Mapping Stream based Applications to an Intel IXP Network Processor using Compaan

Sjoerd Meijer (PhD Student)
University Leiden, LIACS
smeijer@liacs.nl
Outline

• Need for multi-processor platforms
• Problem: how to program them
• Solution: compiler support
• Realization: the IMCA back-end
• Conclusion
Problem Description

Storage arrays (R1) are located in Global Memory

Sequentially Ordered

We need tools to convert the sequential program to a parallel equivalent

Communicate via unbounded FIFOs

Processes run autonomously
Our Solution

Sequential Application Specification

```
for j = 1:1:5,
    for i = 1:1:5,
        [r(i,j)] = ReadMatrix_Zeros_64x64(x);
        end
    end
for k = 1:1:6,
    for j = 1:1:5,
        [r(j,k), x(k,h), t] = Vectorize( r(j,h), x(k,h) );
        for i = j+1:1:5,
            [r(i,j), x(k,i), t] = Rotate( r(i,j), x(k,i), t );
            end
        end
    end
end
```

DIFFICULT to map

EASY to map

IMCA

COMPAAN

APPLICATION

DIFFICULT to specify

PARALLEL APPLICATION SPECIFICATION

P1

P2

P3

P4

P5
Questions

• Q1: Can we use the IXP architecture for stream-based applications?
• Q2: Can we map applications written as a KPN onto the IXP?
• Q3: Can we program the IXP using Compaan?
Intel IXP2400 Network Processor

• Optimized for streaming
  – 2.6 Gbit ethernet connection
• Build to operate in real-time on internet traffic
  – 8 microengines with 8 hardware supported threads
• Completely programmable in C
  – A SDK available for programming
IXP2400

- 8 microengines
- 1 XScale core
- MSF interface
- SRAM: 128 MB
- DRAM: 1 GB
- Scratchpad: 4 K
Microengine

• Local memory: 640 K
• Registers
  – GPR
  – Read Xfer
  – Write Xfer
  – Next neighbour
• Instruction store: 4 K
• 8 Threads
Intel IXP Network Processor

• Not used on a large scale
• Difficult to program:
  – Write code per microengine
  – Infrastructure
  – Synchronisation
  – Non-unified complex memory model

Conclusion: easier programming needed
Static Affine Nested Loop Programs

- Nested loop: all statements occur within loops
- Static: control flow known at compile time
- Affine: $ax+b$ expressions
- Explicit array references are required to extract data-level parallelism in the application

```c
01 int i,j;
02 matrix A;
03 for (i = 1, i < 5, i++) {
04   for (j = i, j < 12, j++) {
05     A[i,j] = 3*(i+j) - 3;
06     if (j > 3)
07       A[i,j+1] = i+j;
08   }
09 }
```
Kahn Process Network (KPN)

- Process Networks
  - Processes run autonomously
  - Communicate via unbounded FIFOs
  - Synchronize via blocking read
- Process is either
  - executing (Execute)
  - communicating (Put/Get)
- BENEFITS:
  - Deterministic Behavior
  - Distributed Control
  - Distributed Memory
Signals

• Synchronization between IXP elements
  – Between threads and microengines
  – Memory access

• Wait for signal → context switch

```c
__declspec(sram_read_reg) x; SIGNAL sig;
__declspec(scratch)* addr = 0x400;
scratch_read(&x, addr, 1, sig_done, sig);
do_other_work();
__wait_for_all(&sig);

y = x;
```
FIFO mappings

- Small and frequently used:
  - Scratchpad HW
  - (Next neighbour rings)
- Small and less frequently used:
  - Scratchpad SW
- Large and/or not frequently used:
  - SRAM rings
Process mapping

<table>
<thead>
<tr>
<th>Thread</th>
<th>0:0</th>
<th>0:1</th>
<th>0:2</th>
<th>0:3</th>
<th>1:0</th>
<th>1:1</th>
<th>1:2</th>
<th>1:3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tool Flow Overview

- **IMCA:**
  - **IXP Mapper for Compaan Applications**

  ![Diagram](image)
Code generation

• Visitor design pattern
  – Visits platform description, writes code per element

• One “C” file per microengine

• FIFOs accessed uniformly
  – Static functions to implement FIFO code
  – *Port* struct to specify FIFO variables
Using the IMCA environment
Results

- QR algorithm
  - 5 nodes
  - 12 FIFOs

<table>
<thead>
<tr>
<th>Arch.</th>
<th># clock cycles</th>
<th>CPU freq.</th>
<th>Time micro secs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXP</td>
<td>40247</td>
<td>600 Mhz</td>
<td>67</td>
</tr>
<tr>
<td>FPGA, 5 MB</td>
<td>3865</td>
<td>100 Mhz</td>
<td>39</td>
</tr>
<tr>
<td>FPGA, full HW</td>
<td>213</td>
<td>108 Mhz</td>
<td>2</td>
</tr>
</tbody>
</table>
Discussion

- Work presents a first try, still many open issues
  - Selection of right communication channel
  - Binding of the KPN processes to threads and microengines
  - MSF takes a lot of resources, what is the minimum required.
- Future of the IXP is uncertain, perhaps the Cell Processor is an interesting next research platform
Conclusion

• Q1: The IXP can be used for streaming applications
  – Yes, showed it for QR and DWT
• Q2: We automatically mapped QR
  – Yes, we can map the FIFO communication onto the communication channels and the processes on the threads
• Q3: IMCA back-end generates IXP code
  – Yes, we can use Compaan to automatically generate the Processes and FIFO channels that are subsequently mapped on the IXP