Compiling for notifying memories: issues and challenges

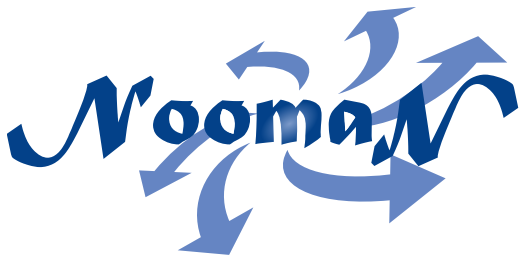
Compiler pour les *Notifying Memories* : enjeux et défis

Kevin J. M. Martin

Univ. Bretagne-Sud
UMR CNRS 6285, Lab-STICC
Lorient, France

11/06/2019

Journées nationales du GDR GPL 2019



Outline

Outline

What memory is needed for?

- storing data
- storing instructions
- saving temporary values
- synchronizing processes/threads

Context

It's the memory, stupid!

- more than 80% of the chip area is dedicated to caches, memories, memory controllers, interconnects and so on, whose sole purpose is to buffer data or control the buffering of data [1]
- $\Rightarrow$ workarounds which are making systems ever more complex
- more than 62% of the entire measured system energy is spent on moving data between memory and the computation units [1]

Context

It's the memory, stupid!

- more than 80% of the chip area is dedicated to caches, memories, memory controllers, interconnects and so on, whose sole purpose is to buffer data or control the buffering of data [1]
- $\Rightarrow$ workarounds which are making systems ever more complex
- more than 62% of the entire measured system energy is spent on moving data between memory and the computation units [1]

Enabling the continued performance scaling of smaller systems requires significant research breakthroughs in three key areas [3]

1. power efficiency
2. programmability
3. execution granularity

Topical subject

Stop or reduce moving data

Process the data where it is: in the memory!

Topical subject

Stop or reduce moving data

Process the data where it is: in the memory!

Sort this out!

Computing-In-Memory, Processing In Memory, In-memory computing, Logic In Memory, Near-Memory Computing, Intelligent Memory, Smart memories, Near-memory processing, Active memory, Memory-driven computing

Topical subject

Active Research

Displaying results **1-25** of **119** for **(("Document Title":in-memory) AND "Document Title":processing)** ×
▾ **Filters Applied:** Conferences × Journals & Magazines ×

Displaying results **1-25** of **136** for **(("Document Title":in-memory) AND "Document Title":computing)** ×
▾ **Filters Applied:** Conferences × Journals & Magazines ×

Year	In-memory AND processing	In-memory AND computing
2019 (up to April)	10	8
2018	30	51
2017	31	32
2016	17	18
2015	9	16
2014	3	4
2013	1	4
2012	0	0
2011	0	0
1995-2010	18	3

Context

Related work

- Processing In Memory (PIM) [Gokhale 1995]: Offload computation in the memory [5]
- Intelligent Memory [Kozyrakis 1997] [6]
- Smart memory [Mai 2000]: Modular reconfigurable architecture [7]
- Active memory processor [Yoo 2012] [9]
- Logic In Memory [Gaillardon 2016]: Fine grained, Technology dependent [4]
- Compute cache [Aga 2017] [2]
- Near Memory Computing [NeMeCo]
- Computation-In-Memory [MNEMOSENE]

Context

Related work

- Processing In Memory (PIM) [Gokhale 1995]: Offload computation in the memory [5]
- Intelligent Memory [Kozyrakis 1997] [6]
- Smart memory [Mai 2000]: Modular reconfigurable architecture [7]
- Active memory processor [Yoo 2012] [9]
- Logic In Memory [Gaillardon 2016]: Fine grained, Technology dependent [4]
- Compute cache [Aga 2017] [2]
- Near Memory Computing [NeMeCo]
- Computation-In-Memory [MNEMOSENE]

In all cases

- Memory are slaves
- No notification feature

What about programmability?

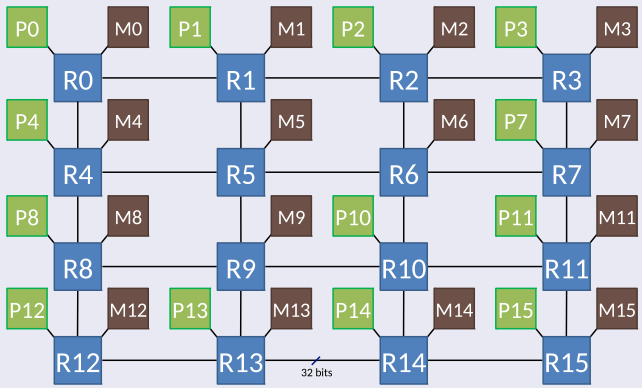
What about execution granularity?

Outline

What are *Notifying Memories*?

Application domain

Network on Chip



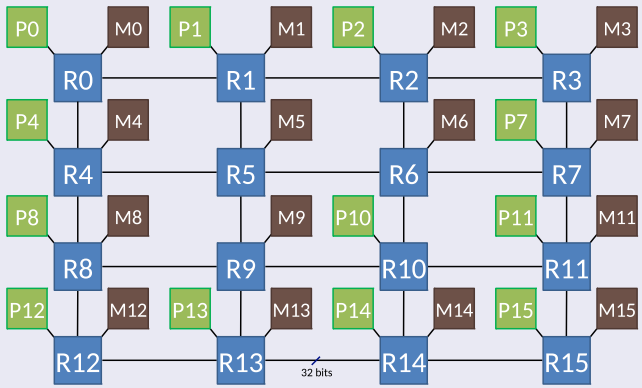
Network on Chip

- × Long latency
- × Sometimes useless for data-flow
- × High Energy consumption (up to 40%)

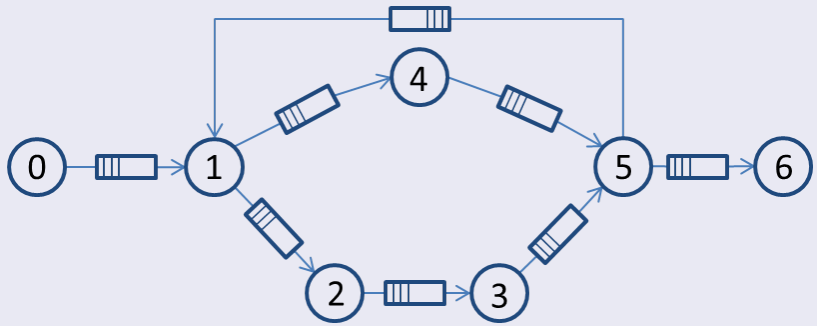
What are *Notifying Memories*?

Application domain

Network on Chip



Data-flow application



Network on Chip

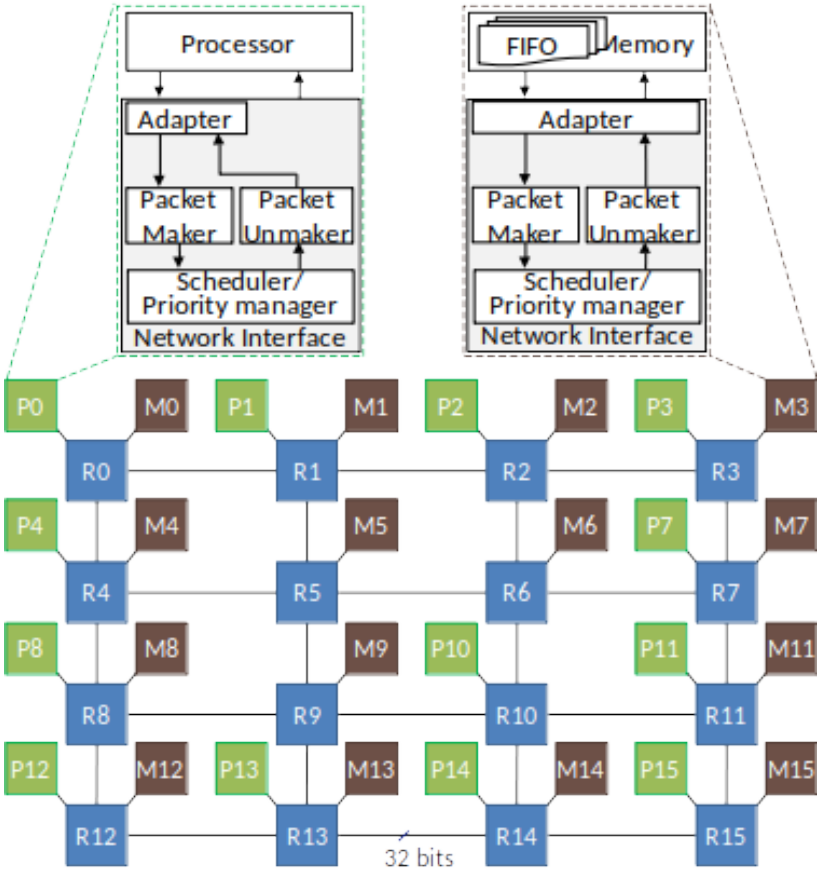
- × Long latency
- × Sometimes useless for data-flow
- × High Energy consumption (up to 40%)

Memory request

- × processor initiates transactions
- × the memory replies
- × several times the same data

Application domain

Network on Chip

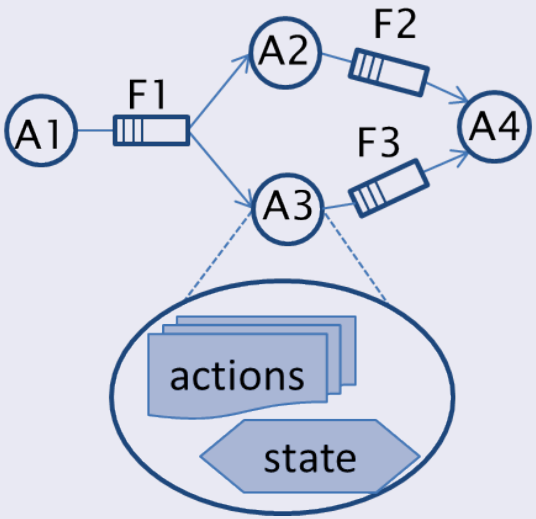


- Interconnection network
  - Routers
  - Network interface
- High bandwidth
- Long latency
- High energy consumption

Application domain

Dynamic Dataflow

Network of actors



Dataflow

- Formal Model Of Computation
- Explicit spatial and temporal parallelism
- Static or dynamic actors
  - Execute actions (”fire” actions)
- Firing rule
  - Enough tokens in input FIFOs
  - Enough space in output FIFOs

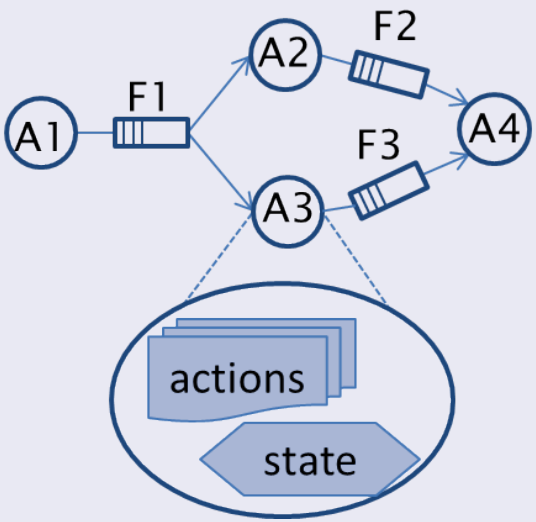
Static actors

- Fixed number of consumed and produced tokens
- Can be solved at compile time

Application domain

Dynamic Dataflow

Network of actors



Dataflow

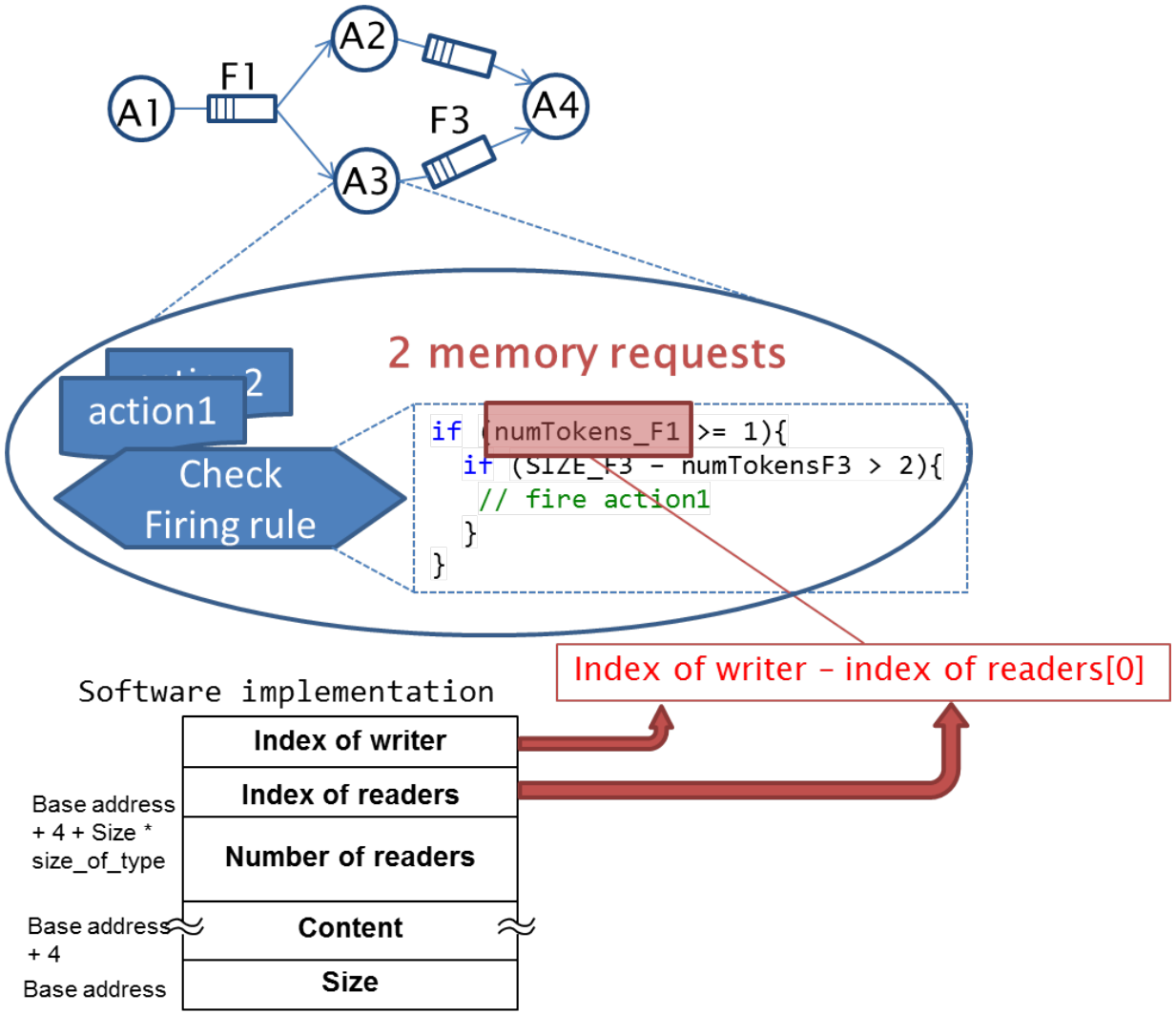
- Formal Model Of Computation
- Explicit spatial and temporal parallelism
- Static or dynamic actors
  - Execute actions (”fire” actions)
- Firing rule
  - Enough tokens in input FIFOs
  - Enough space in output FIFOs

Dynamic actors

- Variable number of consumed /produced tokens
- Must be solved at run time

Application domain

Dynamic Dataflow



Execution model

- 2 memory requests per FIFO
- If no action fired, the same requests are made again and again
- NoC Latency = Huge penalty for Polling

Motivational example

Motivational example

Unsuccessful scheduling by the MPEG4-SP decoder for different video sequences and formats

Video		Useless attempt	Empty input FIFO	Full output FIFO
Sequence	**Format**			
Akiyo	CIF	42.7%	63.7%	36.3%
Parkjoy	720p	21.3%	90.8%	9.2%
Foreman	CIF	34.8%	90.7%	9.3%
Coastguard	CIF	27.8%	98.4%	1.6%
Stefan	CIF	25.9%	83.3%	16.7%
Bridge far	QCIF	23.8%	38.4%	61.6%
Ice	4CIF	45.6%	70.4%	29.6%

Useless Memory Accesses through + long NoC latency Penalty

Motivational example

Motivational example

Unsuccessful scheduling by the MPEG4-SP decoder for different video sequences and formats

Video		Useless attempt	Empty input FIFO	Full output FIFO
Sequence	**Format**			
Akiyo	CIF	42.7%	63.7%	36.3%
Parkjoy	720p	21.3%	90.8%	9.2%
Foreman	CIF	34.8%	90.7%	9.3%
Coastguard	CIF	27.8%	98.4%	1.6%
Stefan	CIF	25.9%	83.3%	16.7%
Bridge far	QCIF	23.8%	38.4%	61.6%
Ice	4CIF	45.6%	70.4%	29.6%

Useless Memory Accesses through + long NoC latency Penalty

Monitor FIFOs and emit notifications about their status

Notifying Memories Concept

Observer design pattern (software engineering)

- Subject: sends the notifications
- Observer: reacts to notifications

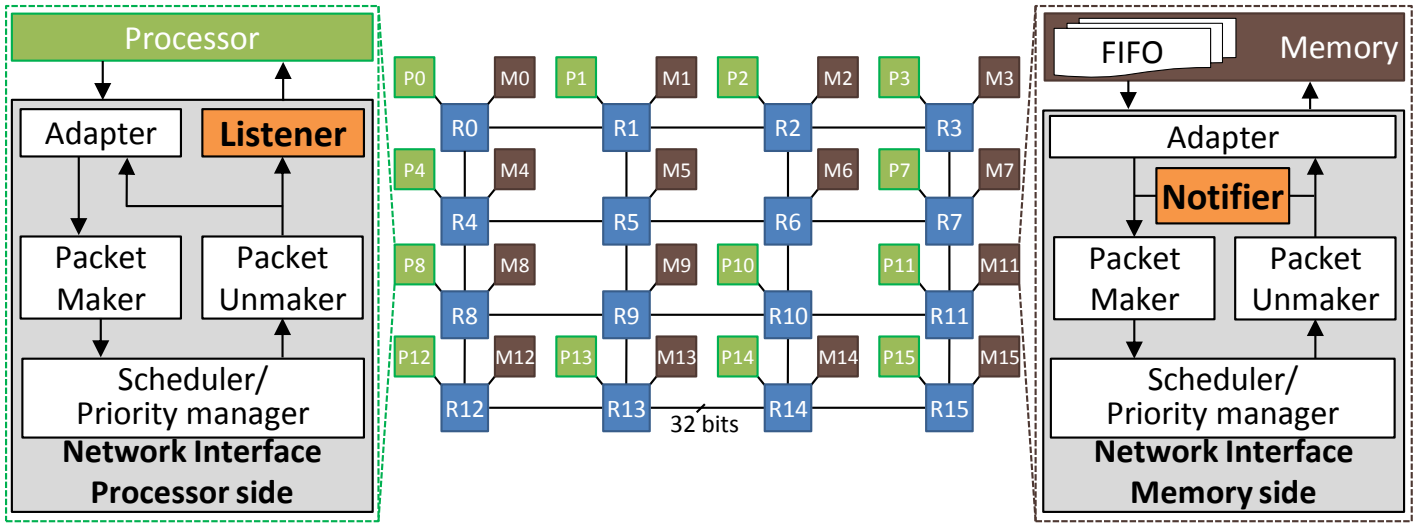
Implementation in the Network Interface (NI)

- Master component that can send packets through the network
- Component that can monitor requests
- Independent from processor, memory, NoC parameters
- The subject is the memory (becomes master)
- The listener is the processor (becomes slave)

Notifying Memories Concept

Listener and notifier: new components of Network Interface

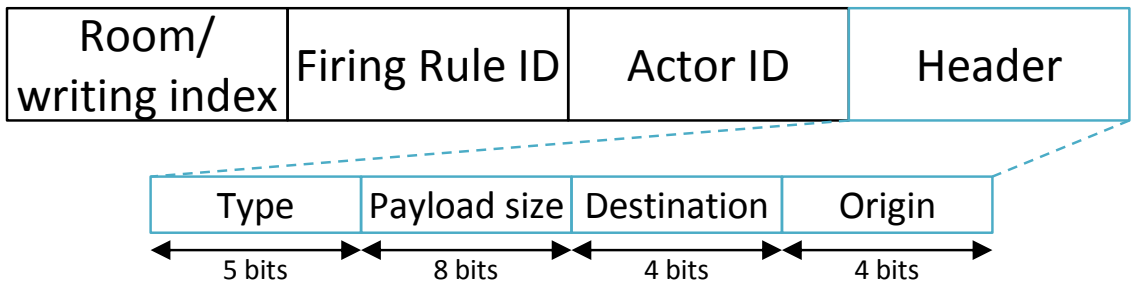
- Notifier on memory side
- Listener on processor side



Notifying Memories Concept

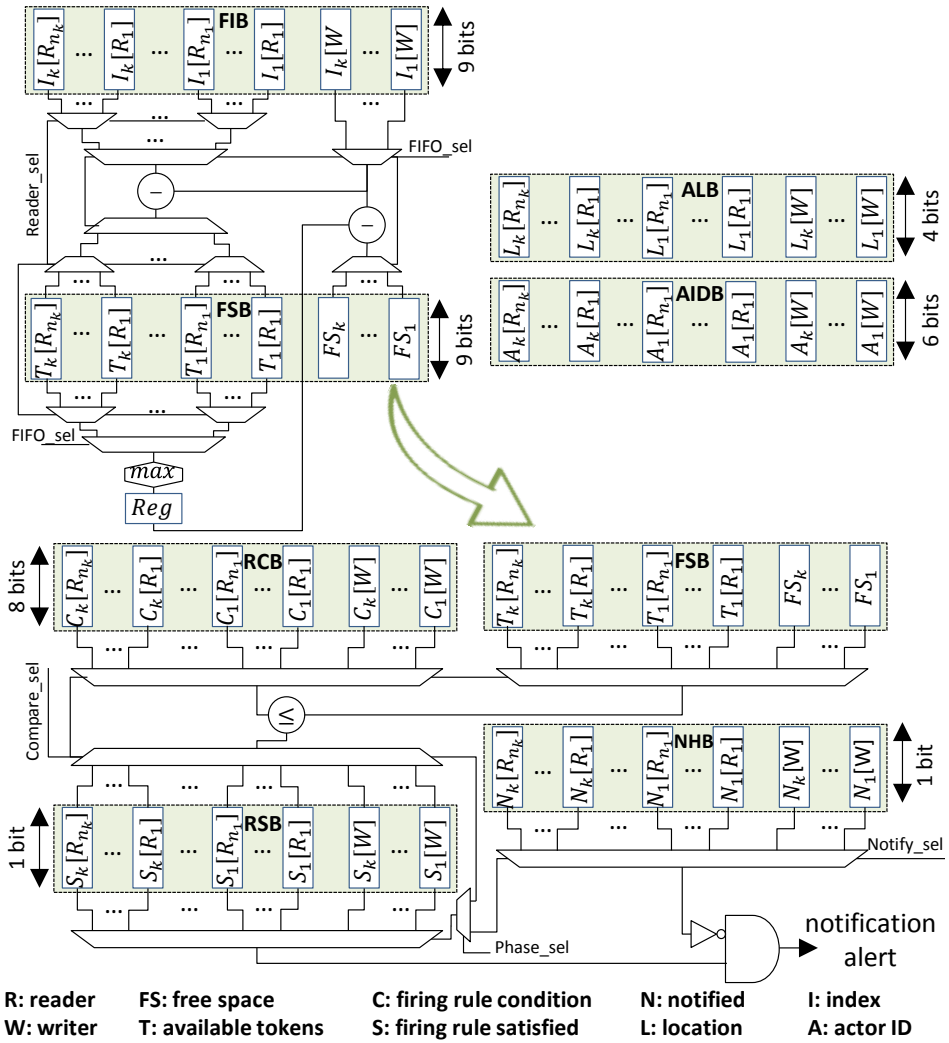
The notifier

1. Configuration phase
   - Specify what FIFOs to monitor
2. Checking phase
   - “Packet sniffer”: retrieves indexes of writers and readers Computes the number of tokens in a FIFO
3. Notification phase
   - Provides the packet maker with the target location, identity number, satisfied firing rule identity number, and the number of available tokens or free space



Notifying Memories Concept

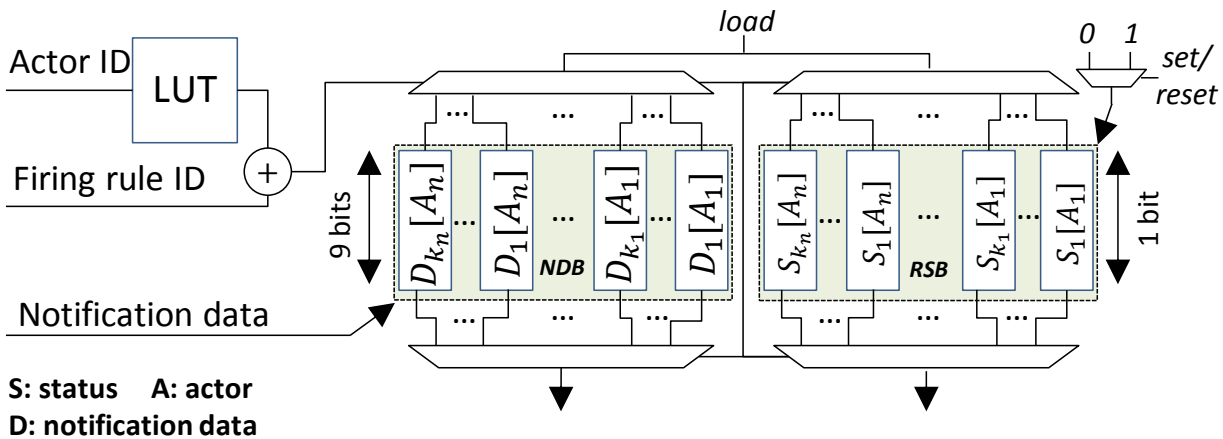
The notifier



Notifying Memories Concept

The listener

1. Configuration phase
   - Specifies what notification to listen to
2. Execution phase
   - Sets the firing rule validity bit when a notification is caught
   - Clears the validity bit when the action is performed



Experiments and results

Experimental Setup

NoC

- 4x4 mesh-based SystemC cycle-accurate model
- 13 processors, 12 memories
- Wormhole packet switching, XY routing algorithm
- Routers: one arbiter per port, one buffer per input port
- Round robin

Application

- MPEG4-SP (H264) decoder
- 41 actors, 70 FIFOs

Mapping

- Manual mapping, minimize number of hops
- FIFOs equally distributed in memories

Experiments and results

Results

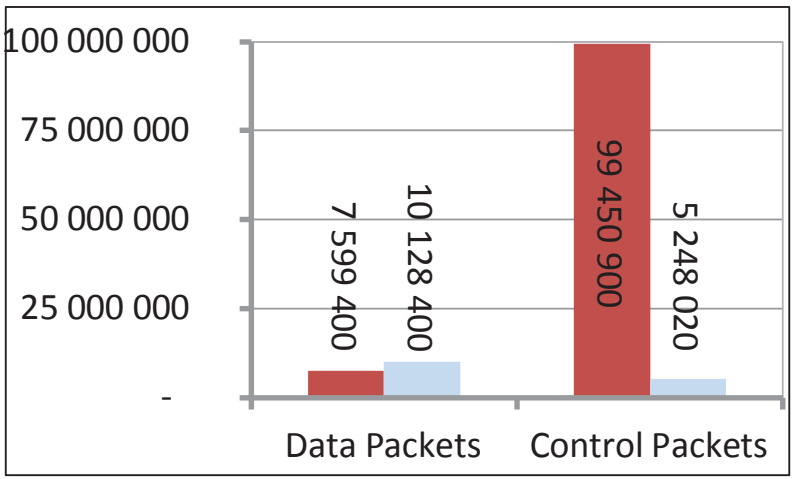
Results of decoding 10 frames of ice video sequence in 4CIF format

Parameter	**Notifying memory**	**Ordinary memory**	**gain**
Latency (μs)	143.42	665.06	-78.44%
Throughput (frames/s)	27.53	23.29	+15.41%
Injection rate(flits/s)	60 167 732	121 635 294	-50.53%
Switch conflicts	71 182 509	288 574 519	-75.33%
Transported flits	109 264 000	261 123 000	-58.16%
Transported packets	15 376 400	107 050 000	-85.64%

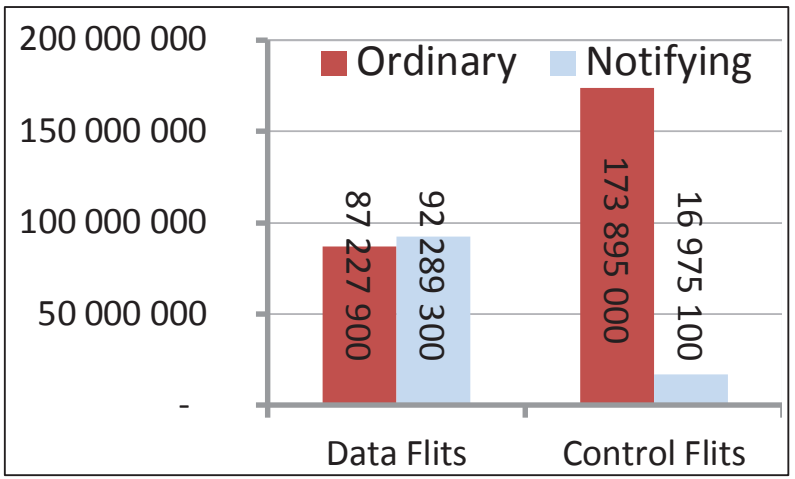
Experiments and results

Results

- Data packets: tokens
- Control packets: mapping information, memory requests, notifications



(a)

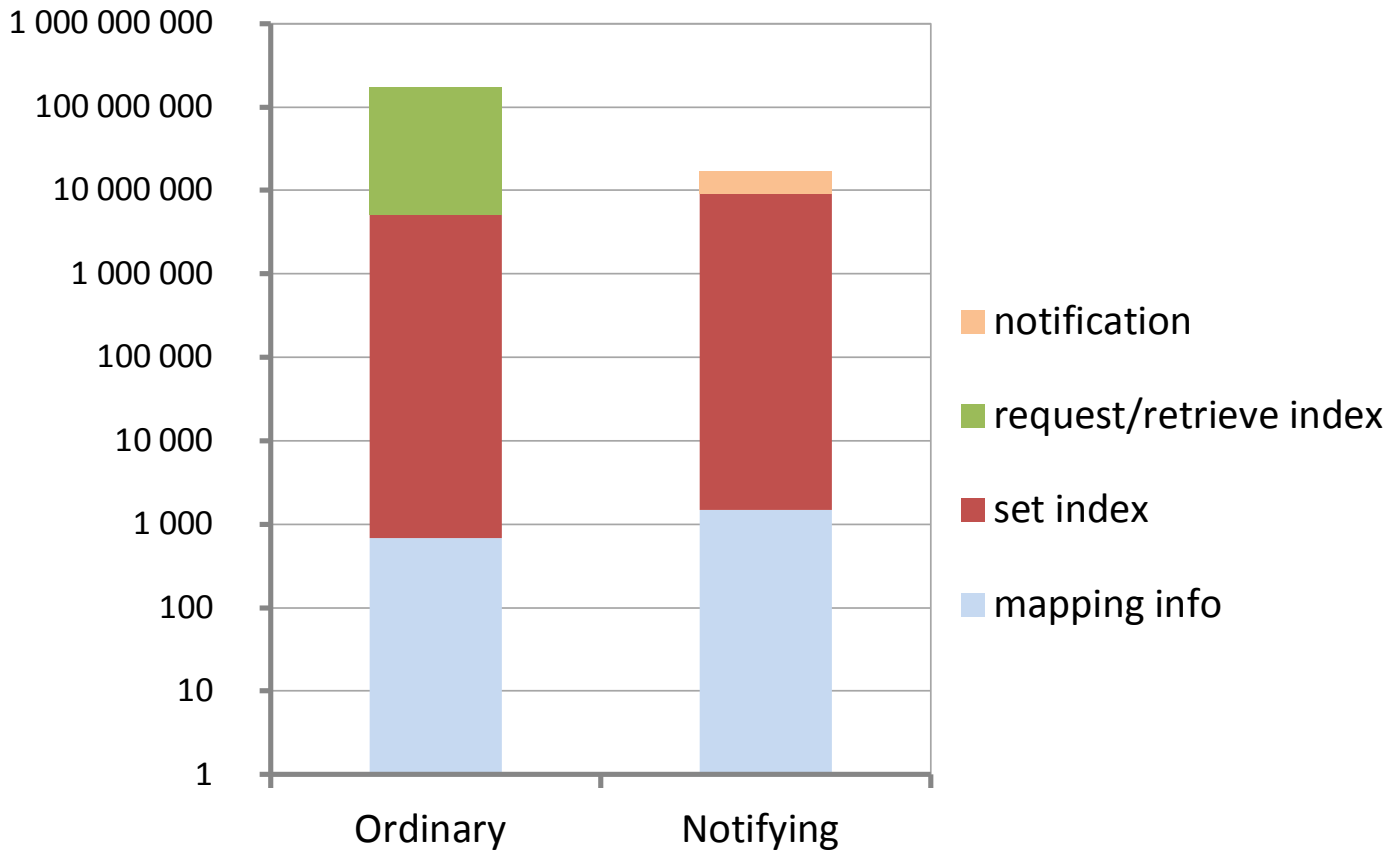


(b)

Experiments and results

Results

- Control flits classification



Experiments and results

Results

Notification memory gain for decoding 10 frames of different video sequences

Video		Throughput	Latency	Injection rate	Switch conflicts	Flits number
Sequence	**Format**					
Bridgefar	QCIF	+15.53%	-73,96%	-45,80%	-71,38%	-54,22%
bus	CIF	+2.84%	-73,79%	-53,40%	-72,90%	-54,73%
grandma	QCIF	+16.79%	-68,96%	-60,78%	-85,50%	-67,36%
foreman	CIF	+14.26%	-78,41%	-46,81%	-72,86%	-54,39%
ice	4CIF	+15.41%	-78,44%	-50,53%	-75,33%	-58,16%

Experiments and results

Preliminary synthesis results

- Worst-case implementation
  - Same notifier in all memories: able to deal with the 70 FIFOs
  - Same listener in all processors: able to deal with the 41 actors
- Cadence Encounter RTL Compiler 65nm (500MHz, 25 degC)
- Leakage and dynamic power
- NoC adopting notifying memories saves 49.1% of energy
- Power overhead of notifying NoC is 16.3%
- Area overhead of notifying NoC is 12.4%

Notifying Memories

Wrap-up

- Notifying memories concept [8]
  - The memories send notifications to processors
  - Notifiers on memory side
  - Listeners on processor side
- SystemC model
  - Notifiers and listeners in the Network Interface of the NoC
  - New kind of packet : the notification packet
- Simulation results
  - Latency (-78%), injection rate (-60%)
- Synthesis results
  - Worst-case implementation
  - +12% area
  - -49% energy

Outline

Issues and challenges

Issues

Architectural assumptions

- Distributed memory architecture
- No cache
- Network on Chip

Issues and challenges

Issues

Architectural assumptions

- Distributed memory architecture
- No cache
- Network on Chip

What about programmability?

- Code handly adapted (rewritten)
- Code handly ”partitioned”
- Code handly mapped

Issues and challenges

Challenges

Automation

Need for a compilation tool to automatically generate:

- Binary code of the application
  - Automatic ”partitionning”
  - Mapping and scheduling?
- Configurations for notifiers and listeners
  - Rules to check on notifier side
  - Rules to catch on listener side

Conclusion

Breaking memory organisation comes with huge challenges for a compilation tool

- How to automatically make use of Notifying Memories?
- How to automatically ”partition” the code?
- How to automatically configure the rules to check and catch?

Conclusion

Breaking memory organisation comes with huge challenges for a compilation tool

- How to automatically make use of Notifying Memories?
- How to automatically ”partition” the code?
- How to automatically configure the rules to check and catch?

Dataflow features

- Firing rules are known
- Isolated in the internal representation

References I

[1] ‘It’s the memory, stupid’: A conversation with Onur Mutlu - Press.

[2] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das.
Compute Caches.
In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492, Feb. 2017.

[3] B. Dally.
Power, Programmability, and Granularity: The Challenges of ExaScale Computing.
In *2011 IEEE International Parallel Distributed Processing Symposium*, pages 878–878, May 2011.

[4] P. E. Gaillardon, L. Amarò, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. D. Micheli.
The Programmable Logic-in-Memory (PLiM) computer.
In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 427–432, Mar. 2016.

[5] M. Gokhale, B. Holmes, and K. Iobst.
Processing in memory: the Terasys massively parallel PIM array.
Computer, 28(4):23–31, Apr. 1995.

[6] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick.
Scalable processors in the billion-transistor era: IRAM.
Computer, 30(9):75–78, Sept. 1997.

[7] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz.
Smart memories: A modular reconfigurable architecture.
In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA ’00, pages 161–171, New York, NY, USA, 2000. ACM.

[8] K. J. M. Martin, M. Rizk, M. J. Sepulveda, and J.-P. Diguet.
Notifying memories: A case-study on data-flow applications with NoC interfaces implementation.
In *53rd DAC Conf.*, New York, NY, USA, 2016. ACM.

[9] J. Yoo, S. Yoo, and K. Choi.
Active memory processor for network-on-chip-based architecture.
IEEE Transactions on Computers, 61(5):622–635, May 2012.