The Canals Language and its Compiler

Andreas Dahlin, Johan Ersfolk, Guifu Yang, Haitham Habli, Johan Lilius
Åbo Akademi University and Turku Centre for Computer Science

SCOPES 2009
24.04.2009
Outline

- Motivation
- Canals language
  - Building blocks
  - Bitstream parsing
  - Scheduling
- Design case study: MPEG-4 decoder
- Canals compiler
- Future work
- Conclusion
Background

- Computational requirements are continuously increasing

- Using several cores for computation is one solution
  - General purpose cores
  - Hardware accelerators

- Programming many-cores is difficult using current programming approaches

- Energy-consumption is important, especially for battery powered embedded systems
What is Canals?

- Canals is a streaming language
  - Focus on dataflow
  - Applications in the streaming domain
  - Enables energy-efficient development
  - Dynamic scheduling

- Canals also aims to
  - have an intuitive visual and textual syntax
  - allow modular scalable designs
Features

- Powerful data flow descriptions
- Code generation for heterogeneous platforms
- Fine-grained scheduling and execution
  - Scheduling algorithms can be described in the language itself
- Automatic bitstream parsing
Building blocks

- Canals is based on the concept of nodes and links, which connect nodes together.
- Two types of nodes
  - **Kernel**
  - **Network**
- Links are denoted **Channels** are intermediate buffers
  - Output data from nodes is stored before the data is consumed by the next node.
- Other elements: scatter, gather, scheduler, dispatcher
Building blocks

- **Kernel** is the Canals element in which all computations are performed

- Three sections:
  - Variable declarations
  - Initialization section
  - Work block

- Variable values are stored between invocations of a kernel
  - Kernels have a state
Building blocks

- **Channel**
  - generic channel builtin
  - channels can be parametrized

- **Scatter/Gather**
  - distribution and collection of data
  - can also act as switches

- **Network**
  - container element
  - one incoming data port
  - one outgoing data port
network  t_in -> t_out N
{  
  add_channel  ch1, ch2, ch3, ch4, ch5, ch6, ch7 <generic_channel>;
  add_kernel  K1 <k_t1>;
  add_kernel  K2 <k_t2>;
  add_kernel  K3 <k_t2>;
  add_kernel  k4 <k_t3>;
  add_network  N_sub <n_t1>;
  add_scatter  SC <rrobin>;
  add_gather  GA <collect>;
  connect NETWORK_IN  ->  K1 -> ch1 -> SC;
  connect  SC.outport[1]  ->  ch2  ->  K2  ->  ch3  ->  N_sub  ->  ch4  ->  GA.inport[1];
  connect  SC.outport[2]  ->  ch5  ->  K3  ->  ch6  ->  GA.inport[2];
  connect  GA  ->  ch7  ->  K4  ->  NETWORK_OUT;
  connect_control  K4  ->  K1;
}

network  dt_1  ->  dt_2  n_t1  {...}
Bitstream parsing

- Bitstream parsing through automatic deserialization of data in network inputs.

- Bitstream parsing can also be performed in a kernel in the top-level network.

- Substream access or conversion between Canals data types is not considered as bitstream parsing.

```datadef frame
{
  Header h;
  MB[*] content;
}
```

```datadef Header {...}```
Scheduling

- Scheduling is concerned with the task of planning the execution of a Canals network
- Detailed top-level control of scheduling and execution of a program
  - Still it can benefit from distributed computations
- A scheduler in Canals is responsible for planning the execution of a network considering data flow
- Decides the execution order for all elements in a network
  - Model of Computation for a network can be changed by altering the scheduler associated with a certain network
Scheduling

- A scheduler becomes **enabled** when the specified amount of data is available at the incoming data port of a network.

- The scheduler can *look* at data that arrives to the network to be scheduled.

- The scheduler does not execute/trigger kernels; this is the task of the dispatcher.
Architectural model and mapping

Scheduler

K1

N1

N2

K2

Mapping

Cell BE Architectural model

Canals Behavioural Model
(Canals program)
MPEG-4 Decoder Design

- Simultaneous decoding of 4 MPEG-4 streams
  - Based on previous work in [LBSL08, BBS07]
  - XviD decoder as starting point

- Some functionality is accelerated
  - IDCT
  - Inverse AC/DC
  - Variable Length Decoding
  - Inverse scan
  - AC/DC add
  - Inverse quantization
MPEG-4 Decoder Design

**MPEG-4 Decoder**

- **Stream Reader & Bitstream Parser**
  - VOP
  - MB
  - MB_d

- **Motion Vector**
  - MB

- **AC/DC buffer**

- **Reference Frame Memory**
  - (Random Access)

- **Motion compensation**
  - MB_d

**Macroblock Decoder**

- **MD Scheduler**
  - Stream Reader & Bitstream Parser
  - VOP
  - MB

- **Motion Vector**
  - MB

- **Inverse Scan**
  - DC/AC Add

- **AC/DC buffer**

- **Upsampling**
  - IDCT
  - Upsampl.
  - MB reconstruct.

- **Frame composition & Display**
  - MB_d

- **VLD**

- **DC/AC Add**

- **IQ**

- **IDCT**

- **MB reconstruct.**
MPEG-4 Decoder Scheduling

Overall encoder functionality is scheduled by the **MPEG-4 Scheduler**

Macroblock decoding is scheduled by the **MD Scheduler**
MPEG-4 Decoder Scheduling

- Macroblock scheduling is performed by a kernel in the MD Scheduler

```c
kernel MB -> Schedule Flow-Shop
{
    variable Schedule s;
    variable MB mb1, mb2, mb3, mb4;

    work get 0 put 1 look 4
    {
        // Look at 4 macroblocks and compute a schedule
        // according to the flow-shop algorithm.
        mb1 = look(0); mb2 = look(1); mb3 = look(2); mb4 = look(3);
        if (mb1.type == I && mb2.type == I
            && mb3.type == I && mb3.type == P)
        {
            // schedule 3 MBs of type I and 1 of type P optimally.
        }
    ...
    put(s);
}
```
Canals Compiler

The structure of the Canals Compiler
Canals Compiler

- Implemented in Java
  - Designed to be extensible and modular

- The goal is to have a development kit, Canals SDK
  - Compiler
  - Editors
Future work

- Scheduling
  - Schedule composition
    - Flattening of hierarchical networks
  - Scheduling of memory transfers

- Architecture and mapping descriptions

- Improvements on the Canals Compiler
Conclusion

- Many-core heterogeneous architectures is one way of dealing with the continuously increasing computational requirements.
- Programming for heterogeneous platforms is error prone and tedious using current approaches.
- Canals is a language suitable for developing applications (in the streaming domain) for heterogeneous architectures.
- Canals provides fine-grained scheduling and execution of a program.


Thank you for your attention!

Questions?