Multiparticionamento de Dados em GPU

Michel B. Cordeiro[1], **Wagner M. Nunan Zola** [1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)

michel.brasil.c@gmail.com, wagner@inf.ufpr.br

Resumo. *Este trabalho propõe um algoritmo de multiparticionamento em GPU. O algoritmo apresenta maior flexibilidade para definição de faixas de particionamento e alcança aceleração de até 83% em relação ao estado da arte.*

1. Introdução

O multiparticionamento é um problema de programação paralela cujo objetivo é reorganizar um vetor os dados em "*bins*" (ou "*buckets*"), contíguos na memória, utilizando uma função fornecida pelo programador para categorizar cada elemento em sua respectiva partição. O algoritmo, é utilizado em diversas aplicações como na construção de tabelas *hash*, em implementações do *Radix Sort*, e construção de *KD-trees*. Sua implementação eficiente é crucial para garantir alto desempenho nessas aplicações, entretanto, o multiparticionamento em GPU, se considerado como uma primitiva paralela em si, recebeu pouca atenção até o momento na literatura. Uma implementação eficiente, denominada *GPU multisplit*, foi apresentada por [Ashkiani et al. 2017] que, embora represente o estado da arte, apresenta algumas limitações, como a exigência de que o número de *bins* seja uma potência de dois e não ultrapasse 256. O presente trabalho propõe uma alternativa eficiente de multiparticionamento em GPU que suporta quantidade de partiçoes bem acima de 256 e sem apresentar a restrição de que o número de *bins* seja uma potência de dois.

2. Descrição do Algoritmo

O algoritmo proposto é dividido em duas etapas. Inicialmente, o vetor é segmentado em faixas, e a contagem de elementos em cada *bin* é realizada, gerando um histograma local para cada faixa e um histograma global para toda a entrada. Em seguida, aplica-se a operação de *scan* exclusivo aos histogramas, permitindo determinar a posição de cada *bin* no vetor de saída. A segunda etapa consiste na geração do vetor de saída, e para isso, duas abordagens foram consideradas: A primeira versão, denominada *buffer-per-warp*, utiliza *buffers* para armazenar temporariamente, na memória compartilhada, os elementos de cada *bin*, gravando-os na memória global apenas quando o *buffer* atingir 32 elementos. Isso permite que um *warp* inteiro realize a escrita de forma coalescida, liberando espaço no *buffer* para que outras *threads* possam armazenar seus elementos. A principal otimização é que a sobrecarga de manter os *buffers* é significativamente menor do que a de realizar um multiparticionamento em cada faixa, como feito na proposta de [Ashkiani et al. 2017], além de não impor limites tão rígidos em relação à quantidade de *bins* na qual o vetor de entrada será particionado. A desvantagem é a necessidade de manter um *buffer* de 32 elementos para cada *bin* na memória compartilhada, o que limita a quantidade máxima de *bins* que essa versão pode processar. A segunda abordagem, chamada de *large*, simplesmente aplica a função de categorização e grava o elemento no seu *bin* no vetor final. Essa versão usa memória global, e a memória compartilhada é utilizada apenas para produzir os histogramas e realizar o *scan* na primeira etapa do

algoritmo. Por isso, o algoritmo proposto neste artigo, que será chamado de *Multipartition*, seleciona a melhor abordagem dependendo da quantidade de *bins* e do tamanho da memória compartilhada da GPU utilizada.

3. Resultados e Discussões

Para avaliar a eficiência do *Multipartition*, foram conduzidos experimentos comparando-o com o *GPU multisplit*. Foram gerados conjuntos de dados com tamanhos de 1, 8, 16 e 32 milhões de elementos do tipo *unsigned int*, utilizando uma distribuição uniforme. Também foram realizados testes com dados gerados por uma distribuição binomial, porém, como os resultados foram similares aos obtidos com a distribuição uniforme, eles não foram reportados. A quantidade de *bins* em que cada conjunto de dados foi particionado variou de 40 a 12288. A função de categorização utilizada nos experimentos classifica a entrada em *bins* seguindo a fórmula $f(x) = \lfloor x / \max \rfloor$, onde $\max$ representa o maior valor no conjunto de dados de entrada. Os testes foram realizados 100 vezes e a vazão média de elementos processados por segundo foi reportada. Também foram calculados intervalos de confiança de 95%, mas não foram observadas variações maiores do que 0,5% em relação à média. Os experimentos foram conduzidos em processador Intel Xeon Silver 4314 @ 2.40GHz, utilizando CUDA versão 12.4, sistema operacional Linux Ubuntu 20.04.3 LTS e GPU NVIDIA A4500, com 48 KiB de memória compartilhada por bloco de *threads*. Como o algoritmo mantém um *buffer* por *bin* na memória compartilhada, a quantidade máxima de *bins* que pode ser processada nesta máquina é 361. Para valores superiores, é adotada a abordagem *large*, que pode processar até 12288 *bins*.

N	Versão	Quantidade de bins						
		40	64	100	128	200	256	361
1 Milhão	GPU Multisplit	-	**22.7**	-	**20.4**	-	14.3	-
	Multipartition	**16.7**	16.7	**16.7**	16.7	**16.9**	**16.9**	**17.9**
	Speedup	-	0.73	-	0.82	-	1.19	-
8 Milhões	GPU Multisplit	-	**33.2**	-	29.2	-	19.3	-
	Multipartition	**31.4**	31.3	**30.4**	**29.9**	**29.5**	**29.5**	**29.4**
	Speedup	-	0.94	-	1.02	-	1.53	-
16 Milhões	GPU Multisplit	-	34.7	-	30.3	-	19.6	-
	Multipartition	**35.8**	**35.3**	**33.8**	**31.8**	**31.6**	**31.9**	**31.9**
	Speedup	-	1.02	-	1.05	-	1.63	-
32 Milhões	GPU Multisplit	-	35.2	-	30.4	-	18.3	-
	Multipartition	**37.3**	**36.7**	**35.0**	**34.0**	**33.5**	**33.4**	**33.4**
	Speedup	-	1.04	-	1.12	-	1.83	-

Vazão do Multisplit (Bilhões de elementos processados/segundo)

Tabela esquerda: pequenas quantidades de bins — **Tabela direita: grandes quantidades de bins**

N	Versão	Quantidade de bins							
		512	1024	2048	3000	4096	5000	8192	12288
1 Milhão	GPU Multisplit	-	-	-	-	-	-	-	-
	Multipartition	11.6	10.9	10.5	10.2	10.1	9.8	8.5	7.4
8 Milhões	GPU Multisplit	-	-	-	-	-	-	-	-
	Multipartition	8.4	6.8	6.0	5.9	5.9	5.8	5.7	5.6
16 Milhões	GPU Multisplit	-	-	-	-	-	-	-	-
	Multipartition	7.5	6.1	5.5	5.2	5.1	5.1	5.1	5.0
32 Milhões	GPU Multisplit	-	-	-	-	-	-	-	-
	Multipartition	7.2	5.7	5.1	4.9	4.8	4.8	4.7	4.7

Tabela 1. Resultados dos experimentos. Valores anotados com '-' representam limitações do algoritmo *GPU multisplit*. Nesses casos, o *speedup* do *Multipartition* em relação ao *GPU multisplit* não pode ser calculado.

Os resultados estão descritos na Tabela 1. Observa-se que o *Multipartition* apresenta bom desempenho para grandes quantidades de elementos e *bins*, alcançando um *speedup* de até 1.83 em relação ao *GPU Multisplit* para 32 milhões de elementos e 256 *bins*. Isso demonstra a eficácia das estratégias adotadas, que permitem manter uma vazão consistente mesmo para maior quantidade de *bins*. Por outro lado, há uma queda significativa na vazão do algoritmo quando há grandes quantidades *bins*, devido ao uso da memória global. Portanto, novas abordagens para mitigar esse problema devem ser exploradas em trabalhos futuros.

Agradecimentos
Parcialmente suportado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processo 407644/2021-0, bem como pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Programa de Excelência Acadêmica (PROEX).

Referências

Ashkiani, S., Davidson, A., Meyer, U., and Owens, J. D. (2017). GPU Multisplit: an extended study of a parallel algorithm. *ACM Transactions on Parallel Computing*.