Simulation and estimation for MPSoC programming tools

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

Rapido Workshop. HiPEAC Conference
Amsterdam, January 21st 2015
Acknowledgements

- German Cluster of Excellence: Center for Advancing Electronics Dresden (www.cfaed.tu-dresden.de)
- Silexica Software Solutions GmbH
- Institute for Communication Technologies and Embedded Systems (ICE), RWTH Aachen: Prof. Leupers
Agenda

- MPSoc compilation
- Simulation & estimation for timing information
- Simulation: other use-cases
Agenda

- **MPSoC compilation**

- Simulation & estimation for timing information

- Simulation: other use-cases
Multi-Processor on Systems on Chip (MPSoCs)

- HW complexity
  - Increasing number of cores
  - Increasing heterogeneity
- Multi-cores everywhere
  - Ex.: Smartphones, tablets and e-readers
MPSoCs: SW productivity gap

- SW-productivity gap: complex SW for ever-increasing complex HW
  - Cannot keep pace with requirements
  - Cannot leverage available parallelism

Applications

Platforms

Source: Qualcomm, Texas Instruments

© J. Castrillon. Rapido workshop, Jan. 2015
### MAPS MPSoC Compiler

#### Application
- Sequential C legacy code
- Parallel KPN code

#### Architecture model
- PE, multi-tasking and communication APIs

**Eclipse integration**
- Sequential/parallel code profiling & tracing
- Code partitioning / Mapping and scheduling
- Automatic target C code generation
- Native C compilers for PEs (vendor compilers)

**Heterogeneous MPSoC**

**SW performance estimation or measurement on real HW**

**VP (ARM9, VLIW, RISC)**

**TI SoCs: OMAP, Keystone**

**Tegra 3 tablet**
Handling sequential C code

Purpose: Expose parallelism from a sequential specification

- Frontend: Build graph model of the application (dynamic DFA)
- Clustering: Group statements into potential parallel tasks
- Pattern search: Annotate graph with parallelism patterns
- Global analysis: Fix final configuration (implementation)

[DAC08, ASP-DAC10, SAMOS11]
Handling parallel code

- Input: Kahn Process Networks (KPN) & other flavors of dataflow models
  - A node (process) represents computation
  - An edge (channel) represents communication
- Output: Valid heterogeneous mapping (comply to constraints)
  - Process and channel mapping
  - Buffer sizing: Memory allocated for communication
Dataflow models: static vs. dynamic

- **Static: Synchronous dataflow models**
  - No “hardcoded” rates
  - More expressiveness
  - More difficult to analyze

- **KPNs (& DDF)**
  - No “hardcoded” rates
  - More expressiveness
  - More difficult to analyze
C Extension for KPNs

- FIFO Channels
  ```c
  typedef struct { int i; double d; } my_struct_t;
  __PNchannel my_struct_t C;
  __PNchannel int A = {1, 2, 3}; /* Initialization */
  __PNchannel short C[2], D[2], F[2], G[2];
  ```

- Processes & networks
  ```c
  __PNkpn AudioAmp __PNin(short A[2]) __PNout(short B[2])
  __PNparam(short boost)
  while (1)
  __PNin(A) __PNout(B) {
  for (int i = 0; i < 2; i++)
  B[i] = A[i]*boost;
  }
  __PNprocess Amp1 = AudioAmp __PNin(C) __PNout(F) __PNparam(3);
  __PNprocess Amp2 = AudioAmp __PNin(D) __PNout(G) __PNparam(10);
  ```
Dealing with dynamic behavior: tracing

- White model of processes: source code analysis and tracing

```plaintext
... for (;i < x;i++) {
    write(&c2);
    f1(...);
    read(&c1);
    f2(...);
    read(&c1);
    ...
```
Mapping: Trace-based heuristics

- Mapping & scheduling: Analyze traces and propose mapping
- Iterate: Improve mapping (if required)
Obtaining/generating timing information

- Computation time elapsed between events
  - For different processor types
  - Fine-grained information

- Parallel execution: Models for
  - OS
  - Communication
  - Task management and synchronization
Agenda

- MPSoC compilation
- **Simulation & estimation for timing information**
- Simulation: other use-cases
Timing information for MPSoC compilation

Sequential performance estimation
- Annotations & cost functions
- Abstract operation cost models
- Processor models/simulators
- Measurements

Parallel performance estimation
- Abstract cost models: OS, multi-tasking APIs, interconnect & memories
- System simulators/emulators
- Boards
Cost-tables and annotations

- Cost tables
  - Equivalence: Low-level IR → Assembly instructions
  - Coarse estimation of instruction-level parallelism

- Annotation: Coarser and parametrizable
  - Datasheet parametrizable equations for hardware accelerators

---

1. Points
2. Data format

---

FFT HW ACC

---

© J. Castrillon. Rapido workshop, Jan. 2015
Processor performance models

- Architecture description languages (e.g., LISA) & abstract processor models

![Diagram showing the process from SSA-IR (LLVM) to Processor model, including lowering, scheduling, and performance estimation.](image)

- Execution counts, branch stats, and execution traces

Courtesy: J. Eusse
Processor performance models (2)

- Abstract models for compiler emulation
  - Set of resources (functional units, register banks)
  - Set of operations (pipeline effects, SIMD, addressing modes, predicated execution)
  - SW-related costs (calling convention, register spilling, C-lib calls)

![Processor Model Diagram]

Conventions:
- Prologue: 1 + 2 * X_{arg}
- Epilogue: 4
- Branch_overhead: 3

External library costs:
- malloc: 9 + 0.3 * N
- fsqrt: 235

Courtesy: J. Eusse
Processor performance models: Results

Average gain:
248x (PD-RISC)
67x (TI DSPs)
(CA vs. profiling + estimation time)
Speeding-up system simulators

- Research on instruction set simulators: Interpretative, compiled, just-in-time compiled, dynamic binary translation, ...
- System simulators (i.e. Virtual Platforms): parallel SystemC
  - ParSC: conservative, synchronous simulation (delta-cycle), good for cycle-accurate simulation

![Diagram showing the process flow of ParSC with nodes labeled sc_start, eval, update, notify, and barrier sync.](CODES/ISSS10)

Courtesy: J. Weinstock
R. Leupers
Speeding-up system simulators (2)

- System simulators: parallel SystemC (Cont.)
  - **SCope**: conservative, time-decoupled simulation (quantum-based), good for TLM and instruction-accurate simulation

![Diagram of two threads with SC_start, lab, eval, update, notify, lookahead-based synchronization](image)

Courtesy: J. Weinstock, R. Leupers
1. Speedup vs. plain SystemC

Simulation host: Quad-Core simulation host (Intel i7 920), 4 threads

2. Speedup vs. Lookahead

Simulation host: Quad-Core simulation host (Intel i7 920), 4 threads

Courtesy: L. Murillo, J. Weinstock
EURETILE EU Project
Agenda

- MPSoC compilation
- Simulation & estimation for timing information
- Simulation: other use-cases
Virtual platforms: use cases

- Debugging: exploit observability and controllability
- Power/energy estimation: exploit abstraction
Debugging with virtual platforms

- Interactive debugging
  - Get snapshots of the system state
  - Full system stop
  - Track progress irrespective of mapping

Virtual platform

- Application & Mapping config
- KPN-level source information
- Debugging layer
- OS-descractor

Virtual platform

Internal state of the MPSoC schedule: assigned, blocked and running tasks
Deterministic replay and automatic bug exploration

Courtesy: L. Murillo
Debugging with virtual platforms (3)

- Controlling and swapping event orders

Simulation slowdown: **2-80x**
With no intelligent control: **~1600x**
(ARM Versatile simulator running an ocean simulation applications)

![Diagram showing event control and behavior trace](image-url)
ESL power estimation

- Callibrate abstract power models from hardware measurements
  - Run HW and ESL models with same input
  - Record traces for calibration

\[
P_{\text{est}} = S \cdot \alpha
\]

\[
\alpha := S^+ \cdot P_{\text{ref}}
\]

minimize mean squared error

Determining \( \alpha \) so that

\[
P_{\text{est}} \approx P_{\text{ref}}
\]

ESL power model

\[
\alpha = (a_1 \ldots a_N)^T
\]

\[
P_{\text{est}} = S \cdot \alpha
\]

Courtesy: S. Schürmans, R. Leupers
ESL power estimation (2)

- Evaluation: AMBA AXI and Custom NoC design (post P&R)
- Results
  - **Error < 22%** (accurate prediction of phases)
  - **Speedup 900x** (vs. low-level power simulation)

![Power Consumption of NoC 8x4 (scaled to 40nm, scenario case 1h)](image1)

![Power Consumption of AXI 8x16 (scaled to 40nm, scenario case 1h)](image2)

Courtesy: S. Schürmans, R. Leupers
Summary

- Discussed academic (and commercial) MPSoC programming tools

- The role of simulation/emulation technology for MPSoC compilers
  - Different technologies for different design phases
  - Deal with heterogeneous architectures
  - Deal with system-level simulators

- New, important use-cases for virtual platforms
  - Debugging
  - Power/energy estimation
References


