Estudo de desempenho e de eficiência energética em simulação de dinâmica de fluidos multifásicos nas arquiteturas NVIDIA Volta V100 e Grace Hopper GH200

Thiago Teixeira[1], **Frederico L. Cabral** [1], **Micaella Coelho** [1],**Luciano Leite** [2], **Rodrigo Surmas** [3], **Marcio Borges** [1], **Carla Osthoff**[1]

[1]Laboratório Nacional de Computação Científica - LNCC
Av. Getulio Vargas, 333 - Quitandinha, Petrópolis - RJ, 25651-075

[2]COPPE-Federal University of Rio de Janeiro,
Mailbox 68506, Rio de Janeiro, Rio de Janeiro, 21941-972, Brazil

[3]CENPES-Petrobras, Av. Horácio Macedo, 950, Rio de Janeiro,
Rio de Janeiro, 21941-915, Brazil

{tteixeira,fcabral,micaella,mrborges,osthoff}@lncc.br,
lleite@lamce.coppe.ufrj.br, surmas@petrobras.com.br

Abstract. *Este estudo apresenta uma análise de desempenho e eficiência energética do Simulador Multifásico baseado no método Lattice Boltzmann (LBM) utilizando as arquiteturas NVIDIA V100 e NVIDIA GH200. O método LBM é amplamente utilizado para simulações de dinâmica de fluidos computacional devido à sua capacidade de modelar fluxos de permeabilidade relativa e complexos. Neste trabalho, foram realizadas simulações com malhas sintéticas variadas para avaliar o desempenho e o consumo energético em ambas as arquiteturas. Os resultados indicam que a arquitetura GH200 supera consistentemente a V100 em termos de desempenho, especialmente em malhas maiores, com ganhos de até 2.82 vezes. Além disso, a GH200 demonstrou uma eficiência energética superior, consumindo até 53% menos energia em comparação à V100. A análise de perfilagem com o NVIDIA NSight Compute identificou os principais fatores que contribuem para a perda de desempenho, destacando a necessidade de otimização no acesso à memória.*

1. Introdução

Simulações de fluidos multifásicos são de extrema importância na indústria de Óleo e Gás. Esses fenômenos e fluxos surgem quando dois ou mais fluidos, que não se misturam facilmente, como ar e água, entram em interação. As interações de fluidos multifásicos são amplamente presentes em processos naturais e industriais. Esses fluxos podem envolver fluidos multifásicos de um único componente, como água e seu próprio vapor, ou de múltiplos componentes, como óleo e água. Exemplos práticos incluem a recuperação de recursos petrolíferos de reservatórios, o comportamento da água no solo, operações de células de combustível e a evolução de nuvens.

O método Lattice Boltzmann (LBM) tem se desenvolvido rapidamente nas últimas duas décadas, tornando-se uma ferramenta poderosa e essencial nos estudos de fluxo

de fluidos, especialmente em fluxos multifásicos na dinâmica dos fluidos computacional. O LBM apresenta várias vantagens em comparação com os métodos tradicionais de dinâmica dos fluidos computacional [Chen et al. 1994].

Este estudo apresenta uma análise de desempenho e eficiência energética de um código de simulação de dinâmica de fluidos multifásicos que utiliza uma biblioteca chamada GRAD-LBM nas arquiteturas NVIDIA Volta V100 e NVIDIA Grace Hopper GH200. O objetivo é realizar uma perfilagem em ambas as arquiteturas para identificar melhorias, diferenças e problemas de desempenho decorrentes dos avanços tecnológicos da nova arquitetura.

A segunda seção discute o método numérico, suas particularidades e eficiência paralela. A terceira seção revisa trabalhos relacionados. A quarta seção descreve o algoritmo, os passos do simulador e a construção da estrutura de dados. A quinta seção detalha a metodologia utilizada na análise de desempenho e eficiência energética, incluindo a realização dos testes e os resultados obtidos. A sexta seção apresenta as conclusões do estudo e trabalhos futuros.

2. Método numérico

O Método Lattice-Boltzmann (LBM) é uma técnica poderosa para simulações de dinâmica de fluidos computacional. Este método é particularmente eficaz para simulações de fluxo de fluidos em microescala através de espaços porosos, permitindo a modelagem de problemas complexos de fluxo. Aplicações do LBM incluem fluxos geofluídicos, como o fluxo de gás, água, óleo ou magma através de meios porosos, transporte de solutos em macroescala e dispersão de contaminantes atmosféricos em ambientes urbanos, entre outros [Chen et al. 1994].

As simulações LBM são modeladas em grades unidimensionais, bidimensionais ou tridimensionais, onde cada nó na grade representa uma região quantizada do espaço contendo fluido ou sólido. O fluido é representado por pacotes que se propagam através da grade em passos de tempo discretos, colidindo entre si nos nós da grade. Devido à natureza local das colisões, o cálculo depende apenas dos dados dos nós vizinhos, permitindo uma implementação eficiente em ambientes paralelos, como GPUs. No entanto, essas simulações apresentam desafios devido à necessidade de comunicação entre os nós vizinhos a cada passo de tempo.

A disposição de velocidades utilizada no método é o D3Q19, onde ”D”representa as dimensões e ”Q”representa as velocidades calculadas. No caso do D3Q19, os pacotes de fluidos podem se mover em 19 direções, incluindo o nó central e 18 nós vizinhos. Métodos alternativos podem utilizar outras disposições como D3Q15, D3Q27 ou, em duas dimensões, D2Q9, entre outros.

Essa disposição de velocidades determina as direções discretas nas quais as partículas podem se mover e colidir em cada nó da grade. A escolha dessa disposição depende do tipo de problema de fluxo a ser resolvido e do nível de precisão desejado.

O modelo utilizado no algoritmo simula o fluxo multifásico de partículas, utilizando ambos os fluidos para os cálculos. Esta nomenclatura é usada para identificar fluidos diferentes. Para estender o método Lattice-Boltzmann ao fluxo de dois fluidos imiscíveis, é necessário duplicar a informação em cada ponto da rede. No LBM, isso é

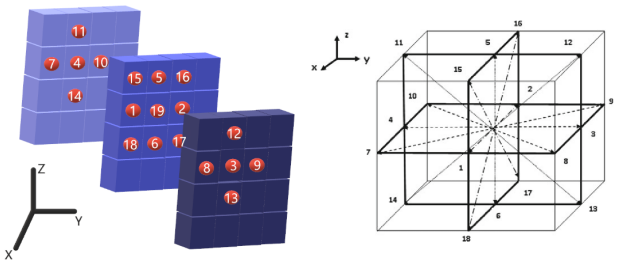

Figura 1. Disposição das velocidades utilizadas no *stencil* D3Q19.

feito postulando uma distribuição de partículas para cada fluido[Huang et al. 2015]. Para simular o fluxo multifásico, novas regras de dispersão são criadas para causar a separação dos fluidos na interface e emular os efeitos da tensão superficial. A força em cada ponto da rede depende da densidade de cada fluido nos pontos vizinhos e empurra cada fluido na direção de maior densidade do próprio fluido.

A força resultante é adicionada ao operador de colisão de forma que não altere o momento total dos dois fluidos. A magnitude da força é ajustada para produzir a tensão superficial desejada entre os dois fluidos. Sob essas regras, uma mistura inicial aleatória dos fluidos se separa, formando uma distribuição de equilíbrio onde uma interface esférica separa os dois fluidos, replicando o comportamento físico esperado em misturas de óleo e água.

O tempo é avançado na simulação Lattice Boltzmann em passos de tempo discretos compostos por duas partes: uma etapa de colisão, na qual o momento é trocado entre os pacotes de fluido em cada nó, e uma etapa de fluxo, na qual os pacotes de fluido são deslocados para o próximo nó ao longo de seu caminho. Durante a etapa de fluxo, dezoito dos pacotes de fluido saindo de cada nó são distribuídos para as localizações correspondentes de pacotes de fluido entrantes nos nós vizinhos. No passo de tempo seguinte, os nós utilizam os pacotes de fluido entrantes e um pacote de fluido estacionário da etapa de colisão. O método Lattice Boltzman foi codificado para a linguagem C++ com diretivas CUDA gerando um *kernel* para cada uma das duas etapas. Cada *kernel* acessando sua memória local e transferindo informações no final da execução, gerando uma grande quantidade de transferência de dados entre as memórias. De forma a otimizar o tempo de execução do código em GPU, foi gerado um novo algoritmo em [Bailey et al. 2009] para permitir o compartilhamento de memória entre os *kernels*, ao custo de gerar um aumento no tempo de sincronização na execução dos mesmos, chamado de padrão A-A, que está sendo utilizado neste trabalho. O trabalho de [Bailey et al. 2009] também demonstra que os custos com a sincronização são desprezíveis em relação aos ganhos com a diminuição dos gastos com a transferência de dados de memória.

O padrão A-A necessita apenas de um conjunto de pacotes de fluido na memória, reduzindo pela metade a exigência de memória para um dado tamanho de grade. Esse padrão alterna a execução de duas rotinas: a rotina para iterações pares (A-A:1) executa uma única colisão, e a rotina para iterações ímpares (A-A:2) executa uma etapa combinada de fluxo-colisão-fluxo. Ambas as rotinas escrevem os dados dos pacotes de fluido de saída nos mesmos locais de onde os dados de entrada foram lidos, permitindo manter apenas um conjunto de pacotes de fluido na memória e aumentando significativamente

os tamanhos das grades que podem ser armazenadas e computadas inteiramente na GPU. A rotina (A-A:2) evita dependência de dados durante suas três etapas, permitindo que os nós sejam computados em qualquer ordem, o que é essencial para a arquitetura da GPU.

3. Trabalhos Relacionados

Nas últimas décadas, o método Lattice-Boltzmann (LBM) tem sido cada vez mais reconhecido como uma ferramenta valiosa na dinâmica de fluidos computacional. Os algoritmos mais conhecidos para a implementação da equação padrão de Lattice-Boltzmann (LBE) são os algoritmos de duas grades e de dois passos. [Mattila et al. 2007] introduz um novo algoritmo, denominado algoritmo de troca, para a implementação do LBE. Os resultados obtidos por esse algoritmo de troca é apresentado em uma arquitetura *multicore* AMD *Opteron* 246 processadores e 4GB de memória, e apresentou bons resultados nessa arquitetura quando comparada à versão LBE. Uma variação dessa estratégia é utilizada no Simulador apresentado nesse trabalho, porém aplicada em arquiteturas GPU.

Em [Rigon et al. 2024] é explorado a implementação e análise de um sistema Multi-GPU para a aplicação do Método Fletcher na exploração geofísica. A estratégia proposta enfatiza uma abordagem criteriosa na divisão de carga de trabalho, considerando a localização dos dados e a capacidade de processamento das GPUs. Foi criada uma metodologia para analisar e comparar os resultados de múltiplas distribuições de carga de trabalho multi-GPU, além do seu consumo energético. Os testes foram realizados em uma arquitetura de GPUs V100. Essa metodologia foi utilizada neste trabalho buscando comparar os resultados de desempenho e consumo energético do Simulador nas arquiteturas V100 e GH200.

Em [Bailey et al. 2009] é discutido que os Métodos Lattice Boltzmann (LBM) são usados para a simulação computacional da dinâmica de fluidos Newtonianos. As simulações baseadas em LBM são prontamente paralelizáveis. Entre os três métodos, as implementações em GPU alcançaram o maior desempenho de simulação por chip. A arquitetura utilizada foi a NVIDIA GeForce 8800 Ultra e diversas estratégias foram analisadas e melhoram os resultados anteriores de LBM em precisão simples em GPU para o modelo D3Q19, aumentando a ocupação do multiprocessador da GPU, resultando em um aumento de desempenho máximo de 20%, e introduzindo um método de armazenamento eficiente que reduz os requisitos de RAM da GPU em 50% com uma leve degradação no desempenho. O trabalho atual utiliza uma variação das estratégias apresentadas, utilizando principalmente a estratégia de redução 50% do uso da memória RAM.

O trabalho de [Herschlag et al. 2018] destaca que o desempenho do método Lattice Boltzmann (LBM) em GPUs depende fortemente dos padrões de acesso à memória. Quando o LBM é avançado com GPUs em domínios computacionais complexos, os dados geométricos são normalmente acessados indiretamente, e os dados de malha são acessados lexicograficamente no *layout* Estrutura de *Array* (SoA). Uma análise da cada uma das estruturas de dados é apresentada e para o *layout* da malha, o *layout* da Estrutura de *Arrays* Coletados (CSoA) supera o *layout* SoA. As analises foram realizadas na arquitetura NVIDIA K40 com múltiplas GPUs. Este trabalho utiliza a estrutura de dados SoA, e apesar da estrutura CSoA apresentar um desempenho melhor, na arquitetura da NVIDIA V100 a estratégia não foi considerada vantajosa após perfilagem. Contudo, esse trabalho apresenta resultados na GH200 que mostram que um possível trabalho futuro na estrutura

de dados, onde a estrutura CSoA pode apresentar vantagens.

4. Algoritmo

A aplicação descrita é uma implementação acelerada do algoritmo LBM da biblioteca GRAD-LBM, desenvolvida em C++ e utilizando diretivas CUDA.

O Algoritmo 1 apresenta os passos do simulador. Inicialmente, um processo MPI é iniciado para cada GPU, processando um subdomínio da malha por placa GPU. Em seguida, é realizada a leitura da malha de dados regular em formato .VTK com informações sobre a porosidade dos sítios, para gerar a decomposição dos domínios. A decomposição de domínio busca balancear a carga dos sítios porosos entre os processos MPI e cada processo tem um subdomínio da malha de dados carregado inteiramente na GPU antes da execução dos *kernels*. Essa estratégia busca uma maior escalabilidade.

Algorithm 1 GRAD-LBM

```
1: Inicialização do MPI
2: Leitura da malha e decomposição de domínio
3: Construção do vetor do padrão A-A
4: Inicialização do problema
5: while passo no tempo <passo final do
6:     Kernel A-A:1
7:     Inicialização e finalização da comunicação
8:     Kernel A-A:2
9:     Inicialização e finalização da comunicação
10:    Saída do problema
11: end while
12: Finalização do problema
13: Finalização do MPI
14: Saída do simulador
```

No passo 3, o simulador cria um vetor que é utilizado pelos *kernels* e contém as 19 velocidades de cada fluido a serem utilizadas no cálculo do *stencil* do método Lattice Boltzmann. A estrutura de dados do vetor é criada usando a estrutura SoA. O vetor possui as velocidades agrupadas por sítio, garantindo que as velocidades dos sítios próximos estejam próximas na região da memória. Essa estrutura apresenta resultados melhores nas GPUs.

A tabela 1 apresenta os dados presentes no vetor criado pelo Simulador e transferido do (CPU)*Host* para o (GPU)*Device*. As variáveis R, B e W possuem os pacotes de fluidos, índice dos vizinhos são necessários para a troca de mensagens e o índice dos poros é utilizado para identificar o nó na malha original.

Tabela 1. Armazenamento de memória da estrutura de dados do Simulador

Estrutura	**Dimensão**	**Tipo de variável**
R	# de poros * distribuição [0-18]	*double* (64-bit float)
B	# de poros * distribuição [0-18]	*double* (64-bit float)
W	# de poros * distribuição [0-18]	*double* (64-bit float)
índice de vizinhos	# de poros * distribuição [0-18]	*unsigned int* (64-bit)
índice dos poros	# de poros	*unsigned int* (64-bit)

A tabela 1 apresenta o calculo necessário para entender quantos Mbs terão o conjunto de dados a ser enviado para o *device*, com isso é possível identificar quantos *devices* são necessários para executar uma malha de dados. A tecnologia *Unified Memory* da NVIDIA é utilizada para realizar a troca de mensagens entre *Host* e *Device*. Essa tecnologia permite que, ao alocar o espaço necessário para os dados no *Host* e no *Device*, a troca de mensagens entre ambos não necessite de parâmetros de envio de mensagens CUDA. Com isso, a transferência de dados do *Host* para o *Device* é realizada ainda nesse passo.

O passo 4 inicializa a execução do método Lattice Boltzmann utilizando o *layout* D3Q19, que é uma grade tridimensional regular (D3) com 19 velocidades distintas de pacotes de fluido (Q19). Entre os passos 5 e 11, ocorre a execução em GPU de dois *kernels*. O *kernel* `A-A:1` realiza a etapa única de colisão, enquanto o *kernel* `A-A:2` realiza a etapa combinada de fluxo-colisão-fluxo. Esta etapa combinada é necessária para o padrão A-A e possui maior intensidade aritmética do que o *kernel* `A-A:1`, que realiza uma única etapa. A maior intensidade aritmética deve-se à utilização do mesmo conjunto de dados para três etapas. Cada etapa 7 e 9 realiza a comunicação entre os sítios, mas não há comunicação *Host* para o *Device*, não gerando bloqueios. Apenas o *Device* envia dados para o *Host*, de forma assíncrona utilizando o *Unified Memory*.

O passo 12 executa a finalização do problema, enviando os dados de cada GPU para a memória da CPU. O passo 13 finaliza os processos MPI, e o passo 14 escreve os dados da malha final de volta no disco.

5. Metodologia

Este trabalho tem o objetivo de apresentar um estudo de desempenho e consumo energético do Simulador Multifásico que utiliza um modelo LBM. Esse perfil foi realizado utilizando as arquiteturas NVIDIA GH200, fornecida pela NVIDIA, e a arquitetura NVIDIA V100 presente no Supercomputador SDumont. O trabalho apresenta também uma perfilagem para analisar as particularidades do simulador e possíveis causas de degradação de desempenho. A escolha das duas arquiteturas se dá devido ao Supercomputador SDumont ter dispositivos V100 e terá um *upgrade* com placas H100, presentes na GH200.

Para os estudos de desempenho e consumo energético, foram escolhidas oito malhas sintéticas criadas especialmente para os testes. Essas malhas são cúbicas e possuem sítios porosos dispostos em um formato cilíndrico. Elas foram criadas com dimensões e sítios porosos múltiplos de 32, visando o melhor desempenho nos dispositivos ao terem tamanhos compatíveis com a quantidade de *warps*. As malhas têm as seguintes dimensões: 278x278x278, 342x342x342, 406x406x406, 470x470x470, 502x502x502, 534x534x534, 630x630x630 e 780x780x780. As oito malhas foram executadas na arquitetura GH200, no entanto, a malha 780^3 não coube no dispositivo V100 e não gerou resultados.

5.1. Arquiteturas

A arquitetura GH200 da NVIDIA conta com um *superchip* com uma CPU Grace de 72 núcleos, uma GPU NVIDIA H100, 480GB de memória LPDDR5X, 96GB de memória interna do dispositivo e 900GB/s de transferência de memória via NVLink.

A arquitetura V100 presente no Supercomputador SDumont conta com 94 nós computacionais com GPU, onde cada nó possui: 2x Intel Xeon Skylake 6252 (24 núcleos

por CPU), 384GB de memória RAM, e 4x NVIDIA Volta V100.

Ao compararmos as duas GPUs utilizadas, foi coletado os dados de cada uma das arquiteturas diretamente dos documentos da NVidia e esses dados foram alimentados na Tabela 2:

Tabela 2. Tabela comparativa das GPUs V100 e H100

	V100	**H100**
GPC	6	8
TPC	7	66
SMs/TPC	2	2
SMs/GPU	84	132
FP32 CUDA Cores/SM	64	128
FP32 CUDA Cores/GPU	5376	16896
FP64 CUDA Cores/SM	32	64
FP64 CUDA Cores/GPU	2688	8448
INT32 CUDA Cores/SM	64	64
INT32 CUDA Cores/GPU	5376	8448
Tensor Cores/SM	8	4
Tensor Cores/GPU	672	528
GPU Boost Clock (MHz)	1530	1830
Memory Size (GB)	32	80
Memory BW (GB/s)	900	3352
Register File / SM (KB)	128	256
Register File / GPU (KB)	10752	33792
Shared Memory / SM (KB)	96	228
L2 Cache (MB)	60	50

As maiores diferenças entre as arquiteturas estão no número de CUDA Cores/GPU nos núcleos de ponto flutuante de precisão simples e dupla, que dobraram de tamanho por SM; no aumento significativo do número de SM por GPU, onde a V100 possui 84 e a H100 possui 128; e no aumento do tamanho de memória L1 e L0 por SM, onde na V100 a memória L1 (memória compartilhada/L1) por SM foi de 96KB para 228KB e a memória L0 (memória dos registradores) por SM foi de 128KB para 256KB.

5.2. Protocolo Experimental

Para os estudos de desempenho, foram realizadas cinco simulações para cada malha. Todas as execuções foram configuradas com um máximo de 10000 passos e com coleta de tempo de execução das funções do *kernel* a cada 100 passos no tempo, coletando a média de 100 passos até o fim das execuções e somando essas médias para gerar a métrica de tempo de execução da função a cada 100 passos. A métrica de média de passos no tempo foi escolhida devido aos *kernels* serem as funções que mais consomem o tempo de execução.

O mesmo procedimento foi realizado para o estudo de consumo energético, utilizando o comando:

```
\textit{nvidia-smi query-gpu=index,gpu_name,timestamp,power.draw,
clocks.sm,clocks.mem,clocks.gr}
```

Esse comando coleta o consumo em *watts* a cada segundo de execução. A média total de consumo para cada uma das execuções foi utilizada para gerar a métrica de consumo em *watts* a cada 100 passos no tempo da função.

Para a perfilagem, foi utilizada a malha 406^3 para coletar dados da execução do simulador em ambas as arquiteturas, com o objetivo de analisar as particularidades do código e suas deficiências de desempenho. O perfilador NVIDIA NSight Compute foi utilizado para realizar a perfilagem em ambas as arquiteturas.

5.3. Testes e Resultados

Os testes realizados em ambas as arquiteturas com 5 execuções para cada malha e a coleta da métrica de média de tempo de execução a cada 100 passos por função geraram o perfil de desempenho apresentado no lado esquerdo da Figura 2.

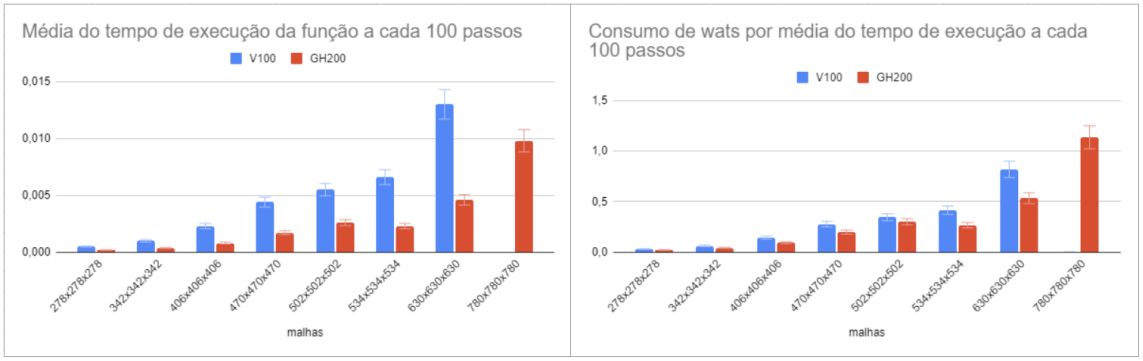

Figura 2. Perfil de desempenho com a coleta do tempo de execução das funções executadas pelos *kernels* e Perfil energético em *watts* gastos por média de tempo de execução a cada 100 passos.

O eixo X apresenta as malhas usadas para as execuções, o eixo Y apresenta os tempos de execução em segundos. O gráfico mostra a média do tempo gasto na execução das funções executadas nos *kernels* `Cuda_w_update`(A-A:1) e `Cuda_LBM_update`(A-A:2) a cada 100 passos. Os valores coletados ao fim das 5 execuções e suas médias são apresentados em azul para a arquitetura V100 e em vermelho para a arquitetura GH200. O gráfico apresenta tempo de execução; ou seja, quanto maior a barra, maior o tempo gasto e menor o desempenho quando comparado a uma barra menor. Como pode ser visto no gráfico, em todas as execuções a arquitetura GH200 tem um desempenho melhor quando comparada à arquitetura V100. Em malhas menores, o desempenho não é tão diferente, porém em malhas maiores a performance da GH200 se mostra muito superior, atingindo um desempenho 2.82 vezes maior no caso da malha 630^3.

Os testes realizados em ambas as arquiteturas com 5 execuções para cada malha e a coleta da métrica de média do consumo em *watts* gastos a cada 100 passos por função geraram o perfil energético apresentado no lado direito da Figura 2. O eixo X apresenta as malhas usadas para as execuções, o eixo Y apresenta o consumo em *watts* por média de tempo de execução a cada 100 passos. O gráfico mostra o consumo em *watts* pela média de tempo gasto na execução das funções executadas nos *kernels* `Cuda_w_update` e `Cuda_LBM_update` a cada 100 passos. O consumo energético foi coletado apenas das GPUs utilizadas; não realizamos a coleta de consumo energético das CPUs. Outro ponto importante é que o sistema NVIDIA GH200 pode equilibrar a potência entre a CPU NVIDIA Grace e a GPU Hopper. O *Superchip Grace Hopper* permite transferir a potência da CPU para a GPU quando necessário, consumindo assim mais *watts* em execuções na

GPU, porém economizando na CPU. Desta forma, o perfil energético possui o consumo apenas das GPUs e, como pode ser visto, o consumo da arquitetura V100 é maior em todas as execuções quando comparado à arquitetura GH200. A performance energética da arquitetura GH200 é melhor em todos os casos, chegando a uma eficiência de 53% na execução com a malha 630^3.

5.4. Perfilagem

O NVIDIA NSight Compute foi utilizado para realizar a perfilagem do Simulador. A coleta de dados das execuções foi realizada buscando identificar o *kernel hotspot* e os problemas que causam perda de desempenho. A coleta de dados foi realizada na malha 406^3 em ambas as arquiteturas. Foram analisados ambos os *kernels* `W_update` e `LBM_update`. O *kernel* `W_update` realiza uma etapa de colisão e o *kernel* `LBM_update` possui três etapas (*Streaming/Collision/Streaming*), sendo considerado o *kernel hotspot* devido à sua maior intensidade aritmética.

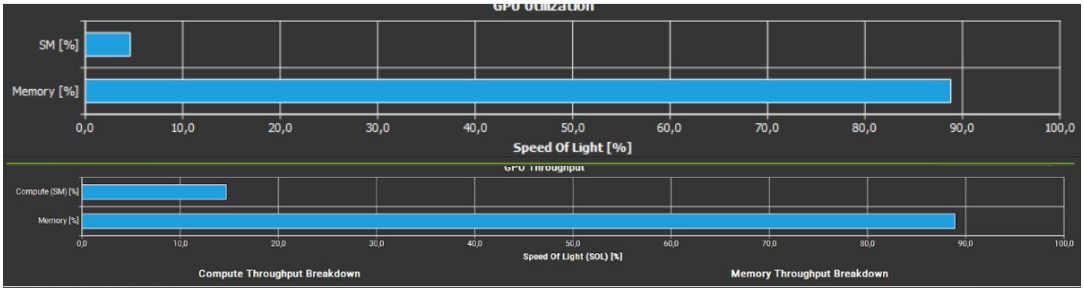

Figura 3. Análise da ocupância para o *kernel* `W_update` nas arquiteturas V100 e GH200, respectivamente.

A Figura 3 mostra a ocupância do *kernel* `W_update` e o consumo do espaço de memória global do dispositivo em porcentagem nas arquiteturas V100 e GH200, respectivamente. A ocupância deste simulador é relativamente baixa devido à quantidade de registradores necessários para cada cálculo. O número de bytes necessários para cada instrução exige uma quantidade significativa de registradores, forçando a subutilização de *warps*. Por esse motivo, podemos ver que na arquitetura GH200, que possui o dobro de memória L1 e L0, a ocupância do *kernel* é maior por utilizar menos registradores por *thread*.

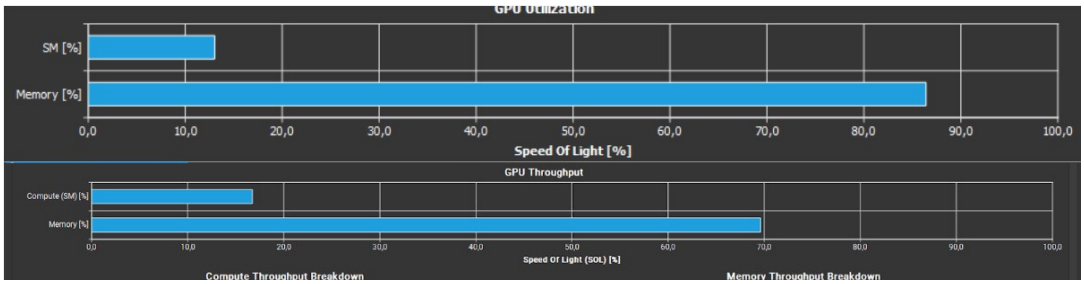

Figura 4. Análise da ocupância para o *kernel* `LBM_update` nas arquiteturas V100 e GH200, respectivamente.

A Figura 4 apresenta a mesma análise anterior, porém agora para o *kernel* `LBM_update`. Aqui, a ocupância é maior devido à maior intensidade aritmética. O

conjunto de dados utilizado para realizar as três etapas (*Streaming/Collision/Streaming*) é o mesmo, resultando em uma menor quantidade de registradores por *thread*. Ainda assim, ao comparar as arquiteturas V100 (imagem superior) e GH200 (imagem inferior), a arquitetura GH200 apresenta maior ocupância.

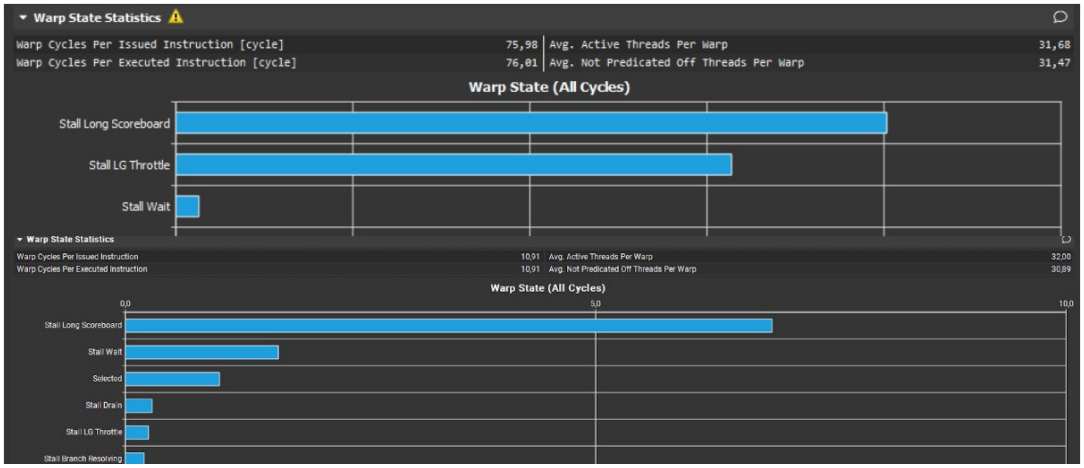

Figura 5. Análise dos problemas causadores de perda de desempenho para o *kernel* `LBM_update` nas arquiteturas V100 e GH200, respectivamente.

A Figura 5 apresenta uma análise dos problemas que causam perda de desempenho no *kernel* `LBM_update`, que é o *hotspot* do Simulador. A imagem superior representa a arquitetura V100 e a inferior a arquitetura GH200. O perfilador NVIDIA NSight Compute identifica várias causas de perda de desempenho. O maior problema de desempenho identificado é o "*Stall Long Scoreboard*", um tipo de *warp stall*. Esse problema ocorre quando a memória L1 não possui o dado solicitado pela L0, resultando em um *cache miss* e uma busca do dado na L2. Outro problema significativo na arquitetura V100 é o "*Stall LG Throttle*", que ocorre quando uma instrução não é iniciada no ciclo devido à falta de dados carregados no L0 dos registradores, causando espera por dados da L1. Esses problemas indicam que a maior limitação de desempenho na arquitetura V100 está na demora para preencher os dados nos registradores e recuperar dados da L2, causada pelo sincronismo entre os *kernels*.

Na análise do mesmo *kernel* na arquitetura GH200, o problema "*Stall Long Scoreboard*"persiste, indicando um claro problema de *cache miss*. A transferência de dados entre as memórias é maior, e o tamanho do L1 e do L0 é maior, mas o tamanho do L2 é menor. Isso elimina o problema de espera por dados no L0, mas ainda há muitos *cache misses* quando a L1 busca dados na L2.

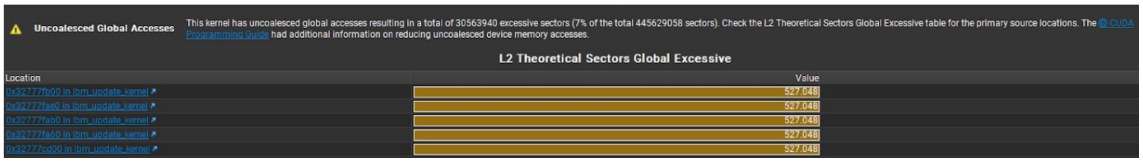

Figura 6. Análise de *cache miss* para o *kernel* `LBM_update` na arquitetura GH200.

A Figura 6 mostra a análise de *cache miss* no *kernel* `LBM_update` na arquite-

tura GH200, confirmando o problema de *cache miss*. O perfilador indica que há não-coalescência dos dados, o que está causando perda de desempenho. Esta análise sugere que o *kernel* `LBM_update` pode ganhar desempenho implementando estratégias de coalescência de dados na arquitetura GH200, algo que não era necessário na arquitetura V100.

6. Conclusão

Este estudo apresentou uma análise de desempenho e eficiência energética do Simulador Multifásico baseado no método Lattice Boltzmann (LBM) utilizando as arquiteturas NVIDIA V100 e NVIDIA GH200. A comparação entre essas arquiteturas revelou diferenças significativas em termos de desempenho e consumo energético, especialmente ao lidar com malhas maiores e mais complexas.

Os resultados demonstraram que a arquitetura GH200 supera consistentemente a V100 em termos de desempenho, com ganhos que chegam a 2.82 vezes no caso da malha 630^3. Isso se deve principalmente ao maior número de CUDA Cores, maior quantidade de memória L1 e L0 por *Streaming Multiprocessor* (SM) e uma melhor capacidade de balanceamento de carga proporcionada pela arquitetura GH200. Adicionalmente, a eficiência energética da GH200 também se mostrou superior em todos os testes, com uma economia de até 53% em relação à V100.

A análise de perfilagem realizada com o NVIDIA NSight Compute identificou os principais fatores que contribuem para a perda de desempenho em ambas as arquiteturas. Na arquitetura V100, os problemas de "*Stall Long Scoreboard*"e "*Stall LG Throttle*"foram os principais causadores de ineficiências, indicando problemas com cache miss e latência na transferência de dados entre registradores e memória. Na arquitetura GH200, embora o problema de "*Stall Long Scoreboard*"persista, a maior quantidade de memória e melhor gerenciamento de recursos mitigam muitos dos problemas encontrados na V100.

Com base nas análises de cache miss, identificamos que a coalescência de dados é uma área crítica que pode ser otimizada para melhorar ainda mais o desempenho do *kernel* `LBM_update` na arquitetura GH200. Estratégias futuras devem focar na otimização de acesso à memória para maximizar a utilização do cache e reduzir a latência de memória.

Este trabalho avaliou que os gastos relativos ao tempo de sincronização entre os *kernels* do padrão A-A para a GH200 não aumentaram em relação a V100, ou seja não aumentam com maior quantidade de memória e com um sistema de transferência de dados de memória mais eficiente.

Este estudo abre várias oportunidades para futuras pesquisas com simulações multifásicas baseadas no método Lattice Boltzmann (LBM) em arquiteturas de alto desempenho. Entre os próximos passos, pretende-se implementar estratégias de coalescência de dados para otimizar o acesso à memória, com base nas análises de *cache miss*, especialmente na arquitetura GH200, o que pode reduzir a latência e melhorar o desempenho do simulador.

A avaliação de novas arquiteturas de GPU e CPU, incluindo capacidades heterogêneas, é promissora para melhorar o desempenho e a eficiência energética das simulações LBM. Também será importante explorar o paralelismo e a escalabilidade em sistemas distribuídos, o que pode otimizar o tempo de execução para simulações de grande

escala usando redes de alta velocidade. Por fim, a aplicação do simulador em cenários reais, como na recuperação de petróleo e no transporte de contaminantes, permitirá validar sua eficácia e identificar áreas de melhoria.

7. Agradecimento

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento -001.

Os autores agradecem à Petrobras pelo suporte técnico fornecido, o qual foi fundamental para a realização desta pesquisa.

Os autores agradecem ao Laboratório Nacional de Computação Científica (LNCC/MCTI, Brasil) por fornecer os recursos de HPC do supercomputador SDumont, os quais contribuíram para os resultados de pesquisa apresentados neste artigo. URL:http://sdumont.lncc.br.

Os autores agradecem à NVIDIA por disponibilizar a plataforma com seus recursos de hardware, que foram essenciais para a realização dos experimentos nesta pesquisa.

Referências

Bailey, P., Myre, J., Walsh, S. D., Lilja, D. J., and Saar, M. O. (2009). Accelerating lattice boltzmann fluid flow simulations using graphics processors. In *2009 international conference on parallel processing*, pages 550–557. IEEE.

Chen, S., Doolen, G., and Eggert, K. (1994). Lattice boltzmann versatile tool for multiphase, fluid dynamics and other complicated flows. *Los Alamos Science*, 20:100–111.

Herschlag, G., Lee, S., Vetter, J. S., and Randles, A. (2018). Gpu data access on complex geometries for d3q19 lattice boltzmann method. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 825–834. IEEE.

Huang, H., Sukop, M., and Lu, X. (2015). Multiphase lattice boltzmann methods: Theory and application. .

Mattila, K., Hyväluoma, J., Rossi, T., Aspnäs, M., and Westerholm, J. (2007). An efficient swap algorithm for the lattice boltzmann method. *Computer Physics Communications*, 176(3):200–210.

Rigon, P. H., Schussler, B. S., Künas, C. A., Lorenzon, A. F., Carissimi, A., and Navaux, P. O. (2024). Otimizando a implementaçao multi-gpu do método fletcher através da paralelizaçao eficiente na computaçao e comunicaçao de dados. In *Anais da XXIV Escola Regional de Alto Desempenho da Região Sul*, pages 5–8. SBC.