Communication between Nested Loop Programs via Circular Buffers in an Embedded Multiprocessor System

Tjerk Bijlsma², Marco Bekooij¹, Pierre Jansen², Gerard Smit²
Outline

- Problem statement
- Target application
- Related methods
  - Communication via circular buffers
    - Circular buffer
    - Sliding windows
    - Extending the nested loop programs
    - Buffer capacities
- Case study
- Conclusion
Problem statement

- How to arrange communication in a multiprocessor system?
  - Application is a task graph
  - A task contains a nested loop program
  - Nested loop programs communicate by writing and reading arrays
Solution direction

- Organize the communication and synchronization of arrays via buffers
- Locate the buffer in the scratch pad memory of the consuming task
Remaining issues

- How do the tasks use the buffer?
- How to determine the buffer capacity?
Target Application
**Target Application**

- Application is an acyclic task graph
- Task contains a nested loop program
  - Single assignment code
  - Side effect free functions
  - Constant loop bounds
  - Index expressions with iterators as variables
  - **Not limited to affine index expressions**

```c
int X[12];
int Y[12];
for i0:0:5
  for i1:0:1{
    X[2i0-i1+1] = F1(~);
    Y[2i0-i1+1] = F2(~);
  }
```

```c
int X[12];
int Y[12];
for i0:0:5
  for i1:0:1{
    X[2i0-i1+1] = F1(~);
    Y[2i0-i1+1] = F2(~);
  }
```

```c
int X[12];
int Y[12];
for i0:0:5
  for i1:0:1{
    X[2i0-i1+1] = F1(~);
    Y[2i0-i1+1] = F2(~);
  }
```
Target Application

- Communication granularity
  - Locations

- Access patterns in array
  - **Out-of-order access**
  
<table>
<thead>
<tr>
<th>Production into Y (α(task1,Y,i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 3 2 5 4 7 6 9 8 11 10</td>
</tr>
</tbody>
</table>

  - **Multiplicity**

<table>
<thead>
<tr>
<th>Consumption from Y (α(task2,Y,i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 5 6 7 8 9 10 11</td>
</tr>
</tbody>
</table>

  - **Skipping**

<table>
<thead>
<tr>
<th>Consumption from Z (α(task3,Z,i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Production into Z (α(task2,Z,i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13</td>
</tr>
</tbody>
</table>
Related methods
Related methods

- Single window (E. de Greef et.al., 1997 and D. Gannon et.al. 1988):
  - Window contains values after they are written until they are read
  - Single processor solution
  - Requires affine index expressions

- FIFO buffer (A. Turjan et.al., 2004):
  - Depending on access pattern a reordering memory and process are required
  - Requires affine index expressions

- Read and write window (K. Huang et.al., 2007):
  - Provides no analysis to determine either window sizes or deadlock freedom of application
Communication via circular buffers
Communication via circular buffers

- Circular buffer
  - Contains a read and a write pointer
  - Mutual exclusive access in either, read or write part
  - Allows out-of-order access, multiplicity and skipping

- Synchronization
  - Requires a memory consistency model
    (Streaming memory consistency [Brand et. al., DSD 2007])
    - Acquire and release each location in the buffer
    - Allows posted writes
Communication via circular buffers

- Circular buffer + synchronization
  - A sequence of acquired locations
  - Every iteration of a nested loop program conditionally acquires and releases one location
  - A nested loop program acquires and releases consecutive locations
    - Results in a sliding read and write window
Communication via circular buffers

<table>
<thead>
<tr>
<th>Location acquired</th>
<th>Production in Y(α(task1,y,i))</th>
<th>Location released</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
<td>1 0 3 2 5 4 7 6 9 8 11 10</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
</tbody>
</table>

- **Content of a sliding window**
  - **Lead-in** \( d_1(t,b) = \max_i \{ \alpha(t,b,i) - i \} \)
    - Number of initially acquired locations
    - Guarantees that accessed location is acquired
  - **Lead-out** \( d_2(t,b) = \max_i \{ i - \alpha(t,b,i) \} \)
    - Number of initial iterations without a release
    - Guarantees that a location is not released before the last access

- **Window size** : \( d_1 + d_2 + 1 \)
Communication via circular buffers

- Extending the nested loop programs to use sliding windows
  - Add acquire and release statements
  - Replace read and write statements

```c
int c1 = 1;
acquire(1,X);
acquire(1,Y);
for i_0:0:5
  for i_1:0:1{
    if(c1 < 12)
      acquire(1,X);
    if(c1 < 12)
      acquire(1,Y);
    write(2i_0-i_1+1,X,F1(~));
    write(2i_0-i_1+1,Y,F2(~));
    if(c1 > 1 )
      release(1,X);
    if(c1 > 1 )
      release(1,Y);
    c1++;
  }
release(1,X);
release(1,Y);
```

<table>
<thead>
<tr>
<th>Buffer</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lead-in (d_1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Lead-out (d_2)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Communication via circular buffers

- Global notion of synchronization to determine the buffer capacities
  - Acquire operation in buffer Z depends upon acquire in buffer X

```c
int c1 = 1;
acquire(1,X);
acquire(1,Y);
for i_0:0:5
  for i_1:0:1 {
    if(c1 < 12)
      acquire(1,X);
    if(c1 < 12)
      acquire(1,Y);
    write(2i_0-i_1+1,X,F1(~));
    write(2i_0-i_1+1,Y,F2(~));
    if(c1 > 1)
      release(1,X);
    if(c1 > 1)
      release(1,Y);
    c1++;
  }
release(1,X);
release(1,Y);
```

```c
int c2 = 1
acquire(0,Y);
acquire(0,Z);
for j_0:0:1
  for j_1:0:6{
    if(c2 < 13)
      acquire(1,Y);
    if(c2 < 15)
      acquire(1,Z);
    write(7j_0+j_1,Z,
         F5(read(5j_0+j_1,Y)));
    if(c2 < 2)
      release(1,Y);
    if(c2 < 0)
      release(1,Z);
    c2++;
  }
release(1,Y);
release(0,Z);
```

```c
int c3 = 1
acquire(11,X);
acquire(0,Z);
for k_0:0:3
  for k_1:0:2{
    if(c3 < 2)
      acquire(1,X);
    if(c3 < 13)
      acquire(1,Z);
    ~ = F3(read(11-3k_0-k_1,X));
    ~ = F4(read(3k_0+k_1,Z));
    if(c3 < 11)
      release(1,X);
    if(c3 > 0)
      release(1,Z);
    c3++
  }
release(11,X);
for k_3:0:1:1 {
  acquire(1,Z);
  release(1,Z);
  }
```

```c
int c1 = 1;
acquire(1,X);
acquire(1,Y);
for i_0:0:5
  for i_1:0:1 {
    if(c1 < 12)
      acquire(1,X);
    if(c1 < 12)
      acquire(1,Y);
    write(2i_0-i_1+1,X,F1(~));
    write(2i_0-i_1+1,Y,F2(~));
    if(c1 > 1)
      release(1,X);
    if(c1 > 1)
      release(1,Y);
    c1++;
  }
release(1,X);
release(1,Y);
```
Communication via circular buffers

- Cyclo static dataflow model gives global notion of synchronization
  - Execution of a loop-nest shows periodic behavior
  - Sufficient buffer capacities can be calculated [Wiggers et.al., DAC2007]
Case study
Case study

- A fragment of a digital audio broadcasting (DAB) channel decoder
  - Demapper (DM)
  - Frequency deinterleaver (FDI)
    - Pseudo random function \( F(j_0) \) determines read location
  - Time deinterleaver (TDI)

```
int x[2048];
for i_0:0:2047{
    x[i_0] = ~;
}

int y[1536];
for j_0:0:1535{
    y[j_0] = x[F(j_0)];
}

for k_0:0:1535{
    ~ = y[k_0];
}

int F(int index){
    int c=0,val=511;
    while (c < index){
        val = ((val * 13)+511)%2048;
        if ((val ≥ 256)/
            (value ≠ 1024)/(val ≤ 1792))
            c++;
    }
    return val;
}
```
Case study

- Sufficient buffer capacities to guarantee deadlock free execution
  - Capacity of buffer:
    - \( X = 3464 \)
    - \( Y = 388 \)

```
int X[2048];
for i0:0:2047{
    X[i0] = ~;
}
```

```
int Y[1536];
for j0:0:1535{
    Y[j0] = X[F(j0)];
}
```

```
for k0:0:1535{
    ~ = Y[k0];
}
```

<table>
<thead>
<tr>
<th>Task</th>
<th>DM</th>
<th>FDI</th>
<th>TDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>X</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>Lead-in (d_1)</td>
<td>0</td>
<td>1713</td>
<td>0</td>
</tr>
<tr>
<td>Lead-out (d_2)</td>
<td>0</td>
<td>1235</td>
<td>0</td>
</tr>
</tbody>
</table>
Conclusions

- We used sliding windows in a circular buffer for inter-task communication
  - Suitable for nested loop programs with non-affine index expressions
  - Use a read and a write window
    - Allows out-of-order access, multiplicity and skipping patterns

- Sufficient buffer capacities computed with the derived cyclo static dataflow model
  - Guarantee deadlock free execution

- Demonstrated approach on a fragment of a digital audio broadcasting (DAB) channel decoder
  - We could handle the pseudo random read function of frequency deinterleaver