STSM Scientific Report

Title: **Generic Task Set Generator for Schedulability Analysis using the TACLeBench benchmark suite**

Home Institution: University of Antwerp, Prof dr. Peter Hellinckx

Host Institution: University of Luxembourg, dr. Sebastian Altmeyer

Purpose of the STSM

The purpose of STSM is the create a task set generation tool. This tool generates task sets based on input parameters (period range, global utilization, target hardware, etc.). The output of the task set generator is one or more task sets with for each task a combination of benchmark programs. The summation of the WCET of each program, equals the wanted WCET of the task[1] (including cache effects).

The goal of the STSM was to develop a first version of this tool, using a small set of benchmark programs.

Work accomplished during the STSM and main results

The tool is split into three parts:

1. The generation the numeric task sets
2. The creation of a benchmark program sequence for each task.
3. Combining the benchmark programs and the compilation of the code for the right target hardware.

During the STSM the first two steps were accomplished in a first version of the program.

First, a set of numeric task sets is generated, this means that only the values of the execution time, deadline and period are generated. In Figure 1, an example of a numeric task set is given. The task sets are generated based in the input parameters, defined by the user (Figure 2).

```
<taskSet name="TaskSet_9_320.0_v3" load="320.0">
<task name="Task 1" p="122000.0" d="122000.0" e="45016.0" />
<task name="Task 2" p="234000.0" d="234000.0" e="46220.0" />
<task name="Task 3" p="328000.0" d="328000.0" e="115343.0" />
<task name="Task 4" p="442000.0" d="442000.0" e="348822.0" />
<task name="Task 5" p="451000.0" d="451000.0" e="81149.0" />
<task name="Task 6" p="539000.0" d="539000.0" e="59980.0" />
<task name="Task 7" p="571000.0" d="571000.0" e="395451.0" />
<task name="Task 8" p="619000.0" d="619000.0" e="95946.0" />
<task name="Task 9" p="697000.0" d="697000.0" e="65190.0" />
<task name="Task 10" p="697000.0" d="697000.0" e="181462.0" />
</taskSet>
```

Figure 1: A numeric task set

[1] The user can define the maximum deviation of the combined WCET compared to the wanted WCET. This combined WCET can only deviate downwards, the wanted WCET is an upper boundary, which may never be exceeded.

```
<taskSet>
<parameter name="name" value="yorick" />
<parameter name="tasksetlocation" value="C:\Users\yorick\Desktop" />
<parameter name="globalUtilizationMin" value="3"/>
<parameter name="globalUtilizationMax" value="4"/>
<parameter name="utilizationStep" value="0.2"/>
<parameter name="numberOfTasks" value="10"/>
<parameter name="numberOfTaskSets" value="5"/>
<parameter name="periodMin" value="100000"/>
<parameter name="periodMax" value="700000"/>
<parameter name="periodStep" value="1000"/>
<parameter name="seed" value="20" />
</taskSet>
```

Figure 2: Input parameters for the task set generator

In this example the task set generator will generate 5 task sets for each global utilization between 3 and 4 (3.0, 3.2, 3.4, 3.6, 3.8 and 4.0). Every task set contains 10 tasks. The utilization of the tasks of a task set is generated using the UUnifast Algorithm (Figure 3). The task sets are written to a XML file (yorick.xml).

```
function vectU = UUniFast(n, U)
sumU = U;
  for i=1:n-1,
  nextSumU = sumU.*rand^(1/(n-i));
  vectU(i) = sumU - nextSumU;
  sumU = nextSumU;
end
vectU(n) = USum;
```

Figure 3: Algorithm to uniformly distribute the load between the tasks. With n the number of tasks and U the global utilization of the task set

The advantage of the tool is that the input of the first part is loosely coupled to the second part. Which means that other techniques can be used to generate task sets, as long as the output uses the same template as the current implemented template in Figure 1.

The second part, creates a benchmark program sequence for each task of a task set. Based on the input parameter for the target hardware and benchmark programs (Figure 4), a set of fitting programs is selected to create the sequence. First, the tool selects the benchmark programs which has a WCET for the selected target hardware .

```
<targetHardware>
<parameter name="architecture" value="TriCore"/>
<parameter name="CPUSpeedMHz" value="150"/>
<parameter name="cores" value="4"/>
</targetHardware>
<benchmark>
<parameter name="location" value="C:/TACLeBench/"/>
<parameter name="floatingPoint" value="no" default="no"/>
</benchmark>
```

Figure 4: Input parameters for the target hardware and the benchmark suite

In the current version, seven benchmark programs from the TACLeBench benchmark suite are used. For each program, a description file is linked. This file contains the location of the program, one or more WCET-architecture combinations and additional extra user-defined parameters.

To create the program sequence for a task, a Binary/Integer Linear Problem solver is used to maximize the following problem:

$$maximize \sum_{i=1}^{n} x_i * e_i$$

$$subject\ to \sum x_i * e_i \leq E \qquad i = 1, \dots, n$$

$$x_i \geq 0 \qquad i = 1, \dots, n$$

With $\{e_0, e_1, \dots, e_n\}$ the WCETs of the set of suitable benchmark programs, and E the wanted WCET of the task, generated in the first part. This equation is a Binary Linear Programming problem, if x is only 0 or 1. This means, for a task, a benchmark program can only be used once. When the deviation of the summation of the WCETs of the result of the binary linear programming problem, is bigger than 2 % (user-defined) compared to the wanted WCET, the linear programming problem is solved again, but x can be a positive integer value.

By using a benchmark program multiple times in one task, has an effect on the cache, thereby on the WCET. Two options were proposed:

1. Cache Flush: after each program execution, flush the cache. When the same program is executed for the second time or more, it is executed as like for the first time.
2. Cold and warm cache: when a program is executed for the second time or more, right after its previous execution, code of the program is located in the cache, which results in a shorter execution time. Basically, two execution times can be calculated for a program: cold cache execution, the program executes with no code in the cache. And warm cache execution, the program executes with code in the cache.

In the current version of the tool, the first option is included to calculated the program sequence, the user can define the time it takes to flush the cache. Each time a program executed, the cache flush time is added to the real execution time.

This solution is the most easy to implement, but also the most radical solution. In some cases, a cache flush is not possible because of OS requirements, etc. In the further development of the tool, better solutions are required.

After the program sequence for each task is calculated, a XML file is created, including the benchmark programs and the number of executions of each program (Figure 5).

```
<task name="Task 7" p="415000.0" d="415000.0" wanted_e="88069.0" real_e="87598.99" >
<program name="adpcm_g721_board_test" n="21" />
<program name="bitonic" n="6" />
<program name="codecs_codrle1" n="5" />
<program name="codecs_dcodhuff" n="13" />
<program name="crc" n="1" />
<program name="fft1" n="1" />
</task>
```

Figure 5: The output of the program sequence creator for one task

The *real_e* is the summation of the benchmark program's WCET, this does not includes the cache flush time after each program. This output is generated by the second part of the tool, including for each task set a folder with the used benchmark programs in the task set.

In our example we used 7 benchmark programs to generate 1000 tasks. 168 of them our solved with x as a binary, the others with x as an integer. By increasing the number of benchmark programs, the number of solutions with x as a binary will also increase.

Future collaboration

For future collaboration with the host institution, we will bring the task generation tool to a more mature level. To realize this, some additional development has to be done. This will be in collaboration with host institution.

First, a whole new part must be added to the task generation tool. After the task sequence creator, the benchmark programs need to be combined into to single c-file and compiled for the right target architecture. As a verification of the combined WCET time, the single c-file can be the input of the timing analysis tool, which was used the calculated the WCET of programs separately, and validate the results against the predicted WCET.

For the numeric task generator, adding different types of task sets is necessary to support a wide range testing areas. Harmonic periods are often used in embedded systems to increase the schedulability of a task set. Because of the harmonic periods, the scheduling of the tasks has a repeating pattern, which improves the schedulability analysis. Another type of tasks are explicit deadline tasks. The deadline of an explicit task, is smaller than or equal to the period of the task.

Regeneration of the same task set, with the same values, is a necessary to redo an experiment or to adjust one parameter of the task set (for example implicit to explicit tasks), without changing the values of the other parameters.

Parameterization of code from the benchmark programs can reduce the number of times the same programs executes within one task. By tweaking loop bounds, the WCET can be shorter or longer. This information about loops and loop durations needs to implemented into the program description file.

Most timing analysis express their outputs in number of clock cycles and not a fixed time. By dividing the number of cycles by the clock frequency of the processor, exact timing can be calculated. However, by using the number of clock cycles in the description file, add support for target hardware with the same architecture, but different clock frequency. The transformation from clock cycles to exact timings needs to be implemented into the tool.

Foreseen publications/articles to result from the STSM

A joint publication/article between the home and host institution is planned after the completion of the previous describe adjustments/additions, which results in to a much richer tool. The publication/article will present the task generation tool including validation using a timing analysis tool.