EspamAI-Sesame manual

This is a manual for espamAI module, that allows to generate C++ Sesame applications. The manual concentrates only on Sesame application generation and should be used together with espamAIMan.pdf manual, which describes how to use EspamAI. The espamAIMan.pdf manual is located in the same folder as EspamAI-Sesame manual.

Requirements & Installation

To run espamAI-Sesame, both espamAI and Sesame tools should be installed.
1. espamAI requirements and installation steps are given in README.md at the espamAI root folder: https://git.liacs.nl/lerc/espam
2.Sesame tool and installation instructions: http://sesamesim.sourceforge.net/

General description

Input:

- Path to single DNN model in .onnx or .json format or a single CSDF model + options (see below)

Output:

Sesame application:

- .yml file, containing general application description.
- .cpp / .h file for every DNN/CSDF node, containing description and functionality of the corresponding DNN/CSDF node.

Execution

The espamAI-Sesame module execution is perfomed in three steps:

1. Generate Sesame application (using espamAI).
2. [optional] print and customize WCET specification
3. Simulate application running with Sesame.

Generate Sesame application

To generate the sesame application with espamAI-Sesame:
1. Cd to the Sesame application destination folder (any folder you like).
1. Call espam/bin/espam:

<my path to espam>/bin/espam --generate <input_file> --sesame [options]

Where <my path to espam> is a directory, contains espam files (bin, lib, src and other espamAI source dicrectories) e.g.

Print and customize WCET specification

In espamAI the worst-case times (WCET) specification is defined as a list of k-v pairs *operator: execution time* and stored in .json format. And example of WCET specification can be found in .../espam/src/espam/examples/time_spec/wcet_example.json:

```
{
  "MAXPOOL": 2,
  "CONV(5_5)": 75,
  "CONV": 3,
  "DENSEBLOCK_NONE(1_400,400_120)": 10400,
```

```
  "WRITE": 1,
  "SOFTMAX": 1,
  "READ": 1,
  "MAXPOOL(2_2_10)": 80,
  "DENSEBLOCK_NONE(1_120,120_84)": 2160,
  "DENSEBLOCK_SOFTMAX(1_84,84_10)": 504,
  "ReLU": 1,
  "CONV(5_5_6)": 450,
  "MAXPOOL(2_2_28)": 224,
  "NONE": 0
}
```

The *operator* ia s single opetator, performed in an CSDF node such as Convolution or Subsampling, the *execution time* is an operator WCET in time units. The time unit is an abstract time measurement unit, common for all the operators (e.g. milliseconds, seconds, minutes, etc).

Abstract and parametrized WCETs

The WCET can be *abstract* or *parametrized*. The abstract WCET specifies the operator complexity in time_units/input value, with only the operator name takes into account. An example of an *abstract* WCET is "CONV: 3", which means that “CONV” operator has complexity 3 time_units/input value. The abstract WCETs are used to compute the parametrized operator WCETs.

Unlike the *abstract* WCETs, *parametrized* WCETs takes into account the operator parameters and specify the operator execution time. An example of parametrized WCET is “CONV(5_5_6): 450”, which specifies that operator “CONV” with parameters “5,5,6” requires 450 time units to execute.

For performance evaluation, the parametrized WCETs have a priority over the abstract WCETs, e.g. for the specification above for Convolution 5x5 operator, the "CONV(5_5)": 75 k-v pair will be used to evaluate 5x5 Convolution operator time. To evaluate Convolution 3x3, he "CONV": 3 k-v pair will be used.

Operators, supported by default

The list of the parametrized operators, supported by espamAI by default is provided in ../espam/src/espam/docs/espamAI/DNN_supported_operators.pdf. For this operators the parametrized WCETs are automatically computed from the default abstract operators specification, provided in .../espam/src/espam/examples/time_spec/default_wcet.json.

If during performance evalution, espamAI finds an unsopported (not-default) operator, the operator execution time will be set to 1, and the warning *<operator> unknown execution time. Default time = 1 is set for <operator>* will be printed on console.

Generate WCETs

To generate the WCET specification, exploited by espamAI for provided iinput model

```
$  ./espam --generate  <path/to/model> --wcet
```

Customize WCETs

To set up your own WCET specification, perform following steps:

1. Generate default specification for an input model.
2. Change the .json file manually in any text editor

Simulate application running with Sesame

To simulate generated Sesame application, you will need Sesame simulation tool installed.

With installed Sesame simulation tool:

1. Go to sesame root
2. Set up sesame environment
3. Go to directory with an application, generated by espamAI
4. Provide sesame mapping file or reate virtual mapping for your application
5. Go to application sources.
6. Generate .o files for every CSDF node and .so library, contains all the application + simulate application running.
7. Go back to application root folder
8.Create text files with application processes traces.

Command-line options

Command-line options that take arguments. *Note: The table below contains only the options, related to Sesame application generation. For all other options see espamAIMan.pdf*

option	abbr	arguments	example
General options			
--generate	-g	Path to DNN/CSDF model	--generate ./tests/lenet.json
Model representation options (by default --layer-based option is set)			
--layer-based	-lb	none	-lb
--block-based	-bb	number of blocks	--bb 20
--split-step [optional], for -bb models only	none	Number of child nodes, obtainbed after one layer splitting	--split-step 4
--opt-fi [optional]	none	Level of optimization of application graph for inference 0 – do not optimizie 1 – remove Dropout and Reshape nodes 2 – (default) – remove Dropout and Reshape nodes + incapsulate adding of biases in predcessing nodes	--opt-fi 2
WCET specification			
--time-spec	none	Path to input specification of operators times	--time-spec ./time_spec.json

Command-line flags (The command-line options that are either present or not).

flag	abbr	description
General flags		
--help	-h	Printout help
--verbose	-V	Printout program progress information
WCET specification		
--wcet	none	Generate worst-case times specification for an input model

Examples

Step1: Generate Sesame application

Step1: Generate simple LB Sesame application from ./espam/bin directory

$ cd /home/user1/espam/bin
$ ./espam --generate ../src/espam/examples/DNN/onnx/mnist.onnx --sesame -lb

Description:
Generate Sesame application for ../src/espam/examples/DNN/onnx/mnist.onnx DNN model. Use layer-based mode (for more information about DNN modes see espamAIMan.pdf manual)

Expected output: files folder
./CNTKGraph/app //Sesame application in layer-based mode

Step1 (alternanive): Block-based Sesame application from ./espam/bin directory

Description:
$ ./espam --generate ../src/espam/examples/DNN/onnx/mnist.onnx --sesame -bb 50

Description:
Generate block-based Sesame application for ./tests/onnx/mnist.onnx DNN model. Split DNN model until it reaches <=50 blocks with default --split-step = 2 (For more information about layers splitting see paragraph "Neuron/Layer and Block-based DNN models" of espamAIMan.pdf manual).

Expected output: files folder
CNTKGraph/app //contains parallelised sesame application in -bb mode

Step2: Print and customize WCETs

NOTE: The DNN mode options (--lb, --bb) for Sesame application and WCET spefication should match!

Print WCETS:

$ ./espam --generate ../src/espam/examples/DNN/onnx/mnist.onnx --sesame -lb --wcet
Expected output:
./output_models/CNTKGraph/CNTKGraph_wcet_spec.json file generated and contais


```
{
  "MATMUL_NONE(1_256,256_10)": 512,
  "MAXPOOL": 1,
  "MAXPOOL(3_3_16)": 144,
  "CONV(5_5)": 25,
  "CONV": 1,
  "WRITE": 1,
  "READ": 1,
  "MAXPOOL(2_2_8)": 32,
  "CONV(5_5_8)": 200,
  "ReLU": 1,
```

```
  "NONE": 0
}
```

Customize WCETS:

2. Open the generated CNTKGraph_wcet_spec.json file in any text editor and change it manually, e.g. set "CONV(5_5)": 95.

Step 3: Simulate Sesame application with virtual mapping

An example for todorsNet_NB Sesame application with virtual mapping:

1. $ cd .../sesame/
2. $ source sesame.env
3. $ cd .../todorsNet_NB/
4. $ AppVirtualMapGenerator app/todorsNet_NB_app.yml > todorsNet_NB_appvirt_map.yml
5. $ cd app
6. $ make runtrace
7. $ cd ../
8. $ for i in `ls trace*`; do traceprinter $i > $i.txt; done

Expected output:

Run traces for every CSDF node with sequences of read/execute/write primitives, corresponding to the CSDF graph functionality.

Step 4: Generate mapping file

How to obtain the application mapping file from Sesame?

Work in progress

Integration with Sesame:

1. Sesame mapping from espamAI console.

2. Sesame simulation results with virtual/generated mapping from espamAI console.