CAeSaR: unified Cluster-Assignment Scheduling and communication Reuse for clustered VLIW processors

Vasileios Porpodas † and Marcelo Cintra †*

School of Informatics, University of Edinburgh†
Intel Labs Braunschweig*

CASES 2013
Energy efficiency, VLIWs and Scheduling

- Mobile era.
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.
- Hardware Instruction scheduling consumes a lot of energy.
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.
- Hardware Instruction scheduling consumes a lot of energy.
- VLIW processors: high-performance and statically scheduled.
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.
- Hardware Instruction scheduling consumes a lot of energy.
- VLIW processors: high-performance and statically scheduled.
- Clustered VLIW = scalable VLIW.
Clustered VLIW

• Scalable
• Energy efficient
Clustered VLIW

- Scalable
- Energy efficient
- High frequency
- Inter-Cluster delay
Clustered VLIW

- Scalable
- Energy efficient
- High frequency
- Inter-Cluster delay
- Relies on compiler
- Statically scheduled
- Explicit ILP
Scheduling Overview

- Scheduler for monolithic VLIW

Non-Clustered

... → Instruction scheduling → ...

a. Compilation passes for non-clustered architectures.

Clustered Architecture
Scheduling Overview

- Scheduler for monolithic VLIW
- 2-stage scheduling for clusters

Non-Clustered

... → Instruction scheduling → ...

a. Compilation passes for non-clustered architectures.

Clustered Architecture

... → Cluster assignment → Instruction scheduling → ...

b. Decoupled Cluster Assignment and Scheduling.
Scheduling Overview

- Scheduler for monolithic VLIW
- 2-stage scheduling for clusters

Non-Clustered

```
  ... → Instruction scheduling → ...
```

a. Compilation passes for non-clustered architectures.

Clustered Architecture

```
  ... → Cluster assignment → Instruction scheduling → ...
```

b. Decoupled Cluster Assignment and Scheduling.
Scheduling Overview

- Scheduler for monolithic VLIW
- 2-stage scheduling for clusters
- Unified scheduler (State-of-the-art)

Non-Clustered

- a. Compilation passes for non-clustered architectures.

Clustered Architecture

- b. Decoupled Cluster Assignment and Scheduling.

- c. UAS: Unified cluster Assignment and Scheduling.
Scheduling Overview

- Scheduler for monolithic VLIW
- 2-stage scheduling for clusters
- Unified scheduler (State-of-the-art)
- 2-stage ICC-reuse
Scheduling Overview

- Scheduler for monolithic VLIW
- 2-stage scheduling for clusters
- Unified scheduler (State-of-the-art)
- 2-stage ICC-reuse

Non-Clustered

.. → Instruction scheduling → ...

a. Compilation passes for non-clustered architectures.

Clustered Architecture

.. → Cluster assignment → Instruction scheduling → ...

b. Decoupled Cluster Assignment and Scheduling.

c. UAS: Unified cluster Assignment and Scheduling.

d. Decoupled UAS and ICC-Reuse.
Scheduling Overview

- Scheduler for monolithic VLIW
- 2-stage scheduling for clusters
- Unified scheduler (State-of-the-art)
- 2-stage ICC-reuse
- Unified ICC-reuse
Outline

Introduction

CAeSaR Scheduling

Experimental Setup and Results

Conclusion
CAeSaR Scheduler

- State-of-the-art (UAS)
CAeSar Scheduler

- State-of-the-art (UAS)
- High Priority INSTR

High Sorted Ready List

Clustering Heuristic

Low
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR

Diagram:
- Try Next
- High
- Low
- Sorted Ready List
- Clustering Heuristic
- INSTR
- Can Issue INSTR + ICCs?
- NO
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
- CAeSaR

![Diagram of CAeSaR Scheduler]

1. High Priority INSTR
2. Can reuse ICCs?
3. Clustering Heuristic
4. Sorted Ready List
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
- CAeSaR
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
- CAeSaR
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
- CAeSaR
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
- CAeSaR
- ICC-reuse

![Flowchart Diagram]

Try Next

High → Low

Sorted Ready List

INSTR

INSTR+ICCs?

Can Issue

YES →

CAN Issue INSTR?

NO →

Can Issue ICCs?

YES

CAN Issue INSTR and ICCs?

NO

CAN Issue ICCs and INSTR

NO →

E B D H

Clustering Heuristic

CAeSaR
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
- CAeSaR
- ICC-reuse
CAeSaR Scheduler

- State-of-the-art (UAS)
- High Priority INSTR
- If can issue, DONE
- CAeSaR
- ICC-reuse
Example

Monolithic

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>B</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>J</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. VLIW

Clustered Architecture

f. Data Flow Graph (DFG)
Example

- 2-step scheduler generates bad schedule

<table>
<thead>
<tr>
<th>Monolithic</th>
<th>Clustered Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 A</td>
<td>0 A CL0 CL1</td>
</tr>
<tr>
<td>1 B F</td>
<td>1 B ICC</td>
</tr>
<tr>
<td>2 C G</td>
<td>2 C F</td>
</tr>
<tr>
<td>3 D H</td>
<td>3 D ICC</td>
</tr>
<tr>
<td>4 I J</td>
<td>4 H G</td>
</tr>
<tr>
<td>5 E</td>
<td>5 ICC</td>
</tr>
<tr>
<td>a. VLIW</td>
<td></td>
</tr>
<tr>
<td>0 A</td>
<td>0 A CL0 CL1</td>
</tr>
<tr>
<td>1 B ICC</td>
<td>1 B ICC</td>
</tr>
<tr>
<td>2 C F</td>
<td>2 C F</td>
</tr>
<tr>
<td>3 D ICC</td>
<td>3 D ICC</td>
</tr>
<tr>
<td>4 H G</td>
<td>4 H G</td>
</tr>
<tr>
<td>5 ICC</td>
<td>5 ICC</td>
</tr>
<tr>
<td>6 I</td>
<td>6 I ICC</td>
</tr>
<tr>
<td>7 ICC</td>
<td>7 ICC</td>
</tr>
<tr>
<td>8 J</td>
<td>8 J</td>
</tr>
<tr>
<td>9 E</td>
<td>9 E</td>
</tr>
<tr>
<td>b. Decoupled</td>
<td></td>
</tr>
</tbody>
</table>

f. Data Flow Graph (DFG)
Example

- 2-step scheduler generates bad schedule
- UAS improves it by being ICC-aware
Example

- 2-step scheduler generates bad schedule
- UAS improves it by being ICC-aware
- UAS with ICC-reuse has fewer ICCs

<table>
<thead>
<tr>
<th>Monolithic</th>
<th>Clustered Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. VLIW</td>
<td>Cl0</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>ICC</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
</tr>
<tr>
<td>7</td>
<td>ICC</td>
</tr>
<tr>
<td>8</td>
<td>J</td>
</tr>
<tr>
<td>9</td>
<td>E</td>
</tr>
</tbody>
</table>

b. Decoupled
c. UAS
d. UAS+ICC-reuse

f. Data Flow Graph (DFG)
Example

- 2-step scheduler generates bad schedule
- UAS improves it by being ICC-aware
- UAS with ICC-reuse has fewer ICCs
- CAeSaR solves phase-ordering

<table>
<thead>
<tr>
<th>Monolithic</th>
<th>Clustered Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CL0</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>ICC</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
</tr>
<tr>
<td>7</td>
<td>ICC</td>
</tr>
<tr>
<td>8</td>
<td>J</td>
</tr>
<tr>
<td>9</td>
<td>E</td>
</tr>
</tbody>
</table>

- a. VLIW
- b. Decoupled

f. Data Flow Graph (DFG)
Register File Coherence

- Reused value must be correct

a. Before ICCs
Register File Coherence

- Reused value must be correct

a. Before ICCs

b. Wrong: No Coherence
Register File Coherence

- Reused value must be correct

a. Before ICCs

b. Wrong: No Coherence
Register File Coherence

- Reused value must be correct
- CAeSaR follows an approach similar to write-invalidate cache coherent protocols

a. Before ICCs

b. Wrong: No Coherence

c. CAeSaR Register Coherence
Register File Coherence

- Reused value must be correct
- CAeSaR follows an approach similar to write-invalidate cache coherent protocols
Register File Coherence

- Reused value must be correct
- CAeSaR follows an approach similar to write-invalidate cache coherent protocols

![Diagram](slide8.png)

a. Before ICCs

b. Wrong: No Coherence

c. CAeSaR Register Coherence
Experimental Setup

- Compiler: GCC-4.5.0, Modified Haifa-Scheduler
Experimental Setup

- Compiler: GCC-4.5.0, Modified Haifa-Scheduler

- Architecture
  - IA64-based 4 issue clustered VLIW
  - 2,4 clusters
  - 1 cycle Inter-Cluster Latency
Experimental Setup

- Compiler: GCC-4.5.0, Modified Haifa-Scheduler

- Architecture
  - IA64-based 4 issue clustered VLIW
  - 2,4 clusters
  - 1 cycle Inter-Cluster Latency

- Benchmarks: SPEC CINT2000 and MediabenchII Video
Results: 4-cluster 4-issue, ICC ratio

- ICCs are a significant bottleneck
Results: 4-cluster 4-issue, ICC ratio

- ICCs are a significant bottleneck
- 1 ICC in 5 non-ICCs
Results: 4-cluster 4-issue, Sched cycles

- CAeSaR generates 13.8% more compact schedules on average
**Results: 4-cluster 4-issue, Sched cycles**

- CAeSaR generates 13.8% more compact schedules on average
- ICC-reuse same performance as UAS (expected)
Results: 4-cluster 4-issue, ICCs

- CAeSaR reduces the count of ICCs
Results: 4-cluster 4-issue, ICCs

- CAeSaR reduces the count of ICCs
- CAeSaR generates fewer ICCs than ICC-reuse (proof of phase-ordering problem)
Results: 4-cluster 4-issue, Instr Distribution

- Better resource utilization
- Less communication bottlenecks
- Potential for more ILP
Results: 2-cluster 4-issue, ICC ratio

- ICCs are still noticeable
Results: 2-cluster 4-issue, ICC ratio

- ICCs are still noticeable
- 1 ICC in 10 non-ICCs
Results: 2-cluster 4-issue, Sched cycles

- CAeSaR generates 8.4% more compact schedules on average
Results: 2-cluster 4-issue, Sched cycles

- CAeSaR generates 8.4% more compact schedules on average
- ICC-reuse same performance as UAS (expected)
Results: 2-cluster 4-issue, ICCs

- CAeSaR reduces the count of ICCs
Results: 2-cluster 4-issue, ICCs

- CAeSaR reduces the count of ICCs
- CAeSaR generates fewer ICCs than ICC-reuse (proof of phase-ordering problem)
Conclusion

Proposed CAeSaR Scheduling, a scheduler for clustered VLIWs that:

- Eliminates redundant Inter-Cluster copies
- Solves phase-ordering problem between Scheduling and ICC-reuse
- Generates more compact schedules compared to state-of-the-art
CAeSaR: unified Cluster-Assignment Scheduling and communication Reuse for clustered VLIW processors

Vasileios Porpodas † and Marcelo Cintra †*  

School of Informatics, University of Edinburgh† 
Intel Labs Braunschweig*

CASES 2013