Previously on MDSPs
Previously on MDSPs

HEVC Encoder Bloc description

Source: ATEME
Previously on MDSPs

HEVC Decoder Bloc description

VLC
Decoding
Dequantization
Inverse Transform
Filtering

Source: Hervé Yviquel
Applications for MDSPs

Application is naturally described with blocs

Downlink Data Encoding

<table>
<thead>
<tr>
<th>Data (bits)</th>
<th>CRC/Turbo Coding</th>
<th>Rate Matching</th>
<th>Interleaving/Scrambling</th>
<th>Modulation</th>
<th>Multi-Antenna Precoding</th>
<th>OFDMA Encoding</th>
</tr>
</thead>
</table>

Symbol Processing

Channel Coding

Channel Estimation

SC-FDMA Decoding/Multi-Antenna Equalization

Data (bits)

Uplink Data Decoding

<table>
<thead>
<tr>
<th>Rate Dematching/HARQ</th>
<th>Turbo Decoding/CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descrambling/Deinterleaving</td>
<td></td>
</tr>
</tbody>
</table>
Previously on MDSPs

Grail of Heterogeneous MPSoCs Programming

- High-level description of Algorithm
- High-level description of Architecture
- Multicore Compiler
- Portable Multicore Program
- Simulator + Debugger + Profiler
- Multicore Runtime

Multicore DSPs – Karol Desnos (kdesnos@insa-rennes.fr)
Course Outline

• Lecture 1 – Maxime Pelcat
  • Introduction to the course
  • Applications for MDSPs

• Lecture 2 – Karol Desnos
  • Languages and MoCs
  • Programming MPSoCs
  • Dataflow MoCs

• Lecture 3 – Maxime Pelcat
  • Hardware Architectures
Course Outline

• Lecture 4 – Karol Desnos
  • Theoretical Bounds
  • Mapping/Scheduling Strategies

• Lecture 5 – Karol Desnos
  • Lab Session
Languages and Models of Computations
What is a Model of Computation (MoC)?

- A set of operational elements that can be composed to describe the behavior of an application.
  - Semantics of the MoC

- Interface between:
  - Computer science
  - Mathematical domain

- A MoC is not a language!
What is a language?

• A set of textual/graphical symbols that can be assembled respecting a well defined grammar to specify the behavior of a program
  ➔ Syntax of the language

• Interface between:
  • Programmer
  • Machine (through a compiler)

• A language implements one or several MoCs
Semantics and Syntax

• Among the 10 most used languages,
  • all 10 are imperative,
  • 7 are object-oriented.

• Other semantics exist. A MoC specifies semantics independently from a language syntax.
Semantics and Syntax

- UML implements object-oriented semantics
- C++ implements object-oriented semantics
- They share semantics but not syntax
A few MoCs

- **Finite State Machine MoCs**
  - Semantics
    - States
    - Transitions based on conditions
  - MoC of imperative languages (C/C++/Java…)
  - Computation is executed on transitions
  - Representing the behavior of a Turing machine
A few MoCs

• **Discrete Event MoCs**
  - Modules react to events by producing events
  - Events tagged in time, i.e. the time at which events are consumed and produced is essential and is used to model system behavior
  - Modules share a clock
  - Used to model HDL behavior

• **Functional MoCs and lambda calculus**
  - No state, a program returns the result of composed mathematical functions: \( \text{result} = f \circ g \circ h(\text{inputs}) \)
  - Based on lambda calculus
  - Haskell, Caml, Scheme, XSLT
  - Functional languages are examples of declarative languages
Languages and MoCs

A few MoCs

• **Petri Nets**
  - Close to state machines but
  - able to represent parallelism
  - Operations are done on transitions

• **Synchronous MoCs**
  - Like in Discrete Events, modules react to events
  - by producing new events
  - Contrary to Discrete Events, time is not explicit and
  - only the simultaneity of events and causality are important
  - Language examples: Signal, Esterel, Lustre…
A few MoCs

• Process Network MoCs
  • concurrent and independent modules (processes) communicate ordered tokens (data quanta) through First-In, First-Out (FIFO) channels
  • Include dataflow process networks
  • Untimed models: the time is abstracted

→ What we naturally used to describe MPEG HEVC and 3GPP LTE processing
Programming MPSoCs
• Parallelisms

• Programming Coarse-Grain Parallelism
Parallelisms

- **Granularity of parallelism**
  - Amount of computation of an atomic operation relatively to the overhead to manage/execute several operations concurrently
Parallelisms

- **Single-Instruction Multiple-Data (SIMD)**
  - Primitive operations (+ - * / >> min max …)
  - Low/No cost when code is suitable (data dependencies)
  - Explicitly specified with intrinsic / implicitly supported by HW

![Diagram](image-url)
Parallelisms

- **Software Pipelining**
  - Loop optimization: start a new iteration before the previous end
  - Low cost when code is suitable (data dependencies)
  - Automatically performed by compilers (-O3)
Parallelisms

- **Instruction Level Parallelism**
  - Out-of-order execution
  - SIMD
  - Register Renaming
- Branch prediction
- SW pipelining
- ...

Overhead

Perfect world

Multicore DSPs – Karol Desnos (kdesnos@insa-rennes.fr)
Programming MPSoCs

Parallelisms

• **Threads**
  • “Independent” sequence of instructions with private context
  • Moderate to high cost => quasi-total hardware abstraction
  • Complex to program: deadlocks, preemption, …
Parallelisms

- Cores + on-chip interconnect
  - Each core runs a bare-metal program (no OS).
  - HW-supported means of communication (Interrupt, DMA, Packet, ...)

---

**Overhead**

- **large**
- **small**

**Operation size**

- **large**
- **small**

**Threads**

**Cores + interconnect**

**Perfect world**

ILP

SW Pipe.

SIMD

Multicore DSPs – Karol Desnos (kdesnos@insa-rennes.fr)
Parallelisms

- **Chips + LAN or WAN**
  - Independent servers/computers communicating through a network
  - Very high cost (communication, distributed memory)
  - High Performance Computing

- MPSoCs

Overhead

- small
- large

Operation size

- small
- large

Perfect world

- ILP
- SIMD

Cores + interconnect

Threads

Chips + L/WAN

Multicore DSPs – Karol Desnos (kdesnos@insa-rennes.fr)
Programming MPSoCs

Parallelisms

- For MPSoCs: Fine and Coarse-grain parallelisms
  - ILP and SW Pipelining: Compiler (& crazy developers)
  - Threads and Interconnect: Developers and/or optimization tools
• Parallelisms

• Programming Coarse-Grain Parallelism
Programming Coarse-Grain Parallelism

- Manually, using threads
  - “Independent” sequence of instructions with private context
  - Usually preemptible
  - Number of thread ≈ number of cores
    - Otherwise, context switching rapidly decreases application performance
      => Test in Laboratories

- Solutions:
  - posix thread (linux/windows)
  - win32 API
  - Boost Threads
  - “Idle” threads on SysBios (TI DSP)
  - ...
Programming Coarse-Grain Parallelism

• Manually, using tasks
  • Short independent sequence of instructions with private context
  • Not preemptible
  • Number of tasks >> number of cores
    • Finer granularity than threads.
    • Using many tasks ensures load-balancing

• Solutions:
  • Multicore Task Management API (MTAPI)
  • OpenEvent Machine
  • StarSs
  • …
Programming Coarse-Grain Parallelism

- Semi-automated, using OpenMP
  - Add `#pragma omp` instruction to parallelize sections of C code
  - Compiler optimization automatically create threads/tasks
  - Shared-Memory targets!
  - Supported on some MPSoCs (including c6x DSPs !!)
- Examples:
  - Split loop iterations among threads
    ```
    #pragma omp parallel for
    for(i=0; i< L; i++) { ... }
    ```
  - Execution barrier for several threads
    ```
    #pragma omp parallel for
    for(i=0; i< L; i++) {
      tab[i] = 0; // init
      #pragma omp barrier
      tab[i] = ... // work on the tab.
    }
    ```
Programming Coarse-Grain Parallelism

And so many other solutions:

- **Message-Passing Interface (MPI), Multicore Communication API (MCAPI)**
  - Communication protocol for parallel computing abstracting architecture concerns.

- **OpenCL, CUDA**
  - GPGPU oriented

- **Cilk, Go Language, Threading Building Blocks, …**
  - Threads without headaches

- **Dataflow programming**
Dataflow MoCs
• Dataflow Models of Computations

• Properties of Dataflow MoCs

• Dataflow Frameworks

• Laboratories Example
Dataflow MoCs

• Compilation for block diagrams
  • Accessible to persons with no knowledge of programming languages

• 1961: BLODI compiler
  • 30 basic DSP blocks (Adder, Filter, Quantizer, …)
  • 1 Punched card for each block.

Dataflow MoCs

• Kahn Process Network (KPN)
  • Actors and Data Ports
  • First-In, First-Out (FIFO) Queues
Dataflow MoCs

• Kahn Process Network (KPN)
• Production/consumption rates of an process can change at each firing, but:
  • Reading from a FIFO is a blocking operation
  • This ensures the determinism of the application
    i.e. Same input stream => Same output stream
Dataflow MoCs

- Kahn Process Network (KPN)
- Process example
  - Process *Sort* separates odd and even values.

```plaintext
Process sort(in int i, out int even, out int odd) {
    int value = i.read(); // Blocking
    if(value % 2 == 0)
        even.write(value);
    else
        odd.write(value);
}
```
Dataflow MoCs

- Kahn Process Network (KPN)
- Determinism example
  - Process *Interleave* output value alternatively taken from its inputs.

```c
Process interleave(in int a, in int b, out int o) {
    static bool = true;
    int value = (bool)? a.read() : b.read();
    bool = !bool;
    o.write(value);
}
```
Dataflow MoCs

- **Dataflow Process Network (DPN)**
- **Firing rules of an actor can change at each firing**
  - Firing rules depend on:
    - State of actors
    - Time / Randomness...
    - Number/value of available tokens

Dataflow MoCs

- Dataflow Process Network (DPN)
- Non-Determinism example
  - Process *Interleave* output value alternatively taken from its inputs.
  - Time-out on FIFO read!

```c
Process interleave(in int a, in int b, out int o) {
    static bool = true;
    int value = (bool)? a.read() : b.read();
    bool = !bool;
    if(no_timeout) o.write(value);
}
```
Dataflow MoCs

• Dataflow Process Network (DPN)
• Non-Determinism example
  • Same inputs with different arrival time produces different output

```c
Process interleave(in int a, in int b, out int o) {
    static bool = true;
    int value = (bool)? a.read() : b.read();
    bool = !bool;
    if(no_timeout) o.write(value);
}
```
• Dataflow Models of Computations

• Properties of Dataflow MoCs

• Dataflow Frameworks

• Laboratories Example
Dataflow MoCs

Properties

• DPN and KPN are Dynamic Dataflow MoCs
  • It is not possible to find a fixed sequence of actor firing that can be repeatedly executed for any input sequence.
  • It is not possible to guarantee that an application will not deadlock for any input sequence.
  • It is not possible to guarantee that an application will always require a finite amount of memory for its execution (tokens can accumulate indefinitely in unbounded FIFOs)
Properties

- **Synchronous Dataflow (SDF)**
  - Stateless actors and data ports with **fixed exchange rates**
  - First-In, First-Out (FIFO) Queues
  - Delays

Properties

• Synchronous Dataflow (SDF)
  - SDF is a static model: It cannot be used to model all applications
  - Production/consumption rates are fixed

• It is possible to find a fixed sequence of actor firing that can be repeatedly executed for any input sequence, with a finite memory footprint and no deadlock.
Dataflow MoCs

Properties

- Synchronous Dataflow (SDF)
  - Data-driven execution: An actor is fired when its input FIFOs contain enough data-tokens.

Properties

- **Synchronous Dataflow (SDF)**
  - Parallelisms: Task / Data / Pipeline / Internal

Dataflow MoCs

Properties

• There exists many other dataflow MoCs.
  • What is the difference between them?

• Decidability
  • Application schedulability can be guaranteed at compile-time

• Determinism
  • Same inputs => Same outputs (regardless of time, randomness, or implementation)

• Compositionality
  • Properties of a graph (topology, firing rules of actors, repetition vector, ...) are independent from the internal implementation of the actors that compose it.
Properties

• **Reconfigurability**
  - Firing rules of actors can change dynamically during the execution of the application, depending on inputs. DPN and KPN are reconfigurable, SDF is not.

• **Predictability (related to reconfigurability)**
  - Amount of time between the reconfiguration of firing rules of an actor and its actual firing. (Relative)

• **Conciseness (or succinctness)***
  - Ability of a MoC to model an application with a small number of actors. (Relative)
Properties

• **Analyzability**
  • Availability of analysis techniques to characterize an application graph (e.g. worst-case latency, memory requirements). (Relative)

• **Expressivity**
  • Complexity of application behavior that can be described. Ex. DPN and KPN are Turing-Complete
    SDF is not.
Properties

• About compositionality: A non-compositional hierarchical model
Properties

- About compositionality: The IBSDF compositional model

Dataflow MoCs
Properties

- About predictability and reconfigurability: The more dynamism, the less predictability
Properties

- There exists many other dataflow MoCs.

<table>
<thead>
<tr>
<th>Feature</th>
<th>SDF</th>
<th>ADF</th>
<th>IBSDF</th>
<th>DSSF</th>
<th>PSDF</th>
<th>PiSDF</th>
<th>SADF</th>
<th>SPDF</th>
<th>DPN</th>
<th>KPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressivity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hierarchical</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Compositional</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reconfigurable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Statically sched.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decidable</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>(X)</td>
<td>(X)</td>
<td>X</td>
<td>(X)</td>
</tr>
<tr>
<td>Variable rates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Non-determinism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

SDF: Synchronous Dataflow
ADF: Affine Dataflow
IBSDF: Interface-Based Dataflow
DSSF: Deterministic SDF with Shared Fifos
PSDF: Parameterized SDF
PiSDF: Parameterized and Interfaced SDF
SADF: Scenario-Aware Dataflow
SPDF: Schedulable Parametric Dataflow
DPN: Dataflow Process Network
KPN: Kahn Process Network
Multicore DSPs – Karol Desnos (kdesnos@insa-rennes.fr)

Dataflow MoCs

Properties

- It is important to choose the appropriate MoC before developing each part of an application.

- LTE example:

```
X Number of root sequences
Noise floor estimation

X Root Sequences X Reception Antennas
X Preambles

Correlate with root sequence
IDFT
Power

Detected Users Timing Advance
```

Static

```
X Reception Antennas
X Preambles

Filter → DFT
Select preamble subcarriers
```

```
X Root Sequences X Reception Antennas
X Preambles

Correlate with root sequence
IDFT
Power
```

```
X Number of root sequences
Noise floor estimation

Peak Search
```

```
Detected Users Timing Advance
```
Dataflow MoCs

Properties

• It is important to choose the appropriate MoC before developing each part of an application.

• LTE example:

```plaintext
X Users
X Code Blocks
Symbol and Bit Processing
Code Block Processing
Transport Block Processing

X Reception Antennas
FFT
Demap Control/Data

Static
Param.
Static
Param.

X symbols (14)
X Reception Antennas
Prepare

Decode Control

Data of Each user

Estimate Channel

Equalize Multiple Antennas

Param.
```
• Dataflow Models of Computations
• Properties of Dataflow MoCs
• Dataflow Frameworks
• Laboratories Example
Dataflow MoCs

Dataflow Framework

- CAL and DDF
  - **CAL** = CAL Actor Language, J. Ecker and J. Janneck
  - Implements the DPN MoC
  - RVC-CAL: Normalized in MPEG
Dataflow Framework

- **CAL and DDF**
  - Via classification, a CAL actor can be transformed into SDF, CSDF in order to gain predictability
  - The Orcc compiler, developed at IETR, is a compiler for CAL
  - From CAL, both HW and SW solutions can be generated
  - Supports real-life applications such as HEVC!

- **Examples**

```plaintext
actor decimate() I => O:
  pant action[ x ] => x \( \neq \) 0: end
  end action[ x ] \( \Rightarrow \) =>
  neg schedule[ s ] => x[ ] x=] > guard s = 0
  end  & a=)1=> end
  s2 ( b ) -- > s1;
  p and r on y[ x ] => guard s = 1
  end
end
```

```plaintext
actor posValue() I == O:
  pos: action[ x ] => O: [ x ]
  guard x \geq 0
  end
neg: action[ x ] => O: end
priority pos > neg;
end
```

```plaintext
actor decimate() I => O:
  s := 0;
  action[ x ] => [ x ]
  guard s = 0
  do
    s := 1;
  end
  action[ x ] => [ x ]
  guard s = 1
  do
    s := 0;
  end
end
```
Dataflow MoCs

Dataflow Framework

- **Ptolemy** (by UC Berkeley)
  - First tool
  - Support for composition of DPN, PSDF and SDF MoCs

- **Lightweight Dataflow Environment (LIDE)** (by U. of Maryland)
  - First tool
  - Support for composition of DPN, PSDF and SDF MoCs

- **MAPS Compiler** (by RWTH Aachen U.)
  - KPN and C language
  - Support C6678

- **SynDEx** (by INRIA)
  - Custom Dataflow MoC (SDF-like) an C Code
  - Similar to Preesm (but closed source)
Dataflow Framework

- **OpenDF (by EPFL)**
  - CAL Code compiler
  - Targeting reconfigurable hardware and multicore systems.

- **SDF For Free (SDF3) (by Technische Universiteit Eindhoven)**
  - SDF, CSDF and SADF
  - Focusing on analysis of SDF application and generation of random graphs

- **Canals (Abo Akademi)**
  - A language and its compiler where schedulers are specified explicitly at each level of hierarchy

- **AccessCore Studio (by Kalray)**
  - Programming for Many-Core MPSoC with 256 cores.
Programming MPSoCs

Programming Coarse-Grain Parallelism

And so many other solutions

Deduce parallelism from Models of Computation

Extract parallelism from code

Simulation oriented

Execution oriented

High Performance Computing

Embedded Systems

Matlab + Simulink

Ptolemy II

PREESM

CAL

OpenMP

SystemC

OpenCL

OpenCL

Matlab + Simulink

Ptolemy II

PREESM

CAL

SystemC

OpenMP

OpenCL

Simulation oriented

Execution oriented

Multicore DSPs – Karol Desnos (kdesnos@insa-rennes.fr)
Dataflow MoCs

- Dataflow Models of Computations
- Properties of Dataflow MoCs
- Dataflow Frameworks
- Laboratories Example
Dataflow MoCs

Laboratories Example

- PiMM Dataflow MoC
  - Use the IBSDF subset + static param


Multicore DSPs – Karol Desnos (kdesnos@insa-rennes.fr)
Laboratories Example

- Sobel Application
  - Sequential version

Dataflow MoCs

Process
Laboratories Example

- Sobel Application
  - Parallel version

Dataflow MoCs

Convolution of a 3x3 Matrix with pixels of the input image

Each slice has a 1-line overlap with the next and previous slices

Split  Merge

Process  Process  Process  Process
Laboratories Example

• Sobel Application
  • Demo time!