Demo

Timing Analysis Solutions for Multicore Systems

Luis Miguel Pinho

CISTER-TR-170601
Timing Analysis Solutions for Multicore Systems

Luis Miguel Pinho
*CISTER Research Centre
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340509, Fax: +351.22.8321159
E-mail: http://www.cister.isep.ipp.pt

Abstract
Timing analysis solutions for multicore systems

Luis Miguel Pinho
ISEP
Outline

• P-SOCRATES at a glance
  • Time predictability and high-performance on parallel architectures

• Quick review of WCET estimation techniques
  • pros & cons and their applicability in P-SOCRATES

• The methodology in the P-SOCRATES SDK
Quick fact sheet

- **P-SOCRATES**: Parallel Software framework for time-Critical Many-core Systems
- Three-year FP7 STREP project (Oct-2013, Dec-2016)
- **Website**: [www.p-socrates.eu](http://www.p-socrates.eu)
- Budget: 3.6 M€
- Partners

[Logos of collaborating institutions]
Industrial Advisory Board

• Review and prioritize requirements, ensure that the project is kept on focus, analyze and validate the results

• Members:

City of Bratislava
Motivation

Embedded Computing

High Performance Computing

Demand of increased performance with guaranteed processing times
Vision

next-generation embedded many-core accelerators

real-time methodologies to provide time predictability

programmability of many-core from high-performance computing
Vision
Vision
Innovation

• A **generic framework**, integrating models, tools and system software, to parallelize applications with **high performance** and **real-time** requirements
for(int i=0; i<3; i++) {
    for(int j=0; j<3; j++) {
        if(i==0 && j==0) { // Task T1
            #pragma omp task depend(inout:m[i][j])
            compute_block(i, j);
        } else if (i == 0) { // Task T2
            #pragma omp task depend(in:m[i][j-1], inout:m[i][j])
            compute_block(i, j);
        } else if (j == 0) { // Task T3
            #pragma omp task depend(in:m[i-1][j], inout:m[i][j])
            compute_block(i, j);
        } else { // Task T4
            #pragma omp task depend(in:m[i-1][j], m[i][j-1],
                                   m[i-1][j-1], inout:m[i][j])
            compute_block(i, j);
        }
    }
}
P-SOCRATES TA Objectives

- Compiler
  - Annotate every node with a WCET estimate

- Schedulability analysis
  - Project developed new schedulability analysis for parallel graphs
Challenges

Safety-critical systems

“Simple” functions → More complex
Programming and design guidelines → No guidelines
Well written and structured code
Simple and predictable hardware → More powerful

strong and reliable V&V processes and tools

Business-critical systems

Performance-critical systems
Reviewed WCET techniques

• Static WCET techniques
• Measurement-based techniques
• Hybrid techniques
• Probabilistic WCET
Static WCET techniques

Phase 1: Flow analysis
Identify the feasible execution path in a program
Static WCET techniques

Phase 1: Flow analysis
Identify the feasible execution path in a program

```c
void main (X) {
    A = funcA(X) ;
    if (A > 10) {
        B = B - A;
    } else {
        B = B + A;
    }
}
```

```c
void main (X) {
    A = funcA(X) ;
    eval (A > 10);
    B = B - A;
}
```

```c
void main (X) {
    A = funcA(X) ;
    eval (A > 10) ;
    B = B + A;
}
```
Static WCET techniques

Phase 1: Flow analysis
Identify the feasible execution path in a program

```c
void main (X) {
    A = funcA(X) ;
    if (A > 10) {
        B = B - A;
    } else {
        B = B + A;
    }
}
```
Phase 2: Low-level analysis

void main (X) {
    A = funcA(X);
    eval (A > 10);
    B = B - A;
}

void main (X) {
    A = funcA(X);
    eval (A > 10);
    B = B + A;
}

∑ = 5750 ns

∑ = 7050 ns
Pros & cons

+ No need to have the actual hardware available

+ Years of experience, reliability proven for simple embedded processors -> very efficient for SC applications

- Little support for multicores

- Long time-to-market due to the inherent complexity

- V&V issues and associated cost

- Accuracy for more complex platforms? (how to deal with IPs in COTS?)
Measurement-based techniques
Pros & cons

+ Estimations available immediately (also, average, etc.)

+ No need to design accurate model -> reduced effort and cost

− Requires the hardware to be available, which may not be the case if the HW is developed in parallel with the SW

− Difficult to set up an environment which acts like the final system

− Intrusive instrumentation code

− Exhaustive testing is impossible
Hybrid techniques

• Combine the merits of static and measurement-based analysis techniques.

X, Y = read();
If (X < 50) then
  // some code
Else
  if Y > 100 then
    // some code
  else
    // some code
  end if
End if
Pros & cons

Do not rely on complex models

Provide accurate estimates

Intrusive instrumentation code

Exhaustive testing is impossible
Probabilistic WCET techniques

- Static pWCET

```c
void main (X) {
    A = funcA(X) ;
    eval (A > 10);
    B = B - A;
}
```

Pipeline

Cache[s]

\[\sum = 5750 \text{ ns}\]
Probabilistic WCET techniques

• Measurement-based
Pros & cons

Allow to derive estimates with confidence level

Require specific hardware support (randomization)

Not mature enough and controversial
Challenges

- Assessed existing tools and methodologies against these new settings and requirements

Static analysis is out the window

Not because of the complexity of the architecture!
Because of the architecture, the programming model, the OS and runtime, the man-power, the rapid evolution of the hardware...
Challenges

✓ **Portable** (out-of-the-box)
The provided tools should be “easily” portable from one platform to another
What we have done

Develop a measurement-based *trace-collecting tool*

Collecting runtime execution traces is **fully automatic**
(process of 12 subsequent steps for the Kalray MPPA)
What we have done

Develop a measurement-based **trace-collecting tool**

Every step is **well defined** and is adaptable to other platforms **with minimal effort**
What we have done

Develop a measurement-based trace-collecting tool

Written in Python 3.4
- cross-platform language
- Can be easily combined with other programming languages
Methodology

- A new approach to tackle the interference problem

Not one but two WCET estimates

One estimate is obtained by running every task in complete isolation (runs on 1 core, the rest of the system stays quiet)
Methodology

- A new approach to tackle the interference problem

Not one but two WCET estimates

The other is obtained by running every task in complete contention (runs on 1 core, the rest of the system does everything possible to interfere with its execution)
Methodology

- A new approach to tackle the interference problem

The gap between ISO and CONT is sometimes huge!
Slow down factor between 7 and 8 in average

Very negative impact on the global schedulability analysis
Methodology

Measurements taken in isolation are not safe as the execution time is subject to variation due to the shared resources.

Measurements taken in a totally congested system are not meaningful.

⇒ **Design processes** that creates a controllable interference on every shared resource.

⇒ **Investigate** how we can re-create a system activity similar to that of the final system.
Methodology

• Processes to perform schedulability analysis
  – Based on both intrinsic and extrinsic WCET estimates
  – One process for the dynamic project approach
    • Task-to-thread mapping is with global queue
    • Thread scheduling is global with limited preemption
    • Maximize average performance
  – Another for the static process approach
    • Fixed task-to-thread mapping (heuristics to minimize makespan)
    • Partitioned per-core scheduling (with limited preemption)
    • Minimize guaranteed response time
for(int i=0; i<3; i++) {
    for(int j=0; j<3; j++) {
        if(i==0 && j==0) { // Task T1
            #pragma omp task depend(inout:m[i][j])
            compute_block(i, j);
        } else if (i == 0) { // Task T2
            #pragma omp task depend(in:m[i][j-1], inout:m[i][j])
            compute_block(i, j);
        } else if (j == 0) { // Task T3
            #pragma omp task depend(in:m[i-1][j], inout:m[i][j])
            compute_block(i, j);
        } else { // Task T4
            #pragma omp task depend(in:m[i-1][j], m[i][j-1],
                                  m[i-1][j-1], inout:m[i][j])
            compute_block(i, j);
        }
    }
}

Annotate the graph with the WCET in CONTENTION

Sched

Yes
No
for(int i=0; i<3; i++) {
    for(int j=0; j<3; j++) {
        if(i==0 && j==0) {
            // Task T1
            #pragma omp task depend(inout:m[i][j])
            compute_block(i, j);
        } else if (i == 0) {
            // Task T2
            #pragma omp task depend(in:m[i][j-1], inout:m[i][j])
            compute_block(i, j);
        } else if (j == 0) {
            // Task T3
            #pragma omp task depend(in:m[i-1][j], inout:m[i][j])
            compute_block(i, j);
        } else {
            // Task T4
            #pragma omp task depend(in:m[i-1][j], m[i][j-1],
                                 m[i-1][j-1], inout:m[i][j])
            compute_block(i, j);
        }
    }
}

Annotate the graph with the WCET in CONTENTION
The big picture (static)

for(int i=0; i<3; i++) {
    for(int j=0; j<3; j++) {
        if (i==0 && j==0) {  // Task T1
            #pragma omp task depend(inout:m[i][j])
            compute_block(i, j);
        } else if (i == 0) {  // Task T2
            #pragma omp task depend(in:m[i][j-1], inout:m[i][j])
            compute_block(i, j);
        } else if (j == 0) {  // Task T3
            #pragma omp task depend(in:m[i-1][j], inout:m[i][j])
            compute_block(i, j);
        } else {  // Task T4
            #pragma omp task depend(in:m[i-1][j],m[i][j-1],
            m[i-1][j-1], inout:m[i][j])
            compute_block(i, j);
        }
    }
}

Annotate the graph with the WCET in CONTENTION
The big picture (static)

Annotate the graph with the WCET in ISOLATION

SUCCESS!
The big picture (static)

Map

Sched

Time

WCET

ISO

Annotate the graph with the WCET in ISOLATION

0100010
1100101
1100110

Yes
No

WCET
CONT

Yes
No

WCET
ISO

WCET
ISO

Map

Sched

Yes
No
The dynamic configuration approach achieves the same average performance than the default SDK.

Static approach achieves higher **Guaranteed Performance**, with similar average performance (~10%).
Open research problems

- Reduce the pessimism of the WCET estimates
Open research problems

- Reduce the pessimism of the WCET estimates
Open research problems

• **Intrinsic:** missing a path that leads to the WCET

  Similar problem as on single-core systems

  – Little we can do here 😞 apart from improving the “path exploration” process.
  – Powerful tools exist to guarantee code coverage. Those may turn to be useful to help find the longest path.

• **Extrinsic:** not observing the maximum interference

  – Extremely likely to happen, if not certain
  – Can we use this information? Can we sort of “extrapolate” the observations to guess the worst-case and possibly adjust/define the safety margin accordingly?
Open research problems

From the traces obtained in isolation and contention modes, we want to analyze how sensitive to concurrent activity the analyzed task really is

- Define safety margin accordingly
- Make recommendation to set up the environment in an appropriate way: dynamic vs. static mapping, PREM, ...
- Restrict the runtime, capture the maximum activity and map it to a pre-defined level of interference intensity created by a “tunable IG”
Post-project work partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234).

http://www.upscale-sdk.com/

Thank you