Way Stealing: Cache-assisted Automatic Instruction Set Extensions

Theo Kluter (EPFL)
Philip Brisk (EPFL)
Paolo Ienne (EPFL)
Edoardo Charbon (EPFL,TU Delft)
Motivation

Set of Embedded Applications

How to automatically customize an Embedded Processor to support efficient execution of complex algorithms?
Motivation

Set of Embedded Applications

Automated Instruction Set Extensions
Motivation

Set of Embedded Applications

Automated Instruction Set Extensions

Cost function of software execution
Motivation

Set of Embedded Applications

Automated Instruction Set Extensions

Cost function of software execution

Cost function of hardware execution

Diagram:

- I$: Fetch
- RF: Decode
- Exec
- D$: Mem
- RF: WB
Automated Instruction Set Extensions

Motivation

Set of Embedded Applications

Cost function of software execution

Cost function of hardware execution

Number of Register File (RF) read and write ports are important parameters

I$ → Fetch → RF → Decode → Exec → D$ → Mem → RF → WB
Motivation

Automated Instruction Set Extensions

Set of Embedded Applications

Cost function of software execution

Cost function of hardware execution

Adds Hardware in form of an Application Specific Computational Unit (ASCU)
Automated Instruction Set Extensions

Set of Embedded Applications

Cost function of software execution

Cost function of hardware execution

Adds ASCU specific Custom Instruction(s)

Adds Hardware in form of an Application Specific Computational Unit (ASCU)

Motivation
Automated Instruction Set Extensions

Cost function of software execution

Cost function of hardware execution

The number of Register File (RF) read and write ports are restricting the bandwidth
Increasing the Data Bandwidth

Adding input bandwidth by Shadow Registers

J. Cong, et al. 2006
Increasing the Data Bandwidth

Adding input bandwidth by Register File Clustering

K. Karuri, et al. 2007
Increasing the Data Bandwidth

Adding input bandwidth by Exploiting Forwarding

J. Jayaseelan, et al. 2006
Increasing the Data Bandwidth

Adding input **AND** output bandwidth by Architecturally Visible Storage

P. Biswas, *et al.* 2006
The Importance of Coherence

Architecturally Visible Storage is similar to a (multi ported) scratch-pad memory.
The Importance of Coherence

Pitfall:
Multiple memories can cause potential memory coherence problems!
The Importance of Coherence

Solution:
Speculative DMA for Architecturally Visible Storage
T. Kluter, et al. 2008
The Importance of Coherence

Solution:
Speculative DMA for Architecturally Visible Storage
T. Kluter, et al. 2008

BUT:
Extensive hardware introduction by using a hardware coherence protocol!
Contents

• Motivation

• Way Stealing

• Benefits of Way Stealing

• Automation
Limiting the Memory Ports

We go back to the simplified block diagram of Speculative DMA with Architecturally Visible Storage (AVS)
Limiting the Memory Ports

Let us restrict the number of read and write ports to the Architecturally Visible Storage (AVS) to 1 each (more are impractical)
Merging the Memories

Recall:
Probable memory coherence problem between Architecturally Visible Storage and Data Cache (D$)
Idea:
Why not move Architecturally Visible Storage to Data Cache (D$)
Idea:
Why not move Architecturally Visible Storage to Data Cache (D$)
Way Stealing

We only look at the “read path”, the writing is similar
Due to “long lines” to the Application Specific Computational Unit (ASCU) pipeline registers are inserted to not influence the processor's critical path.
Pipelining to avoid Critical Paths

This can be done for all the “ways” of the data cache
Flexible Addressing Requirement

Too restrictive:
Normally all the ways of the data cache are addressed with the same “index”
Flexible Addressing Requirement

**Solution:**
We add an Address Generation Unit (AGU) to each “way” of the cache, similar to those used in Digital Signal Processors (DSPs).
Flexible Addressing Requirement

Solution:
We add an Address Generation Unit (AGU) to each “way” of the cache, similar to those used in Digital Signal Processors (DSPs)
Flexible Addressing Requirement

Only a single multiplexor is required in the processor's critical path. It's influence is minimal.
Area Reduction

Recall:
We started with Speculative DMA
Area Reduction

Now we have: Way Stealing
Contents

• Motivation

• Way Stealing

• Benefits of Way Stealing

• Automation
## Motivational Example

<table>
<thead>
<tr>
<th>Array holding 448 values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array holding 448 values</td>
</tr>
<tr>
<td>Array holding 448 values</td>
</tr>
<tr>
<td>Array holding 448 values</td>
</tr>
</tbody>
</table>

- Array holding 448 values
- Array holding 448 values
- Array holding 448 values
- Array holding 448 values

![Motivational Image](image-url)
Motivational Example

Array holding 448 values

Array holding 448 values

Array holding 448 values

Array holding 448 values

\[
Y = 0.299R + 0.587G + 0.114B
\]

\[
I = 0.500B - 0.169R - 0.331G
\]

\[
Q = 0.500R - 0.419G - 0.081B
\]
Motivational Example

Array holding 448 values

Array holding 448 values

Array holding 448 values
Motivational Example

Array holding 448 values
Array holding 448 values
Array holding 448 values
Experimental Setup

All experiments are implemented on a FPGA based emulation platform.
Experimental Setup

All experiments are implemented on a FPGA based emulation platform and in a 90nm standard-cell based ASIC.
Experimental Setup

All experiments are implemented on a FPGA based emulation platform and in a 90nm standard-cell based ASIC

D. Tarjan, et al. 2006

Energy of the memory subsystem only!

Speculative DMA engine

coherence protocol
Performance and Energy

![Graphs showing speed up and relative energy consumption](image)

- [Diagram of a computer pipeline](diagram)

  - Fetch
  - Decode
  - Exec
  - Mem
  - WB

  - I$ to RF
  - RF to Exec
  - Exec to D$
  - D$ to RF

  - Main memory

---

41
Standard Instruction Set Extensions (ISE) not only improves performance, it also reduced dynamic energy consumption.
Speculative DMA increases performance, but consumes significantly more energy due to its communication overhead.
Speculative DMA increases performance, but consumes significantly more energy due to its communication overhead.
Communication Overhead

Speculative DMA increases performance, but consumes significantly more energy due to its communication overhead.
Speculative DMA increases performance, but consumes significantly more energy due to its communication overhead.
Way Stealing not only removes significantly hardware real-estate, it is not dependent on communication, and outperforms previous art.
Memory Wall influence

Memories cannot keep up with technology advances:
The memory wall problem
Memory Wall influence

The standard instruction set extensions are sensitive to the difference in processor to memory speed, similar as the original code.
Memory Wall influence

The required communication makes speculative DMA very sensitive to the difference in processor to memory speed.
Way Stealing removes the communication overheads; therefore, it acts “normal” to the difference in processor to memory speed.
More results

![Graph showing speedup and relative energy consumption]

- **Speedup**
  - Original Code
  - ISE enhanced Code
  - Way Stealing enhanced Code

- **Relative energy consumption**
Contents

• Motivation

• Way Stealing

• Benefits of Way Stealing

• Automation
Automation

Recall:

Automated Instruction Set Extensions

Cost function of software execution

Cost function of hardware execution

Adds Hardware in form of an Application Specific Computational Unit (ASCU)

Adds ASCU specific Custom Instruction(s)

Cost function of software execution

Cost function of hardware execution
Introduction of the communication cost $M(C)$ leads to Architecturally Visible Storage (AVS)

P. Biswas, et al. 2006
Way Stealing only requires simple modifications to the algorithm presented by Biswas et al., namely:
Way Stealing only requires simple modifications to the algorithm presented by Biswas et al., namely:

- Modification of the communication cost function $M(C)$
Way Stealing only requires simple modifications to the algorithm presented by Biswas et al., namely:

- Modification of the communication cost function $M(C)$

- Restriction to a single read and write port on the data-structure memories
Way Stealing only requires simple modifications to the algorithm presented by Biswas et al., namely:

- Modification of the communication cost function $M(C)$
- Restriction to a single read and write port on the data-structure memories
- Only a single data structure each “stolen” way
Programmers View

Array holding 448 values

Array holding 448 values

Array holding 448 values
void convert() {
  int i, R, G, B, Y, I, Q;
  for (i = 0; i < MAX; i++) {
    R=P1[i]; B=P2[i]; G=P3[i];
    Y=0.299*R+0.587*G+0.114*B;
    I=0.500*B-0.169*R-0.331*G;
    Q=0.500*R-0.419*G-0.081*B;
    P1[i]=Y; P2[i]=I; P3[i]=Q;
  }
}
void convert() {
    int i, R, G, B, Y, I, Q;
    for (i = 0; i < MAX; i++) {
        R = P1[i]; B = P2[i]; G = P3[i];
        Y = 0.299*R + 0.587*G + 0.114*B;
        I = 0.500*B - 0.169*R - 0.331*G;
        Q = 0.500*R - 0.419*G - 0.081*B;
        P1[i] = Y; P2[i] = I; P3[i] = Q;
    }
}

The algorithm detects the Way Stealing Custom Instruction (WS_CI) and adds the ASCU.
```c
void convert() {
    int i;

    for (i = 0 ; i < MAX ; i++){
        WS_CI(P1,P2,P3);
    }
}
```
For correct execution of the custom instruction we have to guarantee that the data is contained in the data cache.
void convert() {
    int i;
    prefetch(P1,P2,P3);

    for (i = 0 ; i < MAX ; i++){
        WS_CI(P1,P2,P3);
    }
}

We prefetch the data prior to the execution of the custom instruction
We prefetch the data prior to the execution of the custom instruction, and lock it. After the execution of the custom instruction we unlock the data.
void convert() {
    int i;
    prefetch(P1,P2,P3);
    lock(P1,P2,P3);
    AGU_setup(1,1,MAX);
    for (i = 0 ; i < MAX ; i++){
        WS_CI(P1,P2,P3);
    }
    unlock(P1,P2,P3);
}
Conclusion

• Way Stealing significantly removes silicon real-estate

• Way Stealing is not dependent on the Memory to Processor Distance

• Way Stealing significantly improves in both performance and energy reduction previous art

• Way Stealing is easily usable in an automated Instruction Set Extension algorithm by simple modifications to the state-of-the-art