PROGRAMMING FPGAS
KOEN BERTELS

DELTFT UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING LAB

MICROSOFT SUMMIT

PARIS APRIL 13-15 2011

RECONFIGURABLE COMPUTING COMES OF AGE

BlueBee Multicore Technologies
Overview

- FPGAs and the Multi-core era
- Challenges of FPGA programming
- How can this be done?
- The Molen Abstraction Layer
- The BlueBee Tool Chain
- Summary
The Multi-core Era

- The end of Moore
  - Scaling of technology has reached its limits
    - Power, heat dissipation
  - Rather than one more powerful processor, have multiple ones on the same chip = Multi-core

- Customisation of hardware
  - Use computing cores best suited for the task
  - Different tasks \( \rightarrow \) different hardware = heterogeneity

The era of heterogeneous multi-core platforms is here!

How to use such platforms efficiently is the big challenge
How do FPGAs fit in this picture?

- Hardware efficiency and Software flexibility
- Have dedicated hardware available for certain parts of the application
- Decide what parts of program will become ‘hardware’
- If ‘programmability’ dimension is added, FPGAs score very badly
Typical use of FPGAs for Acceleration

There exist different kinds of hardware platforms:
- Digital signal (multi) processors
- Field Programmable Gate Arrays
  Combined with General Purpose Processors
FPGAs and Industry

- Increasingly used for heterogeneous multi-core computing
  - Convey HC-1, Xilinx Zynq, Atom/Altera
- ES industry: FPGAs, on condition that they are combined with GPPs.
  - How easy are they to program?
  - How to use the existing code base?
Challenges of FPGAs

- Difficult to program
  - ‘pure’ hardware design
  - Limited resources

- Use of hardware compilers
  - Take –ideally- ANSI C as input
  - Not as good as ‘software’ compilers (yet)
  - Many limitations in ANSI C

- Difference between software (sequential) and hardware (parallel)

Challenge: how to make FPGAs easily programmable?
How can this be done?

- BlueBee – spin off from FP6 hArtes project
- By adding pragma’s
- Determine where to put them by using the automatic approach (hartes-bluebee) or do it manually

**Example**

```c
#pragma map call_hw MAGIC 1
void funcA(int *p); // all calls to funcA will be executed on the mAgicV DSP
...
#pragma map call_hw VIRTEX4 2
    funcB(x); // this particular function will be executed on the Virtex4 FPGA
```

- All decisions can be changed by the developer
Is it really that simple?

- Yes: pragma’s are the way to annotate the code and map parts of the application on the FPGA
- No: A lot needs to be done….hopefully as automatic as possible

Key concept: make the FPGA look like a regular computer

The Molen Machine Organisation
The Molen Architecture

Main Memory

Instruction Fetch

Data Load/Store

Arbiter

Data Memory

Mux/Demux

Register File

Core Processor

Reconfigurable microcode unit

FPGA

Schratch pad CCU1

... Schratch pad CCU2

DSP

Exchange Registers
The Molen Abstraction Layer (MAL)

- The Molen Architecture
  - An instance of a heterogeneous system machine organization
  - Extensible (support various processing elements)
  - Can be used for both legacy and newly written code
  - Enables a unifying programming model
- Exposes a uniform machine organisation to the developer, independent of the underlying hardware
- The MAL maps the Molen Architecture on the hardware
- For each platform, a specific MAL is developed
The Molen Abstraction Layer for Virtex 5

Molen Abstraction Layer

- Molen primitives:
  - set
  - execute
  - movfx
  - movfx
  - break
- Molen runtime system:
  - Molen controller runs on GPP
  - Molen proxy server runs on PE
- Molen concepts:
  - shared memory
  - transfer registers
  - reconfigurable functionality

Conceptual view

Main memory (DDR)

Platform view

Core processor (PPC 405)

Virtex5 FPGA
The Molen Machine on Virtex5
The Molen Abstraction Layer - MAL

- Problems addressed
  - Fast integration of custom computing units (CCU) that run on e.g. FPGAs combined with general purpose processor (GPP)
  - Management of dynamical and partial reconfigurability
  - Concurrent execution of multiple kernels

Support a familiar programming paradigm: sequential memory consistency model
FPGA based platforms
Some Facts and Figures

- **Noise Filter ARM**
  - 3x application speedup

- **Wave Field Synthesis PPC**
  - 1.85-3.05 application speedup

- **AES PPC**
  - 20x kernel speedup

- **FPGA**
  - 2 min to compile and generate

**Wavefield Synthesis**
- Kernel function: floating point C based FFT
- Compiled to VHDL using DWARV C-to-VHDL compiler
- Synthesized at 100MHz
- Kernel Speedup 2.49 compared to PPC440@400MHz

**Application** (10 input sources, 32 output channels) – speedups compared to PPC440@400MHz
- Sequential version (1 fft in hw): speedup 1.85
- Parallel version (5 fft-s in hw): speedup 2.23
- Higher parallelism (10 fft-s in hw): speedup 3.04
The BlueBee Tool Chain

int foo(int a, int b){
    int c;
    c=a+b;
    return c;
}
void main(){
    int x,z;
    z=5;
    x=foo(z,21);
}

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
-- two 4-bit inputs and one 8-bit outputs
entity multiplier is
    -- task transformation and decision mapping
    task transformation
    port(.....
        for i in 1 to 3 loop
            if product_reg(0)='1' then
                ........
            end if;
        end loop;
    end task transformation;
    end behv;
Only some pragmas need to be placed, manually or automatically. The BlueBee tool chain takes care of the rest.
static short power2[15] =
{1, 2, 4, 8, 0x10, 0x20, 0x40, 0x80,
0x100, 0x200, 0x400, 0x800, 0x1000,
0x2000, 0x4000};

static int fmult(int an, int srn) {
    short anmag, anexp, anmant, wanexp, wanmant;
    short retval;
    anmag = (an > 0) ? an : ((-an) & 0x1FFF);
    anexp = quan(anmag, power2, 15) - 6;
    anmant = (anmag == 0) ? 32 :
        (anexp >= 0) ? anmag >> anexp :
            anmag <<= -anexp;
    wanexp = anexp + ((srn >> 6) & 0xF) - 13;
    wanmant = (anmant * (srn & 077) + 0x30)  >> 4;
    retval = (wanexp >= 0) ?
        ((wanmant << wanexp) & 0x7FFF) :
            (wanmant >> -wanexp);
    return (((an ^ srn) < 0) ? -retval : retval);
}

static int quan(int val, short *table, int size){
    int i;
    for (i = 0; i < size; i++)
        if (val < *table++) break;
    return (i);
}
The DWARV compiler

Control => FSM

Arithmetic => Combinational Logic
Unfortunately not!

In FPGA based development there are more constraints than in ‘normal’ PC based programming. One example:

- Very little local memory (Bram)
- Where to store data and where to fetch?

CPU intensive is not necessarily the same as Data intensive

Similar as with caches: exploit data locality
Case Study — Canny Edge Detection

- **Primary Goal** → to have an early yet comprehensive and thorough understanding of the application behavior, particularly, its memory access behavior and requirements.
- The Canny edge detection method is well-known for its optimality among other algorithms.
- The experiments were based on the implementation provided by the Computer Vision Laboratory at the University of South Florida
  → [http://marathon.csee.usf.edu/edge/edge_detection.html](http://marathon.csee.usf.edu/edge/edge_detection.html)
sample input image

applying Gaussian filter masks in x- & y-directions (sigma=2)

gradient magnitude

non-maximal suppression

applying hysteresis (th=0.5, tl=0.4)
Profiling framework

Memory bandwidth
Unique memory accesses
Dependencies between functions
Quipu Prediction Modeling Approach
It indirectly implies that canny is the main function doing all the processing by just calling individual functions to carry out different phases in the edge detection algorithm.

The primary share of the total execution time is attributed to `gaussian_smooth`, which is responsible for applying the Gaussian filter to blur the input image.
QUAD profiling: Original Application (Canny flow)

Step 1: Gaussian smooth

Step 2: Sobel operator

Step 3: Non-maximal suppression

Step 4: Hysteresis

Read in the image

Write in the image
QUAD profiling:
Original Application (bottlenecks + hotspots)

Contribution: 69.1%
Contribution: 3.7%
Contribution: 5.6%
Contribution: 19.8%
Contribution: 1.0%
Contribution: 0.7%
Rewriting application for hardware implementation
(remove memory allocation from kernels)

```c
void magnitude_x_y(short int *delta_x, short int *delta_y)
{
    /* Allocate an image to store the magnitude of the gradient. */
    if ((*magnitude = (short *) calloc(rows*cols, sizeof(short))) == NULL) {
        fprintf(stderr, "Error allocating the magnitude image.\n");
        exit(1);
    }
    hw_magnitude_x_y(delta_x, delta_y, rows, cols, *magnitude);
}
```

Identified kernels allocate memory blocks on heap, however, function calling is not supported by the BlueBee DWARV C2VHDL compiler.

Memory allocation on the on-chip BRAM is taken care of during run-time.

The main computational body of the function is extracted to create a new hardware-version of the kernel.
#pragma map generate_hw 1

void hw_apply_hysteresis(short int *mag, unsigned char *nms, int rows, int cols,
float *tlow, float *thigh, unsigned char *edge)
{
    int r, c, pos, numedges, lowcount, highcount,
i, rr, cc, hist[32768],
    short int maximum_mag, sumpix;

This code is very large because of a 128KB local array that is translated into flip-flops. Note: the Virtex 5 on our ML510 board had only 20480 slices.

Estimated Slices: 2720
Contribution: 3.7%

Estimated Slices: 1143
Contribution: 4.9%

Estimated Slices: 5599
Contribution: 19.9%

Estimated Slices: 36617
Contribution: 1.0%

non_max_supp

Merge two hotspot kernels: hw_gaussian_smooth and
non_max_supp. And intermediate kernels

Contribution: 98.4%
Estimated total Slices: 11727

Contribution: 69.8%
QUAD + Quipu : Merge 2

Estimated Slices: 11314
Contribution: 98.4%

Estimated Slices: 36617
Contribution: 1.0%

Merge hysteresis as well.
Contribution: 99.4%
Estimated Total slices: 47931
QUAD + Quipu: What about Memory?

New kernel formed by merging:
- `hw_gaussian_smooth`
- `hw_apply_hysteresis`

Estimated Slices: 47857
Contribution: 99.4%

Very memory intensive (6MB). Can we reduce memory usage?
QUAD + Quipu: What about Memory?

New kernel formed by merging:
- `hw_gaussian_smooth`
- `hw_apply_hysteresis`

Estimated Slices: 47857
Contribution: 99.4%

Very memory intensive (6MB). Can we reduce memory usage?
And you can continue like this… finding good mappings
FPGAs are not low power but energy efficient

One technology to introduce heterogeneity into architectures

BUT

Hardware design is not Software design

Bridging the gap between HW/SW remains challenging

FPGA manufacturers have never invested a lot in good programming support

Necessity for FPGAs to really become mainstream
Summary: FPGAs coming to age

- **BlueBee**
  - Spin off from hartes project
  - Imperial College and Delft University of Technology are co-founders
  - Release of BlueBee v1.0 in Q4 of 2011
  - [www.bluebeetech.com](http://www.bluebeetech.com)