ExaSAT XML Specification

Draft Version 0.3.1

Cy Chan
ExaCT Co-design Center
Lawrence Berkeley National Laboratory

April 24, 2013

This document details the XML file structure used by the ExaSAT (Exascale Static Analysis Tool). In typical usage (see Figure 1), the compiler analysis component reads a source code input to generate an XML file, which is then fed into the performance model component to estimate imporatant performance statistics (e.g. flops, memory traffic, working set, runtime). However, if one wants to estimate the performance of a hypothetical code formulation without writing the actual code, an XML representation of that code can be used in conjunction with the performance model component.

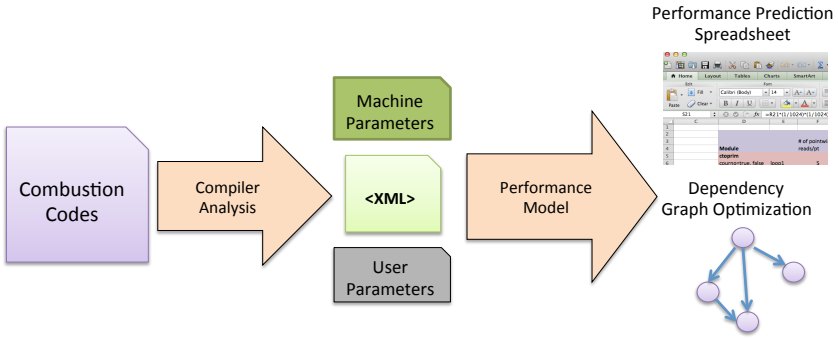

Figure 1: ExaSAT Tool Chain

0.1 XML Description

The XML contains a root `program` element, of which every other element in the file is a descendant.

This section details the attributes and sub-elements found in each element type. An element must have exactly one of each attribute listed for its element type (unless the attribute is designated *optional*) and may have zero or more sub-elements of each of the types listed for its element type. The elements are presented in the order of a depth-first traversal of the XML hierarchy as shown in Figure 2.

0.1.1 The `program` element

- Attributes: NONE
- Sub-Elements:

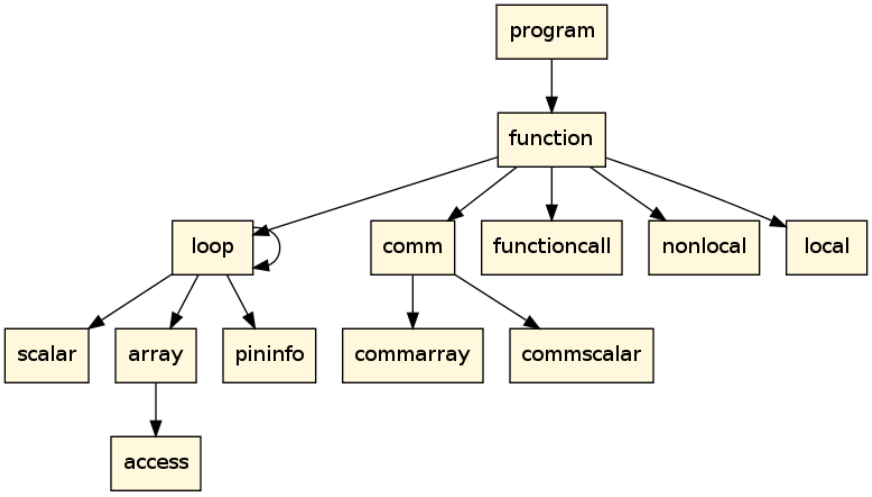

Figure 2: XML Element Hierarchy

  - `function` - a function that appears in the program

0.1.2 The `function` element

- Attributes:
  - `name` - the name of the function
- Sub-Elements:
  - `local` - a local (stack) variable that appears in the function
  - `nonlocal` - a non-local (e.g. global or argument) variable that appears in the function
  - `loop` - a loop that is executed within the function
  - `comm` - a communication that occurs within the function
  - `functioncall` - a function call that is made within the function

0.1.3 The `local` element

- Attributes:
  - `name` - the name of the local variable
- Sub-Elements: NONE

0.1.4 The `nonlocal` element

- Attributes:
  - `name` - the name of the non-local variable
- Sub-Elements: NONE

0.1.5 The `loop` element

This element is used to help estimate the work needed to execute the loop including floating point computation and memory traffic. Elements and attributes associated with a loop pertain to code within that particular loop level but not to code further nested in child loops.

- Attributes:
  - `linenum` - the line number where the loop begins
  - `loopvar` - the loop iteration variable
  - `stride` - the strides used in the loop
  - `lowerbound` - lower bound of loop
  - `upperbound` - upper bound of loop
  - `adds` - the number of floating point additions and subtractions in the loop (exclusive of nested child loops)
  - `multiplies` - the number of floating point multiplications in the loop (exclusive of nested child loops)
  - `divides` - the number of floating point divisions in the loop (exclusive of nested child loops)
  - `specials` - the number of special floating point operations (e.g. *sqrt*) in the loop (exclusive of nested child loops)
- Sub-Elements:
  - `loop` - a loop nested within this loop
  - `scalar` - a scalar integer or floating point variable that is used in the body of the loop (exclusive of nested child loops)
  - `array` - an integer or floating point array that is used in the body of the loop (exclusive of nested child loops)
  - `pininfo` - a special auxiliary element that is used to report performance counter measurements collected during an instrumented trial run

0.1.6 The `scalar` element

This element is used to help estimate the number of registers required to hold state in the loop body without spilling into the L1 cache. It can be used to specify stack variables, global parameters, and subroutine arguments.

- Attributes:
  - `name` - the name of the scalar variale
  - `datatype` - the type (e.g. `int` or `double`) of the variable
  - `isConstant` - `true` if the variable is a constant (e.g. a global or module "parameter" in Fortran), or `false` otherwise
  - `reads` - number of reads from the variable during the loop (exclusive of nested child loops)
  - `writes` - number of writes to the variable during the loop (exclusive of nested child loops)
- Sub-Elements: NONE

0.1.7 The `array` element

This element is used to help estimate the amount of memory traffic required to stream data in and out of memory.

- Attributes:
  - `name` - the name of the array
  - `component` - the component of the array (can leave blank)
  - `datatype` - the type (e.g. `int` or `double`) of the array
  - `accesstype` - `readonly, writeonly, readwrite` - describes the array access type
  - `ghost` - the extent of the ghost/halo region accessed past the index boundaries, given in the form $((gneg_0, gpos_0), (gneg_1, gpos_1), ...)$. If the index space is $((min_0, max_0), (min_1, max_1), ...)$, then the total access space (iteration space plus halos) is $((min_0 + gneg_0, max_0 + gpos_0), (min_1 + gneg_1, max_1 + gpos_1), ...)$. Note that by this definition, $gneg$ values are typically negative.
- Sub-Elements:
  - `access` - an array access that appears in the loop body (exclusive of nested child loops)

0.1.8 The `access` element

This element is used to describe an array location that is accessed in the loop. For relative accesses, the `index` of an access $A(i_0 + c_0, i_1 + c_1, ..., i_k + c_k)$ is given as a tuple of offsets $(c_0, c_1, ..., c_k)$ relative to the loop indices $(i_0, i_1, ..., i_k)$, where k is the nesting depth of the loop body, i_0 refers to the innermost loop variable, and i_k refers to the outermost loop variable. If the loop variables do not appear in the default (innermost to outermost) order for that array access, the optional `indexorder` attribute is used to specify the order in which the loop variables appear.

TODO: Handle mixed relative and absolute indices.

- Attributes:
  - `index` - the index of the access expressed as a tuple of offsets relative to the loop indices
  - `isRelative` - `true` if the index is given relative to the loop indices (e.g. `index` $= (0, 1)$ represents $A(i_0, i_1 + 1)$), or `false` if the index is fixed (e.g. `index` $= (0, 1)$ represents $A(0, 1)$)
  - `indexorder` (*optional*) - a tuple that specifies the order in which the loop variables appear in the array access tuple. For example, if `indexorder` is $(0, 2, 1)$, then `index` $= (1, 2, 3)$ represents $A(i_0 + 1, i_2 + 2, i_1 + 3)$. Defaults to $(0, 1, ..., k)$.
  - `reads` - number of reads from the array index during the loop (exclusive of nested child loops)
  - `writes` - number of writes to the array index during the loop (exclusive of nested child loops)
- Sub-Elements: NONE

0.1.9 The `pininfo` element

This special auxiliary element is used to report counters (measured by the PINTool dynamic analysis tool), which are collected on a per-loop basis during a trial run. This element differs from most other elements in the XML in that it is not used to help describe the static structure of the program, but rather to report the code's observed behavior during execution. This information is not currently used during performance prediction in the ExaSAT performance model.

- Attributes:

  - `entries` - number of entries into the loop
  - `iters` - number of iterations of the loop
  - `ops` - total number of all types of operations performed
  - `reads` - number of read operations
  - `writes` - number of write operations
  - `bytereads` - number of bytes read
  - `bytewrites` - number of bytes written
  - `uniquereads` - number of unique memory addresses read
  - `uniquewrites` - number of unique memory addresses written
  - `adds` - number of additions
  - `subs` - number of subtractions
  - `muls` - number of multiplications
  - `divs` - number of divisions
  - `cachehits` - number of cache hits
  - `cachemisses` - number of cache misses
- Sub-Elements: NONE

0.1.10 The `comm` element

This element is used to describe communications that occur within the function.

- Attributes:
  - `linenum` - the line number where the comm begins
  - `commtype` - the type of communication requested (e.g. `reduce` for a reduction or `ghost` for a ghost halo exchange)
  - `interface` - the name of the library function called
- Sub-Elements:
  - `commscalar` - an integer or floating point variable that is communicated
  - `commarray` - an integer or floating point array that is communicated

0.1.11 The `commscalar` element

Currently only used for `commtype="reduce"`.

- Attributes:
  - `name` - the name of the variable
  - `datatype` - the type (e.g. `int` or `double`) of the variable
- Sub-Elements: NONE

0.1.12 The `commarray` element

Currently only used for `commtype="ghost"`.

- Attributes:
  - `name` - the name of the array
  - `datatype` - the type (e.g. `int` or `double`) of the array
  - `numofcomponents` - the number of array components included in the communication
  - `ghost` - the extent of the ghost halo region to be communicated. Uses the same format as the `ghost` attribute for the `array` subelement of the `loop` element (see Section 0.1.7).
- Sub-Elements: NONE

0.1.13 The `functioncall` element

- Attributes:
  - `linenum` - the line number where the function call occurs
  - `name` - the name of the function call
- Sub-Elements: NONE

0.2 Changelog

0.2.1 Changes in Draft Version 0.3.1

- added `loopvar` attribute to `loop` element
- made `reads` and `writes` attributes mandatory for `scalar` and `access` elements

0.2.2 Changes in Draft Version 0.3

- `loop` elements no longer represent entire loop nests and are hierarchically nested to represent each loop statement individually. Changed `stride`, removed `range`, and added `lowerbound` and `upperbound` to reflect update. Operation counts and scalar and array accesses reflect code in the current loop level only, not nested child loops.
- separated `divides` from `multiplies` attribute in `loop` element
- removed `componentsweep` attribute from `loop` element
- changed `type` attribute to `datatype` in elements `scalar`, `array`, `commscalar`, and `commarray`
- changed `indextype` attribute to `isRelative`
- added `cachemisses` attribute to `pininfo` element

0.2.3 Changes in Draft Version 0.2

- removed `var` elements, moved children (`local, nonlocal, scalar, array`) up one level to `function` or `loop`.
- removed `flops` element, moved children (`add, multiply, special`) up one level to `loop` and changed them to attributes
- changed loop `range` from element to attribute

- added `type` attribute to `array` element
- renamed `access` element's `value` attribute to `index`
- renamed `access` element's `type` attribute to `indextype`
- added optional `reads` and `writes` attributes to `scalar` and `access` elements
- moved `commtype` attribute into `comm` element
- renamed `comm` element's `array` sub-element to `commarray` for uniqueness
- added `commscalar` sub-element to `comm` element (for use with `commtype="reduce"`)
- added `pininfo` sub-element to `loop` element