Multicore Digital Signal Processing

Maxime Pelcat
2014

Slides from M. Pelcat, K. Desnos, J-F. Nezan, D. Ménard, M. Raulet, J Gorin
Introduction: Porting Algorithms to Multicore DSPs
Typical Code Development Environment

- Algorithm Code
- Command Line Options
- Compiler
- Simulator + Debugger + Profiler
- Program
  - OS
  - Core(s)
Possible MPSoC Dataflow-based Development Environment

- Functional Algorithm Model + Code
- Constraints + Options
- Architecture Model
- Simulator + Debugger + Profiler
- Compiler
- Deployment

Diagram showing the integration of different models and tools for MPSoC development, including OS and Core models.
Addressed Problem

• **Distributed embedded systems are harder and harder to program**

• **Difficult Constraints**
  
  - High computation requirements
  - Low power consumption
  - Many hardware and software choices: lack of information/metrics
  - Real-time constraints (hard of soft real-time)
  - Need to reuse legacy code

• **Difficult Goals**
  
  - Design both hardware and software
  - Balance loads
  - Obtain the most from a given architecture
  - Respect constraints
Addressed Problem

• **Traditional code in C abstracts core architecture**
  Amount of registers
  Number of pipeline stages
  Instruction parallelism
  Loop optimizations
  Cache accesses
  Data representation
  …

• **C code can not be efficiently transformed into coarse grain parallel code**
  Assumed global state in a program
  Unique activity point
  Inspired by the Turing machine

• **The solution may come from dataflow MoCs**
Grail of Multi-core DSP Programming

Algorithm
High Level Description

Architecture
High Level Description

Multi-core Compiler

Multi-core Program

Simulator
+ Debugger
+ Profiler

Multi-core Runtime

ARM
ARM
Co-processor
Co-processor

ARM
ARM
DSP
DSP
DSP
DSP

DSP
DSP
DSP
DSP
Multi-core code porting $\rightarrow$ assignment, ordering and timing
Code Porting

Assignment

Core 1

Task 1

Task 3

Core 2

Task 2

Task 4

Task 1

Task 3

Task 4
Code Porting

Core 1

Core 2

Task 1
Task 2
Task 3
Task 4

Timing

Task 1
Task 3
Task 4
Task 2

t

Multicore DSP
Code Porting

• **And other tasks:**
  Choose communication (shared memory, DMA, direct copy)
  Choose communication synchronization (polling or interrupts)
  Allocate in memory, order and time communications
Code Deployment

Many possible assignments and orders

Minimize latency/response time
Minimize execution time
Minimize memory consumption
Minimize power consumption

Complicated data/control dependencies

Complicated interconnections
Grail of Multi-core DSP Programming

- Optimizing / Offering trade-offs between:
  - Latency / Response time
  - Throughput
  - Load Balancing
  - Memory consumption
  - Power consumption

- Algorithm description portable to new device (DSP, GPU, HPC...)

- Algorithm High Level Description
- Architecture High Level Description
- Simulator + Debugger + Profiler

- Multi-core Runtime
- Multi-core Compiler
- ARM
- DSP
- Co-processor

Multicore DSP
General Outline

- Demo Platform
- Applications
- Models of Computation
- Architectures
- Models of Architecture
- Partitioning and Scheduling Problem
- Compile-time and Runtime Tools
Advantech Board TDMSEVM6678L

- EVM = Evaluation Module for the TMS320C6678
TMS320C6678

• **8-Core DSP**
  8 C66x DSP Core Subsystems (C66x CorePacs), Each with:
  – 1.0 GHz or 1.25 GHz C66x Fixed/Floating-Point CPU Core
    40 GMAC/Core for Fixed Point @ 1.25 GHz
    20 GFLOP/Core for Floating Point @ 1.25 GHz
    Total: 320 GMACs + 160 Gflops: hard to reach!
  – 2 levels of Core Memory
    32K Byte L1P Per Core
    32K Byte L1D Per Core
    512K Byte Local L2 Per Core

• **4 MB of Internal Shared Memory**
  Multicore Shared Memory Controller (MSMC)
  L1D and L1P with automatic cache coherency in local
  Non coherent cache of the shared memory

• **Unified memory space for internal/external memory**
Demo Board

- **512 MB of Shared DDR3 on the emulation board**
  Any core can access DDR3, 8G Byte of DDR3 Addressable Memory

- **Hardware coprocessors**
  For repetitive common operations
  Reduced because multi-purpose processor
  Cryptography
  Network

- **XDS510 JTAG**
  via USB
  Possibility to extend to XDS560 via extension

- **Packaging**
  40nm technology, 841-Pin Flip-Chip Plastic BGA (CYP)
Inter-core Communication

- KeyStoneTeraNet switch fabric (Network on Chip)
- Core Interrupt Controller

- Enhanced Direct Memory Access v3 (EDMA3)
  Data movement
  Like a core with only MOV instructions

- Multicore Navigator
  8192 Multipurpose Hardware Queues with Queue Manager
  Data movement or zero-copy

- Shared MSMC and DDR3
  Data movement or zero-copy
Inter-core Communication

• **Multicore Navigator**
  Queue Manager Subsystem (QMSS)
  Packet DMA (PKTDMA) for Zero-Overhead Transfers
  Packet passing system between cores
  Abstracts real data transfer

• **Open Event Machine**
  Software runtime system provided by TI to offload code on cores
  Event driven processing runtime for multicore
Inter-core Communication
• **Cache operations**

The L1 has automatic cache coherence if local L2 is modified.
L1 has no automatic cache coherence if non-local memory is modified.
L2 has no automatic cache coherence.

→ L1 and L2 cache « write back » and « invalidate » must be called.

**Up layer memory request:** write back if modifications

**Low layer memory request:** invalidate if modifications.
## Data Alignment and Performance

- Caches have a strong effect on memory latency

<table>
<thead>
<tr>
<th></th>
<th>L1 Cache</th>
<th>L2 Cache</th>
<th>XMC Prefetch</th>
<th>Single Read</th>
<th>Single Read</th>
<th>Burst Read</th>
<th>Burst Read</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No Victim</td>
<td>Victim</td>
<td>No Victim</td>
<td>Victim</td>
</tr>
<tr>
<td>All</td>
<td>Hit</td>
<td>NA</td>
<td>NA</td>
<td>0</td>
<td>NA</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>Local L2</td>
<td>Miss</td>
<td>NA</td>
<td>NA</td>
<td>7</td>
<td>7</td>
<td>3.5</td>
<td>10</td>
</tr>
<tr>
<td>MSMC (SL2)</td>
<td>Miss</td>
<td>NA</td>
<td>Hit</td>
<td>7.5</td>
<td>7.5</td>
<td>7.4</td>
<td>11</td>
</tr>
<tr>
<td>MSMC (SL2)</td>
<td>Miss</td>
<td>NA</td>
<td>Miss</td>
<td>19.8</td>
<td>20.1</td>
<td>9.5</td>
<td>11.6</td>
</tr>
<tr>
<td>MSMC (SL3)</td>
<td>Miss</td>
<td>Hit</td>
<td>NA</td>
<td>9</td>
<td>9</td>
<td>4.5</td>
<td>4.5</td>
</tr>
<tr>
<td>MSMC (SL3)</td>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>10.6</td>
<td>15.6</td>
<td>9.7</td>
<td>129.6</td>
</tr>
<tr>
<td>MSMC (SL3)</td>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
<td>22</td>
<td>28.1</td>
<td>11</td>
<td>129.7</td>
</tr>
<tr>
<td>DDR (SL2)</td>
<td>Miss</td>
<td>NA</td>
<td>Hit</td>
<td>9</td>
<td>9</td>
<td>23.2</td>
<td>59.8</td>
</tr>
<tr>
<td>DDR (SL2)</td>
<td>Miss</td>
<td>NA</td>
<td>Miss</td>
<td>84</td>
<td>113.6</td>
<td>41.5</td>
<td>113</td>
</tr>
<tr>
<td>DDR (SL3)</td>
<td>Miss</td>
<td>Hit</td>
<td>NA</td>
<td>9</td>
<td>9</td>
<td>4.5</td>
<td>4.5</td>
</tr>
<tr>
<td>DDR (SL3)</td>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>12.3</td>
<td>59.8</td>
<td>30.7</td>
<td>287</td>
</tr>
<tr>
<td>DDR (SL3)</td>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
<td>89</td>
<td>123.8</td>
<td>43.2</td>
<td>183</td>
</tr>
</tbody>
</table>

Source: sprabh2a TI
Signal Input/Output

- **Four Lanes of SRIO 2.1**
  1.24 to 5 GBaud Operation Supported Per Lane → up to 20 Gbauds

- **PCIe Gen2**
  Single port supporting 1 or 2 lanes
  Supports Up To 5 GBaud Per Lane → up to 10 Gbauds

- **HyperLink**
  Supports Connections to Other KeyStone → up to 50 Gbauds
  Architecture Devices Providing Resource Scalability

- **Gigabit Ethernet (GbE) Switch Subsystem**
  Two SGMII Ports
  Supports 10/100/1000 Mbps operation → up to 2 Gbps

- **Other ports**
  UART Interface, I2C Interface, 16 GPIO Pins, SPI Interface

- **Remark**
  Uncompressed 1920x1080 4:2:0 video @ 60Hz = 1.5 Gbps
Code Composer Studio Software

- **Code Composer Studio v5 (CCS) IDE**
  - Based on Eclipse 3.7 Indigo
  - Runs under Windows and Linux
  - Integrable in an existing Eclipse

- **C66x compiler, linker, assembler, simulator…**
  - Delivered with CCS IDE

- **EVM Drivers**
  - Installed with CCS
  - Connect to the EVM JTAG (breakpoints…)
Using DMAs

Principal program (CPU)  DMA Handling (CPU)  transfert (DMA)

Computing data1
Data1 ready

Configuring DMA for data 1 (Chip Support Lib)

Computing data2
Data2 ready

Configuring DMA for data2

Transmit Data1
End of transmission

Event / Interruption
SYS/BIOS
• **Multi-task OS:**
  enables sharing the DSP between several tasks

• **OS with Static and Dynamic configuration**
  Static : « configuration tool », .cfgfile
  Dynamic : specific functions to access interruptions, task creation, logs...

• **Gives many informations on the system for debug**

• **Preemptive RTOS**
  Scheduler
  task priorities

• **Alternative to DSP-BIOS:**
  Enea Solutions : other RTOS for C6x
DSP BIOS Limits

Memory cost

→ Modular but each module has a non negligible cost.

Mono-core system

→ No multicore OS

Time cost:

- system calls
- Interruption calls
Developed at IETR under Eclipse
From a dataflow graph to a multicore code execution
Automates multicore communication/synchronization
Multi-core DSP Programming

Algorithm
High Level Description

Architecture
High Level Description

Simulator
+ Debugger
+ Profiler

Multi-core Compiler

Multi-core Program

Multi-core Runtime

ARM ARM Co-processor Co-processor
ARM ARM DSP DSP DSP DSP
DSP DSP DSP DSP DSP

Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
High Performance DSP Applications

- Overview
- Standization Processes
- MPEG HEVC
- 4G
High Performance DSP Applications : Overview

- **Embedded system applications & High Performance computing applications**
  - Base stations & software-defined radio
  - Image and Audio processing
  - Industrial control systems
  - Aeronautics & transports
  - Radar / Sonar
  - Medical
  - Scientific computing & numerical simulation
    - High Performance Computing (HPC)
  - ...

...
High Performance DSP Applications: Overview

- **Embedded system applications & High Performance computing applications**

<table>
<thead>
<tr>
<th>Key Feature</th>
<th>Digital Media Applications</th>
<th>OMAP Applications Processors</th>
<th>C6000 Digital Signal Processors</th>
<th>C5000 Digital Signal Processors</th>
<th>C2000 Microcontrollers</th>
<th>MSP430 Microcontrollers</th>
<th>Stellaris 32-Bit ARM Cortex-M3 MCUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Automotive</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Communications</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Industrial</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Medical</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Security</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Video</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Wireless</td>
<td>Yellow</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td><strong>Source:</strong> TI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
High Performance DSP Applications : Overview

• **Typical types of operations / tasks / actors**
  - low-pass, band-pass, high-pass and adaptive filtering (FIR and IIR filters)
  - cross/auto, linear/circular correlation (similarity between signals)
  - Convolution (equivalent to multiplication in Fourier domain)
  - transformations between domains (Fast Fourier, DCT, Hadamard, wavelet, Hilbert, Wigner-Ville...)
  - noise removal
  - power computation
  - independent component analysis
  - expected signal detection and extraction
  - data prediction (temporal, spatial)
  - entropy coding
  - complex, vector and matrix operations
  - forward error correction
  - ...
Standardization Processes

• There are many standardization organizations
• Famous standardization organizations regarding signal processing include:
  ISO (International Organization for Standardization)
  IEEE (Institute of Electrical and Electronics Engineers)
  ITU (International Telecommunication Union)
  3GPP (Third Generation Partnership Project)

• MPEG HEVC Video Compression Standard
  developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC JTC1 Moving Picture Experts Group (MPEG)

• 3GPP LTE Radio Telecommunication Standard
  developed by the 3GPP (3rd Generation Partnership Project)
  Respecting (partially) the ITU-R organization IMT-Advanced specification
DSP Application: MPEG AVC and HEVC
MPEG Standards

- **Mpeg-1**: (1992)
- **Mpeg-2**: (1994)
- **Mpeg-4**: (since 1998)
  
  Example: Mpeg-4 Part 2 (DivX until v5, Xvid)
  
  Extension 1: Mpeg-4 part 10 = H.264 (ITU-T) = AVC (Advanced Video Coding)

- **Each standard**: better compression (HEVC: HD@4Mb/s)
Advanced Video Coding Decoder

Bitstream

VLC

VLC environment

Macroblock Generation

Bitstream processing

MB data

MB Image processing

Intra Prediction

Current Decoding Slice

Inter Prediction

Decoded Picture Buffer

Sample Reconstruction

Motion Vectors Reconstruction

Motion Vectors Neighborhood

Inverse Transform

Dequantize

DC Reconstruction

Rescale

Reconstructed Frame

Deblocking Filter

Reconstructed Frame
AVC Profiles

Extended Profile (so-called streaming profile)

Baseline (low latency)

Main/High Profile

AVC Baseline
- Low Delay, Lower Processor Load

AVC Main/High
- Supports Interlaced video, B-Frames

Predictions

Entropy coding

Buffer management

SI / SP slices

Data partitioning

I slices

P slices

Bslices (bidir)

Interlace

CAVLC

Cabac

Slice Groups

Redundant slices

ASO Arbitrary slice ordering

FMO Flexible Macrobloc Ordering

Multicore DSP
DSP Application : 4G
3GPP Standards

- **1G**
  - 1980
  - 10kbps

- **2G**
  - 1990
  - 100kbps
  - 2000
  - 1Mbps

- **3G**
  - 2010
  - 10Mbps

- **4G**
  - LTE Advanced

- **UMTS**
  - HSPA
  - HSPA+

...
LTE Access Network

- Preamble Detection
- Uplink Decoding
- Downlink Encoding

Core Network
3GPP Long Term Evolution rel. 9

- Up to **100 Mbps** in downlink, 50 Mbps in uplink
- Frequency Division Multiplexing, Multiple Input Multiple Output
- Dozens of Users communicating concurrently
- User allocation modified every millisecond
- Strong latency constraints

→ Complex physical layer
Frequency Allocations

- 3GPP LTE Frequency Allocations in France (ACERP October 2011)
  - 800 MHz band (previously UHF TV bands)
  - 2.6 GHz band

### LTE Frequency division: subcarriers

**Spectrum flexibility**
- In both downlink and uplink

<table>
<thead>
<tr>
<th>Bandwidth (MHz)</th>
<th>1.4</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFDM FFT size</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>1536</td>
<td>2048</td>
</tr>
<tr>
<td>Number of available PRBs (downlink)</td>
<td>6</td>
<td>12</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Number of available subcarriers</td>
<td>72</td>
<td>144</td>
<td>300</td>
<td>600</td>
<td>900</td>
<td>1200</td>
</tr>
</tbody>
</table>

Size (in complex values) of one symbol
3GPP LTE Rel. 9 Performance

- **High Data Rates**
  50Mbps(UpLink), 100Mbps (DownLink)

- **High User Equipment Speed**
  Walking to bullet-train (optimized up to 120 km/h)

- **Reduced Latency**
  Quick response time (under 5ms)

- **Cheap Roll-out**
  Bandwidth flexibility

- **Optimized for packet-switching**
  Good support for VoIP and data

- **Up to 100km radius cells (35km for GSM macrocells)**

- **Up to 100 user per cell**

- **Free to consult: search 36.211 and 36.212 in Google**
LTE and OSI Layers

1. Physical Layer
   - RS-232, 802.11a/b/g/n...
   - connection, contention resolution, modulation, channel adaptation, physical signal manipulation, bit flow generation...

2. Data Link Layer
   - Ethernet, PPP...
   - data preparation for physical layer...

3. Network Layer
   - IP...
   - transparent transfers between end-users, packet flow, error correction...

4. Transport Layer
   - TCP, UDP, SSL, TLS...
   - connection-less transfers, variable-length data sequences, packet fragmentation, logical addressing...

5. Session Layer
   - SSH...
   - physical layer control, error correction, point-to-point and point-to-multipoint control, data preparation for physical layer...

6. Presentation Layer
   - ASCII...
   - connection-less transfers, variable-length data sequences, packet fragmentation, logical addressing...

7. Application Layer
   - FTP, HTTP, SMTP, Telnet...
   - transparent transfers between end-users, packet flow, error correction...
Downlink/Uplink in a Base Station

- Application is naturally described by a dataflow graph
- This application requires high DSP computing power
LTE eNodeB DSP Programming
Static versus Variable Algorithms

- Active Users
- Downlink Data per user
- Uplink Data per user
- Downlink Encoding Load
- Uplink Decoding Load
- Preamble Detection Load

Time (ms)
Conclusion from Applications Part

- **Applications are complex!**
  A designer should not need to be an expert in both application and architecture
  Legacy code reuse between systems is absolutely needed
  When a programmer has generated a functional efficient piece of code, he does want to reuse it
  A designer should not need to tweak his code for his target architecture

- **Streaming applications are naturally broken down into dataflow actors**
  When we analyze an application, it is natural to use dataflow graphs
Languages and Models of Computation (MoCs) for Application Description

• Languages and MoCs

• Dataflow MoCs and Languages
  Focus on PiMM and πSDF
  Focus on CAL

• Practical Work Setup
• Among the 10 most used languages, all 10 are imperative, 7 are object-oriented. They share semantics.
• 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
Models of Computation for DSP

- **Useful for**
  - Specification (especially for standards (cf. MPEG RVC))
  - Simulation (performance metrics and constraints checking)
  - Execution (functional description)

- **Families of MoCs:**

- **Finite State Machine MoCs**
  - MoC of imperative languages (C/C++/Java…)
  - States and transitions based on conditions
  - Computation is executed on transitions
  - Representing the behavior of a Turing machine
Models of Computation for DSP

• **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**
  No state, a program returns the result of composed mathematical functions: result = f(g(h(inputs))
  Based on lambda calculus
  Haskell, Caml, Scheme, XSLT
  Functional languages are examples of declarative languages
Models of Computation for DSP

- **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…
Models of Computation for DSP

• **Process Network MoCs**

  concurrent and independent modules (processes) communicate ordered tokens (data quanta) through First-InFirst-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

  ...

• **Any syntax can be used to express these semantics**
Kahn Process Networks

- Computation is done by processes (= Actors)
- Actors communicate only though data infinite FIFOs
  Actors do not share a state and have no internal state
- FIFO reading is blocked until a data arrives
Dataflow Process Networks

- **DPN is a special case of Kahn Process Networks defining:**
  
  Firing rules: conditions on which an actor is ready to execute
  Actor « invocation » (=« firing »)

  Example of firing rule for Set:
  - Set consumes 3 tokens coming from Do and fires an action Set1, producing 1 token
  - Set consumes 1 token on Get and one token on Do and fires action Set2 producing 2 tokens
There Exist Many Dataflow MoCs

- **Differences**
  - Is the number of exchanged tokens fixed/variable?
  - Is it even specified?
  - Does it depend of parameters?
  - Is there an external control flow?
  - Are there actor states

- **Devil is in the details**
  - Dynamic Dataflow (DDF) is not a DPN because it can « peek » FIFOs (look inside The FIFO without popping the token)
Expressiveness
How many different behavior types can the model specify?
How « optimized » is the specification?
How concise is the specification?

Predictability
Can the behavior (production and consumption) be predicted?
at compile-time
in advance at runtime

There Exist Many Dataflow MoCs
There Exist Many Dataflow MoCs

- The dataflow MoC defines a coordination language
- Actors are implemented by a host language
  - Any host language can be used
  - As long as the host language implements the firing rules
  - We can combine and make communicate
    - IPs coded in VHDL
    - High-level software actors written in Java
    - Low-level software actors written in C
Which MoC should we use to program Multicore DSPs?

- **Small grain, Locally to a core**
  Imperative languages such as C/C++ have shown their capacity to use cores efficiently, including with low-level parallelism (SIMD, VLIW…)
  Finite state machine MoC

- **Coarse grain, to combine the cores**
  Dataflow MoCs are good candidates because they decouple actors
  All possible communications between actors are specified, so they can be used to organize data flows between cores
Which Dataflow Model for a Given Application?

3GPP LTE
For example:

- Preamble Detection
- Uplink Decoding
- Downlink Encoding

Expressiveness

- DDF
- KPN
- BDF
- πSDF
- PSDF
- IBSDF
- CSDF
- SDF
- DAG

Predictability
Synchronous Dataflow

- Actors and Data Ports
- FIFO Queues and Delays

SDF Predictability

• By resolving the topology equation, we can check consistency

• By verifying that enough initial tokens have been set, we can check schedulability

Ability to come back to the initial FIFO states

1) Topology Matrix of rank nb actors - 1 => consistency

\[
\begin{pmatrix}
3 & 2 & 0 & 0 \\
3 & 0 & 2 & 0 \\
0 & 2 & 0 & -4 \\
0 & 0 & 2 & -4 \\
\end{pmatrix}
\]

\[ q = 0 \iff \begin{pmatrix}
3 & 2 & 0 & 0 \\
0 & 2 & 2 & 0 \\
0 & 0 & 2 & -4 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

\[ q = 0 \iff q = \begin{pmatrix}
4.x \\
6.x \\
6.x \\
3.x \\
\end{pmatrix}
\]

2) A valid schedule, respecting initial tokens exists.

(4A)(6B)(6C)(3D) => The graph is back to original state

=> The SDF graph is schedulable

Equivalent single rate SDF Graph
Graph Scheduling and Transformation

Synchronous Dataflow (SDF) graph:

Schedule: A (4 B) (3 C)

Single rate graph
Graph Scheduling and Transformation

Cyclo Static Dataflow (CSDF) graph:

Schedule: A (3 B) (3 C)

Single rate
Graph = SDF!!
PiMM and \( \pi \)SDF

- Meta-model developed at IETR
- in collaboration with UMD and TI

- Targeted Dataflow Model of Computation
  - becomes:
    - Hierarchical & Compositional
    - Statically parameterizable
    - Dynamically reconfigurable

- PiMM fosters:
  - Predictability
  - Memory boundedness
  - Parallelism
  - Lightweight runtime overhead
  - Developer-friendliness
A Naive SDF Composition Mechanism

- Hierarchical SDF

[Diagram showing the composition mechanism with nodes and edges labeled with numbers and symbols.]
PiMM Compositionality Mechanism

• Hierarchical Interfaces

Trade-off Between Expressiveness and Analyzability

- The more dynamism, the less predictability

---

Configuring Parameters

- Dynamic Parameters
- Configuration Actors
PiMM Operational Semantics

- PiMM (Parameterized and Interfaced dataflow Meta-Model)
  - Evolution of IBSDF and PSDF

- \( \pi \)SDF Example:
CAL and DDF

- **CAL** = **CAL Actor Language**, J. Ecker and J. Janneck
- **It implements**
  the Dynamic Dataflow (DDF) MoC
  and a host code named CAL specifying firing rules
- **DDF graph gives no information on token flow**
  Firing rules are specified in the CAL language
  consumed tokens, guards, finite state machine, priorities
CAL and DDF

- Via classification, a CAL actor can be transformed into SDF, CSDF, and soon $\pi$SDF
  
  In order to make the model more predictable

- The Orcc compiler, developed at IETR, is a compiler for CAL
CAL and DDF

- From CAL, both hardware and software can be generated
- CAL → Preesm is already working for static actors
- CAL → PiSDF is a future theme of research
Example: Describing 3GPP LTE Base Station Physical Layer
Preamble Detection

X Reception Antennas
X Preambles

Filter → DFT → Select preamble subcarriers

Correlate with root sequence → IDFT → Power

X Root Sequences
X Reception Antennas
X Preambles

X Number of root sequences

Noise floor estimation → Peak Search

Detected Users
Timing Advance
Uplink

X symbols (14)
X Reception Antennas

Prepare → FFT → Demap Control/Data → Estimate Channel → Equalize Multiple Antennas → Decode Control

X Reception Antennas

X Users
Symbol and Bit Processing

X Users
X Code Blocks
Code Block Processing

X Users
X Transport Blocks
Transport Block Processing

Data of Each user
Create Control Signals

Resource Mapping

X symbols (14)
X Transmission Antennas

IFFT

Prepare

Data of Each user

X Users
X Transport Blocks

Transport Block Processing

X Users
X Code Blocks

Code Block Processing

X Users

Bit and Symbol Processing

88: Multicore DSP
Applications are naturally broken down into dataflow actors

When we analyze an application, it is natural to use dataflow graphs

Many models exist with different semantics

Dataflow models express parallelism in algorithms

Syntax ! Semantics

Semantics matter more than syntax
Multi-core DSP Programming

**Algorithm High Level Description**

**Architecture High Level Description**

**Simulator**
- + Debugger
- + Profiler

**Multi-core Compiler**

**Multi-core Program**

**Multi-core Runtime**

- ARM
- ARM
- Co-processor
- Co-processor
- ARM
- ARM
- DSP
- DSP
- DSP
- DSP
- DSP
- DSP
- DSP
- DSP
- DSP

Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
High Performance DSP Architectures

- Types of embedded processors
- DSP vs. GPP
- Multicore DSP architectures
Von Neumann Architecture

DSP : digital signal processors != GPP : General purpose processors
→ processors optimized and adapted to signal processing
Processor Generalities

• **Processor types:**
  - Microcontroller
  - Digital Signal Processors (DSP)
  - General purpose processors (GPP)
  - GP-GPU (General-Purpose Processing on Graphics Processing Units)

• **Choice factors:**
  - Price (architecture complexity, production technology)
  - Power consumption
  - CPU Performances
  - I/O performances
  - Memory capacity (data/code)

---

C2000
- **price**
- Control I/O
- Engine control

C5000
- **Efficiency**
- Watt / price / size
- Phones
- Modem
- Camera

C6000
- **Performance**
- Complex applications
- Image
- Video
### Applications and Processor Types

<table>
<thead>
<tr>
<th>TI Embedded Processors</th>
<th>Microcontrollers (MCUs)</th>
<th>ARM®-Based Processors</th>
<th>Digital Signal Processors (DSPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>16-bit ultra-low power MCUs</strong></td>
<td><strong>32-bit real-time MCUs</strong></td>
<td><strong>32-bit ARM Cortex™-M3 MCUs</strong></td>
<td><strong>ARM Cortex-A8 &amp; ARM9™ MPUs</strong></td>
</tr>
<tr>
<td>MSP430™</td>
<td>C2000™ Delfino™ Piccolo™</td>
<td>Stellaris® ARM Cortex-M3</td>
<td>Sitara™ ARM Cortex-A8 &amp; ARM9</td>
</tr>
<tr>
<td>Up to 25 MHz</td>
<td>40MHz to 300 MHz</td>
<td>Up to 100 MHz</td>
<td>375MHz to &gt;1GHz</td>
</tr>
<tr>
<td>Flash</td>
<td>Flash, RAM</td>
<td>Flash, RAM</td>
<td>Cache, RAM, ROM</td>
</tr>
<tr>
<td>1 KB to 256 KB</td>
<td>16 KB to 512 KB</td>
<td>8 KB to 256 KB</td>
<td>Cache, RAM, ROM</td>
</tr>
<tr>
<td>Analog I/O, ADC, LCD, USB, RF</td>
<td>PWM, ADC, CAN, SPI, I²C</td>
<td>USB, ENET, MAC+PHY CAN, ADC, PWM, SPI</td>
<td>USB, ENET, CAN, SPI, PCIe, EMAC</td>
</tr>
<tr>
<td>$0.25 to $9.00</td>
<td>$1.50 to $20.00</td>
<td>$1.00 to $8.00</td>
<td>$5.00 to $25.00</td>
</tr>
</tbody>
</table>

**DSPs**

- **DSP DSP+ARM**
- **Multi-core DSP**
- **Ultra Low power DSP**

**C6000™**

- **320 GMACS**
- **OMAP™**
- **DaVinci™ video processors**
- **300MHz to >1Ghz +Accelerator**
- **Cache RAM, ROM**
- **USB, ENET, CAN, SPI, PCIe, EMAC**
- **Industrial automation, POS & portable data terminals**
- **Floating/Fixed Point Video, Audio, Voice, Security, Conferencing**
- **$5.00 to $200.00**
- **$40.00 to $200.00**
- **$3.00 to $10.00**

**C5000™**

- **Up to 300 MHz +Accelerator**
- **Up to 320KB RAM**
- **Up to 128KB ROM**
- **USB, ADC, McBSP, SPI, I²C**
- **Audio, Voice**
- **Medical, Biometrics**

*Multicore DSP*
## TI Processors and Applications

<table>
<thead>
<tr>
<th>Key Feature</th>
<th>Digital Media Processors</th>
<th>OMAP Applications Processors</th>
<th>C6000 Digital Signal Processors</th>
<th>C5000 Digital Signal Processors</th>
<th>C2000 Microcontrollers</th>
<th>MSP430 Microcontrollers</th>
<th>Stellaris 32-Bit ARM Cortex-M3 MCUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automotive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medical</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wireless</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete tailored video solution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low power and high performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power-efficient performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance, integration for greener industrial applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ultra-low power</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open architecture software, rich communications options</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Source:** TI
Low cost System-on chip

Locosto (TCS2305)

Very cheap
65-nm technology

ARM7 CPU + c54x
Minimal functionalities
Video Processing DSP

Da Vinci TMS320DM644x

1 DSP tms320c64x+ (720MHz max)  Video I/O for screen
1 ARM9
1 Video Accelerator
Advanced System-on chip

OMAP™ 4 Platform: OMAP4430/OMAP4440

High power of computation
Still low power consumption
Interfaces to modern peripherals

2 General purpose ARM Cortex A9 cores (1GHz)
IVA Supporting 1080p video encoding/decoding
3D graphics accelerator
No DSP!!!
DSPs vs. GPPs

- Optimization of the compilation and instruction set for signal processing
- Reduced Power consumption in DSPs
- Memory Management Unit (MMU) in the general purpose processor:
  - In DSPs, any memory is accessed by addresses: registers, stack, heap, OS memory...
  - Advanced OS (like Linux) need pagination: a virtual memory space in pages
  - The MMU converts the virtual addresses into the physical addresses of the hardware
  - The MMU can protect memory spaces from unwanted accesses
- DSPs use Real-Time OS: the c66x has SYS-BIOS
- Kalray VLIW core is an exception: a DSP with MMU
Power Dissipation

• Less than 2W: no specific dissipation problem

• 2 to 7W: heat sink and hot spot management

• 7W+: heat sink, fan and complex hot spot management
Why Go Multicore?

• **Frequency increase → power consumption → heat**
  heat → need for cooling, more faults, reduced longevity

  Dynamic Power = $Cte \times \text{capacity} \times \text{voltage}^2 \times \text{frequency}$
  But augmenting frequency augments leakage so voltage must be higher

  Frequency $\times 1.5 \rightarrow \text{Power} \times 2$ on Freescale MPC8641
  Cores $\times 2 \rightarrow \text{Power} \times 1.3$ on Freescale MPC8641

  Moreover, augmenting frequency may require longer pipeline

• **To increase MIPS and limit Watts**
  Increase number of core instead than frequency

Source: Freescale « Embedded Multicore: An Introduction»
Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
Trade-off between Flexibility and Energy Efficiency

$E_E : \text{Efficiency : MIPS} / \text{Watt}$

- **Multicore DSP**
  - TI C6678
  - $\sim 20$ MMAC / mW
- **Embedded Processor**
  - Pleiades
  - 10-50 MOPS/mW
- **Reconfigurable Processor**
  - SA110
  - 0.4 MIPS/mW
- **ASIC**
  - 100-1000 MOPS/mW
- **Embedded FPGA**
- **DSP**
  - $2V$ DSP
  - 3 MOPS/mW
- **ASIP**
  - Alpha
  - 0.007 MIPS/mW

Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
Why Go Heterogeneous?

- **3 sources of heterogeneity**
  - Non uniform cores implementing different instruction sets
  - Combining software cores with hardware IPs
  - Non uniform communication performance

- **Heterogeneity improves performance**
  - Repetitive and costly actors can be efficiently computed by hardware logic (ASIC)
  - Actors with some control and reconfiguration needs are suited for DSPs
  - Control tasks with many conditions are suited to be run on GPP

Source: Freescale « Embedded Multicore: An Introduction »

Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
Towards multicore DSPs

TCI6488: Tri-core Telecommunication oriented DSP

- Up to 1GHz/24Gips
- Each core is programmed independently
- 3MB L2 memory
- RSA: specific instruction set for CDMA operation (3G oriented)
- I/O driven by EDMA

Source: TI
TMS320TCI6488 (2007)
Multicore DSP

TMS320C6678 – Keystone I (2011)
New c66x core: floating point arithmetic

Representing real numbers

Floating-point arithmetics: Sign - exponent - mantissa

\[ (-1)^s \cdot (1,m) \cdot 2^e \]

Fixed-point arithmetics: Sign – integer part – fractional part

\[ (-1)^s \cdot e,f \]
TMS320C6678 – Keystone I (2011)

TCI6678: Octo-core DSP
→ Up to 1.25GHz/320 GMACs/160 GFlops
→ Each core is programmed independently
→ I/O driven by Multi-core Navigator to automate transfers between cores
→ Fixed and floating-point ALUs

Core c66x = Arithmetic and logic units (L & S), Data management units (D), Multiplication units (M)

Source: TI
TI TMS320TCI6636 – Keystone II (2013)

Multicore Navigator

Layer 1, 2, 3, Transport Acceleration Pacs
- TCP3 ×2
- VCP2 ×4
- TAC2
- RAC1 ×2
- FFTC ×4
- BCP

System Elements
- DevSecurity
- EDMA
- Power Management
- Debug Trace

EMIF and Low-Speed I/O
- 72b DDR3
- GPIO ×32
- UART ×2
- SPI ×3
- I²C ×3
- USB 3.0
- USIM
- EMIF 16

High-Speed SerDes
- 4 × 1 GbE 4-Port Switch
- SRI0
- 2 × HyperLink
- PCIe

Source: TI
TI TMS320TCI6636 – Keystone II (2013)

• **8 C66x @ 1.2GHz**
  38.4 GMacs/Core for Fixed Point
  19.2 GFlops/Core for Floating Point

• **Memory**
  32KB L1P Per Core
  32KB L1D Per Core
  1MB Local L2 Per Core
  6 MB MSM SRAM Memory Shared by 8 DSPs

• **ARM Cortex A15 Quad Core Cluster @ 1.2GHz**
  32KB L1I Per Core
  32KB L1D Per Core
  4MB L2 Cache Memory Shared by Quad Core
  AMBA 4.0 AXI Coherency Extension Master Port connected to MSMC

• **Multicore Navigator**
  16k Multi-purpose Hardware Queues with hardware queue manager
  Packet-Based DMA for Zero-Overhead Transfers
Freescale MSC8144 Starcore DSP

Source: Freescale
Shared memory costs much energy
Putting a NoC at high level instead

**Source:** Kalray

Multicore DSP
Levels of parallelism in Multicore DSP Architectures
Architecture Levels of Parallelism

Core (MPSoc)

Instruction (ILP, VLIW)

Data (SIMD)

Application

Task / Actor

Operation
DSP Evolution

Data level parallelism

- MAC
- Multi-MAC
- VLIW
- Multi Pro. SoCs

Instruction level parallelism

- Bit level parallelism
- Instruction level parallelism
- Thread level parallelism

Conventional architecture

First generation
- TMS320C20 (5 MIPS)
- TMS320C10 (2.5 MIPS)

Second generation
- DSP56001 (13 MIPS)
- TMS320C54x (100 MIPS)
- ADSP2183 (50 MIPS)

Third generation
- Improved conventional architecture
- Heterogeneous multicores with HW accelerators
- OMAPx (ARM+C64)
- SC8144 (4 cores)
- TMS320C62x
- TigerSharc

Fourth generation
- VLIW
- SWP
- TMS320C66x (8 cores)
- MSC815x (6 cores)
- OMAPx (ARM+C64)
- SC8144 (4 cores)
- TMS320C55x BlackFin

Fifth generation
- Multi-core VLIW SWP
- MultiPro SoCs

Sixth generation
- Many Cores
- Heterogeneous multicores with HW accelerators
- OMAPx (ARM+C64)
- SC8144 (4 cores)
- TMS320C55x BlackFin

High performance DSP

- OMAPx (ARM+C64)
- SC8144 (4 cores)
- TMS320C55x BlackFin

Low power DSP

- OMAPx (ARM+C64)
- SC8144 (4 cores)
- TMS320C55x BlackFin

Normalized performance

Single Instruction Multiple Data

• **Splitting ALU into subparts**
  Example: 2 16-bit additions on 1 32-bit adder

![Diagram showing two 16-bit additions feeding into a 32-bit adder](image)
Very Long Instruction Word

• **VLIW characteristic**
  
  Architecture made-up of parallel FU
  Several instructions per cycle embedded in a macro-instruction
  Homogeneous architecture: more orthogonal
    - Close to RISC processor
  Uniform register file made-up of several registers
  Load-Store architecture
VLIW Example : C64x

Fetch

32x8=256 bits

Dispatch Unit

L: ALU
S: Shift+ALU
M: Multiplier
D: Address U

Functional Unit .L1
Functional Unit .S1
Functional Unit .M1
Functional Unit .D1
Functional Unit .D2
Functional Unit .M2
Functional Unit .S2
Functional Unit .L2

Register File A

Register File B

Data Memory Controller

Data Address 1

Data Address 2

Internal Memory

Multicore DSP
WLIV Compiler: Loop unrolling

For\(i=0; i<N; i++\)
\{
  ACC = ACC + x[i].h[i]
\}

For\(i=0; i<N; i+=3\)
\{
  ACC = ACC + x[i].h[i]
  ACC = ACC + x[i+1].h[i+1]
  ACC = ACC + x[i+2].h[i+2]
\}
WLIV Compiler: Software pipelining

- Improving the throughput at the cost of latency and memory
Convolution C code for VLIW and SIMD

\[ Y(k) = \sum_{l=0}^{N-1} Z(l)C((l-k) \mod N) \]

void fir_circbuf_modified(
    const RACH_cpx *restrict vectorb,
    const RACH_cpx *restrict vectora,
    RACH_cpx *restrict vecorc,
    short size,
    int type
)
{
    int i, j, xindex, imag, real, h1h0, x1x0;
    const short *restrict g = ((short*) vectorb);
    const short *restrict h = (short*) vectora;
    short *restrict r = (short*) vecorc;
    size = 2*size;
    #pragma MUST_ITERATE(4, 4);
    #pragma UNROLL(4);
    for (i = 0; i < size; i += 2)
    {
        imag = 0;
        real = 0;
        #pragma MUST_ITERATE(2, 2)
        for (j = 0; j < size; j += 4)
        {
            xindex = j - i;
            xindex = xindex+size*(xindex<0);
            h1h0 = _mem4((void*)&g[j+0]);
            x1x0 = _mem4((void*)&x[xindex]);
            real += _dotp2(h1h0,x1x0);
            imag += _dotp2(h1h0, _pack1h2(x1x0,x1x0));
            xindex = j - i + 2;
            xindex = xindex+size*(xindex<0);
            h1h0 = _mem4((void*)&g[j+2]);
            x1x0 = _mem4((void*)&x[xindex]);
            real += _dotp2(h1h0,x1x0);
            imag += _dotp2(h1h0, _pack1h2(x1x0,x1x0));
        }
    }
    r[i ] = (imag >> 15);
    r[i+1] = (real >> 15);
}
Core-Level Parallelism
Inter-Processor Communication and Architecture Models
Types of Inter-Processor Communications

- **Transmitting one token via inter-processor communication or shared memory**

- **Direct Signaling**
  
  Interrupting a core from another core to either push or pull data

- **Indirect Signaling**
  
  Delegating the transmission to a DMA (Direct Memory Access) element

- **Atomic arbitration**
  
  Via hardware semaphores in case of shared memory
Direct Signaling Communication

**demand-driven**

- Distributed or shared memory
- In demand-driven case, first interrupt can be avoided if core A is constantly demanding data and core B cannot erase data before it is consumed (or if data can be discarded)

**data-driven**

- Inter-Processor communication can be ethernet, SRIO...
Indirect Signaling Communication: delegating to a DMA

- **Distributed or shared memory**
  DMA must be able to access the whole memory space
  DMA is another master on the memory bus
  Cores A and B are free to compute during DMA transfer
Protecting shared memory accesses by semaphores

- **1-place FIFO on Shared memory**
  Possibility of N-places FIFO with a round buffer and read and write indexes
  2-place FIFO = « ping-pong » buffer

- **More than a mutex**
  The 2 semaphores ensure alternate accesses from cores A and B
Communication Speed

• On Keystone I, data transfers have a speed of about
  L2 access: 5.3 GB/s
  DDR Access: 2.6 GB/s
  MSMC Access: 8 GB/s

  For comparison: Raw HD video 1080p60 4:2:0 = 0.19 GB/s

• Inter-processor communication libs mask the complexity
Modeling Architectures
Models of Architecture

- **SystemC**
  C++ templates and libraries used to simulate hardware modules

- **AADL**
  Separating hardware and software but specifying both and oriented towards threads and processes

- **IP-XACT**
  More a syntax for serialization than a real model

- **Custom models**
  In MAPS compiler, in SynDEx rapid prototyping tool, in PREESM…
Models of Architecture

- Often oriented towards hardware design debug
- Often custom and with no precise semantics
- Often no real separation between application and hardware
System-Level Architecture Model

Processing Element

Communication Nodes
- Parallel Node
- Contention Node

Communication Enablers
- RAM
- DMA

Directed Data Link
Undirected Data Link
Set-up Link
S-LAM Examples

RAM

core1

CN

1 Gbit/s

core2

DMA

core3
S-LAM of a Board with 2 TCI6488

DSP 1

DMA

SCR

RIO

1 Gbit/s

2 GB/s

core1

core2

core3

TCP2

VCP2

DSP 2

DMA

SCR

RIO

1 Gbit/s

2 GB/s

core1

core2

core3

TCP2

VCP2

2 GB/s

1 Gbit/s
Conclusion from Architecture Part

• **Architectures become more and more complex to program**
  - Parallelism rises at instruction and task level
  - Heterogeneity rises
  - Performance necessitates a correct use of DMA, cache, IPC
Multi-core DSP Programming

- Algorithm High Level Description
- Architecture High Level Description
- Simulator + Debugger + Profiler
- Multi-core Compiler
- Multi-core Program
- Multi-core Runtime

Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
Automatic porting of Multicore DSP Applications

- Parallelism Laws
- Multicore Scheduling
- Multicore Tools

Overview

IETR Tools
Parallelism Laws
Amdahl’s Law

• **Developed in 1967 by Gene Amdahl**

• **It gives a generic performance metric for applications**

  Simplifying problem assumption: \( x\% \) of the code is sequential, the rest is perfectly parallel.

  With \( 5\% \) of sequential code, speedup is limited to 7.5 on 12 cores.

  Speedup refers to the acceleration brought by adding cores.

```
1 thread

As many threads as we want
```

```
5% sequential

95% perfectly parallel
```
Amdahl’s Law

• For different parallel section percentages

  Simplifying problem assumption: \( x\% \) of the code is sequential, the rest is perfectly parallel

  With 50% of sequential code, speedup is limited to 1.84 on 12 cores
Amdahl’s Law

• Amdahl’s law has brought many doubts on multicores
  It does not take into account inter-process communication that worsens the speedup

• Why add more cores if the parallelism of applications limits speedups so much?
Gustavson’s Law

• Developed by John Gustavson in 1988

• With a different hypothesis, Gustafson has shown the limits of Amdahl’s law

Hypothesis: more cores imply more parallelism, the sequential section stays the same percentage of execution latency regardless the number of cores

in Amdahl’s law, the percentage tends to 100% because the parallel section time reduces and the sequential section stays unmodified

With 5% of sequential code, speedup is limited to 11.89 on 12 cores
Gustavson’s Law (1988)

• With a different hypothesis, Gustafson has shown the limits of Amdahl’s law

Hypothesis: more cores imply more parallelism, the sequential section stays the same percentage of execution latency regardless the number of cores (in Amdahl’s law, it tends to 100%)
Dataflow Speedup

- The maximum speedup of dataflow execution can be computed.
  It is limited by the critical path length.

Example: ignoring communication times
Dataflow Speedup

• The maximum speedup of dataflow execution can be computed

  It is limited by the critical path length

Example: ignoring communication times

  critical path length = 1 + 5 + 3 + 2 + 1 = 11ms
  work = 19 ms
  max speedup = 19 / 11 = 1.72
Preesm Speedup Assessment Chart

- Algorithm Limited
- Architecture Limited
- Insufficient

- work limit
- critical path limit
- GST: « blind » sched.
- Current Deployment

Number of Cores vs Speedup graph.
• **Speedup Assessment Chart Limitations**
  
  The speedup assessment chart considers **only latency**
  
  → **No pipelining** is taken into account
  
  All cores are considered identical in the chart (main operator)
  All communications have the same speed (main communication node)

• **How to add speedup**
  
  Redescribe the application to find more parallelism
  Add initial tokens (delays) to pipeline
Parallelism in a dataflow graph

There are 3 types of parallelism: task, data and pipeline parallelism
Specific to stream processing applications

Data parallelism
• **Parallelism in a dataflow graph**

There are 3 types of parallelism: task, data and pipeline parallelism

Specific to stream processing applications

**Task parallelism**

![Diagram of task parallelism with nodes labeled Get Pic, Filter1 Pic, Filter2 Pic, and Combine filters, connected by arrows indicating data flow.](Image)
Task/Data/Pipeline Parallelism

- **Parallelism in a dataflow graph**
  
  There are 3 types of parallelism: task, data and pipeline parallelism.

  Specific to stream processing applications.

Pipeline parallelism

- **Get Pic**
- **Filter Luma**
- **Set Pic**

Get Pic → Filter Luma → Set Pic

Cores vs. time

- Get Pic
- Filter Luma
- Set Pic
Parallelism in a dataflow graph

Pipeline parallelism

Input pic 0

Latency

Output pic 0

Cores

time
Task/Data/Pipeline Parallelism

- **Parallelism in a dataflow graph**

Pipeline parallelism

- **Input** pic 0
  - Get Pic
  - Filter Luma
  - Set Pic

- **Input** pic 1
  - Get Pic
  - Filter Luma
  - Set Pic

- **Input** pic 2
  - Get Pic
  - Filter Luma
  - Set Pic

**Throughput**

- **Output** pic 0
- **Output** pic 1
- **Output** pic 2

**Cores**
Multicore Scheduling
Scheduling Strategies

Assignment

Ordering

Timing
## Scheduling Strategies

### More adaptivity

<table>
<thead>
<tr>
<th></th>
<th>Assignment</th>
<th>Ordering</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>fully dynamic</td>
<td>run</td>
<td>run</td>
<td>run</td>
</tr>
<tr>
<td>static-assignment</td>
<td>compile</td>
<td>run</td>
<td>run</td>
</tr>
<tr>
<td>self-timed</td>
<td>compile</td>
<td>compile</td>
<td>run</td>
</tr>
<tr>
<td>fully static</td>
<td>compile</td>
<td>compile</td>
<td>compile</td>
</tr>
</tbody>
</table>

EA Lee *Scheduling strategies for multiprocessor real-time DSP*

### More performance
## Scheduling Strategies

<table>
<thead>
<tr>
<th></th>
<th>assignment</th>
<th>ordering</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>fully dynamic</td>
<td>run</td>
<td>run</td>
<td>run</td>
</tr>
<tr>
<td>static-assignment</td>
<td>compile</td>
<td>run</td>
<td>run</td>
</tr>
<tr>
<td>self-timed</td>
<td>compile</td>
<td>compile</td>
<td>run</td>
</tr>
<tr>
<td>fully static</td>
<td>compile</td>
<td>compile</td>
<td>compile</td>
</tr>
</tbody>
</table>

EA Lee *Scheduling strategies for multiprocessor real-time DSP*

More adaptivity

More performance
Heterogeneous Multicore Scheduling

• **Assignment, ordering and timing**
  Part of « Operational Research »
  ➔ How to organize a company
  ➔ How to organize a project (Gantt chart…)
  ➔ How to take decisions in general

• **NP hard problem**
  the verification that a possible solution of the problem is valid can be computed in polynomial time (verifying that a schedule is valid)

  no polynomial time algorithm for NP-complete problems is known and it is likely that none exists.

  When the problem grows (for example, the number of cores or actors), solving it is becoming more complex exponentially.

M. R Garey and D. S. Johnson “Computers and Intractability”
Multicore scheduling is equivalent to quadratic assignment NP hard problem

N facilities, each pair of facilities \((f,g)\) associated to a flow of communication

N locations to put the facilities, each pair of locations \((l,m)\) associated to a distance

→ In which location (bijection) should we put each facility to minimize traffic (the sum of the distances multiplied by the corresponding flows)
Heterogeneous Multicore Scheduling

- **The real problem is more complex**
  
  M actors, N<M cores to put the facilities, each pair of locations (l,m) associated to a distance
  
  Heterogeneity: actors have different costs on different cores
  
  The objective function is not only communication minimization latency, throughput…
Heterogeneous Multicore Scheduling

• **The problems is solved by heuristics**
  
  Exhaustive methods are useless
  Heuristics explore only parts of the given problem

• **Many heuristics exist**
  
  list scheduling, greedy scheduling
  FAST scheduling (Y-K Kwok)
  Hybrid flow-shop scheduling (J. Boutellier)
  Meta-heuristics (genetic algorithms, ant colonies…)

• **It is not possible to predict the quality of the result of a heuristic**
  
  But models should contain enough information to take decisions
Heterogeneous Multicore Scheduling

- **List scheduling**
  Actors are scheduled in-order (topological order)
  The core that can finish actor execution first wins

```
A  C  B  D  E  F  G
1ms 5ms 1ms 3ms 2ms 2ms 5ms
```

Diagram:

```
A → C → B → D → E → F → G
```

Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
**List scheduling**

Actors are scheduled in-order (topological order)

The core that can finish actor execution first wins

---

Heterogeneous Multicore Scheduling

1. List scheduling
   - Actors are scheduled in-order (topological order)
   - The core that can finish actor execution first wins

---

Diagram:

- **A**, 1ms
- **B**, 1ms
- **C**, 5ms
- **D**, 3ms
- **E**, 5ms
- **F**, 2ms
- **G**, 2ms

---

**Cores**

- **A**
- **B**
- **C**
- **D**
- **E**
- **F**
- **G**

---

**Time**

---

Maxime Pelcat – mpelcat@insa-rennes.fr - June 2013
Heterogeneous Multicore Scheduling

- **Fast scheduling**
  Actors moved around to improve cost function (load balancing…)
  Critical path is more protected

![Diagram showing the scheduling process with cores and time]
Heterogeneous Multicore Scheduling

• Genetic algorithm

Several mapping solutions are kept, they undergo mutations and cross-overs and worst solutions are regularly removed.
Load balancing

• If no information is available on task / actor behavior
  parallelism is brought by balancing the loads
  Execution is managed by task/job/actor queuing

Unbalanced

Balanced
Execution Schemes

• **Load balancing without task behavior prediction**
  Implemented over multi-threading

Great freedom in thread creation

The shared task queue becomes the bottleneck

**Work-queueing**
(Apple Grand Central Dispatch, OpenMP)
Execution Schemes

- **Load balancing without task behavior prediction**
  
  Implemented over multi-threading

  One task queue per core:
  No more bottleneck

  Hard to predict performance
Execution Schemes

- Load balancing without task behavior prediction
  Implemented over multi-threading

One task queue per core:
No more bottleneck

Hard to predict performance
Execution Schemes

- **Scheduling with task behavior prediction**
  - No adaptivity to algorithm modifications

No decision overhead

Decentralized
(Preesm, SynDEx)

!![Diagram of execution schemes]

- Core 2 Thread/Core 1 Actor A
- Actor B
- Actor D
- Actor E
- Core 2 Actor C
- Core 2 Actor C
Execution Schemes

- **Scheduling with task behavior prediction**

  Adaptivity to algorithm variations

  Master core can become a bottleneck

Master/Slave (Compa Runtime)
• Why not use threads and processes instead of actors?

• Threads share memory within a process
  Multicore thread execution necessitates shared memory between cores
  Shared memory is increasingly costly when number of cores grows
  This method of parallelizing is showing its limits already with 8 cores

• Threads are designed for resource sharing
  Cores, memory…
  What we want is more resource combining

• Actors are special types of processes
  With firing rules, i.e. computation is triggered by available data
  Actors are dedicated to stream processing. Threads and processes can implement control-oriented code
Multicore Tools
Some Tools and Initiatives

- Deduce parallelism from Models of Computation
  - Ptolemy II
  - PREESM
  - Matlab + Simulink
  - ORCC
  - SystemC
- Extract parallelism from code
  - OpenMP
  - OpenCL

- Simulation oriented
- Execution oriented

High Performance Computing

Embedded Systems
OpenMP

- Implemented on GCC 4.2
- Implemented in TI compiler for Keystone I
- Adds pragmas (metadata) in C to tell the compiler which loops can be parallelized

Example:

```c
#pragma omp parallel for
For(i=0; i<10; i++){
    T[i] = f(i);
}
```

The compiler knows that the iterations are independent (data parallelism in this case) and can be parallelized.
Other Tools and Initiatives with Common Objectives

• **Model-Driven-Engineering (MDE)**
  Starting from UML meta-model and meta-model transformation
  Defines a whole methodology from specification to final system
  UML profiles like UML Marte have been defined for that purpose

• **Synchronous languages**
  Lustre, Signal, Esterel…
  Specifying synchronizations between operators: close to dataflow

• **C extensions**
  OpenMP, OpenCL, OpenACC, OpenHMPP, Cilk
  Principal objective: a painless migration to coarse-grain parallelism

• **StreaMIT - MIT**
  A language for stream processing, close to dataflow MoCs, and a compiler for massively parallel machines
Other Tools and Initiatives with Common Objectives

- **Task queueing software frameworks**
  Intel Threading Building Blocks, Apple Grand Central Dispatch
  Not specifically oriented towards stream processing applications

- **Task queueing software/hardware frameworks**
  Open Event Machine and Multicore Navigator
  → Available on C6678
Preesm: Rapid Prototyping for Multi-core DSP

• **Eclipse-based Open Source Rapid Prototyping Tool**

• **Collaboration with Texas Instruments Nice**
  Communication Infrastructure Business Unit (CI BU)
  Rapid prototyping of base station algorithms (used for LTE)

• **Goals**
  Combine imperative actor (host) code and dataflow coordination code
  Latency, memory and energy study of embedded code execution
  Before target hardware is available
  With efficient automatic parallelization heuristics
  Generating static multi-core code
  Reuse legacy code
Rapid Prototyping using Pree sm

Algorithm Model

Constraints + Options

Architectu re Model

Pree sm

Self-timed code

Simulator + Debugger + Profiler

Program Program Program Program
Pree sm: Rapid Prototyping for Multi-core DSP

• **Algorithm Model**
  Static Hierarchical SDF with C-like behaviour (IBSDF and PiSDF)
  Combined with C Actor Code

• **Architecture Model**
  System-Level Architecture Model (S-LAM) of heterogeneous architectures
  Focuses on important contention points

• **Prototyping**
  Shared memory / Message passing / DMA transfers /
  Load balancing enhancement
Static Code Generation: Self-Timed

Actor A
Actor B
Actor D
Actor E

Data

Actor C

B<br>C<br>D<br>E
General Conclusion

- **Applications and architectures are increasingly complex**
  Model-based system design helps at several design stages

- **To evaluate languages/models: focus on MoC**
  MoCs offer « pure » semantics, free of syntax

- **No one-fit-all solution to design Multicore DSP systems**
  Many solutions exist now, complex choices have to be made
Demo