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

Sjoerd Meijer, Bart Kienhuis, Johan Walters, David Snuijf
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

- Ferocious appetite for more embedded computation power

**Solution: A need for Multi Processor Systems**

Source: TI, Xilinx – 1 MAC = 8 bit Multiply-Accumulate
Problem description – cont.

- Moore’s Law:
  - More transistors
- Tooling:
  - More lines of code
- Productivity Gap

We know how to build multiprocessor systems, but not how to program them efficiently.
Problem Description

Sequentially Ordered

for( int t=1; t<=P; t++){   for( int i=1; i<=M; i++ ){      for( int j=4; j<=N; j++ ){          r1[i+1][j-3] = F1(...);
        }    }    for( int l=3; l<=M; l++ ){     for( int m=3; m<=N-1; m++ ){         if ( l+m<=7 ){            r2[l][m] = F2( r1[l-1][m-2] );
        } if ( l+m>=8 ){            r2[l][m] = F3( r1[l][N-3] );
        } ... = F4( r2[l][m] );
        }
    }
}

Storage arrays (R1) are located in Global Memory

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

DIFFICULT to specify

Application

EASY to specify

DIFFICULT to map

COMPAAN

PARALLEL Application Specification

P1

P3

P5

P2

P4

EASY to map

IMCA

IXP2400

ME 0.0
ME 0.2
ME 0.1
ME 0.3
ME 1.0
ME 1.2
ME 1.1
ME 1.3
SRAM
DRAM
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
  – Synchronization
  – 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
Threads

- 8 Threads per microengine
- Minimal costs swapping contexts
- Registers and Memory divided between threads:
  - Private: a copy for each thread
  - Shared: all threads same value
- Instruction store shared by all threads
- 1 Process → 1 Thread
- Threads run in Round Robin schedule
- Non-pre-emptive: programmer must swap
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;
```
Available FIFOs

• In hardware:
  – 16 Scratchpad memory rings
  – 128 SRAM rings
  – 7 Next neighbour rings

• In software:
  – Local memory
  – Direct Xfer register access
  – Scratchpad memory
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
Tool Flow Overview

- IMCA:

  IXP Mapper for Compaan Applications

  Application described in Matlab

  COMPAAN compiler

  Application specification as KPN

  IMCA tool

  Mapping specification

  Platform specification

  Common function libraries for microengines

  Program code for microengines

  Configuration files for microengines

  Intel IXP C-compiler

  ME 0:0
  P1
  ME 0:1
  P2
  ME 0:2
  P3
  ME 0:3
  P4
  ME 1:0
  P5
  ME 1:1
  -
  ME 1:2
  -
  ME 1:3
  -

  SRAM

  DRAM

  IXP2400
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
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