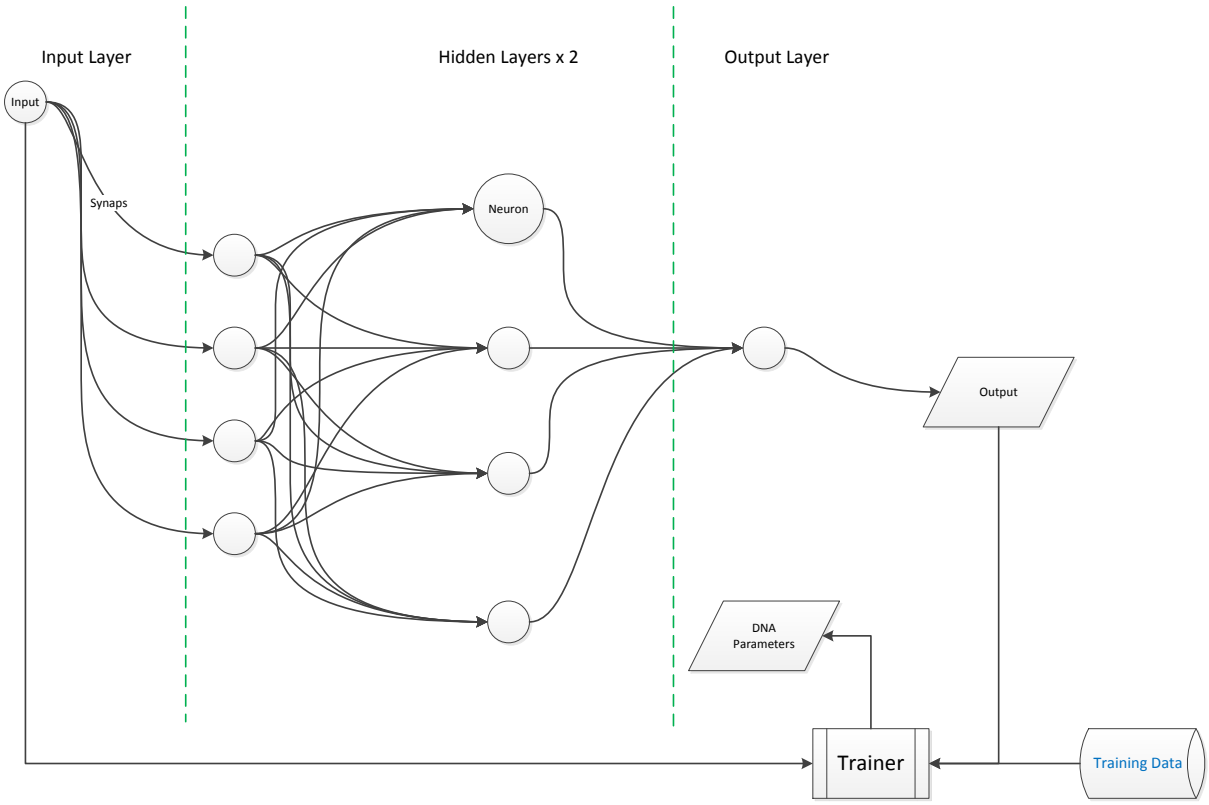
Execution Example

This will illustrate some specific properties of a CorTeX simulation.

Building the network

Let's start with building the network and select a very basic model. To compare with other existing neural network APIs we select a basic [input-4-4-output] network.



The code to realize this network in cortex is…

```
// Create a brain
ctxCortexPtr cortex=new ctxCortex;

// define input output as derived classes of my input and output
ctxCortexInput *input=new MyCortexInput();
ctxCortexOutput *output=new MyCortexOutput();

// Create an output layer that has the right size of output
output->addLayerNeurons(output->getResultSize());

// Add input/output to brain
cortex->addInput(input);

// Add input to brain
cortex->addOutput(output);

// create internal layers. two layers with 4 neurons in each
cortex->addNeuronLayers(2,4);

// Connect all neurons feed forward
cortex->connectNeurons(CTX_NETWORK_TYPE_FEED_FORWARD);
```

The cortex layers are now defined and the connections (synapses) are connected using the forward feed pattern. All neurons between each layer are fully connected to each other.

Generating the code

We will now tell the network to generate an assembly a like program from the network that can be used to evaluate the network. First we compile it…

```
// Compile it and clear all internal data, possibly optimize it
cortex->compile(TRUE,optimize);
```

And then we set some random values in the DNA parameters…

```
// Provide some random start values
cortex->getContext()->randomParameters(1.0f/(neurons));
```

We now have generated an assembly a like program that possibly in the future could be run by a dedicated HW. We can take a look at the “disassembler” of this program to understand the content.

The disassembly looks like…

```
-- Compiled Cortex Program --
Instructions:311
ValueStates:33
Parameters:33
LatencyStates:0
SignalSources:1
InputSignals:1

------------ SubRoutines ---------------
 Threads:1
----------------------------------------

---- Thread:0 ----
 Offset:0
 Length:310
 Pass:0
---- Commands ----
RCL Input (0)               { Recall input register SP: +1 }
MULT Param (0)              { multiply with param register }
TEST Drop Value (0)         { Test drop value register }
STO Value (0)               { Store value register SP: -1 }
RCL Value (0)               { Recall value register SP: +1 }
ADD Param (1)               { add param register }
NFU (7)                     { Neuron function }
STO Value (1)               { Store value register SP: -1 }
RCL Value (1)               { Recall value register SP: +1 }
MULT Param (2)              { multiply with param register }
TEST Drop Value (2)         { Test drop value register }
STO Value (2)               { Store value register SP: -1 }
RCL Value (2)               { Recall value register SP: +1 }
RCL Input (0)               { Recall input register SP: +1 }
MULT Param (3)              { multiply with param register }
TEST Drop Value (3)         { Test drop value register }
STO Value (3)               { Store value register SP: -1 }
RCL Value (3)               { Recall value register SP: +1 }
ADD Param (4)               { add param register }
NFU (7)                     { Neuron function }
STO Value (4)               { Store value register SP: -1 }
RCL Value (4)               { Recall value register SP: +1 }
MULT Param (5)              { multiply with param register }
TEST Drop Value (5)         { Test drop value register }
STO Value (5)               { Store value register SP: -1 }
RCL Value (5)               { Recall value register SP: +1 }
RCL Input (0)               { Recall input register SP: +1 }
MULT Param (6)              { multiply with param register }
TEST Drop Value (6)         { Test drop value register }
STO Value (6)               { Store value register SP: -1 }
RCL Value (6)               { Recall value register SP: +1 }
```

Information about instructions and parameters

There are 33 value states. One for each node and one for each synapse. There is one input register and 33 parameter registers. These correlate to the nodes different internal attributes. Compare with bias and weights in a normal network.

```
ADD Param (7)              { add param register }
NFU (7)                    { Neuron function }
STO Value (7)              { Store value register SP: -1 }
RCL Value (7)              { Recall value register SP: +1 }
MULT Param (8)             { multiply with param register }
TEST Drop Value (8)        { Test drop value register }
STO Value (8)              { Store value register SP: -1 }
RCL Value (8)              { Recall value register SP: +1 }
RCL Input (0)              { Recall input register SP: +1 }
MULT Param (9)             { multiply with param register }
TEST Drop Value (9)        { Test drop value register }
STO Value (9)              { Store value register SP: -1 }
RCL Value (9)              { Recall value register SP: +1 }
ADD Param (10)             { add param register }
NFU (7)                    { Neuron function }
STO Value (10)             { Store value register SP: -1 }
RCL Value (10)             { Recall value register SP: +1 }
MULT Param (11)            { multiply with param register }
TEST Drop Value (11)       { Test drop value register }
STO Value (11)             { Store value register SP: -1 }
RCL Value (11)             { Recall value register SP: +1 }
SUM Stack (4)              { Sum stack values SP: -3 }
ADD Param (12)             { add param register }
NFU (7)                    { Neuron function }
STO Value (12)             { Store value register SP: -1 }
RCL Value (12)             { Recall value register SP: +1 }
MULT Param (13)            { multiply with param register }
TEST Drop Value (13)       { Test drop value register }
STO Value (13)             { Store value register SP: -1 }
RCL Value (13)             { Recall value register SP: +1 }
RCL Value (1)              { Recall value register SP: +1 }
MULT Param (14)            { multiply with param register }
TEST Drop Value (14)       { Test drop value register }
STO Value (14)             { Store value register SP: -1 }
RCL Value (14)             { Recall value register SP: +1 }
RCL Value (4)              { Recall value register SP: +1 }
MULT Param (15)            { multiply with param register }
TEST Drop Value (15)       { Test drop value register }
STO Value (15)             { Store value register SP: -1 }
RCL Value (15)             { Recall value register SP: +1 }
RCL Value (7)              { Recall value register SP: +1 }
MULT Param (16)            { multiply with param register }
TEST Drop Value (16)       { Test drop value register }
STO Value (16)             { Store value register SP: -1 }
RCL Value (16)             { Recall value register SP: +1 }
RCL Value (10)             { Recall value register SP: +1 }
MULT Param (17)            { multiply with param register }
TEST Drop Value (17)       { Test drop value register }
STO Value (17)             { Store value register SP: -1 }
RCL Value (17)             { Recall value register SP: +1 }
SUM Stack (4)              { Sum stack values SP: -3 }
ADD Param (18)             { add param register }
NFU (7)                    { Neuron function }
STO Value (18)             { Store value register SP: -1 }
RCL Value (18)             { Recall value register SP: +1 }
MULT Param (19)            { multiply with param register }
TEST Drop Value (19)       { Test drop value register }
STO Value (19)             { Store value register SP: -1 }
RCL Value (19)             { Recall value register SP: +1 }
RCL Value (1)              { Recall value register SP: +1 }
MULT Param (20)            { multiply with param register }
TEST Drop Value (20)       { Test drop value register }
STO Value (20)             { Store value register SP: -1 }
RCL Value (20)             { Recall value register SP: +1 }
RCL Value (4)              { Recall value register SP: +1 }
MULT Param (21)            { multiply with param register }
TEST Drop Value (21)       { Test drop value register }
STO Value (21)             { Store value register SP: -1 }
RCL Value (21)             { Recall value register SP: +1 }
RCL Value (7)              { Recall value register SP: +1 }
MULT Param (22)            { multiply with param register }
TEST Drop Value (22)       { Test drop value register }
STO Value (22)             { Store value register SP: -1 }
RCL Value (22)             { Recall value register SP: +1 }
RCL Value (10)             { Recall value register SP: +1 }
MULT Param (23)            { multiply with param register }
TEST Drop Value (23)       { Test drop value register }
```

```
STO Value (23)             { Store value register SP: -1 }
RCL Value (23)             { Recall value register SP: +1 }
SUM Stack (4)              { Sum stack values SP: -3 }
ADD Param (24)             { add param register }
NFU (7)                    { Neuron function }
STO Value (24)             { Store value register SP: -1 }
RCL Value (24)             { Recall value register SP: +1 }
MULT Param (25)            { multiply with param register }
TEST Drop Value (25)       { Test drop value register }
STO Value (25)             { Store value register SP: -1 }
RCL Value (25)             { Recall value register SP: +1 }
RCL Value (1)              { Recall value register SP: +1 }
MULT Param (26)            { multiply with param register }
TEST Drop Value (26)       { Test drop value register }
STO Value (26)             { Store value register SP: -1 }
RCL Value (26)             { Recall value register SP: +1 }
RCL Value (4)              { Recall value register SP: +1 }
MULT Param (27)            { multiply with param register }
TEST Drop Value (27)       { Test drop value register }
STO Value (27)             { Store value register SP: -1 }
RCL Value (27)             { Recall value register SP: +1 }
RCL Value (7)              { Recall value register SP: +1 }
MULT Param (28)            { multiply with param register }
TEST Drop Value (28)       { Test drop value register }
STO Value (28)             { Store value register SP: -1 }
RCL Value (28)             { Recall value register SP: +1 }
RCL Value (10)             { Recall value register SP: +1 }
MULT Param (29)            { multiply with param register }
TEST Drop Value (29)       { Test drop value register }
STO Value (29)             { Store value register SP: -1 }
RCL Value (29)             { Recall value register SP: +1 }
SUM Stack (4)              { Sum stack values SP: -3 }
ADD Param (30)             { add param register }
NFU (7)                    { Neuron function }
STO Value (30)             { Store value register SP: -1 }
RCL Value (30)             { Recall value register SP: +1 }
MULT Param (31)            { multiply with param register }
TEST Drop Value (31)       { Test drop value register }
STO Value (31)             { Store value register SP: -1 }
RCL Value (31)             { Recall value register SP: +1 }
SUM Stack (4)              { Sum stack values SP: -3 }
ADD Param (32)             { add param register }
NFU (7)                    { Neuron function }
STO Value (32)             { Store value register SP: -1 }
RCL Value (32)             { Recall value register SP: +1 }
DROP                       { drop stack value SP: -1 }
RCL Value (32)             { Recall value register SP: +1 }
DROP                       { drop stack value SP: -1 }
---------------------------------------
 Passes:1
 MinLen:310
 MaxLen:310
 TotLen:310
 AvgLen:310
 Pass:0  Threads:1  Len:310  AvgLen:310
---------------------------------------
```

Lots of instructions ☺

Anyway the principle execution of this network is explained here…

First the program wants to read the input. It uses the RCL (recall) input register 0 which is the only input in this case. The input is fed onto the stack and the SP:+1 tells us that the stack pointer is incremented with 1.

The next instruction MULT param(0) multiplies the stack with parameter registry 0 and saves the value back to the stack. The stack result is changed but the pointer remains at the same position.

The next instruction TEST drop value (0) is used for connection dropouts if we want to use statistics to drop values calculated down to a 0 value.

And the next instruction STO value(0) stores the result for later usage in a value register. This shows the basic principle for a simple stack based assembly program. The compiler in the network can generate these programs for evaluation, delta calculations, back propagation, genetic evaluation etc… The important thing is that it will be able to generate these instructions from any type of network and that they are now fully decoupled from the graf.

Now this is the very first step towards a “neural network” HW. The instructions here is now interpreted by a VM just like java but imagine the possibilities having a dedicated HW to do this. The software that generates these instructions works pretty much like a c++ compiler and now the network can be run without knowledge about the graf or data structures.

Getting up to speed

But a sequential program could be very very slow when the complexity grows. In many stages there are NxM complexity introduced in instructions when layers are fully connected.

This is where the parallelize optimizer kicks in. The assembly language is constructed in a way that the optimal parallel execution can be calculated by an optimizer. Lets take a look at the same program when it is parallelized…

Even this simple network generates a lot of instruction but that’s life with assembler…

The most interesting part right now is actually printed at the end of the disassembly. It is..

```
---------------------------------------
 Passes:6
 MinLen:8
 MaxLen:16
 TotLen:304
 AvgLen:9
 Pass:0  Threads:4  Len:32  AvgLen:8
 Pass:1  Threads:4  Len:32  AvgLen:8
 Pass:2  Threads:16  Len:128  AvgLen:8
 Pass:3  Threads:4  Len:64  AvgLen:16
 Pass:4  Threads:4  Len:32  AvgLen:8
 Pass:5  Threads:1  Len:16  AvgLen:16
---------------------------------------
```

This tells us that the previous sequential program is now divided into 6 sequential programs but where each program have a number of concurrent threads. Especially pass 2 is very parallelized with 16 concurrent threads and this is typical a parallel concurrency level in a 2 hidden layers with 4 neurons (4x4) complexity.

This concurrency will increase a lot when the network interconnections will increase.

The other interesting part is that instead of 310 sequential instructions we have about 8+8+8+16+8+16=64 sequential instructions in parallel so this parallel program will run 310/64=4.8 times faster theoretically (some latencies here and there)

The **dependency** info shows what thread we need to wait for and **pass** info shows what sequence slot we execute in.

```
-- Compiled Cortex Program --
Instructions:311
ValueStates:33
Parameters:33
LatencyStates:0
```

```
SignalSources:1
InputSignals:1

------------ SubRoutines ---------------
 Threads:33
---------------------------------------
---- Thread:0 ----
 Offset:0
 Length:8
 Pass:0
---- Commands ----
RCL Input (0)              { Recall input register SP: +1 }
MULT Param (0)             { multiply with param register }
TEST Drop Value (0)        { Test drop value register }
STO Value (0)              { Store value register SP: -1 }
---------------------------------------
---- Thread:1 ----
 Offset:8
 Length:8
 Pass:1
 Dependancy:0
---- Commands ----
RCL Value (0)              { Recall value register SP: +1 }
ADD Param (1)              { add param register }
NFU (7)                    { Neuron function }
STO Value (1)              { Store value register SP: -1 }
---------------------------------------
---- Thread:2 ----
 Offset:16
 Length:8
 Pass:2
 Dependancy:1
---- Commands ----
RCL Value (1)              { Recall value register SP: +1 }
MULT Param (2)             { multiply with param register }
TEST Drop Value (2)        { Test drop value register }
STO Value (2)              { Store value register SP: -1 }
---------------------------------------
---- Thread:3 ----
 Offset:24
 Length:8
 Pass:0
---- Commands ----
RCL Input (0)              { Recall input register SP: +1 }
MULT Param (3)             { multiply with param register }
TEST Drop Value (3)        { Test drop value register }
STO Value (3)              { Store value register SP: -1 }
---------------------------------------
---- Thread:4 ----
 Offset:32
 Length:8
 Pass:1
 Dependancy:3
---- Commands ----
RCL Value (3)              { Recall value register SP: +1 }
ADD Param (4)              { add param register }
NFU (7)                    { Neuron function }
STO Value (4)              { Store value register SP: -1 }
---------------------------------------
---- Thread:5 ----
 Offset:40
 Length:8
 Pass:2
 Dependancy:4
---- Commands ----
RCL Value (4)              { Recall value register SP: +1 }
MULT Param (5)             { multiply with param register }
TEST Drop Value (5)        { Test drop value register }
STO Value (5)              { Store value register SP: -1 }
---------------------------------------
---- Thread:6 ----
 Offset:48
 Length:8
 Pass:0
---- Commands ----
RCL Input (0)              { Recall input register SP: +1 }
MULT Param (6)             { multiply with param register }
```

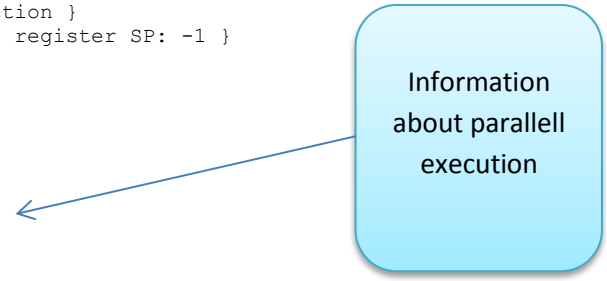
```
TEST Drop Value (6)         { Test drop value register }
STO Value (6)               { Store value register SP: -1 }
----------------------------------------
---- Thread:7 ----
 Offset:56
 Length:8
 Pass:1
 Dependancy:6
---- Commands ----
RCL Value (6)               { Recall value register SP: +1 }
ADD Param (7)               { add param register }
NFU (7)                     { Neuron function }
STO Value (7)               { Store value register SP: -1 }
----------------------------------------
---- Thread:8 ----
 Offset:64
 Length:8
 Pass:2
 Dependancy:7
---- Commands ----
RCL Value (7)               { Recall value register SP: +1 }
MULT Param (8)              { multiply with param register }
TEST Drop Value (8)         { Test drop value register }
STO Value (8)               { Store value register SP: -1 }
----------------------------------------
---- Thread:9 ----
 Offset:72
 Length:8
 Pass:0
---- Commands ----
RCL Input (0)               { Recall input register SP: +1 }
MULT Param (9)              { multiply with param register }
TEST Drop Value (9)         { Test drop value register }
STO Value (9)               { Store value register SP: -1 }
----------------------------------------
---- Thread:10 ----
 Offset:80
 Length:8
 Pass:1
 Dependancy:9
---- Commands ----
RCL Value (9)               { Recall value register SP: +1 }
ADD Param (10)              { add param register }
NFU (7)                     { Neuron function }
STO Value (10)              { Store value register SP: -1 }
----------------------------------------
---- Thread:11 ----
 Offset:88
 Length:8
 Pass:2
 Dependancy:10
---- Commands ----
RCL Value (10)              { Recall value register SP: +1 }
MULT Param (11)             { multiply with param register }
TEST Drop Value (11)        { Test drop value register }
STO Value (11)              { Store value register SP: -1 }
----------------------------------------
---- Thread:12 ----
 Offset:96
 Length:16
 Pass:3
 Dependancy:2
 Dependancy:5
 Dependancy:8
 Dependancy:11
---- Commands ----
RCL Value (2)               { Recall value register SP: +1 }
RCL Value (5)               { Recall value register SP: +1 }
RCL Value (8)               { Recall value register SP: +1 }
RCL Value (11)              { Recall value register SP: +1 }
SUM Stack (4)               { Sum stack values SP: -3 }
ADD Param (12)              { add param register }
NFU (7)                     { Neuron function }
STO Value (12)              { Store value register SP: -1 }
----------------------------------------
---- Thread:13 ----
 Offset:112
```

```
 Length:8
 Pass:4
 Dependancy:12
---- Commands ----
RCL Value (12)              { Recall value register SP: +1 }
MULT Param (13)             { multiply with param register }
TEST Drop Value (13)        { Test drop value register }
STO Value (13)              { Store value register SP: -1 }
----------------------------------------
---- Thread:14 ----
 Offset:120
 Length:8
 Pass:2
 Dependancy:1
---- Commands ----
RCL Value (1)               { Recall value register SP: +1 }
MULT Param (14)             { multiply with param register }
TEST Drop Value (14)        { Test drop value register }
STO Value (14)              { Store value register SP: -1 }
----------------------------------------
---- Thread:15 ----
 Offset:128
 Length:8
 Pass:2
 Dependancy:4
---- Commands ----
RCL Value (4)               { Recall value register SP: +1 }
MULT Param (15)             { multiply with param register }
TEST Drop Value (15)        { Test drop value register }
STO Value (15)              { Store value register SP: -1 }
----------------------------------------
---- Thread:16 ----
 Offset:136
 Length:8
 Pass:2
 Dependancy:7
---- Commands ----
RCL Value (7)               { Recall value register SP: +1 }
MULT Param (16)             { multiply with param register }
TEST Drop Value (16)        { Test drop value register }
STO Value (16)              { Store value register SP: -1 }
----------------------------------------
---- Thread:17 ----
 Offset:144
 Length:8
 Pass:2
 Dependancy:10
---- Commands ----
RCL Value (10)              { Recall value register SP: +1 }
MULT Param (17)             { multiply with param register }
TEST Drop Value (17)        { Test drop value register }
STO Value (17)              { Store value register SP: -1 }
----------------------------------------
---- Thread:18 ----
 Offset:152
 Length:16
 Pass:3
 Dependancy:14
 Dependancy:15
 Dependancy:16
 Dependancy:17
---- Commands ----
RCL Value (14)              { Recall value register SP: +1 }
RCL Value (15)              { Recall value register SP: +1 }
RCL Value (16)              { Recall value register SP: +1 }
RCL Value (17)              { Recall value register SP: +1 }
SUM Stack (4)               { Sum stack values SP: -3 }
ADD Param (18)              { add param register }
NFU (7)                     { Neuron function }
STO Value (18)              { Store value register SP: -1 }
----------------------------------------
---- Thread:19 ----
 Offset:168
 Length:8
 Pass:4
 Dependancy:18
---- Commands ----
```

```
RCL Value (18)              { Recall value register SP: +1 }
MULT Param (19)             { multiply with param register }
TEST Drop Value (19)        { Test drop value register }
STO Value (19)              { Store value register SP: -1 }
----------------------------------------
---- Thread:20 ----
 Offset:176
 Length:8
 Pass:2
 Dependancy:1
---- Commands ----
RCL Value (1)               { Recall value register SP: +1 }
MULT Param (20)             { multiply with param register }
TEST Drop Value (20)        { Test drop value register }
STO Value (20)              { Store value register SP: -1 }
----------------------------------------
---- Thread:21 ----
 Offset:184
 Length:8
 Pass:2
 Dependancy:4
---- Commands ----
RCL Value (4)               { Recall value register SP: +1 }
MULT Param (21)             { multiply with param register }
TEST Drop Value (21)        { Test drop value register }
STO Value (21)              { Store value register SP: -1 }
----------------------------------------
---- Thread:22 ----
 Offset:192
 Length:8
 Pass:2
 Dependancy:7
---- Commands ----
RCL Value (7)               { Recall value register SP: +1 }
MULT Param (22)             { multiply with param register }
TEST Drop Value (22)        { Test drop value register }
STO Value (22)              { Store value register SP: -1 }
----------------------------------------
---- Thread:23 ----
 Offset:200
 Length:8
 Pass:2
 Dependancy:10
---- Commands ----
RCL Value (10)              { Recall value register SP: +1 }
MULT Param (23)             { multiply with param register }
TEST Drop Value (23)        { Test drop value register }
STO Value (23)              { Store value register SP: -1 }
----------------------------------------
---- Thread:24 ----
 Offset:208
 Length:16
 Pass:3
 Dependancy:20
 Dependancy:21
 Dependancy:22
 Dependancy:23
---- Commands ----
RCL Value (20)              { Recall value register SP: +1 }
RCL Value (21)              { Recall value register SP: +1 }
RCL Value (22)              { Recall value register SP: +1 }
RCL Value (23)              { Recall value register SP: +1 }
SUM Stack (4)               { Sum stack values SP: -3 }
ADD Param (24)              { add param register }
NFU (7)                     { Neuron function }
STO Value (24)              { Store value register SP: -1 }
----------------------------------------
---- Thread:25 ----
 Offset:224
 Length:8
 Pass:4
 Dependancy:24
---- Commands ----
RCL Value (24)              { Recall value register SP: +1 }
MULT Param (25)             { multiply with param register }
TEST Drop Value (25)        { Test drop value register }
STO Value (25)              { Store value register SP: -1 }
```

```
----------------------------------------
---- Thread:26 ----
 Offset:232
 Length:8
 Pass:2
 Dependancy:1
---- Commands ----
RCL Value (1)               { Recall value register SP: +1 }
MULT Param (26)             { multiply with param register }
TEST Drop Value (26)        { Test drop value register }
STO Value (26)              { Store value register SP: -1 }
----------------------------------------
---- Thread:27 ----
 Offset:240
 Length:8
 Pass:2
 Dependancy:4
---- Commands ----
RCL Value (4)               { Recall value register SP: +1 }
MULT Param (27)             { multiply with param register }
TEST Drop Value (27)        { Test drop value register }
STO Value (27)              { Store value register SP: -1 }
----------------------------------------
---- Thread:28 ----
 Offset:248
 Length:8
 Pass:2
 Dependancy:7
---- Commands ----
RCL Value (7)               { Recall value register SP: +1 }
MULT Param (28)             { multiply with param register }
TEST Drop Value (28)        { Test drop value register }
STO Value (28)              { Store value register SP: -1 }
----------------------------------------
---- Thread:29 ----
 Offset:256
 Length:8
 Pass:2
 Dependancy:10
---- Commands ----
RCL Value (10)              { Recall value register SP: +1 }
MULT Param (29)             { multiply with param register }
TEST Drop Value (29)        { Test drop value register }
STO Value (29)              { Store value register SP: -1 }
----------------------------------------
---- Thread:30 ----
 Offset:264
 Length:16
 Pass:3
 Dependancy:26
 Dependancy:27
 Dependancy:28
 Dependancy:29
---- Commands ----
RCL Value (26)              { Recall value register SP: +1 }
RCL Value (27)              { Recall value register SP: +1 }
RCL Value (28)              { Recall value register SP: +1 }
RCL Value (29)              { Recall value register SP: +1 }
SUM Stack (4)               { Sum stack values SP: -3 }
ADD Param (30)              { add param register }
NFU (7)                     { Neuron function }
STO Value (30)              { Store value register SP: -1 }
----------------------------------------
---- Thread:31 ----
 Offset:280
 Length:8
 Pass:4
 Dependancy:30
---- Commands ----
RCL Value (30)              { Recall value register SP: +1 }
MULT Param (31)             { multiply with param register }
TEST Drop Value (31)        { Test drop value register }
STO Value (31)              { Store value register SP: -1 }
----------------------------------------
---- Thread:32 ----
 Offset:288
 Length:16
```

```
 Pass:5
 Dependancy:13
 Dependancy:19
 Dependancy:25
 Dependancy:31
---- Commands ----
RCL Value (13)              { Recall value register SP: +1 }
RCL Value (19)              { Recall value register SP: +1 }
RCL Value (25)              { Recall value register SP: +1 }
RCL Value (31)              { Recall value register SP: +1 }
SUM Stack (4)               { Sum stack values SP: -3 }
ADD Param (32)              { add param register }
NFU (7)                     { Neuron function }
STO Value (32)              { Store value register SP: -1 }
----------------------------------------
 Passes:6
 MinLen:8
 MaxLen:16
 TotLen:304
 AvgLen:9
 Pass:0  Threads:4  Len:32  AvgLen:8
 Pass:1  Threads:4  Len:32  AvgLen:8
 Pass:2  Threads:16  Len:128  AvgLen:8
 Pass:3  Threads:4  Len:64  AvgLen:16
 Pass:4  Threads:4  Len:32  AvgLen:8
 Pass:5  Threads:1  Len:16  AvgLen:16
```



So now we have automatically made a fully parallel program execution of the network. Even if the network contains parallel layers that would be solved by an ordinary neural network implementation as a sequence, it will in this implementation be fully parallelized.

If we want to balance the execution on X parallel thread, we can easily schedule each pass using X threads to do the load balancing.

As you can se also there is no dependency on the graf structure as well. So the parallel program can solve LSTM, recurrent networks, recursive (with a level limit) networks with both forward and backward connections. Connections far away (axons) that reaches across multiple layers is also dealt with as a normal connection. The network can even be generated by genetic rules and evolution (***tada!, This is my aha moment***)

Really really getting up to speed…

To really get up to speed and capacity we need to actually execute these instructions in a fast way. You can see that even if the instructions represent a lean machine code program it isn't yet.

We need to accelerate the execution. To make a JIT compiler just like java or even a dedicated HW for this purpose will help us. We have the goal to execute 20.000.000.000 neurons to represent the 6 cortex layers of a human brain and that's a lot….

The answer or actually my strategy right now is to have backed compiler that right now allows me to execute on existing HW in a better way.

E.g lets use an intel x64 simd aware platform. How does the code look then…

```
00250003  push        ebx
00250004  push        edx
00250005  push        esi
00250006  mov         eax,dword ptr [ebp+8]
00250009  mov         ebx,dword ptr [eax+0A8h]
0025000F  mov         edx,dword ptr [eax+48h]
```

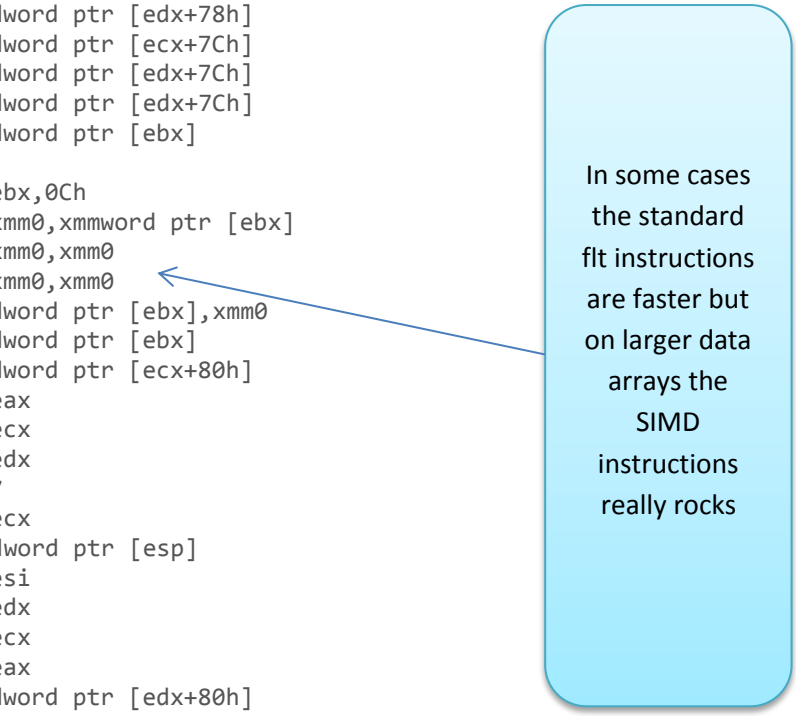
```
00250015  mov         ecx,dword ptr [eax+18h]
0025001B  wait
0025001C  fninit
0025001E  mov         esi,dword ptr [eax+3Ch]
00250024  fld         dword ptr [esi]
0025002A  fmul        dword ptr [ecx]
00250030  fstp        dword ptr [edx]
00250036  fld         dword ptr [edx]
0025003C  fadd        dword ptr [ecx+4]
00250042  push        eax
00250043  push        ecx
00250044  push        edx
00250045  push        7
0025004A  push        ecx
0025004B  fstp        dword ptr [esp]
0025004E  mov         esi,0F13C6E9h
00250053  call        esi
00250055  pop         edx
00250056  pop         ecx
00250057  pop         eax
00250058  fstp        dword ptr [edx+4]
0025005E  fld         dword ptr [edx+4]
00250064  fmul        dword ptr [ecx+8]
0025006A  fstp        dword ptr [edx+8]
00250070  fld         dword ptr [edx+8]
00250076  fstp        dword ptr [ebx]
00250078  mov         esi,dword ptr [eax+3Ch]
0025007E  fld         dword ptr [esi]
00250084  fmul        dword ptr [ecx+0Ch]
0025008A  fstp        dword ptr [edx+0Ch]
00250090  fld         dword ptr [edx+0Ch]
00250096  fadd        dword ptr [ecx+10h]
0025009C  push        eax
0025009D  push        ecx
0025009E  push        edx
0025009F  push        7
002500A4  push        ecx
002500A5  fstp        dword ptr [esp]
002500A8  mov         esi,0F13C6E9h
002500AD  call        esi
002500AF  pop         edx
002500B0  pop         ecx
002500B1  pop         eax
002500B2  fstp        dword ptr [edx+10h]
002500B8  fld         dword ptr [edx+10h]
002500BE  fmul        dword ptr [ecx+14h]
002500C4  fstp        dword ptr [edx+14h]
002500CA  fld         dword ptr [edx+14h]
002500D0  add         ebx,4
002500D3  fstp        dword ptr [ebx]
002500D5  mov         esi,dword ptr [eax+3Ch]
002500DB  fld         dword ptr [esi]
002500E1  fmul        dword ptr [ecx+18h]
002500E7  fstp        dword ptr [edx+18h]
002500ED  fld         dword ptr [edx+18h]
002500F3  fadd        dword ptr [ecx+1Ch]
002500F9  push        eax
002500FA  push        ecx
002500FB  push        edx
002500FC  push        7
00250101  push        ecx
00250102  fstp        dword ptr [esp]
```

```
00250105  mov         esi,0F13C6E9h
0025010A  call        esi
0025010C  pop         edx
0025010D  pop         ecx
0025010E  pop         eax
0025010F  fstp        dword ptr [edx+1Ch]
00250115  fld         dword ptr [edx+1Ch]
0025011B  fmul        dword ptr [ecx+20h]
00250121  fstp        dword ptr [edx+20h]
00250127  fld         dword ptr [edx+20h]
0025012D  add         ebx,4
00250130  fstp        dword ptr [ebx]
00250132  mov         esi,dword ptr [eax+3Ch]
00250138  fld         dword ptr [esi]
0025013E  fmul        dword ptr [ecx+24h]
00250144  fstp        dword ptr [edx+24h]
0025014A  fld         dword ptr [edx+24h]
00250150  fadd        dword ptr [ecx+28h]
00250156  push        eax
00250157  push        ecx
00250158  push        edx
00250159  push        7
0025015E  push        ecx
0025015F  fstp        dword ptr [esp]
00250162  mov         esi,0F13C6E9h
00250167  call        esi
00250169  pop         edx
0025016A  pop         ecx
0025016B  pop         eax
0025016C  fstp        dword ptr [edx+28h]
00250172  fld         dword ptr [edx+28h]
00250178  fmul        dword ptr [ecx+2Ch]
0025017E  fstp        dword ptr [edx+2Ch]
00250184  fld         dword ptr [edx+2Ch]
0025018A  add         ebx,4
0025018D  fstp        dword ptr [ebx]
0025018F  fldz
00250191  sub         ebx,0Ch
00250194  movaps      xmm0,xmmword ptr [ebx]
00250197  haddps      xmm0,xmm0
0025019B  haddps      xmm0,xmm0
0025019F  movss       dword ptr [ebx],xmm0
002501A3  fadd        dword ptr [ebx]
002501A5  fadd        dword ptr [ecx+30h]
002501AB  push        eax
002501AC  push        ecx
002501AD  push        edx
002501AE  push        7
002501B3  push        ecx
002501B4  fstp        dword ptr [esp]
002501B7  call        esi
002501B9  pop         edx
002501BA  pop         ecx
002501BB  pop         eax
002501BC  fstp        dword ptr [edx+30h]
002501C2  fld         dword ptr [edx+30h]
002501C8  fmul        dword ptr [ecx+34h]
002501CE  fstp        dword ptr [edx+34h]
002501D4  fld         dword ptr [edx+34h]
002501DA  fstp        dword ptr [ebx]
002501DC  fld         dword ptr [edx+4]
002501E2  fmul        dword ptr [ecx+38h]
```

```
002501E8  fstp        dword ptr [edx+38h]
002501EE  fld         dword ptr [edx+38h]
002501F4  add         ebx,4
002501F7  fstp        dword ptr [ebx]
002501F9  fld         dword ptr [edx+10h]
002501FF  fmul        dword ptr [ecx+3Ch]
00250205  fstp        dword ptr [edx+3Ch]
0025020B  fld         dword ptr [edx+3Ch]
00250211  add         ebx,4
00250214  fstp        dword ptr [ebx]
00250216  fld         dword ptr [edx+1Ch]
0025021C  fmul        dword ptr [ecx+40h]
00250222  fstp        dword ptr [edx+40h]
00250228  fld         dword ptr [edx+40h]
0025022E  add         ebx,4
00250231  fstp        dword ptr [ebx]
00250233  fld         dword ptr [edx+28h]
00250239  fmul        dword ptr [ecx+44h]
0025023F  fstp        dword ptr [edx+44h]
00250245  fld         dword ptr [edx+44h]
0025024B  fadd        dword ptr [ebx]
0025024D  sub         ebx,4
00250250  fadd        dword ptr [ebx]
00250252  sub         ebx,4
00250255  fadd        dword ptr [ebx]
00250257  fadd        dword ptr [ecx+48h]
0025025D  push        eax
0025025E  push        ecx
0025025F  push        edx
00250260  push        7
00250265  push        ecx
00250266  fstp        dword ptr [esp]
00250269  call        esi
0025026B  pop         edx
0025026C  pop         ecx
0025026D  pop         eax
0025026E  fstp        dword ptr [edx+48h]
00250274  fld         dword ptr [edx+48h]
0025027A  fmul        dword ptr [ecx+4Ch]
00250280  fstp        dword ptr [edx+4Ch]
00250286  fld         dword ptr [edx+4Ch]
0025028C  fstp        dword ptr [ebx]
0025028E  fld         dword ptr [edx+4]
00250294  fmul        dword ptr [ecx+50h]
0025029A  fstp        dword ptr [edx+50h]
002502A0  fld         dword ptr [edx+50h]
002502A6  add         ebx,4
002502A9  fstp        dword ptr [ebx]
002502AB  fld         dword ptr [edx+10h]
002502B1  fmul        dword ptr [ecx+54h]
002502B7  fstp        dword ptr [edx+54h]
002502BD  fld         dword ptr [edx+54h]
002502C3  add         ebx,4
002502C6  fstp        dword ptr [ebx]
002502C8  fld         dword ptr [edx+1Ch]
002502CE  fmul        dword ptr [ecx+58h]
002502D4  fstp        dword ptr [edx+58h]
002502DA  fld         dword ptr [edx+58h]
002502E0  add         ebx,4
002502E3  fstp        dword ptr [ebx]
002502E5  fld         dword ptr [edx+28h]
002502EB  fmul        dword ptr [ecx+5Ch]
```

```
002502F1  fstp        dword ptr [edx+5Ch]
002502F7  fld         dword ptr [edx+5Ch]
002502FD  fadd        dword ptr [ebx]
002502FF  sub         ebx,4
00250302  fadd        dword ptr [ebx]
00250304  sub         ebx,4
00250307  fadd        dword ptr [ebx]
00250309  fadd        dword ptr [ecx+60h]
0025030F  push        eax
00250310  push        ecx
00250311  push        edx
00250312  push        7
00250317  push        ecx
00250318  fstp        dword ptr [esp]
0025031B  call        esi
0025031D  pop         edx
0025031E  pop         ecx
0025031F  pop         eax
00250320  fstp        dword ptr [edx+60h]
00250326  fld         dword ptr [edx+60h]
0025032C  fmul        dword ptr [ecx+64h]
00250332  fstp        dword ptr [edx+64h]
00250338  fld         dword ptr [edx+64h]
0025033E  fstp        dword ptr [ebx]
00250340  fld         dword ptr [edx+4]
00250346  fmul        dword ptr [ecx+68h]
0025034C  fstp        dword ptr [edx+68h]
00250352  fld         dword ptr [edx+68h]
00250358  add         ebx,4
0025035B  fstp        dword ptr [ebx]
0025035D  fld         dword ptr [edx+10h]
00250363  fmul        dword ptr [ecx+6Ch]
00250369  fstp        dword ptr [edx+6Ch]
0025036F  fld         dword ptr [edx+6Ch]
00250375  add         ebx,4
00250378  fstp        dword ptr [ebx]
0025037A  fld         dword ptr [edx+1Ch]
00250380  fmul        dword ptr [ecx+70h]
00250386  fstp        dword ptr [edx+70h]
0025038C  fld         dword ptr [edx+70h]
00250392  add         ebx,4
00250395  fstp        dword ptr [ebx]
00250397  fld         dword ptr [edx+28h]
0025039D  fmul        dword ptr [ecx+74h]
002503A3  fstp        dword ptr [edx+74h]
002503A9  fld         dword ptr [edx+74h]
002503AF  fadd        dword ptr [ebx]
002503B1  sub         ebx,4
002503B4  fadd        dword ptr [ebx]
002503B6  sub         ebx,4
002503B9  fadd        dword ptr [ebx]
002503BB  fadd        dword ptr [ecx+78h]
002503C1  push        eax
002503C2  push        ecx
002503C3  push        edx
002503C4  push        7
002503C9  push        ecx
002503CA  fstp        dword ptr [esp]
002503CD  call        esi
002503CF  pop         edx
002503D0  pop         ecx
002503D1  pop         eax
```

```
002503D2  fstp        dword ptr [edx+78h]
002503D8  fld         dword ptr [edx+78h]
002503DE  fmul        dword ptr [ecx+7Ch]
002503E4  fstp        dword ptr [edx+7Ch]
002503EA  fld         dword ptr [edx+7Ch]
002503F0  fstp        dword ptr [ebx]
002503F2  fldz
002503F4  sub         ebx,0Ch
002503F7  movaps      xmm0,xmmword ptr [ebx]
002503FA  haddps      xmm0,xmm0
002503FE  haddps      xmm0,xmm0
00250402  movss       dword ptr [ebx],xmm0
00250406  fadd        dword ptr [ebx]
00250408  fadd        dword ptr [ecx+80h]
0025040E  push        eax
0025040F  push        ecx
00250410  push        edx
00250411  push        7
00250416  push        ecx
00250417  fstp        dword ptr [esp]
0025041A  call        esi
0025041C  pop         edx
0025041D  pop         ecx
0025041E  pop         eax
0025041F  fstp        dword ptr [edx+80h]
00250425  fld         dword ptr [edx+80h]
0025042B  fstp        dword ptr [ebx]
0025042D  fld         dword ptr [edx+80h]
00250433  fstp        dword ptr [ebx]
00250435  mov         al,1
00250437  pop         esi
00250438  pop         edx
00250439  pop         ebx
0025043A  mov         esp,ebp
0025043C  pop         ebp
0025043D  ret         4
```



This is the sequential version of the code compiled by the backed compiler into x64 assembler. There are calls to the neuron function subroutine but all the standard *adds* and *muls* and *sums* are inline into pure SIMD assembler. The execution speed of this is about 19x times the speed of the virtual machine so there is a substantial speedup.

C++ Backend Single Thread

One of the backend compilers available is a C++ generator. It takes the Cortex Assembler opcodes and generates a generic C++ program.

This is the generated C++ program for the example network. 2 hidden layers and 4 neurons in each layer. The single thread C++ code runs in one pass using one thread. It has yet no optimization for unnecessary instructions so the compiler will take care of some of the redundant code lines in its optimizer.

```
gzBool t0_103_13_13_0_0_0(ctxVirtualMachineProgram *program)
{
	gzFloat stack[5];
	gzFloat *sp(stack);

	*sp=program->input[0];
```

```
        *sp*=program->parameters[0];
        program->values[0]=*sp;
        *sp=program->values[0];
        *sp+=program->parameters[1];
        ctxNeuronFunc_7(sp);
        program->values[1]=*sp;
        *sp=program->values[1];
        *sp*=program->parameters[2];
        program->values[2]=*sp;
        *sp=program->values[2];
        ++sp;
        *sp=program->input[0];
        *sp*=program->parameters[3];
        program->values[3]=*sp;
        *sp=program->values[3];
        *sp+=program->parameters[4];
        ctxNeuronFunc_7(sp);
        program->values[4]=*sp;
        *sp=program->values[4];
        *sp*=program->parameters[5];
        program->values[5]=*sp;
        *sp=program->values[5];
        ++sp;
        *sp=program->input[0];
        *sp*=program->parameters[6];
        program->values[6]=*sp;
        *sp=program->values[6];
        *sp+=program->parameters[7];
        ctxNeuronFunc_7(sp);
        program->values[7]=*sp;
        *sp=program->values[7];
        *sp*=program->parameters[8];
        program->values[8]=*sp;
        *sp=program->values[8];
        ++sp;
        *sp=program->input[0];
        *sp*=program->parameters[9];
        program->values[9]=*sp;
        *sp=program->values[9];
        *sp+=program->parameters[10];
        ctxNeuronFunc_7(sp);
        program->values[10]=*sp;
        *sp=program->values[10];
        *sp*=program->parameters[11];
        program->values[11]=*sp;
        *sp=program->values[11];
        sp-=3;
        *sp=*sp+sp[1]+sp[2]+sp[3];
        *sp+=program->parameters[12];
        ctxNeuronFunc_7(sp);
        program->values[12]=*sp;
        return TRUE;
}
```

The code generator generates a pass/thread info structure as well

```
ctxVirtualMachineProgramFunc program_info_pass_0[] = {
        t0_103_13_13_0_0,
};

ctxVirtualMachineProgramInfo program_info[] = {
        {program_info_pass_0,1},
};

const gzUInt32 program_info_length=1;

const gzUInt32 program_info_maxthreads=1;
```

By reading this structure you can see that the program calls one function in one thread.

C++ Backend Multi Thread

The backend can also generate parallel threads for the execution of this network.

```
// Cpp file for ctxVirtualMachineProgram : 0_103_13_13 Pass:0 Thread:0 Module:0

#include "ctxVirtualMachine.h"

gzBool m0_103_13_13_0_0_0(ctxVirtualMachineProgram *program)
{
	gzFloat stack[2];
	gzFloat *sp(stack);

	*sp=program->input[0];
	*sp*=program->parameters[0];
	program->values[0]=*sp;
	return TRUE;
}
```

The above code is the code generated for pass 0, Thread 0

```
// Cpp file for ctxVirtualMachineProgram : 0_103_13_13 Pass:1 Thread:2 Module:6

#include "ctxVirtualMachine.h"

gzBool m0_103_13_13_1_2_0(ctxVirtualMachineProgram *program)
{
	gzFloat stack[2];
	gzFloat *sp(stack);

	*sp=program->values[6];
	*sp+=program->parameters[7];
	ctxNeuronFunc_7(sp);
	program->values[7]=*sp;
	return TRUE;
}
```

The above code is the code generated for pass 1, thread 2

```
// Cpp file for ctxVirtualMachineProgram : 0_103_13_13 Pass:3 Thread:0 Module:12

#include "ctxVirtualMachine.h"

gzBool m0_103_13_13_3_0_0(ctxVirtualMachineProgram *program)
{
		gzFloat stack[5];
		gzFloat *sp(stack);

		*sp=program->values[2];
		++sp;
		*sp=program->values[5];
		++sp;
		*sp=program->values[8];
		++sp;
		*sp=program->values[11];
		sp-=3;
		*sp=*sp+sp[1]+sp[2]+sp[3];
		*sp+=program->parameters[12];
		ctxNeuronFunc_7(sp);
		program->values[12]=*sp;
		return TRUE;
}
```

And the above code is the final pass 3, thread 0 that sums all the values into value[12] register.

The pass info now looks like this

```
// Global Header file for ctxVirtualMachineProgram : 0_103_13_13

#pragma once
#include "ctxVirtualMachine.h"

#include "p0_103_13_13_0.h"
```

```
ctxVirtualMachineProgramFunc program_info_pass_0[] = {
              t0_103_13_13_0_0,
              t0_103_13_13_0_1,
              t0_103_13_13_0_2,
              t0_103_13_13_0_3,
};
#include "p0_103_13_13_1.h"


ctxVirtualMachineProgramFunc program_info_pass_1[] = {
              t0_103_13_13_1_0,
              t0_103_13_13_1_1,
              t0_103_13_13_1_2,
              t0_103_13_13_1_3,
};
#include "p0_103_13_13_2.h"


ctxVirtualMachineProgramFunc program_info_pass_2[] = {
              t0_103_13_13_2_0,
              t0_103_13_13_2_1,
              t0_103_13_13_2_2,
              t0_103_13_13_2_3,
};
#include "p0_103_13_13_3.h"


ctxVirtualMachineProgramFunc program_info_pass_3[] = {
              t0_103_13_13_3_0,
};

ctxVirtualMachineProgramInfo program_info[] = {
              {program_info_pass_0,4},
              {program_info_pass_1,4},
              {program_info_pass_2,4},
              {program_info_pass_3,1},
};

const gzUInt32 program_info_length=4;

const gzUInt32 program_info_maxthreads=4;
```

The code can be augmented with pragmas for OpenMP and for Xeon Phi processors.

Other Backends

Other backend could be OpenCL (implemented) or CUDA or dedicated FPGAs or dedicated CPUs for these instructions that allow the execution to really use the parallelism of the code. This is about where I am right now in my quest for the EANN brain named CorTeX.

There are of course a lot of potential in the other parts which is part of the ecosystem around EANN but this document describes the execution of the compiler-optimizer-backend.

/Anders Modén (CTO, ToolTech Software)

Annex A

Illustration of scheduling on fixed number of cores.

In this case we want to feed as much as possible in ex. Two threads. In the previous example we saw that the optimizer tried to schedule as parallel as possible but in many cases this is bad as we fill each thread with too little to do compared to the overhead of executing each tread.

In the previous example we saw that the optimizer wanted to schedule most of the passes in 4 threads and one pass (pass 2) with 16 threads.

If we instead want the optimizer to schedule in this example on two threads we get the following debug output.

Lets see what happends

```
------------ SubRoutines ---------------
 Threads:11                                        // Ok. 11 threads in 5 passes
```

```
---- Thread:0 ----
 Offset:0
 Length:16
 Pass:0
---- Commands ----
RCL Input (0)                                      { Recall input register SP: +1 }
MULT Param (0)                                     { multiply with param register }
TEST Drop Value (0)                                { Test drop value register }
STO Value (0)                                      { Store value register SP: -1 }
RCL Input (0)                                      { Recall input register SP: +1 }
MULT Param (3)                                     { multiply with param register }
TEST Drop Value (3)                                { Test drop value register }
STO Value (3)                                      { Store value register SP: -1 }
```

Now we have two synapses in value reg 0 and 3

```
---- Thread:1 ----
 Offset:16
 Length:16
 Pass:0
---- Commands ----
RCL Input (0)                                      { Recall input register SP: +1 }
MULT Param (6)                                     { multiply with param register }
TEST Drop Value (6)                                { Test drop value register }
STO Value (6)                                      { Store value register SP: -1 }
RCL Input (0)                                      { Recall input register SP: +1 }
MULT Param (9)                                     { multiply with param register }
TEST Drop Value (9)                                { Test drop value register }
STO Value (9)                                      { Store value register SP: -1 }
```

Now value reg 6 and 9 are updated. Note that there is a drop connection test as well. This ends the first pass.

```
---- Thread:2 ----
 Offset:32
 Length:16
 Pass:1
 Dependancy:0              // Dep of value reg 0,3
---- Commands ----
RCL Value (0)                                      { Recall value register SP: +1 }
```

```
ADD Param (1)                                   { add param register }
NFU (7)                                         { Neuron function }
STO Value (1)                                   { Store value register SP: -1 }
RCL Value (3)                                   { Recall value register SP: +1 }
ADD Param (4)                                   { add param register }
NFU (7)                                         { Neuron function }
STO Value (4)                                   { Store value register SP: -1 }
```

```
---- Thread:3 ----
 Offset:48
 Length:16
 Pass:1
 Dependancy:1
---- Commands ----
RCL Value (6)                                   { Recall value register SP: +1 }
ADD Param (7)                                   { add param register }
NFU (7)                                         { Neuron function }
STO Value (7)                                   { Store value register SP: -1 }
RCL Value (9)                                   { Recall value register SP: +1 }
ADD Param (10)                                  { add param register }
NFU (7)                                         { Neuron function }
STO Value (10)                                  { Store value register SP: -1 }
```

End of pass 2. Pass 2 contains all the neuron functions executed on the synapses

```
---- Thread:4 ----
 Offset:64
 Length:64
 Pass:2
 Dependancy:2
 Dependancy:3
---- Commands ----
RCL Value (1)                                   { Recall value register SP: +1 }
MULT Param (2)                                  { multiply with param register }
TEST Drop Value (2)                             { Test drop value register }
STO Value (2)                                   { Store value register SP: -1 }
RCL Value (4)                                   { Recall value register SP: +1 }
MULT Param (5)                                  { multiply with param register }
TEST Drop Value (5)                             { Test drop value register }
STO Value (5)                                   { Store value register SP: -1 }
RCL Value (7)                                   { Recall value register SP: +1 }
MULT Param (8)                                  { multiply with param register }
TEST Drop Value (8)                             { Test drop value register }
STO Value (8)                                   { Store value register SP: -1 }
RCL Value (10)                                  { Recall value register SP: +1 }
MULT Param (11)                                 { multiply with param register }
TEST Drop Value (11)                            { Test drop value register }
STO Value (11)                                  { Store value register SP: -1 }
RCL Value (1)                                   { Recall value register SP: +1 }
MULT Param (14)                                 { multiply with param register }
TEST Drop Value (14)                            { Test drop value register }
STO Value (14)                                  { Store value register SP: -1 }
RCL Value (4)                                   { Recall value register SP: +1 }
MULT Param (15)                                 { multiply with param register }
TEST Drop Value (15)                            { Test drop value register }
STO Value (15)                                  { Store value register SP: -1 }
RCL Value (7)                                   { Recall value register SP: +1 }
MULT Param (16)                                 { multiply with param register }
TEST Drop Value (16)                            { Test drop value register }
STO Value (16)                                  { Store value register SP: -1 }
RCL Value (10)                                  { Recall value register SP: +1 }
MULT Param (17)                                 { multiply with param register }
TEST Drop Value (17)                            { Test drop value register }
STO Value (17)                                  { Store value register SP: -1 }
```

This is a typical collection phase of all the activation signals from the neurons. Lots of MULT that can be converted to a REG reads – MULT(many) – Test drop (many) – REG writes (Not yet imp)

```
---- Thread:5 ----
 Offset:128
 Length:64
 Pass:2
 Dependancy:2
 Dependancy:3
---- Commands ----
RCL Value (1)                                   { Recall value register SP: +1 }
MULT Param (20)                                 { multiply with param register }
TEST Drop Value (20)                            { Test drop value register }
STO Value (20)                                  { Store value register SP: -1 }
RCL Value (4)                                   { Recall value register SP: +1 }
MULT Param (21)                                 { multiply with param register }
TEST Drop Value (21)                            { Test drop value register }
STO Value (21)                                  { Store value register SP: -1 }
RCL Value (7)                                   { Recall value register SP: +1 }
MULT Param (22)                                 { multiply with param register }
TEST Drop Value (22)                            { Test drop value register }
STO Value (22)                                  { Store value register SP: -1 }
RCL Value (10)                                  { Recall value register SP: +1 }
MULT Param (23)                                 { multiply with param register }
TEST Drop Value (23)                            { Test drop value register }
STO Value (23)                                  { Store value register SP: -1 }
RCL Value (1)                                   { Recall value register SP: +1 }
MULT Param (26)                                 { multiply with param register }
TEST Drop Value (26)                            { Test drop value register }
STO Value (26)                                  { Store value register SP: -1 }
RCL Value (4)                                   { Recall value register SP: +1 }
MULT Param (27)                                 { multiply with param register }
TEST Drop Value (27)                            { Test drop value register }
STO Value (27)                                  { Store value register SP: -1 }
RCL Value (7)                                   { Recall value register SP: +1 }
MULT Param (28)                                 { multiply with param register }
TEST Drop Value (28)                            { Test drop value register }
STO Value (28)                                  { Store value register SP: -1 }
RCL Value (10)                                  { Recall value register SP: +1 }
MULT Param (29)                                 { multiply with param register }
TEST Drop Value (29)                            { Test drop value register }
STO Value (29)                                  { Store value register SP: -1 }
```

End of pass 2. All mults are done

```
---- Thread:6 ----
 Offset:192
 Length:32
 Pass:3
 Dependancy:4
---- Commands ----
RCL Value (2)                                   { Recall value register SP: +1 }
RCL Value (5)                                   { Recall value register SP: +1 }
RCL Value (8)                                   { Recall value register SP: +1 }
RCL Value (11)                                  { Recall value register SP: +1 }
SUM Stack (4)                                   { Sum stack values SP: -3 }
ADD Param (12)                                  { add param register }
NFU (7)                                         { Neuron function }
STO Value (12)                                  { Store value register SP: -1 }
RCL Value (14)                                  { Recall value register SP: +1 }
RCL Value (15)                                  { Recall value register SP: +1 }
RCL Value (16)                                  { Recall value register SP: +1 }
RCL Value (17)                                  { Recall value register SP: +1 }
SUM Stack (4)                                   { Sum stack values SP: -3 }
ADD Param (18)                                  { add param register }
```

```
NFU (7)                                          { Neuron function }
STO Value (18)                                   { Store value register SP: -1 }
```

In this pass we see q lot of SUMS. This is already imp and adapted to SUM(4) SIMD instruction

```
---- Thread:7 ----
 Offset:224
 Length:32
 Pass:3
 Dependancy:5
---- Commands ----
RCL Value (20)                                   { Recall value register SP: +1 }
RCL Value (21)                                   { Recall value register SP: +1 }
RCL Value (22)                                   { Recall value register SP: +1 }
RCL Value (23)                                   { Recall value register SP: +1 }
SUM Stack (4)                                    { Sum stack values SP: -3 }
ADD Param (24)                                   { add param register }
NFU (7)                                          { Neuron function }
STO Value (24)                                   { Store value register SP: -1 }
RCL Value (26)                                   { Recall value register SP: +1 }
RCL Value (27)                                   { Recall value register SP: +1 }
RCL Value (28)                                   { Recall value register SP: +1 }
RCL Value (29)                                   { Recall value register SP: +1 }
SUM Stack (4)                                    { Sum stack values SP: -3 }
ADD Param (30)                                   { add param register }
NFU (7)                                          { Neuron function }
STO Value (30)                                   { Store value register SP: -1 }
```

Pass 3 finished with all the SUMs

```
---- Thread:8 ----
 Offset:256
 Length:16
 Pass:4
 Dependancy:6
---- Commands ----
RCL Value (12)                                   { Recall value register SP: +1 }
MULT Param (13)                                  { multiply with param register }
TEST Drop Value (13)                             { Test drop value register }
STO Value (13)                                   { Store value register SP: -1 }
RCL Value (18)                                   { Recall value register SP: +1 }
MULT Param (19)                                  { multiply with param register }
TEST Drop Value (19)                             { Test drop value register }
STO Value (19)                                   { Store value register SP: -1 }
```

```
---- Thread:9 ----
 Offset:272
 Length:16
 Pass:4
 Dependancy:7
---- Commands ----
RCL Value (24)                                   { Recall value register SP: +1 }
MULT Param (25)                                  { multiply with param register }
TEST Drop Value (25)                             { Test drop value register }
STO Value (25)                                   { Store value register SP: -1 }
RCL Value (30)                                   { Recall value register SP: +1 }
MULT Param (31)                                  { multiply with param register }
TEST Drop Value (31)                             { Test drop value register }
STO Value (31)                                   { Store value register SP: -1 }
```

```
---- Thread:10 ----
 Offset:288
 Length:16
 Pass:5
 Dependancy:8
 Dependancy:9
```

```
---- Commands ----
RCL Value (13)                                  { Recall value register SP: +1 }
RCL Value (19)                                  { Recall value register SP: +1 }
RCL Value (25)                                  { Recall value register SP: +1 }
RCL Value (31)                                  { Recall value register SP: +1 }
SUM Stack (4)                                   { Sum stack values SP: -3 }
ADD Param (32)                                  { add param register }
NFU (7)                                         { Neuron function }
STO Value (32)                                  { Store value register SP: -1 }
```

A final pass with the SUM of all weighed synapses in the final output

Value reg 32 is the result of the entire network

```
Passes:6
MinLen:16
MaxLen:64
TotLen:304
AvgLen:27
Pass:0  Threads:2  Len:32  AvgLen:16
Pass:1  Threads:2  Len:32  AvgLen:16
Pass:2  Threads:2  Len:128  AvgLen:64
Pass:3  Threads:2  Len:64  AvgLen:32
Pass:4  Threads:2  Len:32  AvgLen:16
Pass:5  Threads:1  Len:16  AvgLen:16
```

Thats the end of this example…

/Anders Modén (CTO, ToolTech Software)