Compiler Improvements for the VERSAT Reconfigurable Processor

Gonçalo da Conceição Reis dos Santos

Supervisor: Prof. José João Henriques Teixeira de Sousa

Member of the Committee: Prof. Paulo Ferreira Godinho Flores

Presentation context

- Objectives
- VERSAT architecture
    - VERSAT data engine
    - VERSAT configuration module
    - VERSAT controller (picoVersat)
- Compilers
    - CGRA compilers
    - Existing VERSAT compiler
    - gcc compiler
    - lcc compiler
    - Other compilers
- Compiler selection
- Architecture
    - Back-end development
    - Data engine incorporation
    - Data engine incorporation - loops
    - The ‘versat()’ function approach
- Project timing

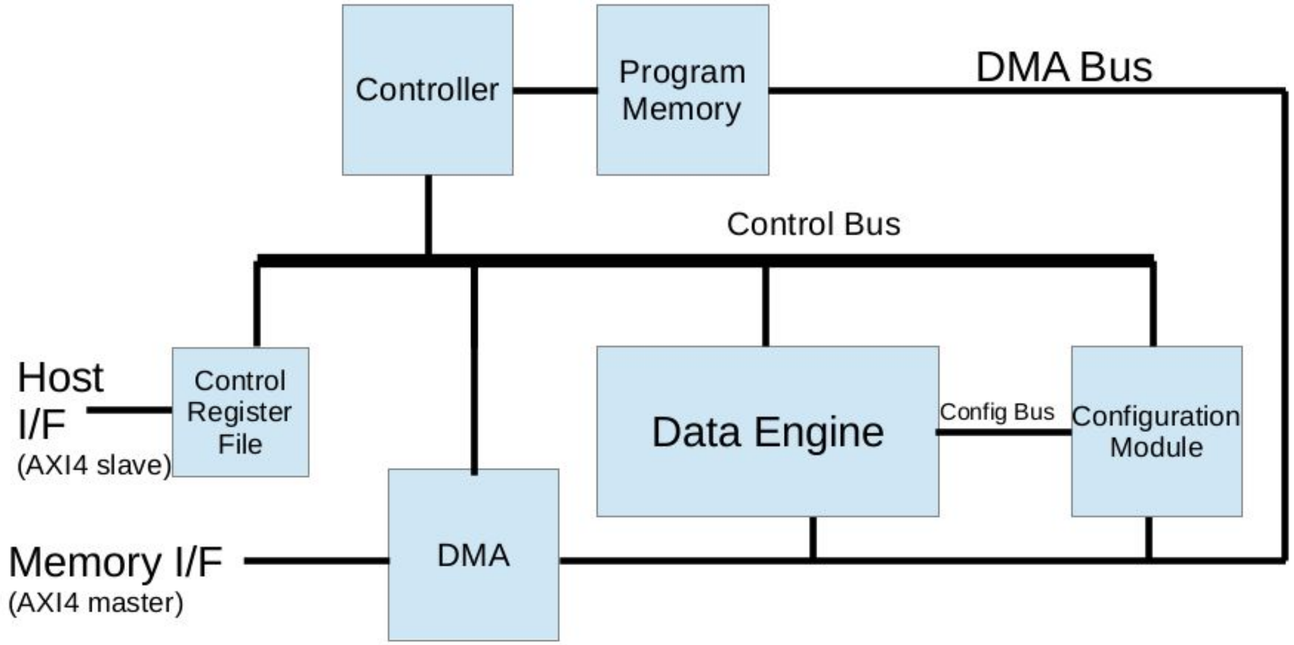
Objectives

Support full high level language (C) algorithms with minimal code changes

Low level control over the CGRA (VERSAT) inner workings in real time

- Support for VERSAT partial reconfiguration
- Support a variable number of functional units
- Support variable declaration
- Support function calls

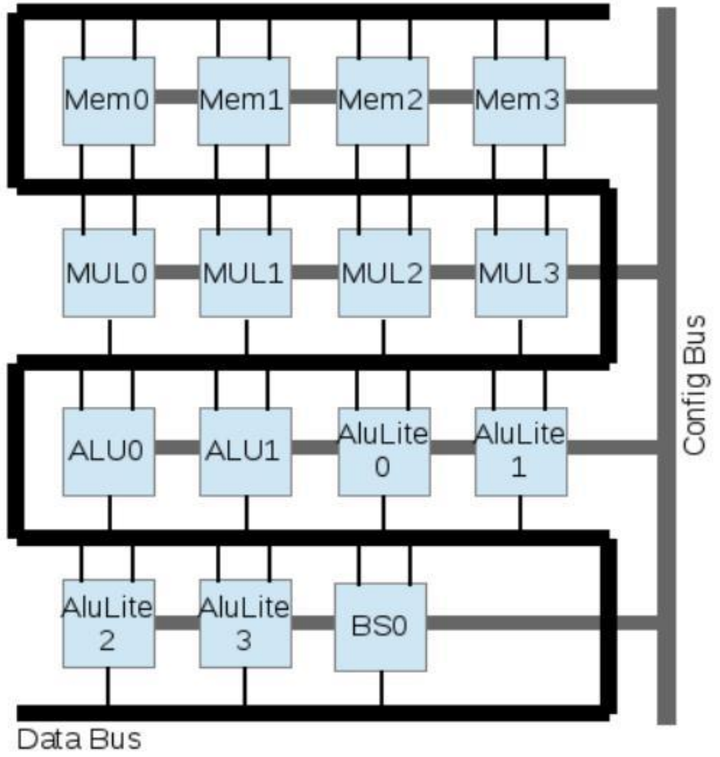
VERSAT architecture



Main components:

- Data engine
- Configuration module
- Controller

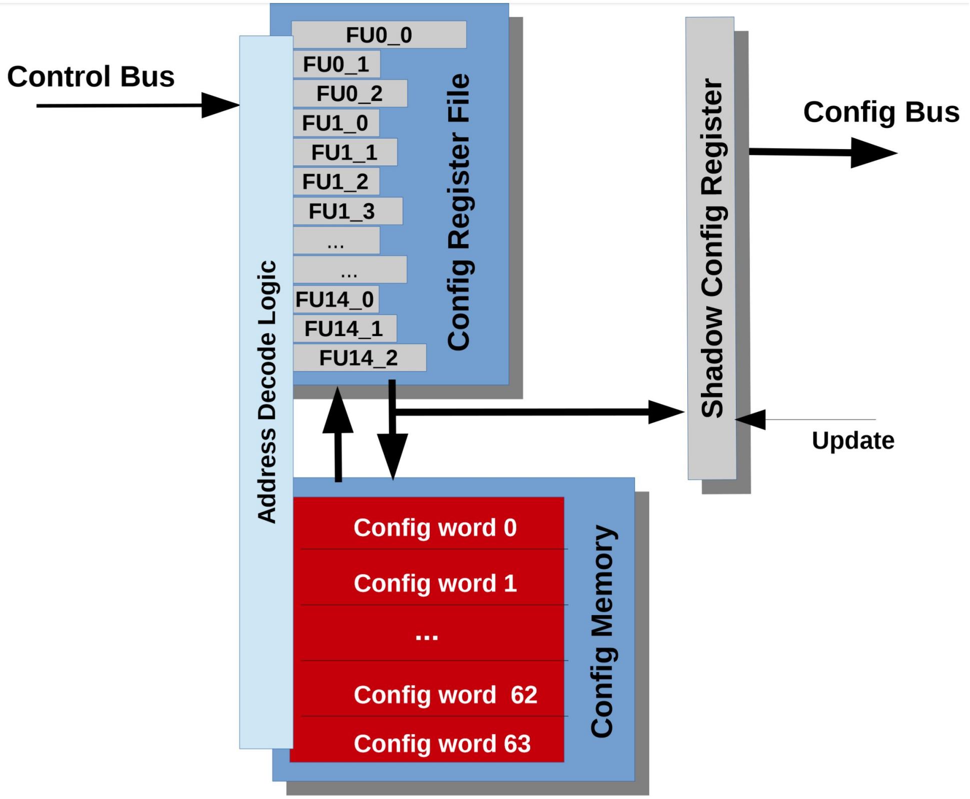
Data Engine



Types of Parallelism Available:

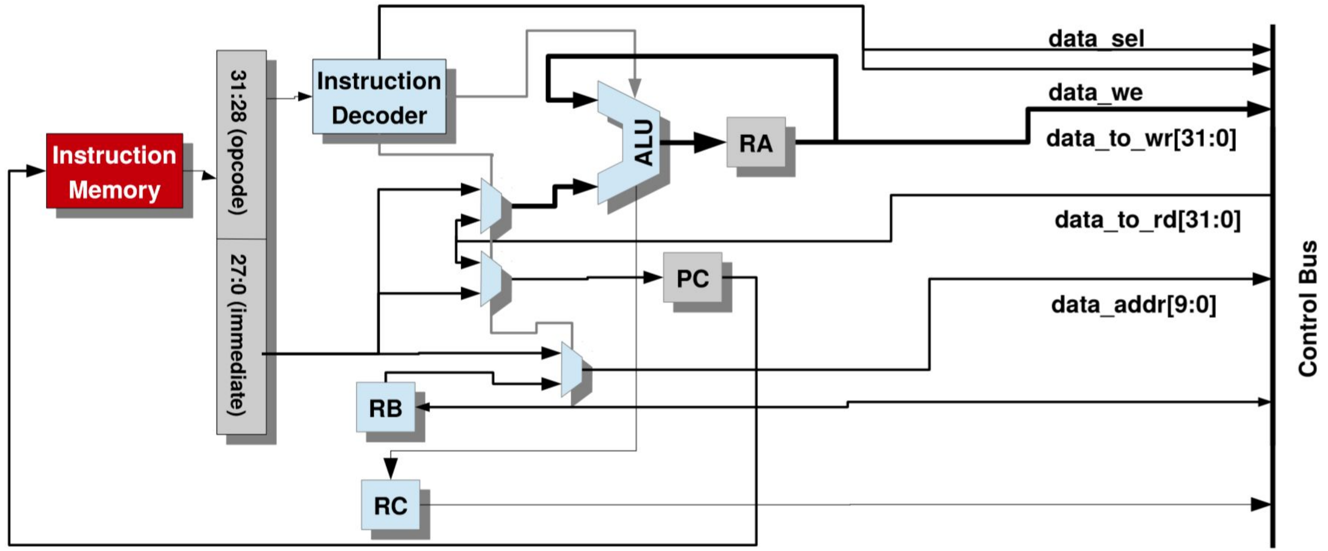
- Data Level Parallelism
- Instruction Level Parallelism
- Thread Level Parallelism

Configuration module



VERSAT is managed by the controller through this module, the controller being the target for the compiler.

picoVersat (Controller)

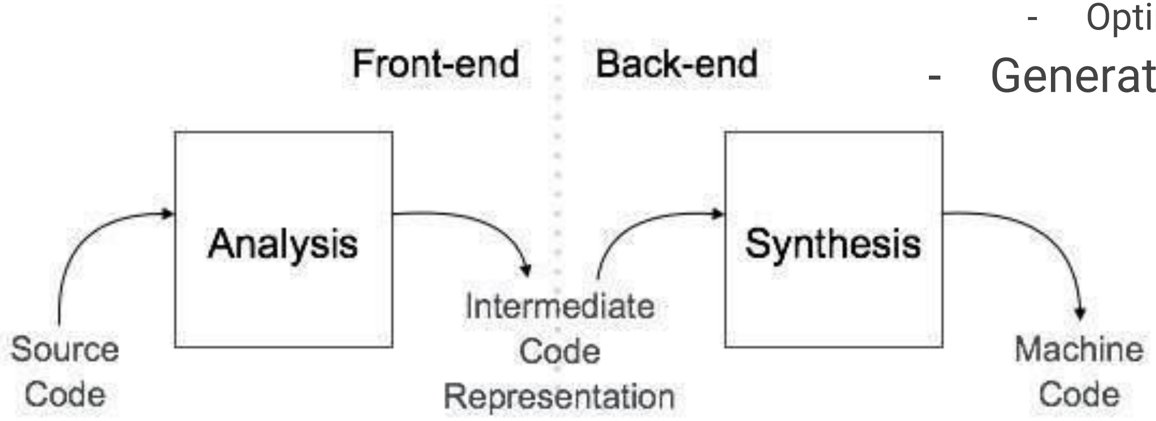


This controller has the ability to work on its own even though it only supports a reduced set of instructions.

Compiler structure

Compiler components:

- Front-end (Analysis):
    - Lexical analysis
    - Syntactic analysis
    - Semantic analysis
- Intermediate code generation
- Back-end (Synthesis):
    - Instruction selection and scheduling
    - Register allocation
    - Optimization and data flow analysis
- Generate machine code



CGRA compilers

Types of CGRAs and their compilers:

- Dynamically reconfigurable
- Statically reconfigurable
- Hybridly reconfigurable

Main problems with current compilers:

- Are not adapted to the specific architecture
- Can only work off of intermediary code

There is possibility of using compiler frameworks like IMPACT, which is able to parse C source code into the intermediate representation (IR).

However, because of the IMPACT front-end, most algorithms don’t produce the desired results given only the IR.

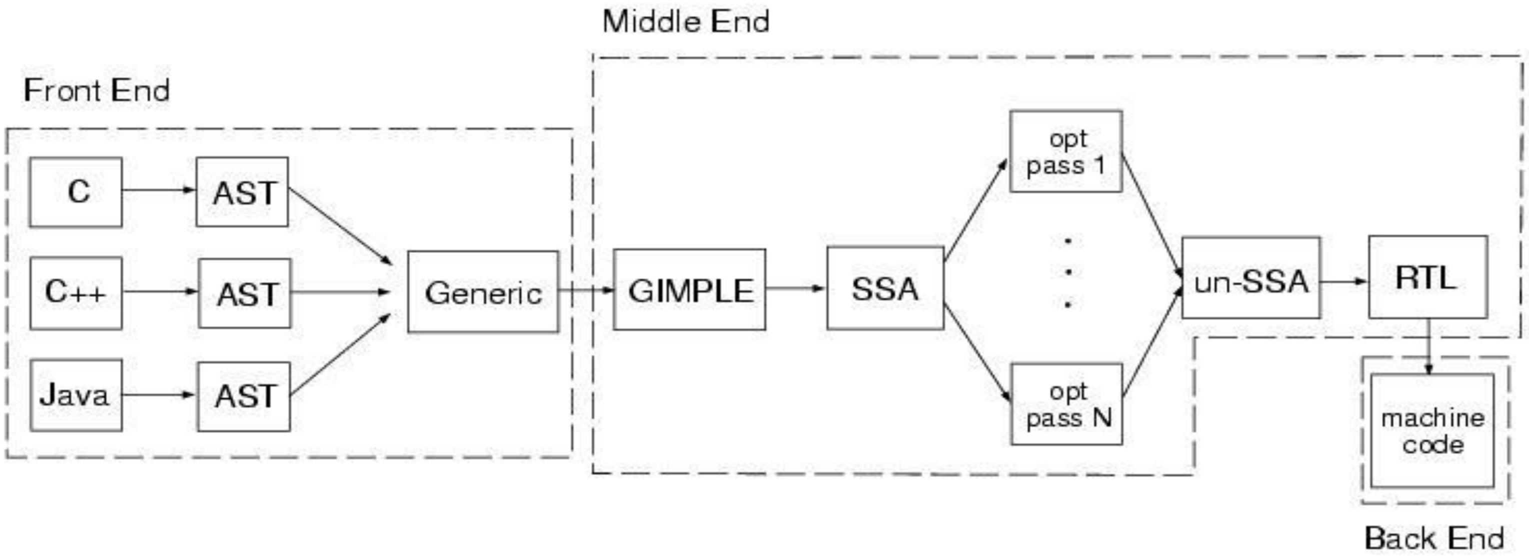
Existing VERSAT compiler

The existing compiler was made from scratch and was based around object oriented programming languages like *JAVA* and *C++*.

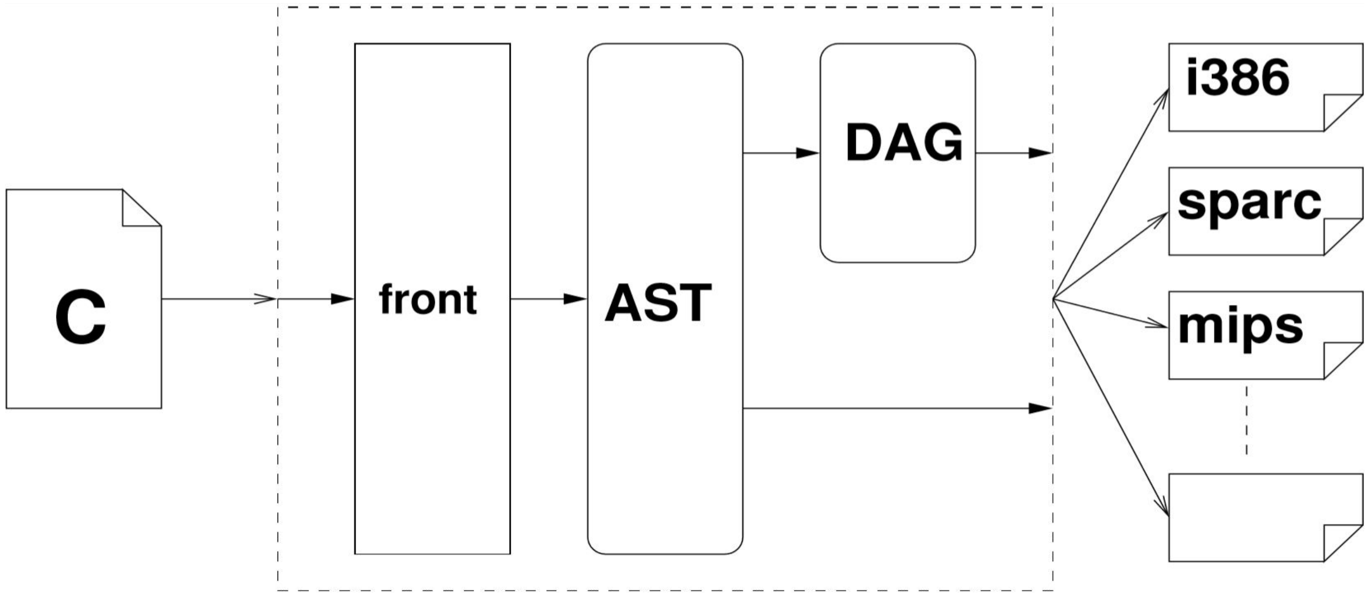
It claims to be a *C++* dialect, it provides no classes, not even C structures, variables or declarations, and no object-oriented capabilities like encapsulation or polymorphism.

Since a true object oriented approach is far too complex for the architecture of a CGRA, the result ended up looking like an object oriented language but not working like one.

gcc compiler



lcc compiler



Other Compilers

- B compiler language
- llvm
- tcc
- Portable C compiler
- Amsterdam compiler kit
- Small device C compiler

Compiler selection

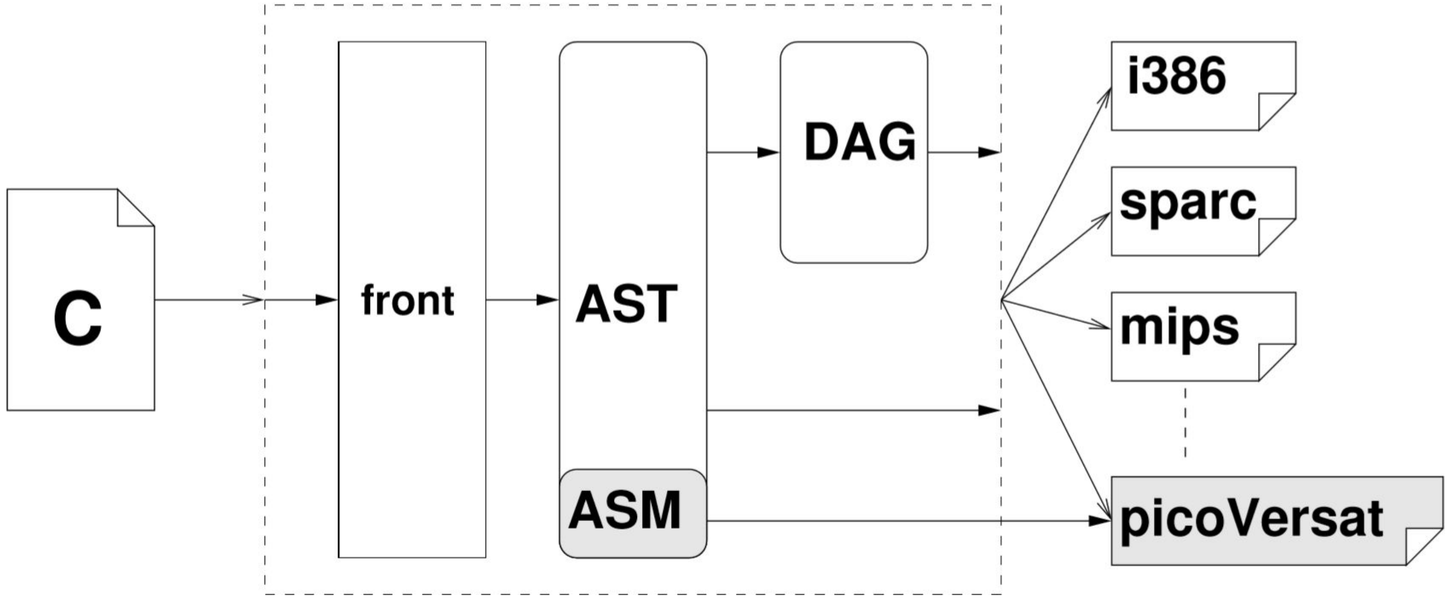
gcc is too complex for the VERSAT’s controller, since it can’t take advantage of most of what’s available in gcc’s.

We want a compiler that is simple enough to incorporate into VERSAT’s architecture while still having all of its functionalities available.

To change the current compiler, which currently does not even take advantage of all of the VERSAT capabilities, would be unwise (it took as a base principle a object oriented language).

The compiler needs to work out of more than just the intermediary code.

Architecture



The VERSAT architecture controller is the target option for the compilation, generating the machine code for it.

Back-end development

At this stage, all back-end low-level operations must be mapped into picoVersat assembly.

The mapping of the C language constructs into such a small instruction set is a complex task.

Some registers must be assigned to compiler management tasks (stack and frame pointers) and others may require temporary allocation for tasks.

A compiler like lcc must also be extended to support the *asm* directive. Such instruction, even if very useful for the tasks required, is not a part of any C language standard.

Data engine incorporation

Even though picoVersat (Controller) can work on it’s own for very simple problems or auxiliary calculations, it’s main purpose is to manage VERSAT.

The controller is meant to manage the data engine in order to take full advantage of the accelerator.

All code written needs to be compatible with the syntax used for the controller.

For example the port connection function would need to called using the same syntax as in C, for instance:

```
connectPortA(alu, alu_port, aluLite, aluLite_port);
```

Data engine incorporation - loops

Loops are the main portions of the execution that can be accelerated by a CGRA.

In order to replace the existing complex *for()* instruction of the previous compiler,

```
for(j=0; j<R6; j++) {
    for(i=0; i<R14; i++) {
        mem0B[R1+j*R13+i] = (mem1A[R1+j*R13+i] * mem2B[1025+j*R2+R10*i]) -
                            (mem0A[R1+j*R13+i] * mem2A[1024+j*R2+R10*i]); } }
```

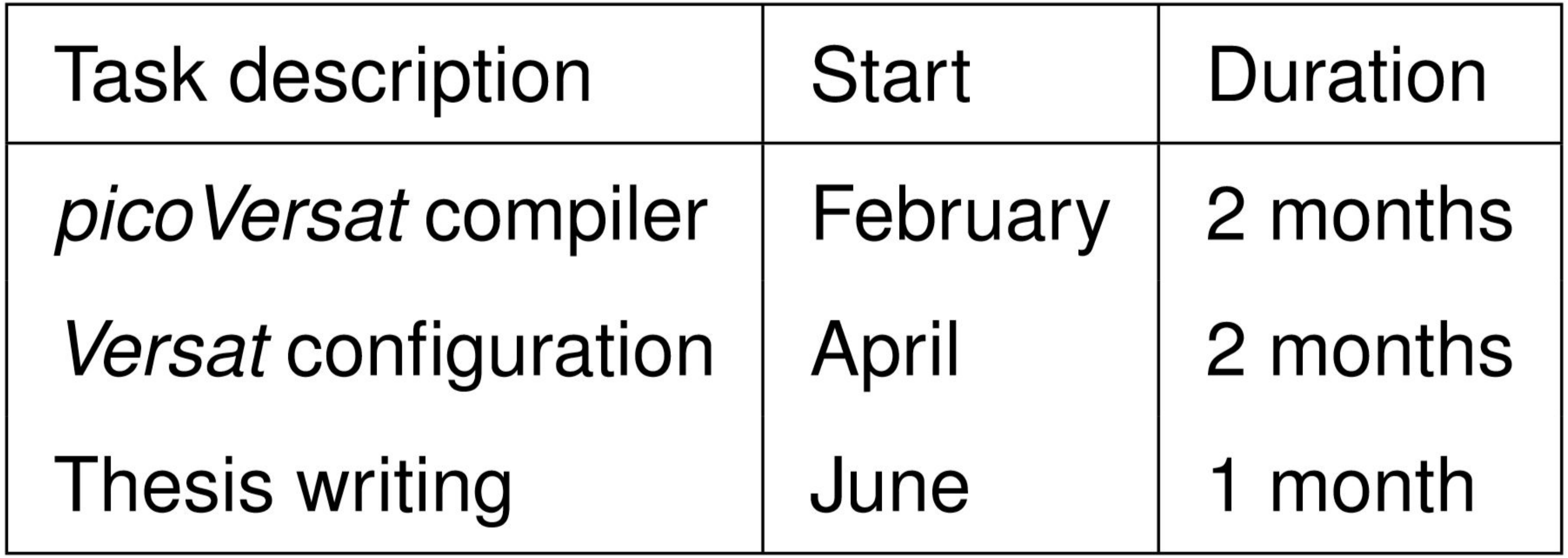
A general purpose C language *for()* instruction representation must be made. For this purpose, a *versat()* function was developed.

The *versat()* function approach

```
int main() {
int *a, *b, *c, *op, *mem;
int N, M;
/* … */
/* insert values into parameters to get desired operations */
versat(N, M, a, b, c, op, mem);
/* calculate values for next configuration */
versat(N, M, a, b, c, op, mem);
/* … */
return 0; }
```

- Include the VERSAT compatibility function as a library inclusion: #include "versat.h"

Project Timing



Task description	Start	Duration
picoVersat compiler	February	2 months
Versat configuration	April	2 months
Thesis writing	June	1 month

Questions?