HW/SW Codesign
HS 2015

WCET Analysis

25. November 2015

Pengcheng Huang
pengcheng.huang@tik.ee.ethz.ch
Know their/your game

- The game here is all about *timing*
- Two main paradigms (confluence starts)

CPS must operate dependably, safely, securely, efficiently and in real-time.\textsuperscript{Raj10}

Cloud...Grids generally support many different kinds of applications, ranging from high performance computing (HPC) to high throughput computing (HTC).\textsuperscript{Foster08}

\textsuperscript{Foster08} I. Foster et al. (2008). Cloud Computing and Grid Computing 360-Degree Compared.
Know their/your game

• The game here is all about *timing*
• Two main paradigms (confluence starts)

**Hard**

CPS must operate dependably, safely, securely, efficiently and in *real-time*. \(^{Raj10}\)

**Soft**

Cloud...Grids generally support many different kinds of applications, ranging from *high performance* computing (HPC) to *high throughput* computing (HTC). \(^{Foster08}\)

---


The “hard” one

• Hard timing guarantees
• Hard analysis (formal analysis, worst-case)

CPS must operate dependably, safely, securely, efficiently and in real-time.\textsuperscript{Raj10}

• Key intuitions / methods may apply to the “soft” problem as well

How? (The big view)

• Abstraction

Timing guarantee

Scheduling (+analysis)

System model

Timing (basic entity)

Program

Architecture

Can be instantiated for different
• Model of computation
• Hardware architecture
• Programming language
• Scheduling technique

Basic timing could include combinations of processor execution, memory operation, communication
How? (The big view)

• Abstraction

Timing guarantee

Scheduling (+analysis)

System model

Timing (basic entity)

Program

Architecture

Can be instantiated for different
• Model of computation
• Hardware architecture
• Programming language
• Scheduling technique

All the above are changing fast in today’s technology landscape, creating many unsolved problems
How? (The big view)

- Abstraction

  - Timing guarantee
  - Scheduling (+analysis)
  - System model
  - Timing (basic entity)

  - Program
  - Architecture

Basic Building block for analysis

Our focus today (von neumann architecture, processing + reading/writing, singlecore)
How? (The big view)

• Abstraction

  Timing guarantee

  Scheduling (+analysis)

  System model

  Timing (basic entity)

  Program

  Architecture

Reason
• Well studied problem
• For shared-memory architecture / programming model, still not well understood

Basic Building block for analysis

Our focus today (von neumann architecture, processing + reading/writing, singlecore)
Outline

• First hour: discuss WCET analysis, revisit slides, ask doubts
  – I will lead you through lecture slides again (not that I am lazy, just not meaningful to recreate the same slides....)
  – I will show some related problems you can work on (possible theses fitting your backgrounds)

• Second hour: solve exercise problem, discuss solution
1st part

Revisit lecture slides

Timing (basic entity) → Basic Building block for analysis

Our focus today (von neumann architecture, processing + reading/writing, singlecore)
What?!

It is all about

Abstraction, separation, formal analysis
Overall Approach: Modularization

- **Micro-architecture Analysis:**
  - Uses Abstract Interpretation
  - Excludes as many Timing Accidents as possible
  - Determines WCET for basic blocks (in contexts)

- **Worst-case Path Determination**
  - Maps control flow graph to an integer linear program
  - Determines upper bound and associated path
Overall Structure

Micro-architecture Analysis

Static Analyses
- Value Analyzer
- Cache/Pipeline Analyzer

Timing Information

Micro-Architecture

Path Analysis
- ILP-Generator
- LP-Solver

Evaluation

WCET-Visualization

Loop Bounds

Worst-case Path Determination

Control-Flow-Graph

to improve WCET bounds for loops
Contents

• *Program Path Analysis*
• Value Analysis
• Caches
  – must, may analysis
• Pipelines
  – Not covered
Control Flow Graph (CFG)

```c
what_is_this {
  read (a, b);
  done = FALSE;
  repeat {
    if (a > b)
      a = a - b;
    elseif (b > a)
      b = b - a;
    else done = TRUE;
  } until done;
  write (a);
}
```
Program Path Analysis

- **Program Path Analysis**
  - which sequence of instructions is executed in the worst-case (longest runtime)?
  - *problem*: the number of possible program paths grows exponentially with the program length

- **Model**
  - we know the upper bounds (number of cycles) for each basic block from static analysis
  - number of loop iterations must be bounded

- **Concept**
  - transform structure of CFG into a set of (integer) linear equations.
  - solution of the Integer Linear Program (ILP) yields bound on the WCET.
Basic Block

• **Definition:** A basic block is a sequence of instructions where the control flow enters at the beginning and exits at the end, without stopping in-between or branching (except at the end).

\[
\begin{align*}
\text{t1} & := c - d \\
\text{t2} & := e \times \text{t1} \\
\text{t3} & := b \times \text{t1} \\
\text{t4} & := \text{t2} + \text{t3} \\
\text{if } \text{t4} & < 10 \text{ goto L}
\end{align*}
\]
Basic Blocks

- *Determine basic blocks of a program:*
  
  1. *Determine the first instructions of blocks:*
     
     the first instruction
     
     targets of un/conditional jumps
     
     instructions that follow un/conditional jumps

  2. *determine the basic blocks:*
     
     there is a basic block for each block beginning
     
     the basic block consists of the block beginning and runs until the
     next block beginning (exclusive) or until the program ends
/ * k >= 0 */

s = k;
WHILE (k < 10) {
    IF (ok)
        j++;
    ELSE {
        j = 0;
        ok = true;
    }
    k ++;
}

r = j;

B1 s = k;

B2 WHILE (k<10)

B3 if (ok)

B4 j++;

B5 j = 0;
ok = true;

B6 k++;

B7 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 are determined using the static analysis.
- how to determine $x_i$?
  - structural constraints given by the program structure
  - additional constraints provided by the programmer (bounds for loop counters, etc.; based on knowledge of the program context)
Structural Constraints

Flow equations:

\[ d_1 = d_2 = x_1 \]
\[ d_2 + d_8 = d_3 + d_9 = x_2 \]
\[ d_3 = d_4 + d_5 = x_3 \]
\[ d_4 = d_6 = x_4 \]
\[ d_5 = d_7 = x_5 \]
\[ d_6 + d_7 = d_8 = x_6 \]
\[ d_9 = d_{10} = x_7 \]
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:

\[ x_5 \leq 1 \cdot x_1 \]
WCET - ILP

- **ILP with structural and additional constraints:**

  \[
  WCET = \max \left\{ \sum_{i=1}^{N} c_i \cdot x_i \mid d_1 = 1 \land \sum d_j = \sum d_k = x_i, \ i = 1 \ldots N \right\}
  \]

  - program is executed once
  - structural constraints
  - additional constraint

  \[
  \sum_{j \in \text{in}(B_i)} d_j = \sum_{k \in \text{out}(B_i)} d_k = x_i, \ i = 1 \ldots N \land
  \]
Contents

• Program Path Analysis
• *Value Analysis*
• Caches
  – must, may analysis
• Pipelines
  – Not covered
Overall Structure

Executable program

Control-Flow-Graph
to improve WCET bounds for loops

Static Analyses
- Value Analyzer
- Cache/Pipeline Analyzer

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

Micro-Architecture
Timing Information

Loop Unfolding

ILP-Generator
LP-Solver
Evaluation
Loop-Bounds
WCET-Visualization
Abstract Interpretation (AI)

- **Semantics-based method** for static program analysis

- **Basic idea of AI**: Perform the program's computations using value descriptions or *abstract values* in place of the concrete values, start with a description of all possible inputs.

- AI supports *correctness* proofs.
Abstract Interpretation – the Ingredients

• **abstract domain** – related to concrete domain by abstraction and concretization functions, e.g. \( L \rightarrow \text{Intervals} \), where \( \text{Intervals} = \text{LB} \times \text{UB} \), \( \text{LB} = \text{UB} = \text{Int} \cup \{-\infty, \infty\} \) instead of \( L \rightarrow \text{Int} \)

• **abstract transfer functions** for each statement type – abstract versions of their semantics e.g. \( + : \text{Intervals} \times \text{Intervals} \rightarrow \text{Intervals} \) where \( [a,b] + [c,d] = [a+c, b+d] \) with \( + \) extended to \(-\infty, \infty\)

• **a join function** combining abstract values from different control-flow paths e.g. \( \cup : \text{Interval} \times \text{Interval} \rightarrow \text{Interval} \) where \( [a,b] \cup [c,d] = [\min(a,c),\max(b,d)] \)
Value Analysis

- **Motivation:**
  - Provide access information to data-cache/pipeline analysis
  - Detect infeasible paths
  - Derive loop bounds

- **Method:** calculate intervals at all program points, i.e. lower and upper bounds for the set of possible values occurring in the machine program (addresses, register contents, local and global variables).
Value Analysis

D1: [-4,4], A0: [0x1000,0x1000]

- Intervals are computed along the CFG edges
- At joins, intervals are "unioned"

Which address is accessed here?

access [0x1000,0x1008]
Contents

- Program Path Analysis
- Value Analysis
- *Caches*
  - must, may analysis
- Pipelines
  - *Not covered*
Caches: Fast Memory on Chip

• **Caches are used**, because
  – Fast main memory is too expensive
  – The speed gap between CPU and memory is too large and increasing

• Caches work well in the *average case*:
  – Programs access data locally (many hits)
  – Programs reuse items (instructions, data)
  – Access patterns are distributed evenly across the cache
Caches

access takes ~ 1 cycle

Processor

access takes ~ 100 cycles

Cache

Bus

Memory

fast, small, expensive

(relatively) slow, large, cheap
Caches: How they work

- CPU wants to *read/write at memory address* \( a \), sends a request for \( a \) to the bus.

- **Cases:**
  - Block \( m \) containing \( a \) is in the cache (hit): request for \( a \) is served in the next cycle.
  - Block \( m \) is not in the cache (miss):
    - \( m \) is transferred from main memory to the cache,
    - \( m \) may replace some block in the cache,
    - request for \( a \) is served asap while transfer still continues.

- Several *replacement strategies*: LRU, PLRU, FIFO,... determine which line to replace.
4-Way Set Associative Cache
LRU Strategy

- Each cache set has its own replacement logic => Cache sets are independent. Everything explained in terms of one set
- **LRU-Replacement Strategy:**
  - Replace the block that has been Least Recently Used
  - Modeled by Ages
- **Example:** 4-way set associative cache

<table>
<thead>
<tr>
<th>access</th>
<th>age 0</th>
<th>age 1</th>
<th>age 2</th>
<th>age 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m_0$</td>
<td>$m_1$</td>
<td>$m_2$</td>
<td>$m_3$</td>
</tr>
<tr>
<td>$m_4$ (miss)</td>
<td>$m_4$</td>
<td>$m_0$</td>
<td>$m_1$</td>
<td>$m_2$</td>
</tr>
<tr>
<td>$m_1$ (hit)</td>
<td>$m_1$</td>
<td>$m_4$</td>
<td>$m_0$</td>
<td>$m_2$</td>
</tr>
<tr>
<td>$m_5$ (miss)</td>
<td>$m_5$</td>
<td>$m_1$</td>
<td>$m_4$</td>
<td>$m_0$</td>
</tr>
</tbody>
</table>
Deriving a Cache Analysis

- **Reducing** the semantics (to what concerns caches)
  - e.g. from values to locations,
  - ignoring arithmetic.
  - obtain “auxiliary/instrumented” semantics

- **Abstraction**
  - Changing the domain: sets of memory blocks in single cache lines
Cache Analysis

- How to statically precompute cache contents:
  - **Must Analysis:**
    For each program point (and calling context), find out which blocks are in the cache. Determines safe information about cache hits. Each predicted cache hit reduces WCET.

  - **May Analysis:**
    For each program point (and calling context), find out which blocks may be in the cache. Complement says what is not in the cache.

    Determines safe information about cache misses. Each predicted cache miss increases BCET.
Abstract Domain: Must Cache
Abstract Domain: Must Cache

Concretization

\[ z, x \in \{ S \in \{ \} \} \]

\[ \gamma \]
Cache with LRU: Transfer for must

**Concrete**

[access s]

\[
\begin{array}{c}
\text{x} \\
\text{y} \\
\text{t} \\
\text{z}
\end{array}
\rightarrow
\begin{array}{c}
\text{s} \\
\text{x} \\
\text{y} \\
\text{t}
\end{array}
\]

"young"

\[
\begin{array}{c}
\text{x} \\
\text{t} \\
\text{y} \\
\text{s}
\end{array}
\rightarrow
\begin{array}{c}
\text{s} \\
\text{x} \\
\text{t} \\
\text{y}
\end{array}
\]

"old"

**Abstract**

[access s]

\[
\begin{array}{c}
\{ \text{x} \} \\
\{ \} \\
\{ \text{y, t} \} \\
\{ \}
\end{array}
\rightarrow
\begin{array}{c}
\{ \text{s} \} \\
\{ \text{x} \} \\
\{ \} \\
\{ \text{y, t} \}
\end{array}
\]

Age

"young"

"old"
Cache Analysis: Join (must)

Join (must)

```
{ a }
{ }  { c }
{ c, f }  { e }
{ d }  { a }
{ d }  { d }
```

“intersection + maximal age”

**Interpretation:**
memory block a is definitely in the (concrete) cache ⇒ always hit
Abstract Domain: May Cache

Abstraction
Abstract Domain: May Cache

Concretization

\[ m, n, o \in \{z, s, x, t\}, \quad p \in \{z, s, x, t, a\} \]

\[ \gamma \]

\[
\begin{array}{c}
\{z, s, x\} \\
\{t\} \\
\{\} \\
\{a\}
\end{array}
\]
Cache with LRU: Transfer for may

*concrete*

[ access s ]

x

y

t

z

**concrete**

“young”

Age

“old”

*abstract*

[ access s ]

\{ x, t \}

\{ y, s \}

\{ z \}

\{ \}

\{ s \}

\{ x, t \}

\{ y, z \}

\{ \}
Cache Analysis: Join (may)

Join (may)

```
{ a }
{ } 
{ c, f } 
{ d } 
```

```
{ c }
{ e }
{ a }
{ d } 
```

“union + minimal age”

Interpretation:
all blocks may be in the cache; none is definitely not in the cache.
Contribution to WCET

• Information about cache contents sharpens timings.

ref to s

\[ t_{\text{hit}} \rightarrow t_{\text{miss}} \]

\[
\begin{align*}
\text{if } s \text{ is in must-cache:} \\
& t_{\text{WCET}} = t_{\text{hit}} \\
\text{otherwise} \\
& t_{\text{WCET}} = t_{\text{miss}} \\
\text{if } s \text{ is in may-cache:} \\
& t_{\text{BCET}} = t_{\text{hit}} \\
\text{otherwise} \\
& t_{\text{BCET}} = t_{\text{miss}}
\end{align*}
\]
Contribution to WCET

- Information about cache contents sharpens timings.

```plaintext
while ... do [max n]
  ...
  ref to s
  ...
  od

within loop
  n * t_{miss}
  n * t_{hit}
  t_{miss} + (n - 1) * t_{hit}
  t_{hit} + (n - 1) * t_{miss}
  ...
```
Contexts

- Cache contents depends on the context, i.e. calls and loops

- First Iteration loads the cache:
  - Intersection loses most of the information.

- Distinguish as many contexts as useful:
2nd part

Relevant problems?
(both to you and to the topic)

We are calling for student projects!
Massively Parallel Architectures

- Up to 1024 processing cores!
- Amazing performance/power ratio
- VLIW core
- C/C++/OpenCL
- High speed (predictable) network on chip
- Low latency processing
Massively Parallel Architectures

Exciting applications!

Single cloud on a chip
- data analytics
- Image processing
- High performance computing

Industrial applications
- Flight control / management systems
- Smart car systems
Massively Parallel Architectures

Challenges!

Single cloud on a chip
- Performance optimization
- Programmability
- Energy / reliability / thermal issues
- ...

Industrial applications
- Strict predictability
- Extreme dependability
- High efficiency
- ...

Swiss Federal Institute of Technology
Massively Parallel Architectures

We are calling!

• Benchmarking / modeling / understanding
  – Safety-critical applications
  – Is the behavior of all resources predictable?
  – Can machine learning help us reverse engineer h/w resources?

• Programming / Optimization
  – Model driven design
  – Scheduling
  – Energy efficiency...
Massively Parallel Architectures

We are calling!

• Benchmarking / modeling / understanding
  – Safety-critical applications
  – Is the behavior of all resources predictable?
  – Can machine learning help us reverse engineer h/w resources?

• Programming / Optimization
  – Model driven design
  – Scheduling
  – Energy efficiency...
Massively Parallel Architectures

We are calling!

Visit our webpage!
http://www.tec.ethz.ch/theses.php
3rd part

Problem today
3rd part

```c
a = 1;
if (b >= 0) then {
  i = a + 2;
  n = (a + 3)^2 + 2;
  m = a + 3;
  b = 6;
} else {
  l = a + 6;
  n = l + 8;
  i = a + 4;
}
while (n >= i) do {
  if (b > 5) {
    a = 3;
    l = l + 1;
  } else {
    m = m + 1;
    a = 4;
  }
  i = i + 1;
}
```

- the processor has no pipeline
- the processor has no registers, i.e. all variables are stored in memory
- in an assignment, predicate evaluation, or arithmetic operation ... right to left e.g.
  ```c
  a = b + c
  ```
  variables are accessed in the order c, b, a
- consider only the data cache
...
3rd part

a = 1;

if (b >= 0) then {
    i = a + 2;
    n = (a + 3)^2 + 2;
    m = a + 3;
    b = 6;
} else {
    l = a + 6;
    n = l + 8;
    i = a + 4;
}

while (n >= i) do {

    if (b > 5) {
        a = 3;
        l = l + 1;
    } else {
        m = m + 1;
        a = 4;
    }

    i = i + 1;
}
Solution

• 30 min for solving questions
3\textsuperscript{rd} part (a)

\begin{verbatim}
a = 1;
if (b >= 0) then {
i = a + 2;
n = (a + 3)^2 + 2;
m = a + 3;
b = 6;
} else {
l = a + 6;
n = l + 8;
i = a + 4;
}
while (n >= i) do {
if (b > 5) {
a = 3;
l = l + 1;
} else {
m = m + 1;
a = 4;
}
i = i + 1;
}
\end{verbatim}
3rd part (b)

```plaintext
a = 1;

if (b >= 0) then {
    i = a + 2;
    n = (a + 3)^2 + 2;
    m = a + 3;
    b = 6;
} else {
    l = a + 6;
    n = l + 8;
    i = a + 4;
}
```

```
B1:
    a = 1
    a ∈ [1,1]
    b, i, l, m, n ∈ (−∞, ∞)

b >= 0
    a ∈ [1,1]
    b, i, l, m, n ∈ (−∞, ∞)
```
3rd part (b)

\[ a = 1; \]

if \((b >= 0)\) then {
    \[ i = a + 2; \]
    \[ n = (a + 3)^2 + 2; \]
    \[ m = a + 3; \]
    \[ b = 6; \]
} else {
    \[ l = a + 6; \]
    \[ n = l + 8; \]
    \[ i = a + 4; \]
}
3rd part (b)

while (n >= i) do {
    if (b > 5) {
        a = 3;
        i = i + 1;
    } else {
        m = m + 1;
        a = 4;
    }
}

i = i + 1;

First pass through the loop:

B4:
    while (n >= i)
        i ∈ [3, 5]
        n ∈ [15, 18]
        a ∈ [1, 1]
        b ∈ (−∞, 6]
        l, m ∈ (−∞, ∞)

B5:
    if (b > 5)
        i ∈ [3, 5]
        n ∈ [15, 18]
        a ∈ [1, 1]
        b ∈ (−∞, 6]
        l, m ∈ (−∞, ∞)

B7:
    m = m + 1
    i ∈ [3, 5]
    n ∈ [15, 18]
    a ∈ [1, 1]
    b ∈ (−∞, 5]
    l, m ∈ (−∞, ∞)
    a = 4

Join of B6 and B7:
    i ∈ [3, 5]
    n ∈ [15, 18]
    a ∈ [3, 4]
    b ∈ (−∞, 6]
    l, m ∈ (−∞, ∞)

B6:
    a = 3
    i ∈ [3, 5]
    n ∈ [15, 18]
    a ∈ [3, 3]
    b ∈ (−∞, 6]
    l, m ∈ (−∞, ∞)
    l = l + 1
    i ∈ [3, 5]
3\textsuperscript{rd} part (b)

while \((n >= i)\) do {
    if \((b > 5)\) {
        \(a = 3;\)
        \(l = l + 1;\)
    } else {
        \(m = m + 1;\)
        \(a = 4;\)
    }
    \(i = i + 1;\)
}
3\textsuperscript{rd} part (b)

while (n >= i) do {
  if (b > 5) {
    a = 3;
    l = l + 1;
  } else {
    m = m + 1;
    a = 4;
  }
  i = i + 1;
}

First pass through the loop:

B4:
  while (n >= i)
    i \in [3, 5]
    n \in [15, 18]
    a \in [1, 1]
    b \in (-\infty, 6]
    l, m \in (-\infty, \infty)

B5:
  if (b > 5)
    i \in [3, 5]
    n \in [15, 18]
    a \in [1, 1]
    b \in (-\infty, 6]
    l, m \in (-\infty, \infty)
  else {
    m = m + 1
    a = 4
  }

B6:
  a = 3
  i \in [3, 5]
  n \in [15, 18]
  a \in [3, 3]
  b \in (5, 6]
  l, m \in (-\infty, \infty)

B7:
  m = m + 1
  i \in [3, 5]
  n \in [15, 18]
  a \in [1, 1]
  b \in (-\infty, 5]
  l, m \in (-\infty, \infty)

Join of B8, B2, and B3:
  i \in [3, 6]
  n \in [15, 18]
  a \in [3, 4]
  b \in (-\infty, 6]
  l, m \in (-\infty, \infty)
3rd part (b)

while (n >= i) do {
    if (b > 5) {
        a = 3;
        l = l + 1;
    } else {
        m = m + 1;
        a = 4;
    }
    i = i + 1;
}

First pass through the loop:

B4:
    while (n >= i)
        i ∈ [3, 5]
        n ∈ [15, 18]
        a ∈ [1, 1]
        b ∈ (−∞, 6]
        l,m ∈ (−∞, ∞)
B7:
    m = m + 1
        i ∈ [3, 5]
        n ∈ [15, 18]
        a ∈ [1, 1]
        b ∈ (−∞, 5]
        l,m ∈ (−∞, ∞)
B6:
    a = 3
        i ∈ [3, 5]
        n ∈ [15, 18]
        a ∈ [3, 3]
        b ∈ (−∞, 6]
        l,m ∈ (−∞, ∞)
B8:
    i = i + 1
        i ∈ [4, 6]
        n ∈ [15, 18]
        a ∈ [3, 4]
        b ∈ (−∞, 6]
        l,m ∈ (−∞, ∞)
Join of B8, B2, and B3:
    i ∈ [3, 6]
    n ∈ [15, 18]
    a ∈ [1, 4]
    b ∈ (−∞, 6]
    l,m ∈ (−∞, ∞)
3rd part (c)

\[
\begin{align*}
a &= 1; \\
\text{if } (b \geq 0) \text{ then } &\{ \\
& \quad i = a + 2; \\
& \quad n = (a + 3)^2 + 2; \\
& \quad m = a + 3; \\
& \quad b = 6; \\
\} \text{ else } &\{ \\
& \quad l = a + 6; \\
& \quad n = l + 8; \\
& \quad i = a + 4; \\
\end{align*}
\]

H for hit, M for miss,
\(ci\) is WCET in cycles for the \(i\)-th basic block
Initial cache state: \((b,-,-,-)\)

**B1:**
\[
\begin{align*}
a &= 1 \\
& \quad M, (a,b,-,-) \\
& \quad \text{if } (b \geq 0) \\
& \quad H, (b,a,-,-) \\
& \quad c1: 102
\end{align*}
\]

**B2:**
\[
\begin{align*}
i &= a + 2 \\
& \quad H, (a,b,-,-) \\
& \quad M, (i,a,b,-) \\
& \quad n = (a + 3)^2 + 2 \\
& \quad H, (a,i,b,-) \\
& \quad M, (n,a,i,b) \\
& \quad m = a + 3 \\
& \quad H, (a,n,i,b) \\
& \quad M, (m,a,n,i) \\
& \quad b = 6 \\
& \quad M, (b,m,a,n) \\
& \quad c2: 406
\end{align*}
\]

**B3:**
\[
\begin{align*}
l &= a + 6 \\
& \quad H, (a,b,-,-) \\
& \quad M, (l,a,b,-) \\
& \quad n = l + 8 \\
& \quad H, (l,a,b,-) \\
& \quad M, (n,l,a,b) \\
& \quad i = a + 4 \\
& \quad H, (a,n,l,b) \\
& \quad M, (i,a,n,l) \\
& \quad c3: 306
\end{align*}
\]

Join of B2 and B3: \([{},{}],[a],[n]\)
3rd part (d)

Structural constraints:

\[ d_1 = d_2 + d_3 = x_1 \]
\[ d_2 = d_4 = x_2 \]
\[ d_3 = d_5 = x_3 \]
\[ d_4 + d_5 + d_{11} = d_6 + d_{12} = x_4 \]
\[ d_6 = d_7 + d_8 = x_5 \]
\[ d_7 = d_9 = x_6 \]
\[ d_8 = d_{10} = x_7 \]
\[ d_9 + d_{10} = d_{11} = x_8 \]

Assume program is executed once: \( d_1 = 1 \).

Loop bounds →

\[ x_5 \leq 16 \times (x_2 + x_3) \]

Objective function:

\[ WCET = \max \left\{ \sum_{i=1}^{8} c_i \times x_i \right\} \]
That is all, folks!

• See you in 2 weeks