A Component-Based Multicore Programming Environment

Pierre Paulin
Director, SoC Platform Automation
STMicroelectronics (Canada) Inc.

Core’s Law (for Embedded SoCs)

Number embedded cores in SoC products doubling every ~2 years

Source: Casual observation
Core’s Law: What’s Next?

- **Cores per Chip**
  - 2002: 1
  - 2004: 4
  - 2006: 8
  - 2008: 16
  - 2010: 32
  - 2012: 64
  - 2014: 128
  - 2016: 256

- **Corezilla**
- **Corrama**
- **Multi-core**
- **Coremporium**

- **STMicroelectronics**
Lessons of History

- 50 years of sequential programming has taken us to the edge of the abyss

- With parallel programming, we will all take a huge leap forward
Platform Programming Models

Multi-Processor SoC for

DUNA

Controller, DMAs
PE PE PE PE
PE PE PE PE
PE PE PE PE

Shared L1 MEM

PE PE PE PE
PE PE PE PE
PE PE PE PE

Controller, DMAs
PE PE PE PE
PE PE PE PE
PE PE PE PE

PE PE PE

Shared L1 MEM

PE PE PE

PE PE PE

PE PE PE

I/O Mem H/W

RISC DSP MCU Bus

NoC

Controller, DMAs

PE PE PE PE

PE PE PE PE

PE PE PE PE

PE PE PE

PE PE PE
Outline

- Platform 2012 Multicore Fabric
- Platform 2012 Programming Environment
  - Component-based programming models
  - Component-aware debug and visualization tools
- Case Studies
  - Video High-Quality Rescaling
    - Mapped to S/W platform
    - Mapped to H/W-S/W platform
  - VC1 codec
The P2012 Scalable Tile

- P2012 Fabric can integrate up to 32 clusters
- 1-16 configurable cores / cluster
- Optional H/W Processing Elements
- Few 100 GOPs to several TOPS
P2012 Cluster Overview

Fabric Interconnect (ANoC – Asynchronous NoC)

GALS I/F

Local Interconnect (ANoC – Asynchronous NoC)

Cluster Controller

ENCore<n>

Cluster Controller

Debug & Test Unit

STMicroelectronics
P2012 Design Flow

Application

Runtime SW libraries & drivers

Customizable tile

Customizable core

NI DMA L1 CC

F1 F2 F3 F4

Tools

V1 SW fabric

SW only cluster

HW/SW cluster

HW cluster

[FPU]

[Bit-Stream Processing]

[VECx extension]

STxP70 [config I/S]

OCE

9
Outline

- Platform 2012 Multicore Fabric
- **Platform 2012 Programming Environment**
  - Component-based programming models
  - Component-aware debug, visualization and analysis tools
- Case Studies
  - Video High-Quality Rescaling
    - Mapped to S/W platform
    - Mapped to H/W-S/W platform
  - VC1 codec
Programming Tools Outline

- MIND Component Infrastructure
- Component-based Programming models
- Programming tools flow
- Runtime
- Apex Application Modeling
- Trace, Visualization and Analysis
- Component-aware Debug
Component-based Progr. Models

- Encapsulation & Interfaces
  - Good for distributed memory
- Binding through link components
  - Heterogeneity
- Control interface
  - Introspection
  - Observability
- Semantic neutral
  - Can be used to support multiple prog. models
S/W Components

- Support of Multiple Communication/Execution Semantics

- Synchronous Call
- Asynchronous Call
- Proper Glue-Logic Automatically Generated
- Priority-based
- Preemptive
- Non-preemptive
- Real-time

RPC

S/W Architecture, Components, Interfaces

Tools

IDL

ADL

FIFO

Scheduler
Explicit description of the application architecture
- ADL: Architecture Description Language
- IDL: Interface Description Language

Built on Fractal MIND Component infrastructure
- Open source (LGPL) available on OW2 (mind.ow2.org)
MIND Toolchain

**interface** comm.QueueWriter {
    **void** createQ(size_t sz, **int** nb);
    **void** destroyQueue();
    **void** * readNext();
}

`int METH(itf, func) (void) {
    while((in= CALL(input, readNext))() !=0) {
        compute_function(in, out, PRIVATE.arg);
    }
} `

**Inputs**

**ADL**

**IDL**

**C**

**minda**

**Generated C**

**mpp**

**gcc**

**ADL**: Architecture Description Language

**IDL**: Interface Definition Language

**minda**: MIND ADL/IDL parser and C code generator

**mpp**: MIND PreProcessor

**Executable/Loadable binary**
Prog. Tools & Runtime Outline

- MIND Component Infrastructure
- **Component-based Programming models**
- Programming tools flow
- Runtime
- Apex Application Modeling
- Trace, Visualization and Analysis
- Component-aware Debug
Programming Models Objectives

- **Efficiency:**
  - Max parallelism with minimum overhead

- **Productivity:**
  - Abstraction
  - Ease of debugging
  - High-level analysis

- **Scalability:**
  - More resources $\rightarrow$ more performance

- **Platform independence:**
  - Patterns designed from applications perspective
PPM Development: Dual Approach

- Build basic set of Parallel Programming Patterns
  - For exploiting data-level (DLP) & task-level parallelism (TLP)
  - Communication, synchronization and memory management patterns
  - Constructions for thread-based programming
    - Thread creation/assignment, synchronization, msg. passing
  - Constructions for dataflow programming (streaming)
    - Execution engines (schedulers), filters template, queues

- Refine PPPs from application experience
  - Video Codecs (VC1, H.264)
  - Image Quality Improvnt. (HQR, TMNR, TNR, MC-DEI)
  - Image analysis (pedestrian recognition)
Parallel Programming Models

Execution model
- Run-to-Compl.
- PEDF
- Thread pool
- DDF
- SDF
- Thread

Communication/synchronization
- Exchanger
- Queue iterator
- FIFO
- Synchr Buffer
- ...
Parallel Programming Patterns

- High-level set of patterns used to parallelize applications
  - Exploiting different types of parallelism
  - Interchangeable implementations
- Component-based (MIND framework)
Communication patterns (examples)

- **Exchanger:**
  \[
  \text{buf} = \text{exchange}(\text{buf})
  \]
  - Swapping buffers between two participants

- **Queue Iterator:**
  \[
  \text{buf} = \text{writeNext}(\text{buf});
  \text{buf} = \text{readeNext}()
  \]
  - Iteration based communication between producer(s) and consumer(s)
  - Single queue
  - Split / Join / Broadcast
Communication patterns (examples)

- **Synchronized buffer:**
  \[\text{request/release\{read,write\}}(\text{buf})\]
  - Synchronized read/write on shared buffer
    - Sliced Synchronized buffer:
      Specialization to access a large buffer in smaller slices
  - **FIFO:** \(\text{push()}; \text{pop()}; \text{peek()}\)
  - Packet based streaming between producer and consumer
    - Buffer copy or buffer pointer passing
    - Single queue
    - Split / Join / Broadcast
Memory access patterns

- **Prefetcher:** key=load(ptrL3, size), ptrL1=get(key), free(key)
  - Prefetching data from L3 to L1
    - load() programs the prefetch and returns a key without waiting for the transfer
    - get() returns a pointer to L1, blocks until the transfer is completed
    - Also supports 2D arrays: key=load(ptr, width, height)

- **Async. Prefetcher:** load(ptr, size, handler(key)), free(key)
  - Enables efficient thread-pool execution engines
    - load() programs the prefetch and returns waiting for the transfer
    - handler(key) is invoked by the execution engine when the transfer is completed
    - Also supports 2D arrays: load(ptr, width, height, handler(key))
Execution Patterns: Static Threads

- Static mapping of kernels on a set of platform resources
- Minimal runtime overhead
  - No kernel multiplexing required
- Manual load balancing
  - Similar computational requirements for each kernels
- Example usage
  - Video High-Quality Rescaling (HQR)
  - Mapped to S/W
Execution Patterns: Thread Pool

- Dynamic dispatch of kernels on a set of platform resources
- Some runtime overhead
  - Mux K kernels on R resources
- Dynamic load balancing
  - Different heuristics offered
    - Job stealing
    - Cache awareness
- Example usage
  - H.264 Motion Estimator
  - Mapped to S/W
Execution Patterns: Dataflow

- Predicated Execution Data-Flow (PEDF)
- Host Communication Component
  - Models part of application that communicates with host
- Mode Controller
  - Configures control parameters, steps pipeline
- Filters
  - Perform actual data computation
- Example use: Video HQR
  - Mapped to H/W-S/W
Prog. Tools & Runtime Outline

- MIND Component Infrastructure
- Component-based Programming models
- *Programming tools flow*
- Runtime
- Apex Application Modeling
- Trace, Visualization and Analysis
- Component-aware Debug
P2012 Mapping Flow

Application Capture

- Streaming: PEDF, DDF
- Parallel Thread

Programming Tools

- Component-based
  - Analysis, Optimization
  - Communication gen\textsuperscript{n}
  - Execution engine config\textsuperscript{n}
  - Trace synthesis, debug info generation

Runtime

- Deployment
- Execution Engines
- QoS, Power mgr.

Performance Analysis

P2012 Platform

- L2 MEM
- CC
- DMA
- HWS
- L1 MEM
- PE0
- PE1
- ... PEn

Visualization/debug

Traces
Application-to-Platform Mapping

Mapping tools: assignment, transformations, comm. generation, runtime link

Runtime: dynamic deployment, assignment, & scheduling
Apex: Application Modeling & Simulation Environment

C Reference Model

Refinement

Component-based Models

Apex Objectives
- Capture, validation of high-level sequential & parallel descriptions
- Verification of parallel correctness

Apex benefits
- No need for TLM platform
- Fast simulation: >100x TLM speed
- High-level validation parallelism
- Compatible with mapping tools

Sequential Model

Parallel Model

Firmware on P2012

P2012 mapping tools

Apex Objectives

Apex benefits

Apex

Host

Apex Objectives

Apex benefits

STMicroelectronics
Visualization and analysis flow

Mapping tools

- Correspondence between application description and platform resources
- Application mapping results
- Used resources
- Symbolic names / physical IDs

Visualization and analysis tools (in STWorkbench)

P2012

SoC

Fabric Ctrl

Host

System Trace Module

Trace Database

Low-level Traces
prog. Model-aware visualization

- Displays broadcast, exchanger, split/join, sync. buffer, exec. patterns
NPL Visualization

- Mutex critical section
- Mutex ownership change
- Barrier
- Nested functions
Component-aware multi-core debug

- Component-aware debug
  - Print state variables (attributes, private data)
  - Break on component method, conditional breakpoint on a specific instance
  - Jump over compiler-generated interface stubs
  - Print instance hierarchy of the application
  - Print current location in the hierarchy

- Multicore features
  - Single cockpit controlling multiple debugger instances
  - Support for identical program images
  - Support for heterogeneous program images planned
P2012 Debug flow

Mapping tools

Component hierarchy description

STWorkbench

Terminal

STMicroelectronics
Outline

- Platform 2012 Multicore Fabric
- Platform 2012 Programming Environment
  - Component-based programming models
  - Component-aware debug, visualization and analysis tools
- Case Studies
  - Video High-Quality Rescaling
    - Mapped to S/W platform
    - Mapped to H/W-S/W platform
  - VC1 video codec
- HD 1080p, 120 fps
- SDF model variant
  - One “token” on in/out per link per filter firing
    - Or simple static multi-rate
  - Tokens typically a line of pixel data
  - Multiple modes (on frame-by-frame basis)
  - Some dynamic control flow, exceptions
    - E.g. dynamic bypass of a filter
Two Mapping Approaches

- Map to S/W-based platform
  - Data-level parallelism
    - Structured threading model
    - Multi-processor & SIMD
  - All tasks for a given data element assigned to single PE

- Map to H/W-dominated platform
  - Task-level parallelism
    - Dataflow programming model
    - Software-based control
  - Tasks assigned to a single H/W Processing Unit
    - DLP inside each H/W PU
S/W Mapping: HQR example

- Data-level parallelism
  - Each image line split into stripes
  - Each PE runs all filters for a stripe
  - SIMD optimization of each filter

- Parallel Progr. Patterns
  - Data iterator split and join patterns
  - Synchronization between PEs using “exchanger” pattern (for border pixels)

![Diagram of data-level parallelism and parallel program patterns](Diagram)

- Stripe Width = W
- Pixels Accessed = W + 2 + 2
S/W Mapping: HQR example

- Data-level parallelism
  - Each image line split into stripes
  - Each PE runs all filters for a stripe
  - SIMD optimization of each filter
- Parallel Progr. Patterns
  - Data iterator split and join patterns
  - Synchronization between PEs using “exchanger” pattern (for border pixels)
HQR PPP Profiling Results

- Concrete impact on performance in the case of HQR
  - Less than 1%
## Vectorization results (16 way VECx EFU):
- Results for standalone CA-ISS
- Average vector unit utilization 79% 

## Parallel processing results (1 vs. 4 PEs)

<table>
<thead>
<tr>
<th></th>
<th>Single CPU</th>
<th>4 CPU Initial</th>
<th>4 CPU Burst communication</th>
<th>4 CPU Buffer dimensionning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>19179956</td>
<td>2 X</td>
<td>3.3 X</td>
<td>3.9 X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9406673</td>
<td>5731049</td>
<td>4811882</td>
</tr>
</tbody>
</table>

*Results on cycle-approx TLM platform*
HQR Optimization Process

- Reference code: frame-based (no vectorization)
- Line-based (vectorization unoptimized)
- Line/column switch before hor. spline filter (simplifies processing)
- Line/column switch before last filter (vectorization optimized)
- Use new specialized SIMD instructions
- ~5X cost of H/W solution
- 697 PEs
- 200 PEs
- 26 PEs
- 16 PEs
- 12 PEs
- 8 PEs
- Wide EFU
Task-level parallelism
- Assignment of each filter to a H/W PU
- Grouping of highly communicating PUs to a single PE

In contrast with S/W mapping, where
- Data-level parallelism exploited (Multi-PE and SIMD)
- Each PE performs all tasks
PEDF Dataflow Programming Model

Predicated Execution Data Flow

- **Host Communication Component**
  - Gets host request params.
    - Frame data
    - Required processing type
    - Processing parameters

- **Mode Controller**
  - Configures control parameters, steps pipeline

- **Filters**
  - Actual data computation

- **Auto-generated Iteration Controller**
Overview of PEDF Mapping Flow

- Abstract PEDF Application Capture
- Mapping tools
  - Single PEDF description
  - Mapped to Host with Apex tool
  - Mapped to TLM platform
    - Control code on STxP70
    - Microcode on data streaming ports of H/W PEs

Apex Host Execution
HW-SW Interaction in P2012

Hardware Processing

Data streaming

Load/Store

Cluster Control

Data Mover

STMicroelectronics

STXp70
Mapping to HW/SW Platform

Functional MODEL

Connectivity “Glue”

Configuration Controller

Iteration Controller

TLM / COSIM

STxP70 ISS

NOC Model

- Application capture using DF variant prog. model
- Host execution (APEX) for functional validation
- Automatic control code generation for TLM/COSIM for performance analysis
TNR Performance Comparison
FlexMap v.s. hand-coded (STxP70 Instrns/Frame)

- More abstract capture allows for more optimization

>50% reduction in execution time

- Original Reference
- Static
- Dynamic
- Dynamic/ASR
Multiple Programming Models

- Top-level: Dynamic Dataflow pipeline
- Interchangeable Thread/DLP and PEDF implementations of HQR
- Components act as semantic-neutral structuring mechanism
VC1 overview: task-level parallelism

- **Input bitstream**
- **Ctrl**
- **Broadcast**
- **VLD**
  - Bit Parse
  - VLD Intra
  - VLD Inter
- **Simple Queue**

**MV Pred**
- Prefetch manager
- I/F to obtain the reference MB
- Uses the CDMA

**Intra Prediction**
- AC/DC
- IZZ

**Motion Compens**
- MC
- Intensity compensn.

**Loop Filter**
- Deblock
- Range mapping

**IDCT Reconstruction**
- IQ
- IDCT
- Smooth

**Queue**
- Queue
- Queue (MB)
- Queue (2 lines)

**Comm components**
- Available in a library
- Binding between parallel components

**Processing functions**
- Vectorial (SIMD) data parallelism

**Application components**
- Pipelined task-level parallelism

- Frame/slice control read by all
- Process one segment of macro-block at a time (dataflow)
- Contains motion vectors + data
- 2 decoded reference frames in L3. Circular buffer
- 1 decoded frame
- 1 decoded frame
- Uses the CDMA
- 2 decoded reference frames
- Display

- Uses the CDMA
- Uses the CDMA
Component-based Prog. Models

- Demonstrated value of components
  - Supports multiple programming models
    - DLP/threads S/W mapping
    - PEDF HW/SW mapping
  - Multiple & evolving execution targets
    - H/W-S/W partitioning
  - Multiple simulation environments
- Platform independence
  - Abstraction of communication
- No overhead in practical use
  - S/W mapping: <1% overhead
  - H/W-S/W mapping: 50% execution time reduction
Multi-Processor SoC for Smart People

Programming Models:
- Higher productivity
- More platform independence