# Timing Analysis of Parallel Software Using Abstract Execution

Björn Lisper
School of Innovation, Design, and Engineering
Mälardalen University

bjorn.lisper@mdh.se

2014-09-10

EACO Workshop 2014

# Motivation

Increasing need for high-performance computing in time-critical embedded systems

(E.g., “smart car” collision avoidance systems)

Parallel systems the only way to get enough performance at reasonable cost

Timing analysis of parallel SW/HW becomes important

Difficult area, few results, lack of theoretical underpinnings

# What We Have Done

A Worst-Case Execution Time (WCET) analysis algorithm for thread-parallel programs with shared memory

Theoretical work – we treat a small model language with threads. HW timing model is assumed to be given

The algorithm is called *Abstract Execution*, extends a previous algorithm for analysis of sequential programs. Based on abstract interpretation

Proof of *soundness* (execution times are never underestimated)

Joint work with Andreas Gustavsson

Papers so far: WCET'12, VMCAI'14

# WCET Analysis for Sequential Programs

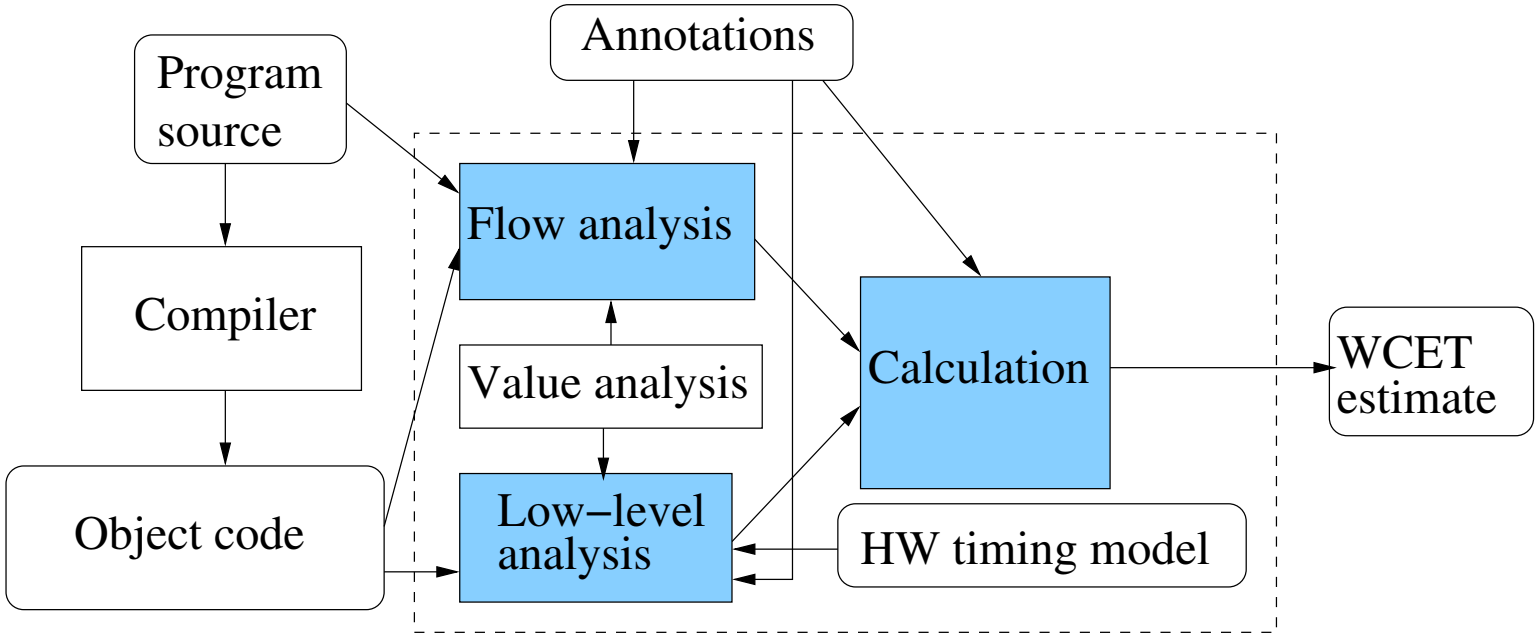
WCET for sequential program $P$: longest possible time to execute $P$ uninterrupted (on some given HW)

WCET *analysis* aims to find safe upper bounds to WCET – used for verification of hard RT systems

Typically broken down into the following steps:

- Constrain possible program flows (“high-level”, or “flow” analysis)
- Estimate hardware impact to bound WCET for program fragments (“low-level” analysis)
- Combine this information to produce a safe WCET estimate

# Structure of WCET Analysis



# Example

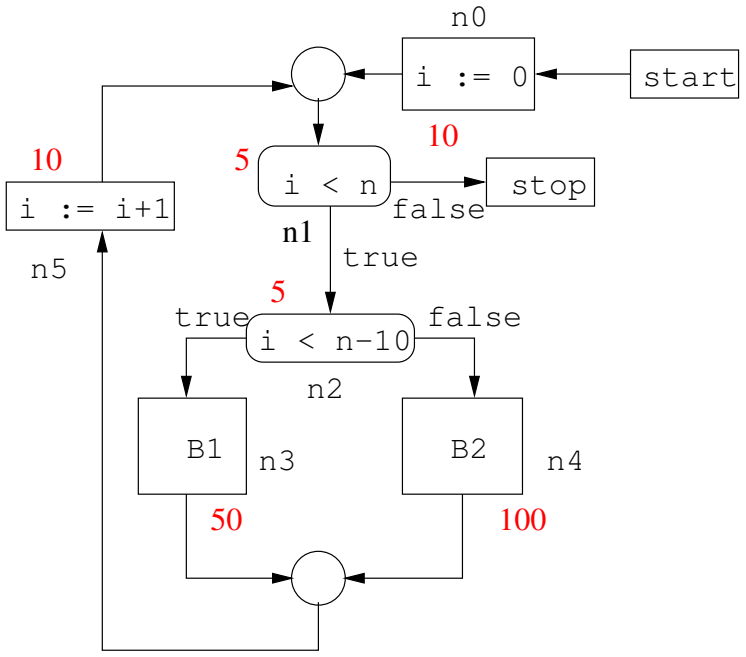
Red numbers $=$ local WCETs for basic blocks

Assume $\mathtt{n} \in [0, 20]$

Then # of loop iterations $\leq 20$

WCET bound: 2415 cycles

(Tighter bound can be found by a closer analysis of the condition)



# WCET Analysis of Sequential Programs, Status

The problem is well understood

Quite a few tools exist, also commercial (aiT, RapiTime, Bound-T)

Still issues with:

- complex sequential processor architectures (low-level analysis)
- level of automation (flow analysis)

# WCET Analysis for Parallel Systems

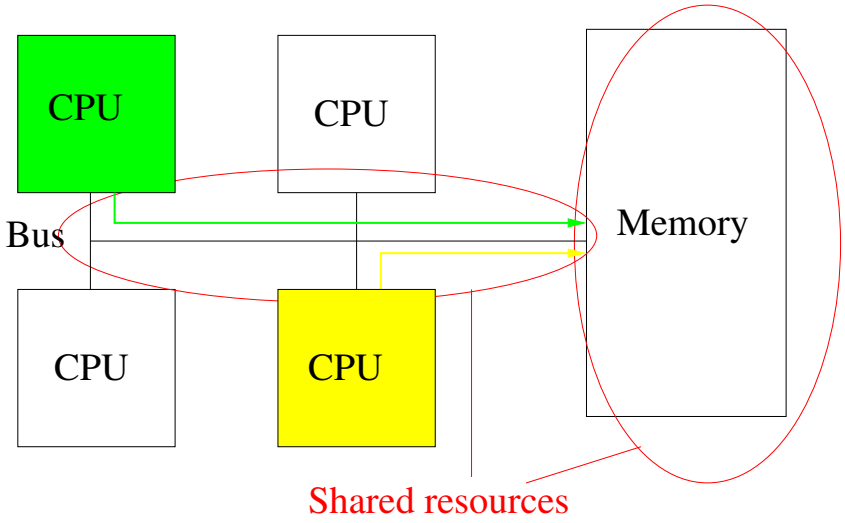
Much less understood than WCET analysis for sequential systems (single-core). Two cases:

- Sequential programs (tasks) running on single cores in a multi-core processor
- Parallel programs running on parallel hardware

The first case has been studied some

Major problem: competition for shared resources makes instruction execution times very unpredictable

# Competition for Shared Resources




Basically a hardware problem: current multi-core architectures are not designed for timing predictability

# WCET Analysis for Parallel Programs

Sequential programs allow the nice division into different analysis stages. In particular, the flow analysis becomes independent of timing

Analysis of **parallel programs** opens a can of worms:

- Race conditions
- Waiting times forced by synchronisation
- Deadlocks

Analysis has to be integrated. Timing (race conditions) can affect program flow, and vice versa

Much less studied. Research results are scarce, and incomplete. This is the problem that we have attacked

# Abstract Execution

Originally a program flow analysis for sequential programs [Gustafsson et al, 2006]

Finds constraints on program flow through a kind of symbolic execution – “abstract execution” (AE)

AE executes the program with *abstract states*, representing sets of real (“concrete”) states

Can be seen as a a kind of value analysis (abstract interpretation)

We have extended it with time [Gustafsson et al, 2011], and to handle thread-parallel programs

# Example

```
i = INPUT; // i = [1..4]
#p = 0;
while (i < 10) {
    // point p
    #p = #p + 1;
    ...
    i=i+2;
}
// point q
```

**(a) Code example**

| p | i at p | i at q |
|---|---|---|
| 1 | [1..4] | impossible |
| 2 | [3..6] | impossible |
| 3 | [5..8] | impossible |
| 4 | [7..9] | [10..10] |
| 5 | [9..9] | [10..11] |
| 6 | impossible | [10..11] |

**(b) Analysis**

**min.**
**#p:** 3

**max.**
**#p:** 5

**(c) Result**

(`#p` is the *execution counter* for program point `p`)

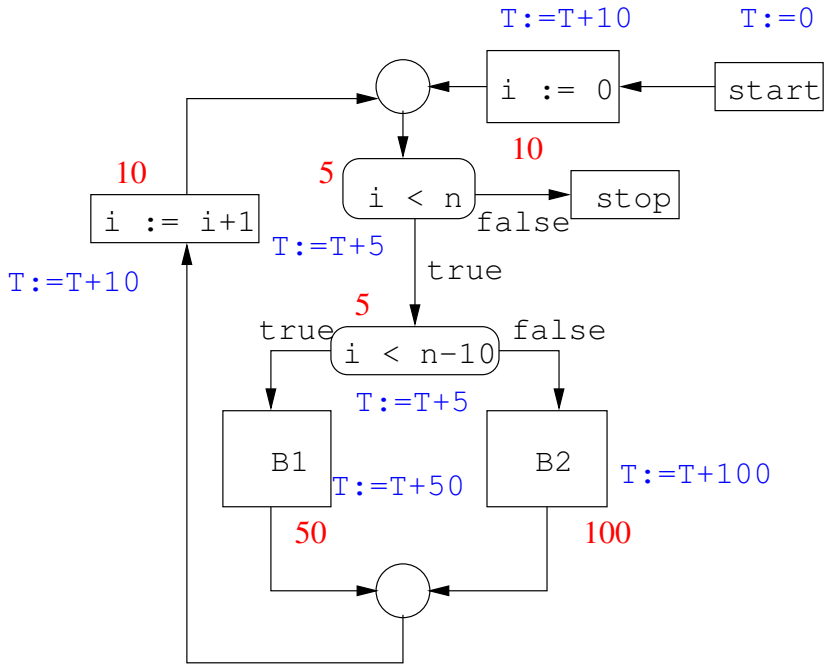
# Abstract Execution with Time

An integrated approach

Include time in the abstract state

Compute an interval containing the possible execution times

Yields bound for WCET (and BCET)



# Abstract Execution for Thread-Parallel Programs

Abstract Execution (with time) is an integrated approach

We have extended it to thread-parallel programs with shared memory

A small language “PPL” with a well-defined semantics including time (transitions between “concrete configurations”)

Abstract states representing sets of concrete configurations (“abstract configurations”), with transition rules (“abstract transitions”)

Abstract configurations have a local state for each thread, including a time interval

Abstract Execution searches the space of abstract configurations for terminated configurations

WCET/BCET bounds can be calculated from the time intervals of these

# PPL

A simple model language. Features:

- Fixed set of threads
- Shared memory (accessible by all threads), thread-private memory (“registers”)
- Locks (accessible by all threads)
- Quite low-level (“abstract instruction set”)

# PPL, Formal Syntax

$$\Pi ::= \{\mathrm{T}_1, \ldots, \mathrm{T}_m\}$$

$$\mathrm{T} ::= (d, s)$$

$$\begin{aligned} s ::= {} & [\texttt{halt}]^l \mid [\texttt{skip}]^l \mid [r := a]^l \mid [\texttt{if } b \texttt{ goto } l']^l \mid [\texttt{load } r \texttt{ from } x]^l \mid \\ & [\texttt{store } r \texttt{ to } x]^l \mid [\texttt{lock } lck]^l \mid [\texttt{unlock } lck]^l \mid s_1; s_2 \end{aligned}$$

$$a ::= n \mid r \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 \,/\, a_2$$

$$b ::= \texttt{true} \mid \texttt{false} \mid \texttt{!}\, b \mid b_1 \,\&\&\, b_2 \mid a_1 == a_2 \mid a_1 <= a_2$$

# Concrete Semantics of PPL

Operational semantics (transitions between configurations)

Includes time

Configurations (system states) consist of:

- Local state for each thread (program counter, register contents, local accumulated time)
- Contents of shared memory
- State of locks

The shared memory state is more complex than usual. For each variable a *history of writes* (time, issuing thread, value). Also the lock state is somewhat non-standard

# Abstract Semantics of PPL

Abstract configurations represent sets of concrete configurations

A transition relation for abstract configurations that safely approximates the possible concrete transitions

Numerical entities (times, values) are abstracted to intervals

The abstract configurations form a complete lattice

A Galois connection between sets of concrete configurations (collecting semantics) and abstract configurations

All set in the framework of Abstract Interpretation

# Abstract Execution of PPL

A worklist algorithm to explore the space of abstract configurations

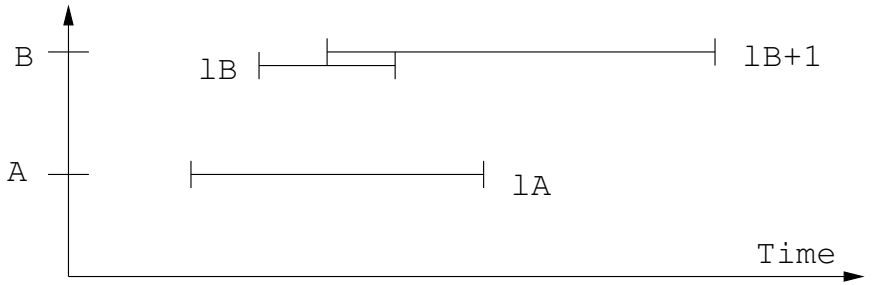
Terminated abstract configurations (all threads have executed a halt instruction) are collected, and abstract execution time intervals are read off

Some potential difficulties:

- Potential state space explosion due to factors like race conditions
- Can be battled by merging abstract configurations, but then risk of low precision
- Deadlocks must be handled
- Inherent risk of non-termination of analysis

# Race Conditions

| thread A : | thread B : |
|---|---|
| ⋮ | ⋮ |
| $[\texttt{load r}_\texttt{A}\ \texttt{from x}]^{l_A}$ | $[\texttt{skip}]^{l_B}$ |
| | $[\texttt{store r}_\texttt{B}\ \texttt{to x}]^{l_B+1}$ |




Different abstract configurations spawned for the different possible race outcomes

Issues:

- May cause a state explosion
- Overapproximated time intervals may lead to false race conditions

# Deadlocks and Non-Termination

The algorithm can detect many cases of deadlocks

Some may however go unnoticed

This can affect the termination properties of the algorithm

Abstract Execution in itself is also potentially non-terminating (even in the sequential case)

Non-termination can be handled by timeout mechanisms

# Soundness

**Theorem**: if Abstract Execution terminates, then the calculated BCET and WCET extimates are safe lower and upper bounds to the possible execution times

# Conclusions

A WCET analysis algorithm for thread-parallel programs with shared memory and locks

Simple but quite general model language

The algorithm can detect deadlocks, and can also be turned into a value analysis

Main result: soundness of algorithm

However many issues: complexity, precision, potential non-termination. Quite some way to go still to a practically useful analysis

These issues really seem inherent in the shared memory paradigm. More structured parallel programming models seem needed to produce timing-analysable parallel software!