Aaa2vhdl: A VHDL backend for cSVM

Phil Tomson
philtomson@gmail.com
February 27, 2005

TASK:

My task has been to create a VHDL backend for the cSVM tool.

BACKROUND:

The cSVM program creates an output file ('.aaa' file) which contains all of the training information for a particular svm (Support Vector Machine). cSVM also comes with some utilities for translating an .aaa file to C or C++ (aaa2c, aaa2cpp). The goal of my project is to create a tranlator to VHDL so that the svm can be realized in hardware (a Xilinx Spartan 3 FPGA in this case) using a synthesis tool.

THE APPROACH:

In order to develop and test the aaa2vhdl, various software tools needed to be developed. I chose to use a combination of Ruby (see Appendix A: Requirements) and C to develop these tools: the aaa2vhdl toolsuite itself is coded in Ruby.

.AAA parser:

The first step was to create the parser for the .aaa files. To do this I examined the current aaa2c and aaa2cpp utililities. The parser class **AAAParser** is defined in the Ruby file: *aaaparser.rb.* Once an aaa file has been parsed into an AAAParser object, all of the various data structures defined in the .aaa files (vectors of floating point numbers, integers, etc.) are available from that object. AAAParser was tested by creating a aaa2cpp program in Ruby using the AAAParser and results were compared with the current aaa2cpp program.

Lookuptable testing:

One of the main issues with implementing the Gaussian kernel SVM in hardware is the exp() function. The following C code illustrates the issue:

```
for(i=0;i<cSVM_ni;i++)
 x_norm[i]=cSVM_cNorm[i][0]*x[i]+cSVM_cNorm[i][1];
  f=0;
  for(i=0;i<cSVM_nsv;i++){
    k=0.;
    for(j=0;j<cSVM_ni;j++){ //foreach entry in the x_norm vector
      k+=(x_norm[j]-cSVM_sv[i][j])*(x_norm[j]-cSVM_sv[i][j]); //norm squared
    }
    f+=cSVM_alpha[i]*exp(-k*cSVM_kRpar/cSVM_ni);
}
```

The exp() function can be realized in one of two ways:

1) as a lookup table

2) using CORDIC

I chose to use a lookup table to implement the *exp(-k*cSVM_kRpar/cSVM_ni)* function. Given this choice, it was then necessary to determine how large the lookup table needed to be and how many bits of precision were needed to ensure satisfactory results. To do this a Ruby class was created to simulate fixed point arithmetic (see the FixedPt class in the file rangeint.rb) and the gaussian kernel was implemented in terms of FixedPt objects. A LookupTable class was created (see: lookuptable.rb) to create the exp lookup tables. It allows for the easy creation of lookup tables with different resolutions, number of entries and bit precision of entries so that each of these options could be tested.

At this point, some method was needed to determine if the lookup table-based Gaussian kernel was giving correct results. There already existed testing files for both the banana and sonar testcases which contained *x* vector inputs and the sign of the expected output. A parser was created for these test files (see the file csvm_test.rb) so that the inputs could be fed into the lookup table-based SVM and the polarity of the output could be compared with the correct value in the test file. After this infrastructure was in place it was possible to run 'simulations' by varying different parameters of the *exp()* lookup table to determine an optimal lookup table size and lower bound.

Experiments determined that the banana testcase could be run with no errors in ~4900 test cases given the following lookup table parameters:

Parameter	***Value***
Number of entries	1024
Bits of precision	19
Min x (lower bound)	-12.5

Note: Min x means the most negative x input to the exp(x) function. Values which are less will be considered to produce an output of 0 from the function.

HDL Generation Issues

Fixed point math

The IEEE.fixed_pkg for fixed point arithmetic would not work with the Xilinx XSE synthesis tool due to a bug in XSE that does not allow negative integers in ranges. In order to work around this I created my own fixed point package (see the file fixed_pt.vhd in the *tests/* subdirectory)

csvm_fsm

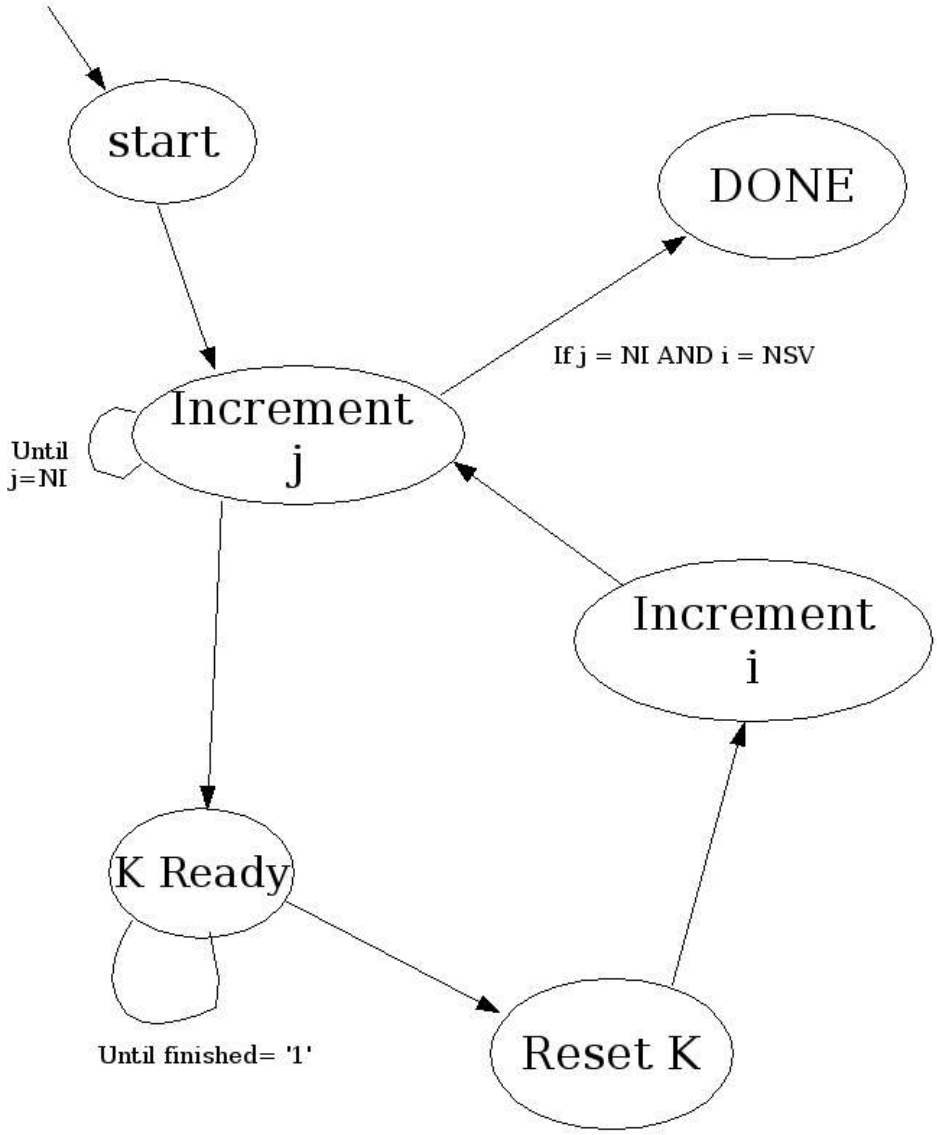

Diagram of FSM which controls the cSVM calculations

The diagram above shows the state machine which controls the cSVM calculations.

The *i* and *j* variables correspond to the i and j counters in the nested for-loops in the C code shown in the lookup table section above. The j counter is incremented until j = ni (number of features) . The variable k accumulates the squared difference between *Xin[i]* and *cSVM_sv[i,j]*. When j=ni the state machine goes to the **K Ready** state indicating that k is ready for use in addressing the exp() function lookup table. The state machine will remain in the K Ready state until the *finished* signal is received from the user of *k*. The next state, **Reset K** produces a signal which resets the **k register.** After this the **i** counter is incremented (State Increment i **)**

Appendix A: Requirements

RUBY:

The aaa2vhdl tool was developed in Ruby 1.8.2, therefore, you will need to have Ruby installed on your machine. Ruby is available on both Windows and Linux platforms. More information on obtaining Ruby can be found here: http://www.ruby-lang.org

Ruby is an object-oriented programming language often classified as a scripting language (similar to Perl or Python). Ruby itself is interpreted so it's speed is not as fast as a compiled language like C or C++. However, it is quite easy to mix C code with Ruby using the RubyInline package available here: http://rubyforge.org/projects/rubyinline/

RubyInline is an optional requirement, however, aaa2vhdl will work without it, it will just run the simulations more slowly (if you're not running any simulations using the aaa2vhdl package this is no problem).

Appendix B: Directory Structure:

cSVM/utils/aaa2vhdl/ - aaa2vhdl Ruby files

tests/ - code and files for testing

docs/ – documentation

ext/ - C extension for lookuptable tests

Appendix C: GHDL

I used the open source GHDL VHDL simulator available here:

http://ghdl.free.fr/

Binaries for Linux are available. To compile the source code for GHDL for use on Windows you will need to install cygwin.