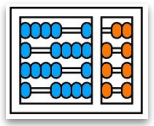
Otimização de *Checkpointing* em Aplicações de Modo Adjunto: Uma Abordagem Baseada em *Prefetching* e Compressão

Thiago J. M. Maltempi
maltempi@ic.unicamp.br

Orientador: Prof. Dr. Sandro Rigo
Co-orientador: Prof. Dr. Guido Araújo



Sobre este trabalho

→ Propõe acelerar checkpointing por meio de combinação de prefetching e compressão de dados, reduzindo tempo bloqueante de comunicação entre memória do host e memória da GPU

→Técnicas avaliadas com:

- Algoritmos de checkpointing: Revolve, zCut e Uniform
- Benchmark: Reverse Time Migration
- Dataset: Large, Marmousi, e Salt.

→ Código gerado foi encapsulado em uma biblioteca intitulada GPUZIP

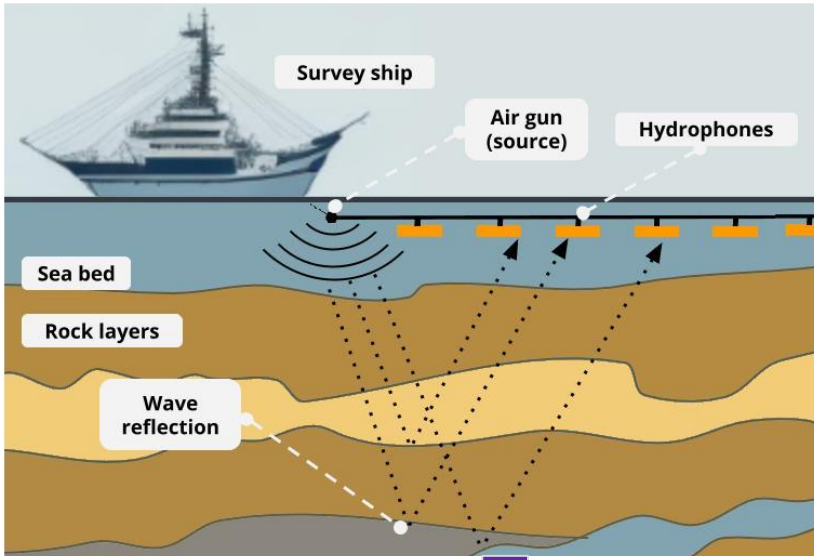
→ Speedups de até 5.1x para o algoritmo Revolve; 8.8x para zCut; e 5.8x para Uniform

SECTION

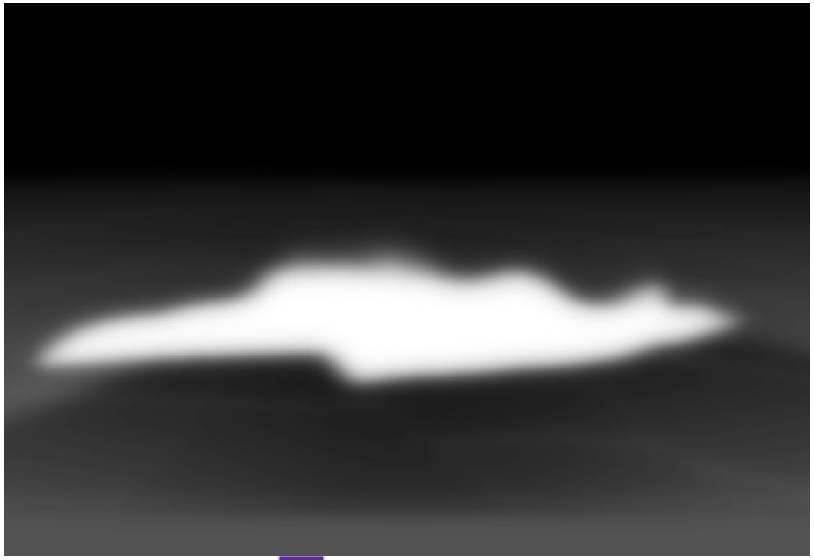
Background

Reverse Time Migration (RTM)

Seismic Survey
(Gera traços sísmicos)

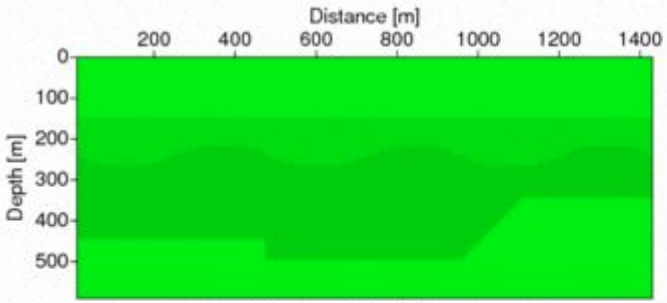


Modelo Velocidade
(Feito por geofísicos)

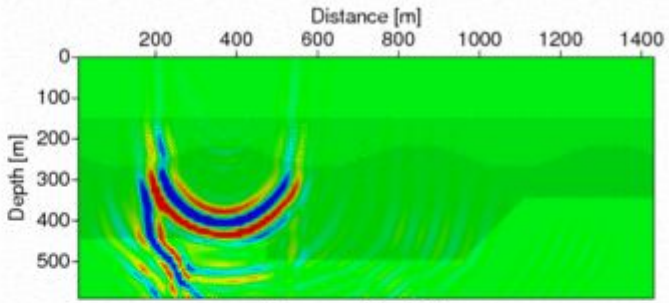


RTM

Reverse Time Migration (RTM)

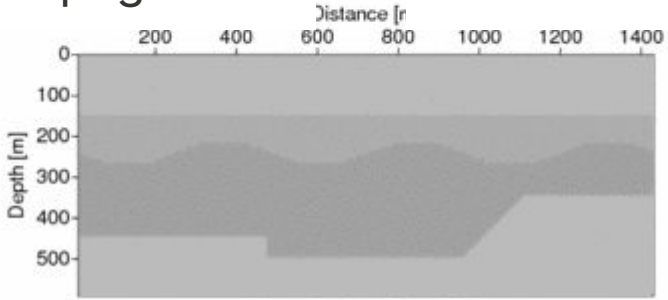


Forward propagation



Backward propagation

time= 10 ms



Correlation

Video credits: https://github.com/antongrin/rtm-migration-tutorial

Implementação *Naïve*

Algorithm 1 Naive forward and backward phases with cross-correlation. Algorithm complexity: $\mathcal{O}(timesteps^2)$.

1: Initialize bwdData;
2: **for** $tsBwd = timesteps - 1$; $tsBwd \geq 0$; $tsBwd = tsBwd - 1$ **do**
3: Initialize fwdData;
4: **for** $tsFwd = 1$; $tsFwd \leq tsBwd$; $tsFwd = tsFwd + 1$ **do**
5: ComputeForward(fwdData, tsFwd);
6: **end for**
7: ComputeBackward(bwdData, tsBwd);
8: Crosscorrelate(fwdData, bwdData);
8: Free(fwdData);
9: **end for**=0

Algoritmos de Checkpointing (Revolve, zCut, e Uniform)

1. Quantidade inicial de *snapshots*
2. Orquestram a aplicação
3. Rápida execução
4. Reutilização dos mesmos *snapshots* em um curto espaço de tempo
5. Algoritmos Determinísticos

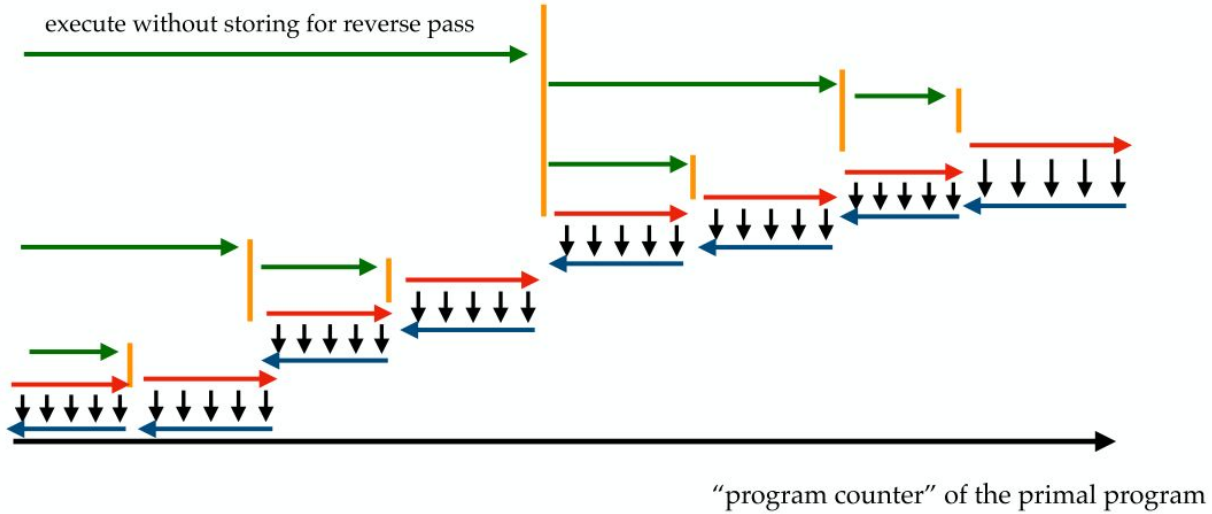
```
action = getAction(timestep)

while action != TERMINATE:
  if action == FORWARD:
    calc_forward(timestep)
  elif action == SAVE:
    save_checkpoint(timestep)
  elif action == RESTORE:
    restore_checkpoint(timestep)
  elif action == BACKWARD:
    calc_backward(timestep)

  action = getAction(timestep)
```

Bisection-based Checkpointing [Griewank 1992]

*assuming the memory / time use each step is O(1)

peak memory use: number of checkpoints: O(log(n))

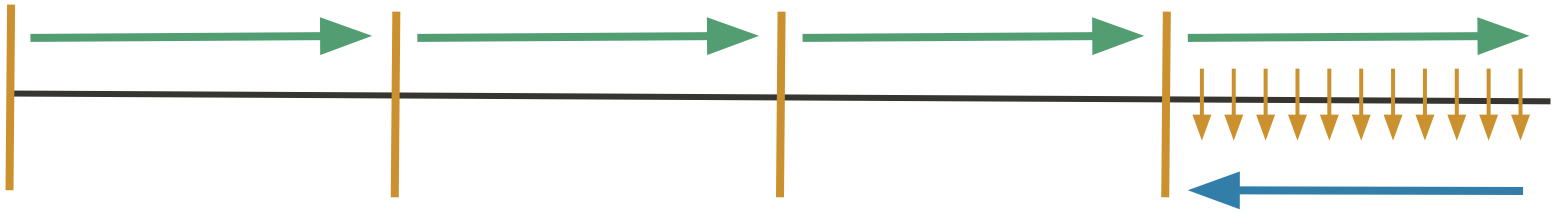
time complexity: sum of green lines + O(n) = O(nlog(n))



Créditos Imagem: UCSD CSE 291 Tzu-Mao Li
cseweb.ucsd.edu/~tzli/cse291/sp2024/lectures/checkpointing.pdf

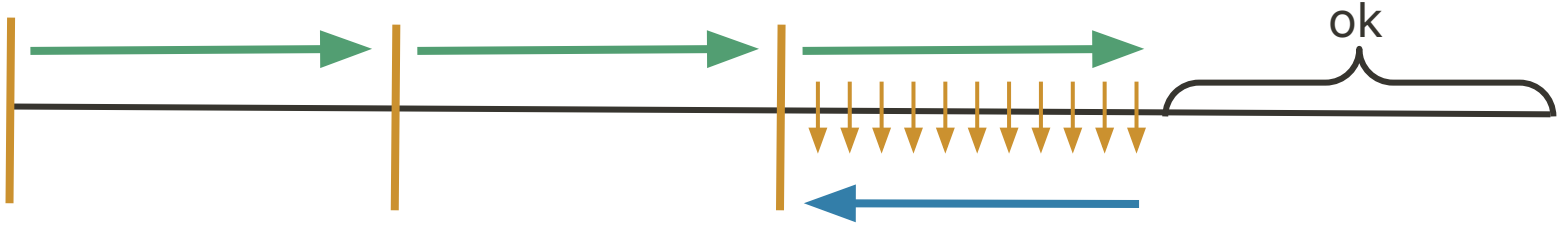
zCut

- Representa o problema computacional como grafos, onde **nós** são os *kernels* e as **arestas** mapeiam as dependências entre as tasks.
- Faz “**cortes**” e dentro destes “cortes” todos os timesteps são salvos



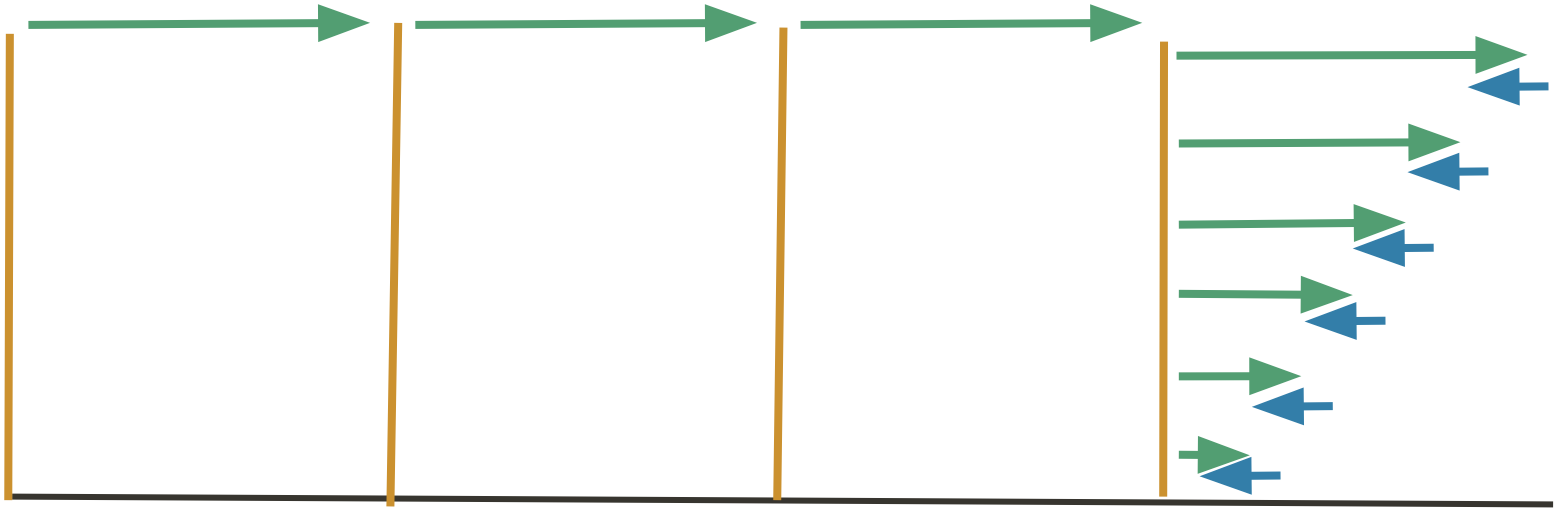
zCut

- Representa o problema computacional como grafos, onde **nós** são os *kernels* e as **arestas** mapeiam as dependências entre as tasks.
- Faz “**cortes**” e dentro destes “cortes” <u>todos</u> os timesteps são salvos



Uniform

- Número de snapshots é a entrada do algoritmo
- É definido um intervalo fixo quando cada *snapshot* será salvo
- Implementação *in-house*



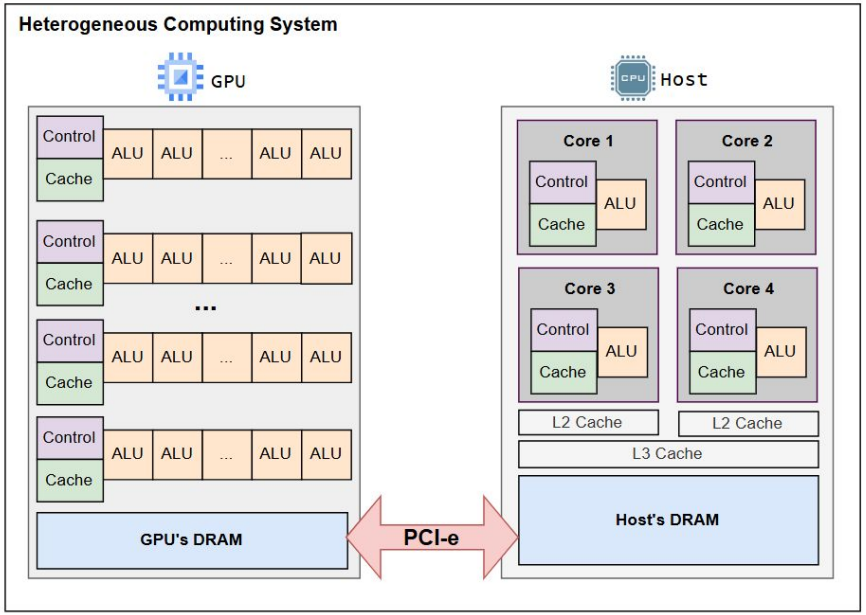
SECTION

Análise do problema

Cenário

- Awave-3D: implementação de Reverse Time Migration (RTM)
  - Kernels em GPU (NVIDIA)
  - Evita recomputação com a técnica de *checkpointing*
  - Memória necessária para armazenar os dados de checkpoint (Salt):
    - Revolve: 16.50 GB
    - zCut: 329.80 GB
    - Uniform: 329.40 GB

Dados de checkpoint estão na memória do host



Ogbon HPC:

NVIDIA Tesla V100 mem.: 32GB
Ogbon HPC Host: 384 GB

} 12x maior

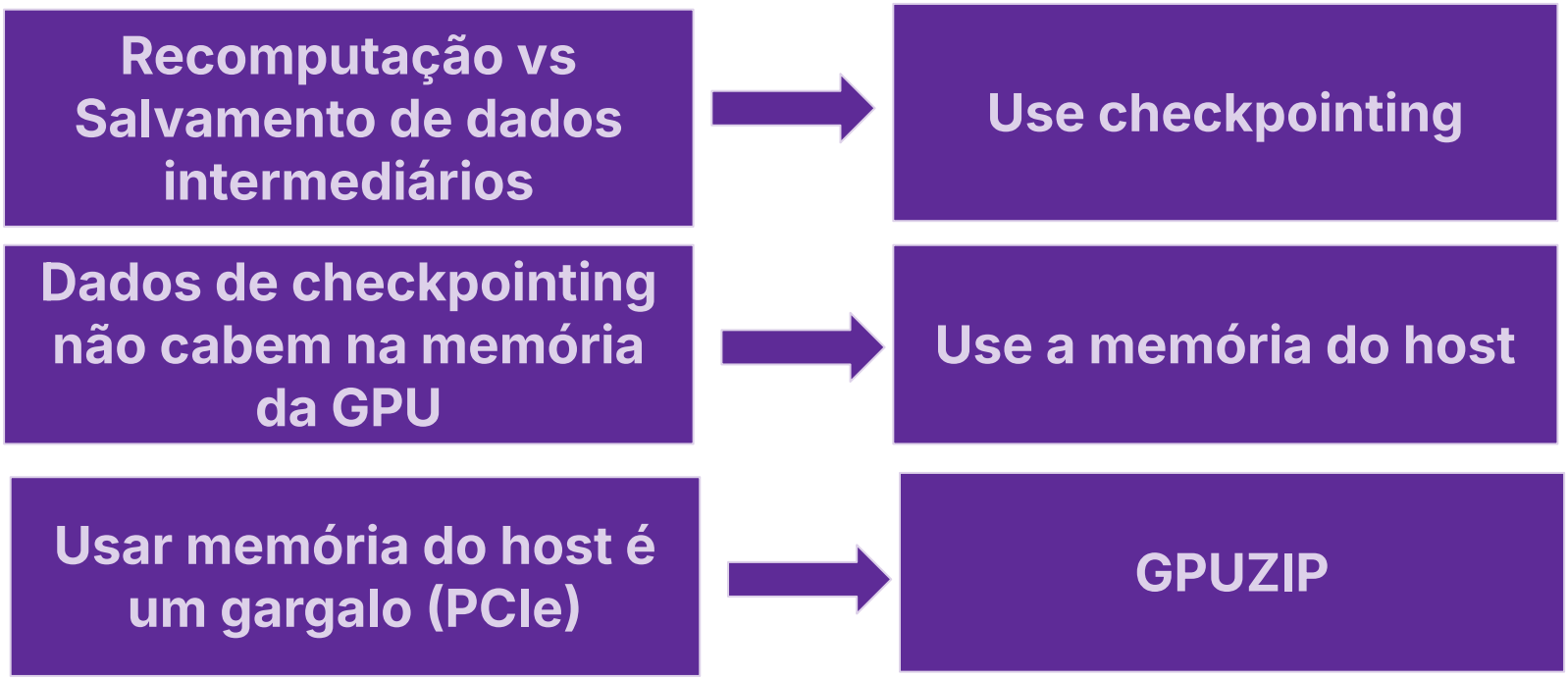
NVIDIA Tesla V100:

HBM2: 900 GB/s
PCI-e: 32 GB/s

} ~28x mais lento

A versão original do Awave-3D gasta 75% do seu tempo transferindo dados de checkpointing.

Para cada solução, um novo problema



Análise

- Os algoritmos tendem a acessar o mesmo snapshot múltiplas vezes em um curto espaço de tempo
    - Temporal Locality ⇒ Cache?
- Algoritmos determinísticos
    - *Hinting* ⇒ Prefetching?
- Os dados do *checkpoint* geralmente são compostos por grandes e tensores de ponto flutuante.
    - Compressão ⇒ Compressão com Perdas (Lossy)?

Objetivo principal

Acelerar a execução do Awave-3D através da redução de tempo bloqueante de transferência de dados GPU-Host causado por *Checkpointing*, explorando técnicas de *prefetching* e compressão.

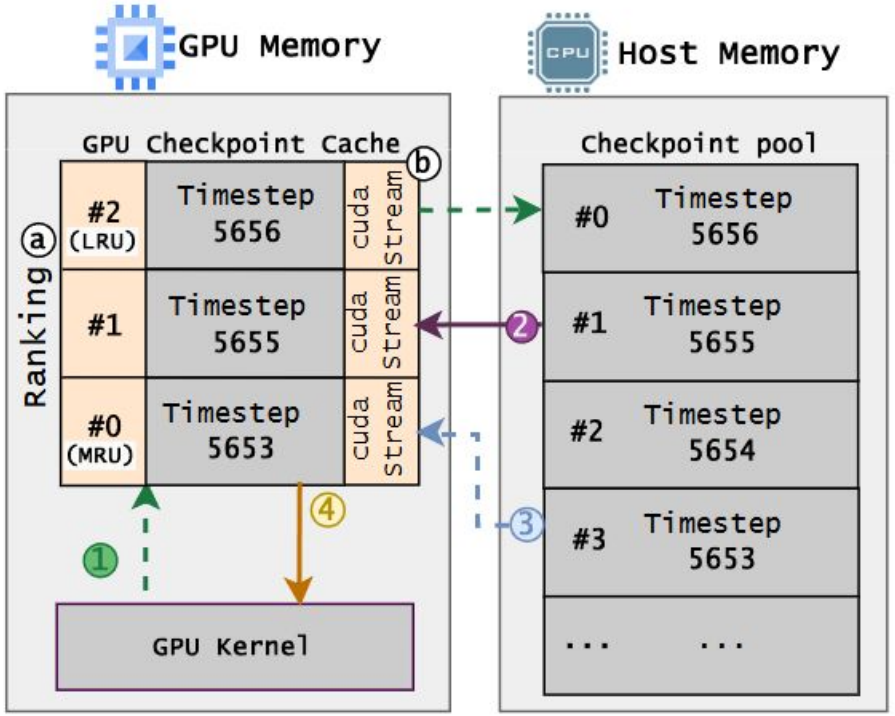
Perguntas de Pesquisa

- **RQ1: É possível Caching & Prefetch eficiente na memória da GPU?**
  - SO1: Criação de Cache
  - SO2: Desenvolvimento de algoritmo de Prefetch
  - SO3: Identificação de tamanho de cache ideal
- **RQ2: Compressão de dados é aplicável?**
  - SO4: Avaliar compressores
  - SO5: Avaliar parametrização dos compressores
- **RQ3: Combinar Prefetch & Compressão traz mais performance?**
  - SO6: Avaliar a combinação de SO1, SO2, SO3 & SO4, SO5

SECTION

Checkpoint Cache & Prefetching RQ1 (SO1, SO2, SO3)

Caching



- LRU Cache na memória da GPU
- *Ranking*: Um array que aponta para o snapshot na memória. MRU ocupa a posição #0 e LRU a última posição
- Cada posição na cache tem seu próprio CUDA *stream*
- Tamanho Cache configurável (mínimo 2)

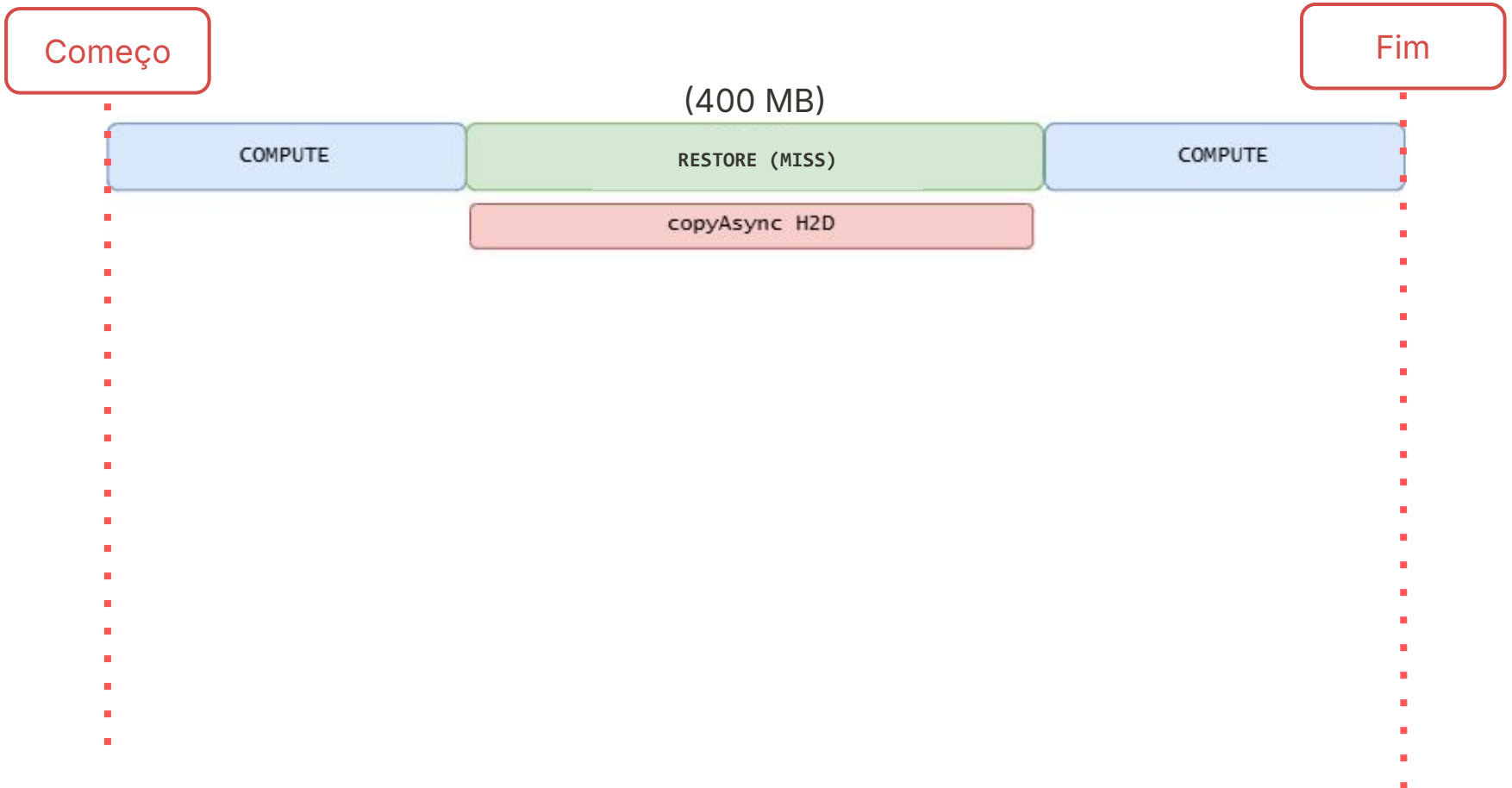
1. Save Checkpoint *Asynchronous Copy*
2. Restore Checkpoint (miss h2d) *Synchronous Copy*
3. Prefetch (h2d) *Asynchronous Copy*
4. Restore checkpoint (d2d) *Synchronous Copy*

Checkpointing data

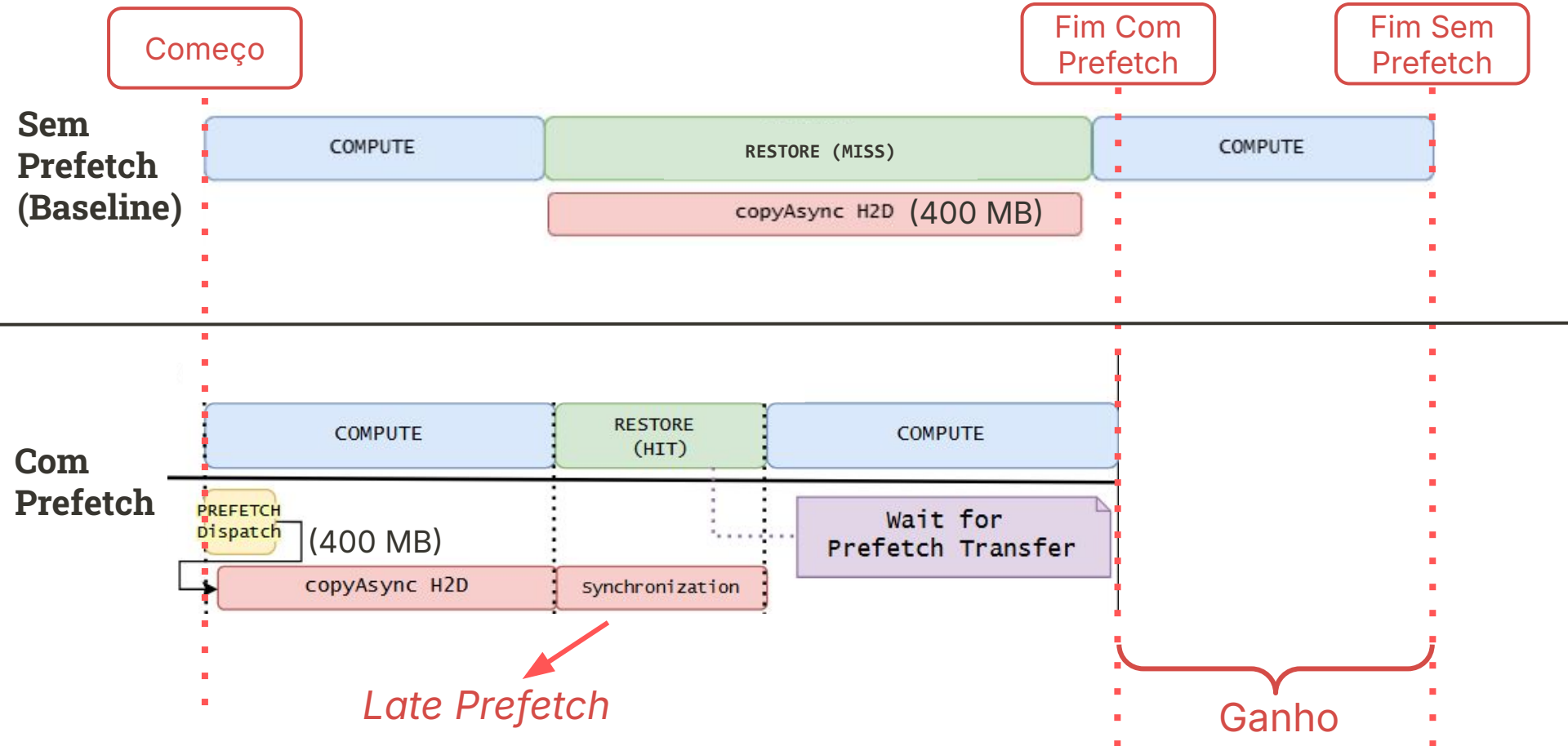
Cache Position Order Array

Cache Miss (c/ Cache 2 Posições)

Custo do cache miss



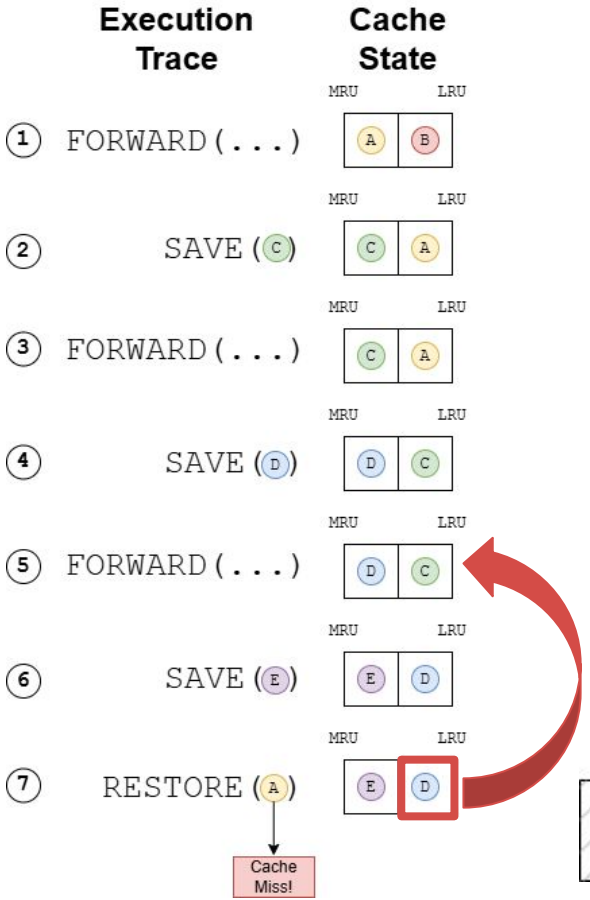
Benefício do Prefetch



Mecanismo de *Prefetching*

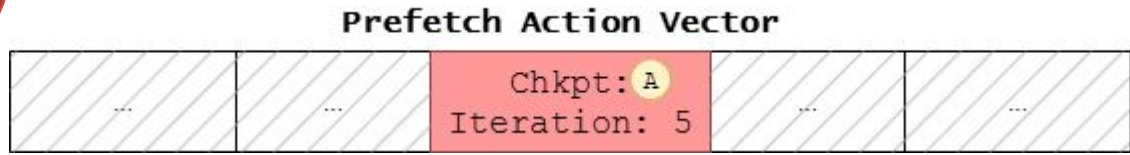
- Aproveita do determinismo das bibliotecas de checkpointing
- Executa o algoritmo de checkpointing em *dry-run* na cache (Prefetch Setup Algorithm) que identifica quando ocorreria um cache miss
- Agenda uma ação de *prefetch* do host para a GPU no Prefetch Action Vector (PAV)

Prefetch Setup Algorithm (PSA)



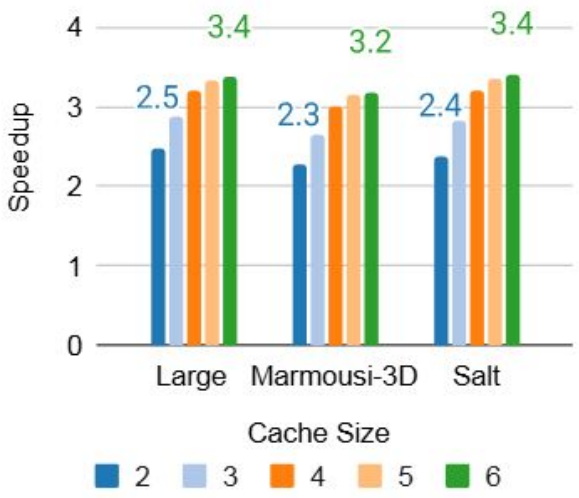
Olhando para essa execução, o checkpoint D (LRU) está “morto” desde a iteração (5)

Candidato para Prefetch: Iteração (5)

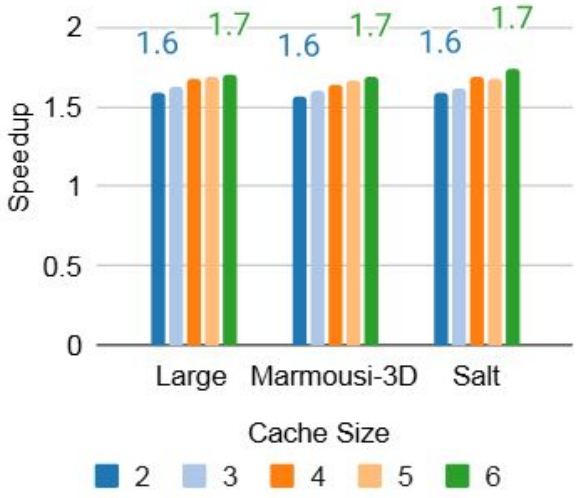


Execução Real

Speedup



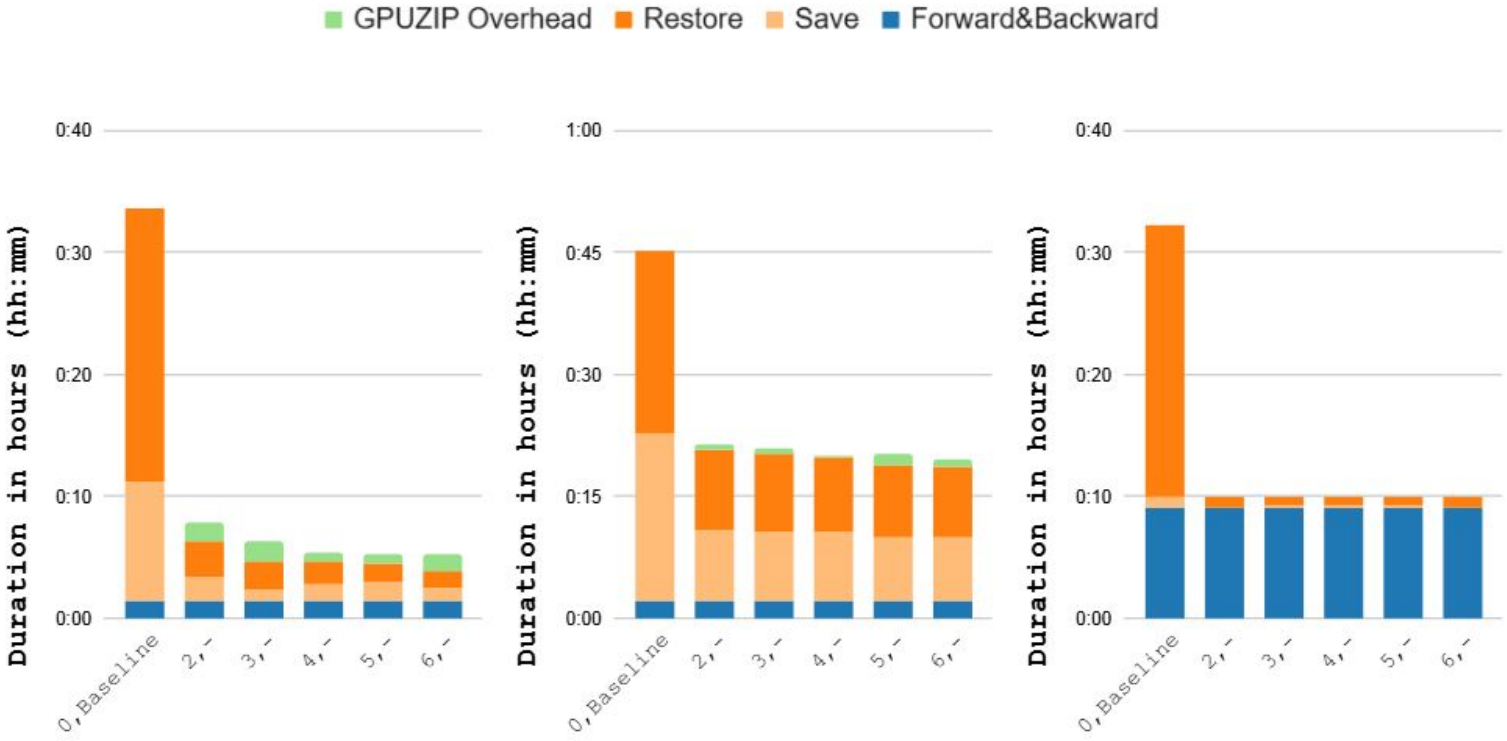
(a) Revolve



(b) zCut

(c) Uniform

Time Breakdown



(a) Revolve (b) zCut (c) Uniform

Figure 4.3: Execution time breakdown. From cache size 0 (baseline) to 6.

Consumo de Memória

CacheSize	Large(GB)	Marmousi3D(GB)	M3D_Larger(GB)	Salt(GB)
2	1.0	0.8	7.0	0.9
3	1.6	1.2	10.5	1.4
4	2.1	1.6	14.0	1.8
5	2.6	2.0	17.5	2.3
6	3.1	2.4	21.0	2.8

Table 4.1: GPU Memory required by the *GPU Checkpoint Cache* on each GPU.

SECTION

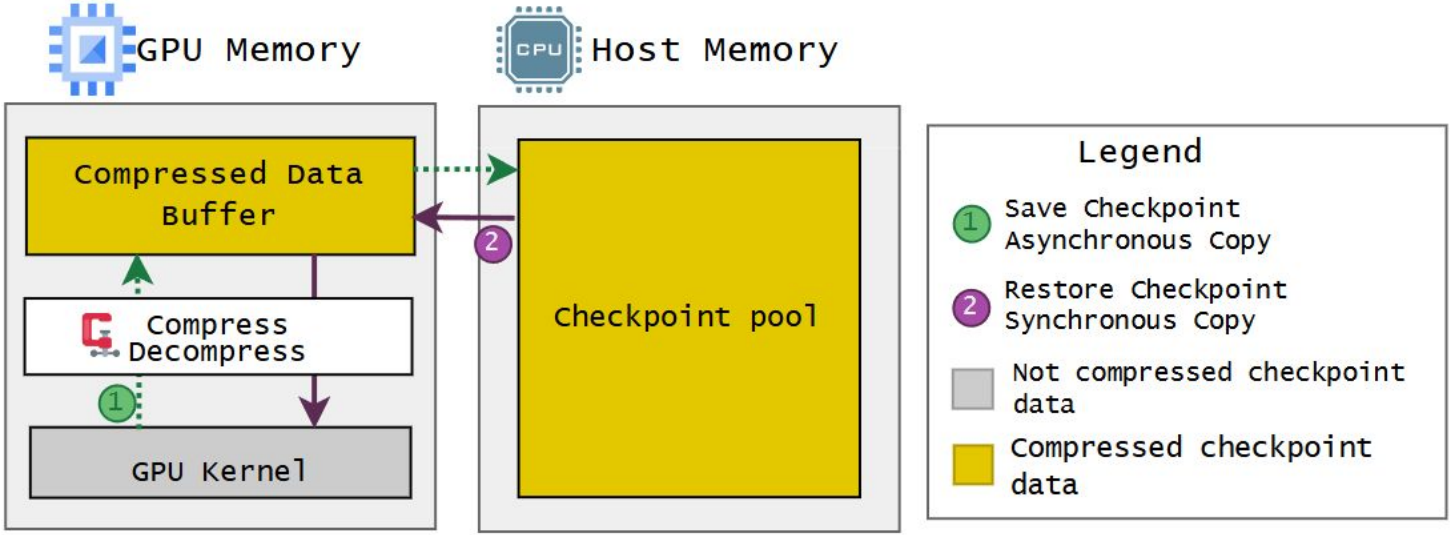
Aplicando Compressão em Dados de Checkpoint RQ2 (SO4, SO5)

Como compressão pode ajudar?

- Dados menores → menor tráfego de dados na PCI-e
- Dados menores → mais espaço de armazenamento disponível → mais espaço para armazenar snapshots.
    - Abordagem testada com Uniform

Integração

- Os dados já estão na GPU no momento do SAVE
- Os dados estarão na GPU na hora do RESTORE
- GPU é ser usada para comprimir os dados



Requisitos para funcionar

1) Impacto na performance

Sendo:

$$T_{\text{total}} = T_{\text{compress}} + T_{\text{transfer\_compressed}} + T_{\text{decompress}}$$

Satisfaça: $T_{\text{total}} < T_{\text{baseline}}$

2) Impacto na qualidade do resultado

Quais compressores utilizar?

→ Implementação para GPU

→ Gratuitos / Open Source

→ Com APIs disponíveis

→ Bons resultados em trabalhos relacionados

Compressores

cuSZp

→ Error bounded parameter

→ Uma das implementações mais recentes da família SZ para GPU

cuZFP

→ Fixed-rate mode apenas

→ Pertence à família ZFP de compressores

NVCOMP Bitcomp

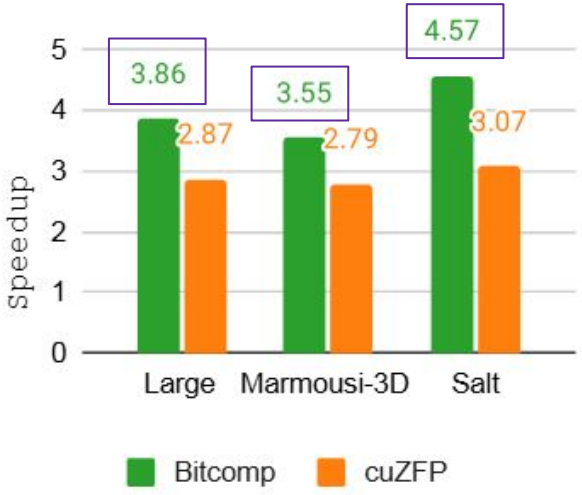
→ Não é open source

→ Error bound control

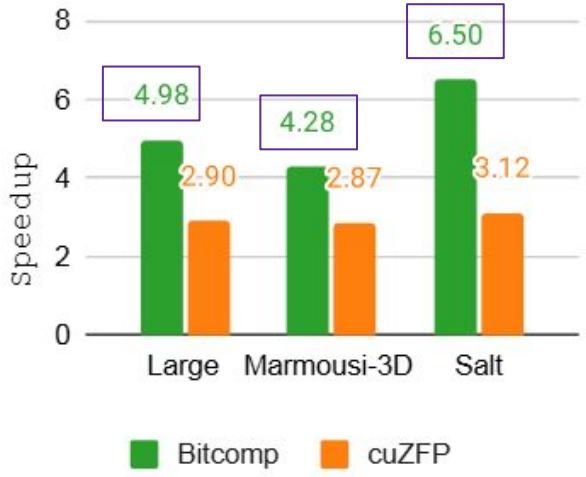
Quais parâmetros de compressão utilizar?

Profile	CR (AVG)	PSNR	SSIM	Time Spent d2h & h2d	Time spent (de)comp.
Marmousi3D					
baseline	-	-	-	1:16:10	-
bitcomp, delta=1e-2	597.03	41.70	0.0057	0:00:04	0:01:00
bitcomp, delta=1e-4	205.95	65.40	0.4589	0:00:31	0:01:04
bitcomp, delta=1e-8	115.96	95.00	1.0000	0:06:49	0:01:27
cuSZp,errBnd=1e-2	13.71	41.42	0.0000	0:03:53	0:06:14
cuSZp,errBnd=1e-4	5.42	41.51	0.0001	0:06:41	0:08:03
cuSZp,errBnd=1e-8	2.38	41.75	0.0001	0:15:22	0:12:51
cuZFP, bitRate=2	15.71	48.03	0.7190	0:02:45	0:02:51
cuZFP, bitRate=4	7.85	68.72	0.9834	0:05:33	0:03:32
cuZFP, bitRate=8	3.93	98.02	0.9996	0:11:16	0:04:01

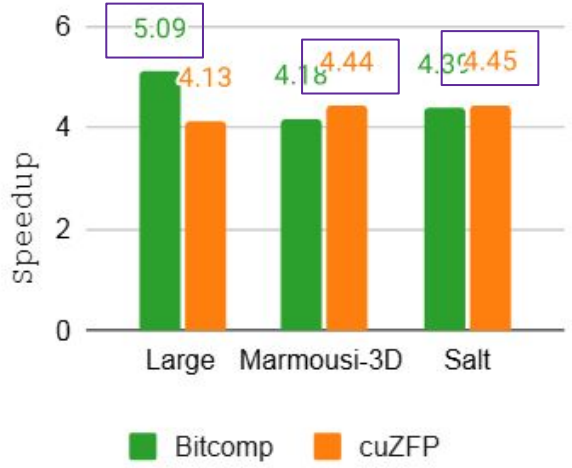
Speedup



(a) Revolve



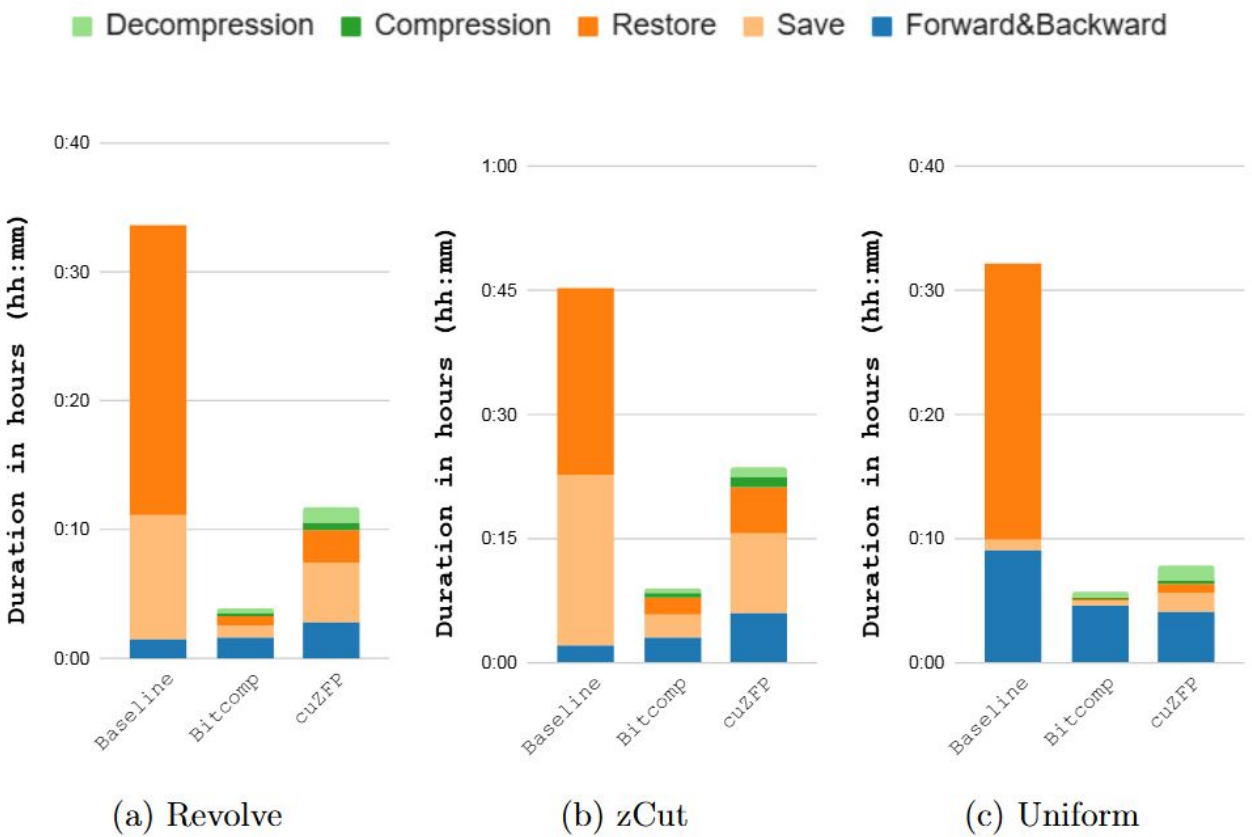
(b) zCut



(c) Uniform

Nota: Se Uniform não aumentasse *snapshots*, o *speedup* seria de 1.88x com bitcomp e 1.77x para cuZFP com Marmousi-3D

Time Breakdown

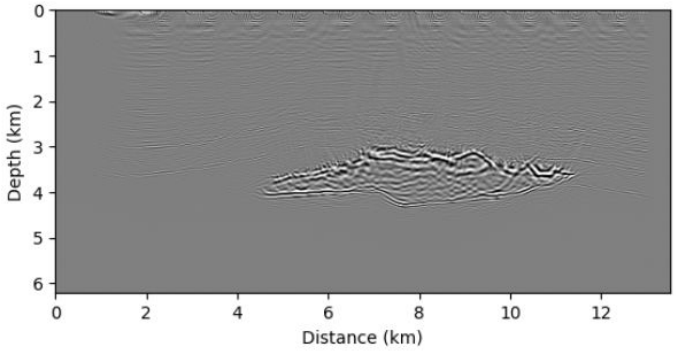


(a) Revolve (b) zCut (c) Uniform

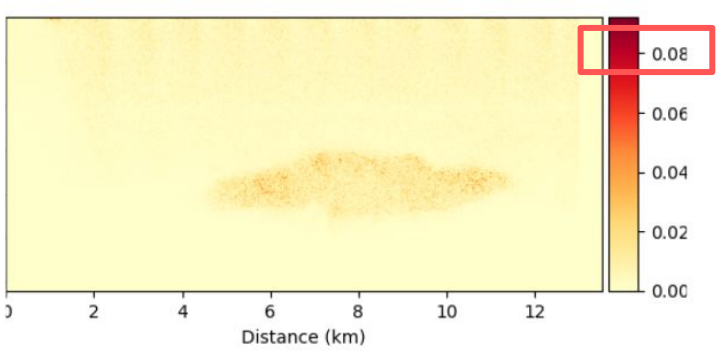
Avaliação de Qualidade

Dataset	Compressor	PSNR	SSIM
Large	cuZFP	105.98	0.99
	Bitcomp	138.06	0.99
Marmousi3D	cuZFP	94.23	0.99
	Bitcomp	120.94	0.99
Salt	cuZFP	78.01	0.99
	Bitcomp	135.82	0.99

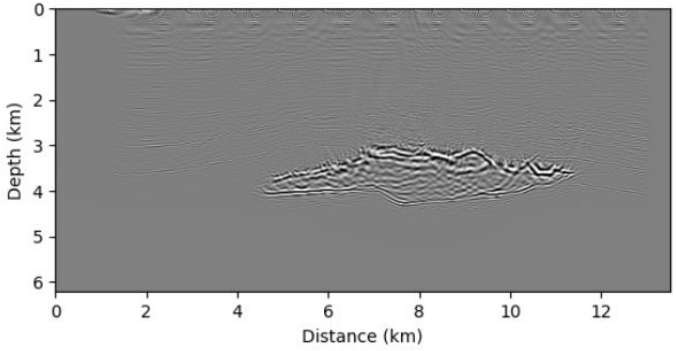
Avaliação de Qualidade



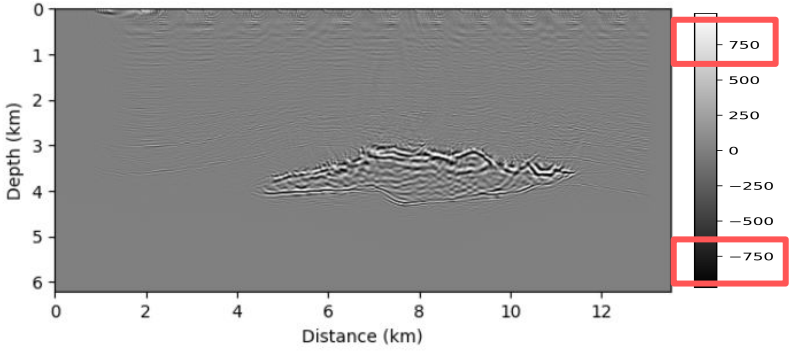
(a) Baseline



(b) Error Map with Bitcomp



(c) cuZFP

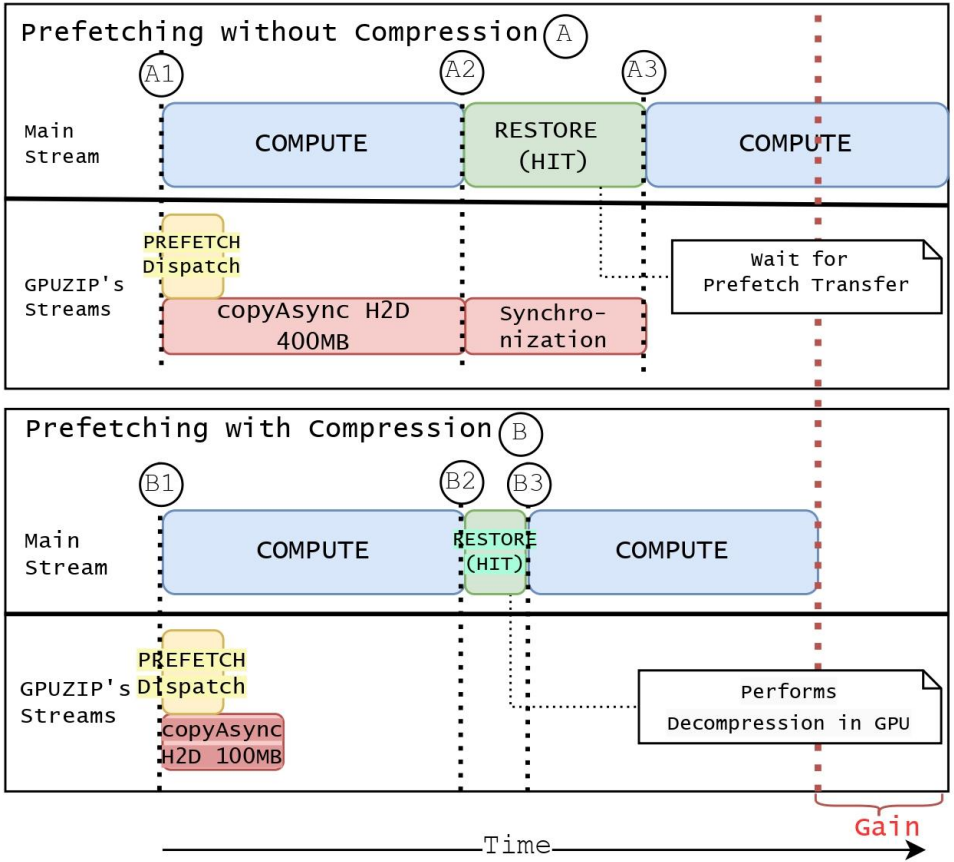


(d) Bitcomp

SECTION

Prefetching + Compressão
RQ 3 (SO6)

Como comprimir dados ajuda o Prefetch?

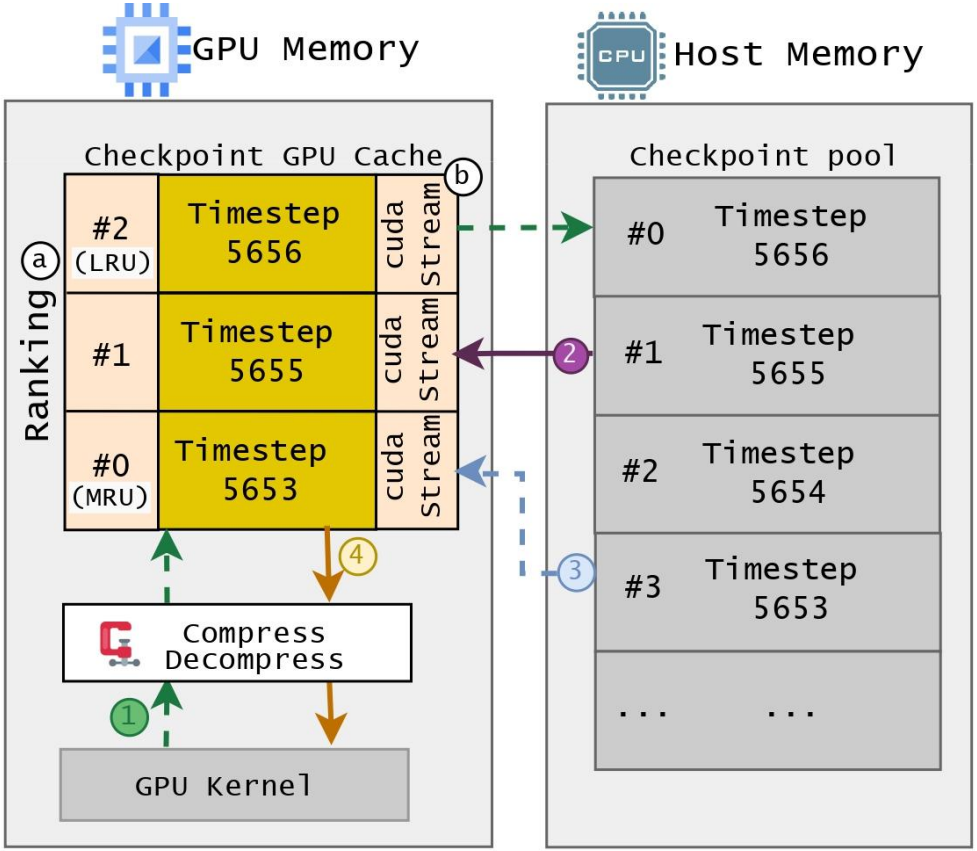


Comprimir os dados

==

Prefetch mais eficiente

Arquitetura



- Benefício de ter baixa transferência de dados na PCI-e que a compressão oferece
- Assincronismo do mecanismo de Prefetching

Legend

1. Save Checkpoint - Asynchronous Copy
2. Retrieve Checkpoint (miss h2d)- Synchronous Copy
3. Prefetch (h2d) - Asynchronous Copy
4. Retrieve checkpoint (d2d)- Synchronous copy

Checkpointing data

Compressed data

Cache Position Order Array

Speedup (Revolve)

+ uso de cache para obter performance

Bitcomp melhor que o prefetch-only e c/ menor consumo de memória

	Prefetch Only					Compression Only		Bitcomp				cuZFP		
	Cache Size					Comp. Type		Cache Size				Cache Size		
	2	3	4	5	6	Bit-comp	cu-ZFP	2	3	4	5	2	3	4
L	2.5	2.9	3.2	3.3	3.4	3.8	2.8	4.5	4.6	4.6	4.6	3.9	4.0	4.1
M	2.3	2.7	3.0	3.2	3.2	3.5	2.7	4.5	4.6	4.7	4.7	3.9	4.1	4.2
S	2.4	2.8	3.2	3.3	3.4	4.5	3.0	5.0	5.1	5.1	5.1	4.2	4.4	4.6

Table 6.1: Overall speedup for *Revolve* (checkpoint prefetching + compression). Datasets: L = Large, M = Marmousi3D, S = Salt.

Exige menos cache para atingir o pico de performance.
Melhor cenário

Speedup (zCut)

Melhor cenário!

	Prefetch Only	Compression Only		Bitcomp	cuZFP
	Cache Size = 2	Bitcomp	cuZFP	Cache Size = 2	Cache Size =2
L	1.6	4.9	2.9	7.8	5.4
M	1.6	4.2	2.8	6.9	5.5
S	1.6	6.5	3.1	8.8	5.5

Table 6.2: Overall speedup for *zCut* (checkpoint prefetching + compression). Datasets: L = Large, M = Marmousi3D, S = Salt.

Speedup (Uniform)

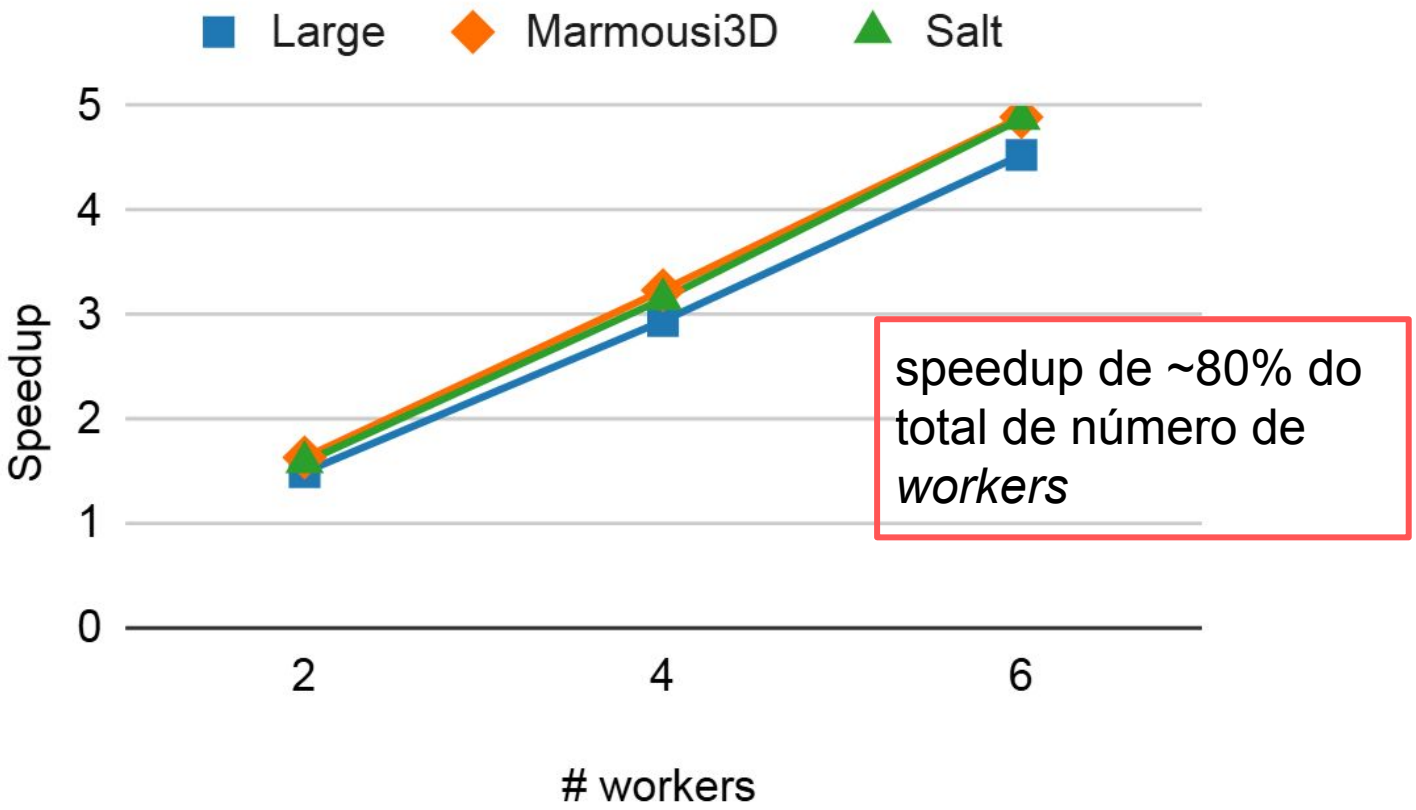
	Prefetch Only	Compression Only		Prefetch + Bitcomp	Prefetch + cuZFP
Ds	**Cache Size = 2**	**bitcomp**	**cuZFP**	**Cache Size = 2**	**Cache Size = 2**
L	2.3	5.09	4.13	5.8	5.1
M	1.8	4.18	4.44	4.4	4.8
S	2.1	4.39	4.45	4.5	5.1

Table 6.3: Overall speedup for *Uniform* (checkpoint prefetching + compression). Datasets: L = Large, M = Marmousi3D, S = Salt.

Uso de memória

Cache Size	Compressor	Large (GB)	Marmousi3D (GB)	M3D_Larger (GB)	Salt (GB)
2	-	1.0	0.8	7.0	0.9
2	Bitcomp	0.3	0.6	2.7	0.4
2	cuZFP	0.2	0.2	1.7	0.3
3	-	1.6	1.2	10.5	1.4
3	Bitcomp	0.4	0.5	4.0	0.6
3	cuZFP	0.7	0.3	2.6	0.4
4	-	2.1	1.6	14.0	1.8
4	Bitcomp	0.5	0.6	5.4	0.8
4	cuZFP	0.5	0.4	3.5	0.5
5	-	2.6	2.0	17.5	2.3
5	Bitcomp	0.6	0.8	6.7	1.0
5	cuZFP	0.6	0.5	4.4	0.7
6	-	3.1	2.4	21.0	2.8
6	Bitcomp	0.8	0.9	8.1	1.2
6	cuZFP	0.7	0.6	5.3	0.7

Escalabilidade em multi-nós



speedup de ~80% do total de número de *workers*

SECTION

GPUZIP: uma biblioteca modular para compressão e prefetch em checkpointing

Setup Inicial


```
typedef struct {
  int checkpointing_algorithm;  -> Revolve / zCut / Uniform

  int cache_capacity;

  int compressor; -> cuSZp, cuZFP, Bitcomp

  int log_level;  -> DEBUG | INFO | WARN | ERROR

  bool enable_performance_log;      -> Útil para experimentação

  bool enable_compression_rate_log; -> Útil para experimentação

  int zfp_bit_rate;

  double cuszp_err_bound;

  int bitcomp_delta_config;

  double bitcomp_range_fraction;

  double bitcomp_num_sigma;

  double bitcomp_delta;

  int bitcomp_algorithm;
} gpuzip_config_t;
```

Parâmetros de compressores

Setup Inicial


```
#include "prefetch/Prefetch.cuh"
#include "prefetch/Checkpointing.hpp"
#include "common/GPUZIPBuilders.cpp"
#include "common/GPUZIPConfig.h"
[ … ]

auto *chkpt = CheckpointingBuilder(gpuzip_config, steps);

auto *comp = CompressorBuilder(gpuzip_config, n1, n2, n3);

auto *prefetch = PrefetchBuilder(gpuzip_config, steps, chkpt);
```

Builders constroem os objetos antes da execução

Checkpointing in Action


```
for (int shot = 0; shot <= data->shots; shot++) {
  prefetch->setup();
  do {
     Action action = chkpt->GetAction();
      prefetch[d]->Dispatch(chkpt->GetIt());
     if (action.actionType == ACTION_SAVE) {
       prefetch[d]->Save(action.ts, &curr, &prev, comp.get(), compprev.get());
     } else if (action.actionType == ACTION_RESTORE) {
       prefetch[d]->Retrieve(action.ts, &curr, &prev, comp.get(), compprev.get());
     } // else if (ACTION_FORWARD or ACTION_BACKWARD…)
     if (action.actionType == ACTION_TERMINATE) { terminate = true; }
  } while(!terminate);
}
```

SECTION

Conclusão

Conclusão

- **RQ1: É possível Caching & Prefetch eficiente na memória da GPU?** ✓
  - SO1: Criação de Cache
  - SO2: Desenvolvimento de algoritmo de Prefetch
  - SO3: Identificação de tamanho de cache ideal
- **RQ2: Compressão de dados é aplicável?** ✓
  - SO4: Avaliar compressores
  - SO5: Avaliar parametrização dos compressores
- **RQ3: Combinar Prefetch & Compressão traz mais performance?** ✓
  - SO6: Avaliar a combinação de SO1, SO2, SO3 & SO4, SO5

Conclusão

- ➔ Técnicas de Prefetch e Compressão separadas trazem speedup, mas a combinação das duas técnicas é sempre o melhor cenário para performance.
- ➔ Todos os algoritmos de checkpointing se beneficiaram da combinação de prefetch+compressão.
- ➔ É escalável em múltiplos nós
- ➔ A biblioteca GPUZIP, é extensível, flexível, e configurável
- ➔ Próximos passos: aplicar a GPUZIP para outros tipos de aplicação, como machine learning.

Contribuições

Publicações:

- Poster na SuperComputing 2023 (Denver/CO - EUA)
- Conference Paper na EuroPAR 2024 (Madrid - Espanha)
- Special Issue na SAGE IJHPCA (The International Journal of High Performance Computing Applications) (2025)

—

Artefatos gerados:

- Biblioteca GPUZIP (Em aprovação para Open Source com Licença MIT)
- Datasets de entrada (Marmousi-3D, Large e Salt) hospedados em plataforma aberta (Unicamp REDU)

Obrigado!

email maltempi@ic.unicamp.br