Massive Visualization of Application Codes

HPC Application developers are hero programmers because writing parallel programs is much harder than writing sequential ones. They have to understand the intricate details of the target architecture (e.g. GPUs, etc), and the programming models to exploit this parallelism and express this in the code structure of the application. Using performance data, they need to go through thousands or millions of lines of code so that they can devise a strategy to port to the target architecture. Typically, most codes start by parallelizing the application across nodes and then by adding in-node parallelisms incrementally to exploit the multi-cores or accelerators available on the node until the target performance is met.

The data set for this challenge contains metadata of the program information for the E3SM (Energy Exascale Earth System Model) application[1]. It contains information about the usage of Fortran features, programming models, subroutine calls, numbers of statements that are parallelized, type of statements, Fortran module usage, source and object file location of subroutines, etc.

Challenge Questions

We need a scalable way to visualize this information. The challenges are:

1) Build a call graph of the application (in the order of 7,000 nodes and 22,000 edges) based on the JSON files.

2) Visualize the call graph in a scalable way using different algorithms to show the distance between the nodes.

3) Classify the nodes of the call graph, for example, based on
    a) directory location of the source files or alternative way(s) to indicate information about the physical model (sea, ice, land, atmosphere)
    b) OpenMP and OpenACC
    c) code usage from a given Fortran module
    d) number of executable statements and/or variables
    e) relationships between function callers/callees
    f) library call invocations and/or invocation frequencies
    g) (any others)

[1] https://climatemodeling.science.energy.gov/projects/energy-exascale-earth-system-model

4) Demonstrate the distribution of the different libraries' usage in application.

5) Show the relative similarities between nodes based on code features or programming techniques such as, for example:
    a) Number of loops or looping structures
    b) Parallelization
    c) Module usage
    d) Call site sequences
    e) Other classifications from #3
    f) (any others)