Uma Análise do Impacto das Transferências de Dados em Aplicações OpenMP 4.5 em uma GPU de Baixo Consumo

Rafael G. Trindade, Bruno M. Muenchen, João V. F. Lima

WSCAD 2018
XIX Simpósio em Sistemas Computacionais de Alto Desempenho
01 a 03 de outubro de 2018 - São Paulo - SP - Brasil

Universidade Federal de Santa Maria
1960

informática
Universidade Federal de Santa Maria

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

CAPES

Roteiro

- Introdução
- Contexto
- Metodologia
- Resultados Experimentais
- Discussão e Conclusão

Introdução

- Advento de GP-GPUs
- GPUs em dispositivos embarcados
  - Memória Compartilhada
- Criação de bibliotecas para computação heterogênea
  - Considerável nível de complexidade
    - CUDA
    - OpenCL

Introdução

- Propostas de bibliotecas com APIs mais simples
  - OpenACC
  - OpenMP
- Ausência de trabalhos que explorem o uso de memória compartilhada nessas arquiteturas com OpenMP

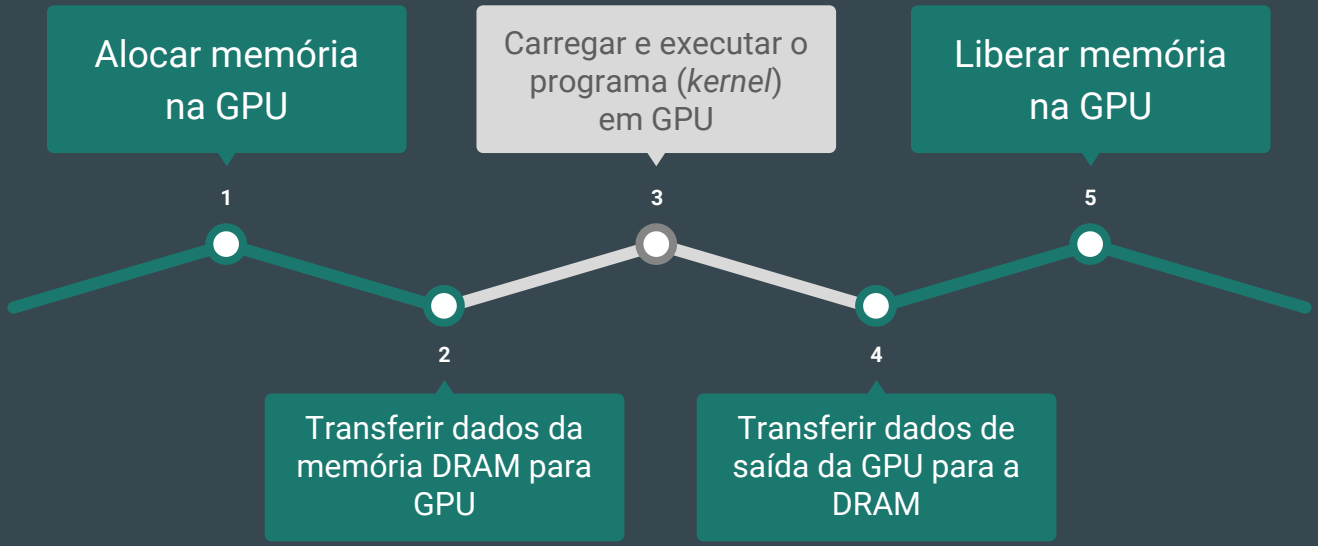
Contexto

Computação e gerenciamento de dados em GPU

- Execuções com estratégia fork/join
- Uso da API CUDA
- Comumente possuem memória dedicada
  - GPUs integradas podem trabalhar com memória unificada
    - *Zero-Copy*
    - *Unified Virtual Address* (UVA)

Contexto

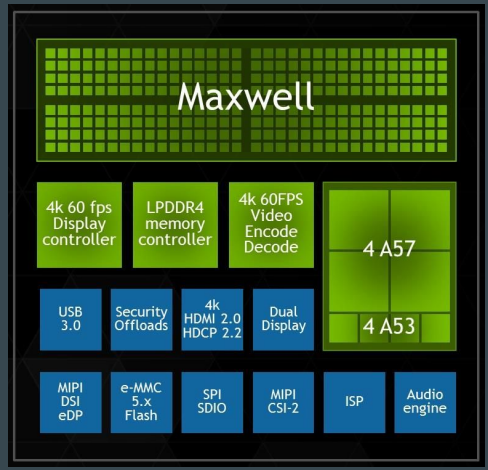
Anatomia de um programa com CUDA

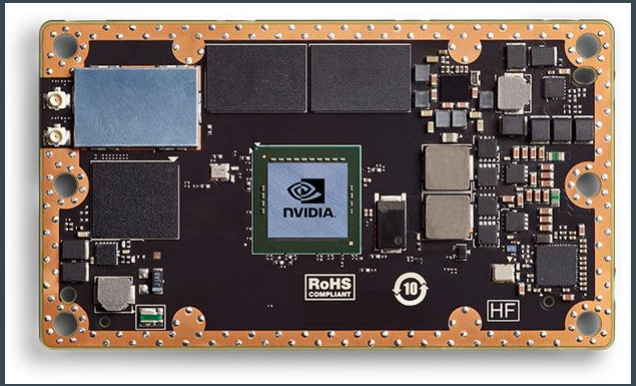


Contexto

NVIDIA Jetson TX1

- CPU ARM Cortex-A57 64-bit x4
  - CPU ARM Cortex-A53 64-bit x4
- GPU Maxwell GM20B
  256 CUDA cores
- 4GB DRAM LPDDR4
- Consumo 5 ~ 15 W





Contexto

OpenMP 4.x

- Versão 4.0 (Julho de 2013)
  - Suporte a aceleradores
    - Offloading de dados e instruções
- Versão 4.5 (Novembro de 2015)
  - Diretivas `target data enter` e `target data exit`

OpenMP

```
#pragma omp target map(to: A[:N]) map(tofrom: B[:N])
#pragma omp parallel for
for(i = 0; i < N; i++)
    B[i] = B[i] + A[i] * alpha;
```

CUDA

```
__global__
void saxpy(int N, float alpha, float *A, float *B) {
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < N)
    B[i] = B[i] + A[i] * alpha;
}

float *d_A, *d_b;
cudaMalloc(&d_A, N * sizeof(float));
cudaMalloc(&d_B, N * sizeof(float));
cudaMemcpy(d_A, A, N * sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, N * sizeof(float), cudaMemcpyHostToDevice);
saxpy<<<GRIDS, BLOCKS>>>(N, alpha, d_A, d_B);
cudaMemcpy(B, d_B, N * sizeof(float), cudaMemcpyDeviceToHost);
cudaFree(d_A);
cudaFree(d_B);
```

Objetivos

- Analisar o desempenho de programas OpenMP 4 em aceleradores de baixo consumo
  - Lattice-Boltzmann
  - LULESH
  - TeaLeaf
- Avaliar o impacto da eliminação de transferências de memória à GPU com duas técnicas
  - Zero-copy
  - Memória Unificada (UVA).

Metodologia

- Alterar implementação do Runtime do OpenMP 4.5
  - Suporte a diferentes tipos de uso de memória em GPU
- NVIDIA Jetson TX1
  - CUDA v 8.0
  - IBM Clang OpenMP 4.5
  - Ubuntu 16.04

Metodologia - Modificação da Runtime OpenMP

Tradicional

Cópia explícita de dados para GPU

- Aloca/desaloca memória em GPU
- Transfere dados de entrada e saída

Zero-Copy

Acesso à área de memória em comum com a CPU

- Evita alocações e transferências adicionais
- Ignora níveis de cache em GPU e CPU

Memória Unificada (UVA)

Acesso à área de memória em comum com a CPU

- Evita alocações e transferências adicionais
- Há gerenciamento de cache para assegurar a coerência dos dados

Metodologia - Aplicações Científicas

Lattice-Boltzmann

- Dinâmica de Fluídos Computacional
- Demanda muita memória e processamento

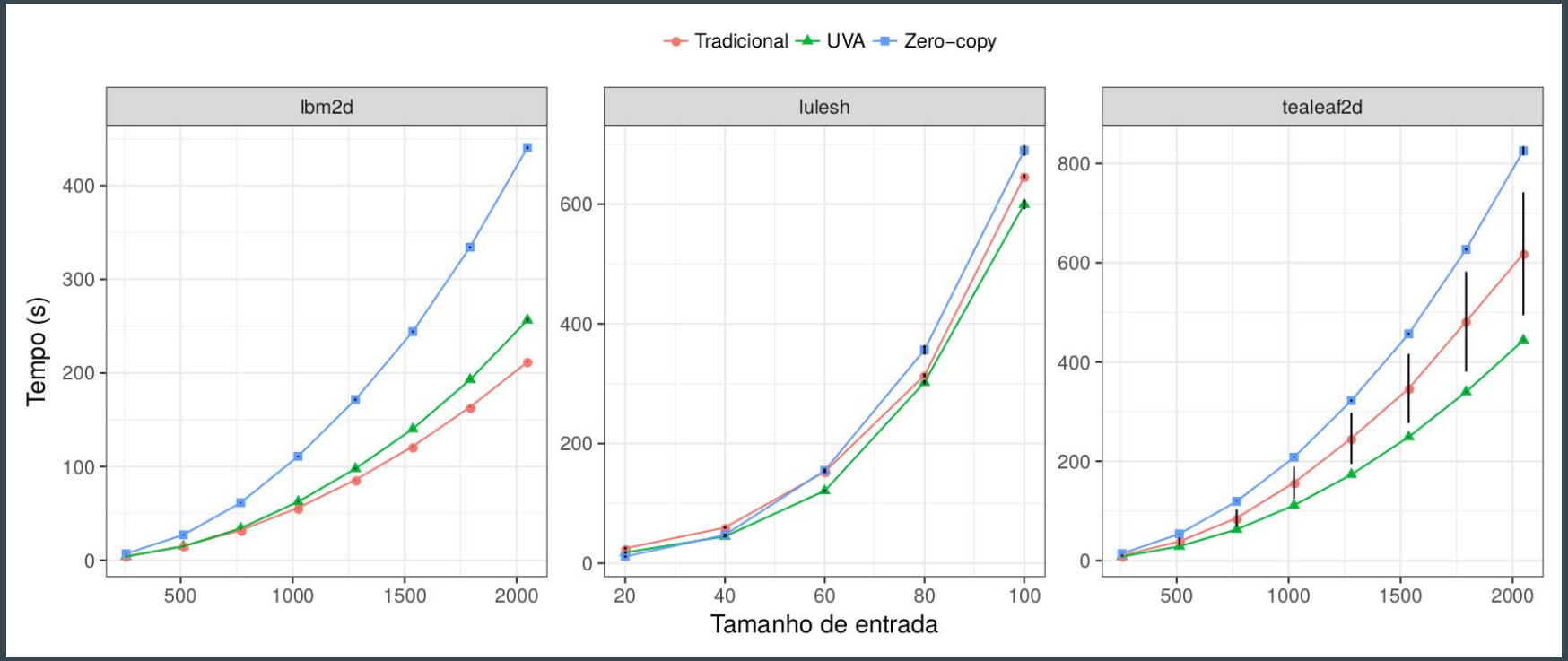
LULESH

- Dinâmica de Fluídos Computacional
- Possui versões com diversas bibliotecas (OpenMP, OpenACC, CUDA, etc)

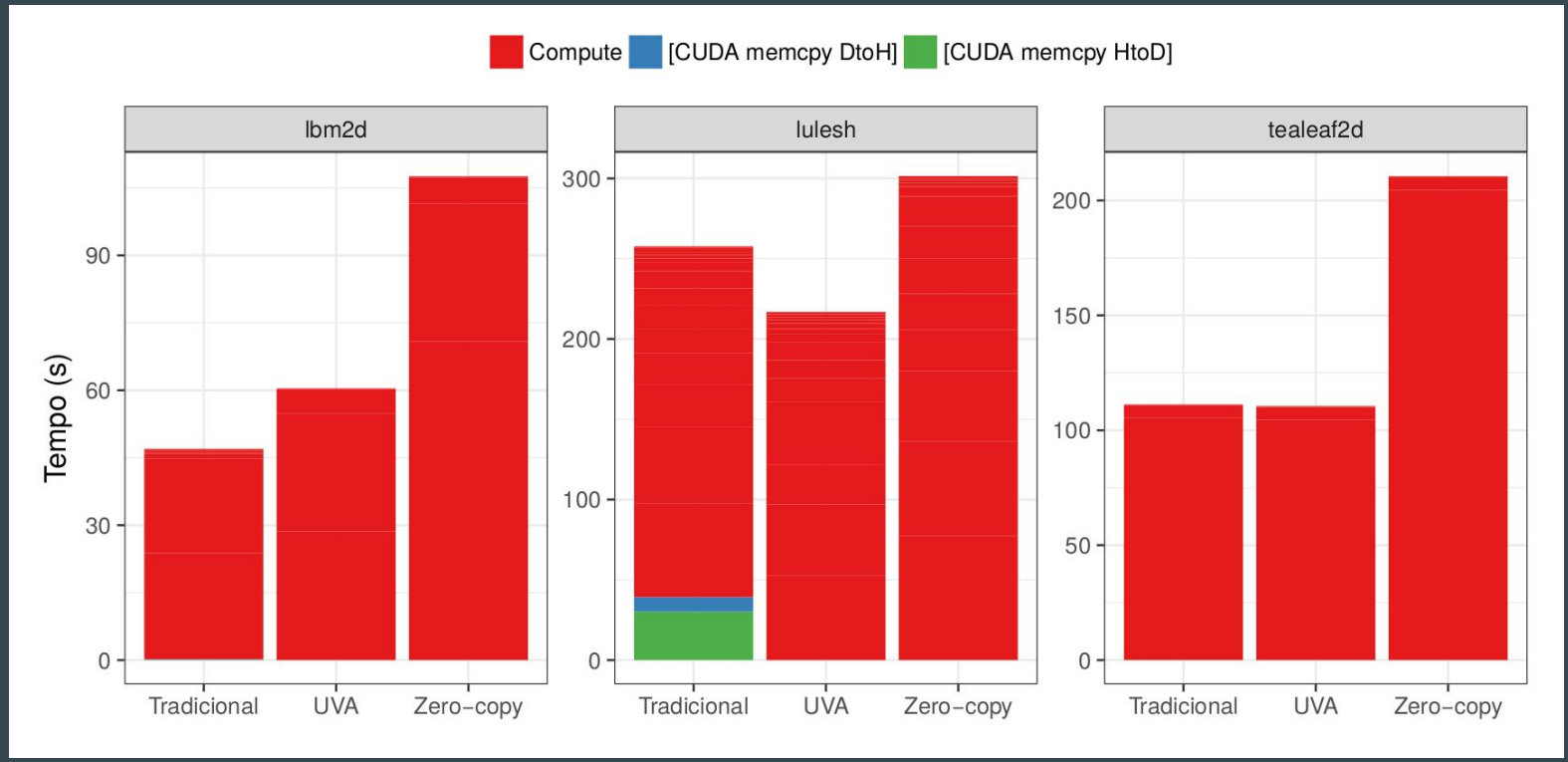
TeaLeaf

- Simulação de condução térmica
- Faz uso de três solucionadores diferentes de matrizes esparsas

Resultados Experimentais - Tempo por tamanho de entrada



Resultados Experimentais - Tempos por implementação



Discussão/Conclusão

- Há benefícios na ausência de cópia de dados com OpenMP
- UVA pode reduzir o tempo de execução em até 28%
- Zero-Copy mostrou-se uma pior opção em todos os casos

Obrigado!

Perguntas?

rtrindade@inf.ufsm.br

Este trabalho foi realizado com suporte das instituições CNPq e CAPES.
Agradecimentos especiais à NVIDIA pela doação da plataforma embarcada utilizada neste trabalho.