WiDir: A Wireless-Enabled Directory Cache Coherence Protocol

Antonio Franques¹ (franque2@illinois.edu), Apostolos Kokolis¹, Sergi Abadal², Vimuth Fernando¹, Sasa Misailovic¹, Josep Torrellas¹

¹University of Illinois at Urbana-Champaign  ²Universitat Politècnica de Catalunya

CCF-1629431  863337 (WiPLASH)
• Current trends are leading to larger manycores
• **Wireless on-chip communication** holds promise for the implementation of fast networks for these multiprocessors
• In complement of a wired NoC, wireless provides
  - Low latency
  - Natural broadcast capabilities
  - Flexibility
On-chip wireless communication for communication-intensive data

Related Work: Replica [ASPLOS '19]

Core

Broadcast memory (up to 512 KB)

L1

L2 (LLC)

Wireless antenna and transceiver
Related Work: Replica [ASPLOS '19]

On-chip wireless communication for communication-intensive data

Master_Thread
... barrier_wait(b)
...
On-chip wireless communication for communication-intensive data

Related Work: Replica\textsuperscript{[ASPLOS '19]}

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
On-chip wireless communication for communication-intensive data

Related Work: Replica \[ASPLOS '19\]

On-chip wireless communication for communication-intensive data
Challenges:

- Bounded size of the broadcast memory (BMem)
  - X Arbitrary data structures do not fit
- Need profiling tools to identify what data to put in BMem
- BMem area/energy overhead (≈ size L2 cache)
- BMem data is replicated to all cores

Related Work: Replica [ASPLOS '19]
Contribution: *WiDir*

- *WiDir*: a directory cache coherence protocol augmented with wireless transactions for highly shared data
  - ✓ 2 networks (wired mesh + wireless), 1 directory
- *WiDir* leverages per-line directory info to identify highly-shared data
- *WiDir* uses an additional state called *Wireless*
  - ✓ Unified memory hierarchy. No BMem needed
- Cache lines dynamically transition wired↔wireless based on access pattern
  - ✓ Transparent to programmer
WiDir Architecture

- **Transceiver**: modulates packet bits into wireless signals and back
- **Data+Tone Antennas**: data channel used for data transmission; tone channel used for acknowledgement (ACK) mechanism
WiDir: Wireless Directory

- **Cache coherence state**: add Wireless state
- Directory **pointer bits** used in wired states are repurposed into **number of sharers** in wireless state
- Add 2-bit **update counter** per cache line in each core to track line accesses
**WiDir: Switching Wired ⟷ Wireless**

- **Baseline problem:** limited number of sharer pointers
  - If sharers overflow, set broadcast bit. If a write comes, invalidate everyone
- **WiDir solution:** if sharers overflow, switch to Wireless state
  - Then repurpose sharers pointers into sharers counter *(SharersCount)* to track number of sharers for the line
  - Every time a node is added to wireless pool → directory increments *SharersCount*
  - If a write comes, broadcast updated word through wireless
WiDir: Switching Wired ←→ Wireless

• **Problem:** if the line is not highly shared anymore, wireless broadcasts are counterproductive
  • Wireless bandwidth is limited. Contention must remain as low as possible
• **WiDir solution:** Use update counter *(UpdateCount)* to track number of local accesses to wireless lines
  • Every time a node receives a wireless update → increments local *UpdateCount*
  • When the node accesses the line → resets *UpdateCount* to 0
  • If *UpdateCount* reaches threshold → cache invalidates line and notifies directory
  • When directory receives notification → decrements *SharersCount*
  • If *SharersCount* below threshold → directory broadcasts message to all sharers, and switch back to wired *Shared* state
Baseline: Cache Coherence Algorithm

- **D**: Dirty bit
- **B**: Broadcast bit
- **P**: Sharers pointers

![Diagram of Cache Coherence Algorithm]

**Legend**:
- **Shared**: Indicates shared data
- **Not present**: Indicates data not present in the cache
- **Main Mem**: Main memory
- **x = 5**: State indication

**Diagram Description**:
- **Core1**: L1 cache
- **Core2**: L1 cache
- **Core3**: L1 cache
- **Core4**: L1 cache
- **Wired Network**: Connection between cores and main memory
- **LLC**: Last level cache
- **Dir**: Directory
- **State**: Shared data state
- **Data**: Data table with states

**Example State**:
- **LLC**: State = Shared
- **Dir**: State = Shared
- **Core1**: L1 cache is shared
- **Core2**: L1 cache is not present
- **Core3**: L1 cache is shared
- **Core4**: L1 cache is not present
Baseline: Sharer Pointers Overflow

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

Read x
Baseline: Sharer Pointers Overflow

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

Read x

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
Baseline: Sharer Pointers Overflow

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

Read x

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
Baseline: Sharer Pointers Overflow

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
What would happen now in standard MESI if a write came?

(Hint: write-invalidate)
Baseline: Write Miss

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

Write x

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
Baseline: Write Miss

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
Baseline: Write Miss

D: Dirty bit
B: Broadcast bit
P: Sharers pointers
Baseline: Write Miss

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

Write $x$
Baseline: Write Miss

D: Dirty bit
B: Broadcast bit
P: Sharers pointers
Baseline: Write Miss

D: Dirty bit
B: Broadcast bit
P: Sharers pointers

Write x
Baseline: Write Miss

Problems:
- Multi-step process with high latency (especially as core count increases)
- Particularly bad for producer-consumer pattern where data is frequently read/written (each write invalidates all sharers, then all suffer read miss)
- Even worse if we have more sharers than pointers (if pointers overflow → invalidation broadcast)

D: Dirty bit
B: Broadcast bit
P: Sharers pointers
Baseline: Write Miss

Solution (**WiDir**):
- Update all sharers of a wireless line upon a write
  - No more read-misses after each write
- Use wireless to broadcast **word** updates
  - Single hop regardless of core count
WiDir: Cache Coherence Algorithm

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir: Write Miss + Switch to Wireless**

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

![Diagram of WiDir system](image)

**Wired Network**

- **Dir**  
  - Data  
  - D  
  - P/C  
  - State  
    - Shared

- **Wired Network**
  - Core1  
    - Shared  
    - L1
  - Core2  
    - Not present  
    - L1
  - Core3  
    - Shared  
    - L1
  - Core4  
    - Not present  
    - L1

**Write x**

1 March 2021  
Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir: Write Miss + Switch to Wireless**

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

1 March 2021  
Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir: Write Miss + Switch to Wireless**

How does the directory know all cores received and processed a wireless upgrade request? 😐

We need a fast mechanism to obtain acknowledgements in wireless!

---

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter
WiDir: Wireless Channels

![Data channel and Tone channel diagram](chart.png)
**WiDir: Tone Channel**

**STEP 1:** A node broadcasts a message through data channel that requires acknowledgement from everybody.

- = Triggering node
- = Node with tone **OFF**
- = Node with tone **ON**
**WiDir: Tone Channel**

**STEP 2:** The message triggers all remote nodes to turn their tone channel ON

- **Gray circle** = Triggering node
- **Light blue circle** = Node with tone OFF
- **Dark blue circle** = Node with tone ON
**WiDir: Tone Channel**

**STEP 3:** After a node has finished processing the message, it turns its tone channel OFF

- = Triggering node
- = Node with tone **OFF**
- = Node with tone **ON**
**WiDir: Tone Channel**

**STEP 4:** When triggering node doesn’t sense any energy in the tone channel, it knows everyone processed message

- **Gray circle** = Triggering node
- **White circle** = Node with tone **OFF**
- **Blue circle** = Node with tone **ON**
WiDir: Tone Channel

**STEP 4:** When triggering node doesn’t sense any energy in the tone channel, it knows everyone processed message.

- Acts as a collective **binary OR** (‘0’ for all silent, ‘1’ for someone transmitting)
- Inexpensive **ACK mechanism**
- Great scalability. It works for **any core count**

- Triggering node
- Node with tone **OFF**
- Node with tone **ON**
**WiDir: Write Miss + Switch to Wireless**

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

---

1 March 2021  
Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir: Write Miss + Switch to Wireless**

D: Dirty bit
P: Sharers pointers
C: Sharers counter

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir: Write Miss + Switch to Wireless**

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

1 March 2021  
Antonio Franques – University of Illinois at Urbana-Champaign
WiDir: Write Miss + Switch to Wireless

- **D**: Dirty bit
- **P**: Sharers pointers
- **C**: Sharers counter

**Diagram:**
- Wired Network
- LLC
- Dir
- Data
- D
- P/C
- State
- Shared→Wireless

**Network Diagram:**
- Wireless L1
- Not present L1
- Wireless L1
- Wireless L1

**Write x:**
**WiDir: Write Miss + Switch to Wireless**

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

![Diagram of WiDir system showing write miss and switch to wireless]

1 March 2021  
Antonio Franques – University of Illinois at Urbana-Champaign
WiDir: Writing to a Wireless Line

D: Dirty bit
P: Sharers pointers
C: Sharers counter

Write $x$
WiDir: Writing to a Wireless Line

D: Dirty bit
P: Sharers pointers
C: Sharers counter

Write x
**WiDir: Writing to a Wireless Line**

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

![Diagram showing the WiDir system architecture with Wireless L1 and Core nodes connected through the Wired Network.](Diagram)

1 March 2021  
Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir: Adding a New Sharer**

- **D**: Dirty bit
- **P**: Sharers pointers
- **C**: Sharers counter

---

**Wired Network**

- **Core1**
- **Core2**
- **Core3**
- **Core4**

**Data**

- **D**: 
- **P/C**: 

**State**

- **Wireless**

---

**Read x**

1 March 2021

Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir**: Adding a New Sharer

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

---

1 March 2021
Antonio Franques – University of Illinois at Urbana-Champaign 46
**WiDir:** Adding a New Sharer

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

---

**Diagram Description:**

- **Data:** Wireless
- **D:** Wireless (Blocked)
- **P/C:** x = 9
- **State:** Wireless

**Wireless Network:**

1. **Core1:** Wireless L1  
2. **Core2:** Inv→Shared L1  
3. **Core3:** Wireless L1  
4. **Core4:** Wireless L1

**Main Mem:** x = 5

**Read x**
**WiDir: Adding a New Sharer**

- **D:** Dirty bit
- **P:** Sharers pointers
- **C:** Sharers counter

![Diagram of WiDir: Adding a New Sharer](image)

**Main Mem**

**Wireless**

**L1**

**Core1**

**Core2**

**Core3**

**Core4**

**LLC**

**Dir**

**State**

Wireless (Blocked)

**Data**

<table>
<thead>
<tr>
<th>x = 9</th>
<th>D</th>
<th>P/C</th>
</tr>
</thead>
</table>

**Read x**

1 March 2021

Antonio Franques – University of Illinois at Urbana-Champaign
**WiDir: Adding a New Sharer**

D: Dirty bit  
P: Sharers pointers  
C: Sharers counter

---

1 March 2021

Antonio Franques – University of Illinois at Urbana-Champaign
How can the directory prevent everyone from updating the line?

- **Option 1:** By sending noise non-stop in the data channel
  - ✔ It is simple to implement
  - ✗ Uses a lot of energy
  - ✗ Prevents other non-blocked wireless lines from being updated
Background: Medium access control (MAC) protocol forces all transmissions to leave idle cycle between header and payload
  • If any core detects a collision after the header, it broadcasts a NACK (negative-ACK) to report it
  • The cores who were transmitting stop transmission and retry later
WiDir: Wireless Jamming Mechanism

How can the directory prevent everyone from updating the line?

- **Option 2:** By leveraging the collision mechanism of the MAC protocol
  - Directory checks if (partial) address in header is a multiple of a blocked line
    - If so, broadcast NACK, effectively jamming the update
  - Only jams blocked lines
  - Partial address matching could lead to false positives, but it is much better than blocking ALL lines
**WiDir: Evaluation Methodology**

- Cycle-level architectural simulations using SST
  - 64 out-of-order cores
  - Inclusive 2-level cache
  - 2D mesh wired network (1 cycle/hop) + wireless network (20 Gbps)

- Applications
  - 20 benchmarks from SPLASH-3 and PARSEC suites
  - Multiple domain: Scientific computing, computer vision, kernels, etc.
**WiDir** reduces total latency in nearly all applications *(35% on average)*

- Reductions are similar in both loads and stores
In *Baseline*, many cycles are wasted to memory stalls

✓ *WiDir* reduces mem stall cycles in many applications *(1/3 on average)*

![Execution Time Chart]

22%
WiDir: Also in the Paper...

• Description of all possible state transitions
• Further analysis on...
  • Data sharing
  • Cache misses
  • Distribution of wired network hops
  • Power, energy and area consumption
  • Sensitivity of architectural parameters
  • Scalability
WiDir: Conclusions

WiDir is a directory cache coherence protocol augmented with wireless transactions for highly shared data

- Leverages per-line directory information to identify highly-shared data
- Uses an additional state called Wireless to update sharers upon a write
- Has 2 networks (wired+wireless) and a single distributed directory
- Lines dynamically transition wired↔wireless based on access patterns

Thank you!

Presenter: Antonio Franques – franque2@illinois.edu (Open to job market)

This presentation and recording belong to the authors. No distribution is allowed without the authors' permission.