Profiling Based Architecture Optimization for Heterogeneous MPSoC

Rainer Leupers, Heinrich Meyr
RWTH Aachen University
Software for Systems on Silicon (SSS)
ISS MPSoC research focus areas

- Wireless+multimedia applications: heterogeneous MPSoC
  - Various processing elements, including ASIPs
  - (Application specific) NoC
- Architecture exploration/optimization via virtual prototypes

Spatial and temporal task mapping

application code

SW performance estimation

Design technology
- LISATek
- rASIP
Design case studies

NoC exploration
Overview

1. Customizable embedded processor design flow
2. MPSoC SW performance estimation framework
3. Future work
Today’s ASIP design technologies

- **ADL based (ASIP-from-scratch)**
  - E.g. LISATek, Target, Expression
  - Max. flexibility + efficiency, but significant design effort

- **Configurable processor cores**
  - E.g. Tensilica Xtensa, MIPS CorExtend, ARC Tangent
  - Pre-designed + pre-verified core
  - Efficiency via custom instruction set extensions (ISE)

- **Special case: reconfigurable processors**
  - E.g. Stretch
Primary goal: maximum application speedup under set of constraints
Complex optimization problem, huge design space, back annotation required

Optimal solution cannot be found within reasonable computation times

Our approach:

- Interactive ISE identification
- Tools generate AT curve
- For each point: use mix of ILP and heuristics to synthesize close-to-optimal ISEs
- User reviews and selects design points of interest for further fine-grained exploration
- RTL generation and logic synthesis
- Short turnaround times
Processor customization flow

(1) Application code analysis
- At C source level
- Execution characteristics + hot spots

(2) Custom instruction (CI) identification
- Optimal set of CIs to speed up hot spot under area and machine constraints

(3) CI implementation
- Interfacing with the configurable processor core
- Meet area and latency constraints
Code analysis by Micro-Profiler

- Fine grained C code profiling tool
- Instrumentation of 3-address C code intermediate format
- Makes all C operations visible
- Predicts compiler optimization effects
- Precise profiling of
  - Operator execution frequencies
  - Operation bit widths
  - Memory accesses/cache hits
  - ...
- Statistics export to ISE synthesis tool
(4) SW adaptation and tools generation

- Rewrite application C code to utilize CIs
- Adapt C compiler, ISS to support CIs

(5) HW architecture implementation

- Generate RTL HDL code for CIs
- Synthesize coprocessor for CIs

```c
/* Original Code of Function F */
extern unsigned long S[4][256];
unsigned long F(unsigned long x)
{
    unsigned short a;
    unsigned short b;
    unsigned short c;
    unsigned short d;
    unsigned long y;
    d = x & 0x00FF;
    x >>= 8;
    c = x & 0x00FF;
    x >>= 8;
    b = x & 0x00FF;
    x >>= 8;
    a = x & 0x00FF;
    y = S[0][a] + S[1][b];
    y = y * S[2][c];
    y = y + S[3][d];
    return y;
}
/* Modified Code with CIs inserted */
extern unsigned long S[4][256];
unsigned long F (unsigned long x)
{
    int _dummy_RS = 1, _dummy_RT = 1;
    unsigned long t68, t66, t64, t76;
    unsigned long t68, ldB28C1, ldB2800;
    unsigned long ldB41C2, ldB3C0;
    // C Code for Block 1
    t68 = C1_1 (x, y);
    ldB28C1 = *= load */ t66;
    ldB2800 = *= load */ t58;
    t84 = C1_1 (ldB2800, ldB28C1);
    t76 = C1_UnLoad_Internal (_dummy_RS, _dummy_RT, 3);
    ldB41C2 = *= load */ t75;
    ldB3C0 = *= load */ t84;
    t66 = C1_3 (dB3C0, ldB41C2);
    return t66;
```
ISE design tools user interface

Workbench approach
(3)+(4) CI implementation + SW tools adaptation

- **CoWare CorXpert tool**
  - C compiler, ISS retargeting, HDL generation for ISE
  - Currently support for MIPS CorExtend
  - Custom extensions to native MIPS tools
  - Used as „backend“ here

![Synthesized ISEs](image1)

![Verilog RTL](image2)

![C compiler, ISS](image3)
Preliminary ISE results

- Good speedup with few ISEs for simple kernels (e.g. DES)
- "Incremental" improvements required: runtime, estimation, ...
- In progress: H.264 reconfigurable MPSoC case study
Overview

1. Customizable embedded processor design flow
2. MPSoC SW performance estimation framework
3. Future work
MPSoC design flow

Application:

Task 1 -> Task 2 -> Task 3 -> Task 4 -> Task 5

MPSoC virtual prototype

MPSoC HW prototype
MPSoC exploration principles

- Divide and conquer
- Separate processing elements from communication
- Early SW performance estimation
Communication: CoWare Architect’s View Framework (AVF)

VPU: virtual processing unit

Enables modeling spatial and temporal task-to-PE mapping
VPU concept

- "Pre-architecture" exploration: abstract ISA modeling w/o ISS (native task C code execution)
- Abstract OS modeling (task context switch times)
- Problem: task timing and memory access (over NoC) modeling
Fast and accurate SW performance estimation

Statistical analysis

Manual time annotation

Instrumentation framework (μ-profiler)

ISS based execution,
(Processor defined & Compiler available!)

\[
time = N(100,10);
\]

\[
... 
\]

\[
a = 1;
...
\]

\[
cycle\_count += 100;
consume(cycle\_count);
... 
\]

\[
LOAD R1, #1;
MUL R1, #4;
ADD R2, R1;
LOAD R3, @R2;
... 
\]
VPU/NoC co-exploration

SW Task 1 (C-Code)  ...  SW Task n (C-Code)

Profiling and interception of memory accesses

VPU

MEM_ACCESS  ...  MEM_ACCESS

Communication Architecture

Memory  Memory  Memory

AVF SystemC TLM environment
Preliminary results for MIPS/Blowfish

- High accuracy of VPU vs. MIPS instruction accurate ISS
- Need to analyze accuracy vs. cycle accurate ISS

![Graph showing relative measurements for initialization, encode, and decode.]
Overview

1. Customizable embedded processor design flow
2. MPSoC SW performance estimation framework
3. Future work
Automated MPSoC VP refinement

Application in abstract task implementation (generic OS & basic comm.)

Early design space exploration system (VPU)

Executed on VPU together with generic OS

ISS based System (e.g. MIPS)

MPSoC tool flow (generates HW & SW)

Requirements fulfilled?

YES

NO

Implementation
Task mapping tools

- Drag & Drop

- Define as HW block

- Core A

- Core B

- HW block

- Configure communication & processing

- Drop communication scheme
Automated MPSoC VP refinement

System description

MPSoC tools

Replace generic OS & communication scheme

HW parts:

- HW
- NoC

SW parts:

- coreN.exe
- core1.exe
- e.g. MIPS & RTEMS

SW parts:

- e.g. MIPS & RTEMS
MPSoC programming: task graph generation/extraction

- How to obtain task graphs?
- Specification is not always given in a parallelized form, but e.g. as sequential C code
- TGFF is not the final solution
- Tool requirements
  - Semi-automatic parallelization
  - Help MPSoC programmers to extract the implicit parallelism inside an application
  - Compiler and profiling technology will be key for task graph generation from C/C++
Thank you!