Programming Heterogeneous Embedded Systems for IoT

Jeronimo Castrillon
Chair for Compiler Construction
TU Dresden
jeronimo.castrillon@tu-dresden.de

Get-together toward a sustainable collaboration in IoT
April 18th, Tunis, Tunisia
Internet of things

- Distributed
- Real time
- Energy efficient
- Resource constrained
- Highly heterogeneous
- Multiple domains

Source: Open clipart
Internet of things

- Distributed
- Real time
- Highly heterogeneous
- Energy efficient
- Resource constrained
- Multiple domains

IoT/Systems-of-Systems: Extremely hard to program and reason about

Source: Open clipart

Independent nodes: Trends

- Single node: HW complexity
  - Increasing heterogeneity
  - Complex interconnect
  - Increasing number of cores

- SW “complexity”
  - Not anymore a simple control loop
  - Need expressive programming models

Independent nodes: Trends

- **Single node: HW complexity**
  - Increasing heterogeneity
  - Complex interconnect
  - Increasing number of cores

- **SW “complexity”**
  - Not anymore a simple control loop
  - Need expressive programming models

---

<table>
<thead>
<tr>
<th>Year</th>
<th>OMAP1</th>
<th>OMAP2</th>
<th>OMAP3530</th>
<th>OMAP3640</th>
<th>OMAP4470</th>
<th>OMAP5430</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outline

- Introduction
- Models of computation
- Tool flows
- Outlook for IoT
- Summary
Outline

- Introduction
- Models of computation
- Tool flows
- Outlook for IoT
- Summary
What is a MoC?

A **model of computation** is the definition of the set of allowable operations used in computation and their respective costs. It is used for measuring the complexity of an algorithm in execution time and or memory space.

Wiki: From computational theory

**Model of computations** are abstract specifications of how a computation can progress.


**A Model of computation** is a collection of rules that govern the interaction of components.

A **Model of computation** is a collection of rules that govern the interaction of components.

- **What are the components**
  - Threads, processes, actors, tasks or procedures
- **Execution and concurrency**
  - Order and determinism
- **Communication mechanisms**: How do components exchange data
  - Asynchronous or rendezvous, perfect or lossy

---

Why MoCs?

- Depending on the rules, MoCs feature different **properties**

  **Properties**
  - Relate to the power: what can be computed with the model
  - Makes it easier for automatic tools to understand and synthesize code
    - E.g., compile-time optimization, support resource constraints, ...

- Examples
  - Can the application run on bounded memory?
  - Can we compute the maximal throughput?
  - Is the execution deterministic?
For IoT in general: Mixture of MoCs interacting (across different domains).

In this talk: Software (concurrent, dataflow + PN + time)
Homogeneous Synchronous Dataflow (HSDF)

- Graph with simple execution semantics

- Functional, graph model can be extended to model
  - Bounded buffers
  - Schedules
HSDF: schedules

- Overlapping schedule

How to delay the execution of a3?, how to prevent a4 from executing earlier?

PE2

PE1

a4 a1 a4 a1 a4 a1

a3 a2 a3 a2 a3 a2

Time
Outline

- Introduction
- Models of computation
- Tool flows
- Outlook for IoT
- Summary
Programming flow: Overview

MoC-based Application

Analysis

Synthesis

Code generation

Non-functional specification

Architecture model

PNargs_ifft_r.ID = 6U;
PNargs_ifft_r.PNchannel_freq_coef = filtered_coef;
PNargs_ifft_r.PNnum_freq_coef = 0U;
PNargs_ifft_r.PNchannel_time_coef = sink_right;
PNargs_ifft_r.channel = 1;
sink_left = IPC11mrf_open(3, 1, 1);
sink_right = IPC11mrf_open(7, 1, 1);
PNargs_sink.ID = 7U;
PNargs_sink.PNchannel_in_left = sink_left;
PNargs_sink.PNnum_in_left = 0U;
PNargs_sink.PNchannel_in_right = sink_right;
PNargs_sink.PNnum_in_right = 0U;
taskParams.arg0 = (xdc_UArg) & PNargs_src;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knl_Task_FuncPtr) & ft_Templ, &taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg) & PNargs_fft_l;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knl_Task_FuncPtr) & ft_Templ, &taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg) & PNargs_ifft_r;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knl_Task_FuncPtr) & ft_Templ, &taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg) & PNargs_sink;
taskParams.priority = 1;
Application model: Beyond HSDF

- **SDF**: Popular model for static applications
  - Possible to derive equivalent HSDF

- Dynamic DF (DDF): For dynamic applications
  - Flexible firing rules (even non-determinism)

- Kahn Process Networks (KPN): Also dynamic
  - Deterministic
  - No firing semantics
Non/extra-functional: Constraints

- Timing constraints
  - Process throughput
  - Latencies along paths
  - Time triggering
- Mapping constraints
  - Processes to processors
  - Channels to primitives
- Platform constraints
  - Subset of resources (processors or memories)
  - Utilization
Key for retargetable tool flows
- Abstract for speed
- Relevant components: processors (types), interconnect and memories

Why? Understand effect of executing an actor on the different resources for a schedule
Abstract models with functional units, latencies

SSA-IR (LLVM)

Profiler (CoEx)

Execution counts, branch stats and execution traces

Modeling communication

- High-level: Available APIs for implementing application channels
- Benchmarking for different platforms

[Arnold13]

[Oden13]
Modeling communication

- High-level: Available APIs for implementing application channels
- Benchmarking for different platforms

![Diagram showing a network with various components and connections, including FPGA-Interface, Router, and various processing units like Duo-PE, ADPLL, PMGT, and DDR-SDRAM-Interface.](image_url)

![Graph showing time in cycles against bytes for different data transfers: Write to scratchpad, Read from scratchpad, Write to RAM, and Read from RAM.](image_url)

---

Analysis and synthesis: Overview

- Application specification
- Architecture model
- Non-functional specification

Analysis: Instrumentation, profiling, tracing
Sequential performance estimation
Time-annotated traces
Mapping and scheduling
Parallel perf. estimation
Increase resources

Mapping configuration

For dynamic models!
Tracing: Dealing with dynamic behavior

- KPNs do not have firing semantics
- **White model of processes**: source code analysis and tracing
- Tracing: instrumentation, token logging and event recording

... for (; i < x; i++) {
    write(&c2);
    f1(...);
    read(&c1);
    f2(...);
    read(&c1);
...
Trace-based synthesis

- Non-functional specification
- Mapping, scheduling, buffer sizing
- Mapping configuration
- Architecture model

Synthesis based on code and trace analysis (using simple heuristics)

- Mapping of processes and channels
- Scheduling policies
- Buffer sizing

Mapping, scheduling and buffer sizing

- Graph representation of event traces
  - Reason about deadlocks
  - Find critical path

Downside: For dynamic portions of the application, graph is input dependent
Multiple traces (ongoing work)

- Different input $\rightarrow$ different behavior (traces)
  - Characterize behaviors and impact on mapping performance

[Goens15]
Multiple traces (2)

- Different input $\rightarrow$ different behavior (traces)
  - Characterize behaviors and impact on mapping performance

[Goens15]

Performance

Behavior

Mapping configuration

Mapping configuration

Mapping configuration
Parallel performance estimation

- Discrete event simulator to evaluate a solution
  - Replay traces according to mapping
  - Extract costs from architecture file (NoC modeling, context switches, communication)
Increasing resources

- Add resources to the synthesis until constraints are met
  - Mapping and scheduling
  - Parallel perf. estimation
  - Increase resources

- Add processors and memories
  - Easy for homogeneous platforms
  - Non trivial for heterogeneous platforms (ongoing work)
Multiple mapping configurations

- In case multiple applications may run on the system (or system of systems)
  - Need different configurations
  - Adapt to resource and energy budgets
- Reason about mapping equivalences

[Goens15]
Analysis for Multi-applications

- Use utilization profiles from the discrete event simulator (TRM)
- Quickly separate good from bad implementations
- Take mapping configuration and generate code accordingly
- From architecture model: APIs, configuration parameters, ...
- Sample targets: experimental heterogenous systems, TI (Keysonte, TDA3x), Parallela/Ephiphany, ARM-based (Exynos, Snapdragon)
Examples

- Same application(s) automatically compiled to different platforms

- Virtual platforms: SystemC models of full systems
  - Bare metal
  - Multi-RISC/VLIW platform

- Real platform: TI Keystone platform
  - Speedup on commercial platforms
  - Code generation against vendor stacks
Example: multi-media applications

- Iterative mapping

![Iterative Mapping Graphs]

- Left graph: Iterative Mapping with LP-AF Trials
  - X-axis: LP-AF Trials
  - Y-axis: Makespan (Mcycles)
  - Lines:
    - Real-time algorithm
    - Real-time constraint

- Right graph: Iterative Mapping with MJPEG Trials
  - X-axis: MJPEG Trials
  - Y-axis: Makespan (Gcycles)
  - Lines:
    - Real-time algorithm
    - Real-time constraint
Manual vs. Automatic: TI Keystone

Image processing

Audio filtering application

LTE digital receiver

[Aguilar14]
Outline

- Introduction
- Models of computation
- Tool flows
- **Outlook for IoT**
- Summary
So far...

MoC-based Application

Non-functional specification

Analysis

Synthesis

Code generation

PNargs_ifft_r.ID = 6U;
PNargs_ifft_r.PNchannel_freq_coef = filtered_coef;
PNargs_ifft_r.PNnum_freq_coef = 0U;
PNargs_ifft_r.PNchannel_time_coef = sink_right;
PNargs_ifft_r.channel = 1;
sink_left = IPC1mlrf_open(3, 1, 1);
sink_right = IPC1mlrf_open(7, 1, 1);
PNargs_sink.ID = 7U;
PNargs_sink.PNchannel_in_left = sink_left;
PNargs_sink.PNnum_in_left = 0U;
PNargs_sink.PNchannel_in_right = sink_right;
PNargs_sink.PNnum_in_right = 0U;
taskParams.arg0 = (xdc_UArg)&PNargs_src;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knl_Task_FuncPtr)&taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg)&PNargs_fft_l;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knl_Task_FuncPtr)&taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg)&PNargs_ifft_r;
taskParams.priority = 1;
ti_sysbios_knl_Task_create((ti_sysbios_knl_Task_FuncPtr)&taskParams, &eb);
glob_proc_cnt++;
hasProcess = 1;
taskParams.arg0 = (xdc_UArg)&PNargs_sink;
taskParams.priority = 1;

DMA, semaphores, PMU

Network Processor

Packet DMA

MEM subsystem

VLIW DSP

L1, L2, L1, A15, A15, A15, A15

Peripherals

Communication support

HW queues

NoC

Architecture model

Analysis

Synthesis

Code generation
What’s missing?

- **Reactivity**
  - Supported for periodic events (trigger)
  - Enough for many applications (automotive)
  - Sporadic (I/O) events may break determinism

- **Better execution guarantees**

- **Multi-SoC**
  - Supported if inter-SoC communication is predictable (same cost-function approach)
  - Scalability: Add a layer of “geographic” coarse mapping
Introduction

Models of computation

Tool flows

Outlook for IoT

Summary
Summary

- IoT complexity: heterogeneity, resource constraints, distributed, ...
- The need for formal MoC for programming and synthesis
- Sample tool flow
  - Heterogeneity: High-level architecture models
  - Support dynamic applications: Analysis and synthesis based on traces
  - Extensions for: multiple traces

- Outlook
  - Adaptivity at runtime
  - Reactivity and multi-SoC case studies
Acknowledgements

- Silexica Software Solutions GmbH

- German Cluster of Excellence: Center for Advancing Electronics Dresden (www.cfaed.tu-dresden.de)
References


