software framework for runtime-Adaptive and secure deep Learning On Heterogeneous Architectures

Project Number 780788

Project Acronym ALOHA

D3.1	Report on automated application partitioning and mapping		
Work Package:	WP3	**Lead Beneficiary:**	IBM
Type:	Report	**Dissemination level:**	Public
Due Date:	30th September 2018	**Delivery:**	30th September 2018
Version:	D3.1_Report on automated application partitioning and mapping_v0.1.doc		

Brief description:

The purpose of this deliverable is to report on the research and development activities converging in the ALOHA automated tool for application partitioning and mapping. This report serves also as a lightweight form of documentation for deliverable D3.2 due at M18.

Deliverable Author(s):

Name	Beneficiary
Michael Masin	IBM

Deliverable Revision History:

Reviewer Beneficiary	Issue Date	Version	Comments
IBM	03/08/2018	v0.1	Released of the Table of Content and introduction
IBM	26/09/2018	v0.6	Full draft
IBM	30/09/2018	v0.7	Incorporated ST-I and ETHZ reviews
IBM	30/09/2018	V1.0	Final version

Disclaimer

The ALOHA Consortium is the following:

#	Participant Legal Name	Acronym	Country
1	STMICROELECTRONICS SRL	ST-I	Italy
2	UNIVERSITA' DEGLI STUDI DI CAGLIARI	UNICA	Italy
3	UNIVERSITEIT VAN AMSTERDAM	UVA	Netherlands
4	UNIVERSITEIT LEIDEN	UL	Netherlands
5	EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH	ETHZ	Switzerland
6	UNIVERSITA' DEGLI STUDI DI SASSARI	UNISS	Italy
7	PKE ELECTRONICS AG	PKE	Austria
8	CA TECHNOLOGIES DEVELOPMENT SPAIN SA	CA	Spain
9	SOFTWARE COMPETENCE CENTER HAGENBERG GMBH	SCCH	Austria
10	SANTER REPLY SPA	REPLY	Italy
11	IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD	IBM	Israel
12	SYSTHMATA YPOLOGISTIKIS ORASHS IRIDA LABS AE	IL	Greece
13	PLURIBUS ONE SRL	P-ONE	Italy
14	MAXQ ARTIFICIAL INTELLIGENCE, LTD (formerly MEDYMATCH TECHNOLOGY, LTD)	MaxQ-AI (formerly MM)	Israel

Table of Contents

Figures

1 Executive Summary

The main objective of this deliverable is to describe the progress of the activities carried out within Tasks 3.1, 3.2, 3.3 and 3.4 from D1.1 report at M6 to M9. This report presents a lightweight specification of the preliminary version of the partitioning and mapping tool.

The deliverable is organized as follows. In Section 2, we extend the information already exposed in deliverable D1.1 about the WP3 components (see section 3.2 of D1.1), by presenting the progress made from M6 to M9. In Section 3, we describe the preliminary activities and our plan for an integration the partitioning and mapping tool into the complete ALOHA toolflow.

1.1 Acronyms and abbreviations

Acronym	**Meaning**
DSE	Design Space Exploration
AOW	Architecture Optimization Workbench
SDF	Synchronous Data Flow
CSDF	Cyclo-Static Data Flow
PiSDF	Parametric & Interfaced Synchronous Data Flow

2 Automation of the algorithm partitioning and mapping

In this section, we'll describe the main steps needed for automating the partitioning and mapping of the algorithm configuration on the available processing cores. We'll extend the information already exposed in deliverable D1.1 by presenting the progress made from M6 to M9.

2.1 System-level DSE engine

2.1.1 Representation of Static DNNs

DNN model internal representation and algorithm partitioning

An internal DNN model representation is extracted from a specification of the input DNN provided in the ".onnx" exchange format. This step is implemented in a tool called ".onnx model reader". The ".onnx" exchange format represents deep neural networks as graphic models in which one graph node corresponds to one DNN layer. Hereinafter this graph representation is referred as layer-based representation. The layer-based representation is compact and easy to understand. Also, it is used by many of the state-of-the-art neural network frameworks. However, it does not reflect the potential for parallel and distributed processing of neurons within one layer, while this potential is vital for deploying deep neural networks on multi-processor embedded systems.

An alternative natural representation of a DNN model, hereinafter called neuron-based representation, associates one neuron of a deep neural network with one graph node (for more details please refer to deliverable D2.1). It exploits the full model parallelism, but suffers from the high complexity of its graph. For example, the AlexNet[1] model, provided by the .onnx models zoo[2], consists of more than 4000 nodes and more than 740000 edges if neuron-based representation is used. The graph size makes the analysis and design space exploration a slow and complicated process. Therefore, an internal block-based DNN representation was introduced. It uses the recursiveness property of the graph structure, which is also characteristic of the DNN graph models. One block, associated with one node of the DNN diagram model, is a generalized abstraction of a neuron-based DNN model subgraph. Unlike the neuron-based model, the block-based model allows storing the network description in a compact form, avoiding copying of common data for identical neurons of one block. In contrast to the layer-based model, it treats neurons as separate functional units.

All three representations, mentioned above, are shown in Figure 1. It should be noted that the neuron-based and layer-based representations could be easily modelled in a block-based notation. In the first case, a block corresponds to a DNN neuron and in the latter case, a block corresponds to a DNN layer. Layer-based and neuron-based models in the block notation are provided by the CNN-to-CSDF converter tool (see below) as possible initial models.

To control the degree of parallelism of the model, several transformations over DNN models are planned. The transformations include the *split* and *merge* transformations. The *split* transformation is intended to increase the initial model parallelism. An example of the *split* transformation is shown on the left side of Figure 2. In the example, the *split* transformation is performed over the Convolutional Layer block and divides the block into two independent blocks. The input data is copied for every independent block. The output data stream coming from the two independent blocks is combined and synchronized by the concatenation node.

---

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton. ImageNet classification with deep Convolutional Neural Networks. *Advances in neural information processing systems*, 2012.

[2] https://github.com/onnx/models/tree/master/bvlc_alexnet

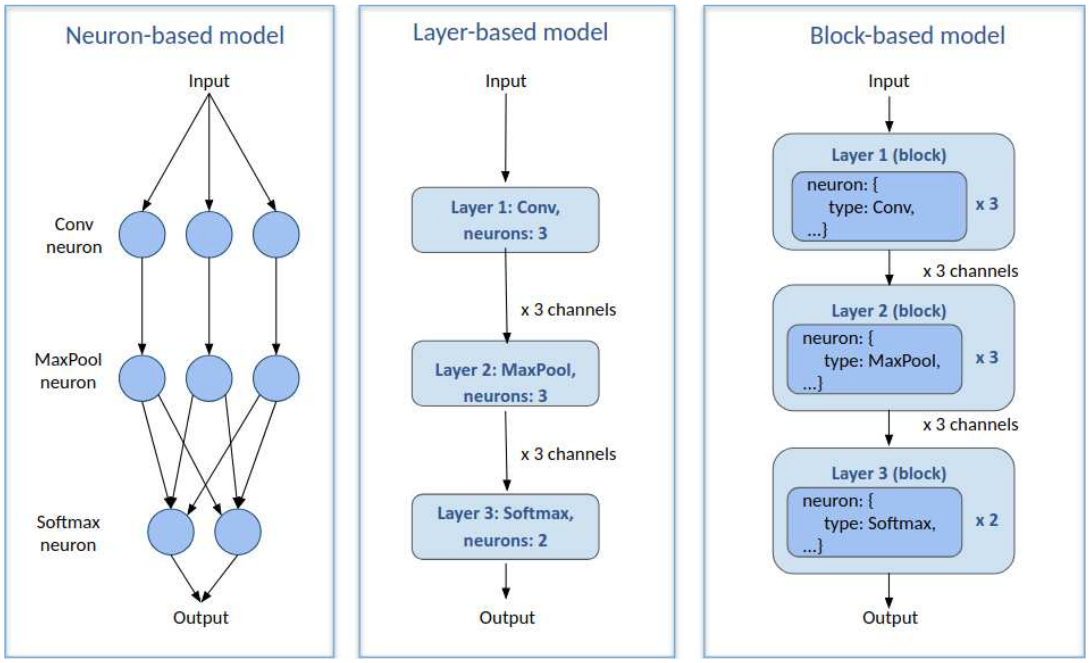

Figure 1: Deep neural network model representation.

The *merge* transformation is the opposite of the *split* transformation. It composes two or more DNN graph nodes/blocks into one block and reduces the model parallelism. An example of the merge transformation is shown on the right side of Figure 2, where the merge operator combines the Convolution and Pooling blocks into one block.

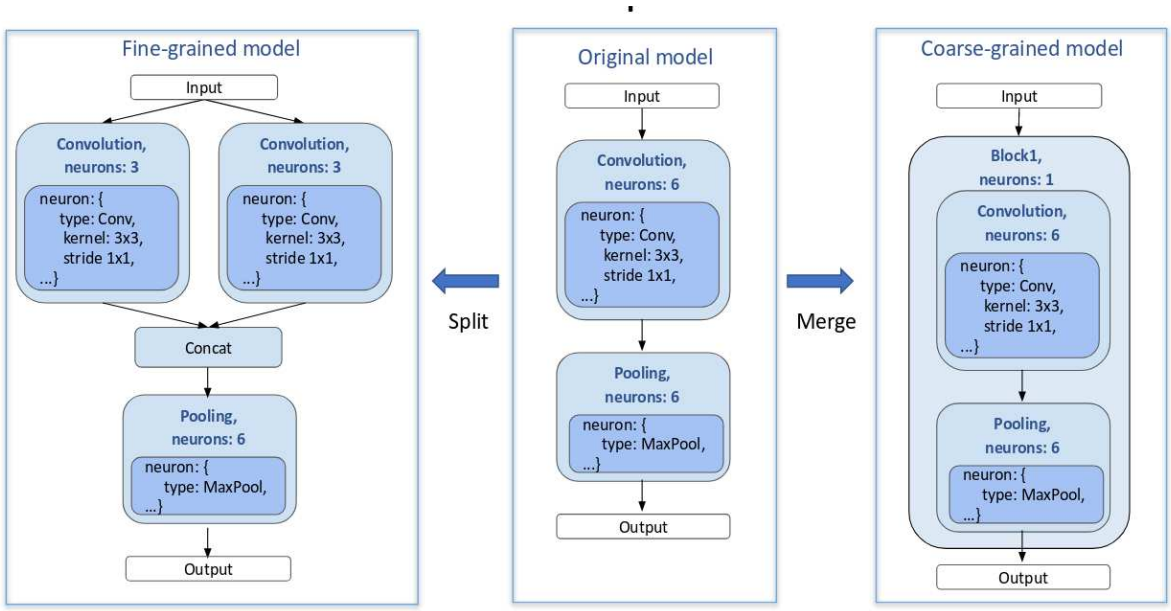

Figure 2: Transformations over the deep neural network model.

Conversion of a DNN model into a Data-Flow model

To ensure compatibility with the system-level DSE and model evaluation tools, developed in the ALOHA project, the internal deep neural network (DNN) model is converted to a functionally equivalent data-

flow model called Cyclo-Static Data Flow (CSDF)[3]. This conversion is implemented in a tool called "CNN-to-CSDF model converter" that generates a CSDF model (in ".json" format) as a graph of concurrent tasks communicating data via FIFOs. CSDF is a well-known model of computation, widely used for model-based design in the embedded systems community. More specifically, it is very suitable for timing, power/energy, and resource usage analysis, optimization, and evaluation. In addition, there exist several open-source tools automating such analysis/evaluation/optimization. An example of the model conversion is shown in Figure 3.

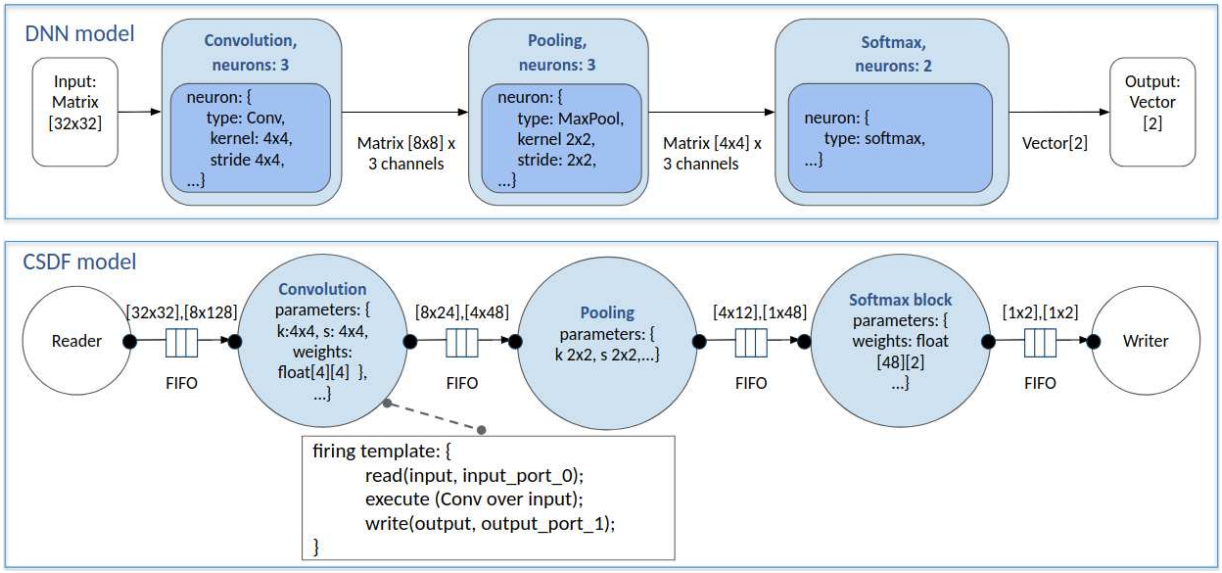

Figure 3: DNN-to-CSDF model conversion.

It should be mentioned that initially the Synchronous Dataflow (SDF)[4] model was used to represent a deep neural network as a dataflow model. However, our research has shown that the CSDF model was more compact for modeling of data overlapping/reuse, which may occur in DNN blocks performing convolutional or pooling operations. The CSDF model is a generalization of the SDF model, where the CSDF nodes may have cyclically changing firing rules. According to these rules, the amount of data consumed and produced by a CSDF node may vary from firing to firing in a cyclic pattern. Firings with different amount of produced/consumed data are called phases. For the cases where the amount of produced and consumed data is constant, a CSDF model becomes a SDF model.

An example of data processing with overlapping (i.e., processing with data reuse) is shown in Figure 4, showcasing the benefits of the usage of the CSDF model. The node with no overlapping corresponds to the Convolution block in Figure 3. The node with overlapping shows the same node with stride, smaller than the kernel size, which causes the data overlapping/reuse. As can be seen from the figure, the amount of input data required to fire the node for all phases, starting from the second phase, is smaller than the amount of data required for the first phase. This is due to the reuse of overlapping data at all phases except the first one. Since the CSDF model does not allow internal state specification in a node, the data overlapping/reuse is modeled as a self-loop FIFO channel, storing the overlapping data (data to be reused) by the node. When the model is deployed on a HW/SW platform, the self-loop channel is implemented inside the node thereby avoiding data transfer/reuse via external self-loop FIFO channels.

[3] G. Bilsen, M. Engels, R. Lauwereins. Cycle-static data flow. *IEEE Transactions on Signal Processing*, 1996.
[4] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. On computers*, 1987.

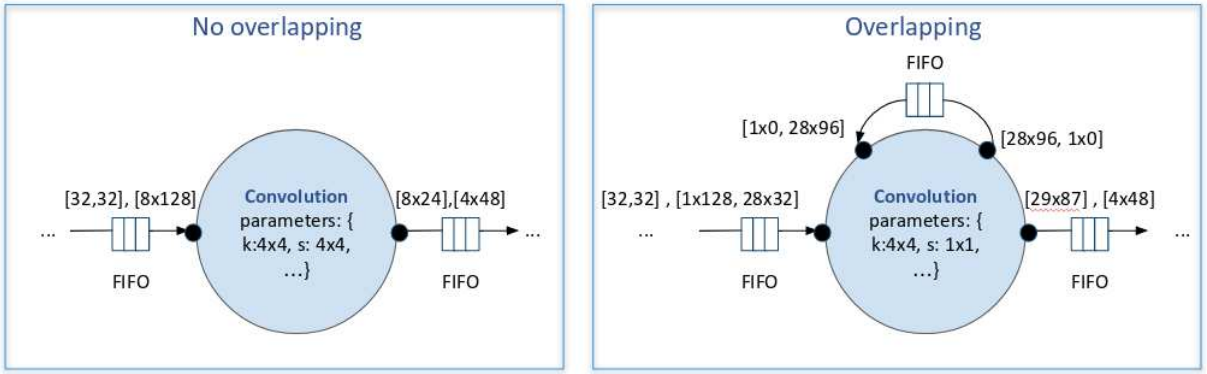

Figure 4: Processing with overlapping/reuse of data.

In addition, we consider using Parameterized & Interfaced Synchronous Dataflow (PiSDF)[5] to represent the whole design space of different levels of model level parallelism of the DNN inference application. With specific parameters, PiSDF models can be reduced to equivalent CSDF or SDF models.

2.1.2 Representation of Dynamic DNNs

The concept of post-training Parsimonious Inference (PI) is to simultaneously adapt kernel coefficients and learn kernel activation rules during a brief training phase following the regular training procedure. For this purpose, each convolutional layer chosen to be subjected to conditional execution has to be modified in a way that enables the transformation from a typical convolutional layer to a layer with dynamic kernel population. The simplest but still effective type of rules can be formulated as simple linear rules on the channel-wise average pooling of a layer's input data tensor. This type of activation rules offers a dual advantage. First, it induces a very small computational overhead, (typically a negligible fraction of the overall FLOPs count) to the overall workload. Secondly, rules can be formulated using standard components of a typical CNNs design, such as pooling layers and 1x1 convolutional kernels, thus can be easily described by industry standards like ONNX or Caffe protobuf format without any modifications needed.

In this context, a convolutional DNN model converted to a model with dynamically executed kernels, has the typical description of the corresponding static DNN, with the addition of a number of 1x1 convolutional layers whose outputs indicate whether the corresponding kernels have to be computed. An example on this type of representation can be seen in Figure 5. For the layers that produce the rules, a simple naming convention like the one illustrated, can be easily used in order to automatically identify and link the rules to the respective kernels. In this setting, the sign of each output of Conv3_2_SW layer indicated whether the corresponding kernel (kernel with the same index) of Conv3_2 layer has to be computed.

[5] K. Desnos, J. Heulot. PiSDF: Parameterized & Interfaced Synchronous Dataflow for MPSoCs Runtime Reconfiguration. In: *METODO* 2014.
Desnos, K., Pelcat, M., Nezan, J.F., Bhattacharyya, S.S., Aridhi, S.: Pimm: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration. In: *SAMOS* XIII 2013.

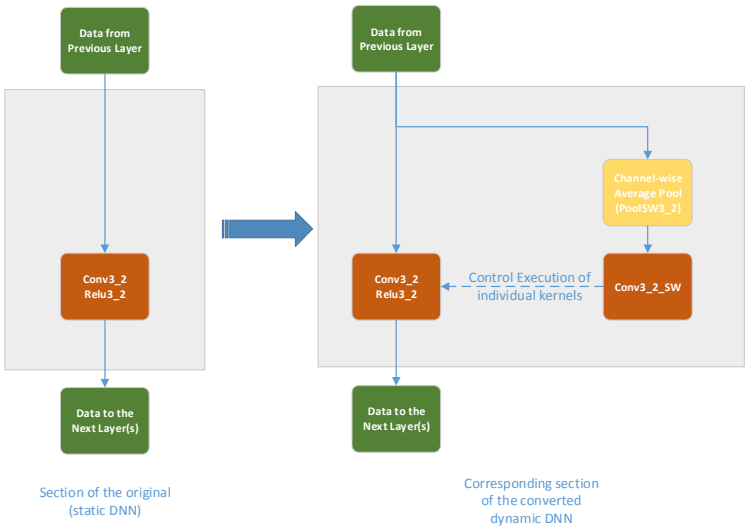

Figure 5. Example of the representation of a DNN with dynamically executed kernels, in the context of post-trainin PI

2.1.3 Representation of Communication and Computation Platform

In D1.1, the ALOHA architecture description format was described. The system-level DSE engine will deploy this format to parse target architectural platform descriptions. From this, different proprietary internal platform descriptions may be generated for the various satellite tools (e.g., for the Sesame system-level simulation framework and IBM's AOW). At this moment, no work has been performed on this part of the tool-flow yet.

2.2 Post-training algorithm refinement for parsimonious inference

The main activities from M6 to M9 in this topic where concentrated along the automation of the Parsimonious Inference (PI) pruning technique:

1. **The utility modification of a DNN** by the addition of appropriate modules able to activate/deactivate a number of kernels. These modules need to be placed to specific convolutional layers, based on empirical rules, and an automation process needs to be developed to mechanize these rules based on an analysis of the model description. To this end, in the time frame under consideration we made a first attempt towards this goal, getting very encouraging results.

The model modification process involves of course the reading, modification and writing of the model description. This also poses the necessity of being compliant with the ALOHAs preferred model description format which decided to be ONNX. Thus, a special module has been developed for reading and modifying this format.

2. **Setting a few additional hyperparameters**, defining the aggressiveness of the PI pruning process, and used by the subsequent post-training process. These hyperparameters have been so far also defined empirically. Although this is not an uncommon thing in DL, we think that a somehow automated determination of the hyperparameter values need to exist, based on some aspects of the model after the initial training processes. To this end, in the time frame under consideration, we made an initial investigation of some possible solutions.

2.3 Sesame-based DSE

The main activities from M6 to M9 concentrated on modelling DNNs in Sesame such that they can be simulated in order to perform mapping DSE. To this end, work is currently ongoing to translate DNNs

into application models based on the Cyclo-Static DataFlow (CSDF) model of computation (see Section 2.1.1). These CSDF-based DNNs can be readily simulated inside the Sesame framework, which would then already allow for relatively simple mapping DSE experiments. We expect to have such an experiment ready by M12 of the project, performing DSE (by means of a genetic algorithm) for a small CSDF-based DNN to be mapped on a bus-based multiprocessor platform containing 2-4 processing elements. The model conversion is performed in four main steps:

1. *CSDF topology generation*: Each block of the neural network model is represented as a node in the CSDF graph. Each connection between neural network blocks is represented as a FIFO communication channel in the CSDF graph. The communication channel is linked to the data producer node via an output port and to the data consumer node via an input port.

2. *Data production/consumption rates calculation*: In this step, the amount of data that is produced/consumed by repetitive firings of a CSDF node to/from an output/input FIFO channel is determined. This calculation is based on the specific properties, characteristics, and parameters of the corresponding block in the DNN model.

3. *CSDF model schedule derivation:* It determines the order and amount of firings for each of the nodes in a CSDF model. At the moment, the DARTS[6] open-source tool is used to derive real-time schedules for CSDF models obtained from DNN models.

4. *Application model generation for Sesame:* It provides an application model, used by the Sesame modeling and simulation environment, for system-level design space exploration.

The application model, used by the Sesame modeling and simulation environment for DSE, is automatically generated by a part of the DNN-to-CSDF convertor tool called "Sesame template generator". The application model is provided as a number of code templates in C++ and ".yml" format. These templates describe a CSDF graph model using coarse-grained reading, writing, and execution events.

- *The reading event* describes the reading of input data to a dataflow node. Input data is coming from communication channels through the node input ports. After the required amount of data is read by the node, the execution event can start.

- *The execution event* describes the node functionality, inherited from the corresponding DNN block. It may contain simple operations such as convolution or pooling or have more complex internal logic.

- *The writing event* describes the recording of the node's firing results to the node output. The result is written to the communication channels through the node's output ports.

Besides of the node's firing description, the generated templates contain some meta-information related to the node such as constant parameters (e.g. weights) description. The example, provided in Figure 6, shows an automatically generated firing template of the pooling block node in Figure 3. For simplicity some internal operations are skipped.

[6] http://daedalus.liacs.nl/daedalus-rt.html

```
// File automatically generated by ESPAM

Layer2::Layer2(Id n, Layer2_Ports *ports) : Layer2_Base(n, ports) {}
Layer2::~Layer2() {}

void Layer2::main() {
  // repetition parameters definition
  int q = 4;
     ...

   while (1) {
    // loop over the repetitions number
    for (int rep = 0; rep < q ; rep ++) {
      phase = rep % phase_len;
       ...
      // read input data from the input port IP0
       for ( int t = 0; t < IP0_tokens; t++) {
        ports->IP0.read(input);
      }

      //simulate MaxPooling execution
      for (int i = 0; i < 4; i++) {
        execute("MAXPOOL(4)");
      }

      // write output data to output port OP0
       for ( int t = 0; t < OP0_tokens; t++) {
        ports->OP0.write(output);
      }
    }// loop over the phases
  } while (1)
} // main
```

Figure 6: Automatically generated code template for Sesame-based DSE.

Since the CSDF-based DNN application models can become large in terms of the number of components (neurons and channels), this may cause significant overhead in Sesame's modeling of mappings of these application models onto the underlying platform as well as in the DSE process itself. To study the scalability of Sesame in this respect, we are currently performing scalability experiments with the modeling, simulation and mapping exploration of large synthetic, fully-connected DNNs (thousands of components) inside Sesame. Preliminary results are promising and seem to indicate that realistic DNN application models can be simulated efficiently in Sesame (thus allowing efficient DSE).

2.4 Architecture Optimization Workbench

The main activities in AOW during M6 to M9 period were focused on modeling of the DNN inference application and system architecture for Mathematical Programming optimization. We consider throughput, latency, and energy objectives balancing potential memory, communication, and computation bottlenecks while utilizing application and intra-actor level parallelism in heterogeneous platforms. The first modeling task includes addition of shared and local memory and communication for efficient computation near data with low latency and energy consumption. The next step will be to define the design space for application modeling with different levels of application and intra-actor level parallelism using dense, sparse and mixed convolution calculations.

3 Integration activities

In deliverable D1.1 initial interfaces had been defined. Continuation of the work on tool's REST container, interfaces between Sesame and AOW, and testing of automatic code generator is planned after the face to face meeting organized by CA in M10 to understand Docker containers, auto code generators and their usage towards C based modules. The face to face meeting will be also used to evaluate integration strategies between the system level design tools.