Programming for adaptive and energy-efficient computing

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

International Conference on High Performance Compilation, Computing and Communications (HP3C)
March 23, 2017. Malaysia
TU Dresden – Department of Computer Science

Est. 1828, one of Germany’s TU9

Est. 1900, 26 professorships
Explore new technologies for information processing that overcome the limits of CMOS.

Advances in Materials Science

- Augmenting CMOS
- Complementing CMOS
- Replacing CMOS

Moore's Law

Saturation (it from bit)

Heterogeneous systems
This talk

Research at the chair for compiler construction
Agenda

- Contextualization
- HAEC overview
- Dataflow programming
- Flexible mappings
- Closing remarks
Agenda

- Contextualization
- **HAEC overview**
- Dataflow programming
- Flexible mappings
- Closing remarks
HAEC: Motivation

Energy Consumption with Reference to 2010 Grid

Current trend: 10x in 20 years

HAEC Challenge: enabling a massive-magnitude reduction!

© Prof. J. Castrillon. HP3C, 2017
HAEC: Highly-adaptive energy-efficient computing

Goal:
Minimize energy via multi-layer SW/HW adaptivity
The HAEC Box
HAEC: Project structure

HAEC-SW

Adaptable Software

HAEC-HW

Flexible hardware

Courtesy: Prof. Fröhlich, TU Dresden

HAEC: Project structure

Flexible hardware
Adaptable Software

HAEC-SW

HAEC-HW

© Prof. J. Castrillon. HP3C, 2017
HAEC: SW – Stack

High-Capacity Knowledge Processing Pipeline

Energy-Aware SW Architectures  Energy-Aware Service Execution

Energy-Efficient Languages & Compilers

Semantic Context Recognition

Energy-Aware In-Memory Storage

Energy-Aware Resource Management

Project Group A

© Prof. J. Castrillon. HP3C, 2017
Agenda

- Contextualization
- HAEC overview
- **Dataflow programming**
- Flexible mappings
- Closing remarks
Dataflow programming

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

- Flavors expose different trade-offs analyzability-expressiveness
  - Synchronous dataflow (SDF): Static schedules
  - Kahn process networks (KPN): Deterministic execution
  - Dynamic dataflow (DDF): Allows non-determinism and peeking into channels
Dataflow programming: Examples

- There are multiple tool flows, languages and frameworks

- Embedded
  - MAPS (now Silexica): KPNs onto heterogeneous multi-cores
  - TURNUS: From DDFs (in Cal) onto heterogeneous, reconfigurable systems
  - Ptolemy: Analysis of interactions between different models
  - Preesm: Parametrized SDFs for signal processing

- HPC
  - Maxeler: Dataflow cores on FPGA fabric
  - OpenMP 4.5 (OmpSs, Teraflux)
Dataflow programming flow

KPN Application

Architecture model

© Prof. J. Castrillon. HP3C, 2017

DMAs, semaphores, PMU

VLIW DSP

NoC

Peripheral

Communication support

HW queues

Network Processor

Packet DMA

MEM subsystem

Non-functional specification

Models of architectures used for analysis

Iterative analysis, (meta)heuristics, profiling information

Analysis

Synthesis

Core and memory assignment, scheduling, buffer sizing, ...

[Castrill11]

Source-to-source compilation

PNargs_ifft_r.ID = 6U;
PNargs_ifft_r.PNChannel_freq_coef = filtered_coef_right;
PNargs_ifft_r.PNnum_freq_coef = 0U;
PNargs_ifft_r.PNChannel_time_coef = sink_right;
PNargs_ifft_r.channel = 1;
sink_left = IPCImrfr.open(3, 1, 1);
sink_right = IPCImrfr.open(7, 1, 1);
PNargs_sink.ID = 7U;
PNargs_sink.PNChannel_in_left = sink_left;
PNargs_sink.PNChannel_in_right = sink_right;
PNargs_sink.PNChannel_in_right = sink_right;
PNargs Sink.PNChannel in right = 0U;
taskParams.arg0 = (xdc_UArg) & PNargs src;
taskParams.priority = 1;
Language: C for process networks

- **FIFO Channels**
  ```c
  typedef struct { int i; double d; } my_struct_t;
  __PNchannel my_struct_t S;
  __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);
  ```

[Sheng14]
Architecture model for heterogeneity

- System model including:
  - Topology, interconnect, memories
  - Computation: cost tables (as backup)
  - Communication: cost function (no contention)
- Example: Texas Instruments Keystone

```xml
<Platform>
  <Processors List="dsp0 dsp1 dsp2 dsp3 dsp4 dsp5 dsp6 dsp7"/>
  <Memories List="local_mem_dsp0 L2 local_mem_dsp1 L2 local_mem_dsp2 L2 local_snmem_dsp1 l2 local_snmem_dsp2 L2 local_snmem_dsp3 L2 local_snmem_dsp4 L2 local_mem_dsp3_DDR local_mem_dsp4_DDR local_mem_dsp5_DDR local_mem_dsp6_DDR local_mem_dsp7_DDR"/>
  <CommPrimitives List="PCIe_SL2 PCIe_DDR EDMA3_SL2 EDMA3_DDR EDMA3_PCIE"/>
</Platform>

<Processor Name="dsp0" CoreRef="DSPC66"/>
<Processors List="dsp0 dsp1 dsp2 dsp3 dsp4 dsp5 dsp6 dsp7"/>

<Memory>
  <LocalMemory Name="local_mem_dsp0 L2" Size="524288" BaseAddress_hex="00800000" ProcessorRef="dsp0"/>
</Memory>
```

[Oden13]
Analysis and synthesis: Overview

CPN application
Architecture model
Non-functional specification

Analysis: Instrumentation, profiling, tracing
Sequential performance estimation
Time-annotated traces
Mapping and scheduling
Parallel perf. estimation
Modify resources
Configurations
Tracing: Dealing with dynamic behavior

- KPNs do not have firing semantics
- **White model of processes**: source code analysis and tracing
- Tracing: instrumentation, token logging and event recording

```c
... for (; i < x; i++) {
    write(&c2);
    f1(...);
    read(&c1);
    f2(...);
    read(&c1);
    ...
```
Parallel performance estimation

- Discrete event simulator to evaluate a solution
  - Replay traces according to mapping
  - Extract costs from architecture file (NoC modeling, context switches, communication)
Trace-based synthesis

- Multiple heuristics available in the literature
  - Simple/fast heuristics based on the traces
  - Evolutionary algorithms

[Castrill010b, Castrill13]
Generating variants

- Modify resources available to the synthesis to generate variants
  
  - For homogeneous platforms: add processors
  - For heterogeneous: Try out different combination of resources

Mapping and scheduling

Parallel perf. estimation

Modify resources
Genetic algorithms provide better results in considerably more time.
Sample configurations for JPEG codec on Odroid XU4 (big.LITTLE)

- **Execution Time (in ms)**
- **Average Power (in W)**

**Image size:**
- 60x60: [Graph Image]
- 100x100: [Graph Image]
Agenda

- Contextualization
- HAEC overview
- Dataflow programming
- **Flexible mappings**
- Closing remarks
Flexible mappings

- Given multiple configurations from a compiler, select one at run-time
- Requires: reasoning about mapping *equivalences* and *similarities*
Mapping equivalences

- Requires analysis of both hardware and software symmetries
- Symmetry: Allows “transformation” w/o changing the “outcome”
- Interesting application of group theory (and inverse semi-groups)
- Boils down to the graph isomorphism problem

[Goens15, Goens17a]
Symmetries in Odroid: Example

Mappings

Architecture Subgraphs

Equivalent mappings

Graph isomorphism

Cortex A7
Cortex A15

[Goens17b]
Flexible mappings: Consequences

- **Scalability:** Automatically prune search space (needed for some metaheuristics)

- **Run-time adaptivity:** Use at runtime to select an equivalent mapping based on available resources
Flexible mappings: Run-time analysis

- Modified linux kernel: symmetrie-aware
- Target: Odroid XU4 (big.LITTLE)
- Multi-application scenarios: audio filter (AF) and MIMO
  - 1 x 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 [Castrillon12]

**Single AF**

---

[Goens17b]

© Prof. J. Castrillon. HP3C, 2017
Flexible mappings: Multi-application results (1)

Predictable performance

Comparable performance to dynamic mapping

instance 1 2 3 4

Mode CFS Dyn T1 T2 T3

[Goens17b]
Flexible mappings: Multi-application results (2)

instance 1 2 3 4

Energy [J]

CFS Dyn T1 T2 T3

Energy [J]

AF MIMO

Mode CFS Dyn T1 T2 T3

Good energy predictability as well

[Goens17b]

© Prof. J. Castrillon. HP3C, 2017
Agenda

- Contextualization
- HAEC overview
- Dataflow programming
- Flexible mappings
- **Closing remarks**
Closing remarks

- Overview of cfaed and HAEC projects
- HAEC
  - Adaptability in HW: Exploit communication paths (e.g., beam-forming, optical)
  - Adaptability in SW: At different levels of the stack / SW development process
- Languages and compilers
  - Dataflow and SW-synthesis methodologies
  - Heuristics to generate multiple variants
  - Run-time strategies to select and transform variants
- Now: Looking into multi-board and flexible communication schemes
Acknowledgements

- Silexica GmbH
- Collaborative Research Center for Highly-adaptive and Energy-efficient Computing (HAEC)
- German Cluster of Excellence: Center for Advancing Electronics Dresden (www.cfaed.tu-dresden.de)
References


