General Instructions

- You can download the source files for the exercise from:
  `https://strec.wp.mines-telecom.fr/TD-CACHE/`

Today's exercises will again be based on the `Otawa` Worst-Case Execution Time[1] analysis tool, the `Patmos` processor[2], and a simple flight control software (`Heli`). Please consult the last assignment for additional details regarding the tools and the used benchmark.

Using the Patmos Tools

In order to get access and use the Patmos tool chain on the command line you need to add the correspond paths to your environment as follows:

```
export PATH=$PATH:/infres/ir600/users/brandner/patmos-misc/local/bin/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:\
  /infres/ir600/users/brandner/patmos-misc/local/lib/:\
  /infres/ir600/users/brandner/patmos-misc/local/lib/otawa/proc/otawa/
```

[1] `http://www.otawa.fr/`
[2] `http://patmos.compute.dtu.dk/`

1 Instruction Cache Analysis (40 minutes)

Aims: *Understand the operation of a realistic analysis of the instruction cache.*

Download the archive `TD-CACHE-STREC.tar.gz` for today's exercises from the course website as indicated above. After decompressing the archive you will find several files:

```
./heli
./heli.ff
./Makefile
./patmos_wcet
./patmos_wcet/caches.xml
./patmos_wcet/memory.xml
./patmos_wcet/pipeline.xml
./patmos_wcet.osx
./src/heli.c
./src/io.h
./ext.lp
```

Only the file `ext.lp` is new, in comparison to the last assignment, while the contents of other files changed. Most notably, the platform configuration file `./patmos_wcet.osx` now specifies analyses that target the instruction and data caches.

- Open the file that specifies the cache organization (`./patmos_wcet/caches.xml`). Verify that the current cache configuration for the data and instruction caches indicates: (i) a cache associativity of 4 (2^2), (ii) a cache block size of 32 bytes (2^5), (iii) and 512 (2^9) cache lines (rows).

  What is the total size of the cache?

- Try to vary the cache block size of the instruction cache and verify the impact on the obtained WCET bound by `Otawa`.

  Attention: Revert your changes after this question.

  What is the impact of reducing/increasing the cache block size?

- In the lecture we heard that a cache analysis reasons about *memory blocks* and whether those memory blocks are present in the cache or not. Open the CFG using `xdot` and try to find annotations that are related to the instruction cache (memory blocks, classification as hit/miss, ...).

  Hint: Search for the string "Instruction Cache". Also recall that the memory blocks have the same size as cache blocks.

- Have a close look at basic blocks 17 through 21, which are part of the function `fixFilter`.

  Hint: You can compute the address of the memory/cache block of a basic block by masking the 5 least-significant bits of the basic block's address.

  Which are the memory blocks considered by the analysis? What are their relations to the basic blocks mentioned above?

- From the observations in the previous exercise. How can the hit/miss classifications associated with the various basic blocks be explained?

How can you explain that the access to memory block `0x214c0` in basic block 19 is classified as `always-miss`. Why is the access to the same memory block in basic block 21 classified `always-hit`? Similarly, explain why the accesses to memory block `0x214e0` in basic block 20 and 21 are classified as `not-classified`?

- The observations from the previous exercise suggest that persistence analysis is not activated. Open the platform configuration file (`./patmos_wcet.osx`) and change the analysis parameter to activate persistence analysis.

  Search for the configuration parameter `otawa::FIRSTMISS_LEVEL` and change its value from `FML_NONE` to `FML_MULTI`.

  How does persistence analysis impact the obtained WCET bound? Check the result for the memory blocks associated with basic block 21 in the visualization of the CFG. What can you see there?

- Open the LP-file generated by `Otawa` and try to find the variables and equations associated with instruction cache misses.

  Hint: The variable names are constructed according to the following schema: `x<basic block number>_miss_<address>_<id>`.

  Find the variables related to the memory blocks of basic block 21. What do the corresponding equations mean? Do the variables appear in the objective function?

2 Data Cache Analysis (40 minutes)

Aims: *Understand the operation of a realistic analysis of the data cache.*

- Open the CFG using `xdot` and try to find information regarding the data cache (memory blocks, classification).

  Hint: Recall the characteristics of the data cache (see the previous exercise).

- Have a closer look at basic block 2, which corresponds to the `for` loop in function `calibrateGyro`. You will see that several loads and stores are considered by the data cache analysis. The first store is classified as `not-classified`, while all other memory accesses are classified `always-hit`.

  Explain what these memory accesses do and why `Otawa`'s data cache analysis computed these classifications. To which C code lines/statements do these accesses belong?

- Try to see whether the classification `not-classified` of the store in basic block 2 can be improved by persistence analysis. Search for the configuration parameter `otawa::dcache::FIRSTMISS_LEVEL` and set it to `DFML_MULTI`.

  Hint: The store evidently refers to the global array `gyro`. Can you say anything about its address or size. Try to determine the memory blocks associated with the array. You can use the following command in order to find the address of the array `gyro`. The command lists all symbols defined in the executable:

  ```
  patmos-llvm-objdump -t heli | grep gyro
  ```

  Is persistence analysis able to improve the classification of the corresponding store instruction in basic block 2 (or any other data access in the program)? Give an explanation why.

- `Otawa` supports an annotation in the flow fact file (`./heli.ff`) that allows to specify the address range of a given memory access.

  The annotation has the following format:

  ```
  memory access <instruction address> <range start> .. <range end>;
  ```

  Add a memory access annotation to the flow fact file for the store instruction at address `0x206c4`. Can the persistence analysis take advantage of the information?

- In the lecture you have heard that loop unrolling is frequently used in order to determine the precise addresses of memory accesses. Maybe this technique can help to find a better classification for problematic store of basic block 2?

  Open the platform configuration file and activate loop unrolling by uncommenting the line with the step `otawa::UNROLLED_LOOPS_FEATURE`. Note that `Otawa` only *peels* the first iteration off and does not unroll the entire loop.

  Hint: The CFG changes a lot when loop unrolling is activated. In particular, basic block numbers change throughout the basic block. In addition, some basic blocks are duplicated. For instance, try first to find the duplicates of the original basic block 2 in the CFG.

Look at the CFG using `xdot` and try to find differences due to loop unrolling. What happened to basic block 2? Did the classification of the problematic store improve?

3 Individual Contribution of Basic Blocks (15 minutes)

Aims: *Determine which parts contribute the most to the obtained WCET bound.*

- Even though unrolling did not improve the classification of the problematic store, apparently it was beneficial somewhere. But where and how did unrolling help the analysis?

  After solving the IPET problem, `Otawa` may, in addition to the obtained WCET bound, provide statistics on the individual contribution of basic blocks as well as addresses. This information is provided in file `./out/bbtimes.log`, which consists of two tables.

  The first table lists each basic block of the CFG, its local execution time, the value of its flow variable of the IPET, as well as its absolute and relative contribution to the WCET bound. Note that due to *inlining* and *unrolling* basic blocks might have been duplicated, i.e., the same code addresses appear several times as different basic blocks. The second table thus shows the accumulated contribution of each unique basic block address. The last line of this table also shows the global contribution of the instruction as well as the data cache.

  Hint: Copy the entire output directory in order to have the statistics along with the control-flow graph and the LP file, e.g., use `cp -r ./out/ ./out-unrolled`. Then use `diff` or `meld` to compare the files with the statistics. Focus on the table showing accumulated numbers first. In particular, look for basic block addresses whose accumulated execution times are above $10\,000$ cycles. Look at the cache states of the corresponding basic blocks in the visualization of the CFG.

  Compare the statistics in the tables with and without unrolling. Where did unrolling improve the analysis? Did the instruction or data cache profit the most from unrolling? Explain why unrolling actually helped.