Distributing Memory Bank Accesses in Many-Core Architectures: Hardware approaches

Arthur Vianes and **Frédéric Rousseau**

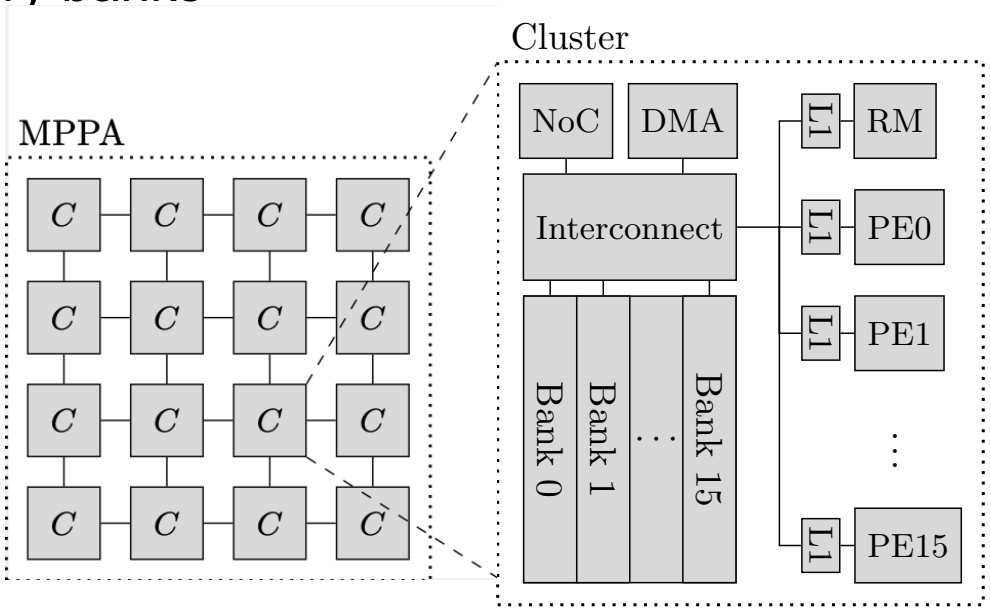
Kalray, France

Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, France

KALRAY
WE ACCELERATE YOUR DATA

Introduction

- Computing power implies a huge pressure on the memory architecture
- A cluster-based many-core architecture: example of the MPPA with local memory split in 16 memory banks
  - Processing Elements
  - Memory Banks
  - . . .

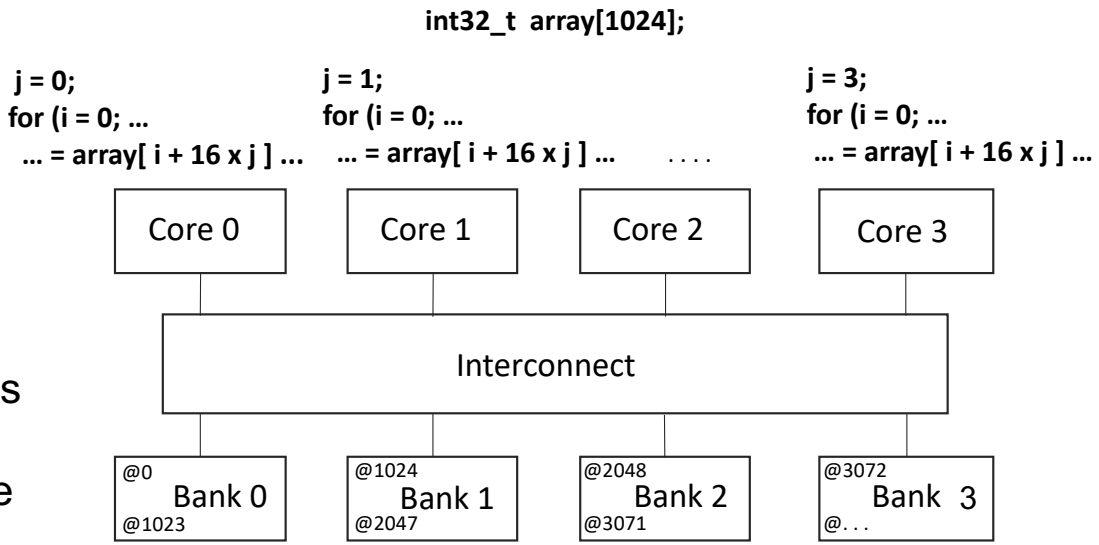


Ideal behavior:

- At each cycle, the memory bank is perfectly distributed and collision-free
- An access to the same memory bank never happens in the same cycle
- In case of collision, only one memory bank access is performed, delaying all the other accesses

Memory bank saturation thru an example

- Distributed computations and memory accesses: focus on **stride access**
  - Typically appears in a majority of high-performance computing scenarios



Use of memory address bits
4 banks, 1kB each
Consecutive address space

Stride size of 16 words (64 bytes)

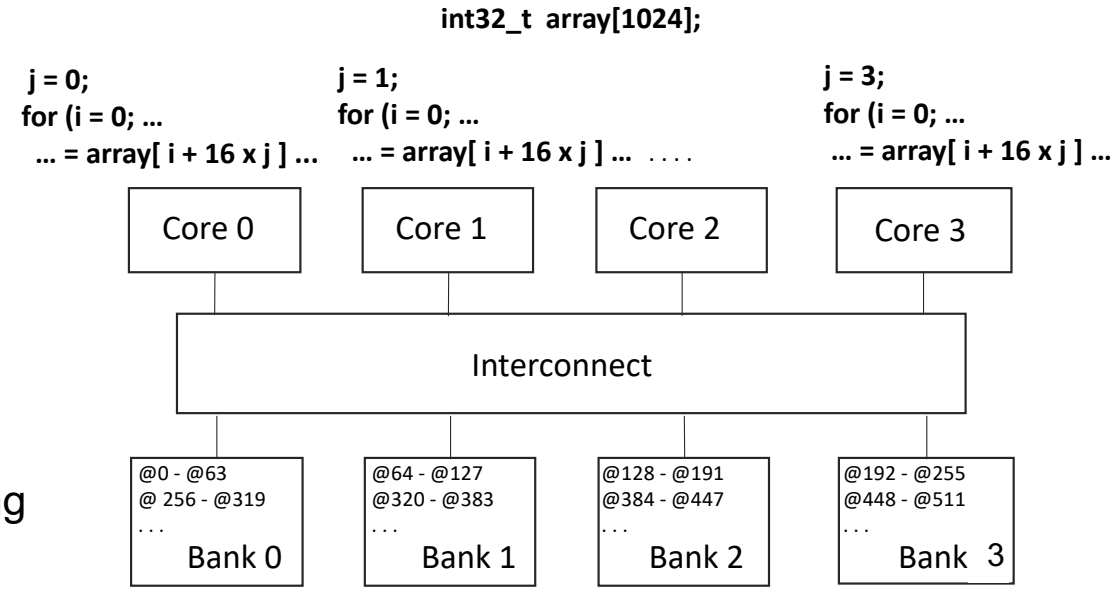
Stride access pattern: addresses spaced by a constant amount

		Core 0	Core 1	Core 2	Core 3
i = 0	Array index	array[0]	array[16]	array[32]	array[48]
	Mem. address	@0	@64	@128	@192
	Mem. bank	0	0	0	0

The 4 cores access the same bank

Is interleaving a solution ?

- Interleaving means slicing up the address range of the memory and then distributing it among the memory banks



Use of memory address bits
4 banks, 1kB each, interleaving

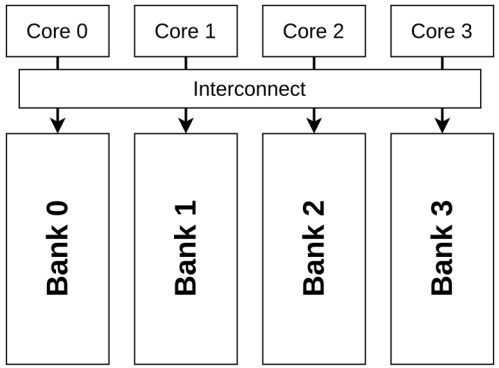
But:
- Better to define a slice size close to what the core is capable to perform at once (same size as the L1 cache line)

		Core 0	Core 1	Core 2	Core 3
i = 0	Array index	array[0]	array[16]	array[32]	array[48]
	Mem. address	@0	@64	@128	@192
	Mem. bank	0	1	2	3

The 4 cores access different banks

How to manage memory bank conflits ?

- Software solutions exist, but these methods require from the programmers to respect strict constraints:
  - A specific array size choice
  - Static memory assignment (ex: SAGE from *C. Chavet and all – 2010*)
  - Strong assumption on the number of memory banks



Example of padding: from a 16 bytes stride to 17 bytes
(adding an unused element – column – in a matrix)

		Bank 0	1	2	3
Address	0	•			
	16	•			
	32	•			
	48	•			
total		4	0	0	0

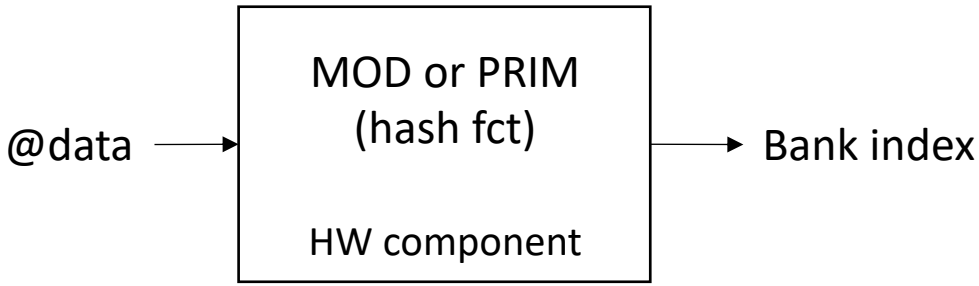
A 4-banks architecture, 1-byte words, with a 16-bytes stride access pattern

		Bank 0	1	2	3
Address	0	•			
	17		•		
	34			•	
	51				•
total		1	1	1	1

A 4 bank architecture, 1-byte words, with a 17-bytes stride

Towards Hardware Solutions

- Hardware solutions may provide solution that release these constraints from the programmer side



- 2 approaches
  - Prime modulus indexing (MOD)
  - Interleaving schemes (PRIM)

Prime Modulus Indexing (MOD)

- The main idea is to avoid common factors between number of banks and the stride size
  - The padding (SW approach) modifies the stride size
  - Prime Modulus Indexing is a HW solution: modification of the number of banks
    - Choising as number of banks a prime number

Number of banks: 5
Stride of 16: no collision (except co-prime of 5)

	Bank				
	0	1	2	3	4
	0	1	2	3	4
	5	6	7	8	9
	10	11	12	13	14
	15	16	17	18	19
	:				

		Bank				
		0	1	2	3	4
Address	0	•				
	16		•			
	32			•		
	48				•	
	total	1	1	1	1	0

Distribution of addresses across 5 banks and distribution of 4 stride accesses of 16 within a 5 bank system

Bank = 16 mod 5 = 1
Index = 16 / 5 = 3

Bank = @data mod N_{bank}
Index = @data / N_{bank}

HW implementation:

- optimized way for some specific numbers as described in (*de Dinechin 1991; Diamond et al. 2014*)
- euclidean division is not needed when the number of banks is prime or simply odd (*Seznec 2015*)

Interleaving Scheme (PRIM)

- PRIM Pseudo-Randomly Interleaved Memory method
  - Proposed by B.R. Rau in 1991
  - Based on polynomial
  - HW implementation based on XOR
  - Perfect distribution for 2^n stride access patterns

	Bank			
	0	1	2	3
	0	1	2	3
	7	6	5	4
	9	8	11	10
	14	15	12	13
	:			

Example: PRIM 7 (111_2) corresponds to $P(x) = x^2 + x + 1$
In which bank is the data at address 13 ?
$13 = 1101_2$ corresponds to $A(x) = x^3 + x^2 + 1$

Polynomial division: $bank = A(x) \div P(x)$

$$R(X) = \begin{array}{r|l} X^3 + X^2 + 1 & X^2 + X + 1 \\ \cline{2-2} X^3 + X^2 + X & X \\ \cline{1-1} X + 1 & \end{array}$$

$R(X) = X + 1$ then the bank is $11_2 = 3$

		Bank			
		0	1	2	3
Address	0	•			
	4				•
	8		•		
	12			•	
	total	1	1	1	1

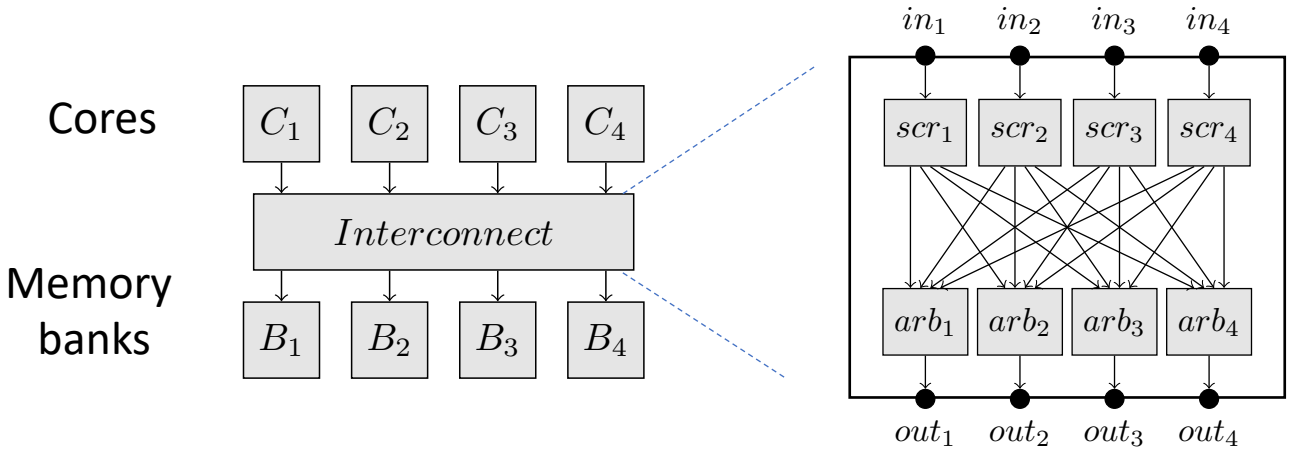
Example of PRIM7 allocation in
a 4 banks architecture,
4 memory accesses with a 4-bytes stride

Implementation on the Kalray MPPA Cluster

KALRAY
WE ACCELERATE YOUR DATA

- Simulator implementation
  - The simulation model is modular and evolutive (exploration of memory architecture, hash functions for memory bank interveaving)
  - A set of simulation components: memory banks, computational cores, arbiters

Details of a 4 cores – 4 memory banks architecture



Scrambler
It scrambles the access to the mem. bank

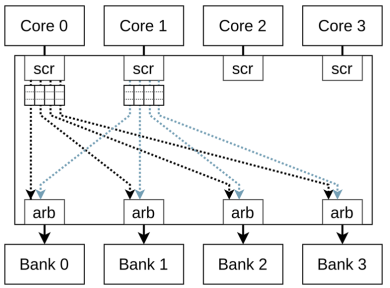
Arbiter

Target architectures: 16 cores – 16 (or 17) memory banks

Interleaving of 32 bytes (cache line size)

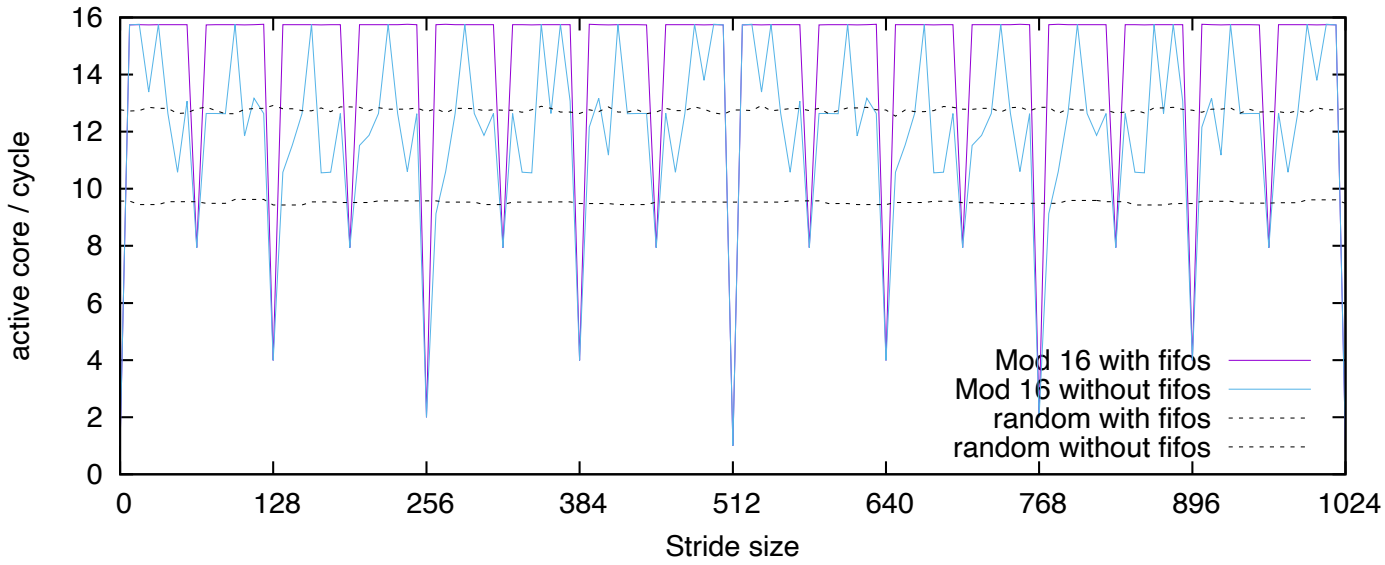
Added FIFOs between cores and memory banks

FIFOs help to smooth out collisions when they are only occasional



MOD16 indexing

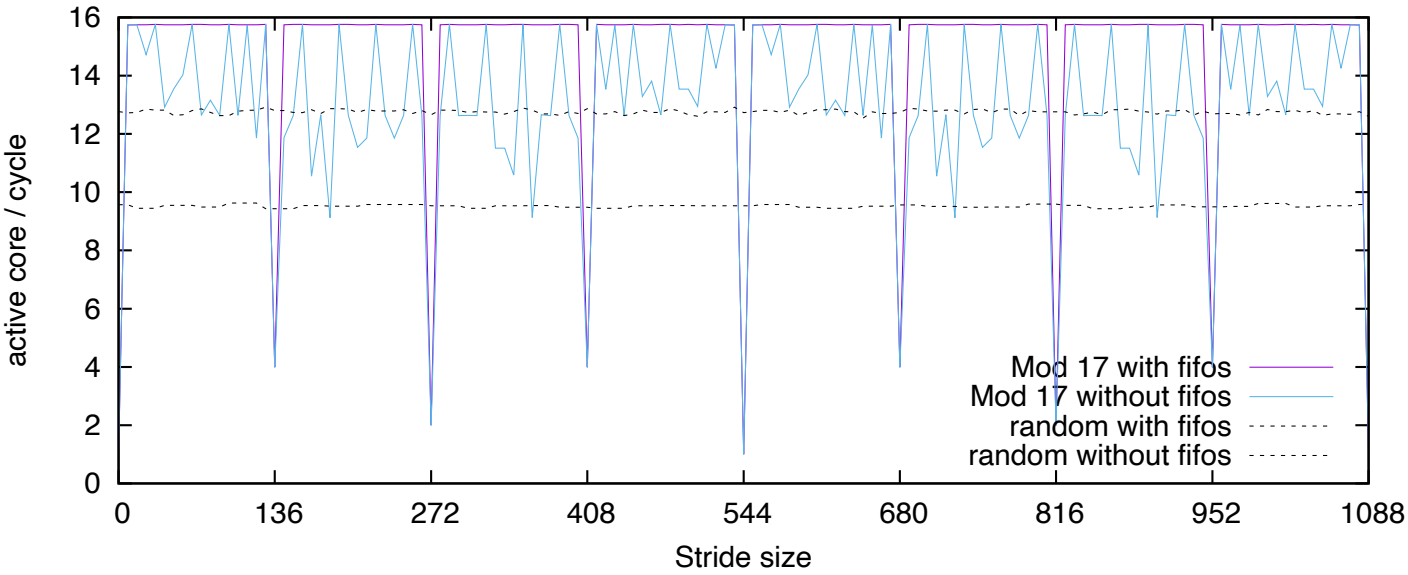
- Target architecture: 16 cores and 16 memory banks
- Pathological behaviors with peak-down pattern on stride sizes of the powers of two and their multiples
  - Example: the stride of 512 bytes indicates only 1 access per cycle for the 16 requests – all cores try to access the same bank
    - Core 0 -> array[0] - @0 in bank 0 ,
    - Core 1 -> array[512] - @1 in bank 0 ((512 ÷ 32) ÷ 16 = 1; (512 ÷ 32) mod16 = 0),
    - Core 2 -> array[1024] - @2 in bank 0 ((1024 ÷ 32) ÷ 16 = 2; (1024 ÷ 32) mod16 = 0),
    - . . .
- Good performance (with FIFO) for strides which are not powers of two



MOD17 indexing

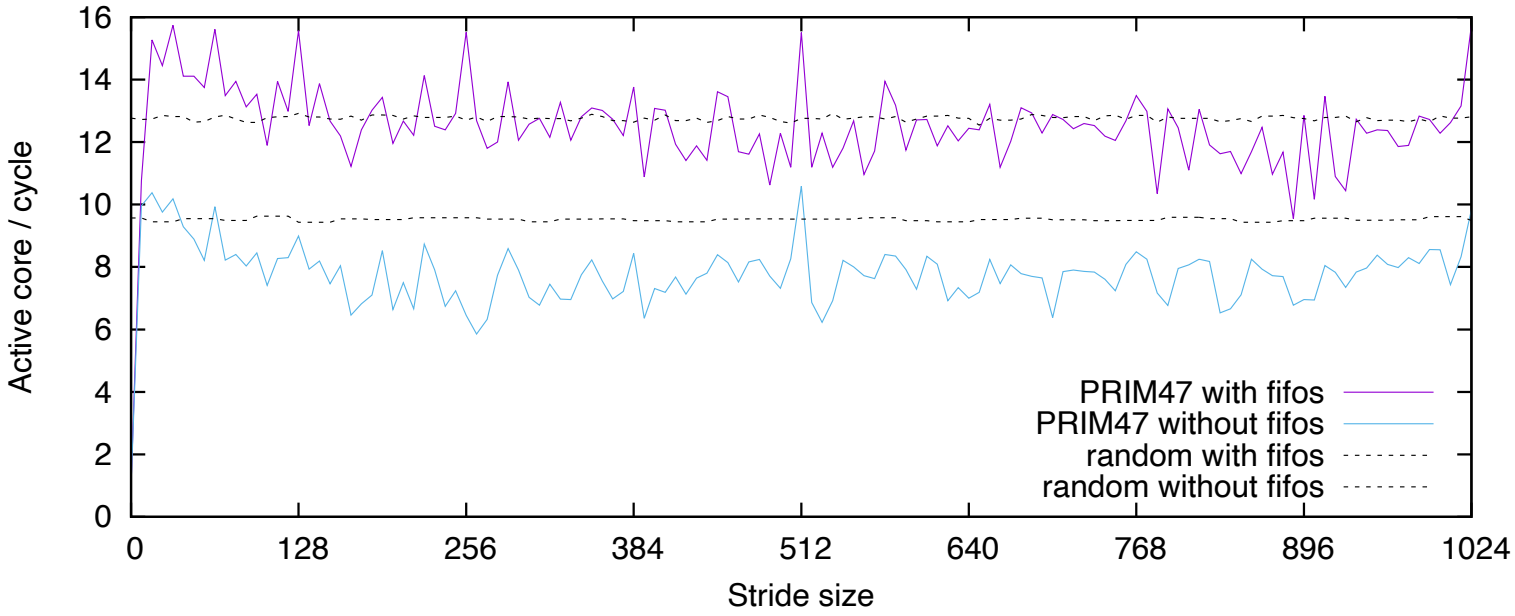
- Target architecture: 16 cores and 17 memory banks (prime number)

- We are looking to get good performance for strides of powers of two !
- Pathological behaviors with peak-down pattern aligned on multiples of 17 x 2^n
  - Example: the stride of 544 bytes indicates only 1 access per cycle for the 16 requested – all cores try to access the same bank
    - Core 0 -> array[0] - @0 in bank 0,
    - Core 1 -> array[544] - @1 in bank 0 ((544 ÷ 32) ÷ 17 = 1; (544 ÷ 32) mod17 = 0),
    - Core 2 -> array[1088] - @2 in bank 0 ((1088 ÷ 32) ÷ 17 = 2; (1088 ÷ 32) mod17 = 0),
    - . . .
- Less pathological accesses (and never for strides of powers of two)



PRIM47 indexing

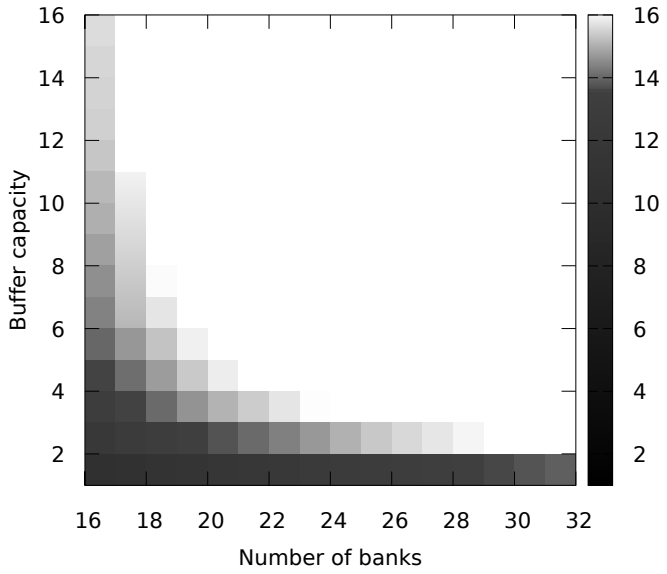
- Target architecture: 16 cores and 16 memory banks
- Pseudo-Randomly Interleaved Memory indexed
  - PRIM 47
- In average, worse than MOD16 and MOD17, but it does not suffer from pathological accesses, all strides have approximately the same performance
- Very good performance for strides powers of 2 (128, 256, 512)



What about random access patterns ?

- Previous approaches are based on predictable access patterns

- Approaches for random access patterns
  - Add of memory banks
    - It reduces collisions but has a HW cost
  - Access re-ordering (add of a re-ordering buffer – equ to load buffer)
    - Re-ordering access when collisions occur, but has an impact on memory consistency
    - Incresing buffer size allows to gain in performance, but it is limited by mem. bank collisions
  - Both compatible with interleaving scheme or hash function distribution



Better results when all approaches are combined

Analyse and conclusion

- There is no universal solution
  - The comparative results of these methods do not show an advantage for a method on all criteria: complexity / effectiveness / usability
  - Hash method such as PRIM or MOD should be combined with other architectural changes, such as adding more memory banks or reordering memory accesses for a more general use

Distributing Memory Bank Accesses in Many-Core Architectures: Hardware approaches

Arthur Vianes and **Frédéric Rousseau**

Kalray, France

Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, France

KALRAY
WE ACCELERATE YOUR DATA