A Fast Heuristic Scheduling Algorithm for Periodic Concurrent Models

Weiwei Chen and Rainer Doemer
Center for Embedded Computer Systems
University of California, Irvine
Outline

• Introduction and Related Work
• ConcurrenC Model Simulation
• Periodic ConcurrenC Scheduling
  – Scheduling of SDF-like Models
  – Scheduling of Periodic ConcurrenC Models
• Experiment Results
• Conclusion and Future Work
Introduction

• Embedded systems are modeled and described at different levels of abstraction.
  – **System-level Description Languages** (SLDLs), like SystemC and SpecC are used for modeling and verification, and are simulated by **discrete event** (DE) simulation mechanism.
  – **C-based SLDLs** are well-defined, but **Modeling** is not.
    • “How to write a good model?”
    • Modeling guidelines are needed.
  – Simulation speed is not fast for **DE-based** single-thread kernels.
ConcurrenC MoC

- **ConcurrenC**, a new model of computation (MoC) was proposed to emphasize the importance of the modeling.
  - A specific subset, called *periodic ConcurrenC*, can be statically scheduled to improve the simulation speed.

Relationship between ConcurrenC and C-based SLDLs
Related Work

• System-level Description Languages
  – SystemC [T.Grotker 2002]
  – SpecC [D.D.Gajski 2000]
  – Discrete Event (DE) simulation [N.Savoiu 2002] [J.Zhu 2001]

• Model of Computation
  – Kahn Process Network (KPN) [G. Kahn 1974]
  – Synchronous Dataflow (SDF) [E.A.Lee 1987]
  – Cyclo-static Dataflow (CSDF) [G.Bilsen 1995]
  – ConcurrenC [W.Chen, R.Doemer, 2009]

• Static Scheduling
    • Builds an incidence matrix of the connected model graph
    • Solves a balance equation to get repetition vectors, and periodic admissible sequential schedule (PASS)
  – Model simulation in SLDL by modifying the simulation kernel [H.Patel 2004, 2005]
  – Analysis tool SDF3 for throughput, storage capacity, and buffer size minimization (NP-complete [S.S.Battacharyya 1996]) [S.Stuijk 2006, 2008]
• **Simulation Strategy**
  - Periodic ConcurrenC Model
    - Not input dependent, KPN-like, Periodically Schedulable
    - static schedulable \(\rightarrow\) **optimized implementation**
    - Otherwise \(\rightarrow\) **Generic SLDL implementation, DE simulated**
  - Non-Periodic ConcurrenC Model \(\rightarrow\) **Generic SLDL implementation, DE simulated**
A SDF-like ConcurrenC example
A SDF-like ConcurrenC example
A SDF-like ConcurrenC example
A SDF-like ConcurrenC example

The Periodic Admissible Sequential Schedule (PASS) is: a, c, c, b
Optimization for Fast Simulation

SpecC description for the SDF-like model

```
behavior Main(void)
{
    c_int_queue ch0(1ul), ch1(2ul), ch2(2ul);
    a b_a(ch0, ch1);
    b b_b(ch0, ch2);
    c b_c(ch1, ch2);
}

int main(int argc, char** argv)
{
    par
    {
        b_a.main();
        b_b.main();
        b_c.main();
    }
    return 0;
}
```

Optimized SpecC description for the SDF-like model

```
behavior a_seq(i_int_sender ch0, i_int_sender ch1)
{
    c_int_queue ch0(1ul), ch1(2ul), ch2(2ul);
    a_seq b_a(ch0, ch1);
    b_seq b_b(ch0, ch2);
    c_seq b_c(ch1, ch2);
}

int main(int argc, char** argv)
{
    while(1)
    {
        b_a.main();
        b_c.main();
        b_b.main();
        b_b.main();
    }
    return 0;
}
```

PASS is (a, c, c, b)
Scheduling Algorithm for SDF-like Models

Start

Scan each process. If the process has initialization part, run it (put it into the scheduling history stack)
Count = 1

Get the candidate Processes’ set to be scheduled.
Pc = {p | there are enough tokens in the channels p is waiting for.}
Count ++

Separate Pc into two sets, Pe and Pne, where
Pe = {p | p is in Pc, p does not produce any token.}
Pne = {p | p is in Pc, but p is not in Pe.}

Get the set Pm
Pm = {p | p is in Pne, and the number of the tokens stored in the input channel is the greatest.}

Select the first p from Pm.
Schedule p, push p into the scheduling history stack, update the lists, Store the Token list into the status history hashing table.

Get one p from Pe.
Schedule p, push p into the scheduling history, update the Token list, remove p from Pe.
Store the Token list into the status history hashing table.

Check the map to see if there is any identical status.
Y
Get the periodic scheduling order
stop

N
stop

Finding periodic scheduling order failed

Y
N

N

Y
SDF-like Model Scheduling Example

List of the number of tokens in each channel (Token[ch])

<table>
<thead>
<tr>
<th>STEP</th>
<th>List of the number of tokens in each channel (Token[ch])</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ch0 0</td>
</tr>
<tr>
<td></td>
<td>ch1 0</td>
</tr>
<tr>
<td></td>
<td>ch2 0</td>
</tr>
</tbody>
</table>

List of the channels that each process is waiting for (the number of tokens required) chWait[proc][ch]

<table>
<thead>
<tr>
<th>process</th>
<th>chWait[proc][ch]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>ch0(1)&amp;ch2(2)</td>
</tr>
<tr>
<td>c</td>
<td>ch1(1)</td>
</tr>
</tbody>
</table>

01/19/2010
SDF-like Model Scheduling Example

STEP 0

<table>
<thead>
<tr>
<th>STEP</th>
<th>0</th>
</tr>
</thead>
</table>

List of the number of tokens in each channel (Token[ch])

<table>
<thead>
<tr>
<th>ch</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ch0</td>
<td>a</td>
</tr>
<tr>
<td>ch1</td>
<td>1</td>
</tr>
<tr>
<td>ch2</td>
<td>2</td>
</tr>
</tbody>
</table>

List of the channels that each process is waiting for (the number of tokens required)

<table>
<thead>
<tr>
<th>proc</th>
<th>chWait[proc][ch]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>ch0(1)&amp;ch2(2)</td>
</tr>
<tr>
<td>c</td>
<td>ch1(1)</td>
</tr>
</tbody>
</table>

01/19/2010 (c) 2009 W. Chen and R. Doemer
STEP 0

List of the number of tokens in each channel (Token[ch])

<table>
<thead>
<tr>
<th>STEP</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ch1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ch2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

List of the channels that each process is waiting for (the number of tokens required)

<table>
<thead>
<tr>
<th>a</th>
<th>ch0(1)&amp;ch2(2)</th>
<th>ch0(1)&amp;ch2(2)</th>
<th>ch0(1)&amp;ch2(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
</tr>
<tr>
<td>c</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
</tr>
</tbody>
</table>
SDF-like Model Scheduling Example

List of the number of tokens in each channel (Token[ch]):

<table>
<thead>
<tr>
<th>STEP</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ch1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ch2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

List of the channels that each process is waiting for (the number of tokens required):

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>chWait[proc][ch]</td>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch1(1)</td>
</tr>
<tr>
<td>ch0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ch1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ch2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
SDF-like Model Scheduling Example

STEP 0

List of the number of tokens in each channel (Token[ch])

<table>
<thead>
<tr>
<th>STEP</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch0</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>ch1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ch2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

List of the channels that each process is waiting for (the number of tokens required)

<table>
<thead>
<tr>
<th>a</th>
<th>-</th>
<th>-</th>
<th>chWait[proc][ch]</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch0(1)&amp;ch2(2)</td>
</tr>
<tr>
<td>c</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
</tr>
</tbody>
</table>
SDF-like Model Scheduling Example

List of the number of tokens in each channel (Token[ch])

<table>
<thead>
<tr>
<th>STEP</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ch1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ch2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

List of the channels that each process is waiting for (the number of tokens required)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch0(1)&amp;ch2(2)</td>
<td>ch0(1)&amp;ch2(2)</td>
</tr>
<tr>
<td>ch1(1)</td>
<td>ch1(1)</td>
<td>ch1(1)</td>
</tr>
</tbody>
</table>

PASS
KPN-like ConcurrentC example that qualifies as CSDF-like
Scheduling of Periodic Concurrent Models

- Additional Data Structure (for channel prediction)
  - Internal state of each process (local control variables).
    ➔ acquired dynamically when scheduling
  - Waiting channel list for each process and the number of input tokens for each process for each iteration. ➔ acquired dynamically when scheduling

- Code Slicing
  - REPLACE: replace channel send / receive with Token[P] list updating (in _t()) and chWait[N][P] predicting (in _ch()).
  - REMOVE: (a) the loop statement, e.g. while(1); (b) statements dealing with the data variables.
  - KEEP: statements dealing with the state variables (in _t()) and make these variables static to the functional block.
Code Slicing Example

SpecC description for the KPN-like model

```c
behavior f(i_int_receiver Y, i_int_receiver Z, i_int_sender W)
{
    int b = 1;
    void main()
    {
        while(1)
        {
            if(b)
                Y.receive(&i);
            else
                Z.receive(&i);
            X.send(i);
            b = (b + 1) % 2;
        }
    }
};
```

Sliced description code for the KPN-like model

```c
void f_slice::f_t()
// update Token[]
{
    //while(1)
    //}
    if(b)
        Token[Y] --;
    else
        Token[Z] --;
    Token[X] ++;
    b = (b + 1) % 2;
    //}
}
```

```c
void f_slice::f_ch()
// update chWait[]
{
    //while(1)
    //}
    if(b){
        chWait[f_proc][Y] = 1;
        chWait[f_proc][Z] = 0;
    } else{
        chWait[f_proc][Y] = 0;
        chWait[f_proc][Z] = 1;
    }
}
```
Periodic Concurrency Model Scheduling Example

List of the number of tokens in each channel. Token[ch]

<table>
<thead>
<tr>
<th>STEP</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

List of the channels that each process is waiting for. chWait[proc][ch] (the number of tokens required for each iteration)

<table>
<thead>
<tr>
<th>h(0)</th>
<th>T1(1)</th>
<th>T1(1)</th>
<th>T1(1)</th>
<th>T1(1)</th>
<th>T1(1)</th>
<th>T1(1)</th>
<th>T1(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(1)</td>
<td>T2(1)</td>
<td>T2(1)</td>
<td>T2(1)</td>
<td>T2(1)</td>
<td>T2(1)</td>
<td>T2(1)</td>
<td>T2(1)</td>
</tr>
<tr>
<td>f</td>
<td>Y(1) &amp; Z(0)</td>
<td>Y(0) &amp; Z(1)</td>
<td>Y(0) &amp; Z(1)</td>
<td>Y(1) &amp; Z(0)</td>
<td>Y(1) &amp; Z(0)</td>
<td>Y(0) &amp; Z(1)</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>X(1)</td>
<td>X(1)</td>
<td>X(1)</td>
<td>X(1)</td>
<td>X(4)</td>
<td>X(1)</td>
<td>X(1)</td>
</tr>
</tbody>
</table>

Internal condition variables

| f.b | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| g.b | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
Experiment Results

• Static Scheduling Algorithm Performance
  – Heuristic static scheduling algorithm
  – Optimized heuristic static scheduling algorithm (status comparisons are reduced)
  – sdf3 scheduling algorithm

•Simulation speed of the models
  – comparison is made between generic DE implementation and static scheduled implementation.
Experiment Results

Speedup Factor for SDF Models

- sdf5
- sdf10
- sdf25
- sdf50
- sdf75
- sdf100

Run Time Ratio

Speedup Factor for CSDF Models

- cedf5
- cedf10
- cedf25
- cedf50
- cedf75
- cedf100
- JPEG_Encoder

Run Time Ratio
Experiment Results

Speedup Factor for SDF Models

- sdf5
- sdf10
- sdf25
- sdf50
- sdf75
- sdf100

exec time, opt exec time, sdf3 exec time

Run Time Ratio

0.00% 100.00% 200.00% 300.00% 400.00% 500.00% 600.00%

Speedup Factor for CSDF Models

- c sdf5
- c sdf10
- c sdf25
- c sdf50
- c sdf75
- c sdf100
- JPEG_Encoder

exec time, opt exec time, sdf3 exec time

Run Time Ratio

0.00% 50.00% 100.00% 150.00% 200.00% 250.00% 300.00% 350.00%

01/19/2010

(c) 2009 W. Chen and R. Doemer
Conclusion and Future Work

• Conclusion
  – A new heuristic scheduling algorithm of periodic ConcurrenC models are discussed.
  – The algorithm achieves speedup than classic matrix-based algorithms and can handle large models which cannot be handled by the previous algorithm in reasonable time.
  – Static Scheduling helps to improve the simulation speed of periodic ConcurrenC models.

• Future work
  – Support for timing information in ConcurrenC blocks.
  – Extend this scheduling strategy into a distributed simulation environment.
backups
Algorithm Performance Optimization

• Main idea: Reduce the time of status comparison.
• Length of PASS is $l$, PASS is found at $ith$ scheduling step
  – Number of comparisons without optimization: $\frac{i(i-1)}{2}$

\[ \text{Diagram showing scheduling steps} \]

(c) 2009 W. Chen and R. Doemer
Algorithm Performance Optimization

- Main idea: Reduce the time of status comparison.
- Length of PASS is $l$, PASS is found at $ith$ scheduling step
  - Number of comparisons without optimization: $i(i-1)/2$

- Compare every $N$ steps, PASS is found at $tNth$ scheduling step ($tN >= i$), the length of PASS is still $l$ (proved in the paper).
  - Number of comparisons without optimization: $t(t-1)N/2$
  - Speedup for comparison is almost $N$. 