Implementação e Avaliação de Políticas de Escalonamento de Warps no Vortex, uma GPGPU de Código Aberto

Samuel A. O. Magalhães[1], **Poliana A. C. Oliveira** [1], **Renan A. Marks** [2]

[1]Departamento de Computação – Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) – Belo Horizonte, MG – Brasil

[2]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS) Campo Grande, MS – Brasil

samuel@aluno.cefetmg.br, poliana@cefetmg.br, renan.marks@ufms.br

Abstract. *Despite the popularity of general-purpose GPUs, proprietary architectures limit direct academic experimentation, motivating the use of open-source projects such as Vortex, a GPGPU based on the RISC-V instruction set. This work extends Vortex with two additional warp scheduling strategies:* `PTA`*, proposed in this work, which prioritizes warps with a larger number of active threads; and* `GTO`*, already established in the literature, which greedily favors active warps until they are stalled. Simulation results using the Rodinia Benchmark reveal that* `GTO` *and* `PTA` *can reduce execution cycles by up to 26% and 21.5%, respectively, compared to Vortex's native Round Robin policy.*

Resumo. *Apesar da popularidade das GPUs de propósito geral, arquiteturas proprietárias limitam experimentações acadêmicas, motivando o uso de projetos em código aberto como o Vortex, uma GPGPU baseada no conjunto de instruções RISC-V. Este trabalho estende o Vortex com duas estratégias adicionais de escalonamento de warps:* `PTA`*, proposta neste trabalho e que prioriza warps com maior número de threads ativas; e* `GTO`*, política já conhecida na literatura, que favorece de forma gananciosa warps ativos até serem bloqueados. Os resultados das simulações com cargas do Rodinia Benchmark revelaram que o* `GTO` *e* `PTA` *podem alcançar uma redução de ciclos de até 26% e 21,5%, respectivamente, quando comparado à política nativa do Vortex, Round Robin.*

1. Introdução

A busca pela velocidade na renderização gráfica em tempo real, intensificada pelo avanço de técnicas de Computação Paralela e vetorial, fomentou o desenvolvimento e a popularidade das GPUs (*Graphics Processing Units*). Inicialmente, as GPUs foram projetadas como co-processadores na renderização de componentes gráficos em aplicações que demandavam alta interatividade e baixo tempo de resposta. Com o tempo, essas unidades foram adaptadas para cenários computacionais mais amplos, passando a ser conhecidas como GPGPUs (*General Purpose Graphics Processing Units*), principalmente por atuarem como aceleradores em demandas como Aprendizado de Máquina, Inteligência Artificial, simulações científicas e demais aplicações que exigem um alto índice de Computação Paralela [Aamodt et al. 2018, Peddie 2022].

As GPUs possuem uma arquitetura capaz de executar simultaneamente grandes volumes de dados favorecendo aplicações com alta demanda de paralelismo [Peddie

2022]. Para organizar a execução paralela, os processadores gráficos agrupam *threads* em unidades lógicas chamadas *warps*, cujas *threads* executam a mesma instrução de forma síncrona e são escalonadas a partir de um algoritmo bem definido. O método responsável pelo escalonamento dos *warps* influencia diretamente o desempenho da aplicação, podendo minimizar *stalls* por sincronização, melhorar a localidade dos dados e otimizar a utilização dos recursos computacionais [Narasiman et al. 2011].

Apesar do protagonismo comercial de arquiteturas proprietárias, como as da AMD e NVIDIA, suas especificações fechadas limitam a condução de pesquisas acadêmicas que demandam experimentação direta sobre a microarquitetura de uma GPU [Khairy et al. 2020]. Nesse cenário, iniciativas como o Vortex [Tine et al. 2021] — uma GPU de código aberto baseada no conjunto de instruções RISC-V — surgiram como alternativa para a popularização de experimentos em GPUs. O acesso irrestrito à arquitetura permite simulações em nível de ciclo com elevado grau de controle, possibilitando o estudo e a implementação de novas políticas, alterações microarquiteturais e análise de desempenho em ambientes altamente reprodutíveis.

O Vortex disponibiliza, por padrão, apenas uma política simples de escalonamento que realiza o revezamento circular (*Round Robin*) dos *warps*. No entanto, o trabalho de [Elsabbagh et al. 2019] aponta que essa abordagem pode limitar o desempenho em configurações com maior número de *warps* e *threads*, e destaca que políticas de escalonamento mais sofisticadas poderiam melhorar a latência de memória. Diante desse contexto, este trabalho tem como objetivo implementar e avaliar, na arquitetura Vortex, duas novas políticas de escalonamento de *warps*: *Greedy Then Oldest* (`GTO`), que já é uma estratégia conhecida na literatura, e Prioridade por *Threads* Ativas (`PTA`), que foi concebida e proposta por esse trabalho.

Como contribuição, este estudo apresenta uma nova proposta de política de escalonamento (`PTA`), a implementação dos algoritmos `GTO` e `PTA` de forma parametrizável, e os disponibiliza[1] no ecossistema de código aberto do Vortex, promovendo sua expansão e incentivando novas pesquisas sobre estratégias de gerenciamento de *warps* em pipelines SIMT. Além disso, os experimentos de simulação realizados com o Rodinia Benchmark [Che et al. 2009] apontam melhorias de desempenho em termos de ciclos de execução e latência de acesso à memória em aplicações específicas indicando uma direção para o desenvolvimento de novas abordagens no futuro.

2. Fundamentação Teórica

2.1. GPU

Um processador gráfico é um modelo de *hardware* projetado para processar grandes volumes de dados de forma massivamente paralela. Diferentemente das CPUs (*Central Processing Units*), as GPUs minimizam a lógica de controle e fluxo de execução, dedicando a maior parte da área do *die* (substrato semicondutor onde os circuitos eletrônicos são implementados) a unidades de execução aritmética. Essa organização arquitetural favorece aplicações que operam sobre grandes conjuntos de dados com baixa dependência entre as operações, permitindo que múltiplas tarefas sejam executadas simultaneamente. Exemplos típicos incluem: o processamento de imagens, onde cada pixel pode ser tra-

[1] https://github.com/SamuelOliveira14/vortex

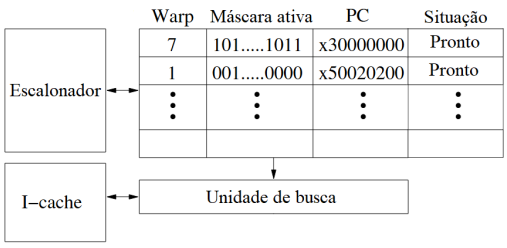
tado de forma independente; e o treinamento de Redes Neurais, que envolve operações vetoriais e matriciais com alto grau de paralelismo [Aamodt et al. 2018].

Para viabilizar o processamento paralelo, as GPUs adotam o modelo *Single Instruction, Multiple Threads* (SIMT), que permite a execução simultânea de uma mesma instrução em um conjunto diverso de dados independentes. Esse modelo potencializa aplicações que se beneficiam do paralelismo em nível de dados (DLP - *Data-Level Parallelism*) ao agrupar um conjunto de *threads* em entidades denominadas *warps*. Cada *thread* representa um fluxo de execução vinculado a uma seção específica do conjunto de dados. Os *warps*, por sua vez, são compostos por um número fixo de *threads* e são atribuídos dinamicamente aos núcleos da GPU para execução no pipeline SIMT [Hennessy and Patterson 2017]. Todas as *threads* de um *warp* processam a mesma instrução simultaneamente, e a ordem de execução de *warps* nos núcleos é definida de acordo com uma técnica de escalonamento projetada para aquele determinado *hardware*.

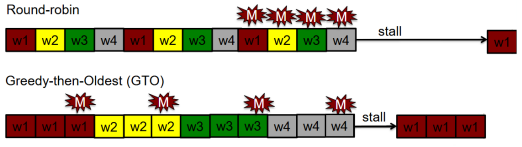
Assim, o *throughput* de uma GPU é medido com base na execução simultânea de um conjunto fixo de *threads* (*warps*) em um núcleo, sendo esse paralelismo expandido pela distribuição assíncrona de *warps* entre os diversos núcleos do *chip*. Durante a execução SIMT, um conjunto de *threads* é definido e o escalonador verifica quais *warps* possuem sua próxima instrução livre de dependências, consultando estruturas de gerenciamento específicas como o *scoreboard*.

2.2. Escalonamento de *warps*

O escalonamento de *warps* em GPUs é o primeiro passo durante o fluxo de execução do pipeline SIMT [Lakshminarayana and Kim 2010]. O escalonador, através de uma política bem definida, é responsável por selecionar quais *warps* estão prontos para emitir instruções a cada ciclo, escolhendo entre os conjuntos disponíveis a partir de uma tabela que possui registros de gerência sobre cada *warp* [Volkov 2016]. Nesse estágio, o escalonador mantém um controle sobre quais *warps* possuem instruções prontas para emissão. Caso a próxima instrução de um *warp* dependa de dados ainda não disponíveis ou esteja aguardando o retorno de uma instrução de acesso à memória, por exemplo, ela é temporariamente suspensa [Volkov 2016]. Quando um *warp* não possui dependências de dados ou não está em *stall*, ele é considerado pronto para execução. A Figura 1a ilustra um exemplo da tabela de gerenciamento de *warps* usada pelo escalonador.



(a) Tabela de gerência de *warps*. Adaptado de [Narasiman et al. 2011]



(b) *Round Robin* e *Greedy Then Oldest*. Adaptado de [Tine 2021]

Figura 1: Gerência e diferenças entre as políticas de escalonamento de *warps*.

A escolha de qual *warp* será escalonado pode seguir diferentes políticas. O algoritmo *Round Robin* (RR), por exemplo, distribui os *warps* de maneira justa, selecionando-os de forma circular a cada novo ciclo. Embora simples, essa abordagem pode provocar

gargalos quando múltiplos *warps* chegam simultaneamente em pontos de alta latência, como ilustrado pela Figura 1b. No exemplo, os *warps* W1, W2, W3 e W4 são escalonados sequencialmente até alcançarem instruções de acesso à memória, representadas pela letra "M". Como todos os *warps* ativos atingem esse ponto praticamente ao mesmo tempo, o *pipeline* entra em *stall* (ociosidade), pois nenhum outro *warp* possui instruções prontas para execução até que os acessos à memória sejam concluídos.

A política `GTO` (*Greedy Then Oldest*), por outro lado, prioriza a execução contínua de um mesmo *warp* enquanto ele estiver apto a emitir instruções. Quando não for possível continuar a execução, seja por dependências de dados ou de acesso à memória, o escalonador escolhe o *warp* mais antigo para ser priorizado. Na Figura 1b, observa-se que W1 executa continuamente até atingir uma instrução de acesso à memória. Em seguida, W2 assume, e assim por diante. Essa ordenação faz com que, no momento em que W4 atinge o ponto de *stall*, as dependências de W1 já estejam prestes a serem resolvidas, permitindo seu reescalonamento quase imediato. Dessa forma, o `GTO` consegue sobrepor melhor os tempos de espera com execução útil (ocultação de latências), reduzindo a ociosidade do *pipeline* em comparação ao *Round Robin* [Lee et al. 2014].

2.3. Vortex GPGPU

Embora muito utilizadas comercialmente, arquiteturas de GPU como as da AMD e NVIDIA são implementações proprietárias, o que dificulta pesquisas que exploram modificações na arquitetura de processadores gráficos modernos [Khairy et al. 2020]. Implementações como o Vortex [Tine et al. 2021] proporcionam uma GPU de código aberto baseada em RISC-V. O Vortex propõe um processador subdividido em quatro entidades: *Processor*, *Cluster*, *Socket* e *Core*. O *Processor* (processador) é formado por um conjunto de *clusters*, que podem compartilhar uma cache L3 para otimizar o acesso à memória. Cada *cluster* pode incluir uma cache L2 opcional e é composto por múltiplos *sockets*, os quais agrupam *cores* que possuem caches individuais de dados e instruções.

Durante o estágio de *schedule*, o Vortex utiliza a política de gerenciamento de *warps* baseada no algoritmo *Round Robin* para realizar o revezamento de *warps* ativos. Como uma GPU de código aberto, o Vortex possibilita experimentação e desenvolvimento de novas abordagens em arquiteturas GPGPU sem as limitações impostas por soluções proprietárias. Restrita a um único algoritmo de escalonamento, esta arquitetura abre margens para pesquisas voltadas à implementação de diferentes políticas de *warp scheduling*, junto com a avaliação dessas estratégias no desempenho e na eficiência computacional de arquiteturas baseadas em RISC-V.

3. Trabalhos Relacionados

O trabalho de [Narasiman et al. 2011] estimula uma estratégia que combina a arquitetura de *Large Warp Microarchitecture* com a política de escalonamento *Two-Level Warp Scheduling* (TLWS), que reduz ciclos ociosos ao segregar os *warps* de um núcleo em subgrupos associando diferentes prioridades de revezamento para cada conjunto. Os *warps* dentro de um mesmo subgrupo são escalonados via *round robin*. A combinação destas abordagens mostrou um ganho médio de 19,1% de IPC (instruções por ciclo) quando comparado a outras políticas de escalonamento.

O estudo de [Lakshminarayana and Kim 2010] investiga como diferentes políticas de escalonamento de *warps* impactam o desempenho de GPUs. A partir da quantidade de

instruções em cada *warp*, os autores classificam aplicações em simétricas e assimétricas e comparam políticas classificadas como justas (*Round Robin*, por exemplo) às estratégias *Oracle-All*, *Oracle-Bar* e *Oracle-Mem-Bar*, que são baseadas nas distâncias entre as instruções pendentes em cada *warp*. A ideia central é priorizar *warps* que possuem mais instruções até a finalização da execução. Os autores concluiram que políticas justas são benéficas para aplicações simétricas, enquanto aplicações assimétricas exigem abordagens mais flexíveis, planejando explorá-las em trabalhos futuros.

A técnica *Earliest Load First* (ELF), elaborada por [Park et al. 2015], introduz um escalonamento coordenado de *warps* e busca de instruções para maximizar o paralelismo no nível de memória (MLP). O ELF utiliza técnicas em nível de compilação e de hardware para priorizar *warps* com menos instruções até a próxima operação de desvio ou memória e emprega um mecanismo de reexecução de acessos à memória para reduzir conflitos na cache. Com essas otimizações, o ELF melhora o desempenho em 4,1% sobre `GTO` e 11,9% quando combinado com outras técnicas.

Embora o Vortex adote um escalonador de *warps* simples e de granularidade fina (*Round Robin*), o trabalho [Elsabbagh et al. 2019] mostra que configurações com mais *threads* por *warp* tendem a reduzir o número de ciclos de execução, mas impõem maior demanda por registradores e unidades funcionais. Neste contexto, a contribuição deste trabalho está na implementação e avaliação de novas políticas de gerência de *warps* na arquitetura do Vortex, além de ampliar as possibilidades de simulação ao permitir a escolha paramétrica de diferentes algoritmos nesta plataforma.

4. Algoritmos implementados no Vortex

Os algoritmos `RR` e `GTO` já são consolidados na literatura. O primeiro está disponível na implementação nativa do Vortex, já o segundo se destaca por ser amplamente adotado e por operar de maneira restrita à camada SIMT, o que facilita sua adaptação ao Vortex, sem demandar modificações profundas na arquitetura (cache, núcleo, *pipeline*, etc.). Já o algoritmo Prioridade por *Threads* Ativas (`PTA`) foi concebido no escopo deste trabalho como uma alternativa que busca explorar de forma mais eficiente os recursos do *pipeline* SIMD, ao priorizar *warps* com maior número de *threads* ativas. Além do apelo conceitual, o `PTA` mostrou-se tecnicamente viável, uma vez que utiliza apenas informações já presentes nas estruturas internas do Vortex, evitando a necessidade de alterações estruturais na microarquitetura da GPU.

4.1. Round Robin

A política *Round Robin* (`RR`) é uma abordagem que distribui a execução de forma equitativa entre todos os *warps* disponíveis. Cada *warp* recebe uma fatia de tempo igual para execução, e o escalonador alterna entre os *warps* de forma circular. Essa política evita a inanição de *warps*, mas pode não ser eficiente em cenários em que os conjuntos chegam em uma instrução de alta latência simultaneamente (Figura 1b), minimizando a capacidade do pipeline de camuflar os *stalls* ao executar outros *warps* [Narasiman et al. 2011].

4.2. Greedy Then Oldest (`GTO`)

O algoritmo *Greedy Then Oldest* é uma abordagem que prioriza a execução de *warps* seguindo uma lógica ”gulosa”, ou seja, priorizando um mesmo *warp* até que ele não esteja mais disponível. Conforme mostra o Algoritmo 1, há duas etapas principais:

- **Fase *greedy* (gulosa)**: o escalonador prioriza a execução consecutiva do mesmo *warp* enquanto ele permanecer no estado pronto. Essa estratégia também visa aumentar a localidade espacial e temporal dos dados, uma vez que o mesmo conjunto de *threads* estará constantemente sendo alocado.
- **Fase *oldest* (mais antigo)**: se o *warp* atual sofrer um *stall* ou precisar ser inativado (por exemplo, devido a uma dependência de dados ou um *cache miss*), o escalonador seleciona, dentre todos os *warps* prontos, aquele que está esperando há mais tempo na fila, a fim de evitar inanição.

Para possibilitar a execução do `GTO`, será necessário realizar uma modificação na arquitetura para adicionar campos que irão armazenar o tempo (ou idade) de cada *warp*, possibilitando a busca pela entidade mais antiga do conjunto.

Algoritmo 1: Greedy Then Oldest (`GTO`)

```
Input: Lista de warps prontos para execução
Input: Último warp escolhido
incrementarIdade(warpsProntos);
if warpEscolhido.stalled then
    warpEscolhido.idade ← 0;
    warpEscolhido ← maisAntigo(warpsProntos);
end
return warpEscolhido
```

4.3. Prioridade por *Threads* Ativas (`PTA`)

Esta política visa maximizar a utilização das unidades de execução da etapa SIMD a cada ciclo. A heurística central é que executar um *warp* com mais *threads* ativas resulta em mais trabalho útil por instrução. O Algoritmo 2 utiliza a máscara de *threads* para identificar qual *warp* possui a maior quantidade de *threads* úteis, priorizando-o. Diferentemente do `GTO`, esta política não necessita de incrementos em relação a novas estruturas para armazenamento, já que a arquitetura do Vortex permite acessar as informações presentes na pilha SIMT durante o escalonamento.

Algoritmo 2: *Prioridade por Threads Ativas* (PTA)

```
Input: Lista de warps prontos para execução (warpsProntos)
maxThreadsAtivas ← 0
foreach warp in warpsProntos do
    numThreads ← contarBitsAtivos(warp);
    if numThreads > maxThreadsAtivas then
        maxThreadsAtivas ← numThreads;
        warpEscolhido ← warp;
    end
end
return warpEscolhido;
```

5. Metodologia

Este trabalho foi estruturado em duas etapas principais: a modificação do simulador para incorporar as políticas de escalonamento propostas e a condução dos experimentos para análise comparativa de desempenho. A primeira etapa consistiu na implementação das políticas `GTO` e `PTA` no código-fonte da GPU. O ambiente de simulação utilizado foi o `SimX` [Elsabbagh et al. 2020], um software desenvolvido em C++ capaz de realizar simulações sobre a arquitetura do Vortex. A implementação ocorreu em dois passos:

(a) **Modificação do processo de *build* do simulador**: foram realizadas modificações nos módulos de configuração do `SimX`, permitindo que cada política de escalonamento fosse escolhida de forma parametrizada no momento da realização dos experimentos. Esta etapa não só viabiliza os experimentos planejados, mas também permite que novas políticas possam ser incorporadas ao simulador sem grandes refatorações na base do código do `SimX`.

(b) **Implementação dos algoritmos**: as políticas `GTO` e `PTA` foram incorporadas ao código-fonte do `SimX`, seguindo as lógicas de escalonamento vistas na Seção 4.

Para os experimentos de avaliação de desempenho foi selecionado um subconjunto de cargas de trabalho do Rodinia Benchmark [Che et al. 2009,Tine et al. 2021]. O Rodinia é formado por programas construídos para a avaliação do desempenho de processadores voltados à computação paralelamente intensiva. Os programas selecionados foram *conv3*, *psort*, *psum*, *saxpy*, *sfilter*, *sgemm*, *transpose* e *vecadd*. Estas cargas de trabalho são destinadas à avaliação do comportamento da GPGPU e, por esse motivo, possuem focos distintos, como, por exemplo, alguns são computacionalmente intensivos (sgemm, vecadd e sfilter), enquanto outros dependem de uma boa gerência dos acessos à memória ou de um bom controle de divergências de controle como o *sxapy* [Tine et al. 2021].

Os experimentos contaram com três configurações distintas: (1) `8W8T` – 8 *warps* e 8 *threads*; (2) `16W16T` – 16 *warps* e 16 *threads*; (3) `32W32T` – 32 *warps* e 32 *threads*. Cada configuração varia o número de *warps* por núcleo e o tamanho de cada *warp* (quantidade de *threads*), mantendo o número total de núcleos fixado em 1. Adicionalmente, para cada uma dessas configurações arquiteturais, as aplicações foram testadas com diferentes tamanhos de entrada: 64, 128, 256 e 512 elementos. Essa variação no volume de dados permite avaliar a escalabilidade das políticas em relação ao tamanho da carga de trabalho.

A quantidade de *warps* e *threads* foi definida com base em experimentos de [Elsabbagh et al. 2019] e a escolha de manter o número de núcleos fixado em 1 está alinhada ao objetivo de investigar o impacto do escalonador no pipeline da GPU, isolando variáveis relacionadas ao gerenciamento de múltiplos núcleos. Deste modo, a avaliação do desempenho das políticas de escalonamento foi baseada nas seguintes métricas: *Total de ciclos de execução*: número total de ciclos necessários para a conclusão da aplicação; *Latência acumulada de acesso à memória*: corresponde à soma dos ciclos em que os *warps* permaneceram em estado de espera (*stall*) devido a operações de acesso à memória (*loads*).

6. Resultados

Os resultados são apresentados em dois blocos principais: *desempenho dos algoritmos com foco na quantidade de ciclos totais* (Seção 6.1) e *análise das latências de acesso à memória* (Seção 6.2). Em ambos os casos, discute-se a influência das variações arquiteturais sobre o comportamento das cargas de trabalho e as implicações sobre o projeto de

escalonadores em arquiteturas de GPGPU. Nas comparações apresentadas, a política `RR` é utilizada como referência. Logo, os resultados serão apresentados em formato de ganho ou perda em relação a essa política, que era a única disponível no Vortex até então.

6.1. Total de Ciclos de Execução

Os resultados dos experimentos em termos do total de ciclos de execução são mostrados pela Tabela 2. Para facilitar a interpretação visual, foi adotado um esquema de formatação condicional com cores baseado em intervalos de variação percentual, conforme mostrado na Tabela 1. A Figura 2 confronta as políticas `GTO` e `PTA` no quesito quantidade de experimentos que obtiveram redução no número de ciclos em relação ao algoritmo base.

Intervalo	Formatação aplicada
$x \geq 15\%$	Vermelho escuro
$15\% > x \geq 7.5\%$	Vermelho médio
$7.5\% > x \geq 2.5\%$	Vermelho claro
$2.5\% > x > 0\%$	Amarelo claro
$x = 0\%$	Branco
$0\% > x > -2.5\%$	Verde claro
$-2.5\% \geq x > -7.5\%$	Verde médio
$-7.5\% \geq x > -15\%$	Verde escuro
$x \leq -15\%$	Verde intenso

Tabela 1: Esquema de formatação condicional aplicado à Tabela 2

Programa	Entrada	GTO			PTA		
		8W8T	16W16T	32W32T	8W8T	16W16T	32W32T
conv3	64	-0,1%	-4,8%	-5,1%	-0,3%	-2,0%	-0,6%
	128	-0,6%	-0,7%	-2,5%	-0,4%	-1,8%	-0,1%
	256	-1,4%	0,3%	-3,1%	-1,2%	-1,1%	-0,5%
	512	-0,7%	5,3%	-2,0%	0,2%	-1,2%	-0,4%
psort	64	-3,9%	-3,2%	-1,9%	-2,0%	-0,4%	-0,2%
	128	-1,9%	-2,9%	-2,8%	-1,2%	0,3%	-1,2%
	256	-3,1%	-17,0%	-2,3%	-2,4%	-15,0%	-1,1%
	512	-2,8%	-26,4%	-14,7%	-2,3%	-21,5%	-14,1%
psum	64	-1,8%	-3,5%	-2,1%	-0,3%	-1,4%	-0,4%
	128	-2,5%	-2,2%	-0,9%	-0,4%	-0,5%	0,0%
	256	-1,7%	0,0%	-2,3%	-0,1%	-1,8%	-2,0%
	512	-2,9%	0,1%	0,5%	-2,1%	-2,1%	0,4%
saxpy	64	0,4%	-10,5%	-3,9%	2,4%	-3,6%	-1,8%
	128	1,2%	-8,3%	-3,7%	2,2%	-2,2%	-1,8%
	256	1,5%	-7,9%	-3,4%	3,1%	-2,6%	-1,0%
	512	1,8%	-11,8%	-2,7%	3,8%	-4,0%	-1,5%
sfilter	64	-3,4%	-8,3%	-3,1%	-0,9%	0,8%	0,2%
	128	-2,1%	-5,0%	-1,1%	2,1%	0,3%	-1,8%
	256	-3,7%	5,8%	-2,1%	0,4%	4,9%	1,4%
	512	-3,3%	9,2%	-1,2%	0,9%	3,5%	1,3%
sgemm	64	9,6%	20,0%	3,9%	5,7%	-0,2%	0,7%
	128	2,2%	-6,7%	-11,6%	1,1%	0,2%	-1,9%
	256	4,3%	8,9%	-21,0%	2,6%	-0,7%	0,3%
	512	5,2%	19,0%	-3,9%	2,8%	-0,1%	1,7%
transpose	64	0,4%	-8,1%	-3,9%	3,0%	-0,8%	-0,1%
	128	0,3%	-2,9%	-2,8%	2,4%	2,9%	0,6%
	256	0,1%	-7,8%	-1,1%	1,4%	-3,4%	-0,2%
	512	0,0%	-1,2%	-2,0%	2,2%	2,2%	-0,2%
vecadd	64	-0,2%	-7,5%	-2,7%	2,8%	-0,4%	0,0%
	128	-0,3%	-6,1%	-2,5%	1,9%	-0,2%	0,0%
	256	0,3%	-4,9%	-2,4%	1,7%	-0,3%	0,1%
	512	0,9%	-6,2%	-2,2%	1,9%	-0,9%	-0,1%

Tabela 2: Variação percentual da quantidade de ciclos das políticas GTO e PTA

Entre as duas políticas analisadas, GTO apresentou os resultados mais expressivos, o que é evidenciado pela redução na quantidade de ciclos em 71 dos 96 experimentos realizados (73,9%), sendo particularmente eficiente em cenários com maior paralelismo. O GTO se beneficia do aumento do número de *warps* e *threads* da GPU: nas configurações 32W32T, 93,7% dos testes mostraram ganhos em relação à política base (Figura 2). Para as configurações 16W16T e 8W8T, as proporções de valores que apresentaram redução na quantidade de ciclos foram de 71,9% e 56,2%, respectivamente. Além do número de resultados positivos, o GTO foi responsável pelos maiores ganhos absolutos observados, com variações percentuais de até -20%, como ocorre em *psort* e *sgemm*.

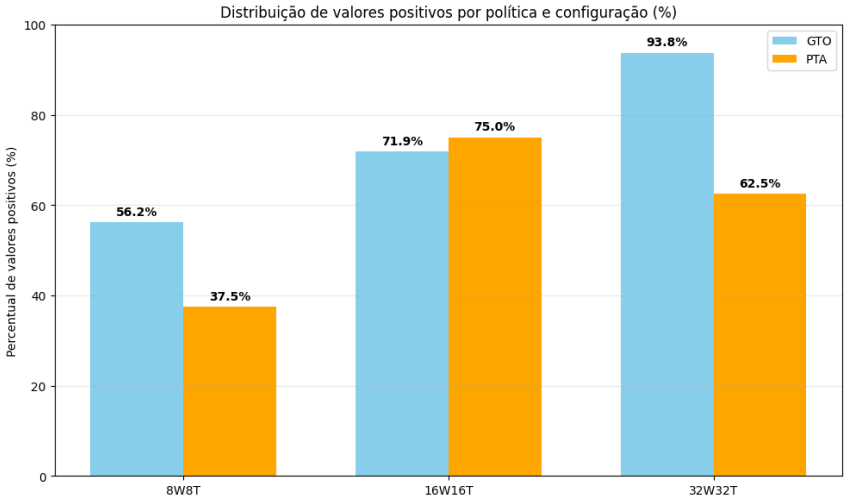


Figura 2: Comparação da quantidade de resultados positivos (redução do número de ciclos em relação ao algoritmo RR) entre GTO e PTA.

A política PTA, por sua vez, apresentou desempenho moderado, superando o RR em 56 dos 96 experimentos (58,3%), com destaque para a aplicação *psort* (redução em 21,5% dos ciclos de *clock*). Embora seja superior ao GTO em apenas 18 dos 96 experimentos (18,75%), a PTA foi melhor que o GTO em relação à quantidade de cenários positivos na arquitetura 16W16T e se manteve em um intervalo mais estável ao longo dos testes, com menor incidência de perdas significativas. O pior resultado foi observado na aplicação *sgemm*, com uma redução de desempenho de 5,7%. Em contrapartida, nessa carga, o GTO apresentou perdas mais acentuadas, com aumento até 20% de ciclos. Por fim, a Tabela 2 ainda indica que o tamanho da entrada não determina, de forma isolada, o comportamento das políticas de escalonamento. Em diversas situações, a variação no volume de dados não seguiu um padrão uniforme em relação ao aumento dos parâmetros do programa, sugerindo que o impacto do tamanho da carga depende da combinação entre a arquitetura, a política de escalonamento e as características específicas de cada aplicação.

6.2. Latência de acesso à memória

A Figura 3 apresenta a relação entre a latência de acesso à memória e o total de ciclos de execução para cada carga de trabalho. Na configuração 8W8T, com menor paralelismo, tanto a GTO quanto a PTA apresentam poucas variações na latência em comparação à RR. Nesse cenário, o impacto da latência é menos significativo, e as diferenças no total de ciclos são pequenas. GTO apresenta leve aumento na latência em algumas cargas, entre-

tanto, consegue reduzir os ciclos em aplicações como *psort* e *sfilter*. Por outro lado, a PTA mantém a latência próxima de RR, porém com pouca redução no número de ciclos.

À medida que o grau de paralelismo aumenta nas configurações 16W16T e 32W32T, as diferenças entre as políticas tornam-se mais evidentes. Em *conv3*, *sfilter* e *psort*, o GTO apresentou piora na latência de acesso à memória, mas, ainda assim, conseguiu entregar uma redução na quantidade de ciclos totais. Isso sugere que, para o GTO, a penalização na latência pode ser compensada por decisões de escalonamento que favorecem o aproveitamento dos ciclos de forma mais eficiente.

Para PTA, os resultados mostram que esta política apresenta uma boa redução da latência nas configurações de 32W32T. Porém, com exceção de *psort*, variações positivas não influenciaram significativamente na quantidade de ciclos totais, o que indica que podem existir outros fatores, como, por exemplo, a inanição de *warps* com poucas *threads* ativas, que corroboraram com a baixa alteração na quantidade de ciclos desta política.

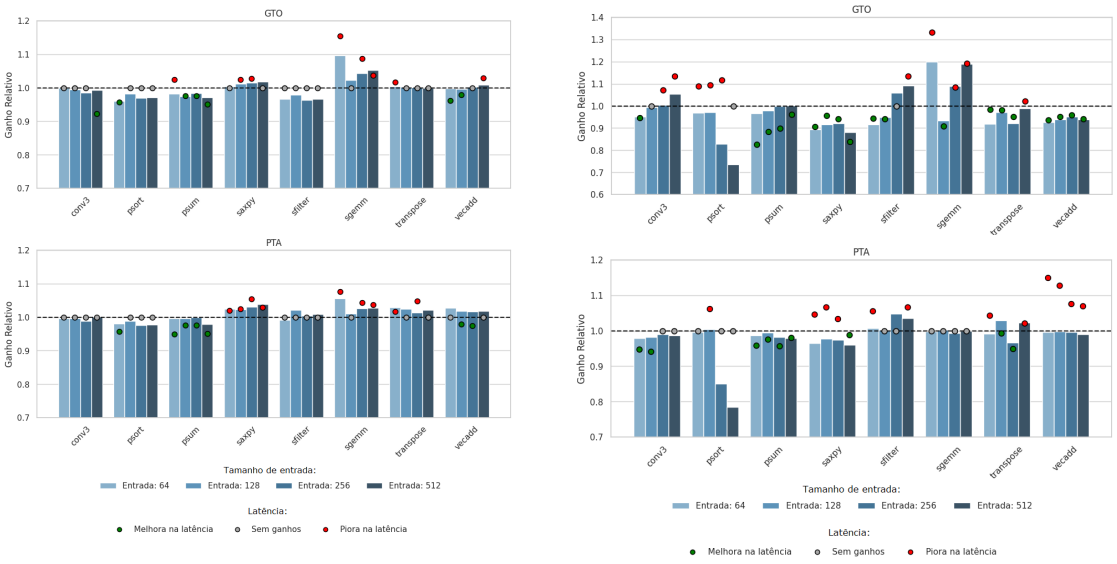
7. Conclusões

A análise dos resultados demonstrou que a política GTO oferece maior potencial de ganho quando submetida às cargas de trabalho do Rodinia, destacando-se pela capacidade de ocultar latências e reduzir o número total de ciclos em aplicações com alto grau de paralelismo. É importante destacar que o ganho de desempenho não está somente atrelado à redução da latência de acesso à memória, mas também à capacidade da política em mitigar seus efeitos sobre o fluxo de execução. Assim, políticas que conseguem esconder latências por meio de escalonamentos mais ajustados tendem a apresentar melhores resultados em configurações que aumentam o número de *warps* e de *threads* por *warp*.

A política PTA, embora mais simples, também demonstrou ganhos consistentes em relação ao *Round-Robin*, especialmente no que se refere à latência de acesso à memória. Em configurações com maior número de *warps* e *threads*, observou-se uma redução mais pronunciada da latência em comparação ao escalonador base. Além disso, embora não tenha alcançado os mesmos níveis de redução de ciclos de execução que o GTO, o PTA apresentou um comportamento mais estável, com desempenho mais equilibrado mesmo nos cenários em que o GTO foi significativamente menos eficiente.

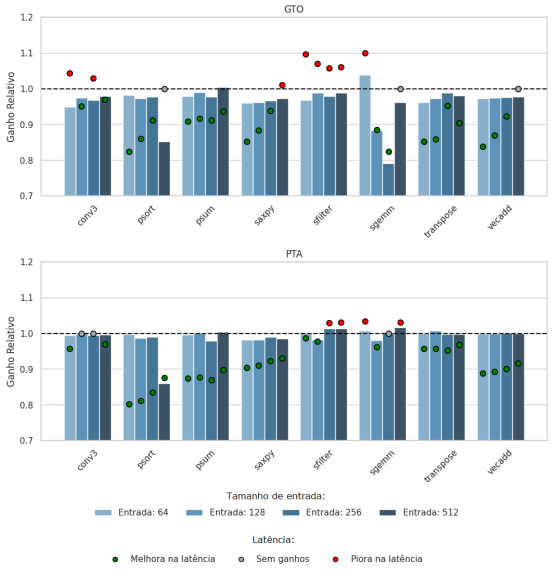
Como contribuição, este trabalho propôs a política de escalonamento de *warps* PTA e disponibilizou as implementações das políticas GTO e PTA na plataforma de código aberto Vortex, permitindo que novas pesquisas possam utilizá-las e/ou estendê-las. Além disso, ele estimula a implementação de outras políticas de escalonamento através das modificações realizadas no SimX, que possibilitam a parametrização de diferentes estratégias na arquitetura do simulador. Os resultados obtidos evidenciam que mesmo algoritmos simples podem impactar o desempenho da GPU.

Trabalhos futuros incluem a implementação de políticas de escalonamento mais sofisticadas na arquitetura do Vortex, bem como a realização de experimentos em configurações multi-núcleo, a fim de avaliar o impacto dos algoritmos em cenários de maior paralelismo. A síntese em *hardware* dos algoritmos propostos é outra direção relevante, permitindo analisar sua viabilidade prática e consumo energético. Por fim, novas pesquisas podem caracterizar as cargas de trabalho utilizadas, investigando a correlação entre as características das aplicações e o desempenho observado para cada política de escalonamento.



(a) 8W8T

(b) 16W16T



(c) 32W32T

Figura 3: Relação entre a latência de acesso à memória e quantidade total de ciclos para as configurações 8W8T, 16W16T e 32W32T. As barras indicam a variação percentual no total de ciclos (menor é melhor) e os marcadores mostram a variação da latência de acesso à memória (verde = redução, vermelho = aumento, cinza = sem alterações).

8. Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (Capes) – Código de Financiamento 001; e da Universidade Federal de Mato Grosso do Sul – UFMS/MEC – Brasil.

Referências

Aamodt, T. M., Fung, W. W. L., Rogers, T. G., and Martonosi, M. (2018). *General-purpose graphics processor architectures*. Morgan & Claypool Publishers. ISBN: 9781627056182.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)*, pages 44–54. Ieee.

Elsabbagh, F., Asgari, B., Kim, H., and Yalamanchili, S. (2019). Vortex RISC-V GPGPU System: Extending the ISA, Synthesizing the Microarchitecture, and Modeling the Software Stack.

Elsabbagh, F., Tine, B., Roshan, P., Lyons, E., Kim, E., Shim, D. E., Zhu, L., Lim, S. K., et al. (2020). Vortex: OpenCL Compatible RISC-V GPGPU. *arXiv preprint arXiv:2002.12151*.

Hennessy, J. L. and Patterson, D. A. (2017). Computer organization and design RISC-V edition: The hardware software interface.

Khairy, M., Shen, Z., Aamodt, T. M., and Rogers, T. G. (2020). Accel-Sim: An extensible simulation framework for validated GPU modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 473–486. IEEE.

Lakshminarayana, N. B. and Kim, H. (2010). Effect of instruction fetch and memory scheduling on GPU performance. In *Workshop on Language, Compiler, and Architecture Support for GPGPU*, volume 88.

Lee, M., Song, S., Moon, J., Kim, J., Seo, W., Cho, Y., and Ryu, S. (2014). Improving GPGPU resource utilization through alternative thread block scheduling. In *2014 IEEE 20th international symposium on high performance computer architecture (HPCA)*, pages 260–271. IEEE.

Narasiman, V., Shebanow, M., Lee, C. J., Miftakhutdinov, R., Mutlu, O., and Patt, Y. N. (2011). Improving GPU performance via large warps and two-level warp scheduling. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 308–317.

Park, J. J. K., Park, Y., and Mahlke, S. (2015). ELF: Maximizing memory-level parallelism for GPUs with coordinated warp and fetch scheduling. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12.

Peddie, J. (2022). *The History of the GPU - Steps to Invention*. Springer. ISBN 9783031109683.

Tine, B. (2021). Vortex microarchitecture. In *54th Annual IEEE/ACM International Symposium on Microarchitecture*. Association for Computing Machinery.

Tine, B., Yalamarthy, K. P., Elsabbagh, F., and Hyesoon, K. (2021). Vortex: Extending the RISC-V ISA for GPGPU and 3D-Graphics. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 754–766, New York, NY, USA. Association for Computing Machinery.

Volkov, V. (2016). *Understanding latency hiding on GPUs*. University of California, Berkeley.