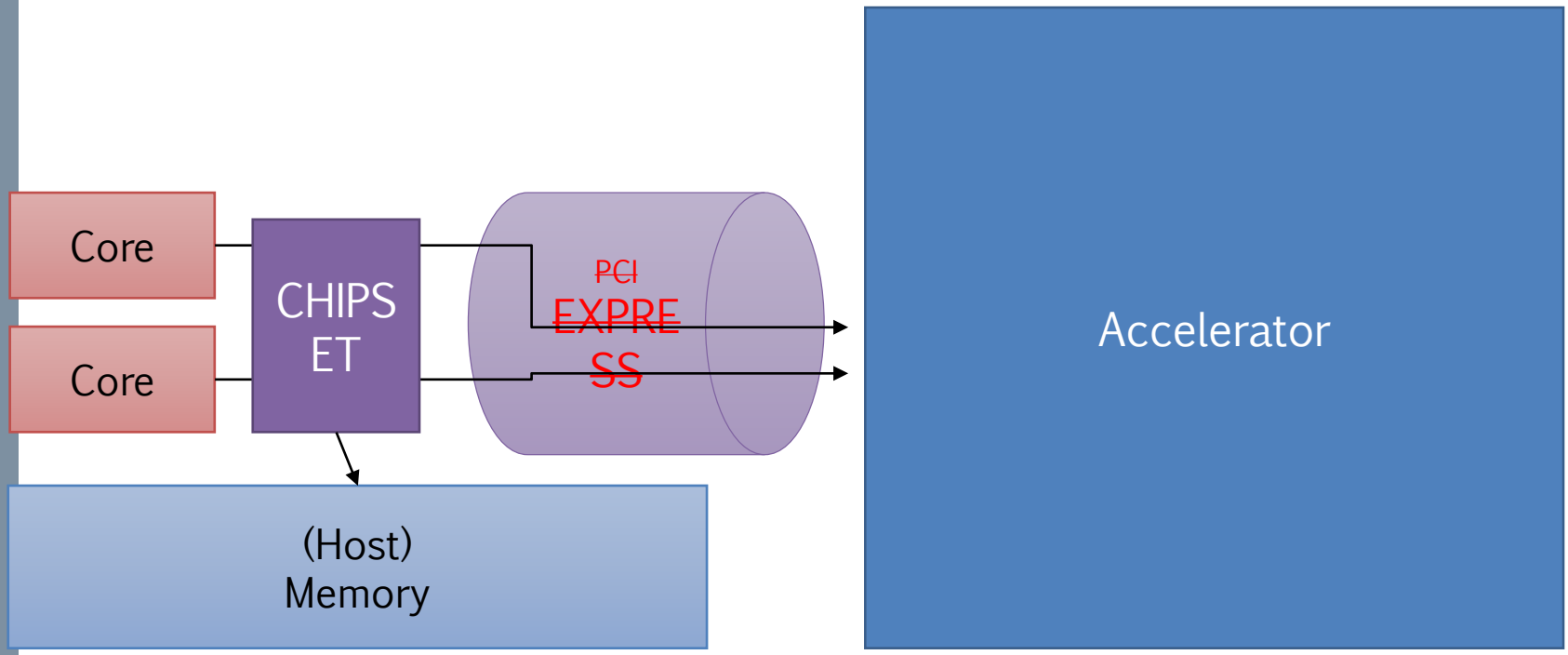
Heterogeneous programing with OpenMP

Paolo Burgio
paolo.burgio@unimore.it

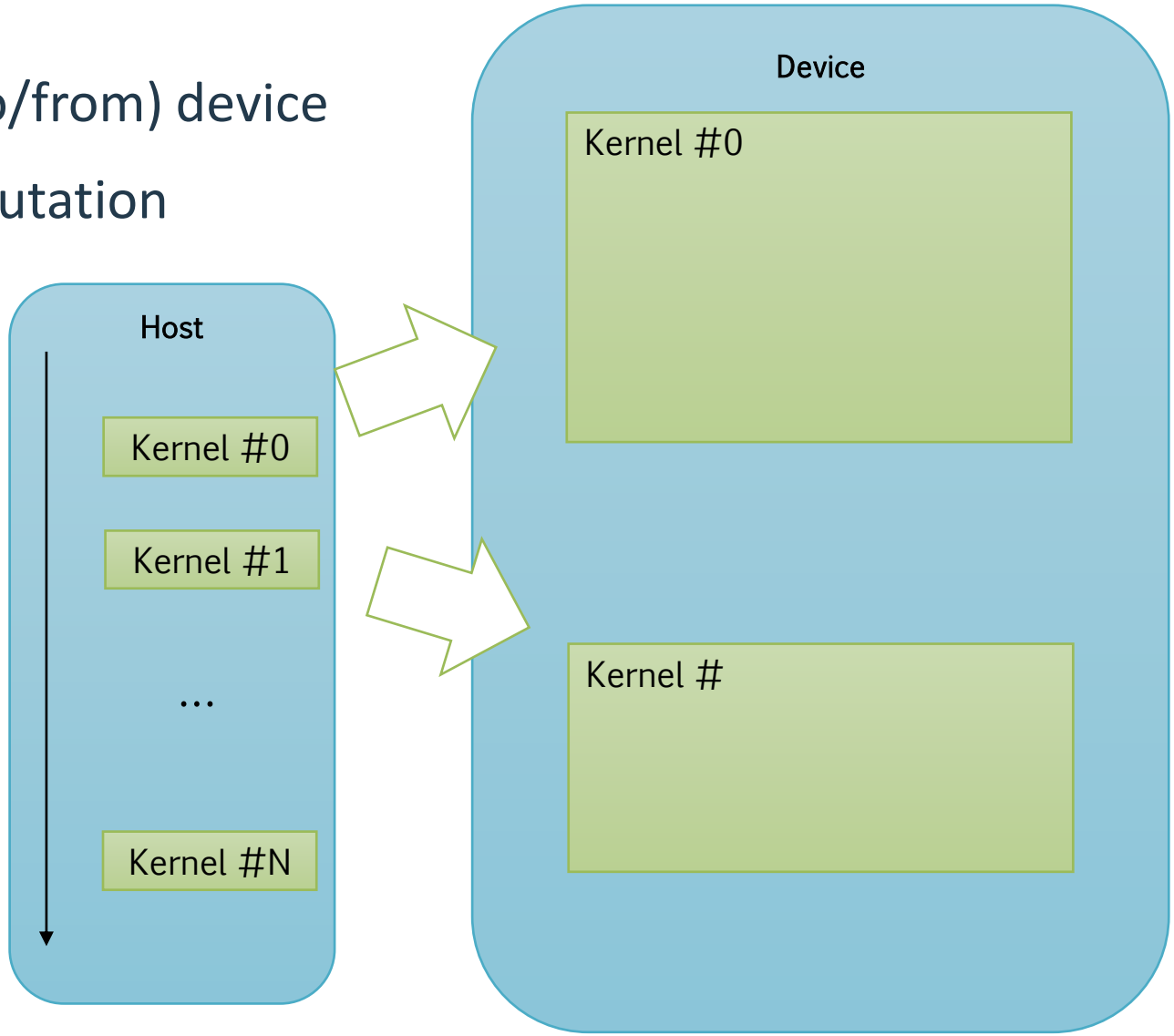
~~GPGPU~~ Heterogeneous programming

- We need a programming model that provides
  1. Simple and generic offloading subroutines
  2. An easy way to write code which runs on thousand threads (might be arranged in clusters)
  3. A way to exploit the NUMA hierarchy (if any)



Offload-based programming

- Move data (to/from) device
- Offload computation
- (Sync)
- OpenMP 4.*
  - (OpenMP 5)



Move DATA (and code)

target data constructs

```
#pragma omp target data [clause [[,]clause]...] new-line
  structured-block
```

Where clauses can be:

```
if([ target :] scalar-expression)
device(integer-expression)
map([[map-type-modifier[,]] map-type: ] list)
is_device_ptr(list)
```

- Map variables to a device data environment for the extent of the region
- The binding set is the generating task

target data constructs

```
#pragma omp target [enter|exit] data [clause [[,]clause]...] new-line
```

Where clauses can be:

```
if([ target :] scalar-expression)
device(integer-expression)
map([[map-type-modifier[,]] map-type: ] list)
depend(dependency-type: list)
nowait
```

- Map variables to a device data environment ~~for the extent of the region~~
- Stand-alone directive
- The binding set is the generating task

target update constructs

```
#pragma omp target update [clause [[,]clause]...] new-line
```

Where clauses can be:

```
if([ target update :] scalar-expression)
device(integer-expression)
depend(dependency-type: list)
nowait
```

- Makes the corresponding list items in the device data environment, consistent with their original list items, according to the specified motion clauses
- Stand-alone directive
- The binding set is the generating task

declare target construct

```
#pragma omp declare target new-line

  declaration-definition-seq

#pragma omp declare target end new-line
```

- Map variables device data environment
- The binding set is the generating task

Threading

`target` construct

```
#pragma omp target [clause [[,]clause]...] new-line
  structured-block
```

Where clauses can be:

```
if([ target :] scalar-expression)
device(integer-expression)
private(list)
firstprivate(list)
map([[map-type-modifier[,]] map-type: ] list)
is_device_ptr(list)
defaultmap(tofrom:scalar)
nowait
depend(dependence-type: list)
```

- Map variables to a device data environment for the extent of the region
- And executes a target task on the region
- The binding set is the generating task

`teams` constructs

```
#pragma omp teams [clause [[,]clause]...] new-line
 structured-block
```

Where clauses can be:

```
num_teams(integer-expression)
thread_limit(integer-expression)
default(shared|none)
private(list)
firstprivate(list)
shared(list)
reduction(reduction-identifier: list)
```

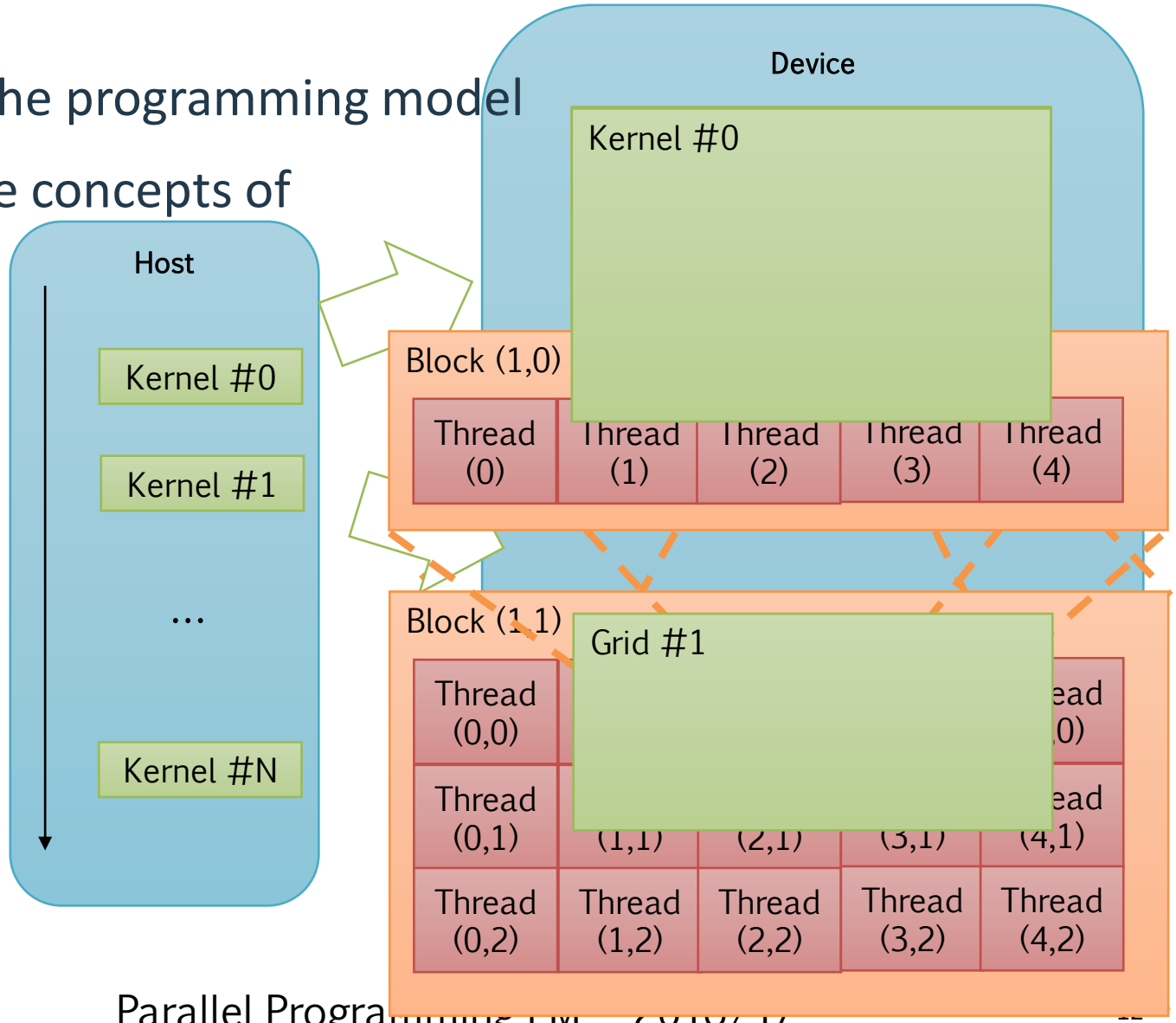
- Creates a league of thread teams and the master thread of each team executes the region.

UNLIKE PRAGMA OMP PARALLEL!!

- Executes on device
- Only master thread executes

CUDA-based programming

- Esposed in the programming model
- Based on the concepts of
  - Grid(s)
  - Block(s)
  - Thread(s)



distribute constructs

```
#pragma omp distribute [clause [[,]clause]...] new-line
  for-loops
```

Where clauses can be:

```
private(list)
firstprivate(list)
lastprivatefirstprivate(list)
collapse(n)
dist_schedule(kind[, chunk_size])
```

› specifies that the iterations of one or more loops will be executed by the thread teams in the context of their implicit tasks.

distribute simd constructs

```
#pragma omp distribute simd [clause [[,]clause]...] new-line
  for-loops
```

Where clauses can be:

```
private(list)
firstprivate(list)
lastprivatefirstprivate(list)
collapse(n)
dist_schedule(kind[, chunk_size])
```

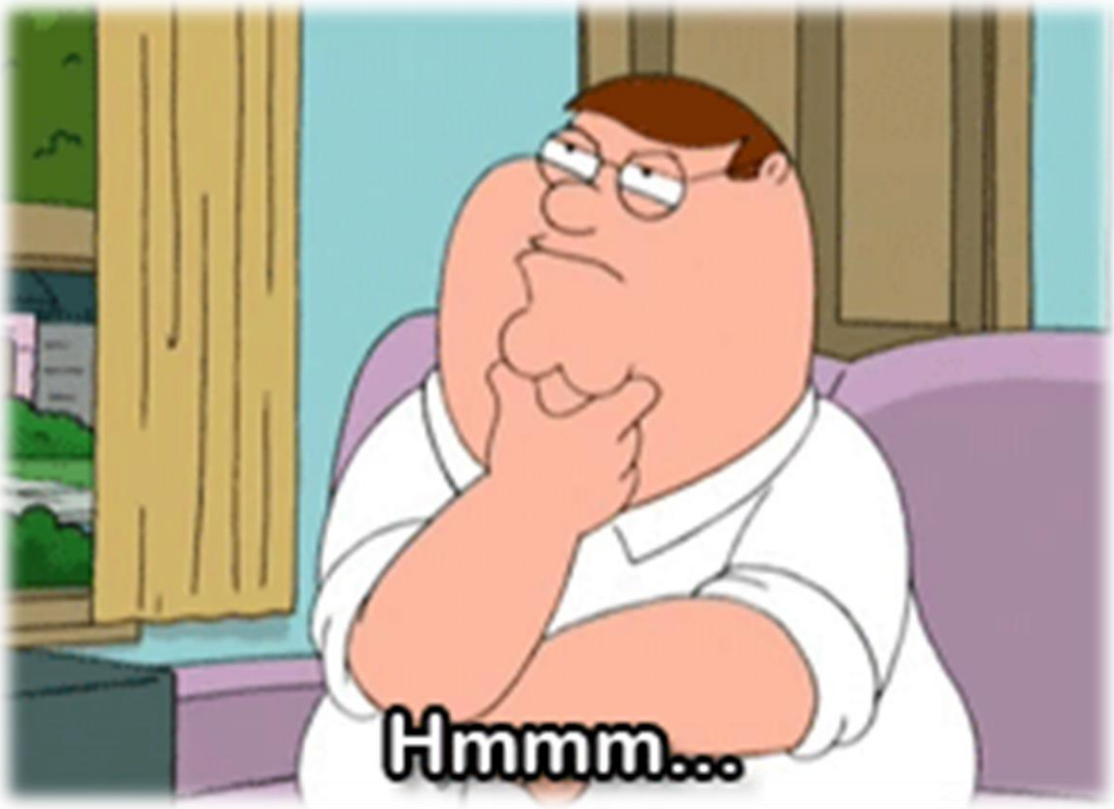
- Specifies that the iterations of one or more loops will be executed by the thread teams in the context of their implicit tasks.
- The loop will be executed in SIMD fashion (GPGPUs…)

`distribute parallel for [simd]`

- Same as previous, but threads can cross the boundaries of teams

WAT??

- Can run on multiple clusters
- …and on SIMD / GPU fashion


Hmmm...

…different devices

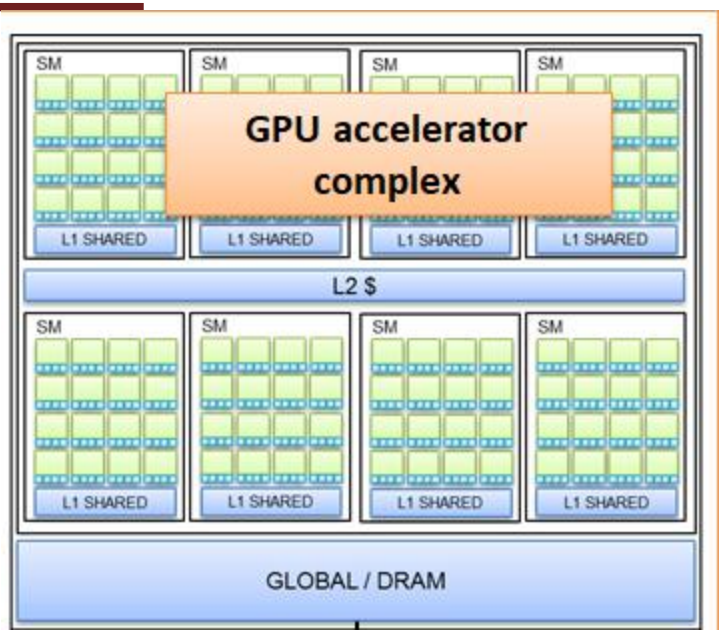
```
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (int i=0; i<N; i++)
  B[i] += sin(B[i]);
```

Intel Xeon Phi

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams (num_blocks) num_threads (bsize)
#pragma omp distribute
for (int i=0; i<N; i+= bsize )
  #pragma omp parallel for firstprivate (i)
  for (b = i; b < i+ bsize; b++)
    B[b] += sin(B[b]);
```

NVIDIA GP-GPU

NVIDIA GPGPU



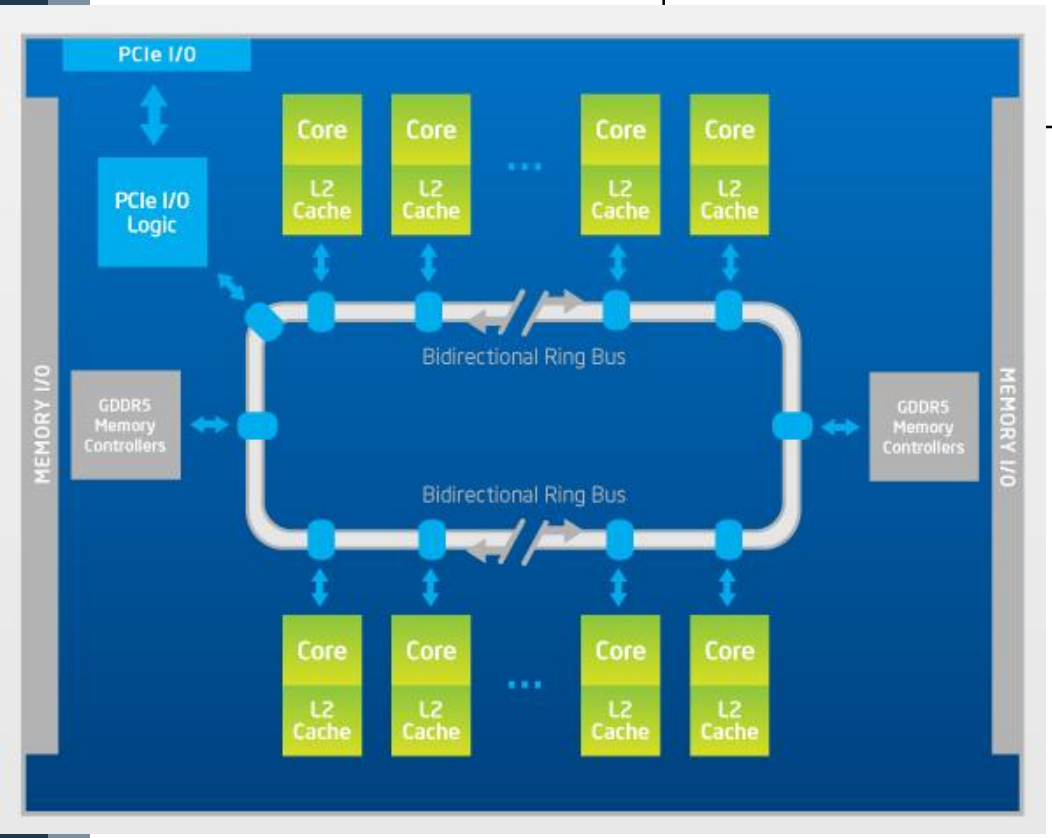

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams (num_blocks) num_threads (bsize)
#pragma omp distribute
for (int i=0; i<N; i+= bsize )
  #pragma omp parallel for firstprivate (i)
  for (b = i; b < i+ bsize; b++)
    B[b] += sin(B[b]);
```

NVIDIA GP-GPU

Intel Xeon Phi

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (int i=0; i<N; i++)
  B[i] += sin(B[i]);
```

Intel Xeon Phi

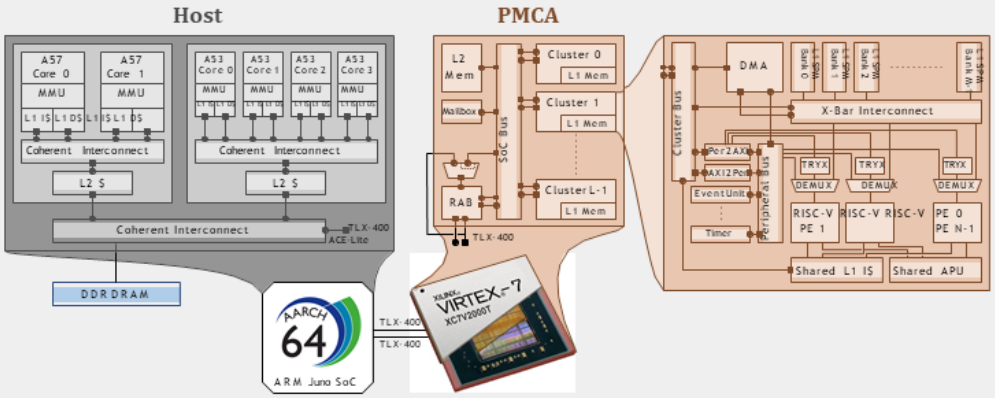


PULP

ETH

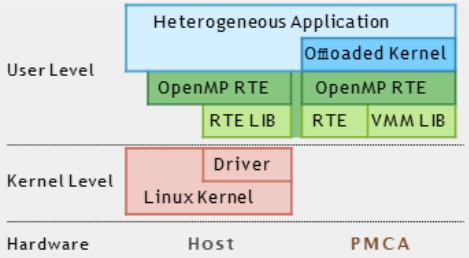
HERO: Open-Source Heterogeneous Research Platform

Heterogeneous Hardware Architecture



Heterogeneous Software Stack

- single-source, single-binary cross compilation toolchain
- OpenMP 4.5
- shared virtual memory for Host and PMCA



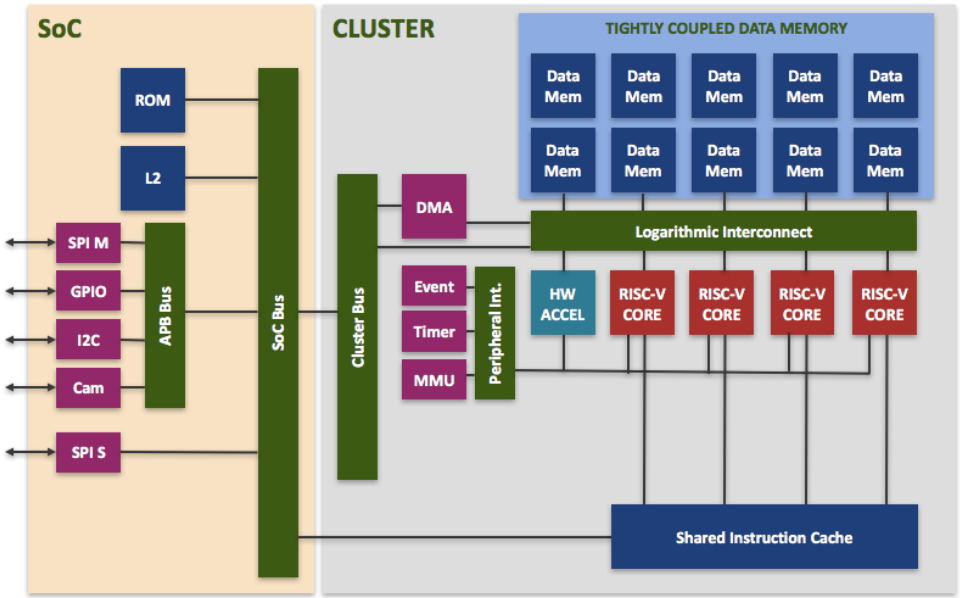
Profiling and automated verification solutions

PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

PULP - An Open Parallel Ultra-Low-Power Processing-Platform

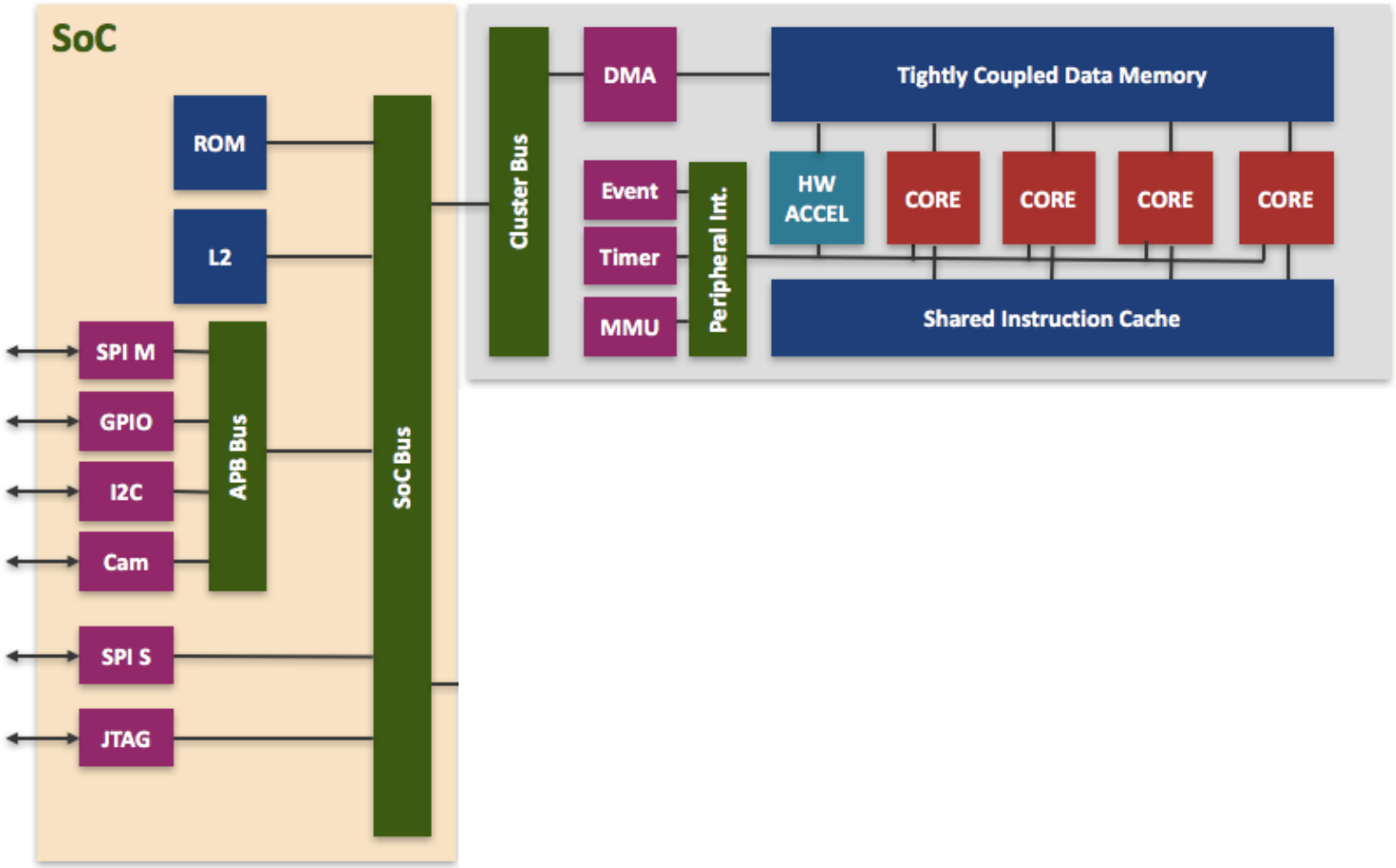
This is a joint project between the Integrated Systems Laboratory (IIS) of ETH Zurich and the Energy-efficient Embedded Systems (EEES) group of UNIBO to develop an **open**, **scalable Hardware and Software** research platform with the goal to break the pJ/op barrier within a power envelope of a few *mW*.

The **PULP platform** is a multi-core platform achieving leading-edge energy-efficiency and featuring widely-tunable performance.



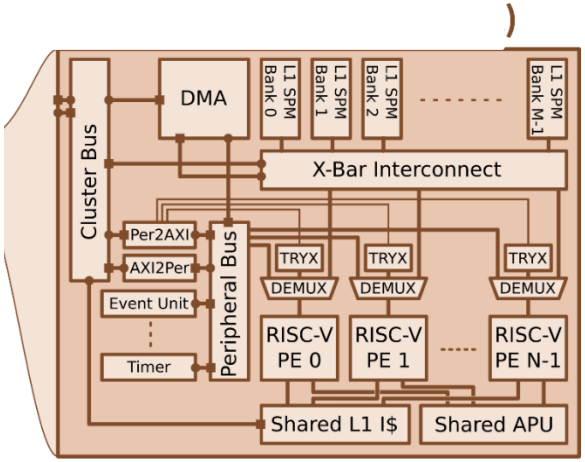
open-source
cluster-based
scalable
silicon-proven
RISC-V

PULP systems in HPC: Multi-cluster PULP accelerators



PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

Multi-Cluster PULP as an Accelerator



CA-managed IOMMU

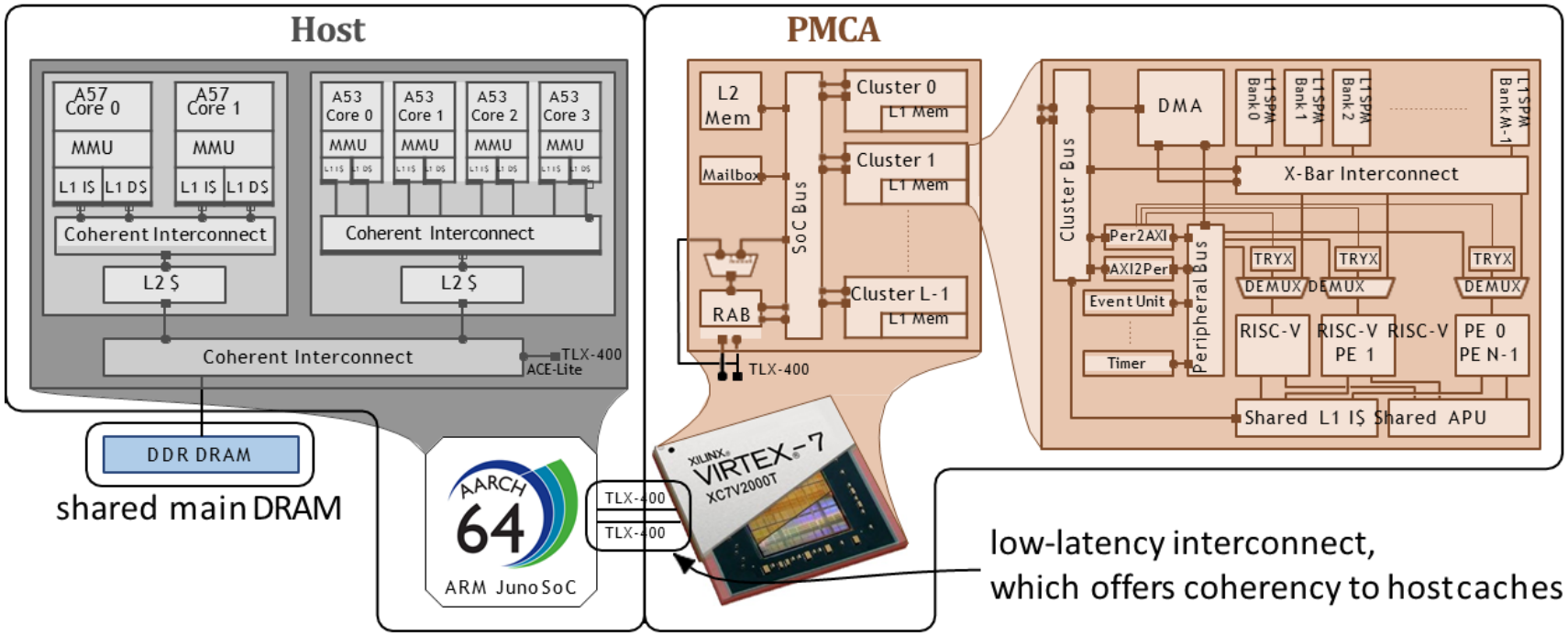
Main **challenges**:

- **Programmability**
- Zero-copy data sharing (pointer passing) via **Shared Virtual Memory**

HERO's Hardware Architecture

industry-standard, hard-macro ARM Cortex-A Host processor

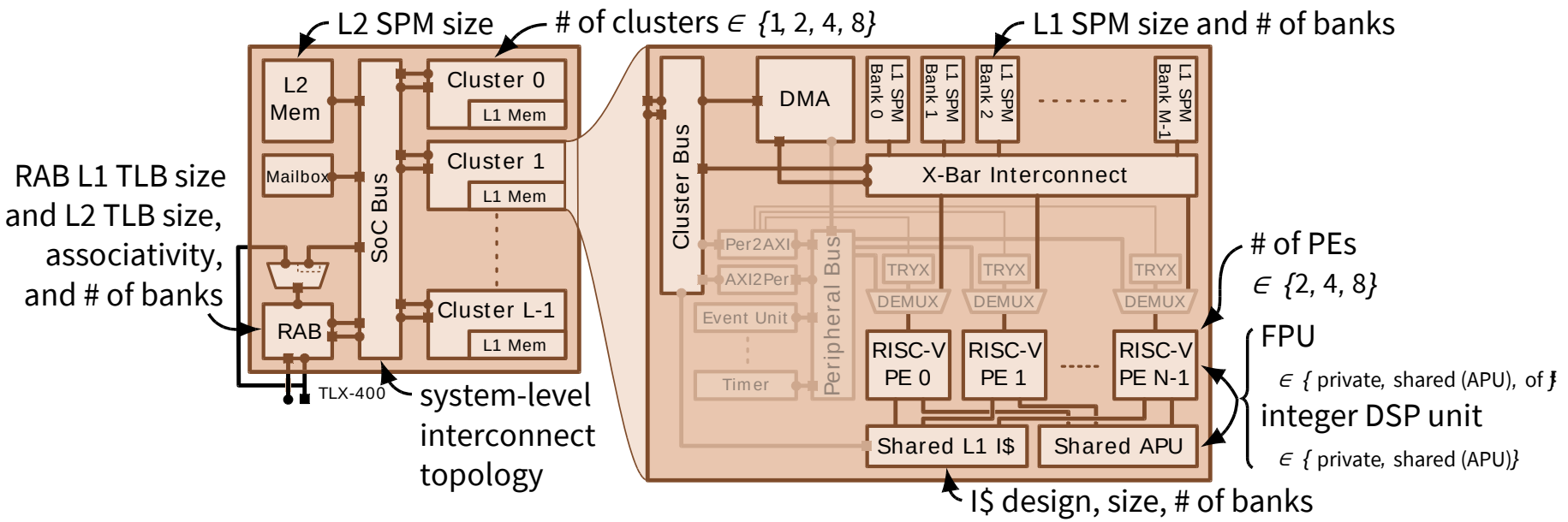
scalable, configurable, modifiable FPGA implementation of a silicon-proven, cluster-based PMCA with RISC-V PEs



HERO's hardware, as implemented on the Juno ADP.

PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

HERO is modifiable and expandable



- Up to 8 clusters, each with 8 cores
- All components are open source and written in System Verilog
- Standard interfaces (mostly AXI)
- New components can easily be added to the memory map

PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

OpenMP

- De-facto standard for shared memory programming
- Support for nested (multi-level) parallelism → *good for clusters*
- Annotations to incrementally convey parallelism to the compiler → *increased ease of use*
- Based on well-understood programming practices (shared memory, C language) → *increases productivity*

	MM		LU		CONV		PI		Histogram	
	T/E	LOC	T/E	LOC	T/E	LOC	T/E	LOC	T/E	LOC
OpenMP	+++	45	++	60	+++	70	+++	32	+	28
TBB	++	45	- - -	59	+	53	+	42	-	58
OpenCL	+	120/108	- -	120/225	-	120/186	-	120/128	- - -	120/150

Table 1: Time/effort (T/E) and number of lines of code for given bench- marks for the three frameworks. MM(Matrix Multiplication), LU(LU Factorization), CONV(Image convolution). [+++ : Least time/effort - - - : Most time/effort]

"OpenCL for programming shared memory multicore CPUs" by Akhtar Ali , Usman Dastgeer , Christoph Kessler

At the moment **GCC** supports OpenMP offloading **ONLY** to:

- **Intel Xeon Phi**
- **Nvidia PTX** (only through **OpenACC**)

PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

OpenMP target example

```
void vec_mult()
{
  double p[N], v1[N], v2[N];

# pragma omp target map(to: v1, v2)\
                    map(from: p)
  {
#   pragma omp parallel for
    for (int i = 0; i < N; i++)
      p[i] = v1[i] * v2[i];
  }
}
```

1. Initialize target device
2. Offload target image
3. Map **TO** the device mem
4. Trigger execution target region
5. Wait termination
6. Map **FROM** the device mem

The ***compiler*** outlines the code within the target region and generates a binary version for each accelerator (multi-ISA)

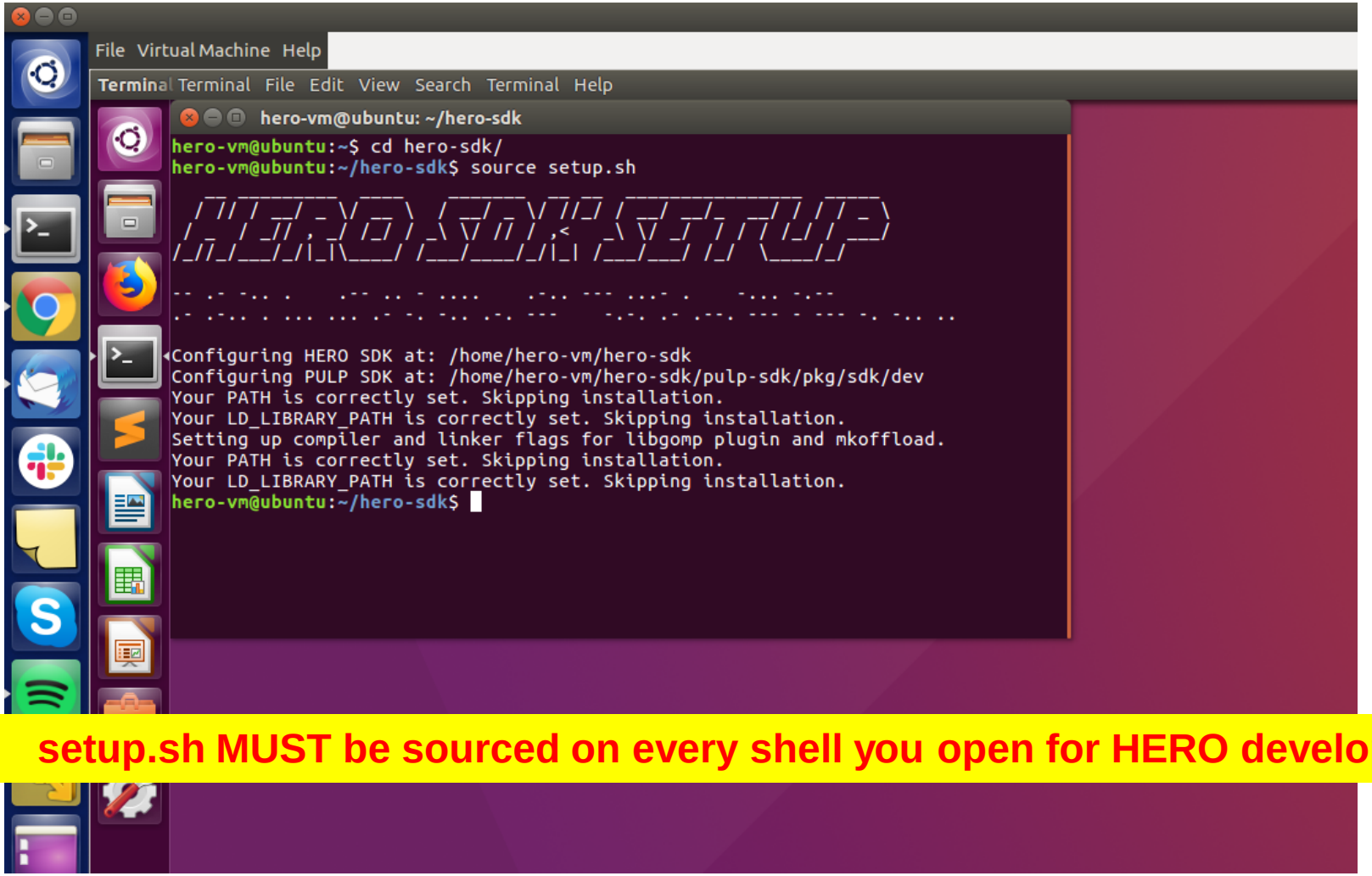
The ***runtime*** libraries are in charge to:

- ***manage*** the accelerator devices
- ***map*** the variables
- ***run/wait*** execution of target regions

Install and configure the HERO SDK (Linux)

- **Build the HERO SDK from the sources (takes around 2h)**
  - $ git clone --recursive https://github.com/pulp-platform/hero-sdk.git
  - $ cd hero-sdk && ./hero-z-7045-builder -A
- **Or Use HERO-VM (Ubuntu 16.04 VM) – No need installation or build**
  - User: hero-vm (Password: hero)
- **Configure the environment on the HERO-VM**
  - Open Terminal
  - $ cd hero-sdk
  - $ source setup.sh

Install and configure the HERO SDK (Linux) - 2



setup.sh MUST be sourced on every shell you open for HERO development!

PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

Install and configure the HERO SDK (Linux) - 3

- **HERO Board Setup**
  - You do not need to setup the board. It is already available at the IP provided during the class.

- **Additional Resources**
  - **HERO SDK Github README** (https://github.com/pulp-platform/hero-sdk)
  - **HERO Website** (https://pulp-platform.org/hero.html)
  - **HERO additional HOW-TO** (https://pulp-platform.org/hero/doc/)

Let's say Hello, Word!

- 1. Write your helloworld.c

```c
#include <omp.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <time.h>
#include <hero-target.h>

struct timespec start, stop;
double start_ns, stop_ns, exe_time;

#pragma omp declare target
void helloworld ()
{
    #pragma omp parallel
    printf("Hello World, I am thread %d of %d\n", omp_get_thread_num(), omp_get_num_threads());
}
#pragma omp end declare target

int main(int argc, char *argv[])
{
    omp_set_default_device(BIGPULP_MEMCPY);

    #pragma omp target
    helloworld();

    helloworld();
    return 0;
}
```

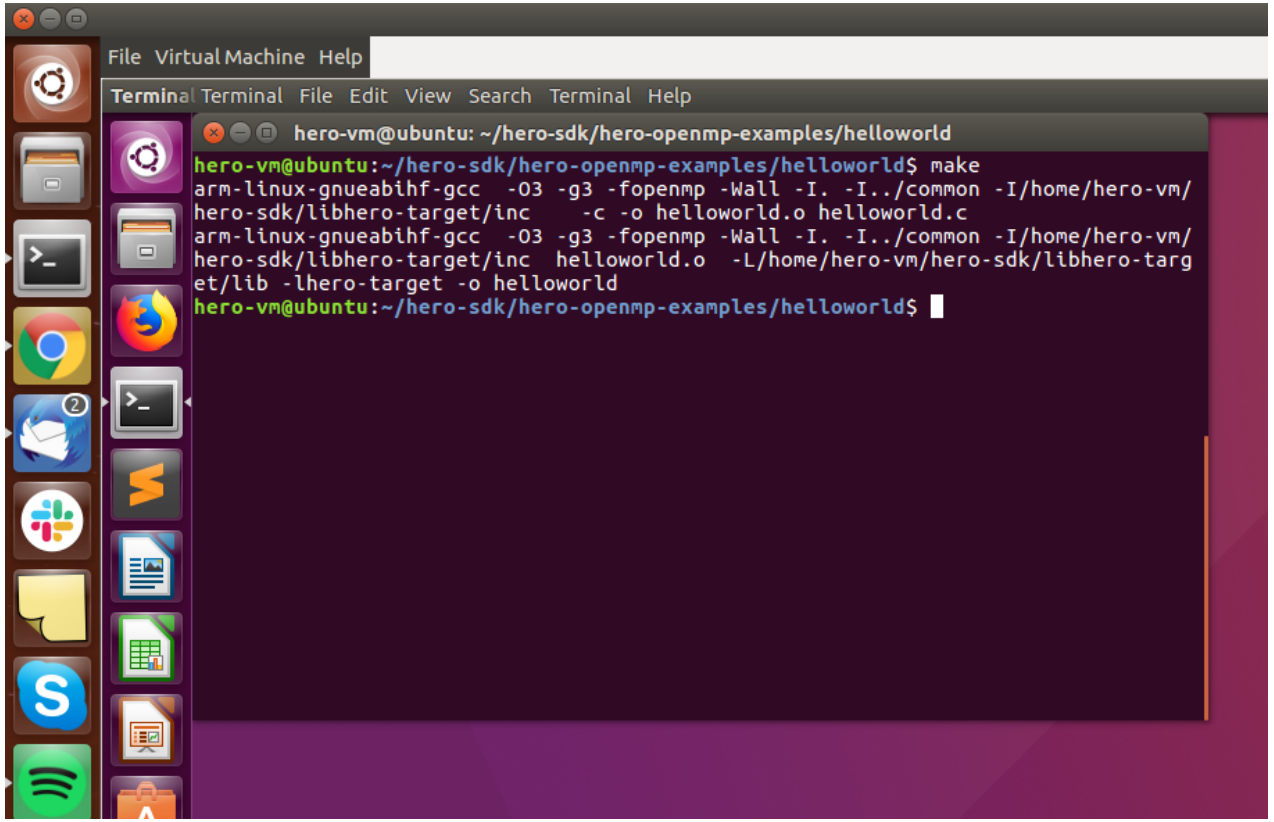
Let's say Hello, Word! - 2

- 1. Write your helloworld.c
- 2. Create a Makefile

```
CSRCS = helloworld.c
CFLAGS=
-include
${HERO_OMP_EXAMPLE_DIR}/../common/default.mk
```

Let's say Hello, Word! - 3

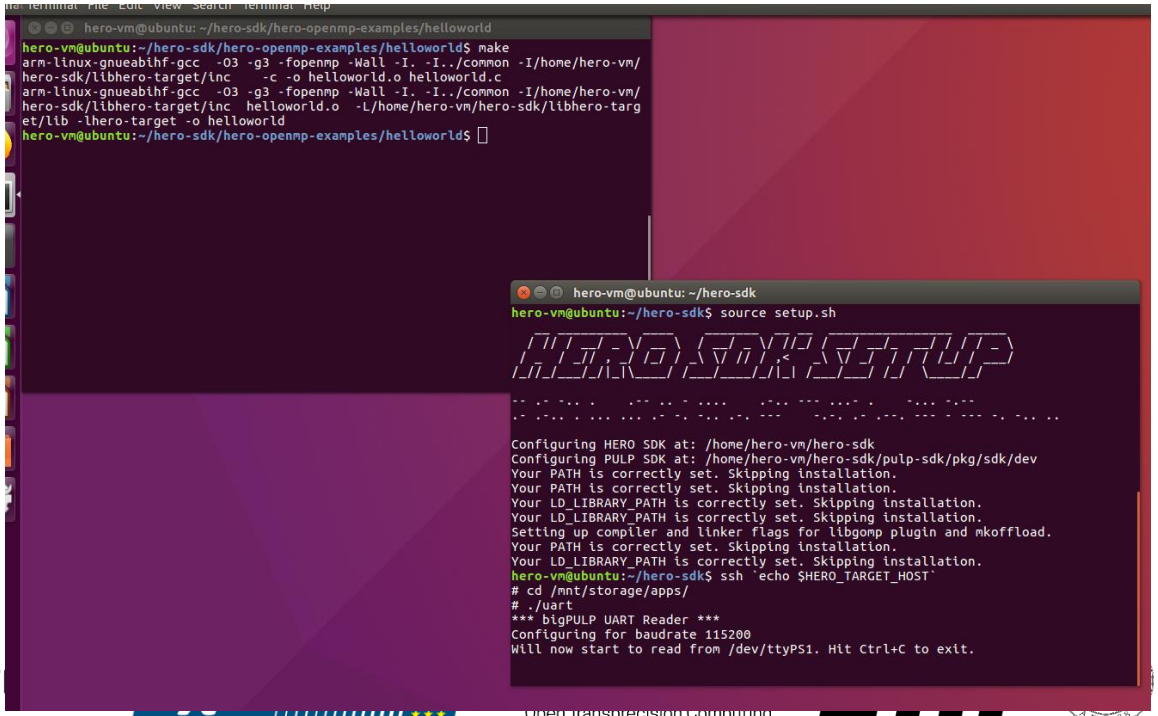
- 1. Write your helloworld.c
- 2. Create a Makefile
- 3. Build



PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

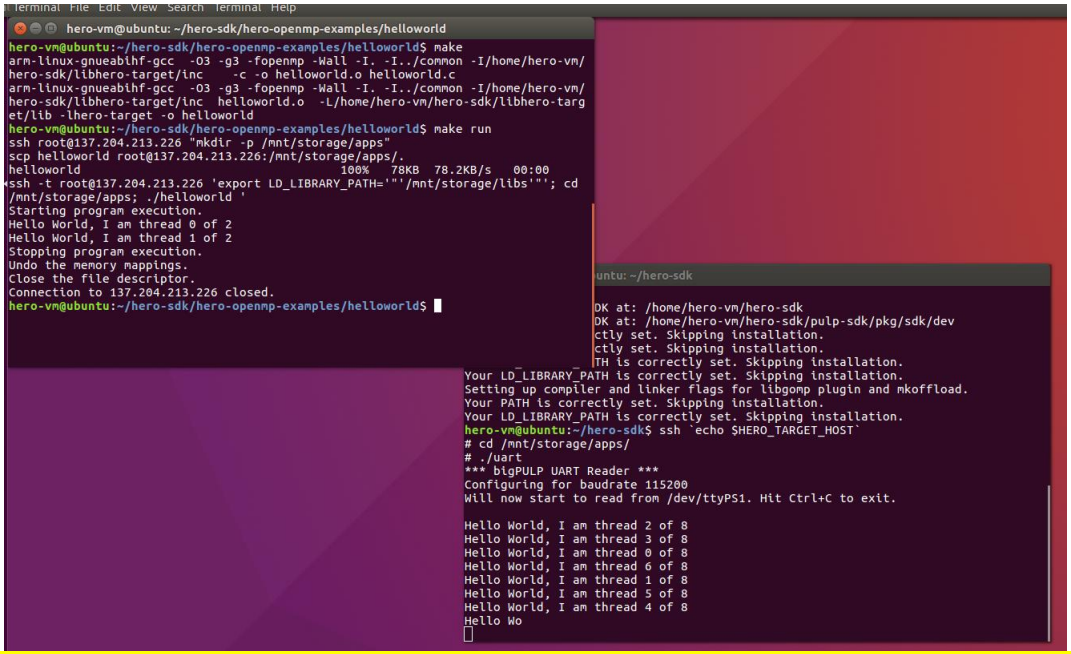
Let's say Hello, Word! - 4

- 1. Write your helloworld.c
- 2. Create a Makefile
- 3. Build
- 4. Open UART from the device. This is required when you want to use print from the accelerator device
  - Open new Terminal
  - $ cd hero-sdk && source setup.sh
  - $ ssh <host>@<board-ip>
  - $ cd /mnt/storage/apps
  - $ ./uart



Let's say Hello, Word! - 5

- 1. Write your helloworld.c
- 2. Create a Makefile
- 3. Build
- 4. Open UART from the device. This is required when you want to use print from the accelerator device
  - Open new Terminal
  - $ cd hero-sdk && source setup.sh
  - $ ssh <host>@<board-ip>
  - $ cd /mnt/storage/apps
  - $ ./uart
- 5. Run the example!



Some UART char could be missing…

PULP Multitherman ExaNoDe PRECOMP Open Transprecision Computing ETH

References

- "HPC" website
  - http://algo.ing.unimo.it/people/andrea/Didattica/HPC/
- My contacts
  - paolo.burgio@unimore.it
  - http://hipert.mat.unimore.it/people/paolob/
- Useful links
  - http://www.openmp.org
  - http://www.google.com