COOPERATION

T-CREST
TIME-PREDICTABLE MULTI-CORE ARCHITECTURE FOR EMBEDDED SYSTEMS

Project Number 288008

D 5.6 Full Compiler Version

Version 1.0
28 March 2014
Final

Public Distribution

Vienna University of Technology

Project Partners: **AbsInt Angewandte Informatik**, **Eindhoven University of Technology**, **GMVIS Skysoft**, **Intecs**, **Technical University of Denmark**, **The Open Group**, **University of York**, **Vienna University of Technology**

Project Partner Contact Information

AbsInt Angewandte Informatik Christian Ferdinand Science Park 1 66123 Saarbrücken, Germany Tel: +49 681 383600 Fax: +49 681 3836020 E-mail: ferdinand@absint.com	**Eindhoven University of Technology** Kees Goossens Potentiaal PT 9.34 Den Dolech 2 5612 AZ Eindhoven, The Netherlands E-mail: k.g.w.goossens@tue.nl
GMVIS Skysoft José Neves Av. D. João II, Torre Fernão de Magalhães, 7 1998-025 Lisbon, Portugal Tel: +351 21 382 9366 E-mail: jose.neves@gmv.com	**Intecs** Silvia Mazzini Via Forti trav. A5 Ospedaletto 56121 Pisa, Italy Tel: +39 050 965 7513 E-mail: silvia.mazzini@intecs.it
Technical University of Denmark Martin Schoeberl Richard Petersens Plads 2800 Lyngby, Denmark Tel: +45 45 25 37 43 Fax: +45 45 93 00 74 E-mail: masca@imm.dtu.dk	**The Open Group** Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 Fax: +32 2 675 7721 E-mail: s.hansen@opengroup.org
University of York Neil Audsley Deramore Lane York YO10 5GH, United Kingdom Tel: +44 1904 325 500 E-mail: Neil.Audsley@cs.york.ac.uk	**Vienna University of Technology** Peter Puschner Treitlstrasse 3 1040 Vienna, Austria Tel: +43 1 58801 18227 Fax: +43 1 58801 918227 E-mail: peter@vmars.tuwien.ac.at

Contents

Document Control

Version	Status	Date
0.1	First outline.	25 March 2014
1.0	Final version.	28 March 2014

Executive Summary

This document describes the deliverable *D5.6 Full Compiler Version* of work package 5 of the T-CREST project, due 30 months after project start as stated in the Description of Work. The deliverable comprises the second version of the compiler framework, providing time-predictable code generation and WCET-oriented optimisations by taking advantage of the architectural features of the Patmos architecture. The tool chain contains support for compiling and debugging large applications and operating systems such as RTEMS.

1 Introduction

The second compiler version is the culmination of 2.5 years effort to address various aspects of compilation for the time-predictable Patmos processor. These efforts began with the adaptation of an existing compiler framework to the Patmos ISA, which resulted in the initial compiler version (D5.2). Since then, the compiler has been extended subsequently to target the special features towards generation of time predictable code (D5.3), and integration with the timing analysis tool and optimisations aiming at a reduction of the computed WCET bound (D5.4). The development has been complemented by the elaboration of coding policies for time-predictability (D5.5). This deliverable (D5.6) comprises the full compiler tool chain that contains all previously developed code generation and optimisation features and provides the basis for the evaluation of the T-CREST platform. As such, this supplementary report only gives an overview of the compilation tool chain, presents latest additions and contains pointers to further documentation and resources.

This document is structured as follows. Section 2 gives a short introduction to the tool chain and describes new capabilities, and new features that aid development and debugging from a user perspective. Section 3 gives a quick guide to getting started and gives references to more detailed sources of information, which go beyond what is presented in this brief supplement.

2 The Patmos Compiler

Figure 1 gives an overview of the compiler tool chain. The Patmos compiler is based on the LLVM compiler construction framework. Since the initial version of the compiler (D5.2), the compilation process starts with the translation of each C source file and libraries to the LLVM intermediate language (*bitcode*) by the C frontend `clang`. At this level, the user application code and static standard and support libraries are linked by the `llvm-link` tool. An advantage of linking on bitcode level is that subsequent analysis and optimisation passes, and the code generation backend have a complete view of the whole program. The `opt` optimiser performs generic, target independent optimisations, such as common sub-expression elimination, constant propagation, etc.

The `llc` tool constitutes the backend, translating LLVM bitcode into machine code for the Patmos ISA, and addressing the target-specific features for time predictability. The backend produces a relocatable ELF binary containing symbolic address information, which is processed by `gold`[1], defining the final data and memory layout, and resolving symbol relocations.

In addition to the machine code, the backend exports supplementary information for WCET analysis and optimisation purposes in form of a *Patmos Metainfo File*. This information contains, among others, flow information (in form of loop bounds provided by symbolic analysis on bitcode level), structural information (targets of indirect branches), and information on memory accesses (memory areas accessed by individual load/store instructions). The `platin` toolkit enhances (e.g., by a hardware model), processes and translates this information to the input format for annotations of the timing analysis tool `aiT`, that uses this information in addition to the ELF binary to compute tight WCET bounds.

[1]gold is part of the GNU binutils, see `http://sourceware.org/binutils/`

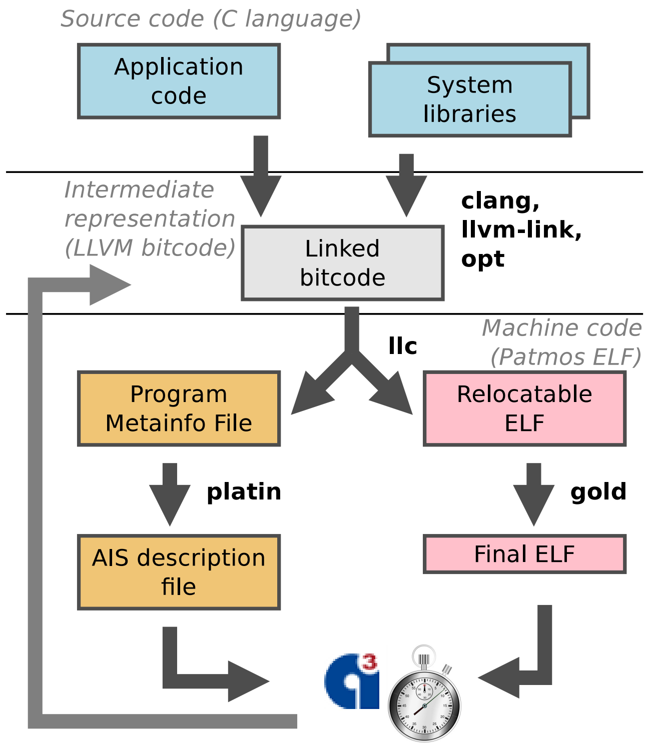

Figure 1: Compiler Tool Chain Overview

2.1 platin – The Portable LLVM Annotation and Timing Toolkit

Besides translation of meta-information from the internal PML to the aiT annotation format AIS, the `platin` toolkit provides other functionality. Below we present a list of the most useful tools.

pml2ais
: Translates information of a PML file relevant to timing analysis to the AIS annotation format.

extract-symbols
: The compiler exports program information at a stage where the final memory layout is not yet defined. This tool reads the final executable and enhances the PML file with information on the final addresses of instructions and data.

analyze-trace
: Based on the structural information of a program in the PML file, the trace analysis tool is capable of extracting flow fact hypotheses based on a simulation run. These are context-sensitive and include, e.g. observed loop bounds and function call targets.

transform
: Transforms flow facts from bitcode to machine code level or simplifies a set of flow facts.

tool-config
: Given a hardware model (in PML format), this tool outputs consistent hardware configuration options/parameters for use during compilation, simulation and WCET analysis.

pml
: Provides validation, inspection and merge facilities for PML files.

visualize
: Visualises structural information of the program in the different program representations.

wcet
: A driver that starts WCET analysis from the command line.

In addition to the platin tools, another command-line utility, `patmos-clang-wcet`, is provided. This tool invokes the compiler (`patmos-clang`), timing analysis, and the compiler a second time (with intermediate calls to platin tools as necessary) for WCET-guided optimisations based on timing-analysis feedback.

2.2 Latest Additions

Since M24, There have been following notable improvements, capabilities and features added to the compiler tool chain:

ISA Revision
: The Patmos ISA has been extended by interrupt support and non-delayed branches. Both the compiler tool chain and the simulator `pasim` have been adapted to the new ISA.

RTEMS Support
: The compiler is capable of compiling the port of the RTEMS operating system as provided by project partner GMV. Simple test applications have been successfully compiled and executed in the simulator.

Debugging
: In order to facilitate debugging, the compiler can emit DWARF4[2] debug information. This is particularly useful for source code annotations of flow constraints (e.g. loop bounds) that can be extracted and utilised by the timing analysis tool `aiT`. Moreover, the `patmos-dwarfdump` utility script can print source code location information (file, line) for binary disassemblies or simulation traces.

Release Builds
: The compiler can be built in release mode, enabling much faster compile times. Though, some information otherwise only available in debug mode (esp. symbolic names for basic blocks) is kept in order to support the export of PML files.

Additional Regression Testing
: The `buildbot` setup described in D5.2 contains now over 1200 programs from the MiBench and Mediabench benchmark suites, the Mälardalen WCET benchmark suite, The PapaBench and Debie benchmarks, as well as the GCC `c_torture` compiler test suite, which are compiled and tested automatically nightly. The tests check for regressions using both the simulator `pasim` and the emulator `patmos-emulator` that is generated from the hardware implementation, and also check for issues in `platin`.

[2] `http://dwarfstd.org/doc/DWARF4.pdf`

3 Installation and Usage

The tool chain can be built on Linux and Mac OSX. To build the tool chain, certain packages are required. A list of required software packages can be found in the `README.patmos` file in the patmos-llvm repository. After installation of the required packages, choose a directory to contain the tool chain sources (e.g. `$HOME/tcrest`).

Change into this directory and clone the patmos-misc repository:

```
$  git clone https://github.com/t-crest/patmos-misc.git misc
```

Optionally, customise the build and installation process by copying the `build.cfg.dist` to `build.cfg` and edit the latter accordingly:

```
$ cd misc
$ cp build.cfg.dist build.cfg
$ vim build.cfg
```

You might want to adapt the root directory, the installation directory is a subdirectory thereof:

```
...
# Root directory for all repositories
ROOT_DIR=$HOME/tcrest
...
# Installation directory prefix
INSTALL_DIR="$ROOT_DIR/local"
```

After saving your changes, you can start the build and installation process by typing

```
$ ./build.sh
```

During the build process, the repositories are cloned as required.

To simplify matters, we recommend to add the directory `$INSTALL_DIR/bin` to your `PATH`.

Once the build process is finished, you can use the `patmos-clang` compiler to compile and link applications for Patmos.

To build a first Hello-World example, put the following C code into a file called `hello.c`.

```c
#include <stdio.h>

int main(int argc, char** argv) {

  printf("Hello␣World!\n");

  return 0;
}
```

To build the example, use

```
$ patmos-clang -o hello.elf hello.c
```

The application can then be executed using the simulator.

```
$ pasim hello.elf
```

The RTEMS port for Patmos can be checked out and built using

```
$ ./build.sh rtems
```

After RTEMS has been compiled and installed, the build script will print out information on how to build example programs for RTEMS.

The full Patmos tool chain released together with this deliverable has been tagged as `release_m31` in all tool chain repositories. The source code of all tool chain components can be downloaded as ZIP packages from `https://github.com/t-crest/`.

Starting with the latest release, we also maintain stable branches of the tool chain. They are branched off stable releases and contain bugfixes but no new features or major changes in order to avoid disruptions during the evaluation phase. At the time of writing, the current stable branch is `stable_m31`.

3.1 Resources

Further information can be found in following places:

The Patmos Handbook
: The Patmos handbook contains up-to-date descriptions of the Patmos ISA, the application binary interface used by the compiler for function calls and stack frame management, as well as a more detailed build process description. It also contains additional information about the tool chain, including commonly used options, library functions to access Patmos-specific hardware features such as the real-time clock and the exception unit and how to use inline assembly or store data on the scratchpad.
`http://patmos.compute.dtu.dk/patmos_handbook.pdf`

README.patmos Files
: Each relevant repository contains a file named "`README.patmos`" in the top-level directory, which contains further Patmos-specific information and usage instructions. The file in the patmos-llvm repository contains requirements and instructions for an installation of the tool chain:
`https://github.com/t-crest/patmos-llvm/blob/master/README.patmos`

4 Requirements

For the sake of completeness, we list the requirements from Deliverable D1.1 that target the compiler work package (WP5) and explain how they are met by the current version of the tool chain. NON-CORE and FAR requirements are not listed here.

4.1 Industrial Requirements

P-0-505 The platform shall provide means to implement preemption of running threads. These means shall allow an operating system to suspend a running thread immediately and make the related CPU available to another thread.

The compiler supports inline assembly, which can be used to implement storing and restoring threads. Further support will depend on the details of the implementation of preemption in the Patmos architecture, developed in the scope of interrupt virtualisation by our project partners (Task 2.6). There is no specific task devoted to the integration of preemption for TUV. Though, we adapted the ISA to allow storing and restoring the contents of the stack cache to and from the external memory. We adapted the compiler and the C library to support compilation of the RTEMS operating system, which features context switching and POSIX threads.

P-0-506 The platform shall provide means to implement priority-preemptive scheduling (CPU-local, no migration).

The compiler supports inline assembly, which can be used to implement storing and restoring threads. Further support will depend on the details of the implementation of preemption in the Patmos architecture, developed in the scope of interrupt virtualisation by our project partners (Task 2.6). There is no specific task devoted to the integration of preemption for TUV. Though, we adapted the ISA to allow storing and restoring the contents of the stack cache to and from the external memory. We adapted the compiler and the C library to support compilation of the RTEMS operating system, which features context switching and POSIX threads.

C-0-513 The compiler shall provide means for different optimisation strategies that can be selected by the user, *e.g.*: instruction re-ordering, inlining, data flow optimisation, loop optimisation.

In the LLVM framework, optimisations are implemented as transformation passes. The LLVM framework provides options to individually enable each transformation pass, as well as options to select common optimisation levels which enable sets of transformation passes.

C-0-514 The compiler shall provide a front-end for C.

The `clang` compiler provides a front-end for C. The compiler has been adapted to provide support for language features such as variadic arguments and floating point operations on Patmos.

C-0-515 The compile chain shall provide a tool to define the memory layout.

The tool chain uses `gold` to link and relocate the executable. The `gold` tool supports linker scripts, which can be used to define the memory layout.

S-0-519 The platform shall contain language support libraries for the chosen language.

The `newlib` library has been adopted for the Patmos platform, which provides a standard ANSI C library.

A-0-521 The analysis tool shall allow defining assumptions, under which a lower bound can be found, *i.e.*a bound that is smaller than the strict upper bound, but still guaranteed to be $>= WCET$ as long as the assumptions are true (*e.g.*instructions in one path or data used in that path fit into the cache).

We adapted the `LLVM` compiler framework to automatically emit flow facts and value facts that are generated by the internal analyses performed by the compiler. The tool chain also supports generation of flow facts from execution traces. The compiler is able to emit debug information which is used by the aiT WCET analysis tool to retrieve source code level flow annotations. Flow annotations at binary level can be used to add additional flow constraints to the WCET analysis.

S-0-522 Platform and tool chain shall provide means that significantly reduce execution time (*e.g.*: cache, scratchpad, instruction reordering).

The LLVM framework provides several standard optimisations targeting execution time, such as inlining or loop unrolling. The data scratchpad memory can be accessed with dedicated macros, which allow the programmer to manually utilize this hardware feature. Instruction reordering is performed statically at compile-time to reduce the number of stall cycles in the processor. The stack cache provides a fast local memory to reduce the pressure on the data cache and to obtain a better WCET bound. The compiler features WCET analysis driven optimisations and optimisations utilising the Patmos hardware features that automatically further reduce the WCET bound.

P-0-528 The tool chain shall provide a scratchpad control interface (*e.g.*: annotations) that allows managing data in the scratchpad at design time.

The SPM API has been integrated into the tool chain, which contains both low-level and high-level functions that allow copying data between SPM and external memory and to use the SPM as buffer for predictable data processing, respectively. Accessing data items on the SPM is possible with dedicated macros that use the address space attribute, which is translated to memory access instructions with the proper type in the compiler backend.

C-0-530 The compiler may reorder instructions to optimise high-level code to reduce execution time.

Instructions are statically reordered to make use of delay slots and the second pipeline, and to minimise stalls during memory accesses.

C-0-531 The compiler shall allow for enabling and disabling optimisations (through *e.g.*: annotations or command line switches).

In the LLVM framework optimisations are implemented as transformation passes. The LLVM framework provides options to individually enable each transformation pass, as well as options to select common optimisation levels which enable sets of transformation passes.

C-0-539 The compiler shall provide mechanisms (*e.g.*: annotations) to mark data as cachable or uncachable.

Variables marked with the `_UNCACHED` macro are compiled using the the cache bypass instructions provided by Patmos to access main memory without using the data cache.

S-0-541 There shall be a user manual for the tool chain.

All tool chain source repositories contain a `README.patmos` file, which explains how to build and use the tools provided by the repository. Additional documentation of the tool chain can be found in the Patmos handbook in the patmos repository. Further information about the LLVM compiler can be found in the LLVM user guide.[3]

4.2 Technology Requirements

C-2-013 The compiler shall emit the necessary control instructions for the manual control of the stack cache.

The compiler emits stack control instructions to control the stack cache, so that data can be allocated in the stack cache. The compiler employs optimisation to reduce the number of emitted stack control instructions.

C-4-017 The compiler shall be able to generate the different variants of load and store instructions according to their storage type used to hold the variable being accessed.

The compiler backend supports all variants of load and store instructions that are currently defined by the Patmos ISA at the time of writing. Support for annotations to select the memory type for memory accesses is provided by dedicated macros.

C-4-018 The storage type may be implemented by compiler-pragmas.

Support for annotations to select the memory type for memory accesses is provided by dedicated macros.

C-5-027 The compiler shall be able to compile C code.

The `clang` compiler provides a front-end for C. The compiler has been adapted to provide support for language features such as variadic arguments and floating point operations on Patmos.

C-5-028 The compiler shall be able to generate code that uses the special hardware features provided by Patmos, such as the stack cache and the dual-issue pipeline.

The compiler uses special optimisations to generate code that uses the method cache, the stack cache and the dual-issue pipeline.

C-5-029 The compiler shall be able to generate code that uses only a subset of the hardware features provided by Patmos.

All code generation passes that optimise code for the Patmos architecture, such as stack cache allocation and function splitting for the method cache, provide options to disable the optimisations and thus emit code that does not use the special hardware features of Patmos.

C-5-030 The compiler shall support adding data and control flow information (*i.e.*: flow facts) to the code, *e.g.*: in form of annotations.

[3] `http://llvm.org/docs/`

Our tool chain extracts flow information from the code automatically and provides the information to the WCET analysis. Furthermore, because the compiler emits debug information, the capabilities of `aiT` to process annotations on the source code can be utilised.

C-5-031 The compiler shall provide information about potential targets of indirect function calls and indirect branches to the static analysis tool.

The compiler emits internal information such as targets of indirect jumps for jump tables. The tool chain provides means to transform this information to the input format of the WCET analysis tool.

C-5-032 The compiler shall pass available flow facts to the static analysis tool.

The compiler emits internal information such as targets of indirect jumps for jump tables. Additional value facts and flow facts are emitted by the compiler based on information generated by internal analyses of the compiler. The tool chain provides means to transform this information to the input format of the WCET analysis tool.

5 Conclusion

This report accompanies the release of the full compiler tool chain for Patmos that comprises all previously developed code generation and optimisation features. The compiler has been successfully tested with over 1200 programs from various benchmark and compiler test suites. It is already used by our industry partners, and an initial port of RTEMS for Patmos developed by GMV is already available.

Support for Patmos specific hardware features and WCET optimisations have been described in the previous reports D5.3 and D5.4. In Deliverable D5.5 we presented coding policies that accompany the compiler analyses and optimisations in order to achieve analysable and predictable code. The evaluation of the compiler will be covered in the upcoming Deliverable D5.7 in month 36.