The Bulk Multicore Architecture for Programmability

Josep Torrellas
Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu
Acknowledgments

Key contributors:
• Luis Ceze
• Calin Cascaval
• James Tuck
• Pablo Montesinos
• Wonsun Ahn
• Milos Prvulovic
• Pin Zhou
• YY Zhou
• Jose Martinez
Challenges for Multicore Designers

- 100 cores/chip coming & there is little parallel SW
  - System architecture should support *programmable environment*
    - User-friendly concurrency and consistency models
    - Always-on production-run debugging
Challenges for Multicore Designers (II)

• Decreasing transistor size will make designing cores hard
  – Design rules too hard to satisfy manually → just shrink the core
  – Cores will become commodity
    • Big cores, small cores, specialized cores…
  – Innovation will be in cache hierarchy & network
Challenges for Multicore Designers (III)

• We will be adding accelerators:
  – Accelerators need to have the same (simple) interface to the cache coherent fabric as processors
  – Need simple memory consistency models
A Vision of Year 2015-2018 Multicore

• 128+ cores per chip

• Simple shared-memory programming model(s):
  – Support for shared memory (perhaps in groups of procs)
  – Enforce interleaving restrictions imposed by the language & concurrency model
  – Sequential memory consistency model for simplicity

• Sophisticated always-on debugging environment
  – Deterministic replay of parallel programs with no log
  – Data race detection at production-run speed
  – Pervasive program monitoring
Proposal: The Bulk Multicore

- **Idea**: Eliminate the commit of individual instructions at a time
- **Mechanism**:
  - Default is processors commit *chunks* of instructions at a time (e.g. 2,000 *dynamic* instr)
  - Chunks execute *atomically* and in *isolation* (using buffering and undo)
  - Memory effects of chunks summarized in *HW signatures*
- **Advantages over current**:
  - Higher programmability
  - Higher performance
  - Simpler hardware
Rest of the Talk

• The Bulk Multicore
• How it improves programmability
• What’s next?
Hardware Mechanism: Signatures  [ISCA06]

- Hardware accumulates the addresses read/written in signatures

- Read and Write signatures
- Summarize the footprint of a Chunk of code
Signature Operations In Hardware

S₁ ∧ S₂

S = Ø?

Inexpensive Operations on Groups of Addresses

Josep Torrellas
The BULK Multicore Architecture
Executing Chunks Atomically & In Isolation: Simple!

\[ W_0 = \text{sig}(B, C) \]
\[ R_0 = \text{sig}(X, Y) \]
\[ W_1 = \text{sig}(T) \]
\[ R_1 = \text{sig}(B, C) \]

\[ W_0 \cap R_1 \lor (W_0 \cap W_1) \]

Thread 0

\[ \text{ld } X \]
\[ \text{st } B \]
\[ \text{st } C \]
\[ \text{ld } Y \]

\[ \textbf{commit} \]

Thread 1

\[ \text{ld } B \]
\[ \text{st } T \]
\[ \text{ld } C \]
Chunk Operation + Signatures: Bulk

- Execute each chunk **atomically and in isolation**
- (Distributed) arbiter ensures a **total order** of chunk commits

**Supports Sequential Consistency** [Lamport79]:
  - **Low hardware complexity**: Need not snoop ld buffer for consistency
  - **High performance**: Instructions are fully reordered by HW
    (loads and stores make it in any order to the sig)
Summary: Benefits of Bulk Multicore

• Gains in HW simplicity, performance, and programmability

• Hardware simplicity:
  – Memory consistency support moved away from core
  – Toward commodity cores
  – Easy to plug-in accelerators

• High performance:
  – HW reorders accesses heavily (intra- and inter-chunk)
Benefits of Bulk Multicore (II)

• High programmability:
  – Invisible to the programming model/language
  – Supports Sequential Consistency (SC)
    * Software correctness tools assume SC
  – Enables novel always-on debugging techniques
    * Only keep per-chunk state, not per-load/store state
      * Deterministic replay of parallel programs with no log
      * Data race detection at production-run speed
Benefits of Bulk Multicore (III)

- Extension: Signatures visible to SW through ISA
  - Enables **pervasive monitoring**
  - Enables **novel compiler opts**

Many novel programming/compiler/tool opportunities
Rest of the Talk

- The Bulk Multicore
- How it improves programmability
- What’s next?
Supports Sequential Consistency (SC)

- Correctness tools assume SC:
  - Verification tools that prove software correctness
- Under SC, semantics for data races are clear:
  - Easy specifications for safe languages
- Much easier to debug parallel codes (and design debuggers)
- Works with “hand-crafted” synchronization
Deterministic Replay of MP Execution

- During **Execution**: HW records into a log the order of dependences between threads
- The log has captured the “interleaving” of threads
- During **Replay**: Re-run the program
  - Enforcing the dependence orders in the log
Conventional Schemes

- Potentially large logs
Bulk: Log Necessary is Minuscule [ISCA08]

- During **Execution:**
  - Commit the instructions in chunks, not individually

If we **fix** the chunk commit interleaving:

Combined Log = NIL
Data Race Detection at Production-Run Speed  [ISCA03]

- If we detect communication between…
  - Ordered chunks: not a data race
  - Unordered chunks: data race
Different Synchronization Ops

- Lock L
- Unlock L
- Lock L
- Unlock L
- Lock L
- Set F
- Unlock L
- Set F
- Wait F
- Barrier
- Barrier
- Barrier
Benefits of Bulk Multicore (III)

- Extension: Signatures visible to SW through ISA
  - Enables pervasive monitoring [ISCA04]
    Support numerous watchpoints for free
  - Enables novel compiler opts [ASPLOS08]
    Function memoization
    Loop-invariant code motion
Pervasive Monitoring: Attaching a Monitor Function to Address

- Watch memory location
- Trigger monitoring function when it is accessed

```c
Watch(addr, usr_monitor)

usr_monitor(Addr) {
    ..... 
}
```

```
instr
Watch(addr, usr_monitor)
instr
*p = ...
instr
```

Thread
Main Program
*\( p = \) Rest of Program Monitoring Function
Enabling Novel Compiler Optimizations

New instruction: Begin/End collecting addresses into sig

```
bcollect Sig
... 
ld r0, x
ld r1, y
st r3, z
... 
... 
ecollect Sig
```
Enabling Novel Compiler Optimizations

New instruction: Begin/End collecting addresses into sig

```
bcollect Sig
...  
ld r0, x
ld r1, y
st r3, z
...  
...  
ecollect Sig

Hardware inserts reference addresses into signature

Collection

Sig
X, Y, Z
```
Instruction: Begin/End Disambiguation Against Sig

```
...  
...  
st r4, A
ld r5, B
st r6, C
...  
...  
```

```
bdisamb Sig
...  
edisamb Sig
```
Instruction: Begin/End Disambiguation Against Sig
Instruction: Begin/End Remote Disambiguation

Core 1

bdisamb_rem Sig

...
...
st r4, A
ld r5, B
st r6, C
...
...
edisamb_rem Sig
Instruction: Begin/End Remote Disambiguation
Example Opt: Function Memoization

- Goal: skip the execution of functions

```python
foo(x);
...
= y
z = ...
foo(x);
```
Example Opt: Function Memoization

- Goal: skip the execution of functions whose outputs are known
Example Opt: Loop-Invariant Code Motion

```c
while(...) {
    ...
    ... = <expr>
    ...
}
```
Example Opt: Loop-Invariant Code Motion

```c
while(...) {
    ...
    ... = <expr>
    ...
}
```

```c
if (conflict) {
    rollback()
    <original loop>
}
```
Rest of the Talk

- The Bulk Multicore
- How it improves programmability
- What’s next?
What is Going On?

**Bulk Multicore Hardware Architecture**

- Compiler (Matt Frank et al)
- Language support (Marc Snir, Vikram Adve et al)
- Libraries and run time systems
- Debugging Tools (Sam King, Darko Marinov et al)
- FPGA Prototype

Josep Torrellas
The BULK Multicore Architecture
Summary: The Bulk Multicore for Year 2015-2018

- 128+ cores/chip, shared-memory (perhaps in groups)

- Simple HW with commodity cores
  - Memory consistency checks moved away from the core

- High performance shared-memory programming model
  - Execution in programmer-transparent chunks
  - Signatures for disambiguation, cache coherence, and compiler opts
  - High-performance sequential consistency with simple HW

- High programmability: Sophisticated always-on debugging support
  - Deterministic replay of parallel programs with no log (DeLorean)
  - Data race detection for production runs (ReEnact)
  - Pervasive program monitoring (iWatcher)
The Bulk Multicore Architecture for Programmability

Josep Torrellas
Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu