Out-Of-Order Execution of Synchronous Data-Flow Networks

Daniel Baudisch, Jens Brandt, Klaus Schneider

Embedded Systems Group
Department of Computer Science
University of Kaiserslautern, Germany

Last Update: May 22, 2013
Outline

1 Introduction

2 Out-Of-Order Execution

3 Results

4 Conclusion
1 Introduction

2 Out-Of-Order Execution

3 Results

4 Conclusion
here: from DPN (given as HSDF) to OOO-Execution in SMP (OOOSMP)
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes > # cores)
  - better balancing (#num nodes > # cores)
  - task ≈ execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes
Dynamic scheduling (automatic balancing at runtime)
⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)
  • decrease overhead of thread-switches (num nodes > # cores)
  • better balancing (num nodes > # cores)
  • task ≈ execution of node for an iteration (like in SmpSS)
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)
  - decrease overhead of thread-switches (#num nodes > # cores)
  - better balancing (#num nodes > # cores)
  - task \(\approx\) execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes
Dynamic scheduling (automatic balancing at runtime)  
⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)  
- decrease overhead of thread-switches (\#num nodes > \# cores)  
- better balancing (\#num nodes > \# cores)  
- task ≈ execution of node for an iteration (like in SmpSS)
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)
    - decrease overhead of thread-switches (#num nodes > # cores)
    - better balancing (#num nodes > # cores)
    - task ≈ execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes

![Diagram showing unbalanced nodes and multiple instances of nodes]
Goal

- Dynamic scheduling (automatic balancing at runtime)
  \(\Rightarrow\) Task based execution (OpenMP, Intel TBB, StarSS, \ldots)
  - decrease overhead of thread-switches (\(\#\)num nodes > \(\#\) cores)
  - better balancing (\(\#\)num nodes > \(\#\) cores)
  - task \(\approx\) execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes

```
    A  1
      \_____/\  \\
      /     \\
   X4 -- X3 -- X2  1  \  \\
        \    \\
           \  \\
     1  \ \\
        \ \\
    B  1
      \_____/\  \\
      /     \\
   V1 -- V0
```
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)
  - decrease overhead of thread-switches (#num nodes > # cores)
  - better balancing (#num nodes > # cores)
  - task ≈ execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)
    - decrease overhead of thread-switches (#num nodes > # cores)
    - better balancing (#num nodes > # cores)
    - task ≈ execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes

Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, ...)
    - decrease overhead of thread-switches (#num nodes > # cores)
    - better balancing (#num nodes > # cores)
    - task ≈ execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes

Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes > # cores)
  - better balancing (#num nodes > # cores)
  - task ≈ execution of node for an iteration (like in SmpSS)

Unbalanced Nodes / Multiple Instances of Nodes

Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  - decrease overhead of thread-switches (#num nodes > # cores)
  - better balancing (#num nodes > # cores)
  - task ≈ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  ⇒ increase parallelism (#num nodes < # cores)
  ⇒ counter unbalanced/varying computational effort

Unbalanced Nodes / Multiple Instances of Nodes
Dynamic scheduling (automatic balancing at runtime)
⇒ Task based execution (OpenMP, Intel TBB, StarSS, ...)
  • decrease overhead of thread-switches (#num nodes > # cores)
  • better balancing (#num nodes > # cores)
  • task ≈ execution of node for an iteration (like in SmpSS)

Execute iterations of single nodes in parallel
⇒ increase parallelism (#num nodes < # cores)
⇒ counter unbalanced/varying computational effort
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)
    - decrease overhead of thread-switches (#num nodes > # cores)
    - better balancing (#num nodes > # cores)
    - task ≈ execution of node for an iteration (like in SmpSS)

- Execute iterations of single nodes in parallel
  ⇒ increase parallelism (#num nodes < # cores)
  ⇒ counter unbalanced/varying computational effort
**Goal**

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, …)
    - decrease overhead of thread-switches (#num nodes > # cores)
    - better balancing (#num nodes > # cores)
    - task ≈ execution of node for an iteration (like in SmpSS)

- Execute iterations of single nodes in parallel
  ⇒ increase parallelism (#num nodes < # cores)
  ⇒ counter unbalanced/varying computational effort

---

Unbalanced Nodes / Multiple Instances of Nodes

![Diagram of Unbalanced Nodes]

Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, ...)
    - decrease overhead of thread-switches (#num nodes > # cores)
    - better balancing (#num nodes > # cores)
    - task ≈ execution of node for an iteration (like in SmpSS)
- Execute iterations of single nodes in parallel
  ⇒ increase parallelism (#num nodes < # cores)
  ⇒ counter unbalanced/varying computational effort

Unbalanced Nodes / Multiple Instances of Nodes
Goal

- Dynamic scheduling (automatic balancing at runtime)
  ⇒ Task based execution (OpenMP, Intel TBB, StarSS, . . .)
    - decrease overhead of thread-switches (#num nodes > # cores)
    - better balancing (#num nodes > # cores)
    - task ≈ execution of node for an iteration (like in SmpSS)

- Execute iterations of single nodes in parallel
  ⇒ increase parallelism (#num nodes < # cores)
  ⇒ counter unbalanced/varying computational effort
    - in-order reads of buffers does not guarantee in-order writes

Unbalanced Nodes / Multiple Instances of Nodes

Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider) 5
Outline

1 Introduction

2 Out-Of-Order Execution

3 Results

4 Conclusion
Out-Of-Order in Hardware

- Tomasulo’s Algorithm
- Execute sequence of instructions in data-flow order
- Reservation Station
  - ring buffer
  - each “line” \(\approx\) one instruction to execute
  \(\Rightarrow\) window of instructions to execute
Central Buffer Station (CBS)
- ring buffer
- each “line” \(\approx\) one execution of HSDF
- each “line” requires each task to be scheduled once
  - here: several schedules / line
  - reservation station: one schedule / line
- contains counter to keep track of remaining dependencies between tasks

Worker threads + task queue
- task based execution
- execute tasks, afterwards release dependencies
**Example**

---

**SDFG**

```
1  1
A  B  C  
```

---

**CBS / Task Queue(TQ) / Worker(WT)**

<table>
<thead>
<tr>
<th>it</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>hD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>√</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ TQ = \{(A, 0); (A, 1); (A, 2); (A, 3)\} \]

---

**What’s happening ?**

- Initialization
What’s happening?

- Workers get tasks from TQ and start processing
Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider) 9

What’s happening?
- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies
Example

**What’s happening?**

- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies
### Example

**SDFG**

![SDFG Diagram]

**CBS / Task Queue (TQ) / Worker (WT)**

<table>
<thead>
<tr>
<th>it</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>hD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ TQ = \{(B, 0); (C, 0)\} \]

<table>
<thead>
<tr>
<th>(WT_1)</th>
<th>(WT_2)</th>
<th>(WT_3)</th>
<th>(WT_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A, 0)</td>
<td>(A, 1)</td>
<td>(A, 2)</td>
<td>(A, 3)</td>
</tr>
</tbody>
</table>

**What’s happening?**

- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies
What’s happening?

- Worker 1 executed A for iteration 0
- Worker 1 has to remove corresponding dependencies
OUT-OF-ORDER EXECUTION OF SYNCHRONOUS DATA-FLOW NETWORKS

Worker 1 is now free to execute the next task.
Introduction
Out-Of-Order Execution
Results
Conclusion

Example

SDFG

CBS / Task Queue (TQ) / Worker (WT)

<table>
<thead>
<tr>
<th>it</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>h_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

TQ = \{(C, 0)\}

WT_1 | WT_2 | WT_3 | WT_4
(B, 0) | (A, 1) | (A, 2) | (A, 3)

What’s happening?

- Worker 1 gets next task from TQ and starts processing
What’s happening?

- Worker 3 executed A for iteration 2
- Worker 3 has to remove corresponding dependencies
What’s happening?

- Worker 3 executed A for iteration 2
- Worker 3 has to remove corresponding dependencies
Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Example

What’s happening?

- Worker 4 executed A for iteration 3
- Worker 4 has to remove corresponding dependencies
### Example

#### SDFG

- Node `A` emits `1` to node `B`.
- Node `B` emits `1` to node `C`.
- Node `C` emits `1` to node `A`.

#### CBS / Task Queue (TQ) / Worker (WT)

<table>
<thead>
<tr>
<th>it</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>h_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- TQ = `{(C, 0); (B, 2)}`

<table>
<thead>
<tr>
<th>WT_1</th>
<th>WT_2</th>
<th>WT_3</th>
<th>WT_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B, 0)</td>
<td>(A, 1)</td>
<td></td>
<td>(A, 3)</td>
</tr>
</tbody>
</table>

#### What's happening?

- Worker 4 executed A for iteration 3
- Worker 4 has to remove corresponding dependencies
**Example**

**SDFG**

**CBS / Task Queue (TQ) / Worker (WT)**

<table>
<thead>
<tr>
<th>it</th>
<th>$AB$</th>
<th>$AC$</th>
<th>$BC$</th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$h_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>√</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>√</td>
<td>√</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

$TQ = \{(C, 0); (B, 2)\}$

<table>
<thead>
<tr>
<th>$WT_1$</th>
<th>$WT_2$</th>
<th>$WT_3$</th>
<th>$WT_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(B, 0)$</td>
<td>$(A, 1)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What’s happening?

- **Worker 4 is now free to execute the next task**
Worker 3, 4 get tasks from TQ and start processing
What’s happening?

- Worker 4 executed B for iteration 2
- Worker 4 has to remove corresponding dependencies
Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)

**What’s happening?**

- **Worker 4 executed B for iteration 2**
- **Worker 4 has to remove corresponding dependencies**

---

### Example

**SDFG**

- Node A
- Node B
- Node C
- Edges indicate dependencies

**CBS / Task Queue (TQ) / Worker (WT)**

<table>
<thead>
<tr>
<th>it</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>h_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>√</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

- **TQ = \{(C, 3)\}**

- **WT_1** = (B, 0)
- **WT_2** = (A, 1)
- **WT_3** = (C, 0)
- **WT_4** = (B, 2)
Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Example

What's happening?

- Worker 3 executed C for iteration 0
- Worker 3 has to remove corresponding dependencies
Example

SDFG

CBS / Task Queue(TQ) / Worker(WT)

<table>
<thead>
<tr>
<th>it</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>h_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>√</td>
<td>√</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ TQ = \{(C, 3)\} \]

\[
\begin{array}{cccc}
WT_1 & WT_2 & WT_3 & WT_4 \\
(B, 0) & (A, 1) & & \\
\end{array}
\]

What’s happening?

- Worker 3 is now free to execute the next task
Example

What’s happening?

- Worker 1 executed B for iteration 0
- Worker 1 has to remove corresponding dependencies
**Example**

**SDFG**

**CBS / Task Queue(TQ) / Worker(WT)**

<table>
<thead>
<tr>
<th>it</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>hD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ TQ = \{(C, 3)\} \]

<table>
<thead>
<tr>
<th>WT₁</th>
<th>WT₂</th>
<th>WT₃</th>
<th>WT₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B, 0)</td>
<td>(A, 1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**What’s happening?**

- Worker 1 executed B for iteration 0
- Worker 1 has to remove corresponding dependencies
What’s happening?

- \( h_D = 0 \Rightarrow \) head can be removed
- Remove dependencies due to initial tokens
What’s happening?

- $h_D = 0 \Rightarrow$ head can be removed
- Remove dependencies due to initial tokens
Introduction
Out-Of-Order Execution
Results
Conclusion

Example

**SDFG**

**CBS / Task Queue** (TQ) / **Worker** (WT)

<table>
<thead>
<tr>
<th>it</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>h_D</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>√</td>
<td>√</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

*TQ = {(C, 3), (A, 4), (B, 3)}*

<table>
<thead>
<tr>
<th>WT_1</th>
<th>WT_2</th>
<th>WT_3</th>
<th>WT_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**What’s happening?**

- \( h_D = 0 \) ⇒ head can be removed
- Remove dependencies due to initial tokens

Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Introduction

Out-Of-Order Execution

Results

Conclusion
Benchmarks - Results

Benchmark systems: 2x Xeon X5450 (≈ 8x 3.00GHz)
i5-750 (4x 2.66GHz)

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Speedup on i5-750</th>
<th></th>
<th>Speedup on 2xXeon</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1n1t⇒io</td>
<td>io⇒ooo</td>
<td>1n1t⇒io</td>
<td>io⇒ooo</td>
</tr>
<tr>
<td>MatrixMult (16x16)</td>
<td>14.40</td>
<td>1.33</td>
<td>23.22</td>
<td>1.06</td>
</tr>
<tr>
<td>MatrixMult (32x32)</td>
<td>11.48</td>
<td>1.12</td>
<td>28.95</td>
<td>1.05</td>
</tr>
<tr>
<td>MatrixMult (48x48)</td>
<td>10.85</td>
<td>1.10</td>
<td>25.83</td>
<td>1.05</td>
</tr>
<tr>
<td>LU Decomp. (4x4)</td>
<td>0.75</td>
<td>1.95</td>
<td>0.67</td>
<td>1.37</td>
</tr>
<tr>
<td>LU Decomp. (8x8)</td>
<td>3.97</td>
<td>1.23</td>
<td>2.54</td>
<td>1.05</td>
</tr>
<tr>
<td>LU Decomp. (16x16)</td>
<td>4.39</td>
<td>1.07</td>
<td>7.17</td>
<td>1.04</td>
</tr>
<tr>
<td>LU Decomp. (32x32)</td>
<td>5.26</td>
<td>1.04</td>
<td>5.46</td>
<td>1.05</td>
</tr>
<tr>
<td>DFT (4 tasks)</td>
<td>6.20</td>
<td>1.00</td>
<td>7.10</td>
<td>0.97</td>
</tr>
<tr>
<td>DFT (8 tasks)</td>
<td>6.64</td>
<td>0.99</td>
<td>9.22</td>
<td>1.01</td>
</tr>
<tr>
<td>DFT (16 tasks)</td>
<td>4.87</td>
<td>1.01</td>
<td>5.56</td>
<td>0.99</td>
</tr>
<tr>
<td>DFT (32 tasks)</td>
<td>4.80</td>
<td>1.02</td>
<td>5.57</td>
<td>0.99</td>
</tr>
<tr>
<td>Landscape Gen. (H=20)</td>
<td>1.84</td>
<td>1.11</td>
<td>2.56</td>
<td>1.12</td>
</tr>
<tr>
<td>Landscape Gen. (H=200)</td>
<td>2.09</td>
<td>1.34</td>
<td>6.75</td>
<td>1.14</td>
</tr>
<tr>
<td>Landscape Gen. (H=400)</td>
<td>3.20</td>
<td>1.43</td>
<td>11.15</td>
<td>1.12</td>
</tr>
<tr>
<td>Landscape Gen. (H=800)</td>
<td>4.84</td>
<td>1.80</td>
<td>17.28</td>
<td>1.28</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Out-Of-Order Execution
3. Results
4. Conclusion
Conclusion

- Dynamic task based execution $\Rightarrow$ better balancing
- Out-Of-Order Execution to improve TLP
  “Rolling-out” SDF $\Rightarrow$ more tasks / parallelism
- Similar to programs running on OOO-HW: more independence, e.g. #nodes / #cores do not have to match

Future Work

- Large arrays / data structures
  - memory-requirement
  - may introduce copy-actions
Thank you for your attention!

Questions? Suggestions? Ideas?
The End

Backup Slides
Motivation

- model-based development of applications
  in particular: embedded systems
- synchronous languages, e.g. Esterel, Lustre, Quartz
  - can be used for embedded systems
  - hide communication latencies
  - execution of instructions in perfect synchrony
- synthesis more difficult
  (especially for heterogeneous/distributed systems)
Guarded Actions (GA)

**System (Example)**

**Interface:**
- Inputs: $i, d, c$
- Output: $o$
- Locals: $a[N], j0, j, p$

**Guarded Actions:**
- $p \Rightarrow \text{next}(p) = \text{True}$
- $c \land p \Rightarrow a[j] = i$
- $p \Rightarrow \text{next}(j) = (j + 1) \mod N$
- $p \Rightarrow j0 = (j - d) \mod N$
- $p \Rightarrow o = a[j0]$
Guarded Actions (GA)

System (Example)

Interface:
Inputs: \( i, d, c \)
Output: \( o \)
Locals: \( a[N], j0, j, p \)

Guarded Actions:
\( p \Rightarrow \text{next}(p) = \text{True} \)
\( c \land p \Rightarrow a[j] = i \)
\( p \Rightarrow \text{next}(j) = (j + 1) \mod N \)
\( p \Rightarrow j0 = (j - d) \mod N \)
\( p \Rightarrow o = a[j0] \)

Generic execution

while(True)
    read system inputs
    execute GA in data-flow order
    update state of system
    generate outputs of system
    write system outputs
endwhile
Guarded Actions (GA)

**Generic execution**

```
while(True)
    read system inputs
    execute GA in data-flow order
    update state of system
    generate outputs of system
    write system outputs
endwhile
```

**Dependency Graph**

- `ReadFromEnv(i, c, d)`
- `c ∧ p ⇒ a[j] = i`
- `p ⇒ next(j) = (j + 1) % N`
- `p ⇒ next(p) = True`
- `p ⇒ j0 = (j − d) % N`
- `p ⇒ o = a[j0]`
Guarded Actions (GA)

**Dependency Graph**

- $c \land p \Rightarrow a[j] = i$
- $p \Rightarrow \text{next}(j) = (j + 1)\% N$
- $p \Rightarrow \text{next}(p) = \text{True}$
- $p \Rightarrow j^0 = (j - d)\% N$
- $p \Rightarrow o = a[j^0]$
- WriteToEnv(o)

**Generic execution**

```python
while(True):
    read system inputs
    execute GA in data-flow order
    update state of system
    generate outputs of system
    write system outputs
endwhile
```
Guarded Actions (GA)

Dependency Graph

- \( \text{ReadFromEnv}(i, c, d) \)
- \( c \land p \Rightarrow a[j] = i \)
- \( p \Rightarrow \text{next}(j) = (j + 1)\%N \)
- \( p \Rightarrow \text{next}(p) = \text{True} \)
- \( p \Rightarrow j^0 = (j - d)\%N \)
- \( p \Rightarrow o = a[j^0] \)
- WriteToEnv(o)

Generic execution

\[
\text{while(True)} \\
\text{read system inputs} \\
\text{execute GA in data-flow order} \\
\text{update state of system} \\
\text{generate outputs of system} \\
\text{write system outputs} \\
\text{endwhile}
\]
Guarded Actions (GA)

Dependency Graph

ReadFromEnv(i, c, d)

\[ c \land p \Rightarrow a[j] = i \]

\[ p \Rightarrow \text{next}(j) = (j + 1)\%N \]

\[ p \Rightarrow \text{next}(p) = \text{True} \]

\[ p \Rightarrow j0 = (j - d)\%N \]

\[ p \Rightarrow o = a[j0] \]

WriteToEnv(o)

Generic execution

while(True)
    read system inputs
    execute GA in data-flow order
    update state of system
    generate outputs of system
    write system outputs
endwhile
Guarded Actions (GA)

Dependency Graph

ReadFromEnv(i, c, d)

\[ c \land p \Rightarrow a[j] = i \]

\[ p \Rightarrow \text{next}(j) = (j + 1) \% N \]

\[ p \Rightarrow \text{next}(p) = \text{True} \]

\[ p \Rightarrow j_0 = (j - d) \% N \]

\[ p \Rightarrow o = a[j_0] \]

WriteToEnv(o)

Generic execution

while(True)
  read system inputs
  execute GA in data-flow order
  update state of system
  generate outputs of system
  write system outputs
endwhile

Out-Of-Order Execution of Synchronous Data-Flow Networks (Daniel Baudisch, Jens Brandt, Klaus Schneider)
Guarded Actions (GA)

DPN

\[ c \land p \Rightarrow a[j] = i \]
\[ p \Rightarrow \text{next}(j) = (j + 1) \mod N \]
\[ p \Rightarrow \text{next}(p) = \text{True} \]
\[ p \Rightarrow o = a[j0] \]
\[ p \Rightarrow o = a[j0] \]
\[ \text{WriteToEnv}(o) \]

Notes

- DPN is gained by replacing dependencies by FIFO-buffers \( \Rightarrow \) “Theory of Latency Insensitive Design” (McMillan et. al)
- writer must be uniquely determined: group nodes writing to the same variable
- grouping of nodes / partitioning increase computation effort per node

Notes

- DPN is gained by replacing dependencies by FIFO-buffers \( \Rightarrow \) “Theory of Latency Insensitive Design” (McMillan et. al)
- writer must be uniquely determined: group nodes writing to the same variable
- grouping of nodes / partitioning increase computation effort per node
Guarded Actions (GA)

Synchronous DPN

- \( c \land p \Rightarrow a[j] = i \)
- \( p \Rightarrow \text{next}(j) = (j + 1) \% N \)
- \( p \Rightarrow \text{next}(p) = \text{True} \)
- \( p \Rightarrow j_0 = (j - d) \% N \)
- \( p \Rightarrow o = a[j_0] \)
- \( \text{ReadFromEnv}(i, c, d) \)
- \( \text{WriteToEnv}(o) \)

Synchronous DPN

- \#num tokens per read/write fixed
- special case in SDPN from sync. lang.: 1 token per read/write per buffer
Guarded Actions (GA)

**Synthesis (PThreads)**
- create thread for each node
- each thread:
  - while(True)
  - read input buffers
  - execute GA of node
  - write output buffers
- endwhile

**Synchronous DPN**

- \( c \land p \Rightarrow a[j] = i \)
- \( p \Rightarrow \text{next}(j) = (j + 1) \% N \)
- \( p \Rightarrow \text{next}(p) = \text{True} \)
- \( p \Rightarrow j0 = (j - d) \% N \)
- \( p \Rightarrow o = a[j0] \)
- \( \text{WriteToEnv}(o) \)