Espam-AI manual

Requirements & Installation

See README.md at the espamAI root folder or at https://git.liacs.nl/lerc/espam

General description

Input:

- Path to single DNN model in .onnx or .json format + options (see below) *or*
- Path to single CSDF model in .json format *or*
- Directory with several DNN models (not recomended for large DNN models)

Output:

- DNN model evaluation results in .json format printed to console *or/and*
- Files, generated from DNN model and intended for CSDF-models for CSDF-models processing tools such as Sesame/DARTS/SDF3

Possibilities:

- evaluation : Evaluation of DNN/CSDF model in terms of power/performance
- generation : Generation of files intended for CSDF-models processing tools such as Sesame/DARTS/SDF3
- consistency checkout : Checkout of input models consistency

Execution

Execution from command line (after espamAI is configured and installed)

1. Execute from espam/bin:

```
cd <my path to espam>/bin
./espam [options]
```

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

Example:

```
cd  /vol/home/user1/espam/bin
./espam [options]
```

2. Execute from arbitrary directory *my_dir*

```
cd /path/to/my_dir
<my path to espam>/bin/espam [options]
```

Example:

```
cd  /vol/home/user1/my_dir
/vol/home/user1/espam/bin/espam [options]
```

Execution from executable jar (after espam.jar is made by make jar-exec command):

1. cd <jar directory>
2. java -jar espam.jar [options]

Tool running progress

(1) *Model(s) reading*: read input model(s)

(2) *Model(s) consistency checkout*: check if input model is consistent

(3) *Model(s) conversion [optional]*:

- conversion of input .onnx/json DNN model(s) to internal Network model(s) (initiated automatically after onnx/json model reading)
- conversion of internal Network model(s) to CSDF model(s): (initiated automatically, if evaluation flag if set or/and one or multiple *-csdf output files generation flags are set)

(4) *Model(s) transformation [optional]*:

-Perform split/merge transformations over the DNN model (see Neuron/Layer and Block-based DNN models)

(5) *Output files generation [optional]:* generates for input model output files, according to set files generation options.

(6) *Model evaluation [optional]:* evaluates one or several input models in terms of power/performance, by means of the DARTS/SDF3 tool.
TODO direct interface to SDF3 tool is not presented for current moment

Input and output files

DNN input models

EspamAI accept DNN models in .onnx or .json format
ONNX format : https://onnx.ai/
JSON format: see .../espam/src/espam/examples/DNN/json/ folder
DNN model in JSON format is represented as a [required] list of layers and [required] list of connections between layers. Each layer contains description of typical neuron inside.

Neuron/Layer and Block-based DNN models

EspamAI provides several types of DNN--> CSDF model granularity.
For layer-based model each node of a CSDF graph represents one layer of an input DNN model.
For neuron-based model each node of a CSDF graph represents one neuron of an input DNN model.
For block-based model each node of a CSDF graph represents one block of an input DNN model, where block is an abstraction of DNN model layer and DNN model neuron.
Block-based model takes on input layer-based DNN model and transforms it according to the following algorithm:

While (number of blocks<expected && split_up flag = true):
    Layer bottleneck = find_bottleneck(); // bottleneck node search is preformed by DARTS
    if (bottleneck can be split up)
        evaluate number of layers after bottleneck layer splitting;
        if (number of layers after splitting>expected)
            return;
        Split bottleneck into –split-step child Layers;
        Split all Dependent layers (nonlinear/maxpool ones), following the bottleneck layer
    else
        return;

As transformation is performed over DNN models,files (.json / .dot etc) generated from the model in -bb mode will be different from ones for input (layer-based) DNN model. If .json for block-based model is already generated, it can be reused without additional splitting in -lb mode.

CSDF input models

EspamAI accept CSDF graph models in internal .json format: see .../espam/src/espam/examples/CSDF/json/ folder
CSDF graph model in JSON format is represented as a [required] list of nodes and [required] list of connections between nodes. Each node contains list of Input/Output ports, related to node I/O connections.

Output files

Output files are generated in output models directory, specified during espamAI configuration. By default output files are generated in espamAI/execution/directory/output_models.
For every model created a directory, named after the model. E.g. for DNN called “LeNet”, the LeNet directory will be created.
Inside the folder there are files generated in accordance with the file generation flags. For all possible files the folder will have the following structure (each file or direcory is optional)

The files structure:
<Model_name>

- app // directory with sesame application
- Model_name_wcet_spec.json //time specification in .json format
- energy_spec.json //time specification in .json format
- dot //directory with DNN model in .dot format
- json //directory with DNN model in .json format
- sdfg //corresponding CSDF graph directory
    - dot //directory with CSDF model in .dot format
    - json //directory with CSDF model in .json format for DARTS
    - xml //directory with CSDF model in .xml format for SDF3

NOTE: if directory with this name already exist it will be overwritten!

Hardware specifications

EspamAI provides a possibility to manage time and energy specifications, used for input models evaluation.

Times (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 WCETs 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

Energy specification

Energy model is based on based on:
Di Liu, Jelena Spasic, Gang Chen, and Todor Stefanov,
Energy-Efficient Mapping of Real-Time Streaming Applications on Cluster Heterogeneous MPSoCs", In Proc. "13th Int. IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia'15), pp. 1-10, Amsterdam, The Netherlands, Oct. 8-9, 2015.

Example

To set up your own energyspecification, perform 3 following steps:
1. Generate current specification for an input model:

```
$  ./espam --generate ../src/espam/examples/DNN/json/lenet_NB.json -nb --wcenergy
```

Expected output:

File with default worst-case energy: ../output_models/lenet_LB/energy_spec.json of structure:

```
{
  "alpha": 3.03E-9,
  "beta": 0.155,
  "b": 2.621
}
```

2. Change the energy specification in any text editor (e.g. set beta to 0.275)

IMPORTANT: Energy specification should always contain alpha, beta and b parameters, only values can be changed!

3. Call the model with changed specification:

```
$  ./espam --evaluate ../src/espam/examples/DNN/json/lenet_NB.json -nb   \
--energy-spec ../output_models/lenet_NB/energy_spec.json
```

Expected output:

{ "id": 0, "execution_time": 359499.0, "energy": 4.237104834907729E-8, "memory": 787744.0, "processors": 14 }

Simulate model running with Sesame

To run model, generated with –sesame espamAI generation flag (see commands), 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.

Example 1: Run 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.

Generate images from .dot files

To generate images from .dot files use graphviz https://www.graphviz.org/

It should installed on Linux by default. In case it is not installed, use

```
$ sudo apt-get install graphviz
```

To see dot options use

$ dot --help

To generate .<format> file from <filename>.dot file use

$ dot -T<format> <filename>.dot -o <filename>.<format>

Example

Generate .png image of ../espam/output_models/todorsNet_NB/dot/todorsNet_NB.dot file:

$ dot -Tpng ../espam/output_models/todorsNet_NB/dot/todorsNet_NB.dot -o \
../espam/output_models/todorsNet_NB/dot/todorsNet_NB.png

Troubleshooting

1. It is recomended to run tests after the tool is build.
TODO provide run tests after tool installation

2. Tool is not accounted on huge graph models, so it is not recomended to use -nb mode for large DNNs. Otherwise, tool may work really long or even fall with memory (JavaHeapSpace/Python) errors.
For the same reason, it is not recommended to use --generation option with too many flags and multiple-models processing (--evaluation or --generation) for heavy models, especially in -nb mode

3. Do not forget to set --in-csdf flag for input CSDF graph models, otherwise CSDF model will not be processed.

4. If you are not sure about input model consistency, use --consistency option to check it.

5. Tool may print warnings about 'unsafe' libraries such as protobuf-java.jar. This warnings would not affect the tool running.

6. If tool is frozen, kill the corresponding command line process and try to use the same command with –verbose (v) option. It will be shown on which step (model reading, model conversion, model evaluation etc.) the problem occured and more details will be given.

7. If CSDF model is sent on input (with flag –in-csdf) , no specific DNN features (such as weights and proper data formats) will be taken into account. Thus, for processing DNN models it is recomended to use DNN .json/.onnx files. JSON files might be even preferable due to their small size.

8. In case of any problems, please contact:
s.minakova@liacs.leidenuniv.nl

Command-line options

Command-line options that take arguments

option	abbr	arguments	example
General options			
--generate	-g	Path to DNN/CSDFG model	--generate ./tests/lenet.json
--evaluate	-e	Path to DNN/CSDFG model	--evaluate ./tests/lenet.json
--consistency	-c	Path to DNN/CSDFG model	--consistency ./tests/lenet.json
Model representation options (by default --layer-based option is set)			
--layer-based	-lb	none	-lb
--neuron-based	-nb	none	-nb
--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
Hardware specification options			
--time-spec	none	Path to input specification of operators times	--time-spec ./time_spec.json
--energy-spec		Path to input specification of model energy	--energy-spec.json ./energy_spec.json

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

flag	abbr	description
General flags		
--help	-h	Printout help
--version	-v	Printout program version
--verbose	-V	Printout program progress information
--copyright	none	Printout copyright
Input model type flag (--in-dnn is set by default)		
--in-dnn	none	Input model is dnn model in .onnx or .json format
--in-csdf	none	Input model is csdf model in .json format
Files generation flags		
--json	none	Generate DNN graph in .json format
--dot	none	Generate DNN graph image in .dot format
--json-csdf	none	Generate CSDF graph in .json format for DARTS
--xml-csdf	none	Generate CSDF graph in .xml fomat for SDF3
--dot-csdf	none	Generate CSDF graph image in .dot format

--sesame	none	Generate code templates for Sesame
--wcet	none	Generate worst-case times specification for an input model
--wcenergy	none	Generate worst-case energy specification for an input model

Files generation examples

For more information about files generation see comments, output files.
**NOTE: All files generation examples are performed from .../espam/bin directory.* To perform examples from arbitrary directory *my_dir*

- Instead of ./ in espam call provide absolute path to espam/bin or relative path to espam/bin from *my_dir*
- Provide absolute path(s) to input file(s) or relative path(s) to input file(s) from *my_dir*

Example 1: DNN model

$ ./espam --generate ../src/espam/examples/DNN/json/lenet_LB.json --sesame -lb

Description:
Generate layer-based CSDF graph for ../src/espam/examples/DNN/json/lenet_LB.json DNN model. Provide it with Sesame application.

Expected output: files folder
<lenet_LB>
-app //sesame application in layer-based mode

Example 2: CSDF model

$ ./espam --generate ../src/espam/examples/CSDF/json/tinyCSDF.json --in-csdf --dot-csdf --sesame

Description:
For ../src/espam/examples/CSDF/json/tinyCSDF.json CSDF graph generate .dot file and Sesame application.

Expected output: files folder
<tinyCSDF>
-app //sesame application for an CSDF graph
-sdfg //CSDFG files directory
-dot //contains tinyCSDF.dot file of CSDF model

Example 3: DNN model in neuron-based mode

$ ./espam --generate ../src/espam/examples/DNN/json/lenet_NB.json --sesame --dot --json-csdf \
--xml-csdf --dot-csdf -nb

Description:
Generate neuron-based CSDF graph for ./tests/json/lenet_NB.json DNN model. Provide it with Sesame application, .dot files of DNN and CSDF model, .json and .xml files of CSDF model.

Expected output: files folder
<lenet_NB>
-app //contains sesame application in -nb mode
-dot //contains lenet_NB.dot file of DNN model
-sdfg //corresponding CSDF graph directory
-dot //contains lenet_NB.dot file CSDF model

-json //contains lenet_NB.json CSDF model suitable for DARTS\
-xml //contains lenet_NB.xml CSDF model suitable for SDF3

Example 4 : DNN model in block-based mode

$ ./espam --generate ../src/espam/examples/DNN/onnx/mnist.onnx --sesame --dot \
--json-csdf --xml-csdf --dot-csdf -bb -100 --split-step 4

Description:

Generate block-based CSDF graph for ./tests/onnx/mnist.onnx DNN model. Split model until it reaches <=100 blocks with –split-step = 4 (For more information about layers splitting see paragraph "Neuron/Layer and Block-based DNN models"). Provide it with Sesame application, .dot files of DNN and CSDF model, .json and .xml files of CSDF model.

Expected output: files folder
<CNTKGraph_NB>
-app //contains sesame application in -bb mode
-dot //contains CNTKGraph.dot file of DNN model
-sdfg //corresponding CSDF graph directory
-dot //contains CNTKGraph.dot file of CSDF model
-json //contains CNTKGraph.json CSDF model suitable for DARTS\
-xml //contains CNTKGraph xml CSDF model suitable for SDF3

Model evaluation examples

Example 1: DNN model

$ ./espam --evaluate ../src/espam/examples/DNN/json/lenet_LB.json -lb

Description:

Evaluate ../src/espam/examples/DNN/json/lenet_LB.json DNN model in terms of power/performance in -lb mode with default time specification.
** For more details about time and energy specification see 'Comments. hardware specification options' chapter below.*

Expected output:

{ "id": 0, "execution_time": 4361469.0, "energy": 5.8683824566753834E-9, "memory": 278280.0, "processors": 2 }

Example 2: CSDF model

$ ./espam --evaluate ../src/espam/examples/CSDF/json/tinyCSDF.json --in-csdf

Description:

Evaluate ../src/espam/examples/CSDF/json/tinyCSDF.json CSDF model in terms of power/performance with default time and energy specification.

Expected output:

{ "id": 0, "execution_time": 18.0, "energy": 6.4387499999999995E-9, "memory": 96.0, "processors": 3 }

Example 3: Time and energy specifications

$ ./espam --evaluate ../src/espam/examples/DNN/json/lenet_NB.json -nb --time-spec \
../src/espam/examples/time_spec/wcet.json --energy-spec \
../src/espam/examples/energy_spec/wcenergy.json

Description:
Evaluate ../src/espam/examples/DNN/json/lenet_NB.json DNN model in terms of power/performance in -nb mode with time specification given in wcet.json file and energy specification given in wcenergy.json file.
Expected output:
{ "id": 0, "execution_time": 359499.0, "energy": 42.371048349077284, "memory": 787744.0, "processors": 14 }

Example 4: DNN in block-based mode

$ ./espam --evaluate ../src/espam/examples/DNN/onnx/mnist.onnx --sesame -bb -100 --split-step 4

Description:
Evaluate ./src/espam/examples/DNN/onnx/mnist.onnx DNN model in terms of power/performance. Split model until it reaches <=100 blocks with –split-step = 4 (For more information about layers splitting see Cooments, block-based models) with default time and energy specifications.
Expected output:
{ "id": 0, "execution_time": 9932160.0, "energy": 4.460457589285961E-9, "memory": 83424.0, "processors": 2 }

Consistency checkout examples

Example 1: DNN model

$ ./espam --consistency ../src/espam/examples/DNN/onnx/mnist.onnx

Description: Check if ../src/espam/examples/DNN/onnx/mnist.onnx model is consistent.

Expected output:
input dnn consistency: true

Example 2: CSDF model

$ ./espam --consistency ../src/espam/examples/CSDF/json/tinyCSDF.json --in-csdf

Description: Check if ../src/espam/examples/CSDF/json/tinyCSDF.json model is consistent.

Expected output:
F0 unknown execution time. Default time = 1 is set for F0
F1 unknown execution time. Default time = 1 is set for F1
F2 unknown execution time. Default time = 1 is set for F2
input csdf graph consistency: true

References

1. EspamAI https://git.liacs.nl/lerc/espam/tree/espamAI
2. DARTS tool http://daedalus.liacs.nl/daedalus-rt.html
3. Energy model Di Liu, Jelena Spasic, Gang Chen, and Todor Stefanov,
Energy-Efficient Mapping of Real-Time Streaming Applications on Cluster Heterogeneous MPSoCs", In Proc. "13th Int. IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia'15), pp. 1-10, Amsterdam, The Netherlands, Oct. 8-9, 2015.
Electronic version is available on http://liacs.leidenuniv.nl/~stefanovtp/publications.html
4. Sesame tool: http://sesamesim.sourceforge.net/
5. ONNX DNN format https://onnx.ai/