Getting up and Running with the OpenMP Cluster Programming Model

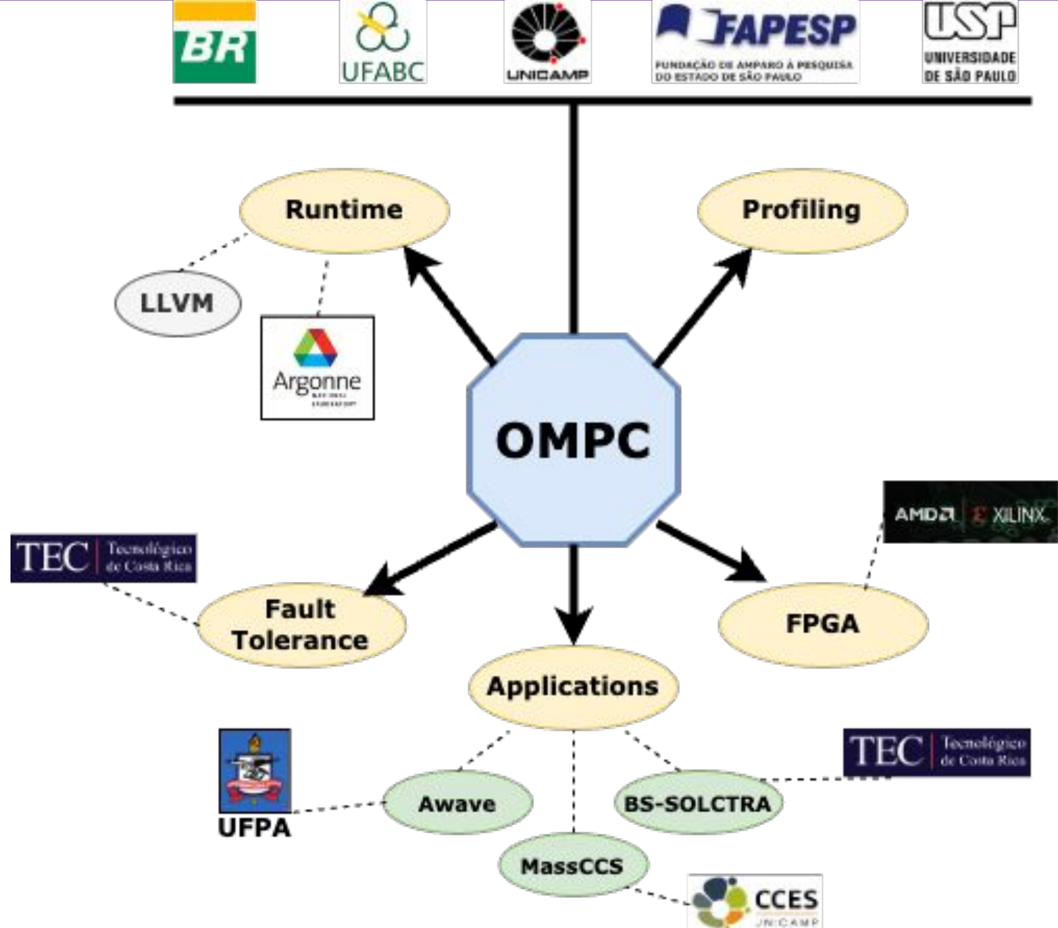
Emilio Francesquini e.francesquini@ufabc.edu.br
Márcio Pereira mpereira@ic.unicamp.br
Guido Araújo guido@unicamp.br
Hervé Yviquel herve@unicamp.br
Sandro Rigo sandro@ic.unicamp.br
and **The OMPC Team**

Universidade Federal do ABC (UFABC)

Universidade de Campinas (Unicamp)

WSCAD 2023 • 17 de outubro de 2023 • Porto Alegre, RS - Brasil

OMPC Project Overview



The OMPC Team

PROJECT LEADER

Guido Araujo

SENIOR MEMBERS

Hervé Yviquel

Marcio Pereira

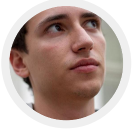
Sandro Rigo

Emilio Francesquini

Alan Souza

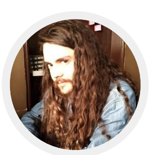
Nusrat Lisa

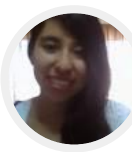
GRADUATE MEMBERS


Guilherme Valarini

Gustavo Leite

Pedro Rosso

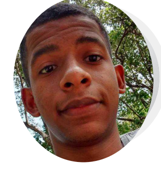

Rodrigo Freitas


Carla Cusihualpa

Vitória Dias

Thiago Maltempi

UNDERGRAD MEMBERS


Jhonatan Cléto

Matheus

Bruno Toso

O Laboratório de Sistemas de Computação (LSC)

LSC COMPUTER SYSTEMS LABORATORY

Aluno do LSC faz estágio na Irlanda

Pedro Rosso

LSC AMD XILINX

Alunos do LSC Fazem Estágio em um dos Maiores Laboratórios dos EUA

LSC COMPUTER SYSTEMS LABORATORY Argonne NATIONAL LABORATORY

Comitê do OpenMP Destaca a extensão do OpenMP Cluster para FPGAs

LSC COMPUTER SYSTEMS LABORATORY OpenMP

https://www.linkedin.com/company/lsc-unicamp

https://www.instagram.com/lsc.unicamp/

O Laboratório de Sistemas de Computação (LSC)

https://www.linkedin.com/company/lsc-unicamp

https://www.instagram.com/lsc.unicamp/

Link para a Apresentação

https://tinyurl.com/wscad23

Antes de começarmos...


```
# Para seguir este tutorial, você precisará de:
# - Máquina Linux
# - Git
# - Python 3.7+
# - Docker

$ git clone https://gitlab.com/ompcluster/conferences/wscad-2023.git
$ git clone https://gitlab.com/ompcluster/ompcbench.git
$ docker pull ompcluster/runtime:latest
```

Parte I

Visão Geral

- Introdução
- Tarefas OpenMP
- OmpCluster Runtime

Introdução

Por que é difícil programar em clusters HPC?

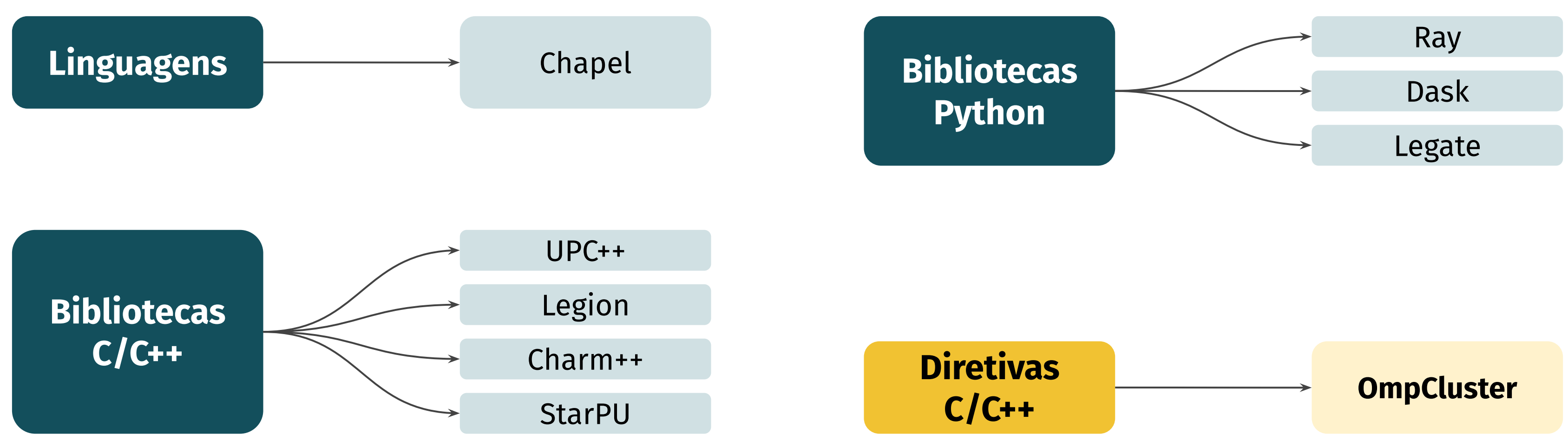
Programar em clusters HPC é difícil e propenso a erros

1. **Muitas linguagens**
   C/C++, Fortran, CUDA, OpenCL, Python.
2. **Muitas bibliotecas**
   MPI, OpenMP, OpenACC, Ray, Charm++, ...
3. **Muitos conceitos**
   Sistemas operacionais, arquitetura de computadores, rede etc

E se fosse possível programar em clusters HPC usando um modelo de programação simples baseado em diretivas como o OpenMP?

Panorama da programação HPC

Paralelismo baseado em tarefas tem se tornado o principal paradigma na programação em HPC



OpenMP: Fácil programação em cluster

Programação baseada em diretivas que especificam *o que fazer* e não *como fazer*.

- Usuário escreve anotações e deixa o compilador fazer o trabalho pesado.
- LLVM OpenMP libomptarget.
- OpenMP Cluster (OMPC)

Estrutura de um cluster HPC

Hardware

- Servidores organizados em *racks*
- Cada computador é chamado de *nó*
- Nós são conectados através de *rede de alta velocidade*
- Nós podem ter *aceleradores* (GPUs, FPGAs etc)

Software

- Geralmente máquinas Linux (SSH, LDAP, ...)
- Gerenciador para agendar jobs do usuário (Slurm, PBS etc).
- Modelo de programação paralela para distribuir a execução (MPI, OpenMP, CUDA etc)

Arquitetura do Cluster

42	24-Port GigE	48-Port GigE	
41		Redundant PS for GigE	
40	IB	IB	
39	48-Port GigE	IB	
38	48-Port GigE	IB	
37	Redundant PS for GigE	IB	
36			
35			
34			4 Compute-Nodes
33			n017-n020
32			4 Compute-Nodes
31			n013-n016
30			4 Compute-Nodes
29			n009-n012
28			4 Compute-Nodes
27			n005-n008
26		UV20	4 Compute-Nodes
25		Service3	n001-n004
24	Login-Node	gn028	gn024
23	Service1	gn027	gn023
22	Head-Node	gn026	gn022
21	Kahuna-bkp	gn025	gn021
20		Head Node	gn020
19	Console	Kahuna-adm	gn019
18	gn046	gn060	gn018
17	gn045	gn059	gn017
16	gn044	gn058	gn016
15	gn043	gn057	gn015
14	gn042	gn056	gn014
13	gn041	gn055	gn013
12	gn040	gn054	gn012
11	gn039	gn053	gn011
10	gn038	gn052	gn010
9	gn037	gn051	gn009
8	gn036	gn050	gn008
7	gn035	gn049	gn007
6	gn034	gn048	gn006
5	gn033	gn047	gn005
4	gn032		gn004
3	gn031	Storage	gn003
2	gn030	Service2	gn002
1	gn029		gn001

Total	85 Nodes

Nós de Login

Usuários se conectam, compilam seus programas e enviam jobs.

Nós de Computação

Onde os jobs são despachados, pode conter aceleradores.

Recursos de Armazenamento

Sistema de arquivos compartilhado entre todos os nós (NFS), pode conter sistema de arquivos distribuído adicional (*ex.* Lustre) para fácil acesso e replicação.

Rede de Interconexão

Ethernet, interconexão de alta velocidade (*ex.* Infiniband).

Tarefas OpenMP

Programação baseada em diretiva

- Anotar sentenças para instruir o compilador como gerar código paralelo.
- Paralelismo de dados *vs.* paralelismo de tarefas
  - Paralelismo de dados torna difícil expressar relações complexas.
  - Paralelismo de tarefas lida com heterogeneidade do problema.

```
void do_something_parallel(int N) {
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        do_something_serial(i, N);
    }
}
```

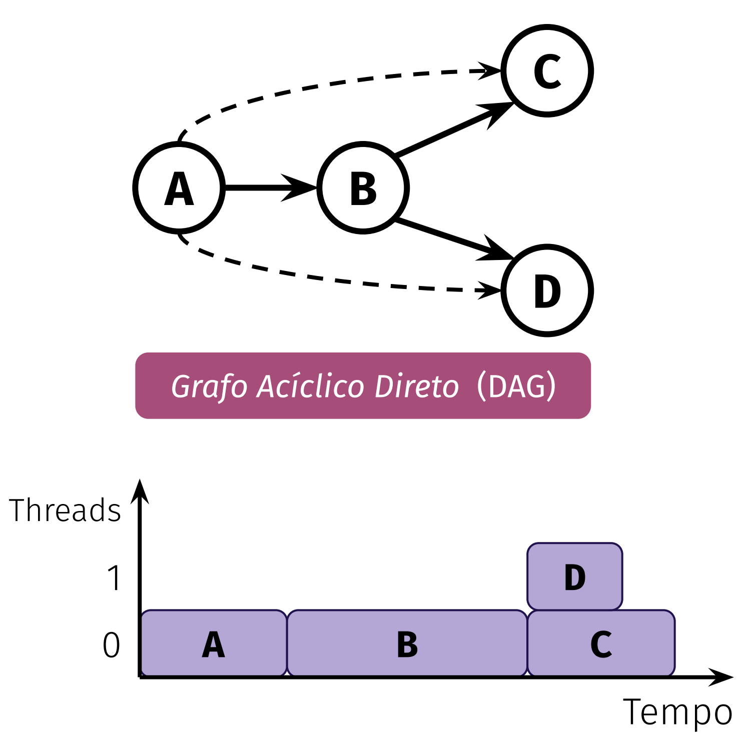
Paralelismo de Tarefas


```
#pragma omp parallel (1)
#pragma omp single (2)
{
    #pragma omp task (3)
    foo(x); (4)
    #pragma omp task
    bar(x);
} (5)
```

1. Inicializa (fork) todas as threads
2. Roda o bloco ao lado apenas na thread de controle
3. A thread de controle cria as tarefas
4. As threads trabalhadoras executam o código das tarefas
5. Compilador insere uma barreira implícita (por padrão)

Dependências de Tarefas & Grafo de Tarefas


```
#pragma omp task depend(out: x)
task_A(&x);
#pragma omp task depend(inout: x)
task_B(&x);
#pragma omp task depend(in: x)
task_C(&x);
#pragma omp task depend(in: x)
task_D(&x);
```



Exemplo • Fibonacci


```c
int fib_serial(int n) {
    if (n < 2)
        return 1;
    int i = fib_serial(n - 1);
    int j = fib_serial(n - 2);
    return i + j;
}
```

```c
int main(int argc, char** argv) {
    // snip...
    fib_serial(input);
    // snip...
}
```

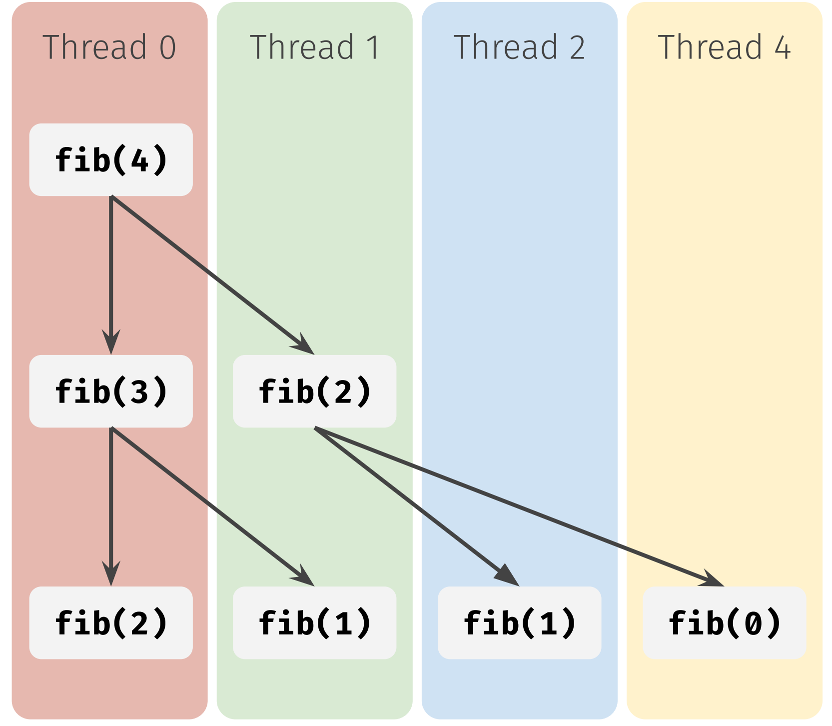
Exemplo • Fibonacci


```c
int fib_parallel(int n) {
    if (n < 2)
        return 1;
     int i, j;
    #pragma omp task shared(i)
    i = fib_parallel(n - 1);
    #pragma omp task shared(j)
    j = fib_parallel(n - 2);
    #pragma omp taskwait
    return i + j;
}
```

```c
int main(int argc, char** argv)
{
   // snip...
    #pragma omp parallel
    #pragma omp single
    fib_parallel(input);
   // snip...
}
```

Exemplo • Fibonacci


```
int fib_parallel(int n) {
    if (n < 2)
        return 1;
     int i, j;
    #pragma omp task shared(i)
    i = fib_parallel(n - 1);
    #pragma omp task shared(j)
    j = fib_parallel(n - 2);
    #pragma omp taskwait
    return i + j;
}
```



OpenMP Target Offloading

A partir do OpenMP 4.5, há uma diretiva para enviar tarefas para aceleradores.

- Cada dispositivo tem sua própria memória.
- Compilador insere chamadas para mover os dados.
- **`#pragma omp target`**

```
#pragma omp parallel
#pragma omp single
{
  #pragma omp target
  {
    printf("Hello from accelerator!");
  }
}
```

Host ⟷ Accelerator

OMPC Runtime

OMPC Runtime

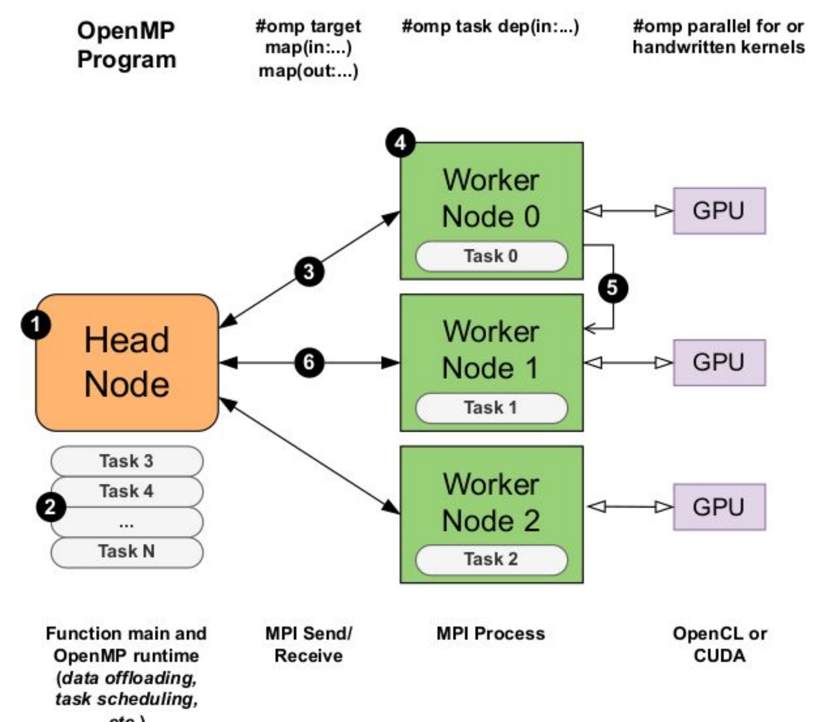
Runtime do OmpCluster (OMPC):

- Aproveita a infraestrutura do OpenMP do LLVM.
- Implementado como um acelerador.
- Usa uma camada MPI para trocar mensagens entre os dispositivos (nós).
- Agendador de tarefas e tolerância à falhas.

```
$ clang -fopenmp -fopenmp-targets=x86_64-pc-linux-gnu \
    program.cc -o executable
$ mpirun -np 3 ./executable
```

Modelo de Execução do OMPC

- Arquitetura distribuída
  - Nó *head* gerencia as tarefas (balanceamento de carga, distribuição de dados etc)
  - Nós trabalhadores executam tarefas
- Execução em 6 passos:
  1. Execução do programa
  2. Geração das tarefas
  3. Distribuição das tarefas
  4. Execução das tarefas
  5. Comunicação entre nós
  6. Recuperar resultado



Cláusulas OpenMP


```
#pragma omp target [clause [clause]*]
```

Cláusulas com suporte ✅

- **nowait**
- **map**(*list*)
- **depend**(*list*)
- **private**(*list*)
- **if**(*expr*)

Cláusulas sem suporte

- **device**(*integer*)
- **in_reduction**(*list*)
- **is_device_ptr**(*list*)
- **allocate**(*list*)
- **uses_allocators**(*list*)

A cláusula **nowait** faz a região target ser executada assincronamente.

- Caso omitida, o host (nó head) vai esperar a execução finalizar antes de proceder.

```
#pragma omp target nowait
auto output = do_work(input);

#pragma omp target nowait
{
   auto output = do_work(input);
   if (output > 0) {
       log(output);
   }
}
```

ATENÇÃO
Você deve sempre usar **nowait** com OMPC!

Cláusulas OpenMP • depend

A cláusula **depend** impõem restrições no agendamento de tarefa (ex. ordenação).

depend(*dep-type*: *locator*)

- Dep-type: **in, out, inout**.

ATENÇÃO
OMPC sempre espera dependências **reais**!

```
int value;

#pragma omp target nowait depend(out: value) \
        map(from: value)
value = producer();

#pragma omp target nowait depend(inout: value) \
        map(tofrom: value)
value = square(value);

#pragma omp target nowait depend(inout: value) \
        map(tofrom: value)
value = add_one(value);

#pragma omp target nowait depend(in: value) \
        map(tofrom: value)
display(value);
```

Cláusulas OpenMP • map

A cláusula **map** especifica quais e quando os dados devem ser transferidos entre os dispositivos.

map(*map-type*: *locator*)

- map-type: **alloc, to, from, tofrom, delete**.

```
int array[N];
int result[4];
const int BS = N/4;

for (int i = 0; i < 4; i++) {
   #pragma omp target nowait \
       map( to: array[i*BS:(i+1)*BS]) \
       map( from: result[i]) \
       depend(out: result[i])
   result[i] = reduce_sum(array);
}
```

Cláusulas OpenMP • `private`

As cláusulas **`private`** e **`firstprivate`** alocam cópias privadas de uma variável para cada tarefa (cópias não inicializadas ou inicializadas).

`private( ... )`
`firstprivate( ... )`

```
int i = 99;

#pragma omp target nowait private(i)
printf("i = %d\n", i); // i = 182348

#pragma omp target nowait firstprivate(i)
printf("i = %d\n", i); // i = 99
```

Gerenciamento de Dados do OpenMP


```
#pragma omp target nowait map( ... )
{
    // Code block for the device
}
```

Mapeia dados para o dispositivo que permanecem “vivos” apenas no escopo do bloco. Adicionalmente, roda o bloco como uma tarefa target no **dispositivo**.

```
#pragma omp target data map( ... )
{
    // Code block for the host
}
```

Mapeia dados para o dispositivo que permanecem “vivos” apenas no escopo do bloco. O código dentro do bloco é executado no **host**.

```
#pragma omp target enter data map( ... )
// Code block for the host
#pragma omp target exit data map( ... )
```

Mapeia dados no *enter* e desmapeia no *exit*. O código entre essas sentenças é executado no **host**.

Gerenciamento de Dados do OpenMP

Por que as regiões de dados são úteis?

- Mapeia uma vez e pode usar em todo lugar.
- Gerenciamento de dados automático sem nó head.
- Enfileira a transferência de dados antes de executar a tarefa.
- Muito importante para performance.

```
#pragma omp enter data nowait \
        map( to: var) depend( out: var)

// Do work on the host

#pragma omp target nowait \
        map( tofrom: var) \
        depend( inout: var)
execute_task(var);

// Do more work on the host

#pragma omp exit data nowait \
        map( from: var) depend( inout: var)
```

OpenMP Cluster • depend

Dependências reais

- **in** (or no dep.): **read-only**
- **out, inout**: **read-write**

```
// Writes to both arrays
void foo(int *bufferA, int *bufferB)

// Always used together
int bufferA[SIZE], bufferB[SIZE];

// Logic dependency: one buffer, dependencies
on pointer
#pragma omp target nowait depend(out: bufferA)
foo(bufferA, bufferB);

// Real dependency: both buffers,
dependencies on data
#pragma omp target nowait depend(out: bufferA,
bufferB)
foo(bufferA, bufferB);
```

Revisão

Parte I

- OpenMP usa diretivas para instruir como compilador deve paralelizar o código.
- OmpCluster extende o modelo para executar tarefas em cluster de HPC.

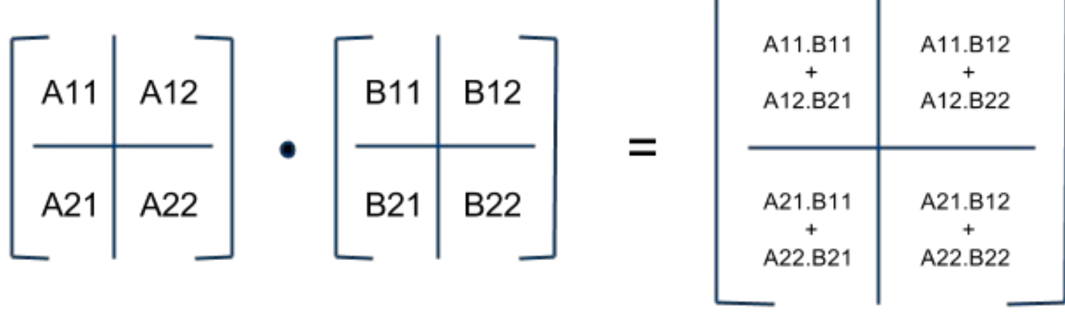
Parte II

Visão Geral

- Multiplicação de matrizes
- Adição de vetores

Exemplo • Multiplicação Matricial

- Multiplicação de matrizes por blocos
  - Calcula a multiplicação para cada bloco de forma independente
  - Algoritmo ideal para arquiteturas distribuidas
- A distribuição dos dados é modelada pelo programador
  - Segmentação explícita das matrizes em blocos



Exemplo

Multiplicação Matricial **Convencional**

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      C[i*N + j] += A[i*N + k] * B[k*N + j];
```

Exemplo

Multiplicação Matricial
Por Blocos

```
// Pointers to current blocks
float *BA,  *BB, *BC;

// Iterate over blocks
for(int i = 0; i < N/BS; ++i)
  for(int j = 0; j < N/BS; ++j)
    for(int k = 0; k < N/BS; ++k) {
      BC = C.GetBlock(i, j);
      BA = A.GetBlock(i, k);
      BB = B.GetBlock(k, j);

      // Multiply blocks
      for(int ii = 0; ii < BS; ++ii)
        for(int jj = 0; jj < BS; ++jj)
          for(int kk = 0; kk < BS; ++kk)
            BC[ii + jj*BS] += BA[ii + kk*BS] * BB[kk + jj*BS];
}
```

Exemplo

Multiplicação Matricial

OmpCluster

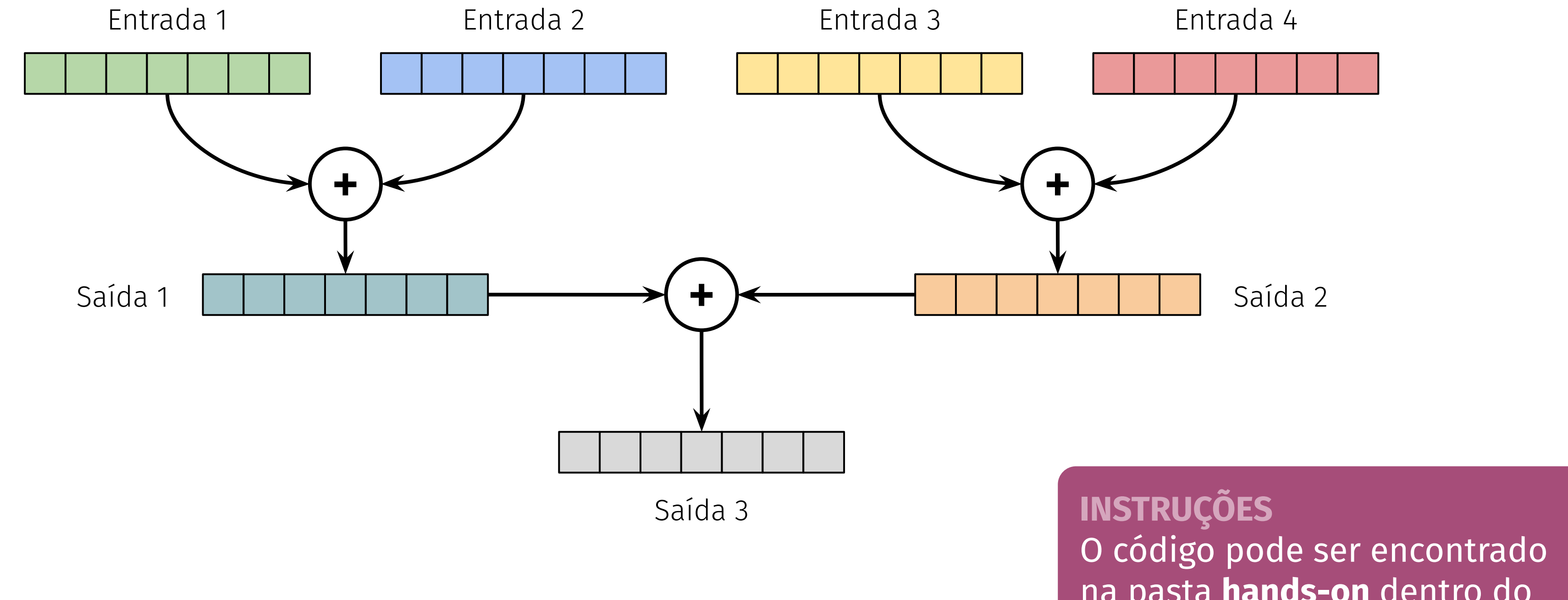
```
#pragma omp parallel -->> Don't forget ;)
#pragma omp single   -->> Ditto!
{
  float *BA,  *BB, *BC;

  for(int i = 0; i < N/BS; ++i)
    for(int j = 0; j < N/BS; ++j)
      for(int k = 0; k < N/BS; ++k) {
        BC = C.GetBlock(i, j);
        BA = A.GetBlock(i, k);
        BB = B.GetBlock(k, j);

        #pragma omp target nowait \ -->> List dependencies and data maps
                depend(in: BA[0], BB[0]) depend(inout: BC[0]) \
                map(to: BA[:BS*BS], BB[:BS*BS]) map(tofrom: BC[:BS*BS])
        #pragma omp parallel for -->> Uses multithreading locally
        for(int ii = 0; ii < BS; ++ii)
          for(int jj = 0; jj < BS; ++jj)
            for(int kk = 0; kk < BS; ++kk)
              BC[ii + jj*BS] += BA[ii + kk*BS] * BB[kk + jj*BS];
  }
}
```

Começando


```
$ git clone https://gitlab.com/ompcluster/conferences/wscad-2023.git
$ cd wscad-23
$ sudo docker run -v $PWD:/wscad-23 -it ompcluster/runtime bash
$ cd wscad-23
$ mkdir build && cd build
$ export CC=clang CXX=clang++
$ cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
$ make
$ mpirun -np N ./bin/matmul <MATRIX-SIZE> <BLOCK-SIZE>
```

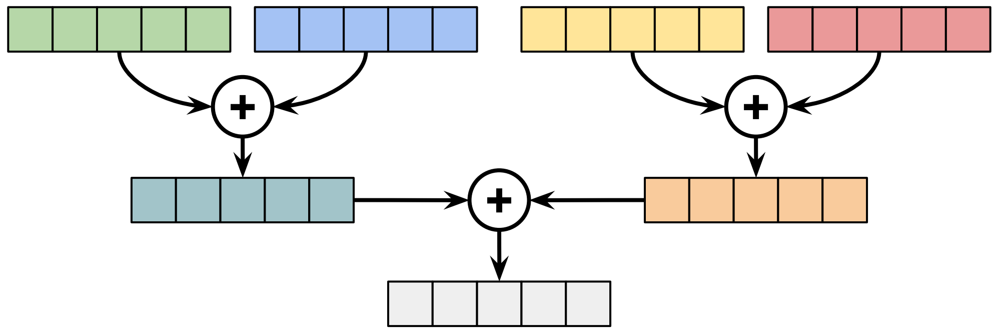
Exercício Prático

Hands-on • Adição de vetores



INSTRUÇÕES
O código pode ser encontrado na pasta **hands-on** dentro do repositório da apresentação

Passos



1. Encapsule cada operação de adição em uma tarefa
2. Adicione as dependências e mapeamentos corretamente
3. Teste!

OMPC Runtime: Profile e Debug

Coletando informações de Debug


```
$ export LIBOMPTAGET_INFO=1
$ mpirun -np 5 /path/to/program

...
Libomptarget device 0 info: Entering OpenMP kernel at reduce-sum.cc:34:7 with 4 arguments:
Libomptarget device 0 info: tofrom(result[i])[4]
Libomptarget device 0 info: firstprivate(i)[4] (implicit)
Libomptarget device 0 info: to(array[i * BS:(i + 1) * BS])[16384]
Libomptarget device 0 info: firstprivate(BS)[4] (implicit)
...
```

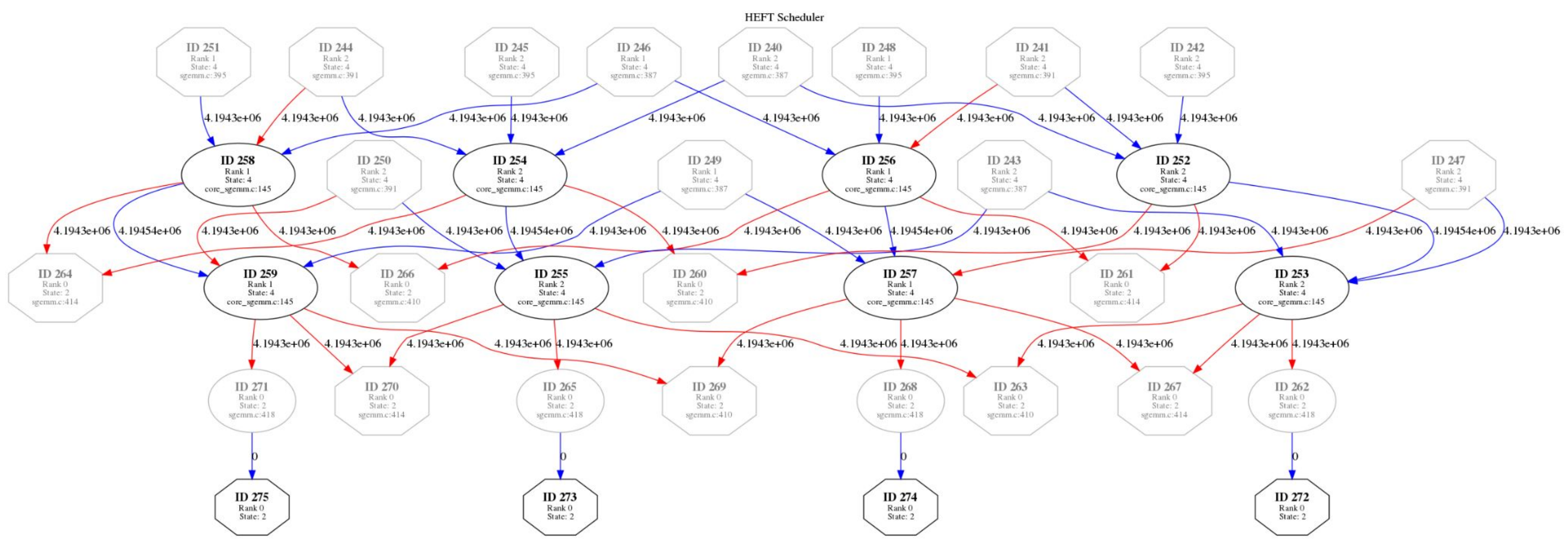
AVISO ⚠️

O programa **precisa** ser compilado com os símbolos de debug para os nomes aparecerem.

Coletando Grafo de Tarefas


```
$ export OMPCLUTER_TASK_GRAPH_DUMP_PATH="program"
$ mpirun -np 5 /path/to/program
$ ls *.dot

program_graph_0.dot
program_graph_1.dot

$ dot -Tpdf program_graph_1.dot > graph.pdf
```

Coletando Grafo de Tarefas

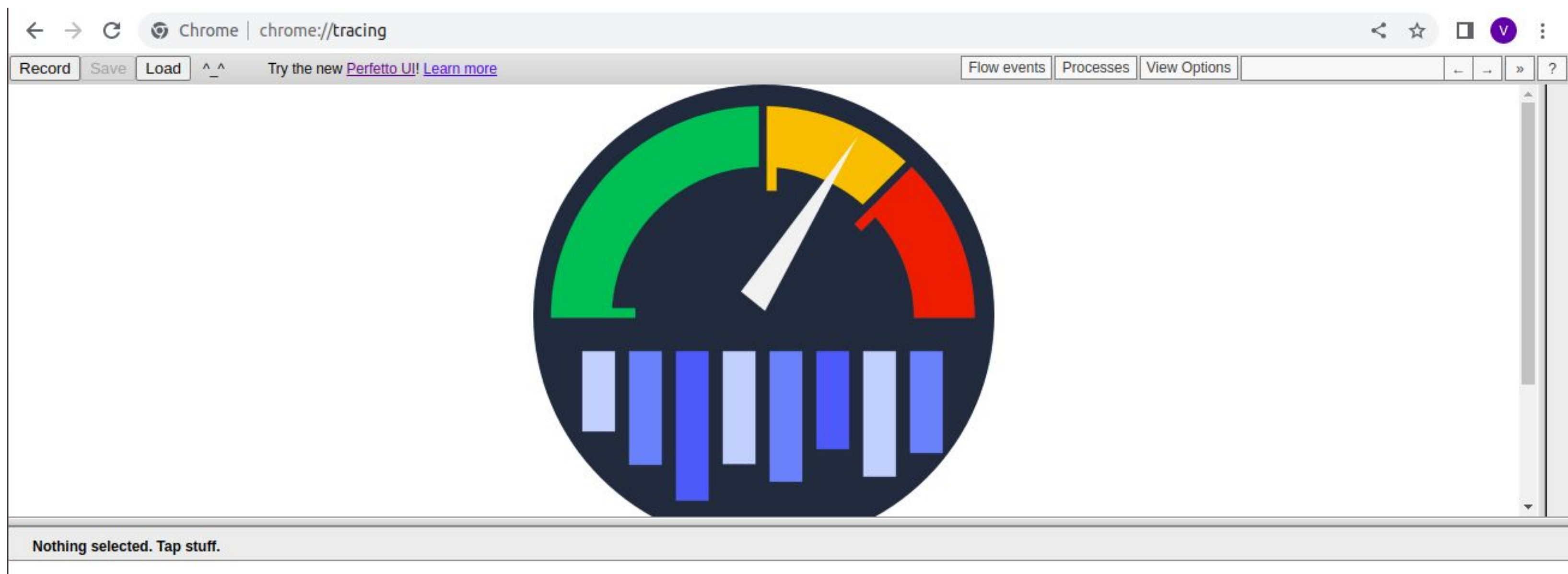


Coletando Profile

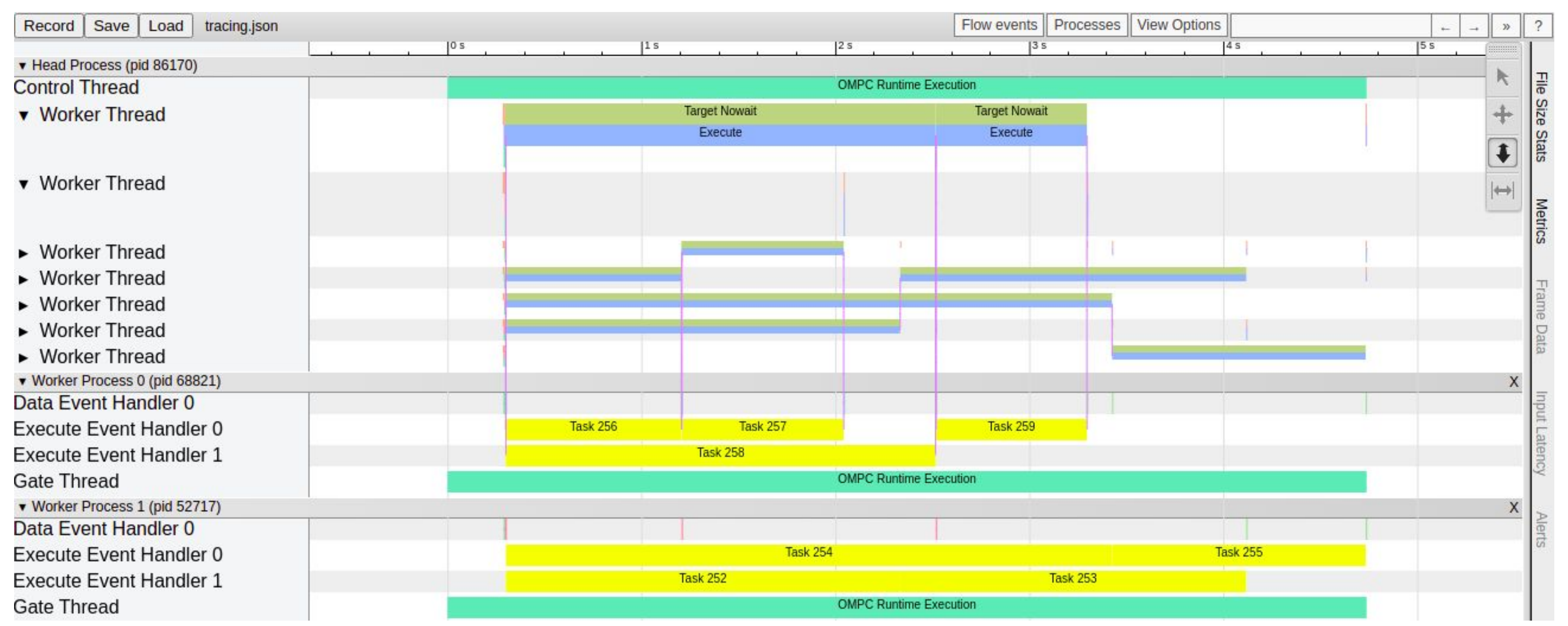

```
$ export OMPCLUSTER_PROFILE="./program_"
$ mpirun -np 5 /path/to/program
$ ls *.json

program_head_process.json
program_worker_process_0.json
program_worker_process_1.json
program_worker_process_2.json
program_worker_process_3.json

$ ompcbench merge -c program_ --developer -o developer.json
$ ompcbench merge -c program_ -o user.json
```

AVISO ⚠️
Traces podem ficar muito grandes.

Coletando Profile



Coletando Profile



Ajustando as Threads do Runtime


```
$ export OMP_NUM_THREADS=8
$ export OMPCLUSTER_NUM_EXEC_EVENT_HANDLERS=8
$ export OMPCLUSTER_NUM_DATA_EVENT_HANDLERS=2
$ export LIBOMP_NUM_HIDDEN_HELPER_THREADS=16  //num_worker*exec_evt_handler
$ mpirun -np 5 /path/to/program

...
Libomptarget device 0 info: Entering OpenMP kernel at reduce-sum.cc:34:7 with 4 arguments:
Libomptarget device 0 info: tofrom(result[i])[4]
Libomptarget device 0 info: firstprivate(i)[4] (implicit)
Libomptarget device 0 info: to(array[i * BS:(i + 1) * BS])[16384]
Libomptarget device 0 info: firstprivate(BS)[4] (implicit)
...
```

AVISO ⚠️
HIDDEN HELPER THREADS controla a contagem de tarefas simultâneas em andamento

Revisão

Parte II

- Blocked matrix multiplication executed in an HPC cluster with minimal effort.
- Vector addition is even more simple.
- Nós vimos como perfilar e debugar código do OMPC.

Obrigado! Perguntas?

Este trabalho é apoiado pela **Petrobras** sob concessão 2018/00347-4, pelo **Centro de Computação em Engenharia e Ciências** (CCES) e **Fundação de Amparo à Pesquisa do Estado de São Paulo** (FAPESP) sob concessão 2020/08475-1, e pelo **Conselho Nacional de Desenvolvimento Científico e Tecnológico** (CNPq) sob concessão 402467/2021-3.