Fast and Accurate Evaluation of Embedded Applications for Manycore Systems

Porto Alegre

Felipe Rosa (UFRGS)
Luciano Ost (University of Leicester)
Ricardo Reis (UFRGS)
1. Introduction
OUTLINE

1. Introduction
2. Open Virtual Platforms
3. Timing Model
4. Energy Model
1. Introduction
2. Open Virtual Platforms
3. Timing Model
4. Energy Model
5. Exploration in Large Scale Systems
Introduction

High performance many-core systems are a reality up to 256 cores available today.
High performance many-core systems are a reality up to 256 cores available today.
High performance many-core systems are a reality up to 256 cores available today.
Introduction

High performance many-core systems are a reality up to 256 cores available today.

What about software development?
Software development challenges comprise:
- inter-processor communication protocol stacks definition
- OS porting and analysis
- parallel programming model porting
- drivers development

Software development costs is increasing…

Source: IBS 2013
Software development challenges comprise:
- inter-processor communication protocol stacks definition
- OS porting and analysis
- parallel programming model porting
- drivers development

Software development costs is increasing…

Simulation is the key tool for many-core research

Source: IBS 2013
Introduction

Full-system simulators are one good option for virtual platforms that emulate hardware behaviour, making software believe that it is running on a real physical hardware.

Support concomitant HW and SW development to improve time-to-market.

Examples of such simulators are:

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Accuracy</th>
<th>Supported processor architectures</th>
<th>License</th>
<th>Active support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simics</td>
<td>Functionally-accurate</td>
<td>Alpha, ARM, MIPS, PowerPC, SPARC, and x86</td>
<td>Private</td>
<td>Yes</td>
</tr>
<tr>
<td>PTLsim</td>
<td>Cycle-accurate</td>
<td>X86</td>
<td>Open</td>
<td>No</td>
</tr>
<tr>
<td>SimpleScalar</td>
<td>Cycle-accurate</td>
<td>Alpha, ARM, PowerPC, and x86</td>
<td>Open</td>
<td>No</td>
</tr>
<tr>
<td>GEM5</td>
<td>Cycle-accurate</td>
<td>Alpha, ARM, MIPS, PowerPC, SPARC, and x86</td>
<td>Open</td>
<td>Yes</td>
</tr>
<tr>
<td>QEMU</td>
<td>Instruction-accurate</td>
<td>ARM, MicroBraze, MIPS, PowerPC, SPARC, x86, and others</td>
<td>Open</td>
<td>Yes</td>
</tr>
<tr>
<td>OVPsim</td>
<td>Instruction-accurate</td>
<td>Alpha, ARC, ARM, MIPS, PowerPC, MicroBraze, and others</td>
<td>Open and Private</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Full-system simulators are one good option for virtual platforms that emulate hardware behaviour, making software believe that it is running on a real physical hardware. This support concomitant HW and SW development, improving time-to-market.

Examples of such simulators are:

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Accuracy</th>
<th>Supported processor architectures</th>
<th>License</th>
<th>Active support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simics</td>
<td>Functionally-accurate</td>
<td>Alpha, ARM, MIPS, PowerPC, SPARC, and x86</td>
<td>Private</td>
<td>Yes</td>
</tr>
<tr>
<td>PTLsim</td>
<td>Cycle-accurate</td>
<td>X86</td>
<td>Open</td>
<td>No</td>
</tr>
<tr>
<td>SimpleScalar</td>
<td>Cycle-accurate</td>
<td>Alpha, ARM, PowerPC, and x86</td>
<td>Open</td>
<td>No</td>
</tr>
<tr>
<td>GEM5</td>
<td>Cycle-accurate</td>
<td>Alpha, ARM, MIPS, PowerPC, SPARC, and x86</td>
<td>Open</td>
<td>Yes</td>
</tr>
<tr>
<td>QEMU</td>
<td>Instruction-accurate</td>
<td>ARM, MicroBraze, MIPS, PowerPC, SPARC, x86, and others</td>
<td>Open</td>
<td>Yes</td>
</tr>
<tr>
<td>OVPsim</td>
<td>Instruction-accurate</td>
<td>Alpha, ARC, ARM, MIPS, PowerPC, MicroBraze, and others</td>
<td>Open and Private</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Full-system simulators are one good option for virtual platforms that emulate hardware behaviour, making software believe that it is running on a real physical hardware. Support concomitant HW and SW development to improve time-to-market.

Examples of such simulators are:

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Accuracy</th>
<th>Supported processor architectures</th>
<th>License</th>
<th>Active support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simics</td>
<td>Functionally-accurate</td>
<td>Alpha, ARM, MIPS, PowerPC, SPARC, and x86</td>
<td>Private</td>
<td>Yes</td>
</tr>
<tr>
<td>PTLsim</td>
<td>Cycle-accurate</td>
<td>X86</td>
<td>Open</td>
<td>No</td>
</tr>
<tr>
<td>SimpleScalar</td>
<td>Cycle-accurate</td>
<td>Alpha, ARM, PowerPC, and x86</td>
<td>Open</td>
<td>No</td>
</tr>
<tr>
<td>GEM5</td>
<td>Cycle-accurate</td>
<td>Alpha, ARM, MIPS, PowerPC, SPARC, and x86</td>
<td>Open</td>
<td>Yes</td>
</tr>
<tr>
<td>QEMU</td>
<td>Instruction-accurate</td>
<td>ARM, MicroBraze, MIPS, PowerPC, SPARC, x86, and others</td>
<td>Open</td>
<td>Yes</td>
</tr>
<tr>
<td>OVPsim</td>
<td>Instruction-accurate</td>
<td>Alpha, ARM, MIPS, PowerPC, MicroBraze, and others</td>
<td>Open and Private</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Why OVP simulator?
Large number of processor architectures (ISAs) supported (e.g. MIPS, ARM, x86, PowerPC)

Simulation speeds up to 2200 MIPS relies on just-in-time (JIT) dynamic binary translation

Complete development environment with APIs
Open source license
Extensive documentation and active forum
Powerful debug environment

Limitation:
OVPsim provides instruction accuracy only, resulting in inaccurate software performance estimation

This work:
propose an integration of an timing and energy model into the OVPsim
Open Virtual Platform - OVP

Software stack separated from the simulator

Source: [Davidmann and Graham 2014]
Target instructions are translated to host machine binary code
ARM Assembly → x86 Assembly

Source: [Davidmann and Graham 2014]
Open Virtual Platform - OVP

Proposed Model Location

Callback

ICM API

Source: [Davidmann and Graham 2014]
Timing Model

Characteristics

- Instruction-driven timing model
- Developed on the basis of OVP APIs
- Run-time based approach
- ISA-based approach

Model called Watchdog

Advantages

- It avoids huge amount of memory
  - No trace files are required

Model is transparent to the software engineer
  - no pre- or post-processing application/software is required

The approach can be developed to other processor architectures

Timing Model

CPU
1. int main () {
2.  int a,b,c,i;
3.  a = 1;
4.  b = 1 + a;
5. for(i=0;i<10;i++)
6.  c = a/c;
7. return 0;
8.  }

Watchdog
- Hash table
- Parsed assembly code
- ISA timing information

Memory
- Memory address
  - 0X2546
  - 0X2548
  - 0X254A
  - 0X254C
  - 0X254E
  - 0X2550
  - 0X2552

BUS

Register bank
Timing Model

1. int main () {
2.     int a,b,c,i;
3.     a = 1;
4.     b = 1 + a;
5.     for(i=0;i<10;i++)
6.         c = a/c;
7.     return 0;
8. }

fetch (0x2546)
the parser module disassembles the binary code and identifies the instruction that must be executed
the parser module disassembles the binary code and identifies the instruction that must be executed

identified instruction is used as a hash table key to ascertain to which class such instruction belongs
the parser module disassembles the binary code and identifies the instruction that must be executed.

identified instruction is used as a hash table key to ascertain to which class such instruction belongs.

cycle count is computed and instruction is executed in the CPU.
the parser module disassembles the binary code and identifies the instruction that must be executed. The identified instruction is used as a hash table key to ascertain to which class such instruction belongs. Cycle count is computed and instruction is executed in the CPU.

What about accuracy?
Timing Model – Experimental Setup

FreeRTOS Key Features:
- kernel version V7.4.21
- POSIX wrapper (freertos_posix)

ARM Cortex-M4 Key Features:
- pipeline (3-stage + branch speculation)
- single precision floating point unit
- sleep modes for low power consumption

Board Key Features:
- 1 MB Flash
- 192 KB RAM
- SWD connector for programming and debugging
Timing Model – Experimental Setup

39 applications benchmarks from different research domains
MiBench, Mälardalen WCET, SPLASH-2, and in-house benchmarks

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adpcm</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>2</td>
<td>Barnes</td>
<td>SPLASH 2</td>
</tr>
<tr>
<td>3</td>
<td>BasicMath</td>
<td>MiBench</td>
</tr>
<tr>
<td>4</td>
<td>Bfsh</td>
<td>in-house production</td>
</tr>
<tr>
<td>5</td>
<td>BinarySearch</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>6</td>
<td>BitManipulation</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>7</td>
<td>Bubble</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>8</td>
<td>Compress</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>9</td>
<td>Counts</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>10</td>
<td>Crc</td>
<td>in-house production</td>
</tr>
<tr>
<td>11</td>
<td>Dijkstra</td>
<td>MiBench</td>
</tr>
<tr>
<td>12</td>
<td>Edn</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>13</td>
<td>Expint</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>14</td>
<td>Factorial</td>
<td>in-house production</td>
</tr>
<tr>
<td>15</td>
<td>Fdct</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>16</td>
<td>Fft</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>17</td>
<td>Fib</td>
<td>in-house production</td>
</tr>
<tr>
<td>18</td>
<td>Fir</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>19</td>
<td>Hanoi</td>
<td>in-house production</td>
</tr>
<tr>
<td>20</td>
<td>Harm</td>
<td>in-house production</td>
</tr>
<tr>
<td>21</td>
<td>InsertSort</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>22</td>
<td>Jfdctint</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>23</td>
<td>Lms</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>24</td>
<td>Lu</td>
<td>SPLASH 2</td>
</tr>
<tr>
<td>25</td>
<td>MatrixInver</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>26</td>
<td>Mdc</td>
<td>in-house production</td>
</tr>
<tr>
<td>27</td>
<td>Patricia</td>
<td>MiBench</td>
</tr>
<tr>
<td>28</td>
<td>Peakspeed</td>
<td>Impearas</td>
</tr>
<tr>
<td>29</td>
<td>Petri</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>30</td>
<td>Prime</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>31</td>
<td>Qsolver</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>32</td>
<td>Qsort</td>
<td>MiBench</td>
</tr>
<tr>
<td>33</td>
<td>Sha</td>
<td>MiBench</td>
</tr>
<tr>
<td>34</td>
<td>Statistic</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>35</td>
<td>Stringsearch</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>36</td>
<td>Sw</td>
<td>SPLASH 2</td>
</tr>
<tr>
<td>37</td>
<td>Tree</td>
<td>in-house production</td>
</tr>
<tr>
<td>38</td>
<td>Ud</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>39</td>
<td>Usqrt</td>
<td>Mälardalen WCET</td>
</tr>
</tbody>
</table>
Timing Model - Results

The graph shows the execution time (clock cycles) on a log scale for different metrics:
- Mismatch sequential
- Timing CPU model (sequential implementation)
- Referential Board

The vertical axis represents the execution time (clock cycles), while the horizontal axis represents the samples from 1 to 39.
mismatch is below 8% in 34 out of the 39 adopted benchmarks
Timing Model - Scalability

Scenario exploration with up to 1000 CPUs
Each CPU has one Watchdog associated
Timing Model - Scalability

Each CPU has one Watchdog associated...
Scenario exploration with up to 1000 CPUs
Each CPU has one Watchdog associated

Timing Model - Scalability

Timing CPU model
Timing CPU model thread extension

million instructions per second (MIPS)

Number of CPU’s

0 200 400 600 800 1000
Scenario exploration with up to 1000 CPUs
Each CPU has one Watchdog associated
Scenario exploration with up to 1000 CPUs
Each CPU has one Watchdog associated

Timing Model - Scalability

Reaming stable

Around 1.8 MIPS
Energy Model

The instruction energy cost information is not available

Characterization phase
Instructions are organized in groups according to their energy cost similarity
Less complexity/computation during the characterization and simulation phases

Reference CPU
PLASMA Core - MIPS Architecture 3-Stage pipeline
100 MHz 65nm low power library from ST Microelectronics

Using Cadence Tools
static and dynamic energy

```c
int main () {
    int a,b,c,i;
    a = 1;
    b = 1 + a;
    for(i=0;i<10;i++)
    c = a/c;
    return 0;
}
```

**WATCHDOG**

- Disassembler: 0X2550 str r3, [r7, #20]
- HashTable:
  - Class: Arithmetic
  - Keys: add, addu, sub
  - Energy: 0.06365281nJ
  - Class: Load Store
  - Keys: lw, lh, sw, lb
  - Energy: 0.08791465nJ
  - Class: Logical
  - Keys: and, or, xor, ori
  - Energy: 0.04407438nJ

**MEMORY**

- Memory address:
  - 0X00C0: 0X2546
  - 0X00C2: 0X2548
  - 0X00C6: 0X254A
  - 0X00CA: 0X254C
  - 0X00CE: 0X254E
  - 0X00D2: 0X2550
  - 0X00D4: 0X2552

**OVP CPU MODEL**

- Fetch (PC 0XD2)
int main () {
  int a, b, c, i;
  a = 1;
  b = 1 + a;
  for(i=0;i<10;i++)
    c = a/c;
  return 0;
}

1. fetch (PC 0XD2)
the parser module disassembles the binary code and identifies the instruction that must be executed
the parser module disassembles the binary code and identifies the instruction that must be executed

identified instruction is used as a hash table key to ascertain to which class such instruction belongs
the parser module disassembles the binary code and identifies the instruction that must be executed.

identified instruction is used as a hash table key to ascertain to which class such instruction belongs.

The energy cost is computed and the instruction is executed in the CPU.
Energy Model

Characterization Flow

- Profiling benchmark
- Cross-Compiler
- .elf binary object code
- OVPSim
- Number of Instructions
- RTL Compiler
- Report Mean Power
- Energy per instruction
- PLASMA Netlist
- Incisive
- .tcl
- .sdc
- Execution time
Energy Model

Characterization Flow

1 - Benchmark Conception
Energy Model

Characterization Flow
1. Benchmark Conception
2. Activity Measurement
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
Energy Model

Characterization Flow
1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
Energy Model

Characterization Flow

1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
3 - Power Acquisition
Energy Model

Characterization Flow

1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
4. Energy Calculation
Energy Model

Characterization Flow
1 - Benchmark Conception
2 - Activity Measurement
3 - Power Acquisition
4 - Energy Calculation
**Energy Model**

Characterization Flow

1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
4. Energy Calculation

<table>
<thead>
<tr>
<th>Groups</th>
<th>Power (mW)</th>
<th>Exec Time (us)</th>
<th>Energy (nJ)</th>
<th># of inst</th>
<th>Energy per Inst (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>6,456</td>
<td>342,755</td>
<td>2212,826</td>
<td>34764</td>
<td>0,0636528098</td>
</tr>
<tr>
<td>Jump</td>
<td>6,046</td>
<td>102,600</td>
<td>620,320</td>
<td>10224</td>
<td>0,0606728873</td>
</tr>
<tr>
<td>Load-Store</td>
<td>4,094</td>
<td>1042,800</td>
<td>4269,223</td>
<td>48561</td>
<td>0,0879146476</td>
</tr>
<tr>
<td>Logical</td>
<td>4,469</td>
<td>349,735</td>
<td>1562,966</td>
<td>35462</td>
<td>0,0440743815</td>
</tr>
<tr>
<td>Move</td>
<td>3,129</td>
<td>480,725</td>
<td>1504,189</td>
<td>39363</td>
<td>0,0382132593</td>
</tr>
<tr>
<td>NOP</td>
<td>2,141</td>
<td>257,155</td>
<td>550,569</td>
<td>26130</td>
<td>0,0210703733</td>
</tr>
<tr>
<td>Shift</td>
<td>3,824</td>
<td>298,735</td>
<td>1142,363</td>
<td>30362</td>
<td>0,0376247494</td>
</tr>
</tbody>
</table>

Average energy = \( \frac{\text{execution time (s)} \times \text{power (mW)}}{\text{executed instructions}} \) (nJ)
Energy Model

Characterization Flow
1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
4. Energy Calculation

<table>
<thead>
<tr>
<th>Groups</th>
<th>Power (mW)</th>
<th>Exec Time (us)</th>
<th>Energy (nJ)</th>
<th># of inst</th>
<th>Energy per Inst (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>6,456</td>
<td>342,755</td>
<td>2212,826</td>
<td>34764</td>
<td>0.0636528098</td>
</tr>
<tr>
<td>Jump</td>
<td>6,046</td>
<td>102,600</td>
<td>620,320</td>
<td>10224</td>
<td>0.0606728873</td>
</tr>
<tr>
<td>Load-Store</td>
<td>4,094</td>
<td>1042,800</td>
<td>4269,223</td>
<td>48561</td>
<td>0.0879146476</td>
</tr>
<tr>
<td>Logical</td>
<td>4,469</td>
<td>349,735</td>
<td>1562,966</td>
<td>35462</td>
<td>0.0440743815</td>
</tr>
<tr>
<td>Move</td>
<td>3,129</td>
<td>480,725</td>
<td>1504,189</td>
<td>39363</td>
<td>0.0382132593</td>
</tr>
<tr>
<td>NOP</td>
<td>2,141</td>
<td>257,155</td>
<td>550,569</td>
<td>26130</td>
<td>0.0210703733</td>
</tr>
<tr>
<td>Shift</td>
<td>3,824</td>
<td>298,735</td>
<td>1142,363</td>
<td>30362</td>
<td>0.0376247494</td>
</tr>
</tbody>
</table>

Characterization Flow Diagram:

- **Profiling benchmark**
  - Cross-Compiler
    - .elf binary object code
      - Incisive
        - OVPSim
          - Number of Instructions
            - RTL Compiler
              - Report Mean Power
                - Energy per instruction
              - Execution time
            - .tcf
            - .sdc
Energy Model

Characterization Flow

1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
4. Energy Calculation

<table>
<thead>
<tr>
<th>Groups</th>
<th>Power (mW)</th>
<th>Exec Time (us)</th>
<th>Energy (nJ)</th>
<th># of inst</th>
<th>Energy per Inst (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>6,456</td>
<td>342,755</td>
<td>2212,826</td>
<td>34764</td>
<td>0,0636528098</td>
</tr>
<tr>
<td>Jump</td>
<td>6,046</td>
<td>102,600</td>
<td>620,320</td>
<td>10224</td>
<td>0,0606728873</td>
</tr>
<tr>
<td>Load-Store</td>
<td>4,094</td>
<td>1042,800</td>
<td>4269,223</td>
<td>48561</td>
<td>0,0879146476</td>
</tr>
<tr>
<td>Logical</td>
<td>4,469</td>
<td>349,735</td>
<td>1562,966</td>
<td>35462</td>
<td>0,0440743815</td>
</tr>
<tr>
<td>Move</td>
<td>3,129</td>
<td>480,725</td>
<td>1504,189</td>
<td>39363</td>
<td>0,0382132593</td>
</tr>
<tr>
<td>NOP</td>
<td>2,141</td>
<td>257,155</td>
<td>550,569</td>
<td>26130</td>
<td>0,0210703733</td>
</tr>
<tr>
<td>Shift</td>
<td>3,824</td>
<td>298,735</td>
<td>1142,363</td>
<td>30362</td>
<td>0,0376247494</td>
</tr>
</tbody>
</table>
Energy Model

Characterization Flow

1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
4. Energy Calculation

<table>
<thead>
<tr>
<th>Groups</th>
<th>Power (mW)</th>
<th>Exec Time (us)</th>
<th>Energy (nJ)</th>
<th># of inst</th>
<th>Energy per Inst (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>6,456</td>
<td>342,755</td>
<td>2212,826</td>
<td>34764</td>
<td>0.0636528098</td>
</tr>
<tr>
<td>Jump</td>
<td>6,046</td>
<td>102,600</td>
<td>620,320</td>
<td>10224</td>
<td>0.0606728873</td>
</tr>
<tr>
<td>Load-Store</td>
<td>4,094</td>
<td>1042,800</td>
<td>4269,223</td>
<td>48561</td>
<td>0.0879146476</td>
</tr>
<tr>
<td>Logical</td>
<td>4,469</td>
<td>349,735</td>
<td>1562,966</td>
<td>35462</td>
<td>0.0440743815</td>
</tr>
<tr>
<td>Move</td>
<td>3,129</td>
<td>480,725</td>
<td>1504,189</td>
<td>39363</td>
<td>0.0382132593</td>
</tr>
<tr>
<td>NOP</td>
<td>2,141</td>
<td>257,155</td>
<td>550,569</td>
<td>26130</td>
<td>0.0210703733</td>
</tr>
<tr>
<td>Shift</td>
<td>3,824</td>
<td>298,735</td>
<td>1142,363</td>
<td>30362</td>
<td>0.0376247494</td>
</tr>
</tbody>
</table>

5. Energy Model Creation
**Energy Model**

### Characterization Flow

1. Benchmark Conception
2. Activity Measurement
3. Power Acquisition
4. Energy Calculation

<table>
<thead>
<tr>
<th>Groups</th>
<th>Power (mW)</th>
<th>Exec Time (us)</th>
<th>Energy (nJ)</th>
<th># of inst</th>
<th>Energy per Inst (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>6,456</td>
<td>342,755</td>
<td>2212,826</td>
<td>34764</td>
<td>0.0636528098</td>
</tr>
<tr>
<td>Jump</td>
<td>6,046</td>
<td>102,600</td>
<td>620,320</td>
<td>10224</td>
<td>0.0606728873</td>
</tr>
<tr>
<td>Load-Store</td>
<td>4,094</td>
<td>1042,800</td>
<td>4269,223</td>
<td>48561</td>
<td>0.0879146476</td>
</tr>
<tr>
<td>Logical</td>
<td>4,469</td>
<td>349,735</td>
<td>1562,966</td>
<td>35462</td>
<td>0.0440743815</td>
</tr>
<tr>
<td>Move</td>
<td>3,129</td>
<td>480,725</td>
<td>1504,189</td>
<td>39363</td>
<td>0.0382132593</td>
</tr>
<tr>
<td>NOP</td>
<td>2,141</td>
<td>257,155</td>
<td>550,569</td>
<td>26130</td>
<td>0.0210703733</td>
</tr>
<tr>
<td>Shift</td>
<td>3,824</td>
<td>298,735</td>
<td>1142,363</td>
<td>30362</td>
<td>0.0376247494</td>
</tr>
</tbody>
</table>

5. Energy Model Creation

---

The flowchart illustrates the process from benchmark conception to energy model creation, including steps for activity measurement, power acquisition, and energy calculation. The table provides detailed energy metrics for various groups, demonstrating the energy consumption per instruction for different operations.
Energy Model – Experimental Setup

Benchmarks

19 applications from different research domains
WCET and other benchmarks created in house
Model estimation compared with a gate-level implementation (PLASMA)

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BFSH</td>
<td>In-House production</td>
</tr>
<tr>
<td>B</td>
<td>BinarySearch</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>C</td>
<td>BitManipulation</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>D</td>
<td>Bubble</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>E</td>
<td>Counts</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>F</td>
<td>Crc</td>
<td>In-House production</td>
</tr>
<tr>
<td>G</td>
<td>Edn</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>H</td>
<td>Expint</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>I</td>
<td>Factorial</td>
<td>In-House production</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>Fft</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>K</td>
<td>Fib</td>
<td>In-House production</td>
</tr>
<tr>
<td>L</td>
<td>Hanoi</td>
<td>In-House production</td>
</tr>
<tr>
<td>M</td>
<td>Harm</td>
<td>In-House production</td>
</tr>
<tr>
<td>N</td>
<td>InsertSort</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>O</td>
<td>MatrixInver</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>P</td>
<td>Mdc</td>
<td>In-House production</td>
</tr>
<tr>
<td>Q</td>
<td>Peakspeed</td>
<td>Imperas</td>
</tr>
<tr>
<td>R</td>
<td>Ud</td>
<td>Mälardalen WCET</td>
</tr>
<tr>
<td>S</td>
<td>Usqrt</td>
<td>Mälardalen WCET</td>
</tr>
</tbody>
</table>
Benchmarks

- 19 applications from different research domains
- MiBench and other benchmarks created in house

Model estimation compared with gate-level
Mismatch is below 6% in 15 out of the 19 adopted benchmarks

**Energy Model – Results**

Mismatch is below 6% in 15 out of the 19 adopted benchmarks
Energy Model – Results

Mismatch is below 6% in 15 out of the 19 adopted benchmarks
Benchmarks
• 19 applications from different research domains
• MiBench and other benchmarks created in house

Mismatch is below 6% in 15 out of the 19 adopted benchmarks
Energy Model – Results

Mismatch is below 6% in 15 out of the 19 adopted benchmarks

Benchmarks
- 19 applications from different research domains
- MiBench and other benchmarks created in house
- Model estimation compared with gate-level level

Mismatch is below 6% in 15 out of the 19 adopted benchmarks

Energy Model – Results
Benchmarks

- 19 applications from different research domains
- MiBench and other benchmarks created in house

Mismatch is below 6% in 15 out of the 19 adopted benchmarks

What about speedup gain?
Comparing each benchmark watchdog estimation execution time with gate-level execution time

- Relative gain varying from 461 to 1577
- Mean relative gain 1118
- Larger execution time, larger relative speed gain

**Energy Model – Relative Speedup**
Comparing each benchmark watchdog estimation execution time with gate-level execution time

Relative gain varying from 461 to 1577
Mean relative gain 1118
Larger execution time, larger relative speed gain
Comparing each benchmark watchdog estimation execution time with gate-level execution time

Relative gain varying from 461 to 1577
Mean relative gain 1118
Larger execution time, larger relative speed gain
The proposed instruction-driven energy model was integrated into a NoC-based MPSoC model proposed in [Mandelli et al. 2013].

Evaluate the mapping process cost of three different heuristics: Nearest Neighbor (NN), first free (FF) and LECDN. Scenarios using an 8x8 MPSoC size instance with 4x4 clusters. Only the heuristic algorithm was observed.

5 applications instances:
- 4 partial MPEG decoder containing 5 tasks
- 1 DTW containing 10 tasks
The proposed instruction-driven energy model was integrated into a NoC-based MPSoC model proposed in [Mandelli et al. 2013]

Evaluate the mapping process cost of three different heuristics Nearest Neighbor (NN), first free (FF) and LECDN

Scenarios using an 8x8 MPSoC size instance with 4x4 clusters

Only the heuristic algorithm was observed

5 applications instances:

4 partial MPEG decoder containing 5 tasks

1 DTW containing 10 tasks

```c
HeuristicMapper()
{
    ...
    Heuristic
    Algorithm
    ...
}
```
The proposed instruction-driven energy model was integrated into a NoC-based MPSoC model proposed in [Mandelli et al. 2013]

Evaluate the mapping process cost of three different heuristics: Nearest Neighbor (NN), first free (FF) and LECDN Scenarios using an 8x8 MPSoC size instance with 4x4 clusters

Only the heuristic algorithm was observed

5 applications instances:
- 4 partial MPEG decoder containing 5 tasks
- 1 DTW containing 10 tasks

![Bar chart showing energy analysis (nano Joules) for FF, NN, and LECDN]
Inclusion of fast and accurate timing and energy models into OVPsim

Extensive evaluation of both models considering several benchmarks, while comparing it to a gate-level simulation (energy model) and comparing it to a real system implementations (timing model)

Approach is ISA/CPU-oriented, thus everything is transparent to the software engineer

Programmers can use the same simulator to have a fast simulation and accurate software performance evaluation
Fast and Accurate Evaluation of Embedded Applications for Manycore Systems

March - Porto Alegre

Felipe Rosa (UFRGS)
Luciano Ost (University of Leicester)
Ricardo Reis (UFRGS)
Conclusion - Limitations

No cache or memory model

Achieved speed is relative low due to the use of callbacks

Improve overall model accuracy
Conclusion - Limitations

No cache or memory model

- OVP provides several cache models with support to capture cache miss as events, similar to RAM memories

Achieved speed is relative low due to the use of callbacks

Improve overall model accuracy
Conclusion - Limitations

No cache or memory model

OVP provides several cache models with support to capture cache miss as events, similar to RAM memories

Achieved speed is relative low due to the use of callbacks

Porting the proposed model to the OVPSim morphing phase

Improve overall model accuracy
Conclusion - Limitations

No cache or memory model

OVP provides several cache models with support to capture cache miss as events, similar to RAM memories

Achieved speed is relative low due to the use of callbacks

Porting the proposed model to the OVPSim morphing phase

Improve overall model accuracy

Study load and stores in more rare patterns
Conclusion - Limitations

No cache or memory model
  OVP provides several cache models with support to capture cache miss as events, similar to RAM memories

Achieved speed is relative low due to the use of callbacks
  Porting the proposed model to the OVPSim morphing phase

Improve overall model accuracy
  Study load and stores in more rare patterns
  Enhance the division and multiplication algorithm estimation
Conclusion - Limitations

No cache or memory model
OVP provides several cache models with support to capture cache miss as events, similar to RAM memories

Achieved speed is relative low due to the use of callbacks
Porting the proposed model to the OVPSim morphing phase

Improve overall model accuracy
Study load and stores in more rare patterns
Enhance the division and multiplication algorithm estimation

Complex processor architectures as Out-Order or Super Pipeline
Conclusion - Limitations

No cache or memory model
OVP provides several cache models with support to capture cache miss as events, similar to RAM memories

Achieved speed is relative low due to the use of callbacks
Porting the proposed model to the OVPSim morphing phase

Improve overall model accuracy
Study load and stores in more rare patterns
Enhance the division and multiplication algorithm estimation

Complex processor architectures as Out-Order or Super Pipeline

Memory access energy cost
Conclusion - Limitations

No cache or memory model

OVP provides several cache models with support to capture cache miss as events, similar to RAM memories

Achieved speed is relative low due to the use of callbacks

Porting the proposed model to the OVPSim morphing phase

Improve overall model accuracy

Study load and stores in more rare patterns
Enhance the division and multiplication algorithm estimation

Complex processor architectures as Out-Order or Super Pipeline

Memory access energy cost
the model already account the number of memory access,
Conclusion - Limitations

No cache or memory model

OVP provides several cache models with support to capture cache miss as events, similar to RAM memories.

Achieved speed is relative low due to the use of callbacks

Porting the proposed model to the OVPSim morphing phase

Improve overall model accuracy

Study load and stores in more rare patterns
Enhance the division and multiplication algorithm estimation

Complex processor architectures as Out-Order or Super Pipeline

Memory access energy cost

The model already account the number of memory access, both read and write.