Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

arm

Hardware Acceleration of Sparse Data Rearrangement Near Memory

Adrian Barredo, **Miquel Moretó** and Jonathan Beard

Arm Research Summit 2019

17/09/2019

Introduction

- Moore's Law
  - Chip density doubling every 2 years
- In the 90's – Early 2000's
  - Increase pipeline depth
  - Increase frequency
- Since mid 2000's: The multicore era
- More recently: heterogenous manycores
- Increased CPU – Memory gap

Memory Wall

Alleviating the Memory Wall

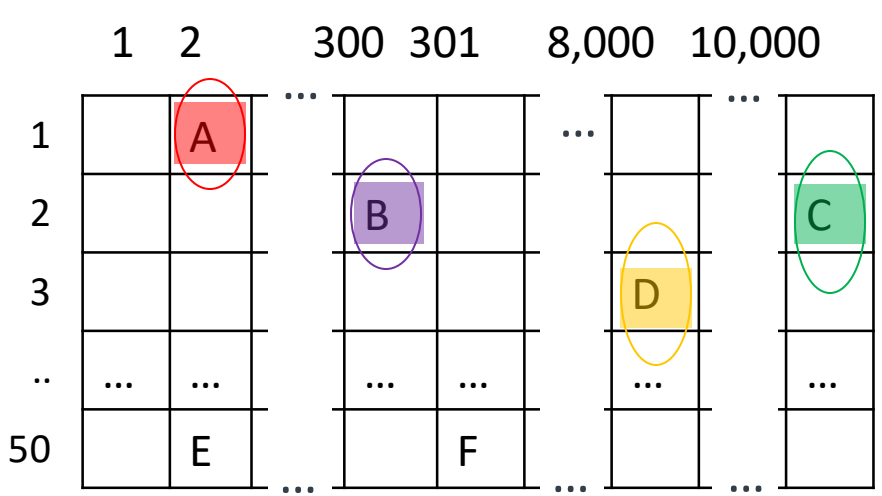
- Deep and large cache hierarchy
- Hardware and software prefetchers
- OS and runtime systems to exploit data locality
- Cache-aware application coding and compiler optimizations

- But…
  - On-chip cache capacity / speed (i.e MB/FLOPS) stagnates or decreases
  - Off-chip bandwidth / speed also decreases

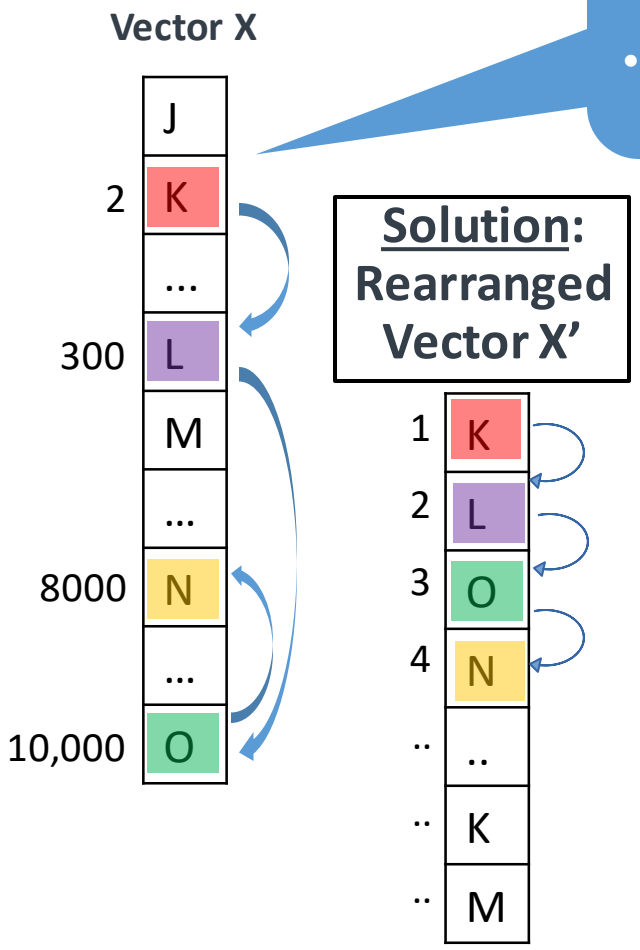
- Specially problematic for applications with **irregular** memory patterns

Sparse Data and the Memory Hierarchy

SpMV – Sparse Matrix-Vector multiply



*



Problem

- Poor cache line utilization (CLU) and temporal locality
- Poor bandwidth utilization by the application

Solution: Rearranged Vector X'

- Using rearranged vector X':
- ✓ Contiguous accesses giving 100% CLU
- ✓ Good prefetching
- ✓ Better utilize bandwidth of memory system

Rearrange function for X -> X' is

X'[i] = X[COLUMN[i]]

Hardware Offload Solutions for Sparse Data

Still an unsolved problem and one that computing community is interested in

Vector gather-scatter

- Better than scalar but still memory bound
- Prefetchers do not work well for irregular pattern

Programmable prefetcher

- Prefetching precise data (not speculative) [Ainsworth18]
- Hide memory latency

Decoupled access-execution

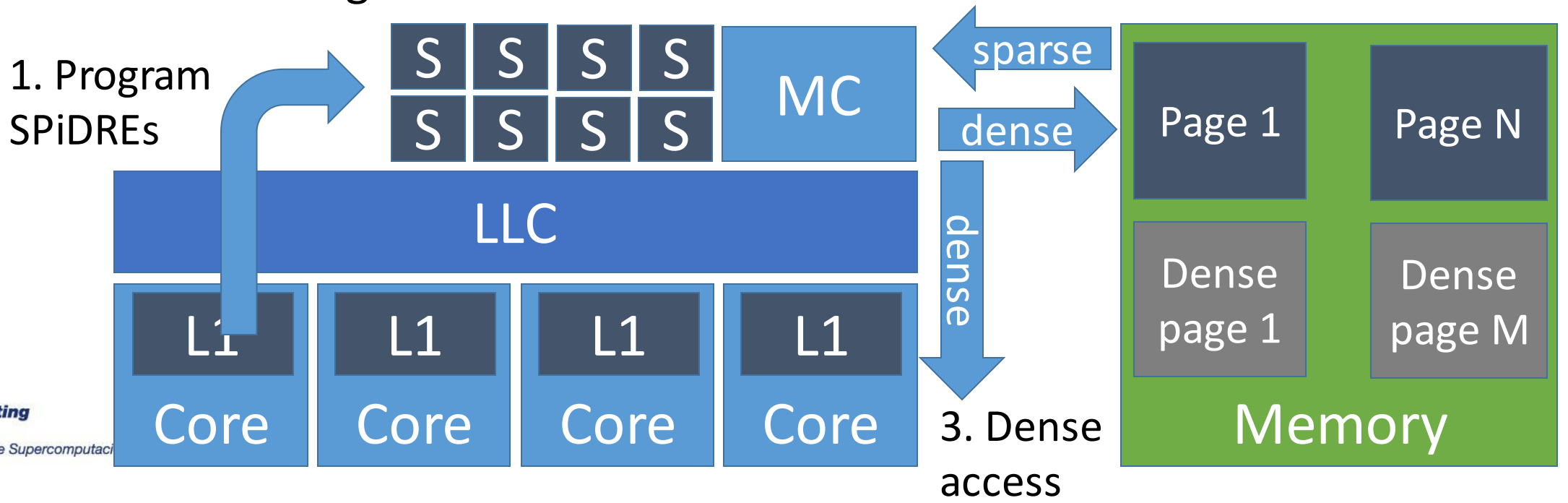
- Let access thread run ahead to hide memory latency

All are viable solutions, still have the problem of poorly utilized caches and memory bandwidth

[Ainsworth18] Sam Ainsworth, Timothy M. Jones: "An Event-Triggered Programmable Prefetcher for Irregular Workloads". ASPLOS 2018.

SPiDRE: Data-Rearrangement Near Memory

- **SP**arse **D**ata **RE**arrangement Engine (SPiDRE)
  - Pool of accelerators near-memory
  - User-directed data rearrangement functions
  - API to enable data rearrangement
  - Rearranged data stored in memory
  - Cores access rearranged dense data



Advantages of using SPiDRE

- Programmer driven rearrangement which avoids the need of specialized data structures
- No limitations in the rearrangement functions
- Data reorganization can be done by several devices, exploiting the memory bandwidth
- Data rearrangement can be overlapped by *host* core computation
- Simple and efficient prefetching
- Efficient vectorization
- Less data movement and higher cache line utilization (CLU)

- But…

Code modifications | **SPiDRE HW cost** | **Performance tradeoffs**

SPiDRE Phases

1. Flushing of sparse data + invalidation of dense data
2. Allocation of SPiDRE accelerators
3. Offloading of rearrangement function to SPiDRE
4. Rearrangement trigger
5. Execution of rearrangement function
6. Synchronization between SPiDRE and host cores
7. Release of SPiDRE accelerators

STRIDE

Original version

```
void stride_kernel(double *x, int *idx, ...){
   ....
   for (len = 0; i  < len; len++) {
      v1s1m3( ... );
      v1s2m3( ... );
      v1s3m3( ... );
      v2s2m3( ... );
      v2s2m4( ... );
      v1s1i3( x, idx, ... );
   }
}

void v1s1i3(double *x, int *idx, ... ){
   ....
   for( j = 0; j < irep; j++ ) {
      t1 = 1.0/(double)(j+1);
      for( i = 0; i < n; i++ )
         y[...] += t1*x[idx[i]];
   }
}
```

SPiDRE version

```
void stride_kernel(double *x, int *idx, ...){
   ....
   for (len = 0; i  < len; len++) {
      Invalidation phase (x_rearr)
      hw_rearr( x, idx, x_rearr, ... );
      v1s1m3( ... );
      v1s2m3( ... );
      v1s3m3( ... );
      v2s2m3( ... );
      v2s2m4( ... );
      v1s1i3_hw_rearr( x_rearr, ... );
   }
}

void v1s1i3_hw_rearr(double *x_rearr, ... ){
   ....
   for( j = 0; j < irep; j++ ) {
      t1 = 1.0/(double)(j+1);
      for( i = 0; i < n; i++ )
         y[...] += t1*x_rearr[i];
   }
}
```

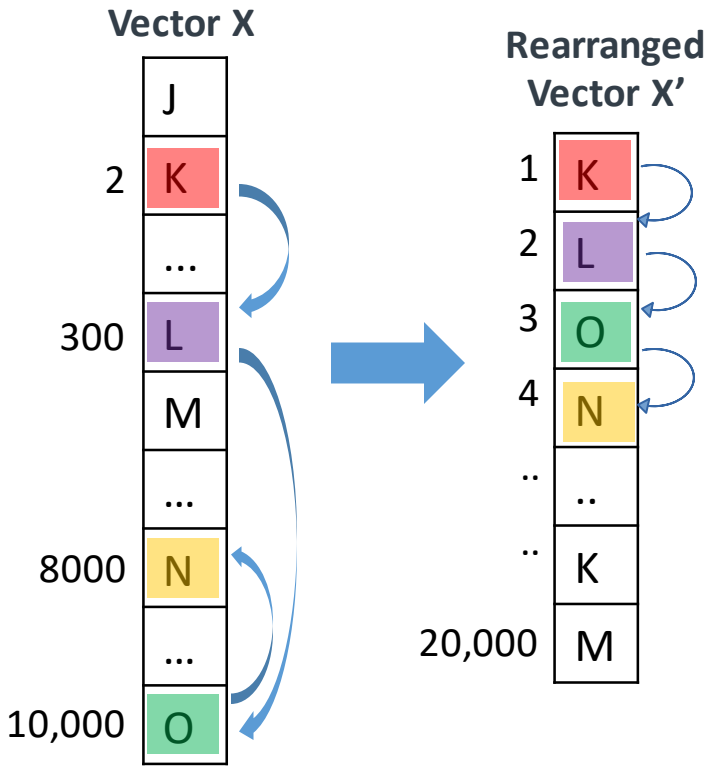
SPiDRE Operational Model

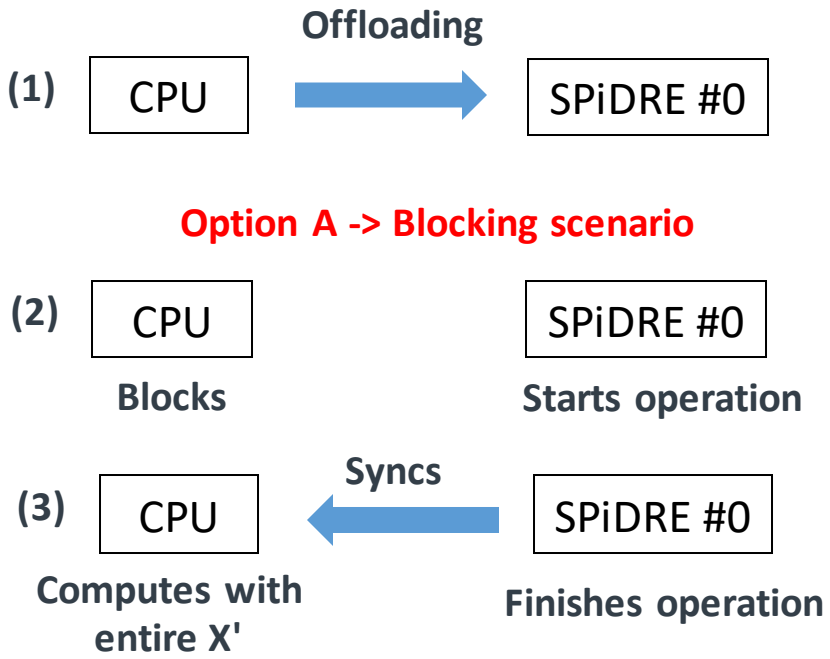
Rearrange function for X -> X' is

X'[i] = X[COLUMN[i]]

Rearrangement Info

X' Pointer, X Pointer, COLUMN Pointer, Index (I)





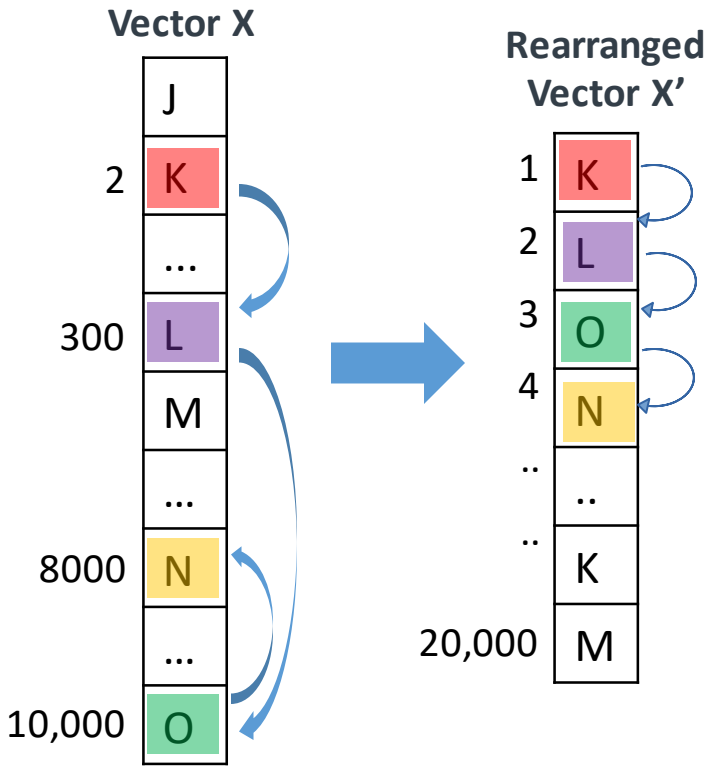
SPiDRE Operational Model

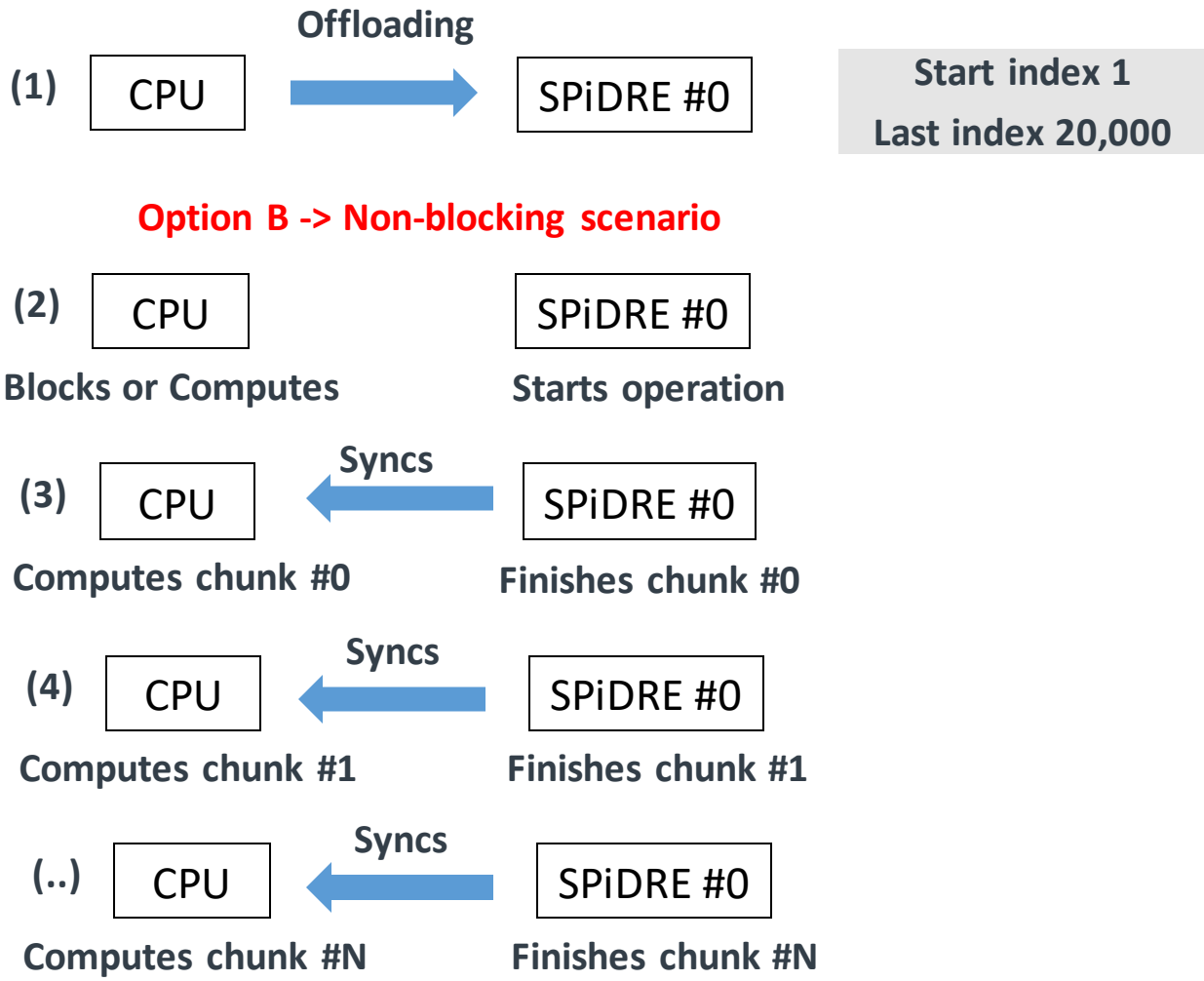
Rearrange function for X -> X' is

X'[i] = X[COLUMN[i]]

Rearrangement Info

X' Pointer, X Pointer, COLUMN Pointer, Index (I)

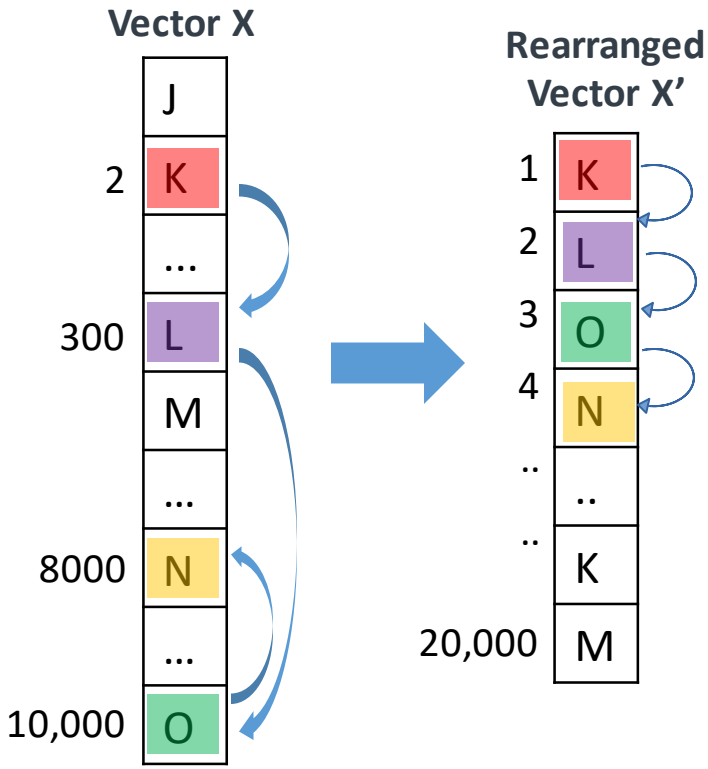




SPiDRE Operational Model

Rearrange function for X -> X' is
X'[i] = X[COLUMN[i]]

Rearrangement Info
X' Pointer, X Pointer, COLUMN Pointer, Index (I)



(1) CPU → Offloading → SPiDRE #0, SPiDRE #1

Start (I) 1, End (I) 10,000

Start (I) 10,001,
End (I) 20,000

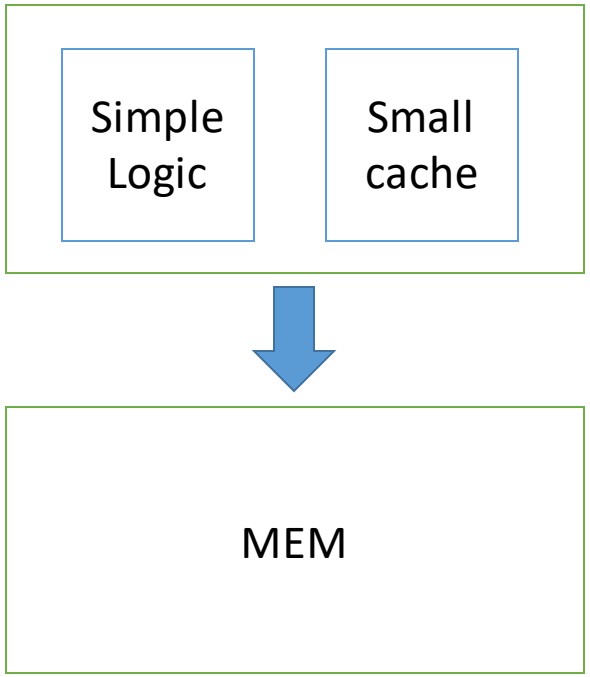
Option B -> Non-blocking scenario

(2)	CPU	SPiDRE #0	SPiDRE #1
	Blocks or Computes	Starts operation	Starts operation
(3)	CPU ← Sync	SPiDRE #0	SPiDRE #1
	Computes chunk #0	Finishes chunk #0	Finishes chunk #N/2 +1
(4)	CPU ← Sync	SPiDRE #0	SPiDRE #1
	Computes chunk #1	Finishes chunk #1	Finishes chunk #N/2+2
(..)	CPU ← Sync	SPiDRE #0	SPiDRE #1
	Computes chunk #N	Finishes chunk #N/2	Finishes chunk #N

X' is populated in half the time

SPiDRE features

- Micro-controller (M0+ class)
  - Has its own stack.
  - Has a special state to know RA info and Master CPU ID.
  - Maintaining memory consistency
    - Requires sparse data to be up-to-date in MEM.
    - Performs flushing to MEM as writes to rearranged data are done.
  - Performs direct virtual-physical memory translation.
    - CPU provides starting physical @
    - Add offset to get physical @



Hardware Modifications to CPU and Cache Side

- It's fundamental not to access X' before is populated
  - A table is required to keep track of the SPDRE syncs
  - Located in CPU
    - In Execute stage. If load access X' and not ready, move it to a FIFO queue
  - Located in prefetchers
    - If generated prefetch address access X' and not ready, discard

X'[120] ready?

Max 140

Yes :)

Rearrangement control table

RA #ID			
SPDRE #ID	Start elem	Last elem	Max elem
0	0	99	32
1	100	199	140
2	200	299	261
3	300	399	310

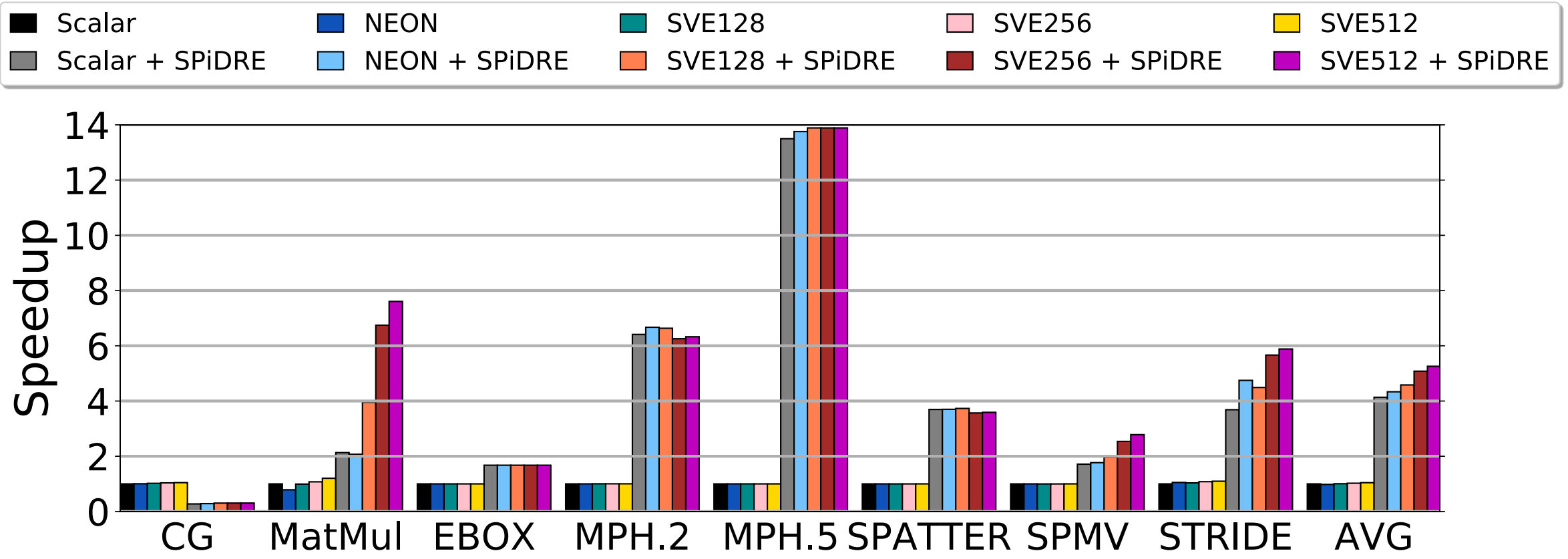
Experimental Setup: Simulation Infrastructure

- Gem5 full-system mode, multi-core with up to 8 out-of-order CPUs
- Detailed cache hierarchy: private L1 cache, shared L2 cache and HW prefetcher

- SPiDRE implemented into fully-system gem5 as an in-order CPU
- SPiDRE programmed through app with new instructions
  - App manually modified to sequentially access data
- SPiDRE timing model
  - Offloading CPU-SPiDRE considered
  - Sync SPiDRE-CPU considered
  - Flushing mechanism implemented (4 req/cycle)
  - Virtual/physical address translation performed within the same cycle
    - Physical memory allocated contiguously

Performance

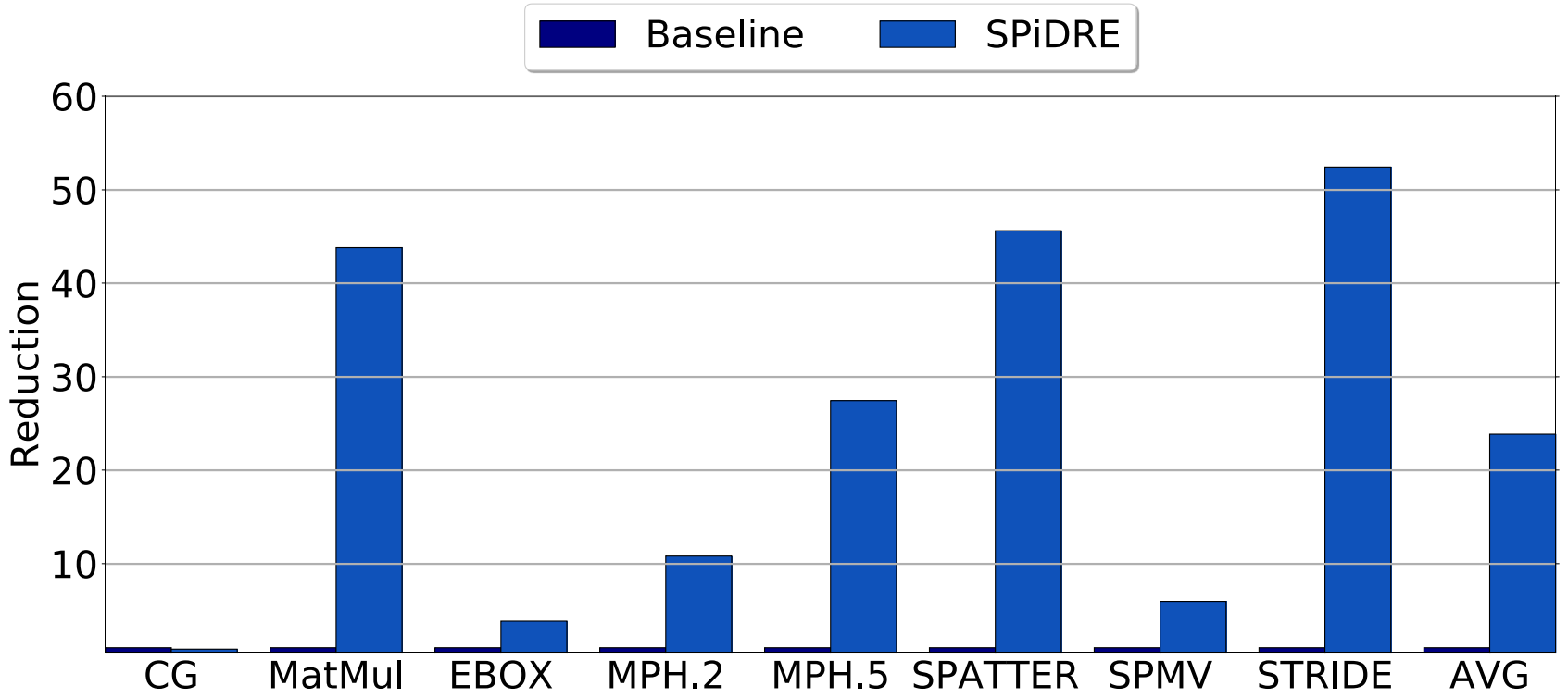
Core vs core + SPiDRE (8 devices)



SPiDRE enables more efficient vectorization!!

Data movement reduction

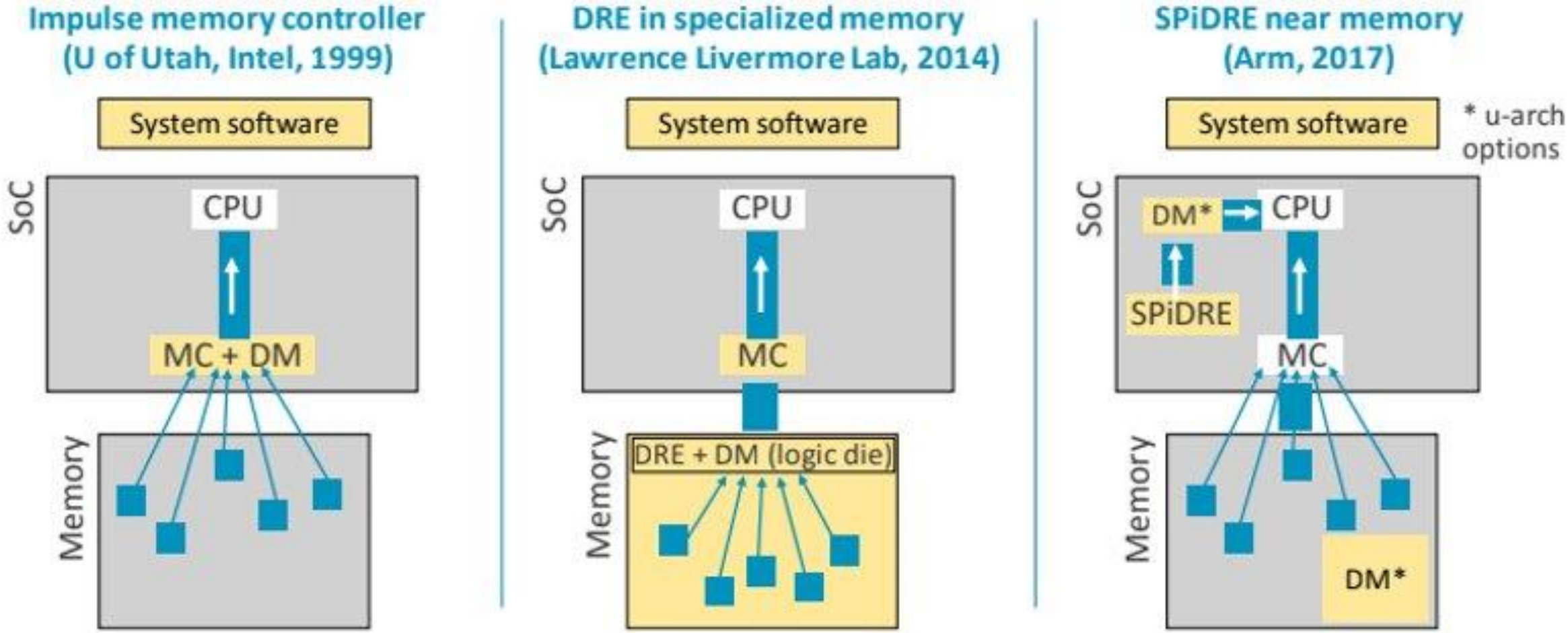
Bytes transferred though the L1D-L2 cache



L1D misses reduced 60%, L2 misses reduced 25%
L1D misses latency reduced 82%
L1D-L2 bus data movement reduced 22x

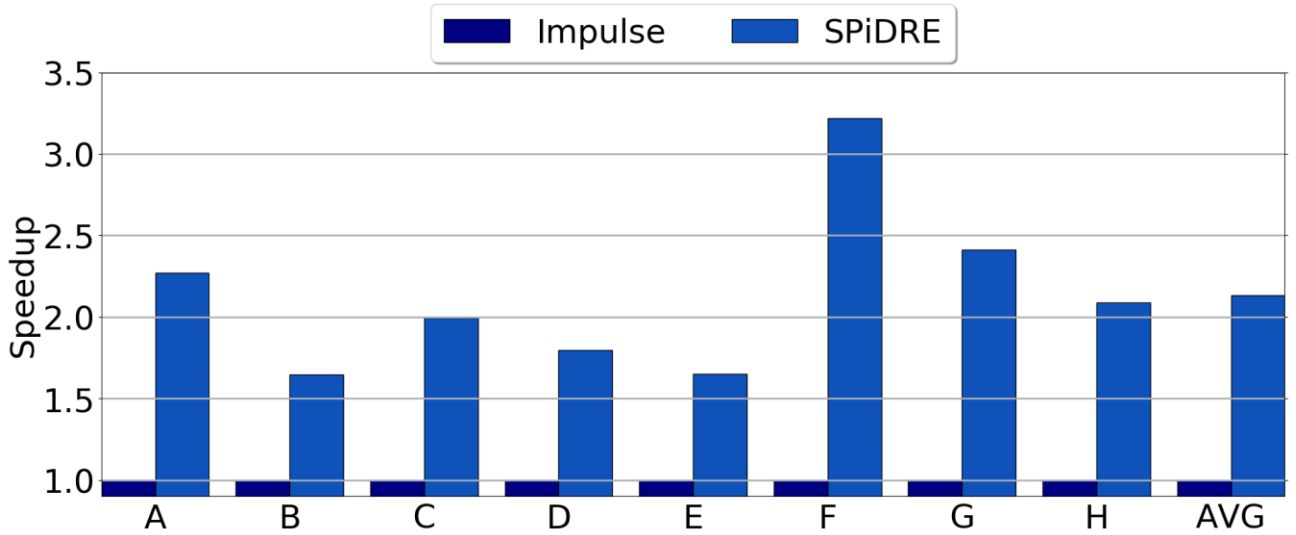
Hardware Rearrangement Approaches

DRE: data rearrangement engine, MC: memory controller, DM: dense data memory



SPiDRE vs Impulse

- Impulse is located in the memory controller.
  - Scalability issues.
- Impulse operates as the core sends a memory request.
  - To choose when and where operate is not possible.
- In Impulse, if data is evicted a scatter phase is triggered.
- In Impulse, if data are evicted and accessed again, a new gather needs to be done.
- In Impulse, there are limitations in the rearrange functions.
- In SPiDRE, there is higher memory-level parallelism (MLP).



SpMV benchmark

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Conclusions

- SPiDRE is a near-memory accelerator
  - User-directed transformation of sparse data into dense data
  - Usage of dense data in subsequent operations
- SPiDRE significantly reduces data movement
  - 22x less bytes are transferred between L1D and L2 caches
- SPiDRE improves the effectiveness of HW prefetchers and enables efficient code vectorization
  - Up to 5.3x average performance speedups

- More info in: [PACT'19] A. Barredo, J. C. Beard and M. Moreto, "POSTER: SPiDRE: Accelerating Sparse Memory Access Patterns", PACT 2019.

STARS

POST-DOCTORAL PROGRAM

- Co-funded by BSC and H2020-MSCA-Cofund programme STARS aims at fostering the training of highly skilled post-doc in **all fields of HPC and related applications**, in a stimulating, international and interdisciplinary environment.
- **Deadline** for application: **30 September 2019**
- Nº of fellows to be recruited in this call : **12**
- Duration of 1 fellowship: **24 months**
- **Eligible Candidates** may not have spent in Spain more than 12 months of the last 36 before the deadline (30.sept 2019), must have either a PhD obtained not more than 6 years ago, or have at least 4 years of postgraduate research activity.
    - https://www.bsc.es/join-us/excellence-career-opportunities/stars

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thank you

miquel.moreto@bsc.es