MPSoC Programming using the MAPS Compiler

Rainer Leupers, Jeronimo Castrillon,
Institute for Integrated Signal Processing Systems
RWTH Aachen University, Germany

ASP-DAC
Taipei, Jan. 2010
Motivation: MPSoCs and the Productivity Gap

- Multi-Processor Systems on Chip are a reality
  - Increased HW and SW complexity

- The productivity Gap: Requirements double every 10 months, HW/SW productivity every 2 years (Ecker, Mueller, Doemer, 2008)

→ Need better support for SW development in the MPSoC era
MAPS: Bridging the Productivity Gap

- **MAPS: MPSoC Application Programming Studio:**
  - Flexible input specification: 85% of embedded programmers use C/C++ ([www.eetimes.com](http://www.eetimes.com))
    - Legacy C-code and partitioning
    - Explicitly parallel C-like programming model (KPN)
  - Abstraction & retargetability:
    - Abstract APIs for early SW design
    - Code generation hides HW dependent SW
  - Functional validation:
    - Abstract simulator (HVP), no processor-specific tool chains involved
  - Mapping & Scheduling frameworks:
    - Manage the huge design space
  - Multiple application of different classes (real-time, best effort)

Source: Virtual Platform of Shapes RDT, SSS RWTH Aachen

Source: Chen, NTU, MPSoC 2008
Motivation

MAPS Overview

- Sequential and Parallel Flows
- Results
- Conclusions and Outlook
MAPS Flow Overview

- **Architecture model for retargetability**
- **Applications:**
  - C code for legacy
  - Parallel code to leverage a-priori knowledge
- **Analysis Phase:**
  - Profile-driven
  - Interactive
- **Mapping/Scheduling:**
  - Extensible
  - Cost-table driven performance estimation
- **Multiple Application**
  - Interaction through ACG
  - Composition approach
  - Different app. classes
Motivation

MAPS Overview

Sequential and Parallel Flows

Results

Conclusions and Outlook
Sequential Flow: How it started…

- **Sequential flow as presented in DAC 2008**

- **Key points:**
  1. Analysis phase: Traces for Dynamic Data Flow Analysis
  2. New analysis granularity: “Coupled” blocks as opposed to basic-blocks, functions,…
  3. Performance estimation: annotated 3-address-code IR via cost table
  4. Heuristic for hierarchical code partitioning

- **Simple code generation for TCT platform (TiTech, Tokyo)**

- **Execution on TCT virtual/real platform**
Sequential Flow: Improvements

- Analyze Strongly Connected Components (SCC): improves parallel efficiency, i.e. less PEs – similar execution time

- SCCs are recognized and a heuristic is used to merge blocks in order to improve the parallel efficiency

- Especial care of nested SCCs
Sequential Flow: Improvements (2)

- **Balance partitions of functions in different locations of the Call Graph**

In partition $i$

- A
- B
- C
- D

In partition $i-1$

- D'
- D''

PE: 1 2 3 4

<table>
<thead>
<tr>
<th>PE: 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>D''</td>
</tr>
</tbody>
</table>

Control flow

Data flow
- Dataflow programming models gain everyday more acceptance... Which to use?
  - HSDFs, SDFs, MRDFs, CFDF, KPN...

- MAPS programming model: Based on the Kahn Process Networks (KPN) Model of Computation (MoC)
  - Better expressiveness compared to other models
  - Simple semantics
  - More difficult to analyze and derive plausible schedules
    - Although comparable when handling multiple applications
Pragma extensions to represent KPN applications. Ex. RLE Decoding:

```c
__fifo int A, B;
#pragma maps process rle_dec, in(A), out(B), prefer(risc)
{ int cnt, val, i; } // Local variables to the process
{ // Process body: Repeated for ever
  cnt = A; // Reads first token: count
  val = A; // Reads second token: value
  for (i = 0; i < cnt; ++i) {
    B = val; // Outputs count times val
  }
}
```

GUI equivalent editor/viewer:
Parallel Flow

- **Parallel flow, details to appear in DATE Mar. 2010**

- **Key points:**
  1. Intermediate *pthread* code generation for tracing
  2. “*Sequentialized*” processes analyzed by traditional MAPS
  3. KPN tracer generates **KPN traces**
  4. Modular framework for scheduling and mapping: RR, RRWS, priority-based, FIFO,…
  5. TRM allows to compare different schedules

- The scheduler descriptor can be used to generate code directly
Parallel Flow: What is a KPN Trace?

- A sequential trace is a series of basic blocks.
- The KPN tracer identifies in which BBs channels were accessed.

- A trace is a sequence of segments, where a segment is a sequence of BBs with a channel access in its last BB.
Handling Multiple Applications

- **Applications organized into classes:**
  - Hard/soft real time
  - Best effort

- The Application Concurrency Graph (ACG) serves to describe the interaction among applications
  - A sub-graph of the ACG represent a use-case or multi-application scenario
  - Schedules for different applications are computed separately

- Use-case analysis via composition:
  - Composed utilization: \( h(t) = f(t) \oplus g(t) \)

Source: Chen, NTU, MPSoC 2008
Outline

- Motivation
- MAPS Overview
- Sequential and Parallel Flows

Results

- Conclusions and Outlook
Results: Sequential Flow

- New partitioning passes: a toy example

```c
1: for (i = 0; i < N1; i++) {
2:     for (j = 0; j < N2; j++) {
3:         ret_param1 = use0(array1, j);
4:         ret_param1 = use0(array1, j);
5:         ret_param1 = use0(array1, j);
6:     }
7:     ret_param2 = def1(array2, ret_param1);
8:     ret_param2 = use1(array2, i);
9:     ret_param3 = def1(array3, ret_param1);
10:    ret_param3 = use1(array3, i);
11:    ret_param2 = fool(array4, 0);
12:    ret_param3 = fool(array4, 1);
13: }
```

In toy example: 3.27X vs. 2.38X
In JPEG: 3.61→5.5X vs. 4.1→9.5X
Results: Parallel & Overall Flow

- The parallel flow has been tested on several real life applications:
  - MPEG2, JPEG, GSM, MIMO,…

- MAPS usability fully tested:
  - Parsing/tracing/profiling
  - Functional validation

- Later verification on different back-ends
  - TI-OMAP, TCT, OSIP

Source: www.ti.com
Outline

- Motivation
- MAPS Overview
- Sequential and Parallel Flows
- Results

Conclusions and Outlook
MAPS – A fairly complete tool set for MPSoC programming was presented:
- Sequential (C) & parallel (KPN) input specification
- Abstraction: functional simulation, APIs
- Mapping & scheduling of single and multiple applications to heterogeneous MPSoCs

... in a user friendly Eclipse-based GUI

Current & future work in MAPS
- C extensions instead of pragmas, aka: CPN
- Compiler development: CLANG, LLVM
- Better performance estimation techniques: TotalProf
- Improving mapping and scheduling heuristics
- Research on composability for KPNs
Thank You!
Questions??

Acknowledgments:
This work has been supported by the UMIC (Ultra High-Speed Mobile Information and Communication) research centre. www.umic.rwth-aachen.de

The team:

maps@iss.rwth-aachen.de