Cape Tool Chain Components

# Components and Input/Output Type



Each box in the diagram represents a component which can be a tool (e.g. Oneview) or scripts (CapeScripts, summarize.py, parse.js, etc). The boxes are in three colors representing their implementation status:

* Blue boxes represent components which are working now already.
* Orange boxes represent components which does not exist yet and to be created.
* Yellow boxes represent components which exists currently but need updates to get it working.

The input and output of the components are the labels on the edges connecting the components. There are four types, which can be seen from the file extension (or the lack of it)

1. Excel file (OneviewFormat.xlsx) is generated by Oneview and is used by summarize.py as inputs.
2. CSV files are generated by various scripts (e.g. SummaryFormat.csv) and can be used by other scripts as inputs (e.g. SummaryFormat.csv is used by generate\_SI.py).
3. PNG files (e.g. SI-plot.png) are generated by plotting scripts as static plot good for document uses.
4. Data visualization is the output of GUI tool developed by Tony with input data generated by other scripts (e.g. qplot-data.csv).

# Input File Formats

In this section we describe the format of files used by the scripts. Here only the scripts in blue (working scripts) are described.

## Input format of summarize.py

Script summary.py accepts input from either “QPROF\_full” tab of an Excel file or a plain CSV file. In both cases, the data is assumed to be a table with first row being header containing column names and starting from second row, each row will be data for each codelet. Following are names of the columns expected grouped by the type:

1. Codelet names.
   * Name of the application containing the codelet: application.name,
   * Name of the codelet: codelet.name,
   * Name of the variant of the codelet generated by DECAN tool: decan\_variant.name.
2. System information
   * CPU current frequency: decan\_experimental\_configuration.frequency
   * CPU advertised frequency: cpu.nominal\_frequency
   * CPU type: cpu.generation
   * Number of cores for run: decan\_experimental\_configuration.num\_core
3. Timing and Iterations
   * Number of times the codelet is executed repeatedly. This column only presents in CapeScrpts outputs but not Oneview: Repetitions. For Oneview data, summarize.py assumes Repetition=1.
   * Number of iterations executed: Iterations. Note that this number is number of iterations for all repetitions, so to get number of iterations for one execution of codelet, summarize.py will divide this number by Repetitions. For all the following metrics, they are normalized to be value per iteration.
   * Number of reference cycles of the codelet: CPU\_CLK\_UNHALTED\_REF\_TSC
   * Number of core cycles of the codelet: CPU\_CLK\_UNHALTED\_THREAD
4. Instruction Count
   * Floating point instruction counts for , and 
     + Scalar counts: Nb\_FP\_insn\_S
     + Vector counts: Nb\_FP\_insn\_P\_
   * Integer instruction counts for , ,
     + Scalar counts: Nb\_scalar\_INT\_arith\_insn\_ and Nb\_scalar\_INT\_logic\_insn\_
     + Vector counts for , , : Nb\_INT\_arith\_insn\_\_ and Nb\_INT\_logic\_insn\_\_
   * Load/Store instruction counts:

Nb\_8\_bits\_loads, Nb\_16\_bits\_loads, Nb\_32\_bits\_loads, Nb\_64\_bits\_loads, Nb\_128\_bits\_loads, Nb\_256\_bits\_loads, Nb\_MOVH\_LPS\_D\_loads,

Nb\_8\_bits\_stores, Nb\_16\_bits\_stores, Nb\_32\_bits\_stores, Nb\_64\_bits\_stores, Nb\_128\_bits\_stores, Nb\_256\_bits\_stores, Nb\_MOVH\_LPS\_D\_stores

1. Cache traffic/bytes Count
   * The expected columns are CPU type dependent. The CPU type is obtained from cpu.generation.
     + Skylake: L1D\_REPLACEMENT, L2\_TRANS\_L1D\_WB, L2\_RQSTS\_MISS, L2\_TRANS\_L2\_WB
     + Haswell: L1D\_REPLACEMENT, L2\_TRANS\_L1D\_WB, L2\_DEMAND\_RQST\_WB\_MISS, L2\_RQSTS\_MISS, SQ\_MISC\_FILL\_DROPPED, L2\_TRANS\_L2\_WB, L2\_DEMAND\_RQSTS\_WB\_MISS
     + Sandy Bridge and Ivy Bridge: L1D\_REPLACEMENT, L1D\_WB\_RQST\_ALL, L2\_LINES\_ALL, SQ\_MISC\_FILL\_DROPPED, L2\_TRANS\_L2\_WB, L2\_L1D\_WB\_RQSTS\_MISS
   * CPU independent columns:
     + DRAM traffic : UNC\_M\_CAS\_COUNT\_RD, UNC\_M\_CAS\_COUNT\_WR, UNC\_IMC\_DRAM\_DATA\_READS, UNC\_IMC\_DRAM\_DATA\_READS
     + L1 Bytes counts: Bytes\_loaded, Bytes\_stored
2. Register Access Byte Count: Bytes\_GP\_addr\_read, Bytes\_GP\_addr\_write, Bytes\_GP\_data\_read, Bytes\_GP\_data\_write, Bytes\_SIMD\_read, Bytes\_SIMD\_write
3. Energy Measurement
   * Package energy: UNC\_PKG\_ENERGY\_STATUS or FREERUN\_PKG\_ENERGY\_STATUS
   * DRAM energy: UNC\_DDR\_ENERGY\_STATUS or FREERUN\_DRAM\_ENERGY\_STATUS
4. CPU Stall cycles
   * These columns are CPU type dependent. The CPU type is obtained from cpu.generation.
     + Skylake: RESOURCE\_STALLS\_RS, RESOURCE\_STALLS\_LB, RESOURCE\_STALLS\_SB, RESOURCE\_STALLS\_ROB, RESOURCE\_STALLS2\_PHT\_FULL, RESOURCE\_STALLS\_LOAD\_, RESOURCE\_STALLS\_ANY, Front\_end\_(cycles)
     + Haswell: RESOURCE\_STALLS\_RS, RESOURCE\_STALLS\_LB, RESOURCE\_STALLS\_SB, RESOURCE\_STALLS\_ROB, RESOURCE\_STALLS2\_ALL\_PRF\_CONTROL, RESOURCE\_STALLS\_LOAD\_MATRIX, RESOURCE\_STALLS\_ANY, Front\_end\_(cycles)
     + Ivy Bridge: RESOURCE\_STALLS\_RS, RESOURCE\_STALLS\_LB, RESOURCE\_STALLS\_SB, RESOURCE\_STALLS\_ROB, RESOURCE\_STALLS2\_ALL\_PRF\_CONTROL, RESOURCE\_STALLS\_LOAD\_MATRIX, RESOURCE\_STALLS\_ANY, Front\_end\_(cycles)
     + Sandy Bridge: RESOURCE\_STALLS\_RS, RESOURCE\_STALLS\_LB, RESOURCE\_STALLS\_SB, RESOURCE\_STALLS\_ROB, RESOURCE\_STALLS2\_ALL\_PRF\_CONTROL, RESOURCE\_STALLS2\_LOAD\_MATRIX, RESOURCE\_STALLS\_ANY, Front\_end\_(cycles)

Note that summarize.py also accept a name shortening mapping file (in CSV format) to generate Short Name and/or Variant column as output. The CSV has three columns:

* The name of codelet to generate shortened names: name
* The corresponding shortened name: short\_name
* The variant name of that codelets. (This may be related to LORE codelets specific details): variant

## Input format of generate\_SI.py

Script generate\_SI.py accepts input from a plain CSV file generated by summarize.py. Data is assumed to be a table with first row being header containing column names and starting from second row, each row will be data for each codelet. The column name could be long human readable form as well as shorter naming form. Normally generate\_SI.py will call the same routine used by summarize.py to first convert the longer column name to shorter ones before further processing. Below the shorter column names are used to describe the expected columns grouped by type:

1. Codelet names.
   * Shortened Name of the codelet: short\_name
2. Data rates
   * Cache transfer rate: l1\_rate\_gb/s, l2\_rate\_gb/s, l3\_rate\_gb/s
3. Compute Rates
   * FLOP rate: flop\_rate\_gflop/s
4. Buffer Utilization
   * Percentage stall time: %frontend