System Design Using Kahn
Process Networks:
The Compaan/Laura Approach

Bart Kienhuis
Assistant Professor
LIACS, Leiden University
The Netherlands
DSP Performance Requirements

Source: TI, Xilinx – 1 MAC = 8 bit Multiply-Accumulate

Applications have a ferocious appetite for more programmable compute power.
Embedded DSP Architectures

- Programmable Interconnect (NoC)
- IPcore
- RPU
- CPU
- Memory
- ... (Weakly coupled Processing elements)

- CPU: A simple Microprocessor
- RPU: Reconfigurable Processing Unit
- IPcore; Dedicated Accelerator block
- NoC: Network on a Chip
Programming Problem

Sequential Application Specification

```matlab
for j = 1:1:N,
    [x(j)] = Source1();
end
for i = 1:1:K,
    [y(i)] = Source2();
end
for j = 1:1:N,
    for i = 1:1:K,
        [y(i), x(j)] = F(y(i), x(j));
    end
end
for i = 1:1:K,
    [Out(i)] = Sink(y(i));
end
```

EASY to specify

DIFFICULT to map

Parallel Application Specification

EASY to map
Outline

- The programming problem
- Kahn Process Networks
- System Design: Compaan/Laura Approach
- Case-study M-JPEG
- Conclusions
To satisfy the computational requirements, these architectures have to exploit:

- Distributed Control
- Distributed Memory
Matlab Code (QR Algorithm)

```matlab
%parameter N 8 16;
%parameter K 100 1000;

for k = 1:1:K,
    for j = 1:1:N,
        [ r(j,j), x(k,j), t ]=Vectorize( r(j,j), x(k,j) );
        for i = j+1:1:N,
            [ r(j,i), x(k,i), t]=Rotate( r(j,i), x(k,i), t );
        end
    end
end
```

Matrices are located in Big Global Memory

QR simple program: but keeps your CPU very busy
Solution

- Change the model of computation in such a way that it better fits the model of architecture.
- Make sure the data-type is of precisely the format that fits the architecture (e.g. Streams)
- What model of Computation would fit this description, when looking at Digital Signal Processing (DSP) applications, Imaging and Multi Media?

Kahn Process Networks
Kahn Process Network (KPN)

- Kahn Process Networks [Kahn 1974][Parks&Lee 95]
  - Processes run autonomously
  - Communicate via unbounded FIFOs
  - Synchronize via blocking read
- Process is either
  - executing (execute)
  - communicating (send/get)
- Deterministic
- Distributed Control
- Distributed Memory
Kahn Process Network (KPN)

- Autonomously operating Processes; no global schedule needed
- Blocking Read simple realize in Hardware
- Buffer Sizes of the FIFOs are quite often very small
The Compaan Tool Chain

Matlab Program

```matlab
% parameter N 8 16;
% parameter K 100 1000;

for k = 1:1:K,
    for j = 1:1:N,
        [r(j,j), x(k,j), t] = Vectorize( r(j,j), x(k,j) );
        for i = j+1:1:N,
            [r(j,i), x(k,i), t] = Rotate( r(j,i), x(k,i), t );
        end
    end
end
```

Polyhedral Reduce Dependence Graph (PRDG)

Data Dependency Analysis

Matlab Application

MatParser

SAC

DgParser

PRDG

Panda

Kahn Process Network

Source

S1

Sink

P1

P2

P3

P1

P2

P3
for i = 1 : 1 : N,
    for j = 1 : 1 : N,
        [ a(i+j) ] = funcA( a(i+j) );
    end
end

The for-next loops define an Iteration Domain

Ax >= b (polytope)
Matlab Code (QR Algorithm)

```matlab
%parameter N 8 16;
%parameter K 100 1000;

for k = 1:1:K,
    for j = 1:1:N,
        [ r(j,j), x(k,j), t ] = Vectorize( r(j,j), x(k,j) );
        for i = j+1:1:N,
            [ r(j,i), x(k,i), t ] = Rotate( r(j,i), x(k,i), t );
        end
    end
end
```
Polyhedral Reduced Dependence Graph

Polytope “C”

Polytope “D”
Linearization

- Linearization is the process of mapping high-order data-structures (e.g., Matrices) on a 1-D stream
- We replaced the indexing of the variable $x(j,i)$ and $x(n-1,m)$ by relative put and get operation on a FIFO buffer (unboxing)
- Is this always possible?
The Laura Tool

Library of IP cores

KPN

KPNtoArchitecture

Network of Virtual Processors

Mapping

Network of Synthesizable Processors

Platform Independent

Platform dependent

Verilog

VHDL

SystemC
The Laura Tool

Structure of an individual processor

DATA FLOW

Execution Unit
IP Core

MUX
DeMUX

Control Tables

Controller

Read Unit
Write Unit

FIFO
System Design Flow

- The Tools in action
  - M-JPEG Example based on the original C-code of the Portable Video Research Group, Stanford University.
  - Simple Target Platform
    - Common PC platform
    - With FPGA board
Motion JPEG encoder

Sequence of $T$ frames

Video stream (4:2:2 YUV format)

JPEG encoding

M-JPEG encoded video stream

observed bitrate
System Design Flow

Software Path
- Software Processes (YAPI)
  - GCC/V++
  - SW Compiler
- Object Code

Application
In Matlab

Compaan Compiler

KPN
HW/SW partitioning
(Workload Analysis)

Hardware Path
- Hardware Processes (Matlab)
  - Compaan/Laura
  - HW Compiler
- Hardware Description
  - VHDL

SW Programming

HW/ SW Programming

Hardware
Design

Virtex-II 2V6000 FPGA
ADM-XRClII board

Microprocessor
Pentium IV

PCI bus

Host Interface

Virtex-II 2V6000 FPGA
ADM-XRClII board

Multiplexer

Address

Control

Data

SW Programming

HW Programming
M-JPEG Specification in Matlab

Parameterized

%parameter NumFrames 1 1000;
%parameter VNumBlocks 16 256;
%parameter HNumBlocks 8 256;

[ QTables, HuffTables, TablesInfo, EndOfFrame ] = P2_L_DefaultTables( );
for k = 1:1:NumFrames,
    [ HeaderInfo ] = P1_L_VideoInInit( );
    for j = 1:1:VNumBlocks,
        for i = 1:1:HNumBlocks,
            [ Block( j, i ) ] = P1_L_VideoInMain( );
        end
    end
end
for j = 1:1:VNumBlocks,
    for i = 1:1:HNumBlocks,
        [ Block( j, i ) ] = OCT( Block( j, i ) );
    end
end
for j = 1:1:VNumBlocks,
    for i = 1:1:HNumBlocks,
        [ Block( j, i ) ] = Q( Block( j, i ), QTables );
        [ Packets, StatisticsB ] = VLE( Block( j, i ), EndOfFrame, HuffTables );
        [ BitRate, StatisticsF, EndOfFrame ] = CtrlF1( StatisticsB );
        [ ] = VideoOut( HeaderInfo, TablesInfo, Packets );
    end
end
[ QTable, HuffTables, TablesInfo ] = P2_L_CtrlF2( BitRate, StatisticsF,
    QTables, HuffTables, TablesInfo );
end

Block( j, i )
Deriving a KPN

Application In Matlab

Compaan Compiler

Workload Analysis to do the HW/SW Partitioning

Functional Verification

Ptolemy II PN Domain

YAPI/C++
Deriving a KPN
The KPN of M-JPEG

```c
struct Block {
    int Y1[64]; /* block 8x8 pixels */
    int Y2[64]; /* block 8x8 pixels */
    int U[64]; /* block 8x8 pixels */
    int V[64]; /* block 8x8 pixels */
};
```
The DCT process is selected to move to Hardware.

Interface code is needed to run with the Software Processes

Observe that ‘Blocks’ are being moved to the FPGA and from the FPGA

```cpp
void DCT::main() {
    // NumFrames = 100;
    // VNumBlocks = 16;
    // HNumBlocks = 8;

    for ( int k=1; k <= NumFrames; k++ ) {
        for ( int j=1; j <= VNumBlocks; j++ ) {
            for ( int i=1; i <= HNumBlocks; i++ ) {
                read( inPort, inBlock );
                outBlock = DCT( inBlock );
                write( outPort, outBlock );
            }
        }
    }
}
```
Matlab of the DCT Process

- Of the DCT process, we make a new Matlab program
  - This exposes more parallelism at a finer lever.
  - Automatic conversion from Blocks to Stream (*Linearization*)

- Compaan produces by default a process per function call.
  - However, using the Preamble ‘P_1’ we can group processes.

```matlab
for k = 1:1:4,
    for j = 1:1:64,
        [ Pixel( k , j ) ] = Source( inBlock );
    end
end
for k = 1:1:4,
    if k <= 2,
        for j = 1:1:64,
            [ Pixel( k , j ) ] = PreShift( Pixel( k , j ) );
        end
    end
    for j = 1:1:64,
        [ Block ] = P_l_PixelsToBlock( Pixel( k , j ) );
    end
    [ Block ] = P_l_2D_dct( Block );
    for j = 1:1:64,
        [ Pixel( k , j ) ] = P_l_BlockToPixels( Block );
    end
end
for k = 1:1:4,
    for j = 1:1:64,
        [ outBlock ] = Sink( Pixel( k , j ) );
    end
end
```
KPN Sub network DCT

Hierarchical Subnet of DCT

DCT

Source → PreShift → Pixel

Pixel → Block

Block → inBlock

OutputBlock

P1 → DCT → Block → P2 → Q → Block → VLE

Packets → VOut

CtrlF1

Qtables

HuffTables

BitRate

StatisticsB → EndOfFrame

StatisticsF → TablesInfo
Programming the CPU

M-JPEG specified in YAPI
Laura DCT Hardware Model

Abstract Hardware Model: Network of Virtual Processors

Sink/Source do the type Conversion

One-to-One Mapping
To get the functionality of the Virtual Processor, we integrated an IPcore.

We have taken the Core (2D-DCT) from the Xilinx Webside.

Make Processor specific for a platform
- Determine the Bit width
- Determine the FIFO sizes
- Take into account the Clock
- Determine the Control tables for the switches
Hw/Sw Solution for M-JPEG

- PCI bus
- YAPI Multithreading Environment
- Pentium IV
- Virtex-II 2V6000 FPGA
- ADM-XRCII board

The way it is programmed; the CPU and FPGA run in parallel
## Processing Time M-JPEG

<table>
<thead>
<tr>
<th></th>
<th>Company</th>
<th>Laura</th>
<th>Other tools</th>
<th>Manually</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-JPEG -&gt; KPN</td>
<td>00:00:22</td>
<td>--</td>
<td>--</td>
<td>00:30:00</td>
<td>00:30:22</td>
</tr>
<tr>
<td>Software Compilation</td>
<td>--</td>
<td>--</td>
<td>00:00:35</td>
<td>--</td>
<td>00:00:35</td>
</tr>
<tr>
<td>DCT Subnet Compilation</td>
<td>00:00:08</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>00:00:08</td>
</tr>
<tr>
<td>Laura</td>
<td>--</td>
<td>00:00:07</td>
<td>--</td>
<td>03:00:00</td>
<td>03:00:07</td>
</tr>
<tr>
<td>Synthesis to FPGA</td>
<td>--</td>
<td>--</td>
<td>00:13:10</td>
<td>--</td>
<td>00:13:10</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td>00:00:30</td>
<td>00:00:07</td>
<td>00:13:45</td>
<td>03:30:00</td>
<td>03:44:22</td>
</tr>
</tbody>
</table>
# Device Utilization for the DCT

<table>
<thead>
<tr>
<th>FPGA resource</th>
<th>Utilization</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MULT18x18s</td>
<td>8 out of 144</td>
<td>5%</td>
</tr>
<tr>
<td>Number of RAMB16s</td>
<td>4 out of 144</td>
<td>2%</td>
</tr>
<tr>
<td>Number of SLICEs</td>
<td>2367 out of 33792</td>
<td>7%</td>
</tr>
<tr>
<td>Number of BUFGMUXs</td>
<td>2 out of 16</td>
<td>12%</td>
</tr>
</tbody>
</table>

Virtex-II 2V6000 FPGA  
(taking 4% of the FPGA)
Real-time performance M-JPEG

- Throughput of the system
  - 10.5 CIF frame (128x128) per second
    - Running Windows 2000
    - Simple Compiler
    - Simple Multithreading architecture
- Required is 25 frames per second
  - Communication FPGA/CPU is too slow (PCI)
- However,
  - 64 bit PCI
  - Running at 66Mhz
  - 4 times increase in performance
- Then 25 frames per second (128x128) not a problem
for j = 1:1:N,
    [x(j)] = Source1( );
end
for i = 1:1:K,
    [y(i)] = Source2( );
end
for j = 1:1:N,
    for i = 1:1:K,
        [y(i), x(j)] = F( y(i), x(j) );
    end
end
for i = 1:1:K,
    [Out(i)] = Sink( y(i) );
end
%parameter N 100 1000;
%parameter K 8 48;

for j = 1:1:N,
    for i = 1:1:K,
        [y(i), x(j)] = F(y(i), x(j));
    end
end

U = [ N, K ]

for j = 1:1:N,
    if mod(j, 2) = 1,
        for i = 1:1:K,
            [y(i), x(j)] = F(y(i), x(j));
        end
    end
    if mod(j, 2) = 0,
        for i = 1:1:K,
            [y(i), x(j)] = F(y(i), x(j));
        end
    end
end

difficult to derive

MatTransform

Compaan
Retiming/Skewing

Skewing matrix

\[ M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \]

Unfolding vector

\[ U = [ u_1, u_2 ] = [2, 1] \]

%parameter N 100 1000;
%parameter K 8 48;
for j = 1:1:N,
  for i = 1:1:K,
    \[ [y(i), x(j)] = F(y(i), x(j)) \]
  end
end

for j = 2:1:N+K,
  for i = max(1, j-N):1:min(j-1, K),
    \[ [y(i), x(j-i)] = F(y(i), x(j-i)) \]
  end
end

if \( \text{mod}(j, 2) = 1 \),
for i = max(2, \text{mod}(j, 2)-1):1:min(j-1, K),
  \[ [y(i), x(j-i)] = F(y(i), x(j-i)) \]
end
end

if \( \text{mod}(j, 2) = 0 \),
for i = max(1, j-N):1:min(j-1, K),
  \[ [y(i), x(j-i)] = F(y(i), x(j-i)) \]
end
end

for j = 1:1:N,
  for i = 1:1:K,
    \[ [y(i), x(j)] = F(y(i), x(j)) \]
  end
end

Compaan

Difficult to derive
Conclusions

- To satisfy tomorrow’s applications, we will see hierarchical multiprocessor systems with a number of CPUs, Memories, IPcores, and RPU.
- Programming these system will be difficult unless the MoC is changed to take Concurrency into account. The key items will be
  - Distributed Memory
  - Distributed Control
- Kahn Process Networks seem to be a very promising programming formalism for tomorrow’s HW/SW codesign platforms
Conclusions

- We showed proof-of-concept with a case in which we Convert M-JPEG into a KPN of which the processes are mapped either on hardware or software.
- In the M-JPEG case, the hardware and software were running concurrently, exploiting task-level parallelism.
- Having good tools, we can start from (legacy) code in Matlab, C, or other imperative languages.
- The results are just the beginning. There is more to achieve when more mature / commercial products are used (RTOS, Compiler, Target Platform, Virtex Pro).
Publications