Estimation of Design Parameters

SS 2010
Hw/Sw Codesign

Christian Plessl
Motivation

- Estimation is to determine design parameters without implementing the system
  - supports design decisions
  - enables design space exploration
  - forms the basis for system optimizations
Overview

- Parameters of estimation methods
- Estimation of hardware metrics
- Estimation of software metrics
### Parameters of Estimation Methods

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Fidelity</th>
<th>Effort (Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Fine</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>
Accuracy

- **Definition**: Let $E(D)$ be an estimated and $M(D)$ an exact (measured) metrics of an implementation $D$. The accuracy $A$ of the estimation is given by:

$$A = 1 - \frac{|E(D) - M(D)|}{M(D)}$$
Fidelity

- **Definition**: Let \( D = \{D_1, D_2, \ldots, D_n\} \) be a set of implementations. The fidelity \( F \) of an estimation method is given by:

\[
F = 100 \cdot \frac{2}{n(n-1)} \cdot \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mu_{i,j}
\]

\[
\mu_{i,j} = \begin{cases} 
1 & \text{if } (E(D_i) > E(D_j) \land M(D_i) > M(D_j)) \lor \\
 & (E(D_i) < E(D_j) \land M(D_i) < M(D_j)) \lor \\
 & (E(D_i) = E(D_j) \land M(D_i) = M(D_j)) \\
0 & \text{else}
\end{cases}
\]
Fidelity - Example

metrics  

estimated  
measured

fidelity = 100 %

fidelity = 33.3 %
Metrics

• Performance
  – hardware: clock period, latency, execution time, throughput
  – software: execution time, worst-case execution time, throughput
  – communication: bit rate, communication time, throughput

• Cost

• Power consumption, energy requirements

• Reliability

• Testability

• Time-to-market

• ...
Overview

• Parameters of estimation methods
  • Estimation of hardware metrics
  • Estimation of software metrics
Hardware - Performance

• Clock period $T$
  – depends on technology, resources

• Latency $L$
  – given by the number of clock steps

• Execution time
  $T_{\text{ex}} = T \times L$

• Throughput
  $R = 1 / T_{\text{ex}}$
Example (1)

Clock period $T = 380$ ns

Latency $L = 1$

Execution time $T_{ex} = 380$ ns

Resources: 2 MUL, 4 ADD
Example (2)

clock period $T = 150$ ns

latency $L = 4$

execution time $T_{ex} = 600$ ns

resources: 1 MUL, 1 ADD
clock period $T = 80$ ns

latency $L = 5$

execution time $T_{ex} = 400$ ns

resources: 1 MUL, 1 ADD
Pipelining

pipelining with $P$ stages of equal length:

$$R = \frac{P}{T_{ex}}$$
Estimation of the Clock Period

- Functional units (operators) $v_k$ with delays $del(v_k)$
  - method of the maximum operator delay
    \[
    T = \max_{k} (del(v_k))
    \]
  - clock slack minimization method
    - search in the interval $[T_{\text{min}} \ldots T_{\text{max}}]$ for the clock period $T$ that maximizes the utilization (minimizes clock slack)
  - ILP search
Clock Slack

\[
slack(T, v_k) = \left( \left\lfloor \frac{\text{del}(v_k)}{T} \right\rfloor \right) \cdot T - \text{del}(v_k)
\]
Clock Slack Minimization

- Let $occ(v_k)$ be the number of operations of type $k$, and $|V_T|$ the number of different operation types. Then, the average clock slack is given by:

$$avgsslack(T) = \frac{\sum_{k=1}^{|V_T|} (occ(v_k) \cdot slack(T, v_k))}{\sum_{k=1}^{|V_T|} occ(v_k)}$$

- The utilization is given by:

$$util(T) = 1 - \frac{avgsslack(T)}{T}$$
FSMD - Model

- Finite state machine + datapath

```
control logic

state register

next state logic
```

```
memory

DR

p1

AR

R1

R2

R3

p2

p3
```
Hardware – Cost Metrics

• Metrics proportional to silicon area
  – mm$^2$, λ$^2$
  – number of transistors, number of gates
  – number of logic blocks (FPGAs)

• Package, number of I/O pins

• FSMD model
  – data path: register, functional units, logic, wiring
  – controller: state register, control logic, next state logic
Hardware - Power / Energy (1)

- CMOS

\[ P = P_{static} + P_{dynamic} \]

\[ P_{short} + P_{load} \]

\[ E_{avg} = \frac{1}{2} \cdot C_{load} \cdot V_{dd}^2 \]

per transition

at clock frequency \( f \)

\[ P_{avg} = \frac{1}{2} \cdot C_{load} \cdot \alpha \cdot f \cdot V_{dd}^2 \]

\( \alpha \ldots \) activity factor
Hardware - Power / Energy (2)

- **Power dissipation**
  - $P \ [W]$
  - important for dimensioning of packaging, power supply, cooling

- **Energy (power-delay product)**
  - $E = P_{\text{avg}} \times T_{\text{exe}} \ [Ws]$
  - important for mobile devices (battery life time)
  - metrics for systems that are operated at a fixed rate
  
  - power normalized to clock period: $[\mu W/MHz]$
  - for processors also:
    - $[\mu W/$MIPS] or [MIPS/$\mu W]$
    - $[\mu W/$SPEC] or [SPEC/$\mu W]$

  Hardware - Power / Energy (2)
• Energy-delay product
  – EDP = E * T_{exe} \ [Ws^2]
  – metrics for systems that are operated at maximum rate

- for processors also:
  - [MIPS^2/\mu W]
  - [SPEC^2/\mu W]
## “Mobile” Processors

<table>
<thead>
<tr>
<th>Processor (Vendor)</th>
<th>Techn. [μ]</th>
<th>V_DD [V]</th>
<th>Clock [MHz]</th>
<th>Power [mW]</th>
<th>MIPS</th>
<th>MIPS/W</th>
<th>MIPS²/mW</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrongARM (Intel)</td>
<td>0.35</td>
<td>2</td>
<td>230</td>
<td>360</td>
<td>268</td>
<td>744</td>
<td>200</td>
</tr>
<tr>
<td>ARM710 (VLSI)</td>
<td>0.8</td>
<td>3.3</td>
<td>25</td>
<td>120</td>
<td>30</td>
<td>250</td>
<td>8</td>
</tr>
<tr>
<td>ARM940T (VLSI)</td>
<td>0.35</td>
<td>3.3</td>
<td>150</td>
<td>675</td>
<td>(e)160</td>
<td>(e)237</td>
<td>(e)38</td>
</tr>
<tr>
<td>MMC2001 (Motorola)</td>
<td>0.35</td>
<td>2</td>
<td>34</td>
<td>80</td>
<td>31</td>
<td>387</td>
<td>12</td>
</tr>
<tr>
<td>TR4102 (LSI)</td>
<td>0.25</td>
<td>1.8</td>
<td>80</td>
<td>40</td>
<td>(e)90</td>
<td>(e)2250</td>
<td>(e)203</td>
</tr>
<tr>
<td>SH7708 (Hitachi)</td>
<td>0.5</td>
<td>3.3</td>
<td>25</td>
<td>95</td>
<td>25</td>
<td>263</td>
<td>7</td>
</tr>
<tr>
<td>SH7750 (Hitachi)</td>
<td>0.25</td>
<td>1.8</td>
<td>200</td>
<td>1600</td>
<td>300</td>
<td>188</td>
<td>56</td>
</tr>
</tbody>
</table>

(e) ... estimated

MIPS ratings for dhrystone benchmark

### VLIW Processors

<table>
<thead>
<tr>
<th>Processor (Vendor)</th>
<th>Techn. [μ]</th>
<th>V\text{DD} [V]</th>
<th>Clock [MHz]</th>
<th>Power [mW]</th>
<th>MIPS</th>
<th>MIPS/W</th>
<th>MIPS^2/mW</th>
</tr>
</thead>
<tbody>
<tr>
<td>'C6201 (Texas Instr.)</td>
<td>0.25</td>
<td>2.5</td>
<td>200</td>
<td>4600</td>
<td>1600</td>
<td>348</td>
<td>557</td>
</tr>
<tr>
<td>SC140 (Motor. /Lucent)</td>
<td>0.13</td>
<td>1.5</td>
<td>300</td>
<td>500</td>
<td>3000</td>
<td>6000</td>
<td>18000</td>
</tr>
<tr>
<td>TM1000 (Philips)</td>
<td>0.35</td>
<td>3.3</td>
<td>100</td>
<td>4000</td>
<td>2500</td>
<td>625</td>
<td>1563</td>
</tr>
<tr>
<td>Merced (HP/Intel)</td>
<td>0.18</td>
<td>?</td>
<td>800</td>
<td>(e)70000</td>
<td>6400</td>
<td>(e)91</td>
<td>(e)585</td>
</tr>
</tbody>
</table>

(e) … estimated  
MIPS ratings are peak numbers  
Overview

• Parameters of estimation methods
• Estimation of hardware metrics
  • Estimation of software metrics
Software - Performance

• Execution time \( T \)

\[
T = I_c \times CPI \times \tau = \frac{(I_c \times CPI)}{f}
\]

- \( I_c \) … instruction count for a given program
- \( CPI \) … cycles per instruction (averaged value)
- \( \tau \) … clock period, \( f \) … clock frequency

• Example

\( I_c = 2000, \ CPI = 0.4, \ f = 400 \text{ MHz} \implies T = 2 \ \mu s \)
Example (1)

- MIPS rate (million instructions per second)

\[
\text{MIPS} = \frac{I_c}{(T \times 10^6)} = \frac{f}{(CPI \times 10^6)}
\]

example:
- processor with clock frequency of 500 MHz
- 3 instruction classes A, B, C with \( \text{CPI}_A = 1 \), \( \text{CPI}_B = 2 \), \( \text{CPI}_C = 3 \)
- 2 different compilers generate (for the same program) following instruction mixes (x \( 10^9 \)):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler 1</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Compiler 2</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example (2)

- clock cycles
  - program1: \((5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9\)
  - program2: \((10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9\)

- execution times
  - program1: \((10 \times 10^9) / (500 \times 10^6) = 20 \text{ sec}\)
  - program2: \((15 \times 10^9) / (500 \times 10^6) = 30 \text{ sec}\)

- MIPS rates
  - program1: \(((5 + 1 + 1) \times 10^9) / (20 \times 10^6) = 350 \text{ MIPS}\)
  - program2: \(((10 + 1 + 1) \times 10^9) / (30 \times 10^6) = 400 \text{ MIPS}\)

program2 has a higher MIPS rate, program1 runs faster!
Software - Performance Metrics (1)

- **MIPS** (million instructions per second)
- **MFLOPS** (million floating-point operations per second)
- **MACS** (million multiply & accumulates per second)
  - important for DSPs
- **MOPS** (million operations per second)
  - counts all possible operations: ALUs, address calculations, DMA, …
  
  **for all**:
  - parallel operations considered
  - sustained (!) peak performance
Software - Performance Metrics (2)

• Execution time
  – profiling: compilation and many test runs ➞ statistical statements
  – estimation on the basis of source- / intermediate- / target code

• Worst-case execution time (WCET)
  – important for real-time systems with hard timing constraints
  – estimation requires
    • program path analysis
      • which sequence of instructions is executed in the worst-case (longest runtime)?
    • microarchitectural modeling
      • taking processor specifics into account: instruction timing, pipelining, caches
• Measurement works only if
  – the worst-case input can be determined, or
  – exhaustive measurement is possible

• The estimated WCET
  – is always higher than the actual WCET; a good estimation method closely approximates the actual WCET
Modern Processor Features

• Modern processors increase performance by using
  – caches
  – pipelining
  – branch prediction
  – ...

• These features make WCET computation difficult because the execution times of single instructions vary widely
  – best case: no cache misses, operands ready, needed resources free, branches correctly predicted
  – worst case: all memory accesses lead to cache misses, resources are blocked, operands not ready, wrong predictions
  – difference between worst and best case can be up to several hundreds of clock cycles
WCET Computation - Approach

Executable program

Control-Flow-Graph

CFG Builder

Loop Unfolding

Static Analyses
- Value Analyzer
- Cache/Pipeline Analyzer

Path Analysis
- ILP-Generator
- LP-Solver
- Evaluation

Timing Information

Micro-Architecture

Loop-Bounds

WCET-Visualization

microarchitectural analysis

worst-case program path analysis

commercialized as: aiT Tool
http://www.absint.com/ait/
Program Path Analysis

• Which sequence of instructions is executed in the worst-case (gives the longest runtime)?

• Problem: the number of possible program paths grows exponentially with the program length

• Model
  – fixed number of cycles for each basic block (derived by static analysis)
  – loops must be bounded

• Approach
  – transform structure of control flow graph (CFG) into an integer linear program (ILP)
  – provide as many additional constraints as possible
  – solution to ILP gives bound on the WCET
/* k >= 0 */
s = k;
WHILE (k < 10) {
    IF (ok)
        j++;
    ELSE {
        j = 0;
        ok = true;
    }
    k ++;
}
r = j;
Calculation of the WCET

- **Definition:** A program consists of \( N \) basic blocks, where each basic block \( B_i \) has a worst-case execution time \( c_i \) and is executed for exactly \( x_i \) times. Then, the WCET is given by:

\[
WCET = \sum_{i=1}^{N} c_i \cdot x_i
\]

- the \( c_i \) values can be determined (by static analysis), because the sequence of executed instructions is known (definition of basic block)

- how to determine \( x_i \)?
  - structural constraints given by the program structure
  - additional constraints provided by the programmer (e.g., bounds for loop counters) based on the knowledge of the program context
Structural Constraints

s = k;

WHILE (k < 10)

if (ok)

j++;

j = 0;

ok = true;

k++;

r = j;

flow equations:

d1 = d2 = x_1

d2 + d8 = d3 + d9 = x_2

d3 = d4 + d5 = x_3

d4 = d6 = x_4

d5 = d7 = x_5

d6 + d7 = d8 = x_6

d9 = d10 = x_7
Additional Constraints

- Loop is executed for at most 10 times:
  \[ x_3 \leq 10 \cdot x_1 \]

- B5 is executed for at most one time:
  \[ x_5 \leq 1 \cdot x_1 \]

```plaintext
s = k;
WHILE (k < 10)
  if (ok)
    j++;
    j = 0;
    ok = true;
  k++;
r = j;
```

- B1
- B2
- B3
- B4
- B5
- B6
- B7
ILP with structural and additional constraints

\[
WCET = \max \left\{ \sum_{i=1}^{N} c_i \cdot x_i \mid d_1 = 1 \land \sum_{j \in \text{in}(B_i)} d_j = \sum_{k \in \text{out}(B_i)} d_k = x_i, \ i = 1 \ldots N \land \text{additional constraints} \right\}
\]
• Processor

• Size of the program memory
  – $\text{instr\_size}(j)$ is the memory requirement for the generic instruction $j$

$$\text{progsize}_{B_i} = \sum_{j \in B_i} \text{instr\_size}(j)$$

• Size of the data memory
  – a program contains a set $D$ of declarations
  – $\text{data\_size}(d)$ is the memory requirement for the declaration $d$

$$\text{data\_size} = \sum_{d \in D} \text{data\_size}(d)$$
Software – Power / Energy (1)

• Simulation on the instruction level
  – assumptions:
    ▪ each instruction needs a certain energy
    ▪ each pair of instructions needs a certain energy
  – simple model, requires only an instruction set simulator
  – for DSPs accuracies > 90% achieved
  – extensions: different energy values for loads/stores
    ▪ depending on the sources/destinations (memory hierarchy)
    ▪ depending on internal / external memory accesses
  – measuring the energy values:

```c
while (1) {
    test_code();
}
```
Software - Power / Energy (2)

- Simulation on the architecture level
  - capacity models for all processor building blocks: ALUs, registers, controllers, cache, …
  - activities of the blocks are simulated
  - complex model, requires cycle-accurate processor simulator
  - higher accuracy than simulation on the instruction level