Automatic WCET Analysis of Real-Time Parallel Applications

Haluk Ozaktas       Christine Rochange       Pascal Sainrat

IRIT – Université de Toulouse

July 9, 2013

13th International Workshop on Worst-Case Execution 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
   - Methodology
   - Results

4 Future work
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 $\rightarrow$ need for more processing cores

$\Rightarrow$ No way of escaping multicores/*many*cores even for real-time embedded systems
Motivation

- Global trends: increasing production/usage multicore
  - near future: *many*cores
- Need for more processing power $\rightarrow$ need for more processing cores
  $\Rightarrow$ No way of escaping multicore/*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 we need more processing power for the critical task?

⇒ We want to run real-time parallel applications on manycore 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 we need more processing power for the critical task?

⇒ We want to run real-time parallel applications on manycore architectures
Challenges and scope

1. Multicore → trouble for timing analysis
   - Conflicts to shared resources
   - A solution: time-predictable multicore/manycore 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/manycore 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
   - Methodology
   - Results

4. Future work
**WCET analysis of parallel applications**

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

*S1, S2, S3*: Stalls due to synchronisations.
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

```c
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
...
}
```

```xml
<barrier id="bar">
    <thread id="0-2">
        <last_sync ref="BEGIN"/>
    </thread>
</barrier>
<cs>
    <thread id="1-2"/>
</cs>
<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
   - Methodology
   - 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**
Experimental results

Results – computation times

- Computation times in seconds

<table>
<thead>
<tr>
<th>#threads</th>
<th>Gauss-Seidel</th>
<th>Jacobi</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.559</td>
<td>0.379</td>
</tr>
<tr>
<td>4</td>
<td>1.046</td>
<td>0.705</td>
</tr>
<tr>
<td>8</td>
<td>2.177</td>
<td>1.446</td>
</tr>
<tr>
<td>16</td>
<td>3.718</td>
<td>2.679</td>
</tr>
<tr>
<td>32</td>
<td>8.796</td>
<td>5.782</td>
</tr>
<tr>
<td>64</td>
<td>17.999</td>
<td>11.855</td>
</tr>
</tbody>
</table>

- Normalised computation times

![Graph showing normalised computation times for Gauss-Seidel and Jacobi methods with increasing number of threads.](image)
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
   - Methodology
   - 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!