Programação Avançada de Múltiplas GPUs com OpenACC

Calebe P. Bianchini
Evaldo B. Costa
Gabriel P. Silva

centro universitário FEI

https://doi.org/10.5753/sbc.16630.8.3

https://www.casadocodigo.com.br/products/livro-programacao-paralela

Casa do Código

centro universitário FEI

Introdução

- Processadores multinúcleo e aceleradores.
- Aceleradores gráficos de propósito geral (GPGPU - General-Purpose Graphics Processing Unit).
- OpenACC é uma API baseada em diretivas que simplifica a tarefa de paralelização de código.
- Outras tecnologias de programação paralela para aceleradores:
  - CUDA
  - OpenMP
  - OpenCL

Arquitetura de Aceleradores

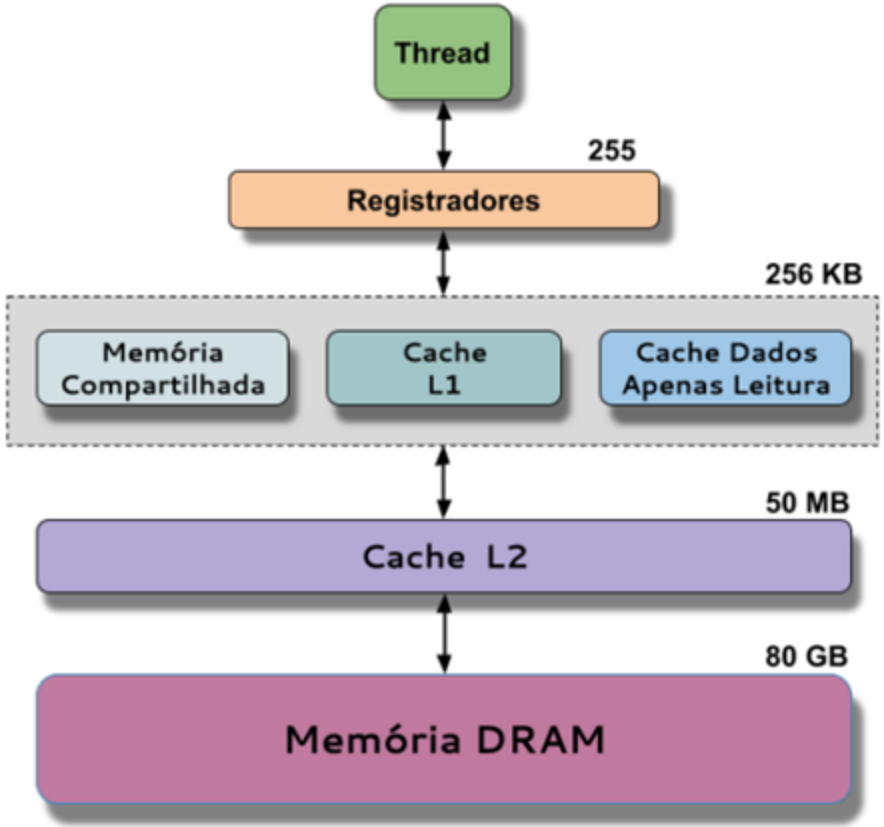
- As arquiteturas dos aceleradores gráficos (GPUs) são bem diferenciadas das arquiteturas dos processadores convencionais.
- Aceleradores são um tipo particular de arquitetura com exploração de paralelismo no nível de thread.
- O paralelismo nos aceleradores gráficos é explorado através de um conjunto maciço de multiprocessadores de fluxo.
- Vamos utilizar como exemplo a arquitetura do acelerador gráfico NVIDIA Hopper.

Arquitetura NVIDIA Hopper

Arquitetura NVIDIA Hopper

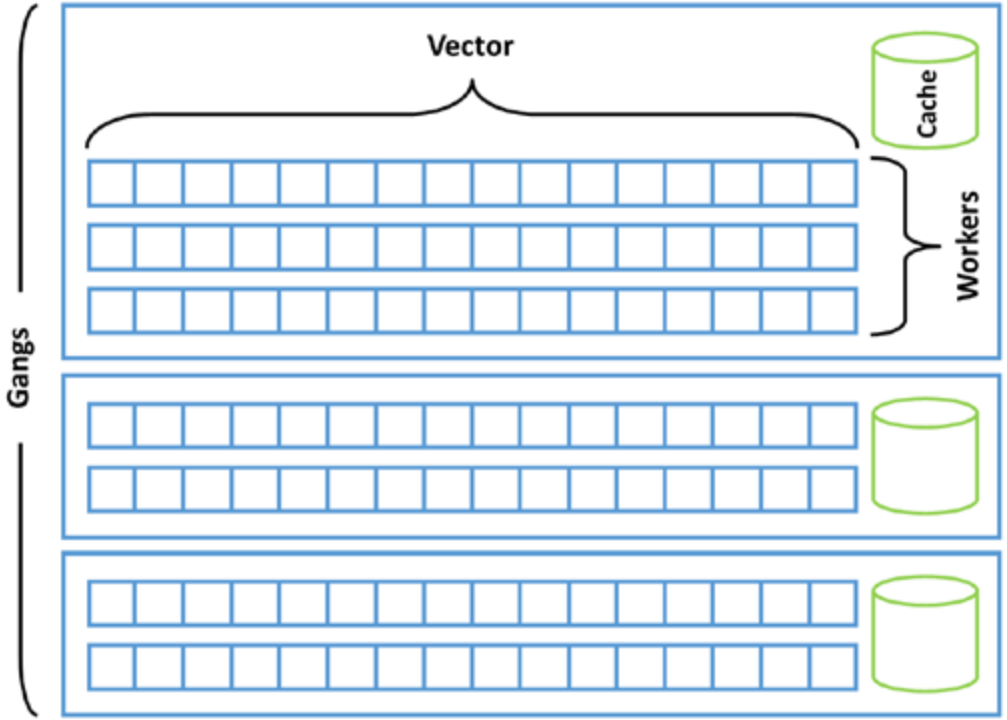
Multiprocessador de Fluxo

Hierarquia de Memória



Conceitos Básicos

Níveis de paralelismo do OpenACC

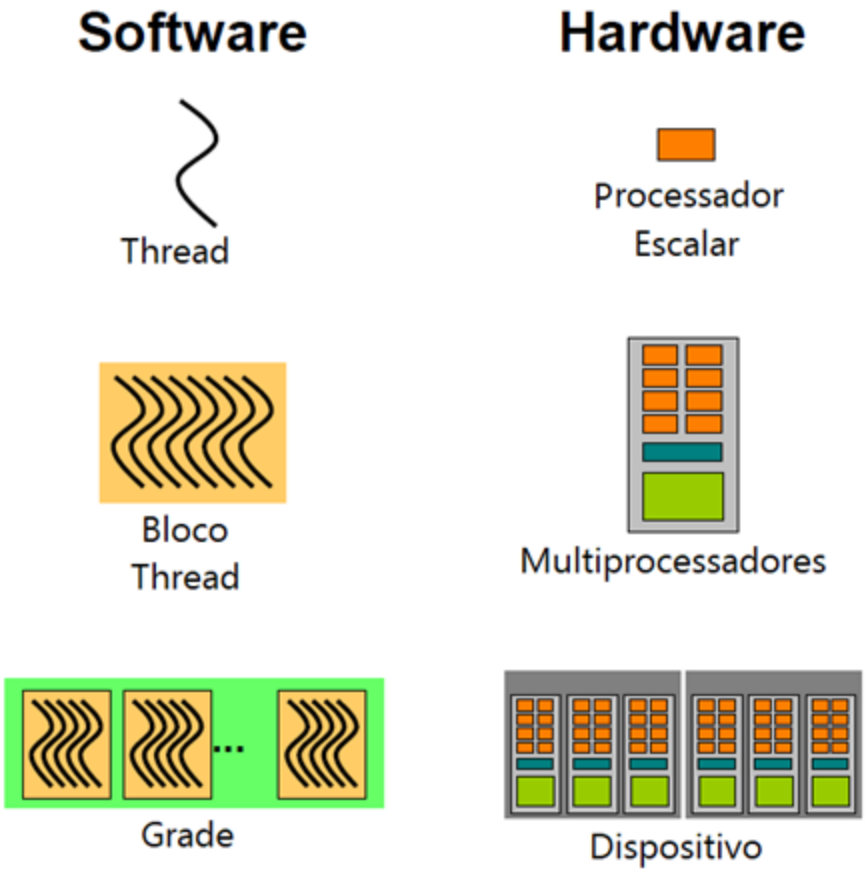


Níveis de paralelismo do OpenACC

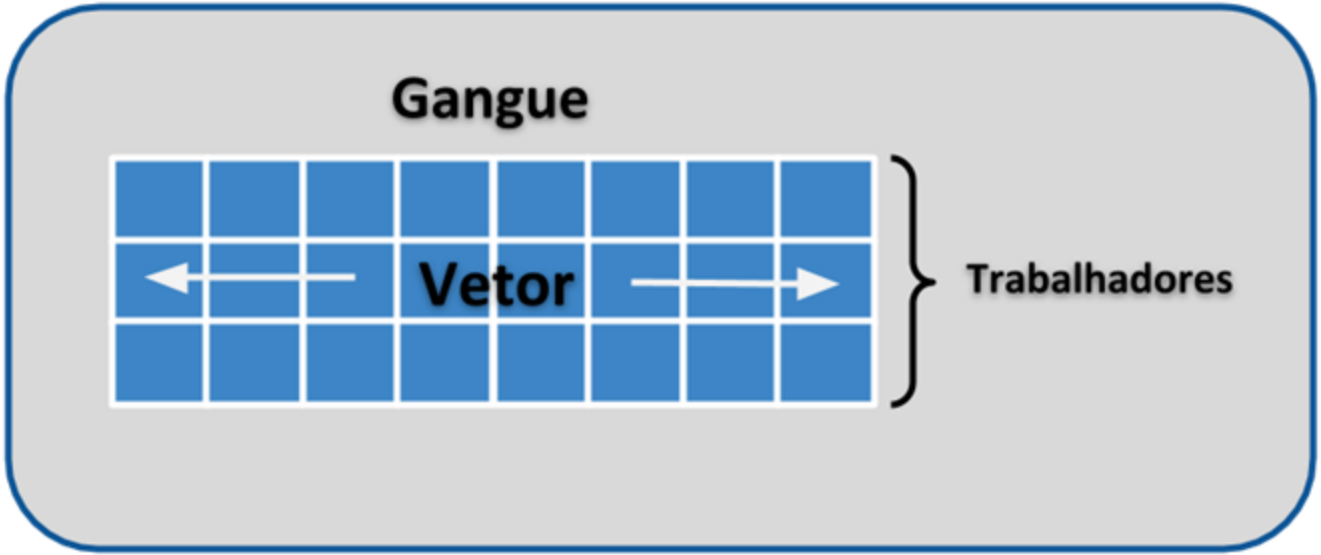
gang	Particiona o laço entre as gangs
worker	Particiona o laço entre os workers
vector	Vetoriza o laço
seq	Não particiona o laço, executar sequencialmente

gang ⇔ block
worker ⇔ warp
vector ⇔ threads

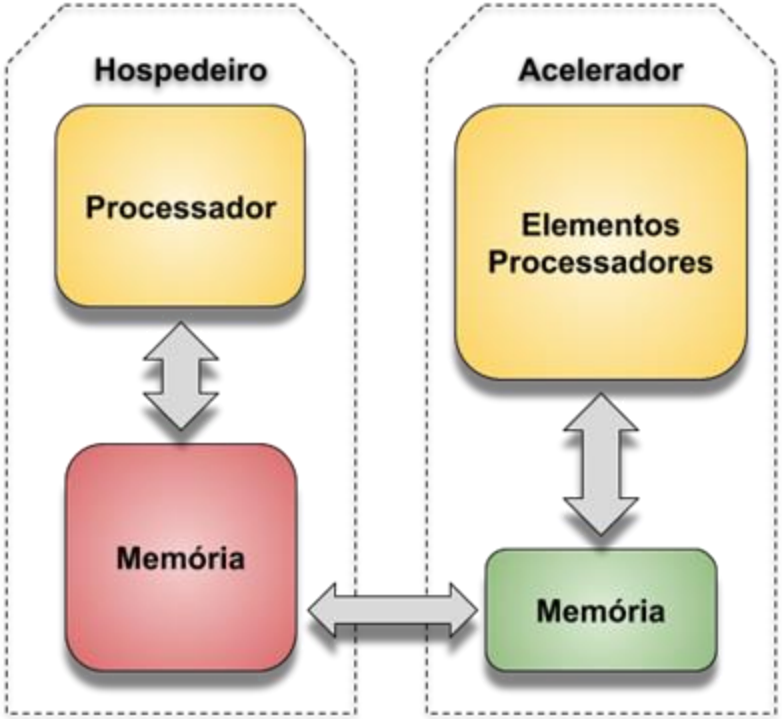
Níveis de paralelismo do OpenACC



Níveis de paralelismo do OpenACC



Movimentação de Dados: hospedeiro - dispositivo



Movimentação de Dados: hospedeiro - dispositivo

- Principais **cláusulas** utilizadas para a movimentação de dados:
  - data
  - copy
  - copyin
  - copyout
  - create
  - present
  - deviceptr

Movimentação de Dados: hospedeiro - dispositivo

```
#pragma acc data
  {
    #pragma acc parallel loop
      for (i=0; i<N; i++) {
           y[i] = 0.0f;
           x[i] = (float)(i+1);
      }

    #pragma acc parallel loop
      for (i=0; i<N; i++)
           y[i] = 2.0f * x[i] + y[i];
  }
```

Diretivas e Cláusulas Avançadas

Cláusula collapse

- A paralelização de um laço em OpenACC está associada ao laço imediatamente a seguir.
- Contudo, alguns laços não têm um número total de iterações suficientemente grande para fazer uso efetivo do acelerador.
- Quais são as vantagens de usar a cláusula **collapse**?
  - Colapsar os laços externos para permitir a criação de mais gangues;
  - Colapsar os laços internos para permitir comprimentos de vetor mais longos;
  - Colapsar todos os laços, quando for possível, para fazer as duas coisas: ter mais gangues criadas e vetores maiores.

Cláusula collapse

```
#pragma acc parallel loop collapse(2)
    for (int i = 0; i < N; i++)
         for (int j = 0; j < M; j++)

#pragma acc parallel loop
    for (int ij = 0; ij < N*M; ij++)...
```

Cláusula tile

- Otimizar o laço através da operação de blocos menores para explorar o acesso à localidade dos dados, aproveitando-se da hierarquia de memória da GPU.

```
#pragma acc parallel loop private(i,j) tile(8,8)
for(i=0; i<rows; i++)
     for(j=0; j<cols; j++)
        out[i*rows + j] = in[j*cols + i];
```

Diretiva routine

- A partir do OpenACC 2.0.
- Criar uma função ou sub-rotina para que possa ser chamada de uma região de dispositivo.
- Permite o uso de funções recursivas.

Diretiva routine

```
#pragma acc routine vector
void foo(float* v, int i, int n) {
  #pragma acc loop vector
  for (int j = 0; j < n; ++j)
      v[i*n+j] = 1.0f/(i*j);
}

#pragma acc parallel loop
for ( int i=0; i<n; ++i) {
  foo(v,i);
  //chamada no dispositivo
}
```

Operações atômicas

- Garante que os dados não sejam acessados simultaneamente.
- Cláusulas da diretiva **atomic**:
  - read
  - write
  - update
  - capture

Operações atômicas

```
#pragma acc parallel loop
for(int i=0; i < N; i++)
    h[i]=0;

#pragma acc parallel loop
for(int i=0; i < N; i++) {
#pragma acc atomic update
    h[a[i]]+=1; }
```

Operações Assíncronas

- Após minimizar as transferências de dados, pode ser possível reduzir ainda mais as perdas de desempenho associadas a essas transferências sobrepondo as cópias de dados com outras operações no hospedeiro, no dispositivo ou em ambos.
  - Cláusula async
  - Diretiva wait

Operações Assíncronas

```
#pragma acc parallel loop async
    for (int i=0; i<N; i++)
         c[i] = a[i] + b[i]
#pragma acc update self(c[0:N]) async
```

```
#pragma acc parallel loop async
    for (int i=0; i<N; i++)
         c[i] = a[i] + b[i]
#pragma acc update self(c[0:N]) async
#pragma acc wait
```

Operações Assíncronas

```
#pragma acc parallel loop async(1)
    for (int i=0; i<N; i++)
         a[i] = i;
#pragma acc parallel loop async(2)
    for (int i=0; i<N; i++)
         b[i] = 2*i;
#pragma acc wait(1) async(2)
#pragma acc parallel loop async(2)
    for (int i=0; i<N; i++)
        c[i] = a[i] + b[i]
#pragma acc update self(c[0:N]) async(2)
#pragma acc wait
```

Programação multi-GPUs

Rotinas

- Para sistemas que contêm mais de um acelerador, o OpenACC fornece um conjunto de rotinas para realizar as operações em um dispositivo específico ou para transferir dados entre dispositivos de forma eficiente.
  - acc_get_num_devices()
  - acc_get_device_num()
  - acc_set_device_num()
  - acc_get_device_type()
  - acc_set_device_type()

Rotinas

```
#include <stdio.h>
#include <openacc.h>
int main() {
    acc_device_t device_type;
    // Obtém o tipo do dispositivo atual
    device_type = acc_get_device_type();
    // Exibe o tipo de dispositivo
    if (device_type == acc_device_gpu()) {
        printf("Dispositivo atual é uma GPU.\n");
    } else if (device_type == acc_device_host()) {
        printf("Dispositivo atual é o hospedeiro (CPU).\n");
    } else {
        printf("Outro dispositivo.\n");
    }
    return 0;
}
```

Verificação Inicial

- Antes de iniciar o processo de execução de uma aplicação para uso de múltiplas GPUs, é preciso verificar se o ambiente possui todas as características para o perfeito funcionamento, garantindo que existam, no mínimo, duas GPUs no ambiente e se elas estão prontas para uso.
    - nvidia-smi -q
    - pgaccelinfo | grep "Device Number"
    - nvc --version

Estudo de caso

Fractal de Mandelbrot

- O fractal de Mandelbrot é um fractal definido como o conjunto de pontos C no plano complexo.
- O conjunto de Mandelbrot é obtido quando submetemos os números complexos a um processo iterativo e recursivo utilizando a fórmula.

$$Z_{n+1} = Z_n^{\,2} + C$$

Fractal de Mandelbrot

- Código de Mandelbrot para uso em múltiplas GPUs está disponível no link abaixo:
  - https://github.com/OpenACC/openacc-best-practices-guide/blob/main/examples/mandelbrot/cpp/task5.multithread/
  - https://github.com/Programacao-Paralela-e-Distribuida/OpenACC

Fractal de Mandelbrot

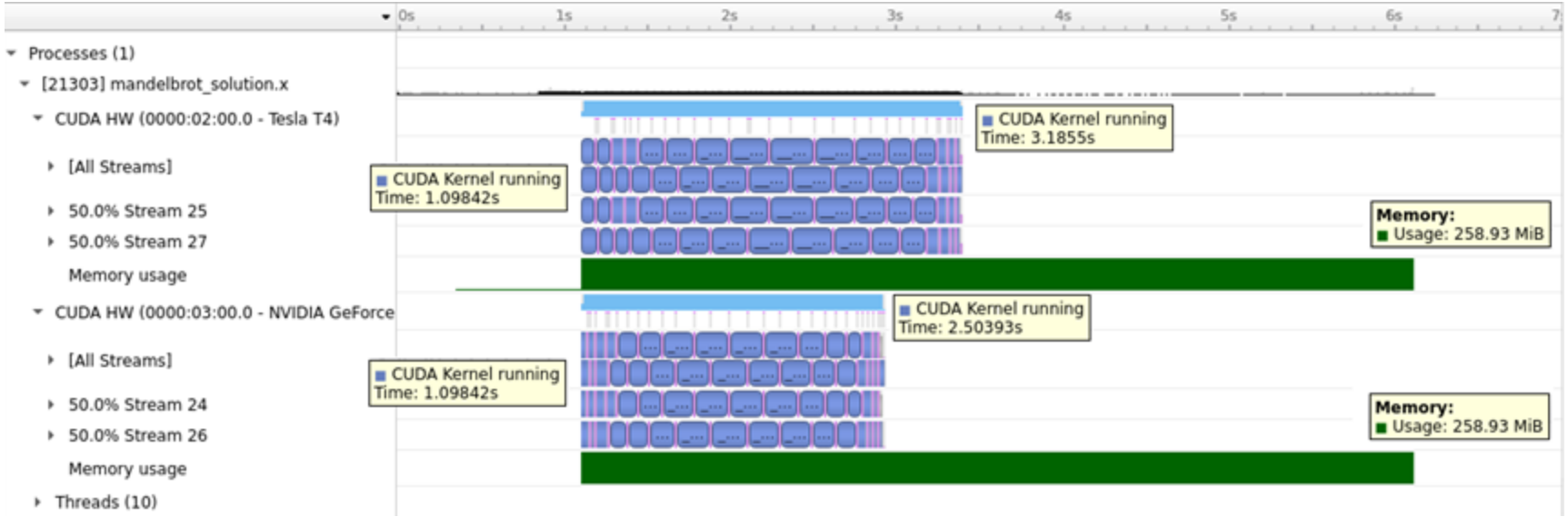
- Compilando e executando o código:

```
$ make mandelbrot_solution.x nvc++ -fast -acc=gpu -
gpu=pinned -Minfo=all -mp -o mandelbrot_solution .x
mandelbrot.o main_solution.o

$ ./mandelbrot_solution.x
    Found 2 NVIDIA GPUs.
    Thread 0 is using GPU 0
    Thread 1 is using GPU 1
    Time: 2.280399 seconds.
```

Fractal de Mandelbrot

- Para a análise da utilização dos recursos durante o processo de execução do código, utilizamos o NVIDIA Nsight Graphics.



Considerações Finais

- A otimização da movimentação de dados entre o hospedeiro e o acelerador com uso das diretivas de dados como data, copy, copyin, copyout, create, present e deviceptr.

- Abordamos as diretivas e as cláusulas avançadas do OpenACC, incluindo collapse para otimizar laços aninhados, routine para criar versões de dispositivo de funções, atomic para garantir acesso seguro à memória compartilhada, e tile para melhorar a localidade de dados nos laços.

  Gestão de múltiplos aceleradores, demonstramos como utilizar as rotinas da API, como acc_get_num_devices(), acc_get_device_num(), acc_set_device_num(), acc_get_device_type() e acc_set_device_type(), distribuindo a carga de trabalho de forma eficiente.

- O estudo de caso do fractal de Mandelbrot.

Programação Avançada de Múltiplas GPUs com OpenACC

Obrigado!

Calebe P. Bianchini
Evaldo B. Costa
Gabriel P. Silva

centro universitário FEI