WCET analysis of a parallel 3D multigrid solver executed on the MERASA multi-core

Christine Rochange, Armelle Bonenfant, Pascal Sainrat (U. Toulouse)
Mike Gerdes, Julian Wolf, Theo Ungerer (U. Augsburg)
Zlatko Petrov, Frantisek Mikulu (Honeywell)

Workshop on WCET Analysis, July 2010
Motivation

- **Hardware trends**
  - common **requirements**
    - low energy consumption
    - high integration / low cost
    - high performance
  - architectural solution: **multicore processors**

- **New issues**
  - time-predictable multicore architectures
    - projects: **MERASA**, **PREDATOR**
  - parallel programming of embedded applications
  - **WCET analysis of parallel programs**
    - reporting experiments on a data parallel industrial application
Objectives:
- develop multi-core processor designs (from 2 to 16 cores) and system-level software support for mixed-critical workloads
- provide WCET analysis tools
  - static analysis: OTAWA
  - hybrid measurement-based: RapiTime
- evaluate with pilot studies submitted by industrial partners

Partners:
- Participants: Univ. Augsburg, Barcelona Supercomputing Center, Univ. Toulouse, Rapita Systems Ltd., Honeywell
- Industrial Advisory Board: Airbus, BAUER Maschinen GmbH, European Space Agency, Infineon, NXP
The MERASA multicore and system software

- **Isolation of critical threads**
  - specific scheduling in pipeline
  - time-predictable bus
  - partitioned L2 cache
  - time-predictable memory controller

- **System software**
  - time-predictable memory management
  - POSIX-compliant time-predictable synchronizations
  - two-level thread scheduling
The 3D multigrid solver

- **Part of application for airbone collision avoidance (Honeywell)**
  - path planning using Laplace’s equation
  - numerical solution:
    - domain partitioning → voxels
    - set potential of each point to average of neighbors
    - several iterations
  - multigrid technique:
    - several phases, varying grid size

- **Version considered in this study**
  - three phases -- each phase = interpolation step + iteration step

```java
for (x=0; x<NX; x++)
  for (y=0; y<NY; y++)
    for (z=0; z<NZ; z++)
      v[x][y][z] = compInterpolate(old_v);

for (i=0; i<NUM_ITE; i++)
  for (x=0; x<NX; x++)
    for (y=0; y<NY; y++)
      for (z=0; z<NZ; z++)
        v[x][y][z] = compIterate(v);
```
The parallelized 3D multigrid solver

- **Domain splitting**
  - 2 threads
  - 4 threads
  - 8 threads

- **Orchestration**
  - main thread
    - `readImage();`
    - `initiate P1S1(interpolation)`
    - wait for child threads
    - `initiate P1S2(iteration)`
    - wait for child threads
    - `... // same for P2 and P3`
  - child thread #1
    - `wait for main thread`
    - `process P1S1(interpolation)`
    - wait for main thread
    - `process P1S2(iteration)`
    - `... // same for P2 and P3`
  - child thread #2
    - `wait for main thread`
    - `process P1S1(interpolation)`
    - `wait for main thread`
    - `process P1S2(iteration)`
    - `... // same for P2 and P3`
Breakdown of the WCET

- **First level**
  - read Image
  - init P1S1
  - process P1S1
  - init P1S2
  - process P1S2
  - init P2S1

- **Second level**

N iterations:
1. $2W_n \times N$  
   (2 threads)
2. $3W_n + 2W_n \times (N-1)$  
   (4 threads)
3. $4W_n + 2W_n \times (N-1)$  
   (8 threads)
Analysis of synchronizations

- Example: mutex lock

```c
int mutex_lock(mutex_t mutex){
    (1) sl_lock(&mutex->guard);
    ...
    while (trylock(&mutex->thelock){
        ... // insert thread into queue
    (2) sl_unlock_and_set_suspended(&mutex->guard);
    (3) spinlock_lock(&mutex->guard); // on wakeup
    }
    ... 
    (4) sl_unlock(&mutex->guard);
}
```
### Analysis of synchronizations (cont’d)

<table>
<thead>
<tr>
<th>Term</th>
<th>depends on …</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># threads</td>
</tr>
<tr>
<td>$T_e$</td>
<td>no</td>
</tr>
<tr>
<td>$T_{w1}$</td>
<td>no</td>
</tr>
<tr>
<td>$T_{w2}$</td>
<td>yes</td>
</tr>
<tr>
<td>$T_{w3}$</td>
<td>yes</td>
</tr>
</tbody>
</table>

- $T_e$: Execution time when the variable is free
- $T_{w1}$: Overhead execution time when the thread has to wait for the variable
- $T_{w2}$: Waiting time related to system-level variables
- $T_{w3}$: Waiting time related to application-level variables

---

![Diagram](attachment:image.png)
Experimental results

- **Estimated WCETs** (OTAWA – perfect ISP – no L2 cache)

<table>
<thead>
<tr>
<th># threads</th>
<th>1</th>
<th>2+1</th>
<th>4+1</th>
<th>8+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>53,990,765</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3 cores</td>
<td>58,297,375</td>
<td>66,849,369</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5 cores</td>
<td>62,603,985</td>
<td>73,372,109</td>
<td>40,389,428</td>
<td>-</td>
</tr>
<tr>
<td>9 cores</td>
<td>71,217,205</td>
<td>86,417,589</td>
<td>47,678,968</td>
<td>28,105,761</td>
</tr>
</tbody>
</table>

- **WCET breakdown**
  - 8+1 threads, 9 cores

  **Computation thread without waits**
  - **Time to wait for producers (iteration step)**
  - Main thread
  - Synchronizations
Lessons learnt and future work

- **WCET analysis of a parallel application requires:**
  - understanding the structure of the application
    - dependency analysis
  - analyzing the synchronizations
    - where?
    - number of concurrent threads?
    - paths on which locks are hold

- **Automatic WCET analysis of parallel code:**
  - retrieve required information (structure + synchronizations)
    - source/object code analysis
    - user annotations
  - orchestrate the WCET computation

- **Parallelisation guidelines**
  - to facilitate WCET analysis
Questions?