Design Space Exploration for MPSoC

Iuliana BACIVAROV
ETH Zurich
Outline

- Overview: MPSoC Design Flow
- MPSoC Modeling
- Design Space Exploration and Optimization
- Performance Evaluation
- Conclusions
What are MPSoC? Why MPSoC?

- **Context:** Multi-Processor System-on-Chip (MPSoC) design

- **Applications:**
  - wireless and audio/video encoders in mobile phones
  - game processors in game consoles
  - xDSL and network processors in routers
  - audio/video encoders in portable DVDs, digital TV, HDTV

- **Advantages:** high integration scale, low power consumption, fast processing times

- **Problems:** the high complexity and the huge design space:
  - architectural parameters? SW? interconnect? mapping? scheduling? ...
Key Issues:

- **Design space exploration**
  - high diversity of solutions and parameters

- **Optimization**
  - multiple (conflicting) objectives

- **Performance evaluation**
  - high evaluation speed
  - high accuracy
Outline

- Overview: MPSoC Design Flow

- MPSoC Modeling
  - Application Model
  - Architecture Model
  - Mapping (application-to-architecture)

- Design Space Exploration and Optimization

- Performance Evaluation

- Conclusions
Application Model

Two parts:
- **Structure:**
  - processes
  - communication channels
- **Behavior** of each process
  - e.g. C program

**Different specification approaches:**
- multi-threading, RPC, MPI, queuing networks, process networks, ...
Application Model Example

- **Example** of a task graph of a packet processing application.
  - several flows
  - each flow includes several sequences of processes

- **Difficulties:**
  - specification:
    - inter-dependencies between flows, dependencies between processes
  - evaluation of properties on a real platform:
    - competition for limited resources, different processing & communication times
Architecture Design Space

Which architecture is better suited for a given application?
(SHAPES) Architecture

**Elements:**
- tile = RISC + DSP + mem. + periph. + interfaces + bus
- chip = multiple tiles + NoC
- board = multiple chips + off-chip interconnects
Mapping

- **Binding**
  - binding of processes to resources
    - processi => processorj
    - passive process => memory
  - binding of communication between processes to hardware links
    - SW channeli => HW pathj

- **Scheduling**: for both processes & communication
  - policy
  - parameters

- Mapping needs to **comply with given constraints**
Mapping Specification

Mapping = binding + scheduling → for both processes and SW channels between processes
Outline

- Overview: MPSoC Design Flow
- MPSoC Modeling

Design Space Exploration and Optimization
- Objectives, constraints and design variables
- Multi-objective optimization

Performance Evaluation

Conclusions
Goal: Optimally map an application onto an architecture
- optimization → early in the design
Optimization – Objectives

- **Goal**: Optimally map an application onto an architecture
  ⇒ obtain the extreme of objective function \( f(x) \), given the constraint function \( g(y) \)

- **Objectives**:
  - good timing performance
    - high system throughput
    - low end-to-end delay
  - low power consumption
  - small area (number of resources)
  - high robustness
  - low cost

- **Multi-objective optimization**: find a set of optimal trade-offs

**Multi-objective optimization** often leads to conflicts that require trade-offs, especially in computer design systems.
Optimization – Constraints

Optimization: obtain the extreme of \textbf{objective function} \( f(x) \), given the \textbf{constraint function} \( g(y) \)

- constraints \( \rightarrow \) taken into account during optimization
  - 2 alternatives:
    - 1. constraints embedded into the objective function
    - 2. optimize only the designs that verify the constraints

- \textbf{Examples}
  - check correct functionality:
    - does the code size fit into the available memory?
    - buffer overflow / underflow?
    - is the system schedulable?
  - comply with Real Time constraints:
    - e.g. system delay \(< \) Real Time delay
    - e.g. signal quality is above “acceptable” threshold
Optimization – Design Variables

- **Application design variables:**
  - e.g. concurrency, exec. T, buffer demands

- **Architecture design variables:**
  - e.g. type/ number of resources?
    CPU=? DSP=? memory? clock frequency? NoC?

- **Mapping design variables:**
  - assignment: processes → processors, communication → interconnection paths; scheduling policy and parameters?

Huge Parameter Space!!
Multi-Objective Optimization – Illustration

- **Aim**: define the objective function in function of **ALL** important objectives
  
  e.g. **Performance = (delay, power consumption, cost)**

- Both architectures are useful, for different applications
Multi-Objective Optimization

**Goal:** e.g. to minimize \((y_1, y_2, \ldots, y_k) = f(x_1, x_2, \ldots, x_n)\)

**Difficulties:**
1. large search space
2. multiple optima

---

Better

Incomparable

Worse

Not dominated

Pareto optimal

---

\[ x_1 \quad x_2 \]

\[ y_1 \quad y_2 \]
Optimization Alternatives

- **1. use of (traditional) single objective optimization methods**
  - e.g.: simulated annealing, integer linear program, iterative heuristic methods
  - **Decision making** (weighting the different objectives) is done **before the optimization**.

- **2. population based optimization methods**
  - e.g.: evolutionary algorithms
  - **Decision making** is done **after the optimization**.
Optimization Alternatives

1. use of (traditional) single objective optimization methods
   - e.g.: simulated annealing, integer linear program, iterative heuristic methods
   - Decision making (weighting the different objectives) is done before the optimization.

\[ y = w_1 y_1 + \ldots + w_k y_k \]

(e.g.: weighting approach)
Optimization Alternatives

1. use of (traditional) single objective optimization methods
   - e.g.: simulated annealing, integer linear program, iterative heuristic methods
   - Decision making (weighting the different objectives) is done before the optimization.

2. population based optimization methods
   - e.g.: evolutionary algorithms
   - Decision making is done after the optimization.
Optimization Alternatives

Evolutionary Algorithms – Principles:

1. Selection
2. Cross-over
3. Mutation

- 2. population based optimization methods
  - e.g.: evolutionary algorithms
  - Decision making is done after the optimization.
Multi-Objective Optimization – Strategy & Tools

- Optimization
  - evolutionary algorithms: mutation, cross-over, selection

- Software toolboxes
  - PISA (Platform and Programming Language Independent Interface for Search Algorithms)
    https://www.tik.ee.ethz.ch/pisa
  - EXPO (tool to explore the design space for network processor architectures)
    https://www.tik.ee.ethz.ch/expo

- Performance evaluation
  - need of fast & accurate performance evaluation !!

- Optimization algorithm
  - (e.g. SPEA2, IBEA, NSGA2, …)

- EXPO Tool Framework
  - Control
  - Population handling
  - EXPO (Main Module)

- GUI

- Optimization problem
  - Problem specification
  - Generation of new solutions
  - Design Evaluation

* SPEA2 = Strength Pareto Evolutionary Algorithm 2
  IBEA = Indicator Based Evolutionary Algorithm
  NSGA2 = Nondominated Sorting Genetic Algorithm 2
Outline

- Overview: MPSoC Design Flow
- MPSoC Modeling
- Design Space Exploration and Optimization

Performance Evaluation

- Why is Performance Analysis Difficult?
- System-Level Performance Methods:
  - Co-Simulation
  - Mixed Approaches: Simulation + Analysis
  - Analytic Approach → Next Talk

Conclusions
Why is the Performance Evaluation Difficult?

- **complex behavior**
  - data dependent behavior
  - variable execution demand
    - input (different event types)
    - internal state (program, cache, ...)

- **interference**
  - same application / multiple applications
    - limited resources
    - scheduling/arbitration
    - anomalies

- **heterogeneity, HW-SW**
System-Level Performance Evaluation Methods

- e.g. delay

- Worst-Case

- Best-Case

- Real System
  - Measurement
  - Simulation
  - Probabilistic Evaluation
  - Worst Case Analysis
Overview over Some Existing Approaches

- **Analysis**
  - e.g. Thiele, Ernst

- **Mixed**
  - e.g. Lahiri, SPADE

- **Simulation**
  - e.g. Jerraya, Benini, Leupers

(speed) vs (accuracy)
Outline

- Overview: MPSoC Design Flow
- MPSoC Modeling
- Design Space Exploration and Optimization

Performance Evaluation
- Why is Performance Analysis Difficult?
- System-Level Performance Methods:
  - Co-Simulation
    - Mixed Approaches: Simulation + Analysis
    - Analytic Approach ➔ Next Talk
- Conclusions
What is Cosimulation? Why Cosimulation?

- heterogeneous systems
  - different execution models
  - different communication models
    - e.g. SW models: native execution+timing annotations; ISS-based execution
    - e.g. HW models: RTL model, TLM cycle-accurate, functional model; Microarchitecture Simulator

- cosimulation environment & interfaces
  - desirable:
    - automatic composition of component execution models
    - abstract interfaces
    - abstract models
HW/SW Cosimulation Techniques

Cosimulation: 1. functional

HW & SW:
- native execution
- functional description

doesn’t take into account:
- HW/SW interactions
- the OS
- execution times
- the HW part

2. OS-based

SW:
- native execution
- functional +OS model

doesn’t take into account:
- the real OS
- OS overhead

3. ISS -based

SW:
- ISS simulated
- functional +OS

-good precision

-but: slow (100x, 1000x slower than the native execution)
HW/SW Cosimulation Techniques

1. Functional Cosimulation
   - HW & SW:
     - native execution
     - functional description

2. OS-based Cosimulation
   - HW & SW:
     - native execution
     - functional + OS model

3. ISS-based Cosimulation
   - HW & SW:
     - ISS simulated
     - functional + OS

Speed:
- High

Accuracy:
- Low

Execution environment (e.g. UNIX)
HW description
SW description

Cosimulation bus

HW simulator
SW description
OS model

ISS

SW description
OS
MPSoC Cosimulation – Examples

- Environments for multiprocessor system cosimulation
  - e.g. ROSES - Jerraya et al., MPARM - Benini et al., etc.

- Simulation models
  - e.g. I. Bacivarov – PhD thesis, TIMA Labs.

![Diagram of MPSoC Cosimulation Examples](image)
Performance Evaluation Model for the SW Subsystem

- **speed** => high level model & native execution
  - processor specific parts => functional model for the HAL – Hardware Abstraction Layer
- **timing accuracy** => code timing annotation ~ target architecture
  - DELAY()
- **functional accuracy** => interactions with adjacent subsystems
  - (IT, synchronization)

![Diagram](image)

- **Execution environment** (e.g. UNIX)
- **HW simulator**
- **HW description**
  - OS code non-dep. of the processor
  - OS code dependent of the processor (HAL)
- **SW description** (application)
  - + DELAY() annotation directly in the code
  - replaced by the functional HAL model + DELAY() annotation

```c
void OS_init() {
    ...
    Delay(25);
    create_task(...);
    ...
}

void create_task() {
    ...
    Delay(6);
    create_abs_cxt(...);
    ...
}
```

- application model
- OS model
- HAL functional model
Simulation Model for the Processor-Dependent Sections

- experiments: OS (application specific) generated from ROSES libraries [Ph.D. Thesis – TIMA Labs., L. Gauthier '01]
- model example:

```c
__cxt_switch ;r0 = old context indicator,
               ;r1 = new context indicator
STMIA r0!,{r0-r14} ;save current task registers
LDMIA r1!,{r0-r14} ;restore new task registers
SUB pc, lr, #0    ;return
END
```

```c
void context_sw(int cur, int new)
{
    Delay(34);
    getcontext(ucp_list[cur]);  // UNIX multi-threading lib.
    setcontext(ucp_list[new]);  // UNIX multi-threading lib.
    Delay(3);
}
```
SW Code Annotation with HW Execution Times

- **INPUT** = the (C) source code
- **OUTPUT** = the same code + execution times

**Steps:**

1. **pre-processing:**
   - compilation for the target processor
   - correspondence: source code <-> assembler code

2. **delays computing:**
   - using the processor Data Sheet

3. **delay annotation:**
   - C code : unmodified
   - code ASM = replaced with delay
     - Delay(ASM execution times)
   - Lex/Yacc Tool

**Example- C language:**

```c
i=4;
i++;
```

**Corresponding assembler:**

```
MOV r0,#4
LDR r1,|L1.40|
STR r0,[r1,#0] ;i=4
LDR r0,|L1.40|
LDR r0,[r0,#0] ;i++
ADD r0,r0,#1
STR r0,[r1,#0]
```

**Computing & annotation with execution delays**

**Annotated C code:**

```c
i=4;  Delay(10);
     Delay(20);
     Delay(20);
i++;  Delay(20);
     Delay(10);
     Delay(20);
```
SW code annotation with HW execution times

- **Delay() function**
  - prototype declared with the application/OS
  - executed in collaboration with SystemC (through the timing cosimulation bus - TBFM)
  - (1) HW/SW timing synchronization;
  - (2) processor IT simulation.

![Diagram showing the process of delaying a function with HW execution times.](image-url)
Execution Times / Simulation Times

- VDSL modem application example:

<table>
<thead>
<tr>
<th>HW IP</th>
<th>ARM7</th>
<th>ARM7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Speed-up improvement
   - factor 10
   - Limitations: HW simulation fixed at RTL, [48.9% of total simulation time]
   - Limitations: synchronization via IPC UNIX, (large access times) [31.8% of total simulation time]

2. Accuracy improvement:
   - real OS simulation, processing times, HW/SW interactions
   - Errors: evaluation method for execution delays, worst case (for instructions ....), w/o pipeline, w/o cache, w/o compiler optimizations
Outline

- Overview: MPSoC Design Flow
- MPSoC Modeling
- Design Space Exploration and Optimization

Performance Evaluation
- Why is Performance Analysis Difficult?
- System-Level Performance Methods:
  - Co-Simulation
  - Mixed Approaches: Simulation + Analysis
  - Analytic Approach ➔ Next Talk

Conclusions
Mixed Approach – Simulation + Analysis

1. Using simulation to build up analytic models
   - 2 phases:
     1. construction phase: building analytic models by using simulation
     2. utilization phase: analytic model becomes independent, and can be used to (faster) evaluate new designs
   - e.g. Lahiri, SPADE

2. Coupled simulation + analysis in the same execution environment
   - interfaces between distinct domains: analysis / simulation
   - separation: what part of the system should be evaluated through analysis / simulation?
   - e.g. S. Kuenzli et al. DATE’06
Combining Simulation and Analysis

T1  →  T2  →  T3  →  T4

formal analysis  simulation  formal analysis

Trace of events:
- Average rate / equations / or more complex models ...
- Timing-event models

\[ \text{trace of events} \]

\[ \begin{align*}
2.5 & \quad t [\text{ms}] \\
\end{align*} \]
Combining Simulation and Formal Methods for System-Level Performance Analysis - S. Kuenzli et al. DATE’06

<table>
<thead>
<tr>
<th>Experiments</th>
<th>Evaluation Method</th>
<th>Evaluation Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPU₁ → T₁ Simulation → CPU₂ → T₂ Simulation</td>
<td>Simulation</td>
</tr>
<tr>
<td>2</td>
<td>CPU₁ → T₁ Formal Analysis → CPU₂ → T₂ Simulation</td>
<td>Hybrid (Formal Analysis + Simulation)</td>
</tr>
<tr>
<td>3</td>
<td>CPU₁ → T₁ Formal Analysis → CPU₂ → T₂ Formal Analysis</td>
<td>Formal Analysis*</td>
</tr>
</tbody>
</table>

*Formal Analysis – RTC Real Time Calculus
Outline

- Overview: Embedded Systems Design Flow
- Embedded Systems Modeling
- Design Space Exploration and Optimization
- Performance Evaluation

Conclusions
Conclusions

- **Design space exploration**
  - high diversity of solutions and parameters
  - “good solutions” have to be selected

- **Optimization**
  - multiple (conflicting) objectives

- **Performance evaluation**
  - requests:
    - high evaluation speed
    - high accuracy
  - problems:
    - resources have complex dependencies and interferences
    - computation, communication, HW, SW, OS, physical environment, … :
    - ⇒ ALL have to be taken into account!
Thank You!

E-Mail: iuliana.bacivarov@tik.ee.ethz.ch
Web: http://www.tik.ee.ethz.ch