de *software* com intuito de aproximar soluções para problemas relevantes no cenário contemporâneo traz consigo desafios além dos técnicos, principalmente, no que diz respeito à utilização de soluções livres para a execução de tais pesquisas. Esta é essencial para que não violemos princípios básicos da pesquisa científica como a entendemos nos dias de hoje, como por exemplo o princípio da reprodutibilidade. No presente projeto, então, verificamos a distribuição de pesquisas em relação à sua utilização de soluções livres ou proprietárias, especificamente no contexto de *GPU General Programming* (GPGPU), assim como as disparidades entre opções livres e proprietárias que complexificam o cenário ideal. Munidos desses dados, buscaremos uma compreensão mais aprofundada da situação, tomando em consideração aspectos técnicos de cada alternativa. A partir deste ponto, exploramos os passos necessários para facilitar a adoção de soluções livres, assim como o seu desenvolvimento.

Palavras-chave

APIs gráficas, GPU, CUDA, Vulkan, Compute, OpenCL, FLOSS, código proprietário

I. Introdução

O computador tornou-se parte indispensável da pesquisa acadêmica principalmente "resolvendo" (aproximando) problemas numéricos, os quais são, em grande maioria, insolúveis sob o olhar analítico.

A aproximação de soluções para problemas numéricos em *software*, embora não seja uma questão nova ou sequer teórica, demanda a utilização de técnicas avançadas de programação exigindo, por vezes, o uso de *software* extremamente particular, complexo e que demanda conhecimento técnico específico, tanto da teoria quanto do *software* em questão.

A. Otimização

Também no que diz respeito aos problemas numéricos, gostaríamos de ressaltar, em especial, a possibilidade de paralelizá-los, agilizando (por vezes desproporcionalmente) a produção de resultados. Em diversos casos, tais otimizações são necessárias para que possamos produzir resultados oportunamente, levando em conta simplesmente a capacidade de paralelização do *hardware* moderno *versus* suas capacidades de execução linear.

A métrica utilizada para a capacidade de um dado *hardware* no contexto de aplicações numéricas é a quantidade de operações de ponto flutuante que podem ser executadas por segundo [FLOPS] (*floating point operations per second*) – veja [1], [2] para uma discussão aprofundada sobre essa métrica e sua relevância na computação moderna.

Notamos que, com frequência, o *hardware* moderno é muito mais eficiente em tarefas que aproveitam sua capacidade de paralelização [3] – não necessariamente aumentando por um fator linear em relação à disponibilidade de recursos (nesse contexto lidamos com *cores* ou *threads*[1]), pois existem diversos fatores a serem levados em conta como, por exemplo, concorrência [5].

A GPU (*Graphics Processing Unit*) é um componente especializado em tarefas de paralelização, de tal forma que, atualmente, é utilizada extensivamente para auxiliar na resolução de problemas nas áreas de física, biológia, química e até mesmo da matemática moderna sendo, por vezes, mais relevante do que a CPU (*Central Processing Unit*) de um dado sistema em aplicações como aquelas da ciência de dados [6].

[1]Para uma discussão extensiva sobre o papel de cada um desses recursos, veja [4].

B. *Sobre o uso de software livre em aplicações científicas e sua importância*

No que tange a utilização de *software* para aplicações científicas, nos preocupamos principalmente com a utilização de *software* livre, que, estritamente falando, é aquele que se alinha perfeitamente com os princípios da pesquisa científica.

Chamamos de ***software* livre** (abreviado por FLOSS ou FOSS em inglês) aquele que é aberto para ser lido e auditado, e que não depende de corporações (embora possa ser auxiliado por estas) para que se mantenha. Possuíndo comunidades autonomas de usuários e desenvolvedores que o mantém de acordo com interesses pessoais e plurais, porém não necessariamente sem o envolvimento de capital.

O *software* livre permeia todo o contexto computacional moderno, sendo utilizado extensivamente na internet (e.g. $\approx 80\,\%$ dos servidores utilizam *Linux* [7]), no contexto do desenvolvimento de *software* e, especialmente, na pesquisa científica.

Contrastamos tal com o ***software* proprietário**, onde a contribuição de um indivíduo sempre será atrelada com interesses corporativos pois pertence à uma empresa e é secreto sendo, portanto, impossível auditá-lo.

Existe ainda o *superset* de *software* livre, que é o ***software* aberto**, onde pode existir uma empresa que o mantém de acordo com seus interesses, porém este não é secreto, podendo ser auditado e aceitando contribuições individuais[2].

Note, então, que a utilização de *software* livre é preferível no contexto científico, pois soluções proprietárias demandam desmedida confiança a interesses corporativos, podendo causar conflitos de interesse tácitos e indo de encontro a princípios da ciência como a temos hoje, onde reprodutibilidade e transparência são essenciais.

Com isso em mente, advogamos pelo uso do *software* livre nesse contexto o que, infelizmente, ainda não é totalmente plausível como será explorado mais a frente.

II. Justificativa

Haja vista que a GPU é um componente de *hardware* sendo, então, necessário comandá-la de forma específica e precisa, idealmente gostaríamos de abstrair detalhes do *hardware* específico, de tal forma que tenhamos fino controle sobre sua utilização, sem comprometer, porém, seu desempenho (i.e. a abstração deve otimizar o que quer que seja tendo em mente o *hardware* específico).

A. *Compiladores*

Classicamente, utilizamos *shaders* para a programação de GPUs, sendo mais ou menos análogos à programação de CPUs em sua sintaxe, mas não necessariamente em sua lógica, o que torna seu uso não-trivial para alguém sem o conhecimento específico. O código do *shader* é compilado em etapas, sendo primeiramente convertido para uma linguagem intermediária, onde é otimizado com base em conceitos primitivos próprios para abordagens de paralelização. Depois, sendo convertido para um binário otimizado para a GPU específica onde será executado, e então enviado para o *hardware*, como ilustrado na 2. No contexto geral do Linux, essa é apenas uma pequena parte do processo de execução de um *shader*, o que podemos ver na 1.

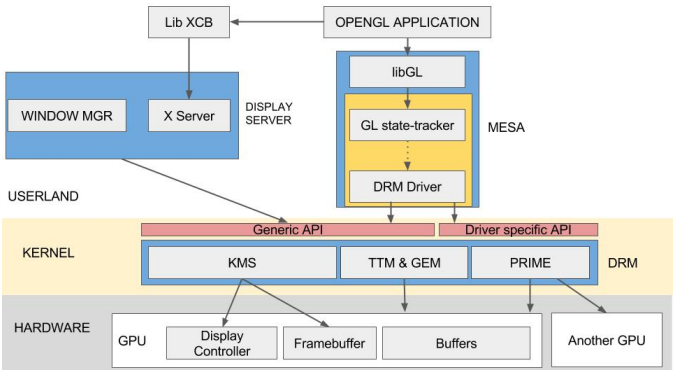

Figura 1: *Stack* gráfica utilizada em sistemas operacionais baseados no *kernel* Linux.

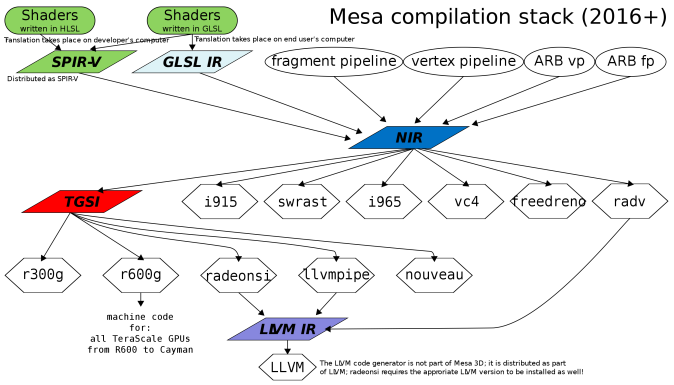

Figura 2: *Stack* mais recente de compilação utilizada pelo Mesa, um *software* responsável por APIs gráficas de código aberto no Linux. O Mesa recebe o código de um *shader* a ser processado, executando os passos descritos acima (II-A).

[2]No presente projeto, no entanto, vamos nos dedicar exclusivamente à problemática do *software* livre *versus software* proprietário.

B. Comparando o software livre com o proprietário para aplicações científicas

No contexto específico de aplicações numéricas, no presente projeto nosso interesse é voltado à **programação geral** com GPUs (GPGPU), onde temos utilização extensiva da linguagem **CUDA**, desenvolvida pela empresa NVIDIA – uma pioneira em computação gráfica.

A *stack* (conjunto de *software* e/ou *hardware*) necessária para a utilização do CUDA é, infelizmente, integralmente proprietária, desde a implementação de sua API (*aplication programming interface*), que é a interface pela qual desenvolvedores podem acessar um dado conjunto de funcionalidades, até os *drivers* necessários para a execução do *software*. Algumas alternativas como o **OpenCL** ou o **Vulkan** têm se tornado cada vez mais populares, porém é necessário notar que o seu uso não implica em uma *stack* livre, já que os *drivers* utilizados podem ser proprietários – e.g. OpenCL + NVIDIA implica que teremos que usar os *drivers* proprietários da fabricante, porém se usamos uma GPU da fabricante AMD, podemos utilizar *drivers* livres. No presente projeto então, focamos estritamente nas APIs, já que os *drivers* envolvem uma camada espessa de abstrações, dificuldades e tecnicidades muito além do escopo e do tempo planejado para o projeto.

Dada sua simplicidade e abrangência, o CUDA segue invicto em aplicações científicas que utilizam GPUs – essa afirmação infelizmente carece de fontes acadêmicas, porém, assumindo que a proporção de projetos abertos[3] (a maioria dos quais está disponível no GitHub) que utilizam uma dada tecnologia para projetos fechados é a mesma, podemos realizar uma busca simples no site:

- Utilizando a barra de pesquisa, buscamos por CUDA, e filtrando por linguagem[4] encontramos projetos que de fato utilizam a tecnologia (1669 resultados).
- Repetimos o processo[5], porém buscando por OpenCL (647 resultados) e Vulkan (798 resultados)[6].
- Estimando que a busca pelo termo "Vulkan" retornou 2/3 de resultados errados, já que ele é utilizado majoritariamente em aplicações que não se tratam de GPGPU, temos $1669/(647 + 798/3 + 1669) \gtrsim 64\,\%$[3] dos resultados utilizando CUDA (não necessariamente exclusivamente).

Notamos então, que a disponibilidade de alternativas livres (e viáveis) é de grande importância de grande importância para a comunidade científica contemporânea, como discutido em seção I-B.

III. Objetivos

Dada a utilização (aproximada) do CUDA em relação a alternativas livres, possuímos diversos questionamentos que devem ser abordados no presente projeto, como:

1) Qual a figura exata da adoção de *software* livre para aplicações científicas?
2) Onde o CUDA se sobressai em comparação com essas alternativas?
3) Como podemos diminuir essa diferença e como facilitar a migração para soluções que usam exclusivamente código livre?
4) O cenário ideal é factível?

Escolhemos o **OpenCL** e o **Vulkan** como objetos de análise nesse projeto. Ambos são fruto de especificações construídas pelo consórcio Khronos Group, do qual 170 organizações são parte e que desenvolve diversas outras especificações amplamente adotadas possuindo, portanto, um grande impacto na computação moderna. Implementações específicas de ambas especifícações serão escolhidas para uso durante o projeto, a fim de realizarmos aperfeiçoamentos e comparações.

Uma das possibilidades de aplicação real do presente projeto no futuro é procurar formas de se integrar o software livre disponível para GPGPU em projetos livres de maior porte como o DemiKernel [8], [9].

[3] Note que estes projetos não necessariamente são utilizados em pesquisas científicas, tomamos o número como base de referência para GPGPU genérica, o que inclui tais aplicações, e que muito provavelmente adotam ainda mais fortemente o uso de CUDA.

[4] A consulta específica utilizada nesse caso foi `cuda language:cuda language:c language:c++ start:>=5 followers:>=5` de tal forma que apenas os projetos com mais de 4 seguidores e mais de 4 estrelas foram mostrados, além de utilizarem qualquer uma das 3 linguagens (C, C++, CUDA). Esses filtros foram utilizados para vermos projetos minimamente relevantes.

[5] Nesse caso, retiramos a linguagem CUDA dos resultados.

[6] Note que, nesse caso, é impossível selecionar somente projetos relacionados à GPGPU, pois muitos destes não incluem essa palavra-chave, ou o termo correto (*Vulkan Compute*).

IV. Cronograma

Semestre	Código	Nome	Pós-graduação	Carga horária
2022.2	MAC0344	Arquitetura de Computadores	não	4 A
	MAC0414	Autômatos, Computabilidade e Complexidade	não	4 A
2023.1	MAC5752	Introdução à Computação Paralela e Distribuída	sim	4 A + 2 P + 4 E
	MAC5743	Sistemas Operacionais	sim	4 A + 2 P + 4 E
2023.2	MAC5716	Laboratório de Programação Extrema	sim	4 A + 2 P + 4 E
	MAC5717	Laboratório Avançado de Métodos Ágeis de Desenvolvimento de Software	sim	4 A + 2 P + 4 E
2024.1	PCS3866	Linguagens e Compiladores	não	4 A

Tabela I: Cronograma de disciplinas do projeto de avançado. Note que "A" representa a carga horária de aulas, "P" representa a carga horária de projeto e "E" representa a carga horária de estudos da disciplina em questão.

Além do que foi listado, também deverão ser feitos cursos extra-curriculares da NVIDIA, com intuito de aperfeiçoar o conhecimento a respeito de CUDA, para que possamos entender melhor suas vantagens e usos com relação às alternativas que serão abordadas.

V. Metodologia

Tendo em mente os objetivos pontuados na seção III, devemos começar estudando formas de contabilizar a quantidade de aplicações científicas que fazem uso de APIs gráficas livres e proprietárias (item 1), assim como as principais capacidades de cada uma e compará-las contra os casos de uso das principais áreas de estudo das ciências (item 2). Essa etapa deve ser realizada no primeiro semestre do projeto (2022.2), enquanto os temas de arquitetura de computadores e a base teórica para compilação são introduzidos nas disciplinas respectivas (MAC0344 e MAC0414).

A arquitetura de computadores é importante pois desejamos entender aspectos de otimização íntrinsecos à parte física desses sistemas complexos, para que assim seja possível implementar soluções viáveis para aplicações científicas (que requerem performance), e também úteis para a comunidade de usuários em geral.

Após adquirir esses dados e um conhecimento básico dos sistemas em questão, estudaremos a paralelização e sistemas operacionais nas disciplinas respectivas (MAC5752 e MAC5743), onde será desenvolvida um conhecimento mais específico das implementações necessárias no contexto de GPGPU, além do contexto global onde são executados. Nesse momento, começamos a explorar possibilidades de melhorias das APIs livres, primeiramente entendendo o que existe atualmente (i.e. estudando seu código) e depois conversando com a comunidade sobre o que seria mais útil ou mais viável dado a duração do projeto.

Por fim, devemos implementar as diversas melhorias fixadas anteriormente, o que poderá ser feito em até um ano de projeto, e pode abranger também a integração de ferramentas livres de GPGPU com outros projetos de interesse (veja a seção III §3 e/ou [8], [9]). Nesse contexto, as disciplinas de programação extrema e métodos ágeis (respectivamente MAC5752 e MAC5743) serão de grande utilidade a fim de se produzir código de qualidade e com metodologia. Além disso, a disciplina de métodos ágeis possui um rico histórico com relação ao *software* livre, sendo o berço do grupo de extensão de código livre na USP (FLUSP). Também serão utilizadas algumas referências bibliográficas como [10]–[15], para que possamos embasar a metodologia de desenvolvimento de software. Ao final do projeto (2024.1) a disciplina de compiladores servirá como concretização dos conhecimentos adquiridos em MAC0414, de tal forma a fornecer o panorama completo dos aspectos práticos da otimização do *software* a ser desenvolvido, nos permitindo aperfeiçoar o que fora desenvolvido até então.

Referências

[1] R. Dolbeau, "Theoretical peak flops per instruction set: a tutorial," *The Journal of Supercomputing*, vol. 74, no. 3, pp. 1341–1377, 2018.

[2] Y. Sun, N. B. Agostini, S. Dong, and D. Kaeli, "Summarizing cpu and gpu design trends with product data," *arXiv preprint arXiv:1911.11313*, 2019.

[3] K. Doi, R. Shioya, and H. Ando, "Performance improvement techniques in tightly coupled multicore architectures for single-thread applications," *Journal of Information Processing*, vol. 26, pp. 445–460, 2018.

[4] W. Magro, P. Petersen, and S. Shah, "Hyper-threading technology: Impact on compute-intensive workloads." *Intel Technology Journal*, vol. 6, no. 1, 2002.

[5] J. Zhou, J. Cieslewicz, K. A. Ross, and M. Shah, "Improving database performance on simultaneous multithreading processors," 2005.

[6] E. Buber and D. Banu, "Performance analysis and cpu vs gpu comparison for deep learning," in *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*. IEEE, 2018, pp. 1–6.

[7] N. W3Techs, "Usage statistics of operating systems for websites," 2022. [Online]. Available: https://w3techs.com/technologies/overview/operating_system

[8] I. Zhang, J. Liu, A. Austin, M. L. Roberts, and A. Badam, "I'm not dead yet! the role of the operating system in a kernel-bypass era," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019, pp. 73–80.
[9] I. Zhang, A. Raybuck, P. Patel, K. Olynyk, J. Nelson, O. S. N. Leija, A. Martinez, J. Liu, A. K. Simpson, S. Jayakar *et al.*, "The demikernel datapath os architecture for microsecond-scale datacenter systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 195–211.
[10] M. Aniche, *Effective Software Testing: A Developer's Guide*. Simon and Schuster, 2022.
[11] G. Narayan, K. Gopinath, and S. Varadarajan, "Structure and interpretation of computer programs," in *2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*. IEEE, 2008, pp. 73–80.
[12] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.
[13] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC Boston, 2018.
[14] D. M. Ritchie, S. C. Johnson, M. Lesk, B. Kernighan *et al.*, "The c programming language," *Bell Sys. Tech. J*, vol. 57, no. 6, pp. 1991–2019, 1978.
[15] B. Stroustrup, *The C++ programming language*. Pearson Education, 2013.