SoC programming in the era of the Internet of Things, machine learning and emerging technologies

Jeronimo Castrillon
Chair for Compiler Construction (CCC)
TU Dresden, Germany

Groupement De Recherche SOC2: System On Chip, Systémes embarqués et Objets Connecté
Montpellier, France. June 20 2019
SoCs: Long history of specialization and interaction with environment
Incredible evolution over the last decades

SoC design and programming: Handling complexity

- Advances in design methodologies
- Model-based programming

![Diagram](image)
SoC design and programming: Handling complexity

Exciting innovations in
- Modeling languages
- Programming languages and compilers
- Costs models of hardware
- System simulators
- Design space exploration (DSE) methodologies

New challenges
- System dynamics (e.g., IoT)
- Ubiquity of machine learning workloads
- Complexity of emerging technologies
Dataflow and Hybrid DSE
Dataflow programming

- Graph representation of applications
  - Implicit repetitive execution of tasks
  - Good model for streaming applications
  - Good match for signal processing & multi-media

The why

- Explicit parallelism
- Often: Determinism
- Better analyzability (scheduling, mapping, optimization)
Dataflow compilation

- Plenty of research on
  - Language, compiler and mapping algorithms
  - Hardware modeling, performance estimation
  - Runtime systems
  - Code generation for heterogeneous multicores

[Springer14]
Example: HW acceleration and SW-defined radio

- Application: MIMO OFDM receiver
- Hardware
  - Platform 1: Baseline software
  - Platform 2: Optimized software
  - Platform 3: Optimized SW + HW

Achieved rate @ 100 MHz

1) bsp1 (sw, unoptimized) 7680
2) bsp2 (sw, optimized) 128000
3) bsp3 (hw) 1758241.75

[SDR'10, ALOG'11]
System dynamics

- Applications not so static anymore!

- Hybrid DSE: a compile and run-time approach
  - Enable adaptivity: malleable, multi-variant
  - Run-time predictability, robustness & isolation

**EUF** Pareto front @ compile-time

@runtime

System load after mapping
Data-level parallelism: Scalable and adaptive

- Change parallelism from the application specification
- Static code analysis to identify possible transformations (or via annotations)
- Implementation in FIFO library (semantics preserving)
Exploiting symmetries

- Intuition
  - SW: Some tasks/processes/actors may do the same
  - HW: Symmetric latencies (CoreX ↔ CoreY)
  - Symmetry: Allows transformations w/o changing the outcome

- No need to analyze all possible mappings (prune search space)
- Work on formalization via inverse semi-groups and efficient algorithms
Symmetries in Odroid: Example

Mappings

Architecture subgraphs

Equivalent mappings

Graph isomorphism

[CES’15, ACM TACO’17]
Flexible mappings: Generalized Tetris

- Given multiple canonical configs by compiler, select one at run-time
- Exploit mapping equivalences and similarities

[SCOPES'17b]
Flexible mappings: Run-time analysis

- Linux kernel: symmetry-aware
- Target: Odroid XU4 (big.LITTLE)
- Multi-application scenarios: audio filter (AF) and MIMO
  - 1x AF
  - 4 x AF
  - 2 x AF + 2 x MIMO
- 3 mappings to two processors
  - T1: Best CPU time
  - T2: Best wall-clock time
  - T3: GBM heuristic [Castrill12]

![Box plots showing CPU time, Energy, and Wall-clock time for different scenarios.](image)

[Caspers’17b]
Flexible mappings: Multi-application results (1)

More predictable performance

Comparable performance to dynamic mapping

[SCOPES'17b]
Robustness

- Static mappings, transformed or not, provide good predictability
- However: Many things out of control
  - Application data, unexpected interrupts, unexpected OS decisions

Can we reason about robustness of mapping to external factors?
Design centering: Find a mapping that can better tolerate variations while staying feasible.

- Studied field, in e.g., biology, circuit design or manufacturing systems.

- Currently
  - Using a bio-inspired algorithm
  - Robust against OS changes to the mapping

[SCOPES'17α]
Evaluation

- Analyze how robust the center really is
  - Perturbate mappings and check how often the constraints are missed
  - Signal processing applications on clustered ARM manycore and NoC manycore (16)

[SCOPES'17a]
Machine Learning & emerging technologies
ML revolution: Frameworks and architectures

- Many existing frameworks, e.g., TVM, Tensor Comprehensions, TensorFlow, ...

- Lots of traction in hardware architectures: TPU, V100, ...

- Lot’s of resources for training, less on inference on edge-devices!

Example flow: TVM [Chen, OSDI'18]
Domain-specific abstractions

- Commonality: Tensor expression languages
- Increase programmer’s productivity
- From compiler perspective: No abstraction toll
  - Easier access to information
  - Larger score for optimization

```plaintext
var input A : matrix &
var input u : tensorIN &

v = (A # A # A # u .
[[5 8] [3 7] [1 6]])

 RWDSL'18  

v_e = (A ⊗ A ⊗ A) u_e
```
Correct by construction

- Especially important in embedded systems
  - Correct by design
  - No abstraction leaks

- Current efforts in formal semantics for safe code generation

```python
A = placeholder((m, h), name='A')
B = placeholder((n, h), name='B')
k = reduce_axis((0, h), name='k')
C = compute((m, n), lambda i, j:
    sum(A[k, i] * B[k, j], axis=k))
```

```
[Array’19]
```
TeML: Results

- Extra control allows for new optimization (vs pluto): changing shapes
- General tensor semantics allows covering more benchmarks than TensorFlow

---

(GPCE'18)
Emerging technologies: Racetrack memories

- Racetrack memories: one of many future alternatives
  - Cf. STT/Re-RAM, hybrid architectures
  - Predicted extreme density at low latency
    - 3D nano-wires with magnetic domains
    - One port shared for many bits
    - Domains move at high speeds (1000 ms\(^{-1}\))

- Sequential: Game changer for current HW/SW stack
  - Memory management
  - Integration with other memory architectures
  - Data layout and allocation

[TVLSI'18, TVLSI'19]

[Parkin-Nature'15]
Compiler research on placement

- Compiler pass to reorganize placement of instructions
  - Instruction fetch is naturally sequential!
  - Layout instructions to reduce shift operations
    - [ISLPED'19]

- Compiler pass to reorganize data for higher-level objects
  - Variable allocation
    - [ArXiv'19]
  - Tensor allocation and program transformation
    - [LCTES'19]
Simulating RTMs

- **RTSim**: Configurable racetrack simulator
  - Allows running software benchmarks
  - Built on top of other simulator technology: NVMAIN 2, Gem5, SystemC, …

[Diagram]

https://github.com/tud-ccc/RTSim

[IEEE CAL’19]
Architecture and data layout optimization

- Architecture – software co-optimization
  - Embedded system for inference: RTM as scratchpad with pre-shifting and other optimizations

[LCTES'19]
Data-layout: Reduce the number of shifts
Latency comparison vs SRAM

- Un-optimized and naïve mapping: Even worse latency than SRAM
- 24% average improvement (even with very conservative circuit simulation)
Energy comparison vs SRAM

- Higher savings due to less leakage power
- 74% average improvement

[LCTES’19]
Discussion
Summary

- System dynamics: More complex in distributed IoT scenarios
  - Hybrid compile-runtime methodologies: Difficult balance, new interfaces
  - Strive at retaining time predictability

- New workloads (e.g., ML) + new techs: Harness domain-specific abstractions
  - More complex decision making (e.g., het. Memory systems)
  - Expression DSLs to ease high-level manipulation and transformation

- Exciting times for DSE, SW/HW co-design formal languages for emerging platforms (example: RTM-scratchpads)
References


