Float Compiler Reference Manual

Los Alamos National Laboratory
MS B287
Los Alamos, NM 87545

1 Introduction

Trident is an high-level language (HLL) compiler for scientific algorithms written in C that use float and double data types. It produces circuits in reconfigurable logic that exploit the micro-instruction and pipelined parallelism. Trident automatically extracts parallelism and pipelines loop bodies using compiler optimizations and hardware scheduling techniques. The Trident compiler consists of four principal steps shown in Figure 1.

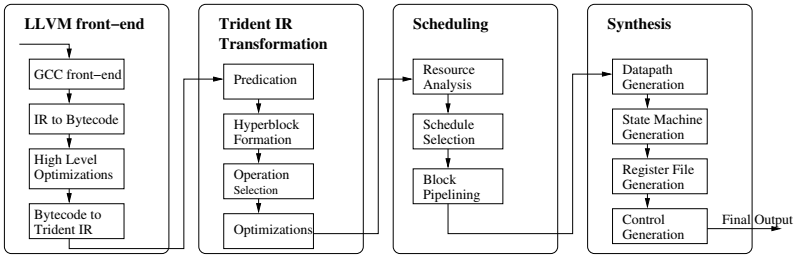

Figure 1. Four principal steps in the Trident Compiler.

The first step in the Trident compilation process is the LLVM C/C++ front-end. The LLVM (Low Level Virtual Machine) compiler framework[1] is used to parse input languages, and produce a low-level platform independent object code. LLVM optimizes the object code and then Trident parses the result into its own intermediate representation (IR). This approach allows Trident to concern itself with hardware compilation and leave the parsing and baseline optimizations in LLVM.

The Trident compiler's second step, IR transformation, accomplishes several important tasks. First, operations are converted into predicated form to allow the creation of hyperblocks. Next, further optimizations are performed to remove any unnecessary operations. Finally, all floating point operations are mapped into a specific hardware library selected by the user.

Floating point data types, with 32- or 64-bit width, present challenges in terms of resource allocation, both on- and off-chip. Floating point modules require significantly more logic blocks

on a reconfigurable chip than small integer operations. The large input width also implies that more memory bandwidth is required for floating point arrays. Scheduling is further complicated by the interaction with resource allocation. The operations will be scheduled with one of four different scheduling algorithms: ASAP, ALAP, Force Directed, and for pipelined loops, Iterative Modulo scheduling.

The final step synthesizes the results of the scheduler into Register Transfer Level (RTL) VHDL. Hierarchy is utilized in the hardware description in order to preserve modularity. The top-level circuit contains subcircuits for each block in the control flow graph input and also a single register file. The register file is shared by all block subcircuits. Each block subcircuit contains a state machine and a datapath subcircuit. The state machine is determined by the initiation interval of the design, and it controls the timing of the block's datapath. The datapath implements the logic needed to represent the flow of data through all of the operations found in the control flow graph. It contains operators, predicate logic, local registers and wires that connect all of the components. All of these elements combine to produce a highly optimized application specific circuit.

Trident provides an open framework for exploration of floating-point libraries and computation in FPGAs. The Trident framework allows for user optimizations to be added at several levels of abstraction. Users can trade-off clock speed and latency through their selection of different floating-point libraries and optimizations. It is also possible for users to import their own floating point libraries. New hardware platforms may be added by defining new interface description files and producing the code to tie the design to the description interface.

2 Source Installation

There are several required elements to use the srouce installation.

- LLVM - obtain version 1.5 from www.llvm.org
- LLVM cfrontend - obtain version 1.5 from www.llvm.org
- ant - obtain 1.6.X from ant.apach.org, 1.7.X is untested.
- antlr.jar - obtain 2.7.3 or later from www.antlr.org
- java-getopt.jar - obtain version 1.11 or later from http://www.urbanophile.com/arenn/hacking/download.html
- Java - obtain from java.sun.com (tested with 1.4.2)
- Python - obtain from www.python.org
- gcc, unix environment, etc.
- Graphviz (dotty and friends) from www.graphviz.org can be useful, but is not required.

First obtain LLVM and compile LLVM. The following works on RHEL 3.0u5 with tcsh, and detailed instructions are available on www.llvm.org. If you prefer bash, use the proper commands for setting variables (e.g., export BLAH=SOME_VALUE and export PATH=$PATH:NEW_PATH_VALUE, etc.).

```
> tar xzf llvm-1.5.tar.gz
> tar xzf cfrontend-1.5.i686-redhat-linux-gnu.tar.gz

#
# must have java
#
> which java
>
> setenv TOP ‘pwd‘

> cd cfrontend/x86/
> ./fixheaders
> cd ../..

> set path = ( $path $TOP/cfrontend/x86/llvm-gcc/bin/ )

> mkdir llvm-obj
> cd llvm-obj

> ../llvm/configure
> make
# wait

> cd ..
> set path = ( $path $TOP/llvm-obj/Debug/bin/ )
```

Now LLVM should be built. This will build the Debug version of LLVM and anything else may not work without changing some of the scripts provided.

Next, unpack the Trident files and execute the following:[1]

```
> tar xzf dist.tgz

# build llv
#
> cd trident/llv-src
> ./mkconf.sh --src=$TOP/llvm
> ./configure --with-llvmsrc=$TOP/llvm/ --with-llvmobj=$TOP/llvm-obj/

> make
> cp Debug/bin/llv ../bin
> strip ../bin/llv
```

[1]This is a continuation of the above and uses the same TOP variable. It could be modified to support a different location than parallel with LLVM.

```
> chmod a+x ../bin/llv

> cd ..

# copy jars to $TRIDENT_TOP/lib
#
> cp somewhere/antlr.jar lib/
> cp somewhere/java-getopt.jar lib/
```

Next, compile Trident. This requires ant, python, java and the jars just added to the above lib directory. Modify your Java classpath to include the jars.

```
> cd trident-src/fp
> ant dist
> cd ../..
> cp trident-src/dist/lib/Trident.jar lib/

> set path = ($path $TOP/trident/bin )

> setenv TRIDENT_TOP $TOP/trident
```

Now, you should have all the tools necessary to run Trident. Test the compiler using the following:

```
cd examples/conditional
tcc -t vhdl conditional_a.c run

less conditional_a.vhd
```

This will generate a VHDL file for the conditional_a.c example. The output of the compiler is extremely verbose and needs to be cleaned up. Serious errors will cause the compiler to exit, the many warnings can usually be safely ignored.

3 Using the Trident Compiler

Assuming that everything worked correctly in the previous section, here we explain some of the options and how to use the Trident compiler.

3.1 Trident

The compiler is made up of three separate parts, which are all integrated in the tcc script. The first part is the LLVM cfront-end, which reads the input C file and generates LLVM bytecode. The next part of the compiler is llv. llv takes the generated LLVM bytecode, optimizes it in LLVM and generates parseable text file, which has the extension .llv. This is still a target independent representation. The .llv file is parsed by the java class fp.Compile, undergoes further optimization and finally is targeted to a particular floating point library. The output result is VHDL.

An example of using tcc:

```
tcc -t vhdl if_a.c run
```

The -t option selects the output type and the next two options if_a.c and run are the input file and the function to be compiled. Trident will only compile a single function from an input file, the function cannot have any parameters and cannot return any results. For example:

```
extern int a,b,c,d;

void run() {
  if (a < b) {
    a = c;
  } else {
    a = d;
  }
}
```

The run function does not have any input parameters. However, variables that are declared as extern can be used as inputs or outputs (or both.) Any variables that are declared as extern will get registers that can be read and written. Other variables may not be able to be seen outside of the datapath.

Options to tcc:

```
Usage: tcc [-h] [--llv=/path/to/llv] [compiler-options] input_file function_name

Version 0.5

View compiler options by using the command: 'tcc -h'
Environment variables LLVMGCCDIR and CLASSPATH must be set correctly
Executables llv and java must be in the user's path
```

Most options to tcc pass to the backend Trident compiler as does the -h options. The backend compiler has the following options:

```
Compile [Options] input_file function_name

Short Options:
-a filename     : Filename of architecture (hardware) description
-b              : Generate Testbench -- experimental
-h              : Get help -- good luck!
-l libname      : Specify library, supporting quixilica, aa_fplib,
-t format       : Target output.  format can be set to "dot" or "vhdl"

Long Options:
--sched=sched_list
        sched_list is comma-separated list which can include
```

```
        asap, alap, or fd, and optionally nomod or mod

        It is case insensitive.  You must choose either asap, alap, or force
        directed.  It is default to run modulo scheduling on all loop blocks,
        but to turn that off, you can add "nomod" within the string.  For
        example:

        --sched=fd,nomod

        You can also, specify "mod", but that is the default.

--sched.options=option_list
        option_list is a comma-separated list which can include
        considerpreds (tell schedule not to ignore predicates)
        dontpack (don't pack multiple less than 1-clock tick instructions within a single cy
        conserve_area (conserve area by only using as many logic units as
                necessary so as to not slow down execution of the design)
        fd_maxtrycnt (tell the compiler how many times to attempt force-directed
                scheduling before giving up)
        ms_maxtrycnt (how many times to attempt modulo scheduling)
        cyclelength (set the length of a clock tick (the default is 1.0))

        The options can be set either by using the long opt, "sched.options" to set several
        once (like this:

        --sched.options=considerpreds,dontpack,conserve_area

        or they can be set individually by saying:

        --sched.options.considerpreds
```

Several options may not be very useful and they also may not be very well tested. **-a** is for specifying a different hardware target. Currently only one hardware target is truly supported – specifying other files here is not well tested. **-b** generates a skeleton VHDL testbench for the top-level cell. It does this by examining the cell, instancing it and inserting some reset statements. Treat this option as experimental – if it works for you, good – if it does not, you were warned. **-l** is for specifying additional libraries. This option is dynamic and supports the floating point libraries it has been told about. aa_fplib is the library I have supplied. **-t** specifies the output format, vhdl or dot. vhdl produces a vhdl netlist and dot is a schematic view of that netlist. Please be aware that hierarchical netlists quickly exceed dotty's (graphviz) ability to render dot files.

The long options are described above. Please note that by default loops are modulo scheduled and non-loop blocks are scheduled using a force-directed algorithm. Other options modify how particulars within the scheduling algorithms.

In addition to Dot circuit representations, Trident currently creates some debug information in Dot format. The *cfile_function*.dot file is the final optimized version of the function before synthesis. The synthesized version is called syn_*cfile*.dot. Other dot files represent the data flow graph of individual hyperblocks.

3.2 Simulation

Trident generates a single VHDL file representing the compiled design. Although, there is just one file, the file contains multiple heirarchical VHDL Design Units and contains the elements shown in Figure 2. To allow for correct compilation, the circuit elements are inserted in the VHDL file from the lowest level of heirarchy (leaves) to the highest level (top). At the top level, all of the inputs and outputs are available as registers as well as *start* and *reset* signals.

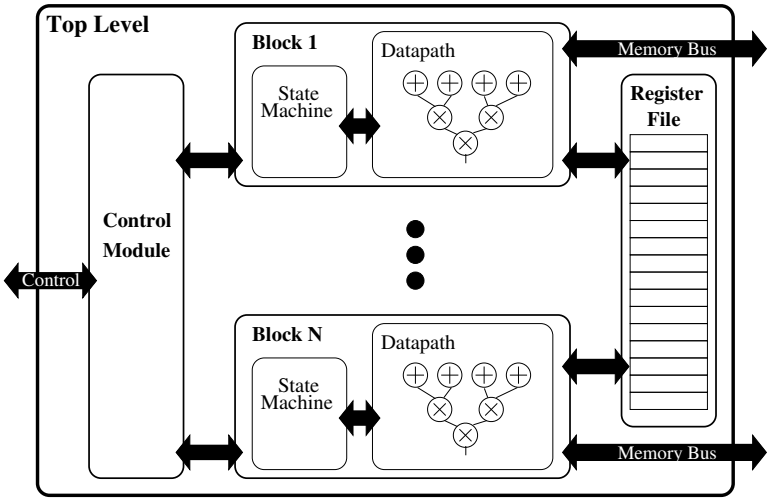

Figure 2. Circuit Elements found in the Trident compiler.

Compilation of Trident generated designs usually requires several support libraries. It is best to compile the libraries first and then compile the design to verify that all circuit objects have been correctly instanced. A floating-point circuit will require a floating-point library as well as the trident support library. The floating-point library should provide its own means of compilation. The trident support library has several operations that are not normally included in most available floating-point libraries. This includes casts and a few other functions(e.g., float to int, double to long, int to float, long to double, float to double, double to float, fpabs, and fpinv).[2]

An example of what compilation may look like:

```
> vlib work
> vmap work work
> vmap fplib /my/path/to/fplib/fplib
> vcom -93 example.vhd tb_example.vhd
```

[2]Casts are ugly operations that are easy to imply when mixing floats with integer types.

The compiler does have the ability to generate a skeleton VHDL testbench for generated designs. This testbench does not actually exercise the circuit, but instances it and produces process that can be used to provide input and test the results. The generated testbench saves the user the effort of writing the testbench from scratch.

The synthesized circuit has a particular model of execution. The testbench must follow this model to ensure proper circuit function. The circuit expects all of the inputs to be available before the start signal is toggled. Reset should be toggled before start and will reset any values written to registers before reset has been toggled. The circuit design does not eliminate the possibility of modifying the registers during operation, however, this may result in incorrect operation.

References

[1] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 Int'l Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, CA, Mar 2004.