Automatic WCET Analysis of Real-Time Parallel Applications

Haluk Ozaktas Christine Rochange Pascal Sainrat

IRIT – Université de Toulouse

July 9, 2013

parMERASA

13th International Workshop on Worst-Case Eexecution Time

1 Introduction
- Motivation
- Challenges and scope

2 WCET analysis of parallel applications
- Overview
- Timing analysis of synchronisations
- Annotation of parallel programs
- Computation of the global WCET

3 Experimental results
- Methology
- Results

4 Future work

Current Section

1. Introduction
   - Motivation
   - Challenges and scope
2. WCET analysis of parallel applications
   - Overview
   - Timing analysis of synchronisations
   - Annotation of parallel programs
   - Computation of the global WCET
3. Experimental results
   - Methology
   - Results
4. Future work

Motivation

- Global trends: increasing production/usage multicores
  - near future: *many*cores
- Need for more processing power → need for more processing cores

⇒ No way of escaping multicores/*many*cores even for real-time embedded systems

Motivation

- Global trends: increasing production/usage multicores
  - near future: *many*cores
- Need for more processing power → need for more processing cores

⇒ No way of escaping multicores/*many*cores even for real-time embedded systems

Motivation

- Single critical task running on a single core
  - Other cores run non-critical tasks to avoid waste of processing power
- Many independent critical tasks each running on a different core
- What if wee need more processing power for the critical task?

⇒ We want to run real-time **parallel** applications on *many*core architectures

Motivation

- Single critical task running on a single core
  - Other cores run non-critical tasks to avoid waste of processing power
- Many independent critical tasks each running on a different core
- What if wee need more processing power for the critical task?

$\Rightarrow$ We want to run real-time **parallel** applications on *many*core architectures

Challenges and scope

1. Multicore → trouble for timing analysis
   - Conflicts to shared resources
   - A solution: time-predictable multicore/*many*core architectures
     - Related projects: T-CREST, parMERASA
   - We assume these architectures are available!

2. Parallel programming itself – we focus on POSIX-like thread synchronisation on a shared memory model
   - Inter-thread synchronisations generate *stalls*
     - barriers, critical sections, condition variables...
   - **We need to bound these stall times!**
   - We assume a time-predictable system software!
     - FIFO queues, bounded thread creation time...

Challenges and scope

1. Multicore → trouble for timing analysis
   - Conflicts to shared resources
   - A solution: time-predictable multicore/*many*core architectures
     - Related projects: T-CREST, parMERASA
   - We assume these architectures are available!

2. Parallel programming itself – we focus on POSIX-like thread synchronisation on a shared memory model
   - Inter-thread synchronisations generate *stalls*
     - barriers, critical sections, condition variables...
   - **We need to bound these stall times!**
   - We assume a time-predictable system software!
     - FIFO queues, bounded thread creation time...

Current Section

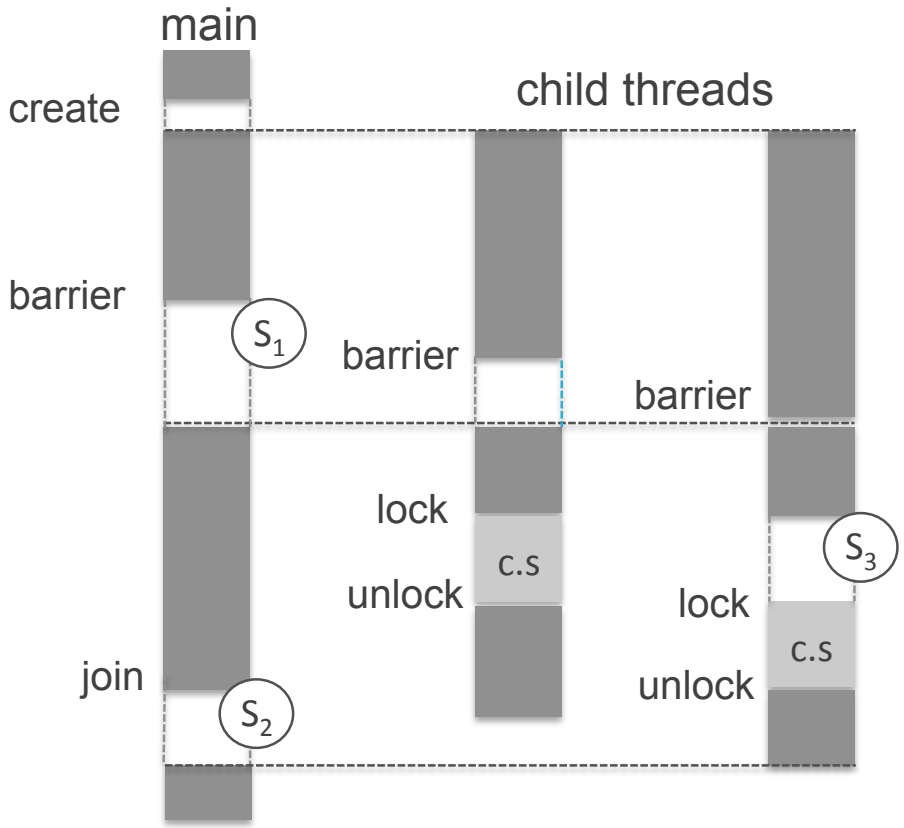
1 Introduction
- Motivation
- Challenges and scope

2 WCET analysis of parallel applications
- Overview
- Timing analysis of synchronisations
- Annotation of parallel programs
- Computation of the global WCET

3 Experimental results
- Methology
- Results

4 Future work

WCET analysis of parallel applications

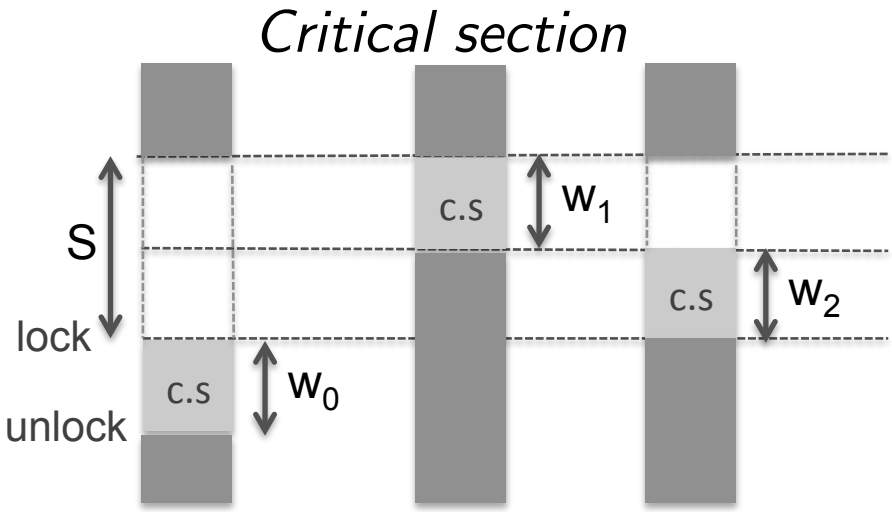


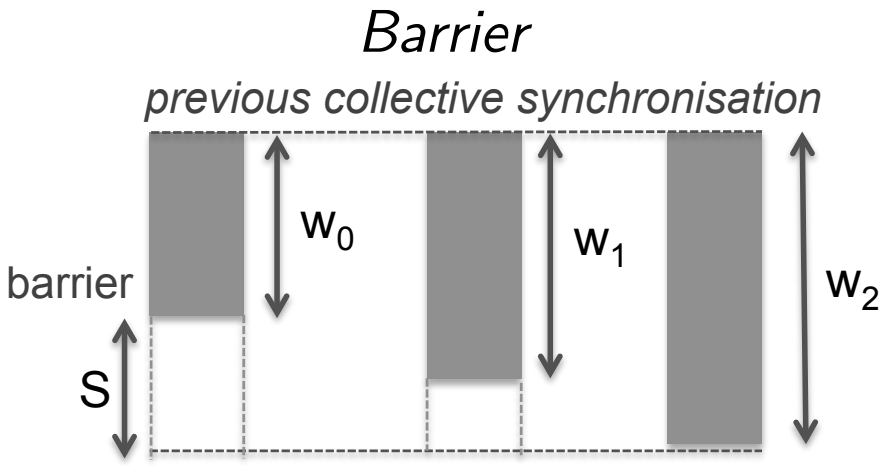
*S1, S2, S3: Stalls due to synchronisations

- Global WCET is the WCET of the main thread
- *Worst-case stall times* (WCSTs) need to be computed

Timing analysis of synchronisations

- Critical sections (guarded by locks)
- Progress synchronisations (barriers, conditions)





$$S = w_1 + w_2$$

$$S = \max(0, (w_1 - w_0), (w_2 - w_0))$$

Annotation of parallel programs

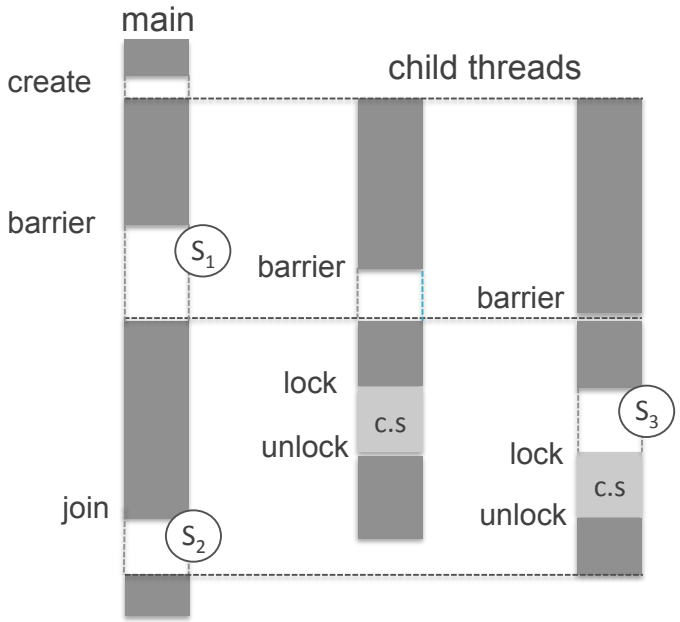
- Information to compute WCSTs are passed through code annotations
  - Calls to synchronisation primitives are given an ID in the source code
  - An xml-based file describes further these synchronisations

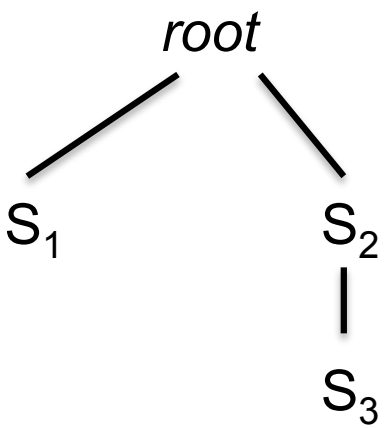
```
int main() {
  for (int i=0; i<2; i++)
    CREATE_THREAD(&work);
  ...
  BARRIER(&bar,3);  // ID=bar
  ...
  for (int i=0; i<2; i++)
    JOIN(i+1);  // ID=join
}

void work() {
  ...
  BARRIER(&bar,3);  // ID=bar
  ...
  MUTEX_LOCK(&lock);  // ID=cs
  ... // critical section
  MUTEX_UNLOCK(&lock);  // ID=cs
  ...
}
```

```
<barrier id="bar">
  <thread id="0-2">
    <last_sync ref="BEGIN"/>
  </thread>
</barrier>
<csection id="cs">
  <thread id="1-2"/>
</csection>
<sync id="join">
  <thread id="0">
    <wait id="1-2">
      <sync ref="END"/>
      <last_sync ref="bar"/>
    </wait>
  </thread>
</sync>
```

Computation of the global WCET





Identification of synchronisations

↓

Building of WCSTs' tree

↓

Computation of WCSTs

↓

Integration of WCSTs to the CFG of main thread
Computation of global WCET

Current Section

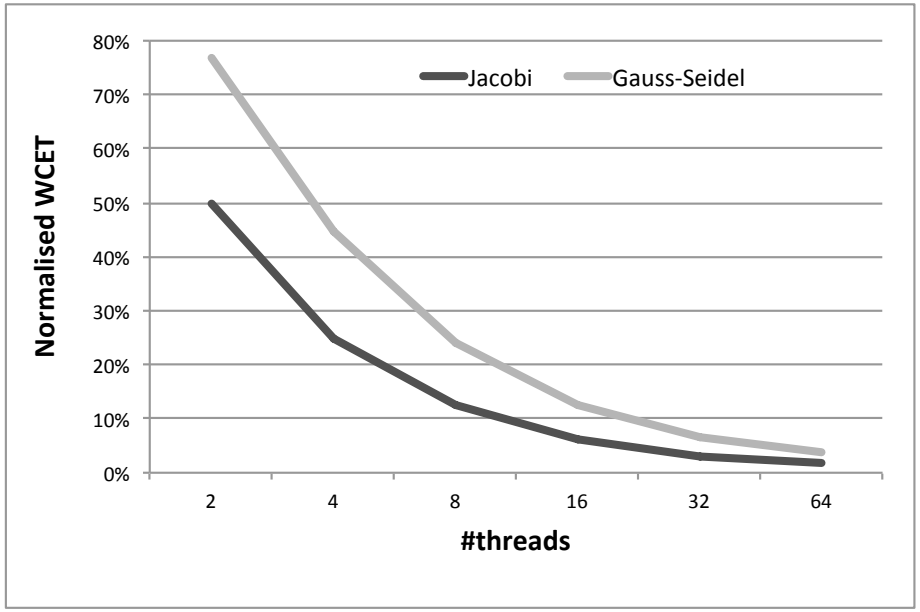
1. Introduction
   - Motivation
   - Challenges and scope
2. WCET analysis of parallel applications
   - Overview
   - Timing analysis of synchronisations
   - Annotation of parallel programs
   - Computation of the global WCET
3. Experimental results
   - Methology
   - Results
4. Future work

Methodology

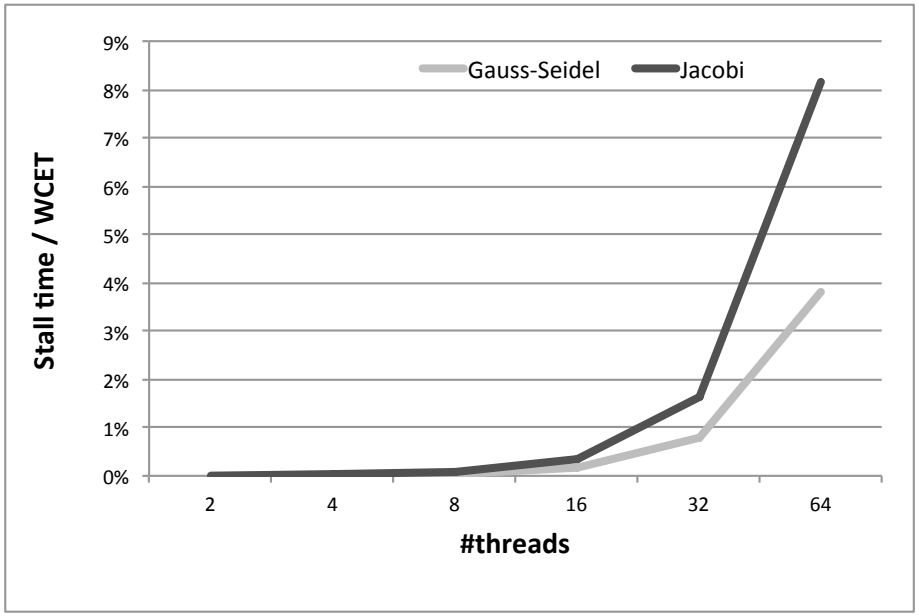
- Partial differential equation solver
  - Gauss-Seidel method
  - Jacobi method
- Simple architecture (1 inst/cycle) with configurable memory latency
- All implementation done in `OTAWA` toolset (www.otawa.fr)

Results

- Normalised WCET



- Impact of stall times

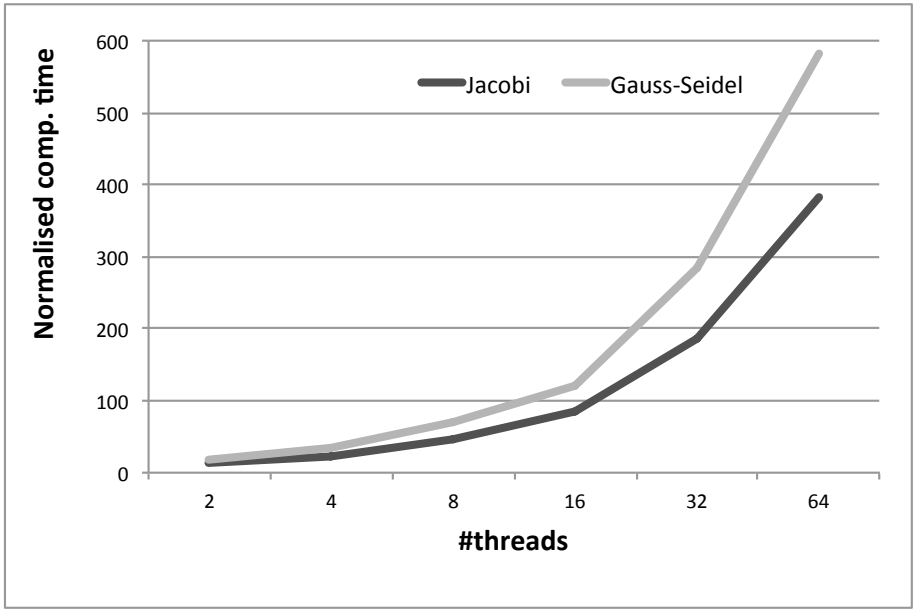


Results – computation times

- Computation times in seconds

	Gauss-Seidel	Jacobi
2	0.559	0.379
4	1.046	0.705
8	2.177	1.446
16	3.718	2.679
32	8.796	5.782
64	17.999	11.855

- Normalised computation times



Current Section

1. Introduction
   - Motivation
   - Challenges and scope
2. WCET analysis of parallel applications
   - Overview
   - Timing analysis of synchronisations
   - Annotation of parallel programs
   - Computation of the global WCET
3. Experimental results
   - Methology
   - Results
4. Future work

Future work

- Ongoing
  - Analysis of industrial applications
  - Visualisation of synchronisation points
  - Eclipse plugin to ease writing annotations
- Planned
  - Automatic extraction of information on synchronisations
    - Contending threads, last synchronisation point...
  - A way to define precise stall point

THANK YOU!