CallFlow

Release 1.1.0

Sep 25, 2020

User Docs

CallFlow is an interactive visual analysis tool that provides a high-level overview of CCTs together with semantic refinement operations to progressively explore the CCTs.

You can get CallFlow from its GitHub repository:

```
$ git clone https://github.com/LLNL/CallFlow.git
```

If you are new to CallFlow and want to start using it, see *Getting Started*, or refer to the full *User Guide* below.

CHAPTER 1

Getting Started

1.1 Prerequisites

The callflow (python package) requires python (>= 3.6) and pip (>= 20.1.1). Other dependencies are checked/installed during the installation of callflow using pip.

Other dependencies are:

1. hatchet
2. pandas
3. networkx (2.2)
4. numpy
5. flask
6. statsmodels, and
7. sklearn

CallFlow is available on GitHub

The callflow app (visualization component) requires node.js (>= 13.7.0) and npm (>= 6.13.7). If there is an older version of node installed, install nvm and use the following command to change version.

```
$ nvm use 13.7.0.
```

1.2 Installation

You can get CallFlow from its GitHub using this command:

```
$ git clone https://github.com/LLNL/CallFlow.git
```

1.2.1 Install callflow python package

To install callflow python package, run the following command using pip.

```
$ pip install .
```

To install in the dev mode,

```
$ pip install -e . --prefix=/path/to/install
```

1.2.2 Check Installation of callflow python package

After installing callflow, make sure you update your PYTHONPATH environment variable to point to directory where callflow was installed.

```
$ python
Python 3.7.7 (default, Jul 11 2019, 01:08:00)
[Clang 11.0.0 (clang-1100.0.33.17)]] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Typing `import callflow` at the prompt should succeed without any error messages:

```
>>> import callflow
>>>
```

1.2.3 Install the Visualization client

```
$ cd app
$ npm install
```

1.3 Supported data formats

Currently, hatchet supports the following data formats as input:

- HPCToolkit database: This is generated by using `hpcprof-mpi` to post-process the raw measurements directory output by HPCToolkit.
- Caliper Cali file: This is the format in which caliper outputs raw performance data by default.
- Caliper Json-split file: This is generated by either running cali-query on the raw caliper data or by enabling the mpireport service when using caliper.

For more details on the different input file formats, refer to the *User Guide*.

CHAPTER 2

User Guide

CallFlow is structured as three components:

1. A Python package callflow that provides functionality to load and manipulate callgraphs.
2. A D3 based app for visualization.
3. A python server to support the visualization client

2.1 Sample Datasets

Sample data and examples are provided in the data and examples.

2.2 Arguments

```
--verbose - Display debug points.
(optional, default: false)

--config - Config file to be processed.
(Either config file or data directory must be provided)

--data_dir - Input directory to be processed.
(Either config file or data directory must be provided)

--process - Enable process mode.
(default: false)

--profile_format - Profile format.
(required, either hpctoolkit | caliper | caliper_json)

--save_path - Save path for the processed files.
(optional, default: data_dir/.callflow)
```

(continues on next page)

(continued from previous page)

```
--filter_by - Set filter by column
(optional, e.g., "time" or "time (inc)")

--filter_perc - Set filter percentage.
(optional, e.g., 10, 20, 30)

--group_by - Set the semantic level for supergraph
(optional, e.g., module to get super graph, name to get call graph, default: 'module')

--read_parameter - Enable parameter analysis.
(optional. This is an experimental feature)
```

2.3 Process datasets

First step is to process the raw datasets to use with CallFlow. The processing can be done either by passing data directory (using –data_dir), or using *config.callflow.json* file (using –config).

1. Using the directory (i.e., –data_dir).

The user can input a directory of profiles of a "same" format for processing.

```
$ python3 server/main.py --data_dir /path/to/dataset --profile_format hpctoolkit --
↪process
```

Note: The processing step typically entails some filtering and aggregation of data to produce the reduced graphs at desired granularity. To do this, the user can currently provide 3 arguments, namely –filter_by, –filter_perc, and –group_by.

2. Using the config file (i.e., –config).

The user can process profiles from different formats using the config file. The parameters of the preprocessing are provided through a config file (see examples of config files in the sample data directories). For example, the user can pass other arguments (e.g., save_path, filter_perc, etc.).

Listing 1: Sample *callflow.config.json* to process 3 datasets

```
{
    "experiment": "experiment_name",
    "save_path": "/path/to/dir",
    "read_parameter": false,
    "runs": [{
            "name": "run-1",
            "path": "/path/to/run-1",
            "profile_format": "hpctoolkit | caliper | caliper_json"
        },
        {
            "name": "run-2",
            "path": "/path/to/run-2",
            "profile_format": "hpctoolkit | caliper | caliper_json"
        },
        {
            "name": "run-3",
            "path": "/path/to/run-3",
            "profile_format": "hpctoolkit | caliper | caliper_json"
```

(continues on next page)

(continued from previous page)

```
        }
    ],
    "schema": {
        "filter_by": "time (inc)",
        "filter_perc": 0,
        "group_by": "name",
        "module_map": {
            "Lulesh": ["main", "lulesh.cycle"],
            "LeapFrog": ["LagrangeNodal", "LagrangeLeapFrog"],
            "CalcForce": ["CalcForceForNodes", "CalcVolumeForceForElems",
↪"CalcHourglassControlForElems", "CalcFBHourglassForceForElems"],
            "CalcLagrange": ["LagrangeElements", "UpdateVolumesForElems",
↪"CalcLagrangeElements", "CalcKinematicsForElems", "CalcQForElems",
↪"CalcMonotonicQGradientsForElems", "CalcMonotonicQRegionForElems",
↪"ApplyMaterialPropertiesForElems", "EvalEOSForElems", "CalcEnergyForElems",
↪"CalcPressureForElems", "CalcSoundSpeedForElems", "IntegrateStressForElems"],
            "Timer": ["TimeIncrement"],
            "CalcConstraint": ["CalcTimeConstraintsForElems",
↪"CalcCourantConstraintForElems", "CalcHydroConstraintForElems"],
            "NA": ["Unknown"],
            "MPI": ["MPI_Barrier", "MPI_Reduce", "MPI_Allreduce", "MPI_Irecv", "MPI_
↪Isend", "MPI_Wait", "MPI_Waitall", "MPI_Finalize"]
        }
    }
}
```

```
$ python3 server/main.py --config /path/to/config.callflow.json --process
```

2.4 Using CallFlow as a web app

To run CallFlow's web app, a python-based WSGI server (handles socket communication and processing of data) and a Vue client server need to be run simultaneously.

1. Run the WSGI server.

Note: Similar to the processing step, the web server can be run either using –config or –data_dir.

```
$ python3 server/main.py --data_dir /path/to/dataset
```

or

```
$ python3 server/main.py --config /path/to/config.callflow.json
```

2. Run the client server.

```
$ cd app
$ npm run dev
```

2.5 Using CallFlow inside Jupyter notebook environment

Use %callflow magic to run CallFlow in Jupyter notebook environment,

1. To load the callflow's magic extension, use the command %load_ext.

```
%load_ext callflow
```

2. Now, %callflow can be used to trigger the user interface like the command line.

```
%callflow --data_dir /path/to/directory --profile_format format
```

or

```
%callflow --config /path/to/config/file
```

This feature will spawn the server and client in the background as child processes to Jupyter. It will also detect if any existing processes are in execution and attach seamlessly.

For reference, see an example notebook

If you encounter bugs while using CallFlow, you can report them by opening an issue on GitHub.

If you are referencing CallFlow in a publication, please cite the following paper:

- Huu Tan Nguyen, Abhinav Bhatele, Nikhil Jain, Suraj Kesavan, Harsh Bhatia, Todd Gamblin, Kwan-Liu Ma, and Peer-Timo Bremer. Visualizing Hierarchical Performance Profiles of Parallel Codes using CallFlow. In IEEE Transactions on Visualization and Computer Graphics, November 2019. DOI

CHAPTER 3

Indices and tables

- genindex
- modindex
- search