Work supported by

P-SOCRATES

FP7 – ICT
GRANT N° 611016

ON THE EFFECTIVENESS OF OPENMP TEAMS FOR PROGRAMMING EMBEDDED MANYCORE ACCELERATORS

Alessandro Capotondi
University of Bologna

Andrea Marongiu
Swiss Federal Institute of Technology in Zurich (ETHZ)
a.marongiu@iis.ee.ethz.ch

OUTLINE

- Motivation
  - Heterogeneous architectures
  - Manycore accelerators
- NUMA communication and OpenMP
  - Flat parallelism
  - Nested parallel regions
  - OpenMP teams and the distribute directive
  - Other constructs for non-loop parallelism
- Experimental setup and results

HETEROGENOUS MANYCORES

Ever-increasing demand for computational power has recently led to radical evolution of computer architectures

Two design paradigms have proven effective in increasing performance and energy efficiency of compute systems

>Many-cores

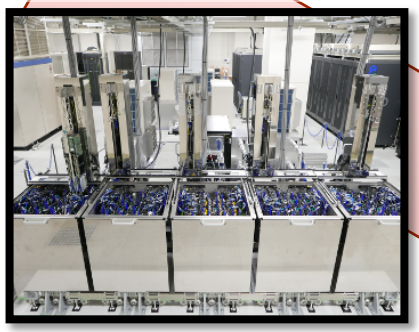
>Architectural **Heterogeneity**

A common template is one where a powerful general-purpose processor (the ***host***) is coupled to ***one or more a many-core accelerators***

HETEROGENOUS MANYCORES

THE GREEN 500

Shoubu RIKEN



Xeon E5-2618Lv3
8C 2.3GHz
Infiniband FDR
PEZY-SC

TOP 500 The List.

Titan Cray X47

Opteron 6274
16C 2.2GHz
Cray Gemini
NVIDIA K20x

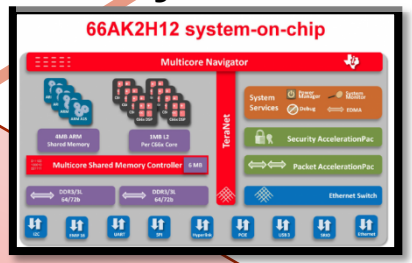
Tianhe-2

Xeon E5-2692
12C 2.2GHz
TH Express-2
Intel Xeon Phi

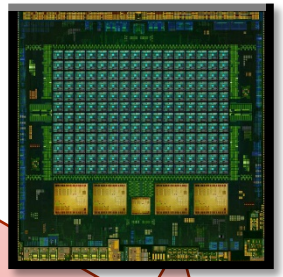
SERVER / SUPERCOMP

SoC

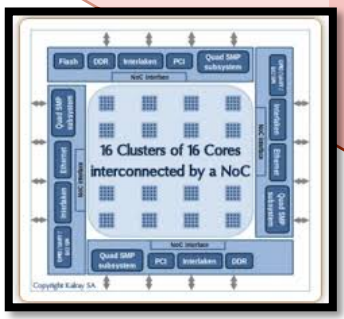
TI Keystonell



NVIDIA Tegra X1



Kalray MPPA256



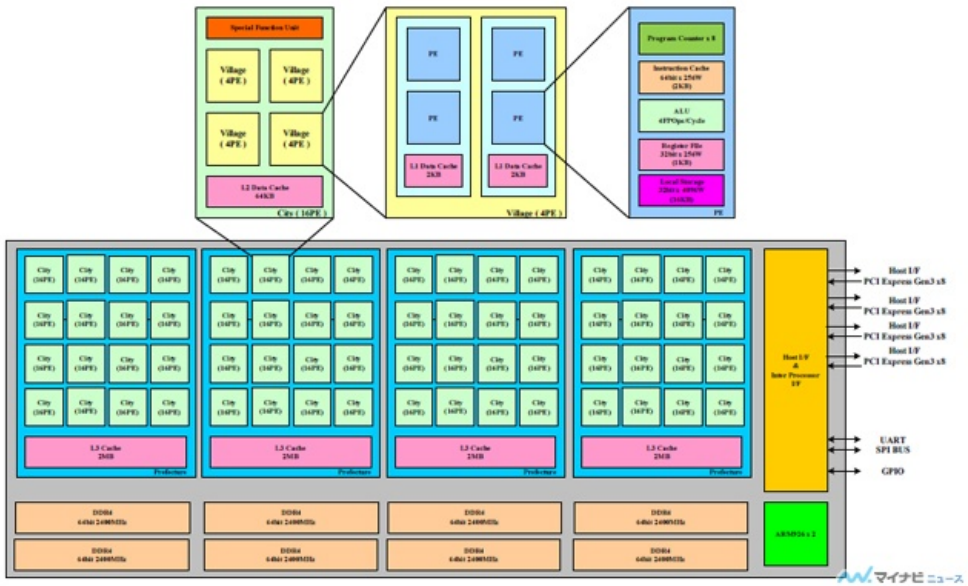
True virtually in every computing domain and at every scale!

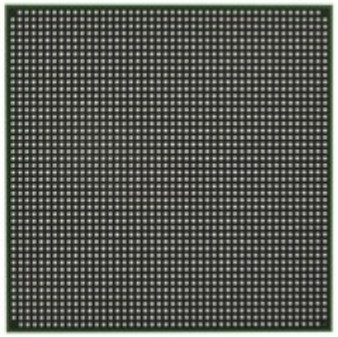
HETEROGENEOUS MANYCORES

- **Manycore accelerators share many commonalities across computing domains..**
  - Cluster-based design → NUMA
  - Explicitly managed memories

PEZY-SC 1024

- Dominates the Green500
  - Heavily cluster-based



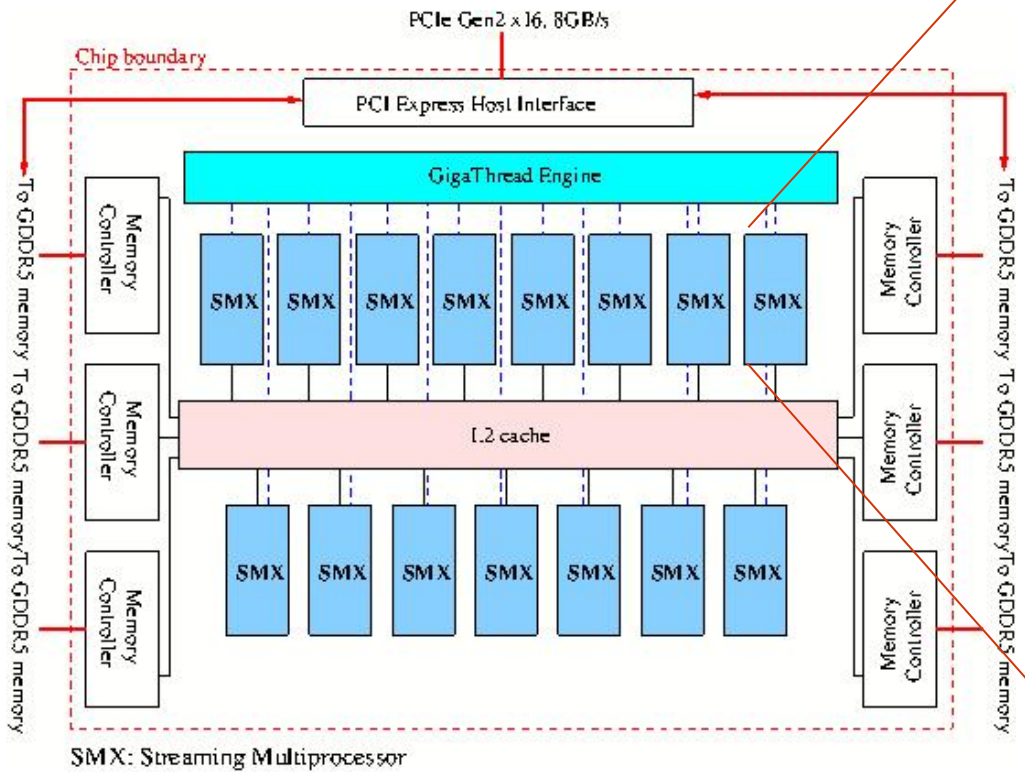


Logic Cores(PE)	1,024
Core Frequency	733MHz
Peak Performance	Floating Point Single 3.0TFlops / Double 1.5TFlops
Host Interface	PCI Express GEN3.0 x8Lane x 4Port (x16 bifurcation available) JESD204B Protocol support
DRAM Interface	DDR4, DDR3 combo 64bit x 8Port Max B/W 1533.6GB/s +Ultra WIDE IO SDRAM (2,048bit) x 2Port Max B/W 102.4GB/s
Control CPU	ARM926 2core
Process Node	28nm
Package	FCBGA 47.5mm x 47.5mm, Ball Pitch 1mm, 2,112pin

http://www.pezy.co.jp/en/products/pezy-sc.html
http://news.mynavi.jp/articles/2014/09/17/pezy/

NVIDIA K20

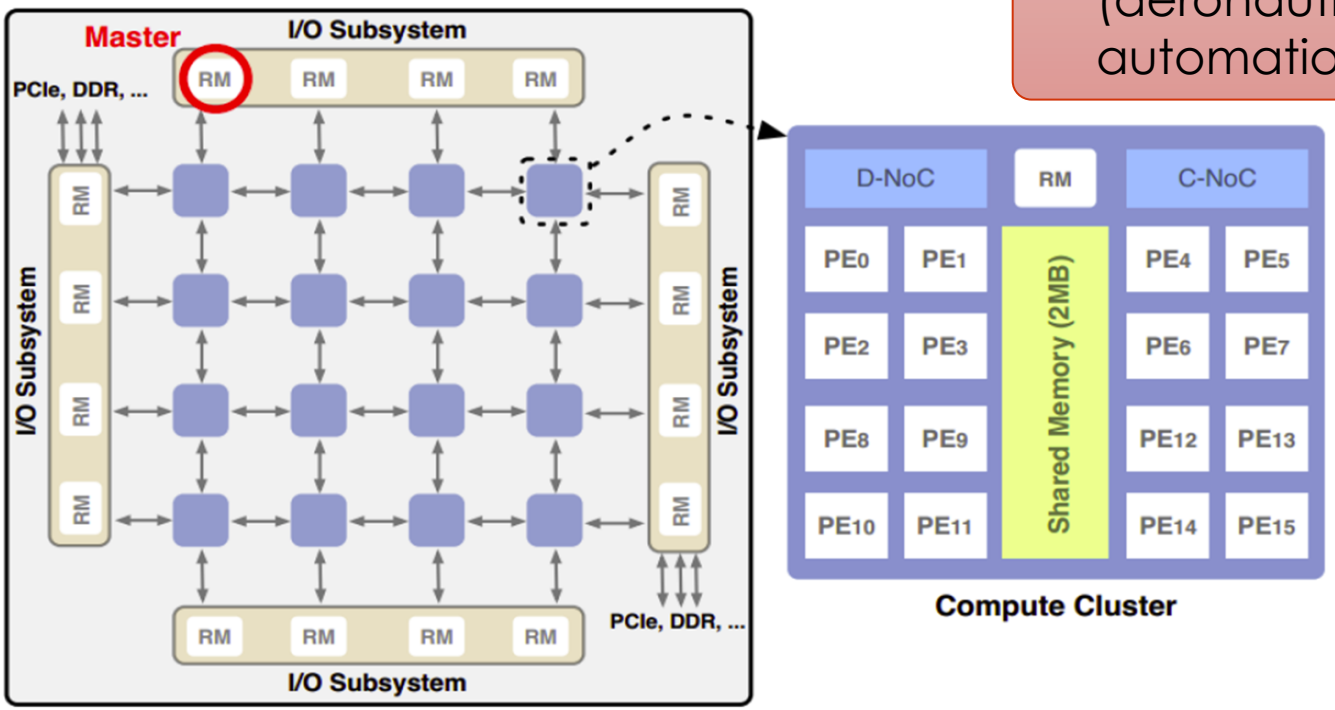
- 15 clusters (SMX) sharing L2 SPM
- 192 cores per SME, sharing 64KB L1 SPM
- 1.17 DP TFLOPS - 3.52 SP TFLOPS
- 235 W



KALRAY MPPA 256

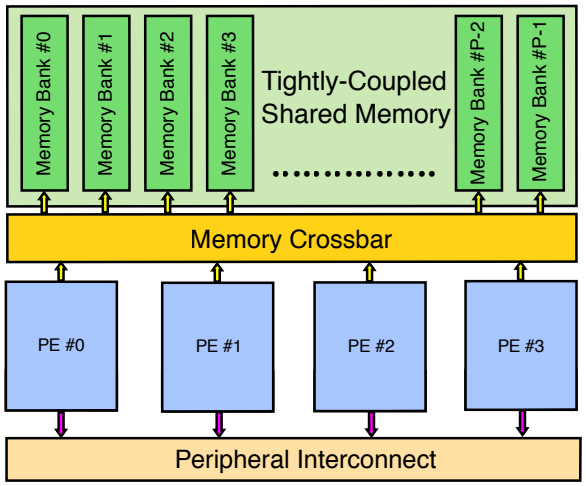
- 16 clusters
- 16 cores per cluster, sharing 2MB L1 SPM/cache
- 1.25 TFLOPS
- 25 GOPS/W – 75 GOPS/W

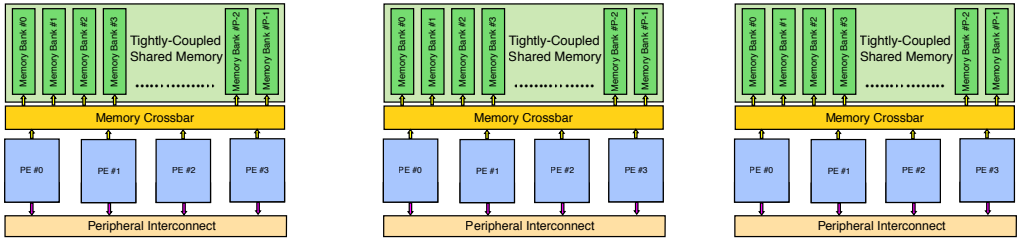
- Image, audio, signal processing
- Control commands (aeronautics, industrial automation)



STHORM

- 4 clusters
- 16 cores per cluster, sharing 256KB L1 data SPM
- 80 GOPS @ 2W





NoC

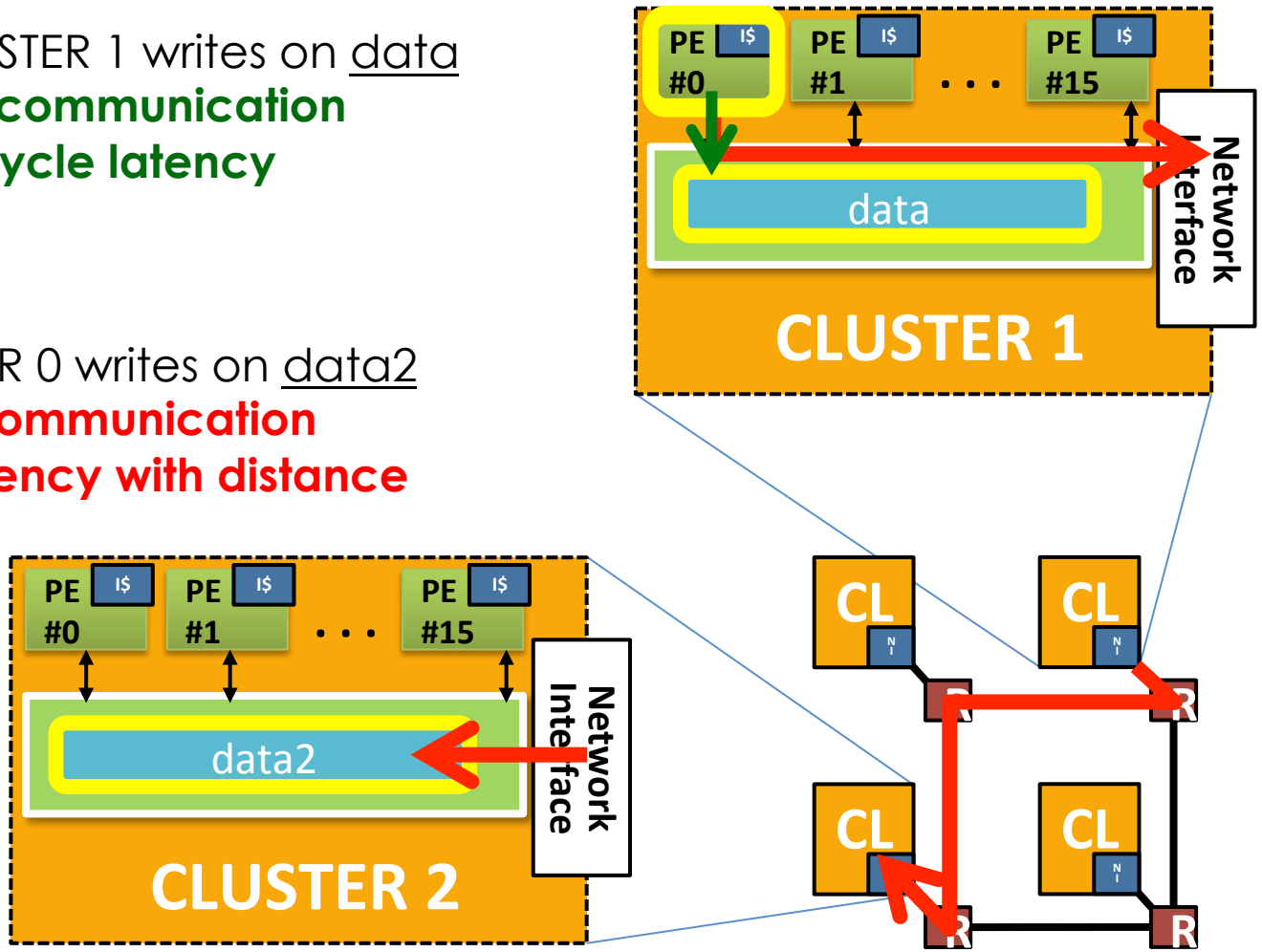
L2 SPM

AXI BRIDGE

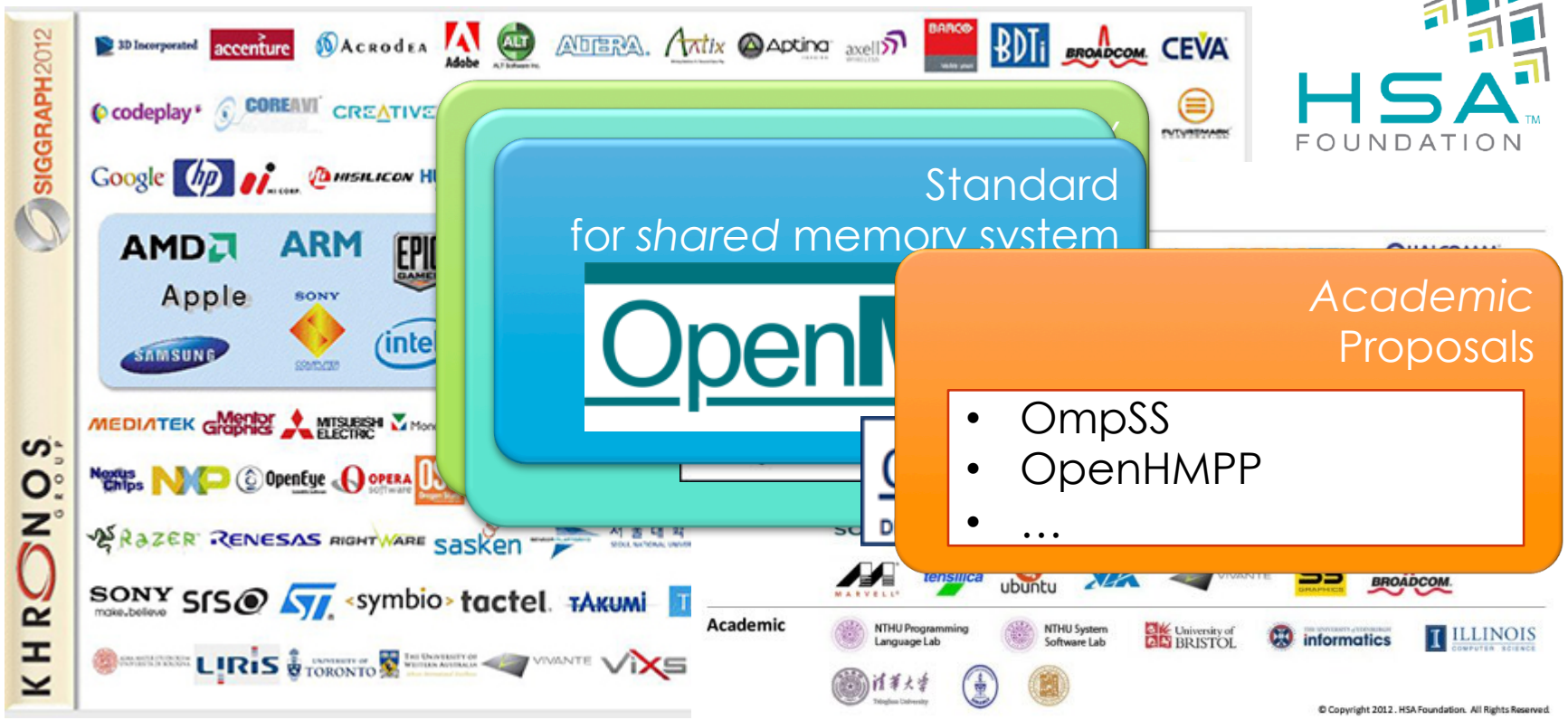
NUMA COMMUNICATION

PE0 on CLUSTER 1 writes on data
Local communication
1 cycle latency

PE0 on CLUSTER 0 writes on data2
Remote communication
Increasing latency with distance



CROWDED CONSTELLATION OF PROGRAMMING MODELS



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

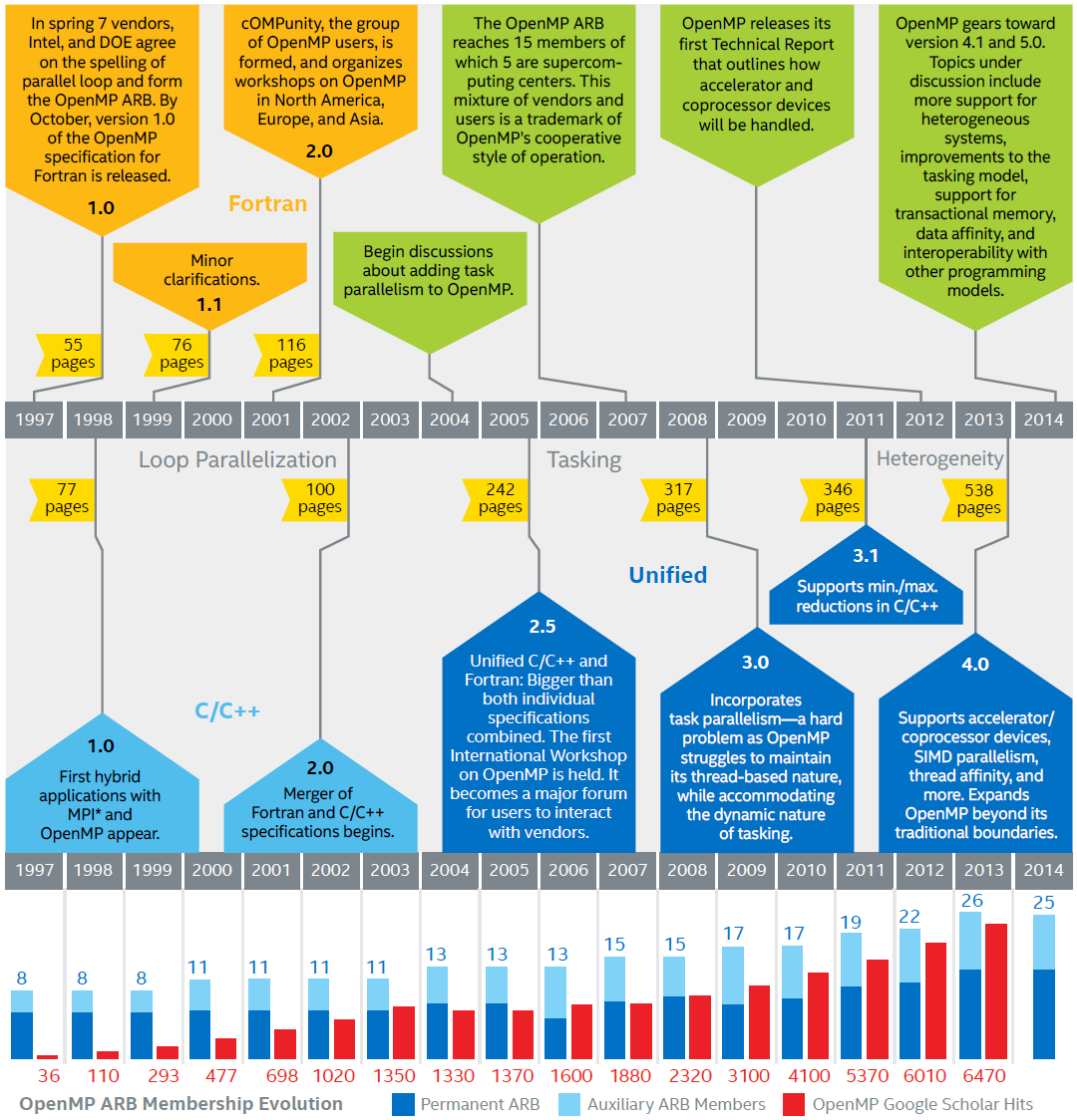
Table 1: Time/effort (T/E) and number of lines of code for given bench- marks for the three frameworks. MM(Matrix Multiplication), LU(LU Factorization), CONV(Image convolution). [+++ : Least time/effort - - - : Most time/effort]

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

An OpenMP* Timeline

By Jim Cownie, *OpenMP Architect,* **Alejandro Duran,** *Application Engineer,* **Michael Klemm,** *Senior Application Engineer,* **and Luke Lin,** *OpenMP Software Product Manager*

1996 Vendors provide similar but different solutions for loop parallelism, causing portability and maintenance problems.
Kuck and Associates, Inc. (KAI) | SGI | Cray | IBM | High Performance Fortran (HPF) | Parallel Computing Forum (PCF)



ENMP

▲ De-f g

▲ Supp *for clusters*

▲ Ann the
com

▲ Base (shared
mer

▼ Desi y

▼ Lack

1.

2.

But…

?

Intel's Parallel Universe magazine – May 2014

FORK-JOIN PARALLELISM

- OpenMP leverages a ***fork-join*** execution model
- The program starts on a single thread (the *master*), then new threads are recruited upon entering a **parallel region**
- Parallelism in application is always found at multiple levels, in a hierarchical manner
  - *How to capture this in the program?*
  - *How to map the wokload in a NUMA-aware manner?*

NESTED PARALLELISM

```
while(1)
{
    #pragma omp parallel num_threads(4)
    {
        #pragma sections
        {
            #pragma section
            {
                #pragma omp parallel num_threads(16)
                ColorScaleConv();
            }
            #pragma section
            {
                #pragma omp parallel num_threads(16)
                cvMoments();
            }
            #pragma section
            {
                #pra
                cvA
            }
        }
    }
}
```

A.

B

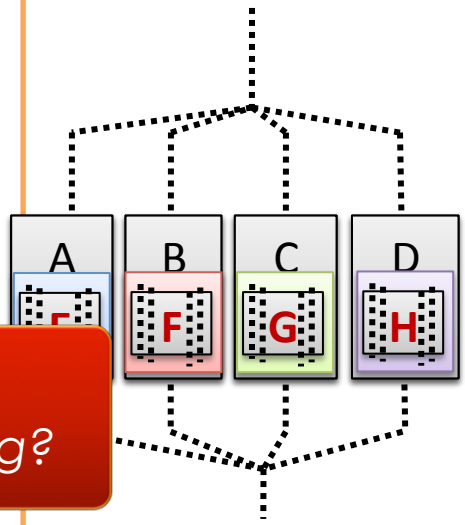
C.

D.

A powerful abstraction for specifying structured parallelism

```
void ColorScaleConv()
{
    #pragma omp for
    for(i = 0; i < FRAME_SIZE; i++)
    {
        [ALGORITH]
    }
}
```

And very suitable for NUMA (cluster-based) systems

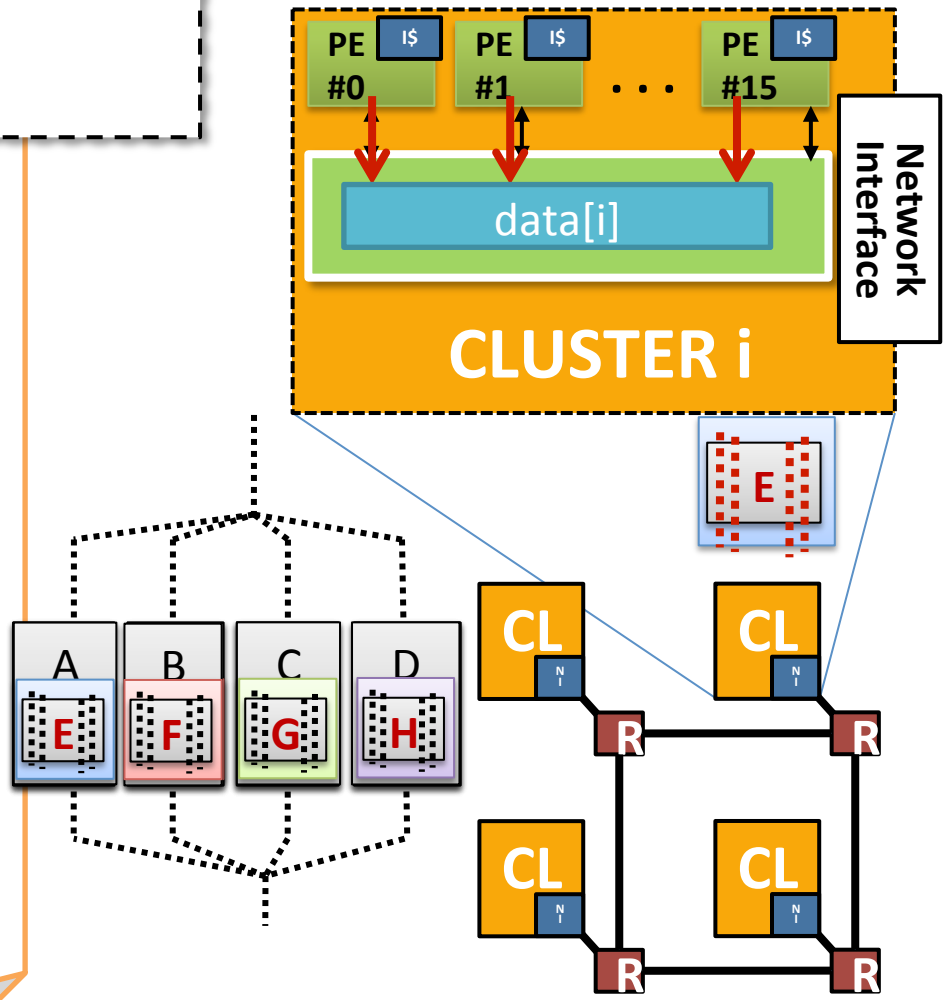


But what about PHYSICAL thread mapping?

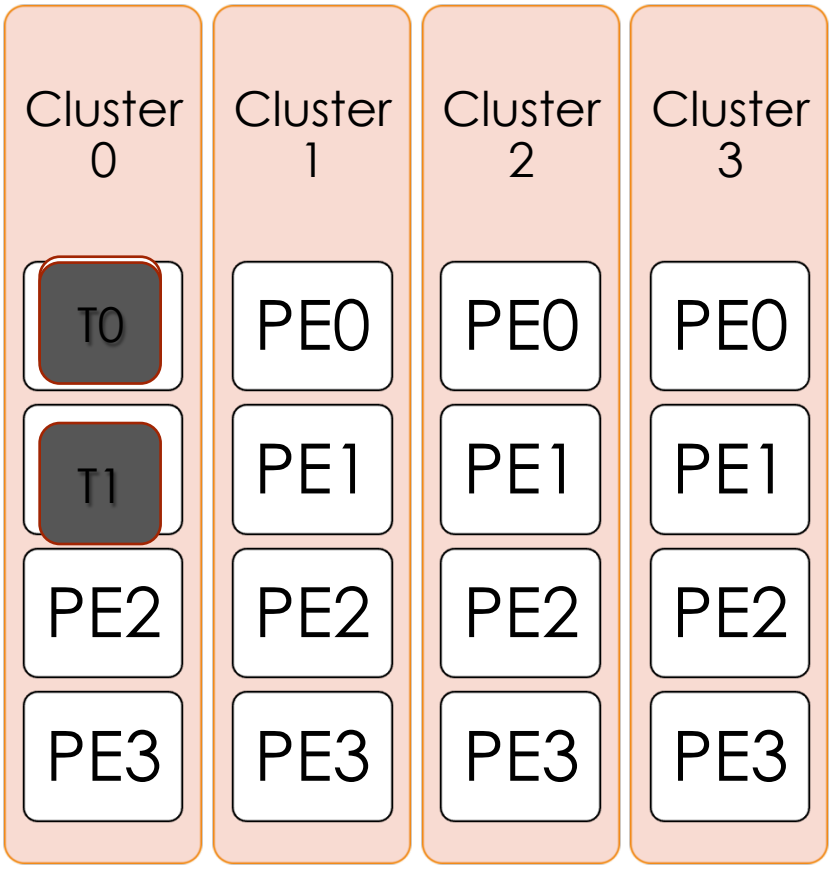
NESTED PARALLELISM CLUSTER-AWARE

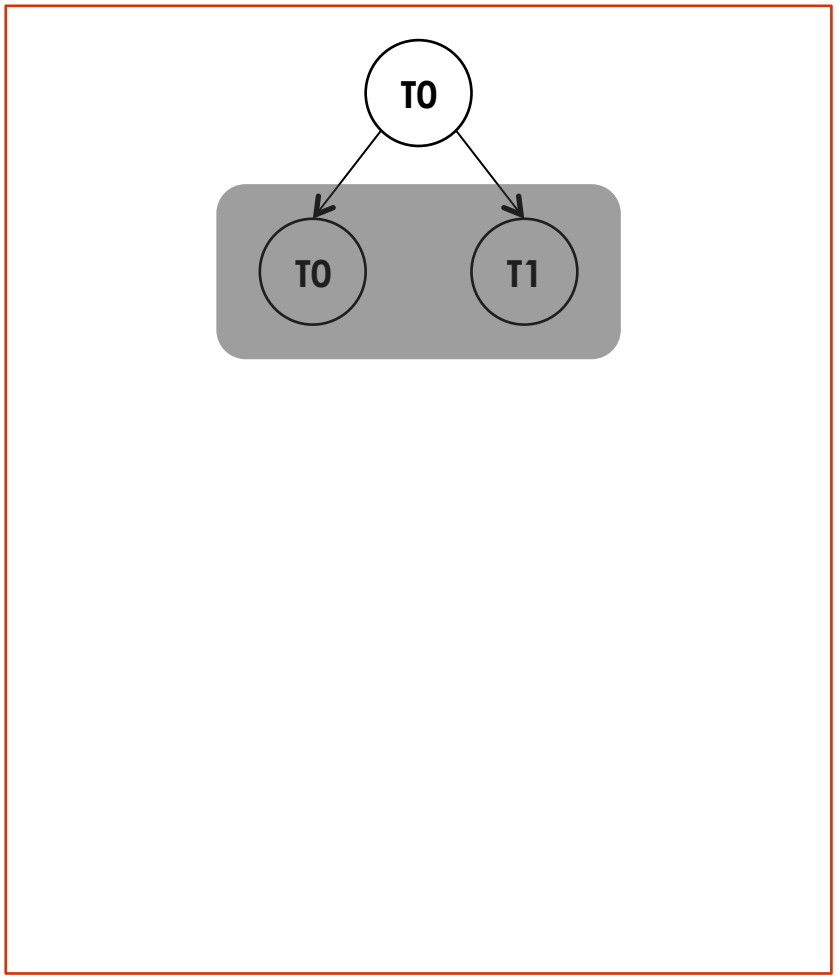
```
while(1)
{
    #pragma omp parallel
    {
        #pragma sections
        {
            #pragma section
            {
                #pragma omp parallel
A.              ColorScaleConv();
            }
            #pragma section
            {
                #pragma omp parallel
B.              cvThreshold();
            }
            #pragma section
            {
                #pragma omp parallel
C.              cvMoments();
            }
            #pragma section
            {

                #pragma omp parallel)
D.              cvAdd();
            }
        }
    }
}
```

```
void ColorScaleConv()
{
    #pragma omp for
    for(i = 0; i < FRAME_SIZE; i++)
    {
        [ALGORITH]
    }
}
```

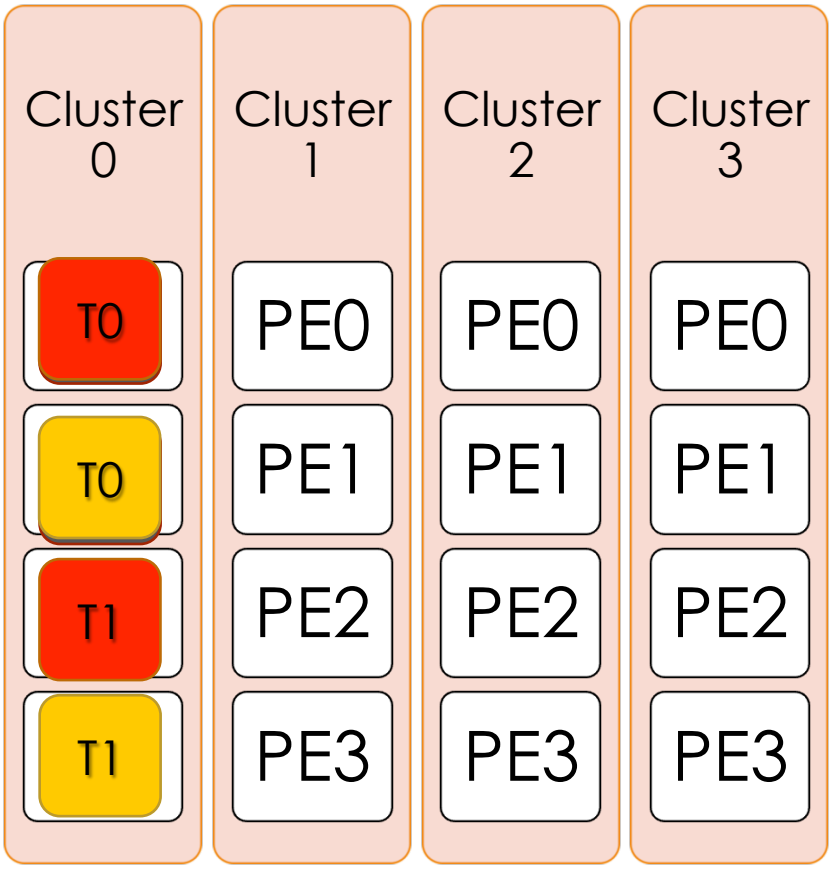


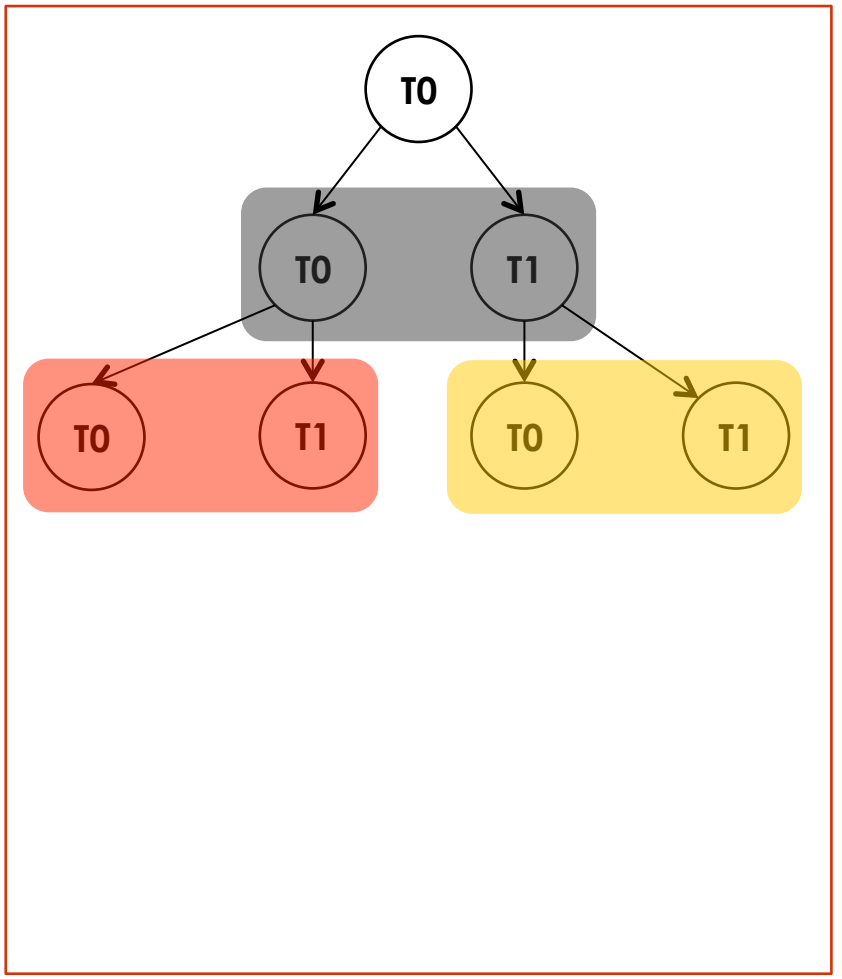
NUMA-UNAWARE THREAD MAPPING



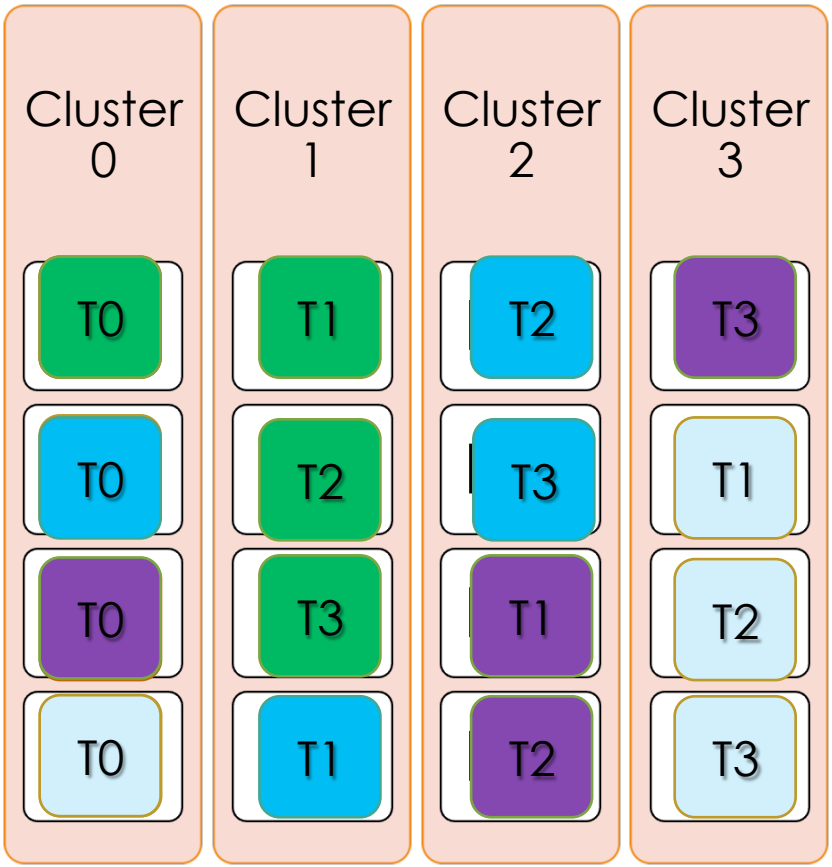


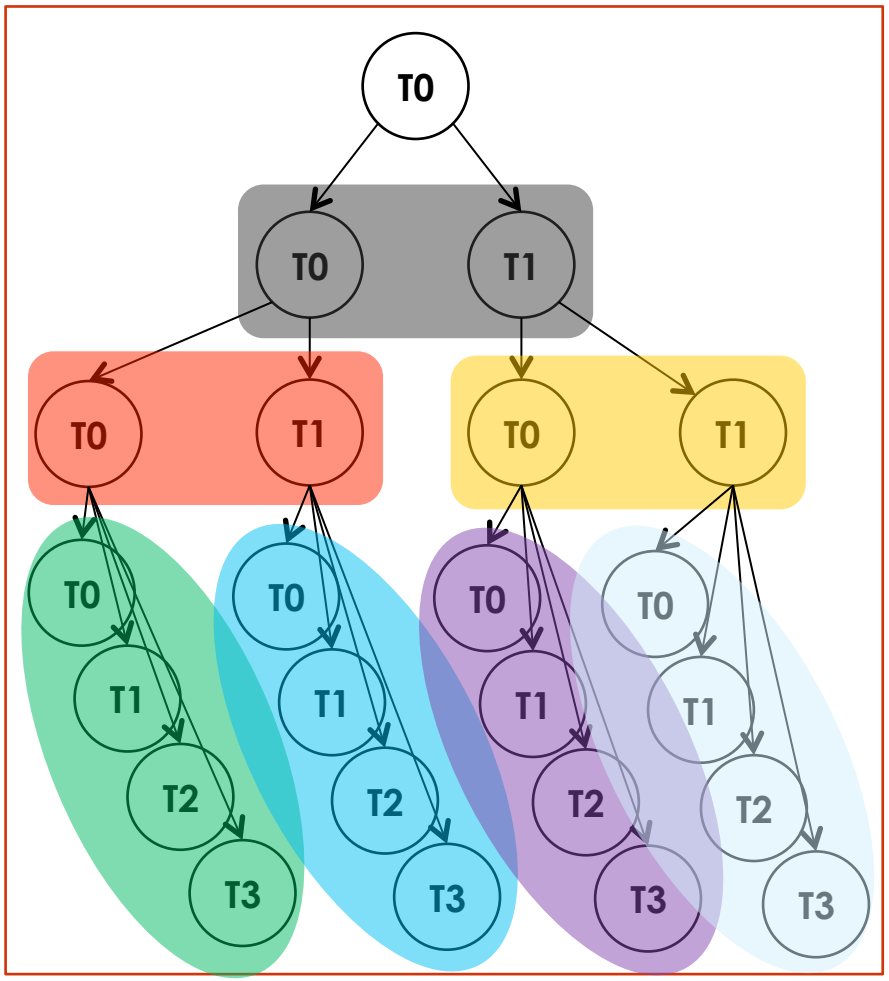
NUMA-UNAWARE THREAD MAPPING





NUMA-UNAWARE THREAD MAPPING





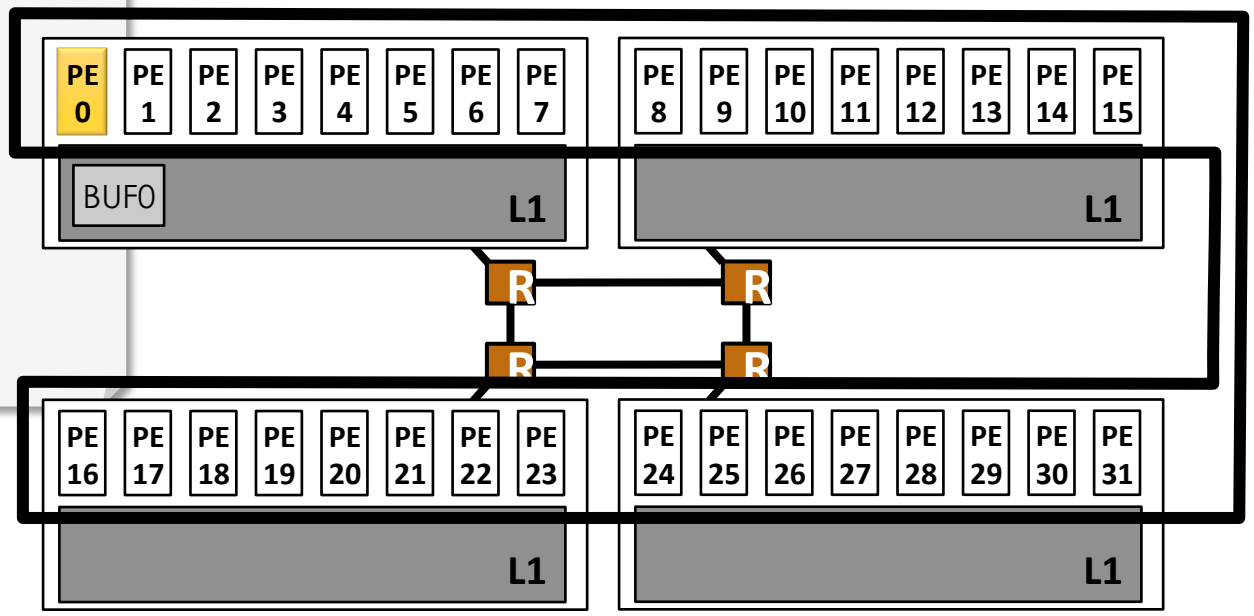
LOCALITY-AWARE WORKLOAD DISTRIBUTION

- How to best distribute parallel workload within a cluster-based manycore accelerator?
  - *Flat* parallelism?
  - Nested **`parallel for`**?
  - The **`distribute`** directive: Teams and leagues
  - And how about non-loop centric parallelism?

```
#pragma omp target map(in, out)
{
 for(stripe = 0;
      stripe < N_STRIPES;
      ++stripe)
  {
     dma_in(in[stripe]);

     KER (in[stripe], out[stripe]);

     dma_out(out[stripe]);
  }
}
```

```
void KER(in1, in2, out) {
  #pragma omp parallel for
    for(i = 0; i < … ; i++){
        [ALGORITHM]
    }
}
```

flat parallel thread team



master thread slave thread

outer team inner team

The role of DMA in explicitly managed memory hierarchies!

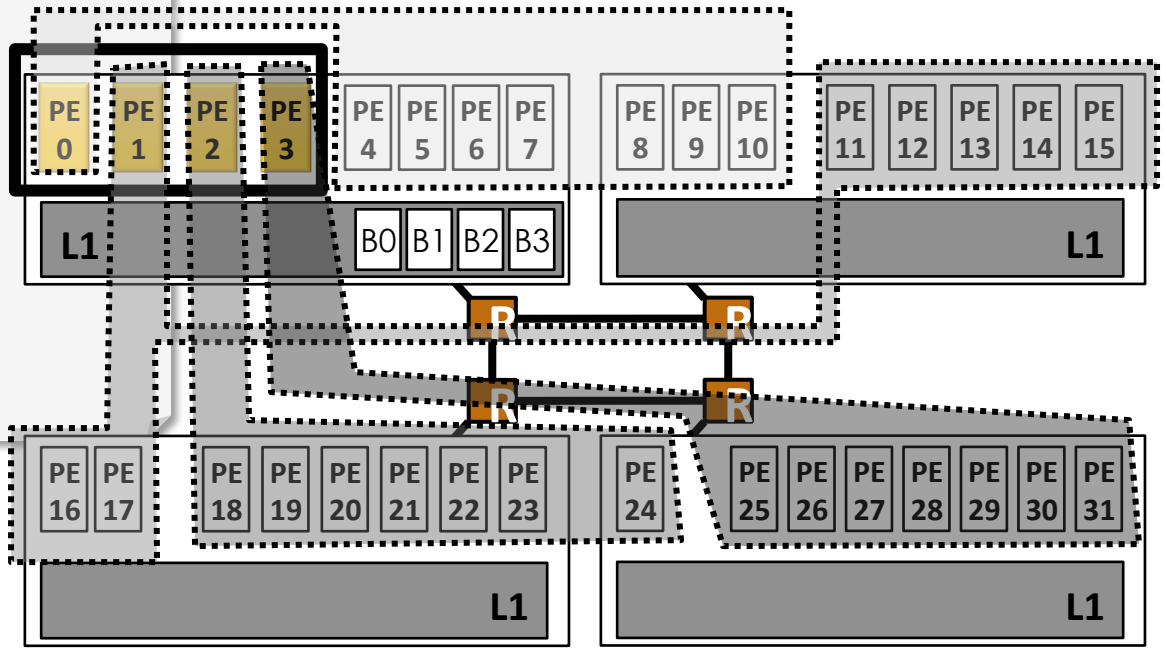
LOCALITY-AWARE WORKLOAD DISTRIBUTION

- How to best distribute parallel workload within a cluster-based manycore accelerator?
  - *Flat* parallelism?
  - Nested **`parallel for`**?
  - The **`distribute`** directive: Teams and leagues
  - And how about non-loop centric parallelism?

```
#pragma omp target map (in, out)
#pragma omp parallel num_threads(4)
for(stripe = 0;
    stripe < N_STRIPES;
    ++stripe)
  {
    dma_in(in[stripe]);

    KER (in[stripe], out[stripe]);

    dma_out(out[stripe]);
  }
}
```

```
void KER(in1, out) {
#pragma omp parallel for  \
        num_threads(16)
    for(i = 0; i < … ; i++)
        [ALGORITHM]
}
```

nested team



master thread slave thread

outer team inner team

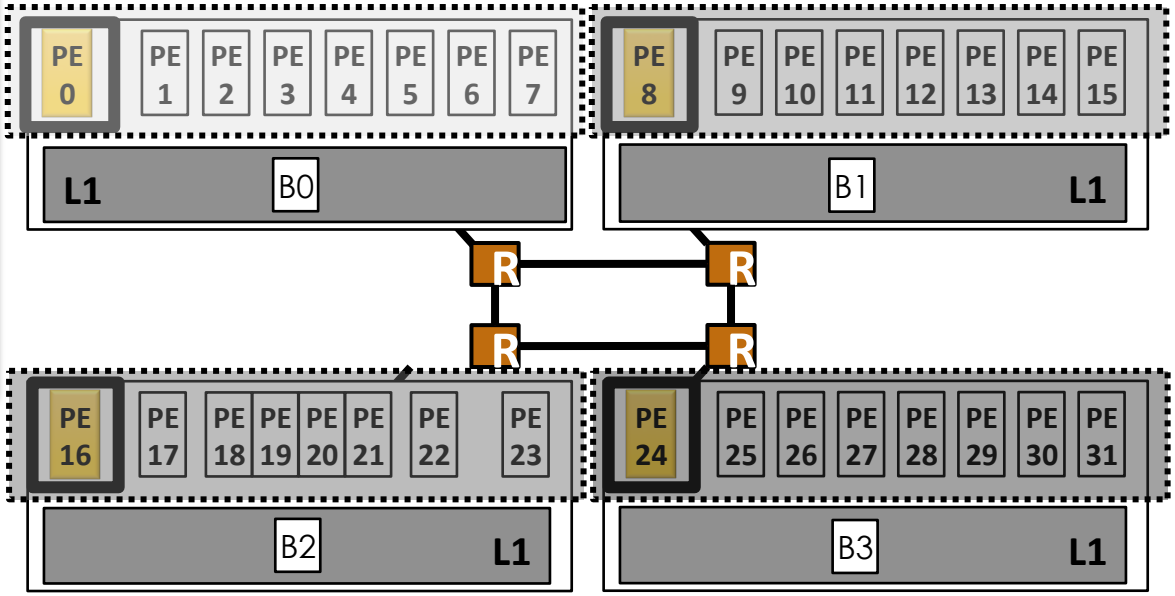
LOCALITY-AWARE WORKLOAD DISTRIBUTION

- How to best distribute parallel workload within a cluster-based manycore accelerator?
  - *Flat* parallelism?
  - Nested **`parallel for`**?
  - The **`distribute`** directive: Teams and leagues
  - And how about non-loop centric parallelism?

```
#pragma omp target teams
num_teams(4) map (in, out)
#pragma omp distribute
for(stripe = 0;
    stripe < N_STRIPES;
    ++stripe)
  {
     dma_in(in[stripe]);

     cvADD (in[stripe], out[stripe]);

     dma_out(out[stripe]);
  }
}
```

```
void KER(in1, out) {
   #pragma omp parallel for
     for(i = 0; i < … ; i++){
          [ALGORITHM]
     }
}
```

distributed nested team



master thread slave thread

outer team inner team

LOCALITY-AWARE WORKLOAD DISTRIBUTION

- How to best distribute parallel workload within a cluster-based manycore accelerator?
  - *Flat* parallelism?
  - Nested **`parallel for`**?
  - The **`distribute`** directive: Teams and leagues
  - **And how about non-loop centric parallelism?**

NON-LOOP CONSTRUCTS?

For NUMA

```
proc_bind ( master | close | spread )
```

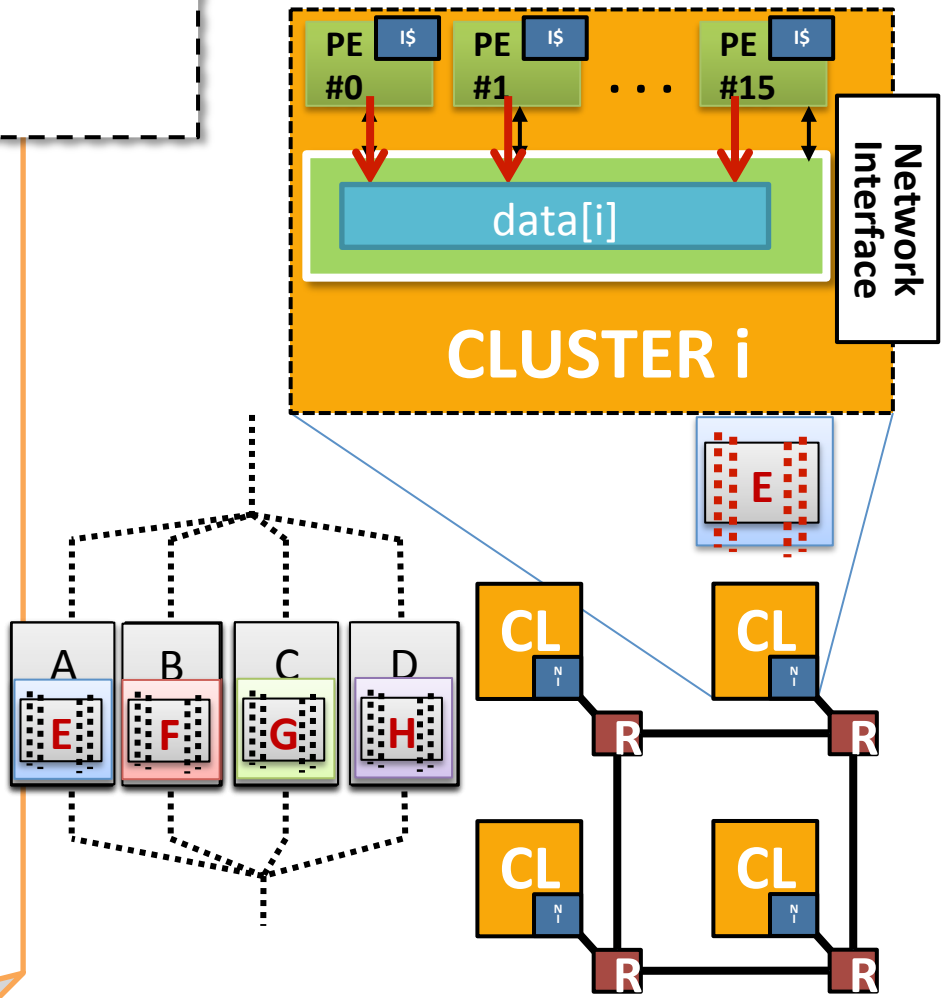
NEW! Clause on specification v4.0 coupled to the **parallel** directive to specify thread mapping:

- **master**: assigns every thread to the same place as the master thread.
- **close**: assigns threads to places close to the place of the parent's Thread.
- **spread**: creates a sparse distribution for a team of T threads among the P places of the parent's place partition.

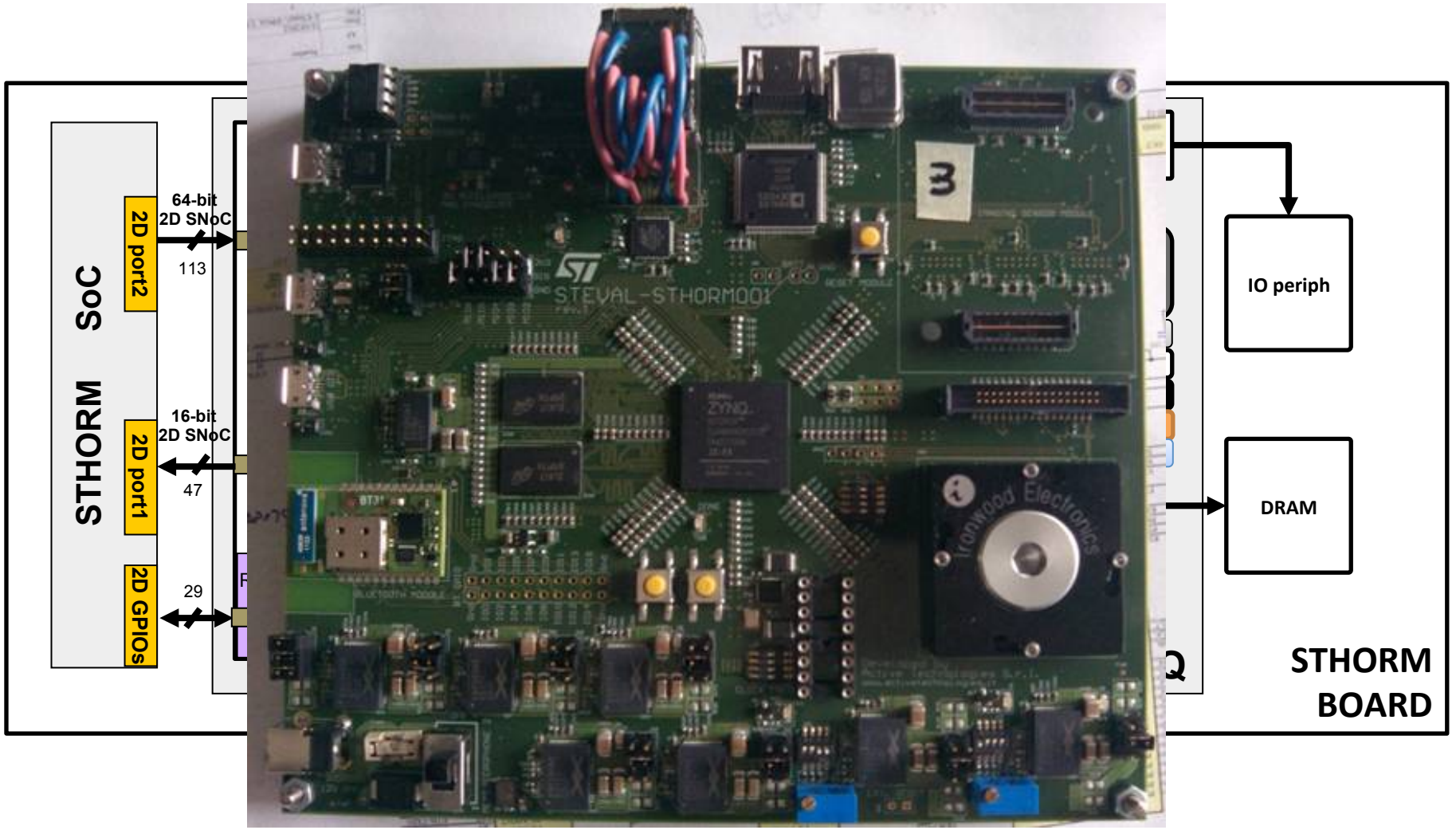
NESTED PARALLELISM CLUSTER-AWARE

```
while(1)
{
   #pragma omp paralle  proc_bind (spread)
   {
      #pragma sections
      {
         #pragma section
         {
            #pragma omp parallel   proc_bind (close)
A.          ColorScaleConv();
         }
         #pragma section
         {
            #pragma omp parallel   proc_bind (close)
B.          cvThreshold();
         }
         #pragma section
         {
            #pragma omp parallel   proc_bind (close)
C.          cvMoments();
         }
         #pragma section
         {

            #pragma omp parallel)  proc_bind (close)
D.          cvAdd();
         }
      }
   }
}
```

```
void ColorScaleConv()
{
   #pragma omp for
   for(i = 0; i < FRAME_SIZE; i++)
   {
      [ALGORITH]
   }
}
```



EXPERIMENTAL SETUP



EXPERIMENTAL SETUP

- We consider 6 benchmarks from the image processing/ computer vision domain

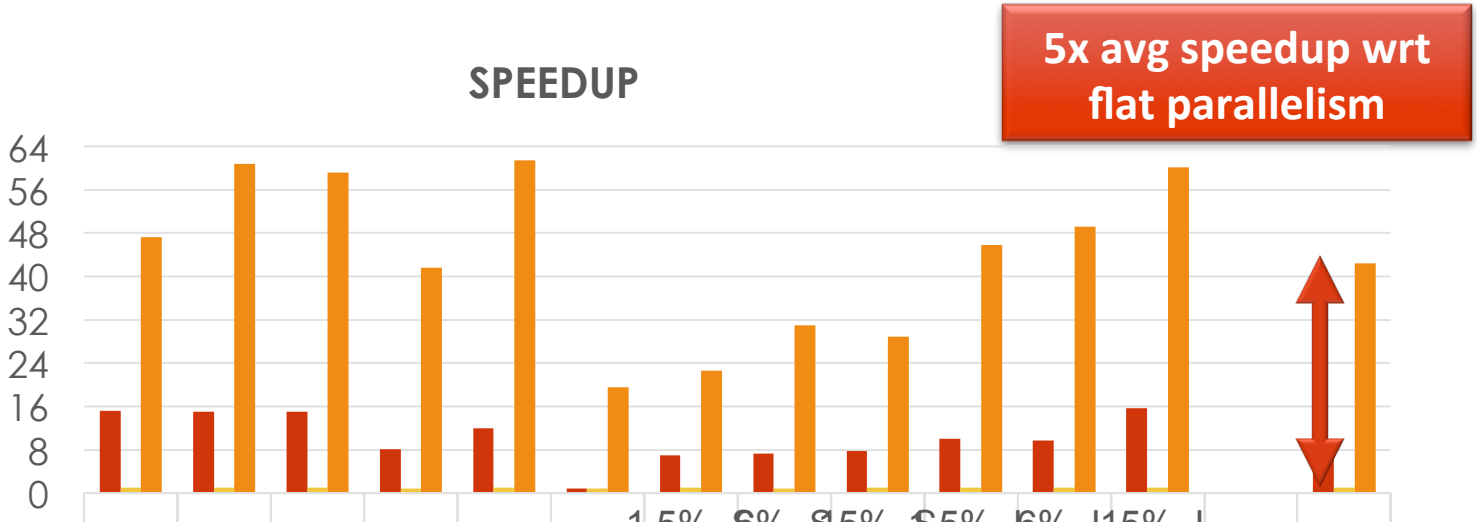
Acronym	Brief explanation
STRAS	Matrix multiplication using Strassen decomposition
GSID	Generalized squared interpoint distance
LRFR	Local reference frame radius (surface matching)
HIST	Histogram interpolation
NCC	Normalized cross-correlation algorithm
CT	Object tracking based on a specific color
FAST	Corner detector

EXPERIMENTAL SETUP (2)

- **FAST** is highly sensitive to input data
  1. Image size
  2. Image composition (number of corners)

Acronym	Brief explanation
1.5%_S	1.5% corner density in a small image (QVGA)
6%_S	6% corner density in a small image
15%_S	15% corner density in a small image
1.5%_L	1.5% corner density in a large image (VGA)
6%_L	6% corner density in a large image
15%_L	15% corner density in a large image

NUMA-AWARE NESTED PARALLELISM



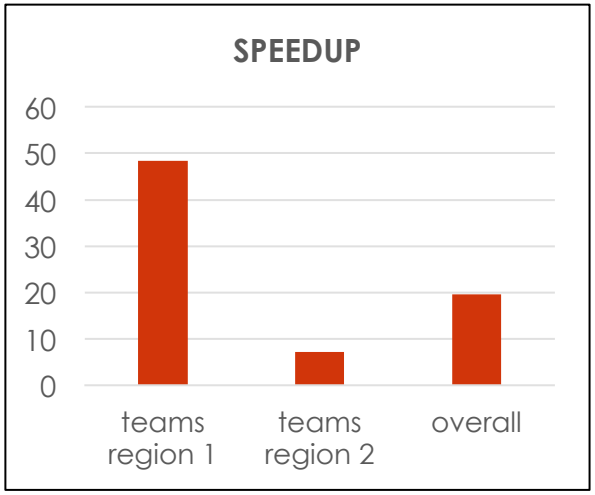
	STRAS	SSID	LR	FR	HIST	NCC	CT	FAST						AVG
								1.5%_S	6%_S	15%_S	5%_L	6%_L	15%_L	
parallel for	15.20	14.96	15.03	8.02	12.02	0.77	6.89	7.30	7.83	10.10	9.67	15.63		9.80
nested parallel for	1.02	1.01	1.02	0.89	0.96	0.83	0.92	0.88	0.99	1.01	0.98	1.04		0.96
distribute	47.26	60.83	59.27	41.61	61.48	19.56	22.67	30.99	28.92	45.76	49.26	60.20		42.51

CT

```
#pragma omp target teams num_teams(4)
#pragma omp distribute
  for(stripe = 0;
    stripe < N_STRIPES;
     ++stripe)
   {
      dma_in(in[stripe]);

      CSC (in[stripe], tmp1[stripe]);
      cvTHR (tmp1[stripe], tmp2[stripe]);
      cvMOM (tmp2[stripe], xy[stripe]);
    }

#pragma omp barrier

#pragma omp distribute
for(stripe = 0;
    stripe < N_STRIPES;
    ++stripe)
  {
     dma_in(in[stripe]);
     dma_in(track[stripe]);

     cvADD (in[stripe], track[stripe], out[stripe]);

     dma_out(out[stripe]);
  }
}
```

```
void CSC(in, tmp1) {
  void cvTHR(tmp1, tmp2) {
    void cvMOM(tmp2, xy) {
      void cvADD(in1, in2, out) {
        #pragma omp parallel for
          for(i = 0; i < … ; i++){
              [ALGORITHM]
          }
      }
    }
  }
}
```

distributed nested team



Work supported by

P-SOCRATES

FP7 – ICT
GRANT N° 611016

ON THE EFFECTIVENESS OF OPENMP TEAMS FOR PROGRAMMING EMBEDDED MANYCORE ACCELERATORS

Alessandro Capotondi
University of Bologna

Andrea Marongiu
Swiss Federal Institute of Technology in Zurich (ETHZ)
a.marongiu@iis.ee.ethz.ch