Portable Libraries and Programming Environments

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

Thematic session: Towards Portable Libraries for Hybrid Systems
Acknowledgements

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

- Silexica Software Solutions GmbH

- German Cluster of Excellence: Ultra High-Speed Mobile Information and Communication (UMIC), RWTH Aachen
Why new programming environments?

Complex applications for complex architectures ➔ SW-productivity gap

Already difficult for heterogeneous multi-core (programmable) systems
More difficult for systems with hardware accelerators and reconfigurable HW
Programming flows for heterogeneous MPSoCs

Code analysis, optimization and generation

Sequential
- Frontend
- Middle-end
- Backend

Parallel
- Frontend
- Middle-end
- Backend

SDR
- Frontend
- Middle-end
- Backend

Simulators:
- functional, early system level

SDR libraries

Flows for single applications

Multi-app. description

Analysis results: app # 1
... Analysis results: app # n

Scenario analysis

Flow for multiple applications

Runtime config.

Feasibility

Outputs
Parallel programming flow: Overview

Constraints → Application

MPSoC Compilation

Profiling, tracing analysis

Mapping (computation, communication)

Code generation (src-to-src)

C files → High-level debug-info

Architecture model

[SAMOS11]
Parallel programming flow: Overview

Constraints

Application

Well-defined MoC, e.g., Kahn Process Networks (KPN)

Profiling, tracing analysis

Models: performance/power estimation

Mapping (computation, communication)

Tracing to understand dynamics

Code generation (src-to-src)

High-performance, real-time, low energy

C files

Use target system’s APIs

High-level debug-info

[161] SAMOS11

Application: Dataflow and process networks

- Graph representation of applications
  - Process **communicate only** over FIFO buffers
  - Good model for streaming applications
  - Good match for signal processing & multi-media

- Stereo digital audio filter

```
src
```
```
fft_l  filter_l  ifft_l
```
```
fft_r  filter_r  ifft_r
```
```
sink
```

Allows control decision upon incoming data
Analysis: Instrumentation, profiling and tracing

- **Process white model**: source code analysis and tracing to deal with control

```c
... for (; i < x; i++) {
    write(&c2);
    f1(...);
    read(&c1);
    f2(...);
    read(&c1);
    ...
```
Performance models: Sequential and parallel

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

Parallel performance estimation
- Abstract cost models: OS, multitasking APIs, interconnect & memories
- System simulators/emulators
- Boards
Performance estimation: Computation

- Cost tables
  - Equivalence: Low-level IR → Assembly instructions
  - Coarse estimation of instruction-level parallelism
- Processor + compiler models
  - Set of resources, operations, low-level APIs

<table>
<thead>
<tr>
<th>FU1</th>
<th>FU2</th>
<th>FU3</th>
<th>Conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>ASR</td>
<td>LD</td>
<td>Prologue: 1+2*X_arg</td>
</tr>
<tr>
<td>SUB</td>
<td>LSL</td>
<td>ST</td>
<td>Epilogue: 4</td>
</tr>
<tr>
<td>MUL</td>
<td>LSR</td>
<td>CMP</td>
<td>Branch_over: 3</td>
</tr>
<tr>
<td>OR</td>
<td>ZEX</td>
<td>BR</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RF1 (16,32)</th>
<th>RF2 (8,32)</th>
<th>External library costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc: 9+0.3*N_size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fsqrt: 235</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance estimation: Communication

- Characterization of different communication APIs
- Example: TI Keystone II
Application mapping: Trace-based

- Heuristics that work on a graph-representation of multiple-traces
  - For buffer sizing
  - For mapping and scheduling
  - Often iterative for real-time

Traces + timing information from platform models

MPSOC Compilation

[DATE10, DAC12, IEEE-TII13]
Debugging: Layers and scripting for virtual platforms

- Interactive debugging
- Deterministic replay for bug exploration

Virtual platform

Application & Mapping config

KPN-level source information

Debugging layer

MPSoC compiler backend

Virtual platform

[LASCAS10, DATE14]
Parallel programming flow: Portable libraries

- Constraints
- Application

MPSoc Compilation

- Profiling, tracing analysis
- Mapping (computation, communication)
- Code generation (src-to-src)

C files

High-level debug-info

Portable libraries: How to add support for specialized HW/SW processes?

Architecture model

Parallel programming flow: Portable libraries

- Compilation
- Profiling, tracing analysis
- Mapping (computation, communication)
- Code generation (src-to-src)

Portable libraries: How to add support for specialized HW/SW processes?

Constraints
Application

C files
High-level debug-info

Architecture model
Solution approach: Nucleus project (for SW-defined Radio)

Nuclei Library

Waveform 1

... Non-Nucleus 1 Waveform x Non-Nucleus 2 Waveform n Transceiver Constraints

Possible nucleus mappings

Board support package:

Platform model + flavor descriptors

[SDR10, ALOG11, Castrillon14]
Nucleus programming flow: Inputs

- **Extended application specification**
  - Selected processes are algorithmic kernels with **algorithmic parameters**

- **Extended platform model**
  - SW/HW accelerated kernels and their **implementation parameters**
Extended mapping

- Find matching HW/SW support: Algorithmic to implementation parameters
- Integrate into trace-based framework

FFT HW

ACC

Points
Data format

Configuration, characterization

fft1
read src
read src
... 
write
write
\( \Delta t \)
demap
demap
t

...read ctrl read ffts read decod read decod read ctrl read ffts ...

...read ctrl read ffts read decod read decod read ctrl read ffts ...

3rd iter.
2nd iter.
1st iter.
finished \( n \) iters.
\( m \) iter.

\( \Delta t_1 \): all iterations
Nucleus programming flow: Code generation

- Bare-metal implementation

```c
int main() {
    ...  
    fft_cfg(...);
    ...  
    while (1) {
        ... 
    }
}
```

Control code

- Control+config (HW flavors)

- Flavor config (SW flavors)

- C code for non-nuclei

- cpn-cc (target)

- C compiler 1

- Linker 1

- SW-Flavor (unconfigured)

- Flavor library
Case study (brief): Portable performance

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

Achieved rate @ 100 MHz

- Platform 1: Baseline software
  - Rate: 7680 bps
- Platform 2: Optimized software
  - Rate: 128000 bps
- Platform 3: Optimized SW + HW
  - Rate: 1758241 bps

Matches manual design
Discussion towards library portability

- Programming flow extension to deal with HW/SW acceleration
- Desired library characterization (standards?)
  - For mapping
    - Times, data-types, rates (and more complex behaviors – time diagrams?)
  - For debugging
    - Debugging interfaces and true integration (so far: bare metal)
    - Interfacing patterns to be proven by trace analysis
  - For code generation (so far: bare metal)
    - OS integration, interaction with resource manager
- Not addressed: Reconfiguration & synthesis
References


