This XML file does not appear to have any style information associated with it. The document tree is shown below.

− <root>
Every newly generated result will be appended after existed ones as a 'simulation' under the 'root'. New result file will be generated if there is no existed one.
− <simulation>
All the information of a design point is collected under a 'simulation'. User can set 'result_print_mode' in the setting file to tune how detail the result file is. This documentation will take the 'complete' version as an example.
<this_documentation_generated_time/>
2020-08-17 23:29:13
− <layer>
A summary of the neural network layer under-test.
<layer_spec/>
Layer dimension notation: 'B' for batch size, 'K' for output channel,'C' for input channel, 'OX/OY' for output X/Y dimension, 'IX/IY' for input X/Y dimension, 'FX/FY' for filter kernel X/Y dimension, 'SX/SY' for stride on X/Y dimension, 'SFX/SFY' for dilated convolution on X/Y dimension.
<total_MAC_operation/>
Total multiply-accumulate operation of the neural network layer.
<total_data_size_element/>
Total number of data element of weight ('W'), input ('I'), and output ('O') in the neural network layer.
<total_data_reuse/>
Total data reuse factor for each element in W, I, and O. Note that for I, it is an average value, which can be non-integer.
</layer>
− <search_engine>
A summary of user-defined settings and constraints for three search engines, and number of valid design points found.The three search engines are Memory Hierarchy Search Engine, Spatial Unrolling Search Engine, and Temporal Mapping Search Engine, a.k.a. Auto-Scheduler.
− <mem_hierarchy_search>
In the memory hierarchy search, user can play with memory level/size /bank/type/area/sharing and so on.
<mode/>
Constraint-driven exhaustive search is the only option for memory search now. More advanced search methods are under development.
<area_constraint/>

Total available area.
<area_utilization_threshold/>
More than certain percent of the total area should be filled in with memory hierarchy.
<consecutive_memory_level_size_ratio/>
More than certain time difference in memory size is required for memory modules at consecutive levels in the hierarchy.
<max_inner_PE_mem_size_bit/>
The upper bound for memory/register/register file size inside of each PE.
<max_inner_PE_mem_level/>
The maximum memory level inside of each PE, a.k.a. memory unrolling level.
<max_outer_PE_mem_level/>
The maximum memory level out of the PE array.
<mem_scheme_index/>
Current memory hierarchy index / total valid memory hierarchy count.
</mem_hierarchy_search>
− **<spatial_mapping_search>**
Spatial mapping, a.k.a spatial unrolling, defines what neural network layer dimensions are unrolled onto the 2D PE array. Such loop unrolling can be 2D (e.g. unroll C|K, C on one array dimension and K on another array dimension) or more than 2D (e.g. unroll OY|(FY|K), OY on one array dimension and FY & K on another array dimension).
<mode/>
Four search methods are supported in the current version: exhaustive / heuristic_v1 / heuristic_v2 / hint_driven.
<PE_array_spatial_utilization_threshold/>
More than certain percent of the array should be covered with valid loop unrolling schemes. It is useful for the first three search methods.
<spatial_mapping_hint_list/>
Let the tool only try out the spatial unrolling options given by user. It is useful for the last search method. Data format example: [{'Col': ['OY']}, {'Row': ['FY','K']}]] indicates unrolling OY on one array dimension and unrolling FY & K on another array dimension.
<unrolling_scheme_index/>
Current spatial unrolling index / total valid spatial unrolling count.
</spatial_mapping_search>
− **<temporal_mapping_search>**
Temporal mapping, a.k.a scheduling, defines the loop blocking and ordering of the neural network layer dimensions upon the memory hierarchy.
<mode/>
Four search methods are supported in the current version: exhaustive / heuristic_v1 / heuristic_v2 / iterative.

<memory_utilization_hint/>
More than certain percent of the memory size is suggested to be filled in with data during the loop blocking stage. Note that this threshold is not a firm constraint.
<valid_temporal_mapping_found/>
Total valid temporal mapping count.
</temporal_mapping_search>
</search_engine>
− <hw_spec>
Hardware specification includes two parts: 'PE array' and 'memory hierarchy'.
− <PE_array>
<precision_bit/>
The data precision in bit used in logic computing and data storing for each operand (W, I, O). Note that for O, the partial sum (partial_O) and the final sum (fianl_O) could have different data precision requirement.
<array_size/>
2D dimensional PE array size (column and row).
</PE_array>
− <memory_hierarchy>
<mem_name_in_the_hierarchy/>
The overall memory hierarchy is represented as three operands' memory hierarchy. Data format example: <W/>['rf512', 'sram128kb', 'sram16Mb'] indicates that W has three memory levels to store data, from the innermost level (closest to MAC logic) to the outermost level are 'rf512', 'sram128kb', and 'sram16Mb', whose name are defined by user in the memory_pool file. Same rule is applied to operands I and O.
<mem_size_bit/>
Each memory module's size in bit of the memory hierarchy.
<mem_bw_bit_per_cycle_or_mem_wordlength/>
Memory bandwidth defines how many bit of each memory module can be accessed per cycle. It is used interchangeably with memory wordlength in the tool. Data format example: 'W': [[32.0, 32.0], [128.0, 128.0], [128, 128]] indicates the memory read and write bandwidth / wordlength at three W memory levels, from the innermost level (closest to MAC logic) to the outermost level.
<mem_access_energy_per_word/>
The energy of accessing one word at each memory level. Data format example: 'W': [[1.858, 2.573], [38.176, 41.952], [240, 240]] indicates the memory read and write cost per word for three W memory levels, from the innermost level (closest to MAC logic) to the outermost level.
<mem_type/>
Three types of memory are currently supported in the tool: single-port double-buffered (sp_db), dual-port single-buffered (dp_sb), and dual-port

double-buffered (dp_db). Data format example: W': ['dp_sb', 'dp_sb', 'dp_sb'] indicates the memory type for each W memory level.
<mem_share/>
It defines which operands' which memory levels are physically one memory module. Data format example: {1: [('O', 'sram16Mb'), ('I', 'sram16Mb'), ('W', 'sram16Mb')]} indicates in the memory hierarchy, memory module 'sram16Mb' stores all three operands W, I, O. The memory module's name ('sram16Mb') is corresponding to the name listed in 'mem_name_in_the_hierarchy'.
<mem_area_single_module/>
Memory area at each memory level for each operand.
<mem_unroll/>
Memory unrolling factor at each memory level for each operand
</memory_hierarchy>
</hw_spec>
− **<results>**
Results including four parts: basic information, energy, performance, and area.
− **<basic_info>**
<spatial_unrolling/>
It is ordered from left to right the MAC level, the innermost memory level, and all the way up to the outermost memory level (the largest memory in the system). Note that it is one level more than the pure memory hierarchy. Data format example: <W/>[[], [[('OY', 13)], [('FY', 3), ('C', 4)]], [], []] indicates that for operand W, 'OY', 'FY', and 'C' are unrolled 13, 3, and 4 times at the innermost memory level, in which 'OY' is unrolled along one array dimension and 'FY' & 'C' are unrolled along another array dimension.
<temporal_mapping/>
It is ordered from left to right the innermost memory level, and all the way up to the outermost memory level. Data format example: <I/>[[('OX', 13), ('K', 4), ('C', 2), ('FX', 3), ('K', 6)], [('C', 12), ('K', 16)], [('C', 2)]] indicates that for operand I, loops ('OX', 13), ('K', 4), ('C', 2), ('FX', 3), ('K', 6) are scheduled at the innermost memory level with ('OX', 13) as the innermost one; loops ('C', 12), ('K', 16) are scheduled at one memory level above; loop ('C', 2) is scheduled at the outermost memory level. In the case that no loop is scheduled at a certain memory level, an empty '[]' is shown.
<data_reuse/>
Data reuse factor for each operand at each level, starting from the MAC level.
<I_pr_diagonally_broadcast_or_fifo_effect/>
For operand I, this parameter indicates the partially-relevant (pr) loop pair's special data reuse opportunity shows up at which level, starting

from the MAC level. User can get more details about pr loop for operand I in our paper.

<used_mem_size_bit/>

Actual data size in bit at each level, starting from the innermost memory level. It is determined by the spatial unrolling and the temporal mapping.

<actual_mem_utilization_individual/>

Memory size utilization at each level without considering memory sharing between operands, starting from the innermost memory level.

<actual_mem_utilization_shared/>

Memory size utilization at each level with considering memory sharing between operands, starting from the innermost memory level.

<effective_mem_size_bit/>

The minimum required memory size for supporting current mapping scheme based on data lifetime analysis.

<total_unit_count/>

The total number of unit at each level, starting from the MAC level.

<unique_unit_count/>

The number of unit that process/hold unique data at each level, starting from the MAC level.

<duplicate_unit_count/>

The number of unit that process/hold duplicate data at each level, starting from the MAC level. Note that due to operand I's pr special data reuse opportunity, the value can be non-integer sometimes.

<mem_access_count_elem_for_I_and_W/>

Number of memory access for each operand at each memory level, starting for the innermost memory level. Data format example: <I/>[[112140288, 691200], [691200, 43200], [43200, 0]] indicates that for operand I, with the current mapping scheme, 112140288 reading access and 691200 writing access happen at the innermost memory level; 691200 reading access and 43200 writing access happen at one memory level above; 43200 reading access and 0 writing access happen at the outermost memory level. Note that the memory access from peripherals writing to the outermost memory level is not considered.

<mem_access_count_elem_for_O/>

For operand O, situation is more complicated due to the existence of partial sum and final sum. Partial sum flows bi-directionally across levels while final sum flows uni-directionally from the innermost memory level to the outermost one. Plus, in most cases, partial sum and final sum have different data precision requirement. Thus, in the tool, partial sum (O_partial) and final sum (O_final) are separately considered. Data format: each '[]' under of the top '[]' indicates one memory level. Inside of each level, [(a , b) , (c , d)] captures all memory accesses, in which 'a' is number of memory read out to the level below (the level below can be

MAC level or another memory level); 'b' is number of memory write in from the level below; 'c' is number of memory read out to the level above; 'd' is number of memory write in from the level above. Thus, the total memory read access is 'a + c' and the total memory write access is 'b + d'.

</**basic_info**>

− <**energy**>

<**total_energy**/>

Total energy currently includes two parts: 'memory access energy' and 'mac energy'.

<**mem_energy_breakdown**/>

Energy spent for each operand at each level, starting from the innermost memory level.

<**mac_energy**/>

</**energy**>

− <**performance**>

Performance includes two parts: 'mac array utilization' and 'latency'.

− <**mac_array_utilization**>

MAC array utilization can drop due to spatial and temporal reasons.

<**utilization_with_data_loading**/>

The overall MAC array utilization with considering the initial stage data loading overhead.

<**utilization_without_data_loading**/>

The overall MAC array utilization without considering the initial stage data loading.

<**utilization_spatial**/>

MAC array utilization is spatially dropped due to the spatial unrolling scheme doesn't completely match the array size. This value shows the MAC array utilization spatial drop.

<**utilization_temporal_with_data_loading**/>

MAC array utilization is temporally dropped due to memory bandwidth bottleneck. This value shows the MAC array utilization temporal drop, with considering the initial stage data loading overhead.

<**mac_utilize_temporal_without_data_loading**/>

This value shows the MAC array utilization temporal drop, without considering the initial stage data loading.

</**mac_array_utilization**>

− <**latency**>

<**latency_cycle_with_data_loading**/>

The total latency cycles with considering the initial stage data loading.

<**latency_cycle_without_data_loading**/>

The total latency cycles without considering the initial stage data loading.

<**ideal_computing_cycle**/>

The ideal computing cycles, assuming the MAC array utilization is 100%.

− <**data_loading**>

<**load_cycle_total**/>

Total data loading cycles in the initial stage.

<**load_cycle_individual**/>

Data loading cycles for individual memory level, starting from the innermost memory level.

<**load_cycle_combined**/>

Data loading cycles for operand W and I.

</**data_loading**>

− <**mem_stalling**>

<**mem_stall_cycle_total**/>

Total stalling cycles due to memory bandwidth bottleneck during computation.

<**mem_stall_cycle_individual**/>

Memory stalling cycles in between of different levels (without considering memory sharing between operands), starting from the data transfer link between MAC logic and the innermost memory level, to the link between the second to last memory level and the last (outermost) memory level. Positive value indicates stalled cycles. Negative value indicate the data transfer speed is more than enough. Zero indicates the data transfer speed just meets the computation requirement, thus no stall.

<**mem_stall_cycle_shared**/>

Memory stalling cycles in between of different levels (with considering memory sharing between operands), starting from the data transfer link between MAC logic and the innermost memory level, to the link between the second to last memory level and the last (outermost) memory level.

<**req_mem_bw_bit_per_cycle_individual**/>

Memory bandwidth (bit/cycle) requirement for preventing computing stall happening (without considering memory sharing between operands). Data format: at each memory level [a , b], 'a' is the memory read bandwidth requirement; 'b' is the memory write bandwidth requirement.

<**req_mem_bw_bit_per_cycle_shared**/>

Memory bandwidth (bit/cycle) requirement for preventing computing stall happening (with considering memory sharing between operands).

<**mem_bw_requirement_meet**/>

Check for each operand at each memory level if the memory bandwidth meets the no-computing-stall requirement. User can adjust the memory bandwidth accordingly to improve system's

```
            performance.
          </mem_stalling>
        </latency>
      </performance>
      <area/>
      Total memory hierarchy area.
    </results>
  </simulation>
</root>
```