Estimating Worst-Case Bounds for Open CPS Runtimes with Genetic Algorithms

Oliver Horst, Uwe Baumgarten, Christian Prehofer

fortiss GmbH
An-Institut Technische Universität München
1. Problem Statement

2. Related Work

3. Utilizing Genetic Algorithms to Estimate Worst-Case Bounds of Open CPS Runtime APIs

4. Research Challenge No.1:
   Defining the Level of Detail of the Hardware Model

5. Research Challenge No.2:
   Quantifying the Quality of the Determined Bounds

6. Conclusion
Characteristics of Open CPS

- Open CPS are CPS that can be extended at runtime with new functionality (so called Apps)
- The final deployment configuration is unknown until runtime
- Only the platform (i.e., runtime and hardware) is known, but everything on top is unknown

Thus, it is important to determine the WCET of each App individually and independently
Characteristics of Open CPS Runtime APIs

- Reduced amount of APIs with focused functionality to lower the risk of attacks
- Interactions between apps and to the hardware are managed and coordinated by the runtime

The API guarantees apps a compositionality with respect to functional and non-functional (e.g., timing) properties, according to [2,3]
Compositional WCET Analysis

The WCET of each app can be calculated depending on the WCET of the API calls of the runtime.

During deployment the composite WCET can be calculated [1].

Allows to make claims on real-time properties for deployed apps.
Related Work

Determining WCETs for API Calls in Open Platforms

Static Analysis

- Faces some limitations
  - Restrictions in reconstructing the CFG
  - Complexity of the underlying hardware, such as, multi-core interference, caches and pipelines
- Requires the final, complete binary
- Determines safe upper bounds for the WCET

Analysis by Genetic Algorithms

- Used in three related areas
  - Search based WCET analysis
  - Evolutionary testing of binaries
  - Deriving stress test for task schedulers
- It has been shown that GA-based approaches can obtain WCETs in the range of 3-10% of predetermined safe upper bounds [4]
- Determines only a lower bound for the WCET
### Estimating Worst-Case Bounds with GAs

#### The General Idea

The GA tries to find an execution scenario that provokes the BCET or WCET of a particular API call.

Each scenario is evaluated for its timing by a model or a simulation to steer the evolution process.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Middleware</td>
<td></td>
</tr>
<tr>
<td>Task Scheduler</td>
<td></td>
</tr>
<tr>
<td>Resource Arbiter</td>
<td></td>
</tr>
<tr>
<td>HW Model, or Simulation</td>
<td></td>
</tr>
</tbody>
</table>

**Simulation / Model**

- Communication Middleware
- Task Scheduler
- Resource Arbiter
- HW Model, or Simulation

**Genetic Algorithm**

1. **simulate**
2. **assess**
3. **mutate**
4. **recombine**
5. **select**
Scenario: Definition and Evaluation

- A scenario models the interaction of tasks with the runtime, interfered by interrupts.
- Each generated scenario consists of three sets:
  - Task set
  - Interrupt set
  - Message set defined over a fixed time interval
- Our model targets platforms with a finite impulse response.
- Determining the WCET/BCET requires the evaluation of three full system cycles.
- The platform specifies the length of one system cycle.
- Programming errors, e.g., buffer overflows, and timing impacts of specific app instructions onto the runtime code are explicitly excluded from our considerations.
Scenario: An Example

Task and Interrupt Set

Core 0

T[0]: 14, S
I[0]: 2
T[1]: 12, S
T[2]: 8, S

Core 1

Message Set

Task 0
M[0]: 6

Task 1
M[1]: 4
M[2]: 8

Task 2
M[3]: 2
M[4]: 2

- Suggested Approach -

Estimating Worst-Case Bounds for Open CPS Runtimes with GAs © fortiss GmbH
Defining the Level of Detail of the HW Model

Research Challenge No.1

- Tension in the field of slow and precise or fast and approximated results
- Benchmarks indicate that $10^5$ calls to the fitness function are needed per dimension for good results \cite{5}
- We assume a complexity of $\sim10^3$ dimensions
- We see two relevant options:
  - Abstract HW Model
  - Cycle Accurate Model
Relevant Hardware Model Options

**Abstract HW Model**
- Ideally, models all hardware internals like bus and device arbitration schemes, memory delays, caches, etc.
- Needs to consider a lot of components and special cases

- How and at which level would you model the hardware?
- How close could the model reproduce the accurate timing?

**Cycle Accurate Model**
- Tests the whole software stack as is with the generated scenario as stimuli
- Can determine very precise and realistic timing

- What are possible simulation environments you would consider as suitable?
- Should something in-between the two options be considered?
Formally proving the convergence of metaheuristics, such as genetic algorithms, is a relatively young research area.

Proofs for simple versions of the algorithm and standard problems exist, but assume an infinite runtime.

Considerations for the convergence speed of metaheuristics are still not mature and very limited with respect to their application fields.

At the moment, it cannot be formally proven that after a certain number of generations a given quality goal is reached!
Non-formal Quantification Options

Comparison against known WCET bounds

- Determine the WCET for known and representative deployment configurations with the help of static-analysis tools
- Compare the results of the GA with the obtained results

Empirical study on mathematically similar functions

- Study the convergence quality of the GA with mathematical functions that calculate the timing for a given scenario and rebuild the structure of the real problem as close as possible

Which of the two variants do you consider as more suitable?

How many samples would you consider as sufficient to prove the convergence quality of the presented approach?
Conclusion

- Characteristics of open CPS and their APIs
- Approach to estimate worst-case bounds of open CPS runtime API calls with genetic algorithms, based on the generation of execution scenarios, which consists of three sets:
  - Task set
  - Interrupt set
  - Message set

Two open research problems:

1. Defining the level of detail of the hardware model
2. Quantifying the quality of the determined bounds
Contact // Oliver Horst

**fortiss GmbH**
An-Institut Technische Universität München
Guerickestraße 25 · 80805 München · Germany

tel +49 89 3603522 15    fax +49 89 3603522 50

oliver.horst@fortiss.org
www.fortiss.org


Mathematical Characterization of the Problem (1/2)

To assess the suitability of specific genetic algorithms for tackling our problem, a mathematical test function is needed. In the mathematical sense, we cope with a discrete function, which is costly to evaluate and is defined by

- multiple dimensions (e.g., > 1,000), with
- many local maxima,
- multiple global maxima,
- plateaus, valleys, and
- cross linked dimensions.

The Black-Box-Optimization-Benchmarking (BBOB) workshop developed a number of benchmarking functions for genetic algorithms. Six of them match our problem statement and could be used to evaluate the suitability of genetic algorithms.
Mathematic Characterization of the Problem (2/2)

A BBOB Function Selection

- **f105**: Rosenbrock with moderate uniform noise
- **f113**: Step ellipsoid with gaussian noise
- **f119**: Different Powers with gaussian noise
- **f122**: Schaffer’s F7 with gaussian noise
- **f126**: Composite Griewank-Rosenbrock with uniform noise
- **f128**: Gallagher’s Gaussian Peaks 101-me with gaussian noise
## Relevant Hardware Model Properties

At the Example of Shared Resources on Multicore Systems

<table>
<thead>
<tr>
<th>Shared resource</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>System bus</td>
<td>Contention by multiple cores</td>
</tr>
<tr>
<td></td>
<td>Contention by other device - IO, DMA, etc.</td>
</tr>
<tr>
<td></td>
<td>Contention by coherency mechanism traffic</td>
</tr>
<tr>
<td>Bridges</td>
<td>Contention by other connected busses</td>
</tr>
<tr>
<td>Memory bus and controller</td>
<td>Concurrent access</td>
</tr>
<tr>
<td>Memory (DRAM)</td>
<td>Interleaved access by multiple cores causes address set-up delay</td>
</tr>
<tr>
<td></td>
<td>Delay by memory refresh</td>
</tr>
<tr>
<td>Shared cache</td>
<td>Cache line eviction</td>
</tr>
<tr>
<td></td>
<td>Contention due to concurrent access</td>
</tr>
<tr>
<td></td>
<td>Coherency: Read delayed due to invalidated entry</td>
</tr>
<tr>
<td></td>
<td>Coherency: Delay due to contention by coherency mechanism read by lower level cache</td>
</tr>
<tr>
<td></td>
<td>Coherency: Contention by coherency mechanism on this level</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shared resource</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local cache</td>
<td>Coherency: Read delayed due to invalidated entry</td>
</tr>
<tr>
<td></td>
<td>Coherency: Contention by coherency mechanism read</td>
</tr>
<tr>
<td>TLBs</td>
<td>Coherency overhead</td>
</tr>
<tr>
<td>Addressable devices</td>
<td>Overhead of locking mechanism accessing the memory</td>
</tr>
<tr>
<td></td>
<td>I/O Device state altered by other thread/application</td>
</tr>
<tr>
<td></td>
<td>Interrupt routing overhead</td>
</tr>
<tr>
<td></td>
<td>Contention on the addressable device - e.g. DMA, Interrupt controller, etc.</td>
</tr>
<tr>
<td></td>
<td>Synchronous access of other bus by the addressable device (e.g. DMA)</td>
</tr>
<tr>
<td>Pipeline stages</td>
<td>Contention by parallel hyperthreads</td>
</tr>
<tr>
<td>Logical units</td>
<td>Contention by parallel applications</td>
</tr>
<tr>
<td></td>
<td>Other platform-specific effects, e.g. BIOS Handlers, Automated task migration, Cache stashing, etc.</td>
</tr>
</tbody>
</table>