Analysis of LLVMTA and Heptane for WCET calculation of dataflow actors

Saltuk Çelik

Supervised by
Florian Krebs

Department of Computer Science
RPTU Kaiserslautern-Landau

10 April 2024

Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Table of Contents

Table of Contents

What is Worst-Case Execution Time (WCET)?

- Maximum time it takes to execute a given piece of code
  - on a given machine
  - in a given application context (inputs, state) [1]

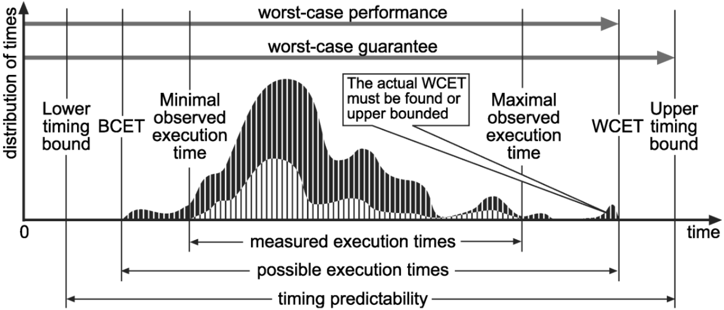
What is Worst-Case Execution Time (WCET)?

- Maximum time it takes to execute a given piece of code
  - on a given machine
  - in a given application context (inputs, state) [1]
- We are interested in WCET to
  - perform schedulability anaylsis
  - ensure meeting deadlines
  - assess resource needs for real-time systems [2]

What is Worst-Case Execution Time (WCET)?

- Maximum time it takes to execute a given piece of code
  - on a given machine
  - in a given application context (inputs, state) [1]
- We are interested in WCET to
  - perform schedulability anaylsis
  - ensure meeting deadlines
  - assess resource needs for real-time systems [2]
- For example the air bag control system in an automobile

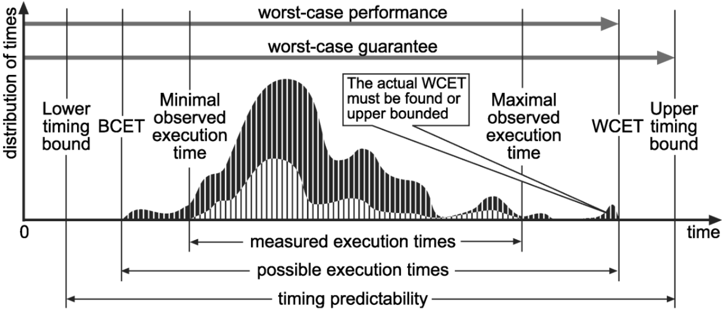
Main WCET methods



[3]

- WCET bounds must be safe and tight

Main WCET methods



[3]

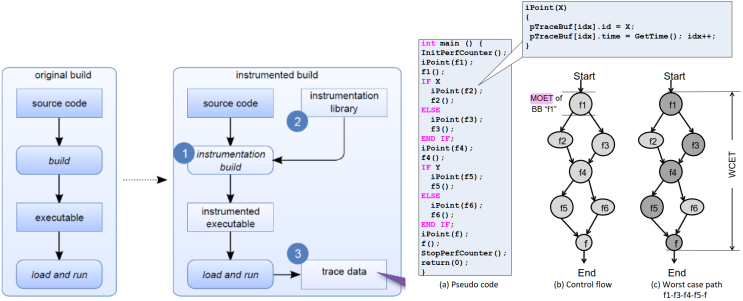
- WCET bounds must be safe and tight
- Main WCET methods are
  - Measurement analysis (optimistic, not all paths are tested)
  - Static analysis (pessimistic, worst theoretically possible WCET)

Measurement analysis

- Measurements of the basic-block execution times on the real HW processor (or a cycle-accurate simulator)
- Building of precise hardware model is not needed [4]

Measurement analysis

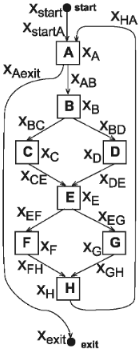
- Measurements of the basic-block execution times on the real HW processor (or a cycle-accurate simulator)
- Building of precise hardware model is not needed [4]



Rapitime, a commercial measurement based (hybrid) WCET tool [4]

Static analysis

1. Obtain the Control Flow Graph (CFG) and find possible paths



// Start and exit constraints

$x_{start} = 1, x_{exit} = 1$

// Structural constraints

$x_{start} = x_{startA}$

$x_A = x_{startA} + x_{HA} = x_{Aexit} + x_{AB}$

$x_B = x_{AB} = x_{BC} + x_{BD}$

$x_C = x_{BC} = x_{CE}$

...

$x_H = x_{FH} + x_{GH} = x_{HA}$

$x_{exit} = x_{Aexit}$

// Loopbound constraint

$x_A \leq 100$

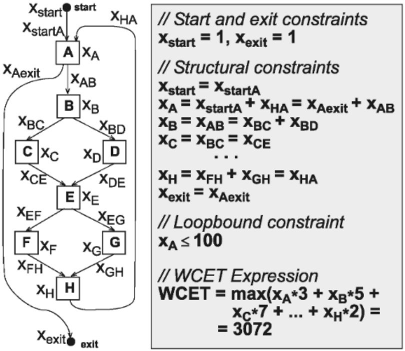
// WCET Expression

$WCET = \max(x_A*3 + x_B*5 + x_C*7 + \ldots + x_H*2) = 3072$

Implicit path enumeration technique (IPET) [3] which is used in LLVMTA and Heptane

Static analysis

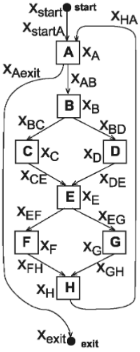
1. Obtain the Control Flow Graph (CFG) and find possible paths
2. Determine the possible execution times of blocks, accounting for the timing effects of microarchitectural features such as pipelining and caching



Implicit path enumeration technique (IPET) [3] which is used in LLVMTA and Heptane

Static analysis

1. Obtain the Control Flow Graph (CFG) and find possible paths
2. Determine the possible execution times of blocks, accounting for the timing effects of microarchitectural features such as pipelining and caching
3. Combine info from 1 and 2 to obtain upper timing bounds [5]



// Start and exit constraints

$x_{start} = 1$, $x_{exit} = 1$

// Structural constraints

$x_{start} = x_{startA}$

$x_A = x_{startA} + x_{HA} = x_{Aexit} + x_{AB}$

$x_B = x_{AB} = x_{BC} + x_{BD}$

$x_C = x_{BC} = x_{CE}$

...

$x_H = x_{FH} + x_{GH} = x_{HA}$

$x_{exit} = x_{Aexit}$

// Loopbound constraint

$x_A \leq 100$

// WCET Expression

$\text{WCET} = \max(x_A*3 + x_B*5 + x_C*7 + \ldots + x_H*2) =$
$= 3072$

Implicit path enumeration technique (IPET) [3] which is used in LLVMTA and Heptane

Table of Contents

LLVMTA

- An academic WCET analysis tool based on the (patched version of) LLVM compiler infrastructure [5]
- Developed in Saarland University

LLVMTA

- An academic WCET analysis tool based on the (patched version of) LLVM compiler infrastructure [5]
- Developed in Saarland University
- Aims for state-of-the-art analysis frameworks, in particular abstract execution graphs (AEG)

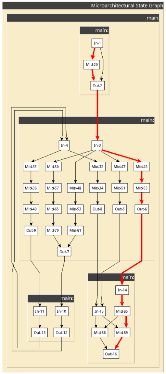
LLVMTA

- An academic WCET analysis tool based on the (patched version of) LLVM compiler infrastructure [5]
- Developed in Saarland University
- Aims for state-of-the-art analysis frameworks, in particular abstract execution graphs (AEG)
- Does not focus on the complexity of modeling real-world hardware architectures (like commercial tools, e.g. aiT)

Abstract execution graphs

- AEG may capture correlations between the timing contributions of different basic blocks, rather than computing a single bound for each basic block

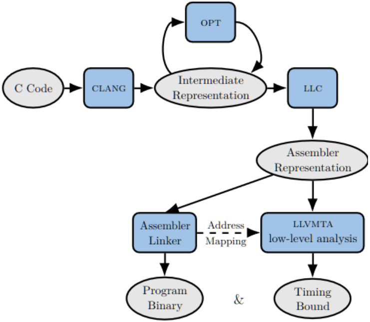
```c
int main() {
    int i = 0;
    while ( i < 50)
        ++i;
    return 0;
}
```

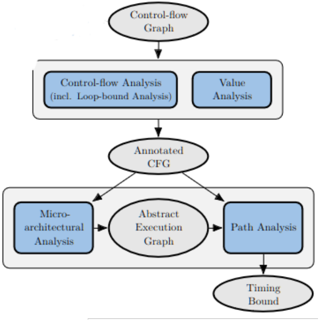


Multiple abstract microarchitectural states are created for the while loop in the main function allowing for higher analysis precision

LLVMTA architecture

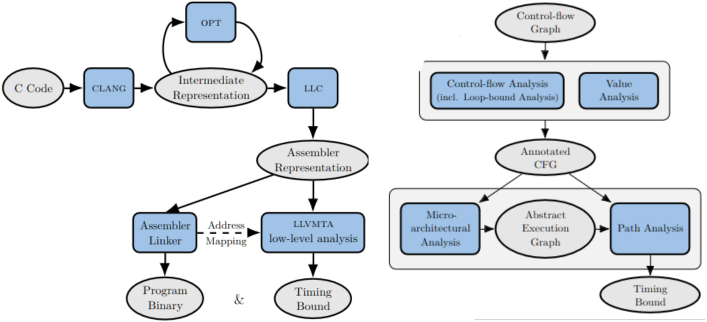
- Analysis are implemented on the final assembler representation in the LLVM backend which is the representation closest to the machine level





LLVMTA architecture

- Analysis are implemented on the final assembler representation in the LLVM backend which is the representation closest to the machine level



- Supports multiple ILP solvers: LPsolve, IBM CPLEX, and Gurobi Optimizer

Using LLVMTA

- As input, a program written in C is provided to LLVMTA

Using LLVMTA

- As input, a program written in C is provided to LLVMTA
- There are many options to specify the architecture, such as
  - ARM and RISC-V ISA
  - In-order, strictly in-order, and out-of-order execution pipelines
  - Scratchpad memories, as well as caches with least-recently-used replacement policy and both write-through and write-back policy

Using LLVMTA

- As input, a program written in C is provided to LLVMTA
- There are many options to specify the architecture, such as
  - ARM and RISC-V ISA
  - In-order, strictly in-order, and out-of-order execution pipelines
  - Scratchpad memories, as well as caches with least-recently-used replacement policy and both write-through and write-back policy
- LLVMTA can automatically derive loop bounds using LLVM's scalar evolution analysis in many cases. If LLVMTA fails to obtain a loop bound, the user has to provide loop bounds manually in a CSV format LoopAnnotations.csv

Using LLVMTA

- As input, a program written in C is provided to LLVMTA
- There are many options to specify the architecture, such as
  - ARM and RISC-V ISA
  - In-order, strictly in-order, and out-of-order execution pipelines
  - Scratchpad memories, as well as caches with least-recently-used replacement policy and both write-through and write-back policy
- LLVMTA can automatically derive loop bounds using LLVM's scalar evolution analysis in many cases. If LLVMTA fails to obtain a loop bound, the user has to provide loop bounds manually in a CSV format LoopAnnotations.csv
- Likewise the user has to provide annotations file for the external functions

An example

```c
int main() {
    int i = 0;
    while ( i < 50 ){
        i = printf(format: "%d\n", i);
    }
    return 0;
}
```

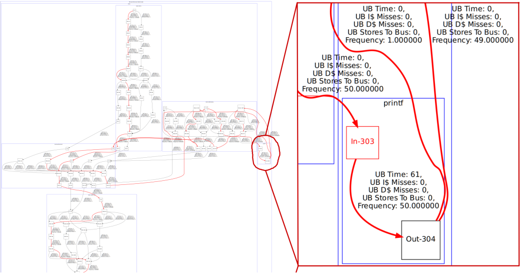
Both the loop bound and the external function printf are unknown

An example

```
int main() {
    int i = 0;
    while ( i < 50 ){
        i = printf(format: "%d\n", i);
    }
    return 0;
}
```

Both the loop bound and the external function printf are unknown

- The annotation files are given such that printf has 61 WCET and the loop has 50 iterations, the AEG becomes



LLVMTA limitations

- Since it operates on the machine-level IR rather than on the binary, results may be incorrect if
  - Another compiler is used for the binary
  - The assembler breaks down pseudo-assembly instructions used in the IR into several machine instructions in the actual binary

LLVMTA limitations

- Since it operates on the machine-level IR rather than on the binary, results may be incorrect if
    - Another compiler is used for the binary
    - The assembler breaks down pseudo-assembly instructions used in the IR into several machine instructions in the actual binary
- The tool is reasonable fast on the standard WCET benchmarks, but will likely not scale to real-world applications

Table of Contents

Heptane

- An open-source software program that estimates upper bounds of execution time [6]
- Developed in University of Rennes

Heptane

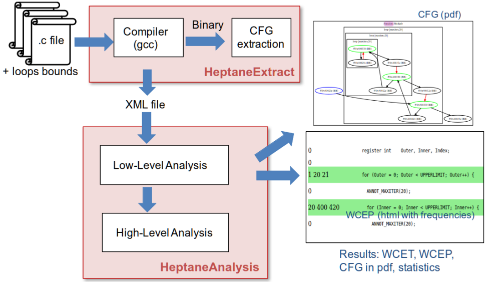
- An open-source software program that estimates upper bounds of execution time [6]
- Developed in University of Rennes
- Uses IPET as well, does contextual analysis, meaning the analysis of a function is performed for every call path

Heptane

- An open-source software program that estimates upper bounds of execution time [6]
- Developed in University of Rennes
- Uses IPET as well, does contextual analysis, meaning the analysis of a function is performed for every call path
- Has limited hardware support

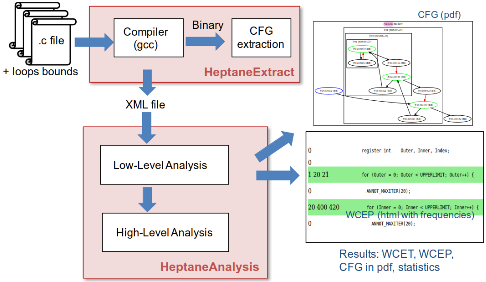
Heptane architecture

- Heptane is divided into two parts



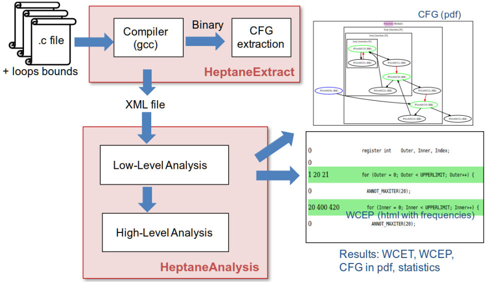
Heptane architecture

- Heptane is divided into two parts
  - HeptaneExtract generates CFG from source-code files written in C or assembly language, which is stored in XML format to be used by HeptaneAnalysis



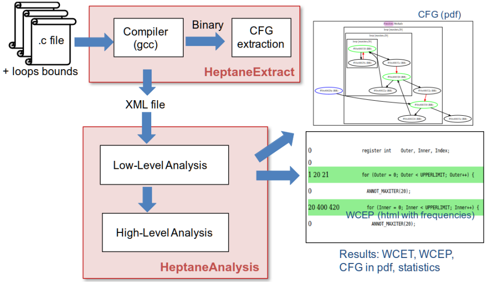
Heptane architecture

- Heptane is divided into two parts
  - HeptaneExtract generates CFG from source-code files written in C or assembly language, which is stored in XML format to be used by HeptaneAnalysis
  - HeptaneAnalysis applies low-level and high-level analyses to produce the WCET estimate



Heptane architecture

- Heptane is divided into two parts
  - HeptaneExtract generates CFG from source-code files written in C or assembly language, which is stored in XML format to be used by HeptaneAnalysis
  - HeptaneAnalysis applies low-level and high-level analyses to produce the WCET estimate



- Supports multiple ILP solvers: IBM CPLEX, LPsolve

Using Heptane

- As input, a program written in C is provided

Using Heptane

- As input, a program written in C is provided
- And an XML configuration file describing the architecture
    - MIPS, RISCV, and ARM ISA
    - Only in-order execution pipeline
    - Different cache hierarchies with various replacement policies and only write-through policy

Using Heptane

- As input, a program written in C is provided
- And an XML configuration file describing the architecture
    - MIPS, RISCV, and ARM ISA
    - Only in-order execution pipeline
    - Different cache hierarchies with various replacement policies and only write-through policy
- Heptane does not include any analysis of maximum numbers of loop iterations, the source code has to be augmented by the user with annotations

Using Heptane

- As input, a program written in C is provided
- And an XML configuration file describing the architecture
  - MIPS, RISCV, and ARM ISA
  - Only in-order execution pipeline
  - Different cache hierarchies with various replacement policies and only write-through policy
- Heptane does not include any analysis of maximum numbers of loop iterations, the source code has to be augmented by the user with annotations
- Heptane does not support handling external functions with annotations either

Table of Contents

simplewhile.c

```c

int main() {
    int i = 0;
    while ( i < 50){

        ++i;
    }
    return 0;
}
```

```c
#include <annot.h>
int main() {
    int i = 0;
    while ( i < 50){
        ANNOT_MAXITER(50);
        ++i;
    }
    return 0;
}
```

Source codes for LLVMTA (left) and Heptane (right)

- For LLVMTA, even though the enable-optimizations flag is set to false, I had to remove -disable-O0-optnone flag for clang from inside the script. Otherwise the loop was removed in compilation and WCET was estimated as 52

simplewhile.c

```c

int main() {
    int i = 0;
    while ( i < 50){

        ++i;
    }
    return 0;
}
```

```c
#include <annot.h>
int main() {
    int i = 0;
    while ( i < 50){
        ANNOT_MAXITER(50);
        ++i;
    }
    return 0;
}
```

Source codes for LLVMTA (left) and Heptane (right)

- For LLVMTA, even though the enable-optimizations flag is set to false, I had to remove -disable-O0-optnone flag for clang from inside the script. Otherwise the loop was removed in compilation and WCET was estimated as 52
- After removing the flag LLVMTA fails to automatically determine the maximum loop bound and asks for annotations

simplewhile.c

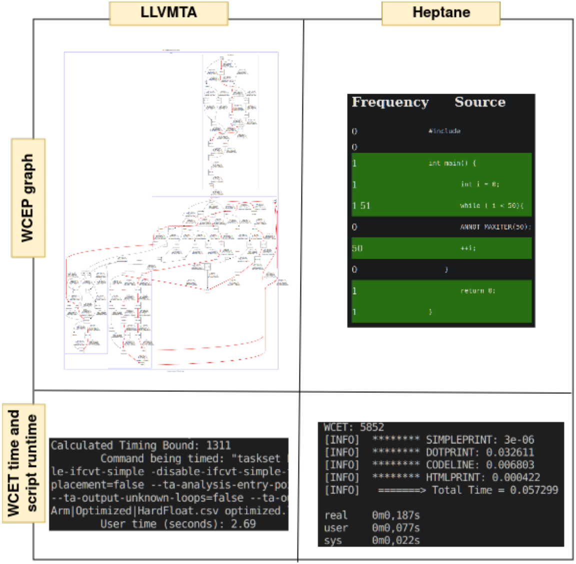
```c

int main() {
    int i = 0;
    while ( i < 50){

        ++i;
    }
    return 0;
}
```

```c
#include <annot.h>
int main() {
    int i = 0;
    while ( i < 50){
        ANNOT_MAXITER(50);
        ++i;
    }
    return 0;
}
```

Source codes for LLVMTA (left) and Heptane (right)

- For LLVMTA, even though the enable-optimizations flag is set to false, I had to remove -disable-O0-optnone flag for clang from inside the script. Otherwise the loop was removed in compilation and WCET was estimated as 52
- After removing the flag LLVMTA fails to automatically determine the maximum loop bound and asks for annotations
- Heptane requires the annotations inside the source code meanwhile LLVMTA looks for it inside a seperate .csv file

Comparison of the results



Comments

- The ISAs are both ARM but the hardwares are not configured equally so its difficult to comment onthe difference in WCET

Comments

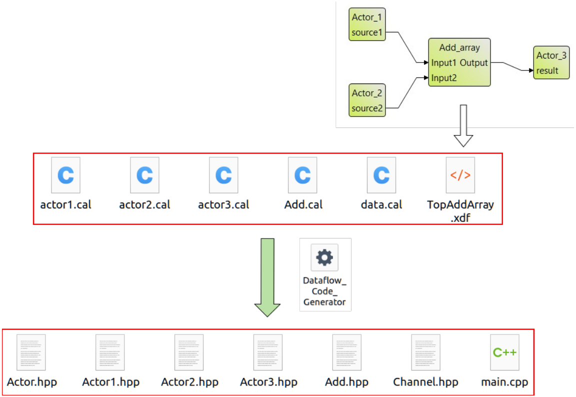
- The ISAs are both ARM but the hardwares are not configured equally so its difficult to comment onthe difference in WCET
- But considering the capabilites of LLVMTA and the logarithmic difference in runtimes, the best decision falls on what is prioritized, performance or accuracy

Comments

- The ISAs are both ARM but the hardwares are not configured equally so its difficult to comment onthe difference in WCET
- But considering the capabilites of LLVMTA and the logarithmic difference in runtimes, the best decision falls on what is prioritized, performance or accuracy
- I continued with LLVMTA because it was easier to be modified to analyze C++ files

Table of Contents

Dataflow Code Generator



- Converts dataflow networks specified in CAL and XDF to C++ code
- Serves as exploration tool for different optimization and mapping strategies

Running LLVMTA for C++

```
208       - clang "${clangopts[@]}" -emit-llvm "$TESTCASE_DIR"/*.c
      208 + clang ${clangopts[@]} -emit-llvm "$TESTCASE_DIR"/*.c*
```

```
3 testcases/dataflowActorsUtils/ClangSpecialOptions.txt
...   @@ -0,0 +1,3 @@
1   + -stdlib=libstdc++
2   + -I/usr/include/c++/11
3   + -I/usr/include/x86_64-linux-gnu/c++/11
```

Two simple tricks

Running LLVMTA for C++

```
208      - clang "${clangopts[@]}" -emit-llvm "$TESTCASE_DIR"/*.c
    208  + clang ${clangopts[@]} -emit-llvm "$TESTCASE_DIR"/*.c*
```

```
3 testcases/dataflowActorsUtils/ClangSpecialOptions.txt
...   @@ -0,0 +1,3 @@
 1  + -stdlib=libstdc++
 2  + -I/usr/include/c++/11
 3  + -I/usr/include/x86_64-linux-gnu/c++/11
```

Two simple tricks

- So not the most elegant solution, and might be the reason for the bugs which will be mentioned

Running LLVMTA for C++

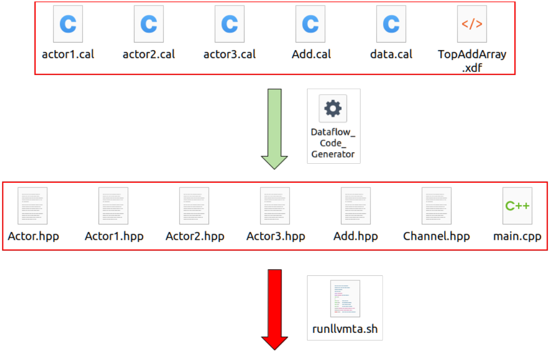
```diff
208       - clang "${clangopts[@]}" -emit-llvm "$TESTCASE_DIR"/*.c
     208  + clang ${clangopts[@]} -emit-llvm "$TESTCASE_DIR"/*.c*
```

```diff
3 testcases/dataflowActorsUtils/ClangSpecialOptions.txt
...   @@ -0,0 +1,3 @@
1     + -stdlib=libstdc++
2     + -I/usr/include/c++/11
3     + -I/usr/include/x86_64-linux-gnu/c++/11
```

Two simple tricks

- So not the most elegant solution, and might be the reason for the bugs which will be mentioned
- Correct cross compiler libraries are needed for sure

Unsuccessful attempt




```
llvmta: /workspaces/llvmta/lib/LLVMPasses/StaticAddressProvider.cpp:173: bool TimingAnalysisPass::StaticAdd
 Assertion `0 && "We have unhandled pseudo instructions"' failed.
PLEASE submit a bug report to https://github.com/llvm/llvm-project/issues/ and include the crash backtrace.
```

The stack trace points to schedule function in main, which is the actual function we are interested in WCET analysis of

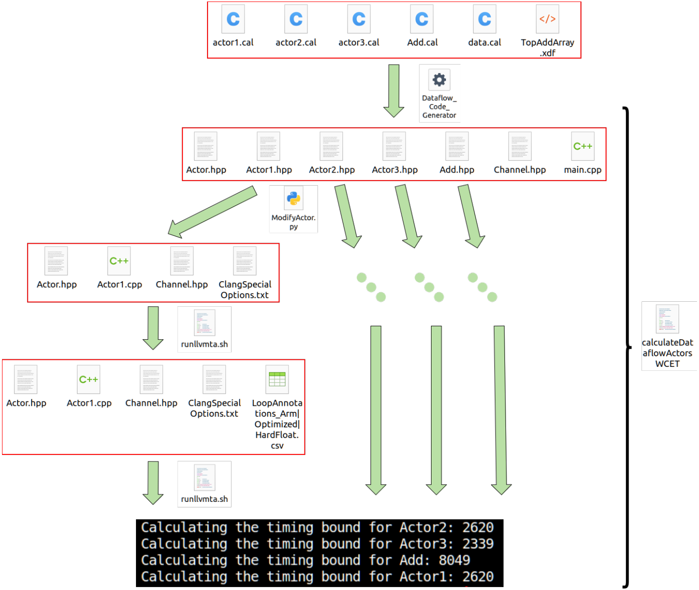
Modifying actor files

```
    void sendData$(void) {
        unsigned char Out;
        Out = SRC1[i];
        i = i+1;
        source1->write(t: Out);
    }
public:
    Actor1(std::string _n, Data_Channel<char>* _source1) {
        actor$name = _n;
        source1 = _source1;
    };
    void schedule(void) {
#ifdef PRINT_FIRINGS
        unsigned firings = 0;
#endif
        for(;;) {
            if (true) {
```

```
    void sendData$(void) {
        unsigned char Out;
        Out = SRC1[i];
        i = i+1;
        source1->write(t: Out);
    }
    int main(int argc, char* argv[]) {
#ifdef PRINT_FIRINGS
        unsigned firings = 0;
#endif
        for(;;) {
            if (true) {
```

Modify all actor files so that we can do LLVMTA run on schedule functions seperately

Final structure




```
Calculating the timing bound for Actor2: 2620
Calculating the timing bound for Actor3: 2339
Calculating the timing bound for Add: 8049
Calculating the timing bound for Actor1: 2620
```

Final remarks

- This workflow throws errors sometimes with more comples networks, at the loop bounds extraction phace of LLVM

Final remarks

- This workflow throws errors sometimes with more comples networks, at the loop bounds extraction phace of LLVM
- A better solution is to create C code from CAL sources and run the WCET tools

References

[1] P. Puschner, R. Kirner, and B. Huber. *Worst-Case Execution-Time Analysis – WCET Analysis*. https://ti.tuwien.ac.at/cps/teaching/courses/real-time-systems/slides/rts06_wcet_analysis-1.pdf. [Online; accessed 09-April-2024].

[2] Björn Franke. *Embedded Systems Lecture 11: Worst-Case Execution Time*. https://www.inf.ed.ac.uk/teaching/courses/es/PDFs/lecture_11.pdf. [Online; accessed 09-April-2024].

[3] Reinhard Wilhelm et al. "The worst-case execution-time problem—overview of methods and survey of tools". In: *ACM Transactions on Embedded Computing Systems (TECS)* 7.3 (2008), pp. 1–53.

[4] Hardik Shah et al. "Measurement based wcet analysis for multi-core architectures". In: *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*. 2014, pp. 257–266.

[5] Sebastian Hahn et al. "LLVMTA: an LLVM-based WCET analysis tool". In: *20th International Workshop on Worst-Case Execution Time Analysis (WCET 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[6] Damien Hardy, Benjamin Rouxel, and Isabelle Puaut. "The heptane static worst-case execution time estimation tool". In: *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. 2017.