PerfFlowAspect: Scalable Composition and Analysis Techniques for Scientific Workflows

https://perfflowaspect.readthedocs.io/en/latest/

CASC
Center for Applied Scientific Computing

May 11, 2023

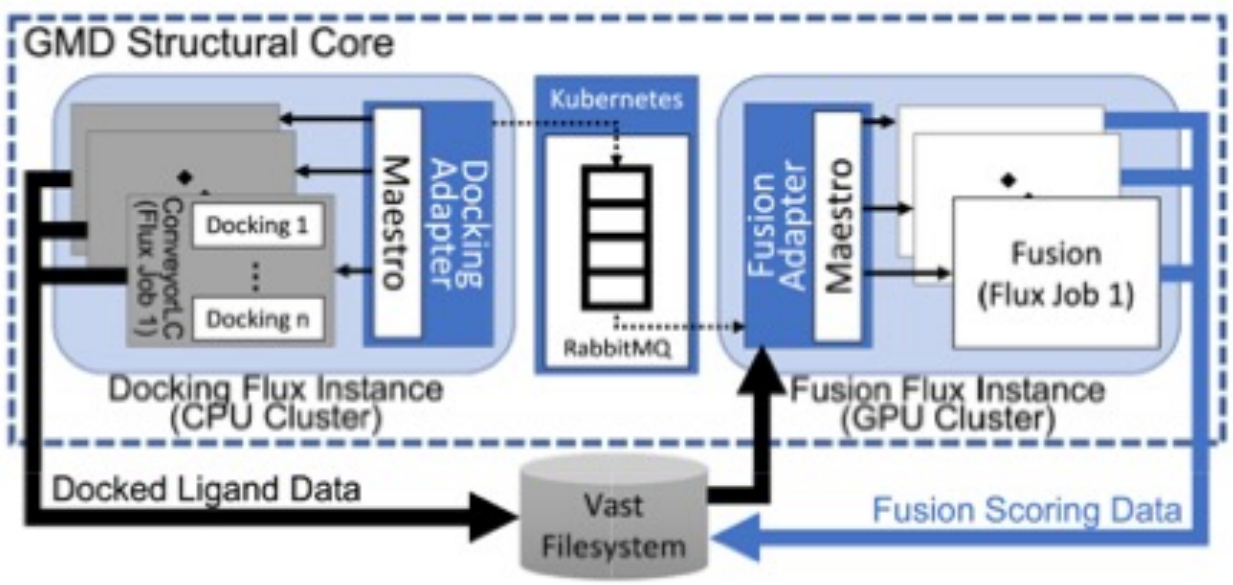
Team: Tapasya Patki, Stephanie Brink, Aliza Lisan , Dan Milroy, Jae-Seung Yeom
Previous members: Dong Ahn, Stephen Herbein , Frank Di Natale

LLNL-PRES- 849105
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

There is a paucity of techniques that can effectively analyze the end-to-end performance of a composite science workflow

- HPC researchers are introducing and composing disparate workflow management technologies to enable scalable end-to-end science
  - MuMMI: cancer workflow, Maestro + Flux
  - COVID-19 drug design: Maestro, Flux, ATOM
  - Generative Molecular Design (GMD) pipeline and micro-services running on on-prem. Kubernetes
  - ECP ExaWorks project (https://exaworks.org): collection of composable tools!



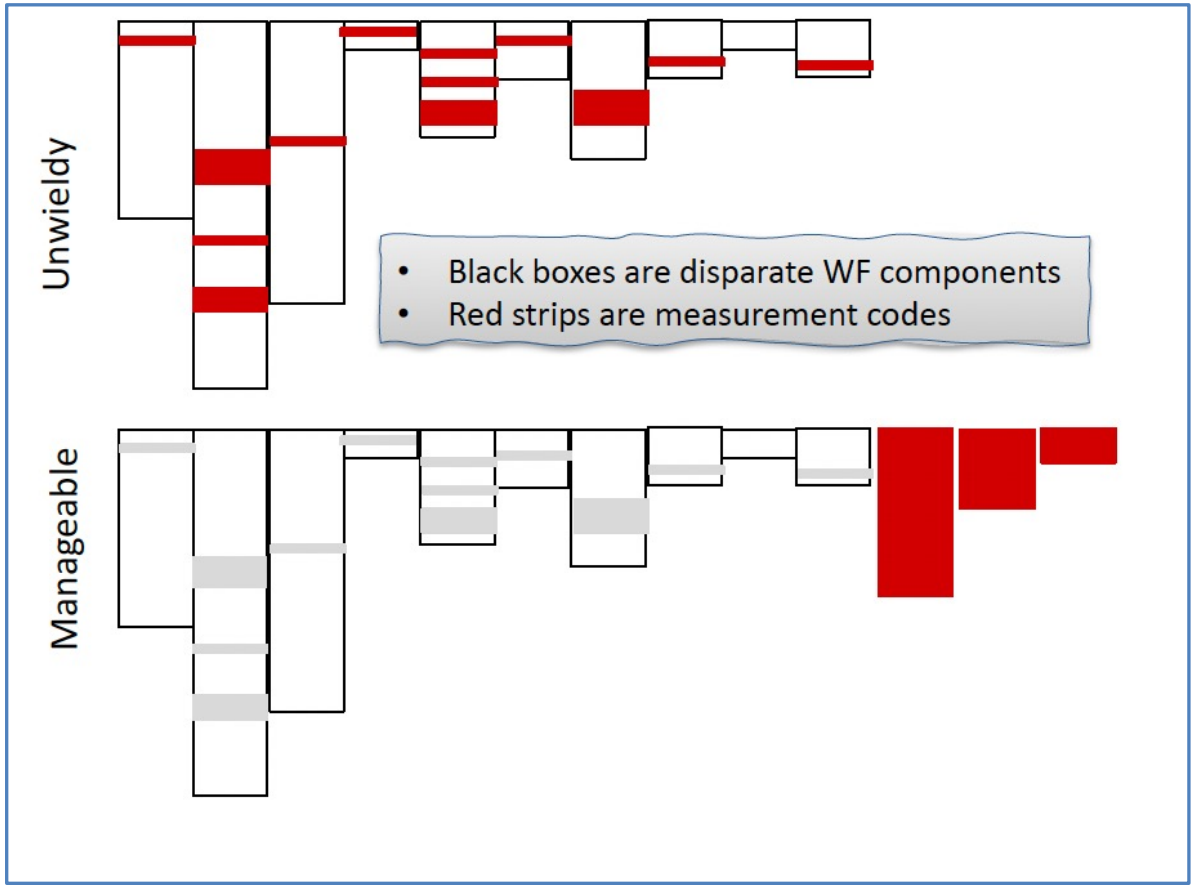
- Holistic performance analysis is challenging due to multiple binaries, underlying frameworks, multiple clusters, etc.

Src: Scalable Composition and Analysis Techniques for Massive Scientific Workflows
Best Paper Award at e-Science, 2022.

PerfFlowAspect : cross-cutting performance-analysis concerns across disparate workflow-management technologies and components.

Aspect-Oriented Programming Paradigm

- Minimum modifications to the disparate workflow-management technologies
- Modularize the performance analysis concerns
- Allow for customization of performance analysis ‘actions’ or ‘advices’: timing or utilization measurements, hardware performance counters, etc

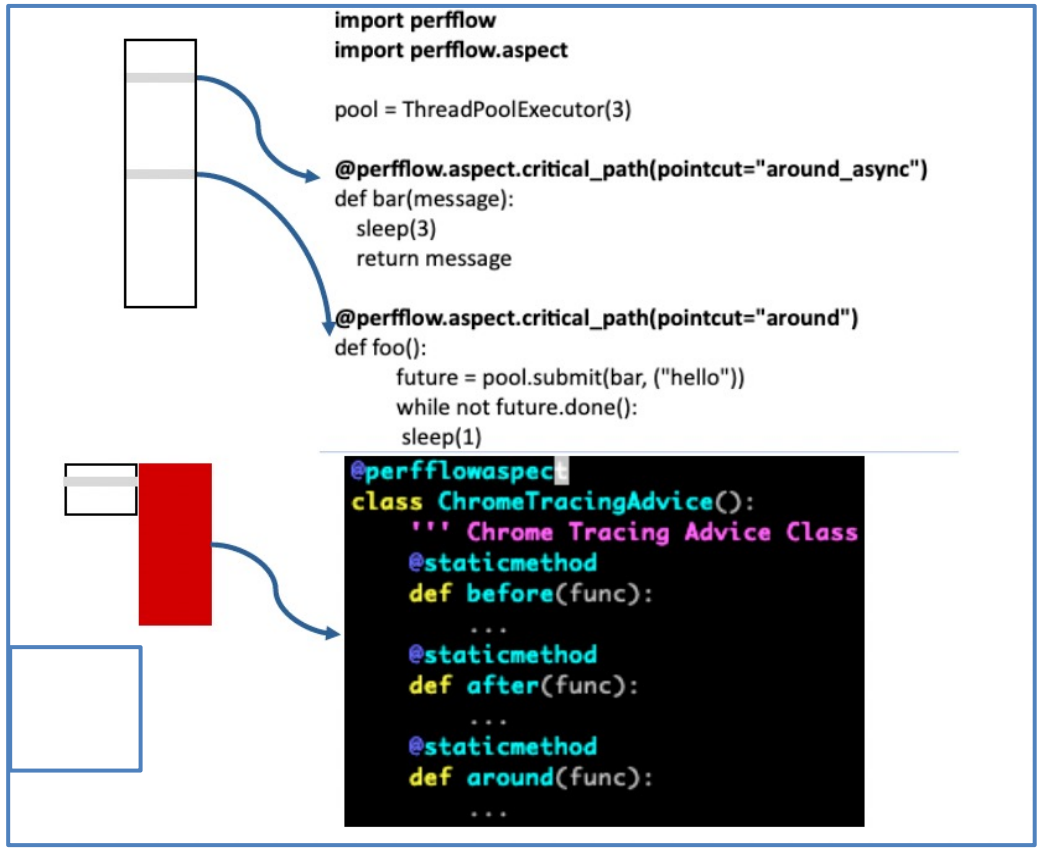


Common Aspect-Oriented Programming (AOP) Terminology

- **Join Points**: A point in a program's execution in which the behavior can be modified by AOP. The candidate points are method invocations.
- **Pointcuts**: An expression used to match join points (creating a set of join points), allows you to specify to AOP where and when in the code to make modifications.
- **Advice**: Defines the additional behavior or action that will be inserted into the code, specifically at each join point matched by the pointcut.
- **Aspect**: The collection of the pointcut expression and the advice.
- **Weaver**: Applies aspects into the code, modifying code at join points with matching pointcuts and associated advices. The combining of aspects and code enables execution of cross-cutting concerns.

PerfFlowAspect : cross-cutting performance-analysis concerns across disparate workflow-management technologies and components.

- Prototype using decorators for Python and LLVM-based library for C++
- Advice (or action) emits tracing event data every time the annotated functions is called
- Utilize Chrome Tracing Format (CTF) and Perfetto visualizer



PerfFlowAspect: Examples of annotations

- Simple, minimally-intrusive annotated functions on the critical path of the execution of the program

- At each annotated function, AOP based performance metrics are collected.

```python
import time
import perfflowaspect
import perfflowaspect.aspect

@perfflowaspect.aspect.critical_path(pointcut="around")
def bas():
    print("bas")

@perfflowaspect.aspect.critical_path(pointcut="around")
def bar():
    print("bar")
    time.sleep(0.001)
    bas()

@perfflowaspect.aspect.critical_path()
def foo(msg):
    print("foo")
    time.sleep(0.001)
    bar()
    if msg == "hello":
        return 1
    return 0

def main():
    print("Inside main")
    for i in range(4):
        foo("hello")
    return 0
```

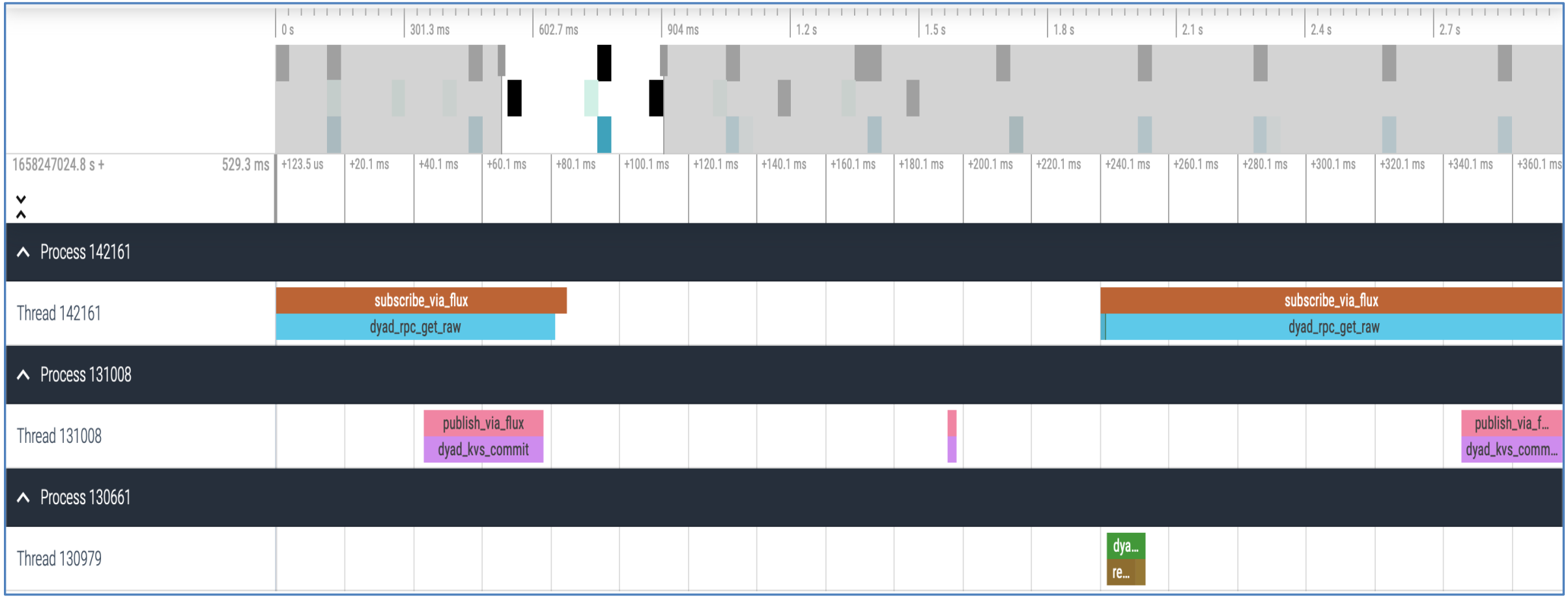
```cpp
__attribute__((annotate("@critical_path()")))
int foo(const std::string &str)
{
    printf("Hello\n");
    bar();
    if (str == "hello")
    {
        return 1;
    }
    return 0;
}

int main(int argc, char *argv[])
{
    printf("Inside main\n");
    foo("hello");
    return 0;
}
```

PerfFlowAspect: Chrome Tracing Format

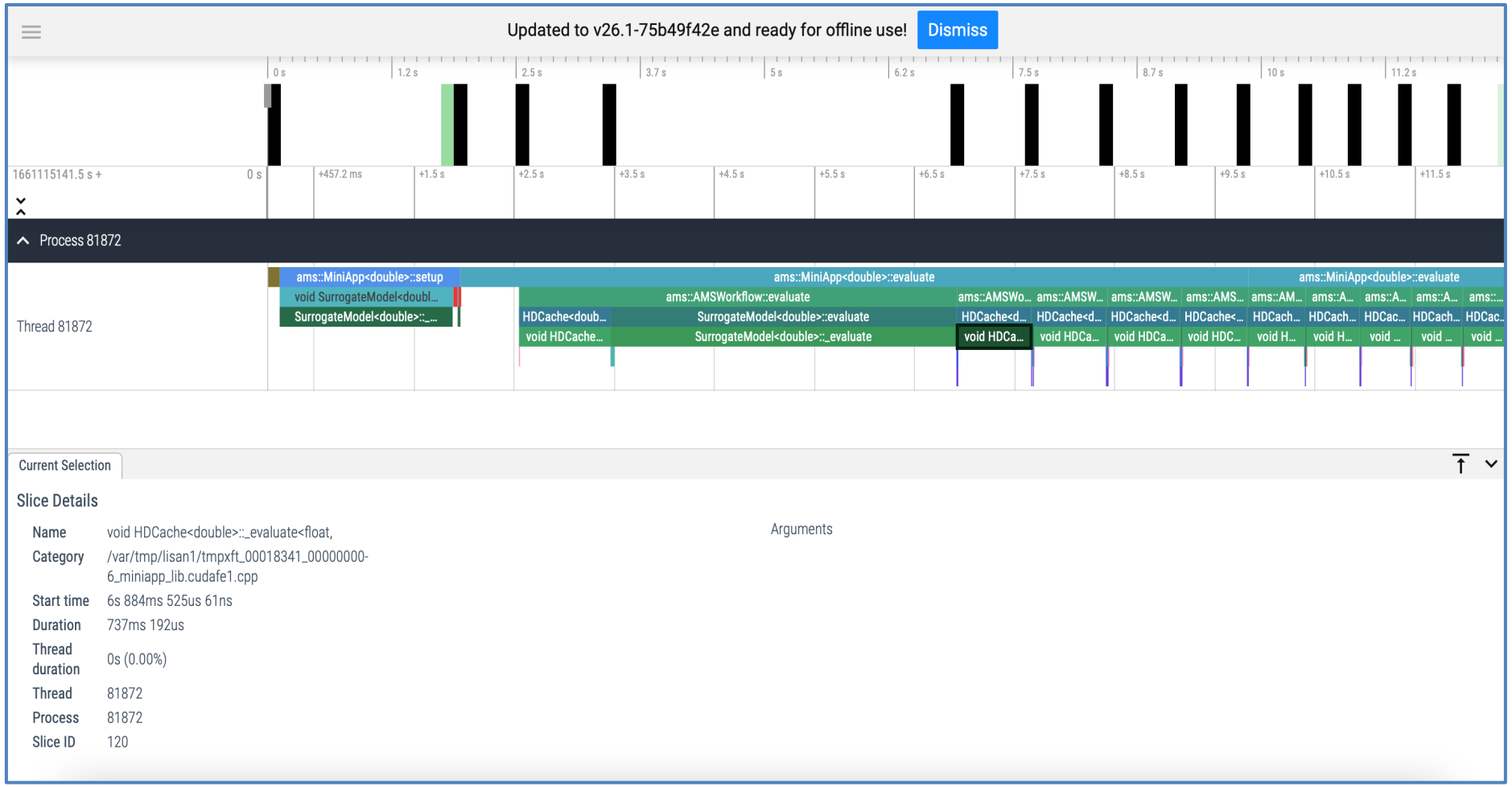
- Running the annotated code and the associated advice with it will produce a performance trace data file which uses the Chrome Tracing Format in JSON.


```
[
{"name": "foo", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919690977.0, "ph": "B"},
{"name": "bar", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919693291.0, "ph": "B"},
{"name": "bas", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695089.8, "ph": "B"},
{"name": "bas", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695089.8, "ph": "C", "args": {"cpu_usage": 97.3, "memory_usage": 4.096}},
{"name": "bas", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695589.8, "ph": "C", "args": {"cpu_usage": 0, "memory_usage": 0}},
{"name": "bas", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695589.8, "ph": "E"},
{"name": "bar", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919693291.0, "ph": "C", "args": {"cpu_usage": 36.6, "memory_usage": 8.192}},
{"name": "bar", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695792.2, "ph": "C", "args": {"cpu_usage": 0, "memory_usage": 0}},
{"name": "bar", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695792.2, "ph": "E"},
{"name": "foo", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919690977.0, "ph": "C", "args": {"cpu_usage": 36.6, "memory_usage": 8.192}},
{"name": "foo", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695952.0, "ph": "C", "args": {"cpu_usage": 0, "memory_usage": 0}},
{"name": "foo", "cat": "__main__", "pid": 53633, "tid": 4613260800, "ts": 1657215919695952.0, "ph": "E"},
```

Visualizing traces using the Perfetto UI: DYAD Example

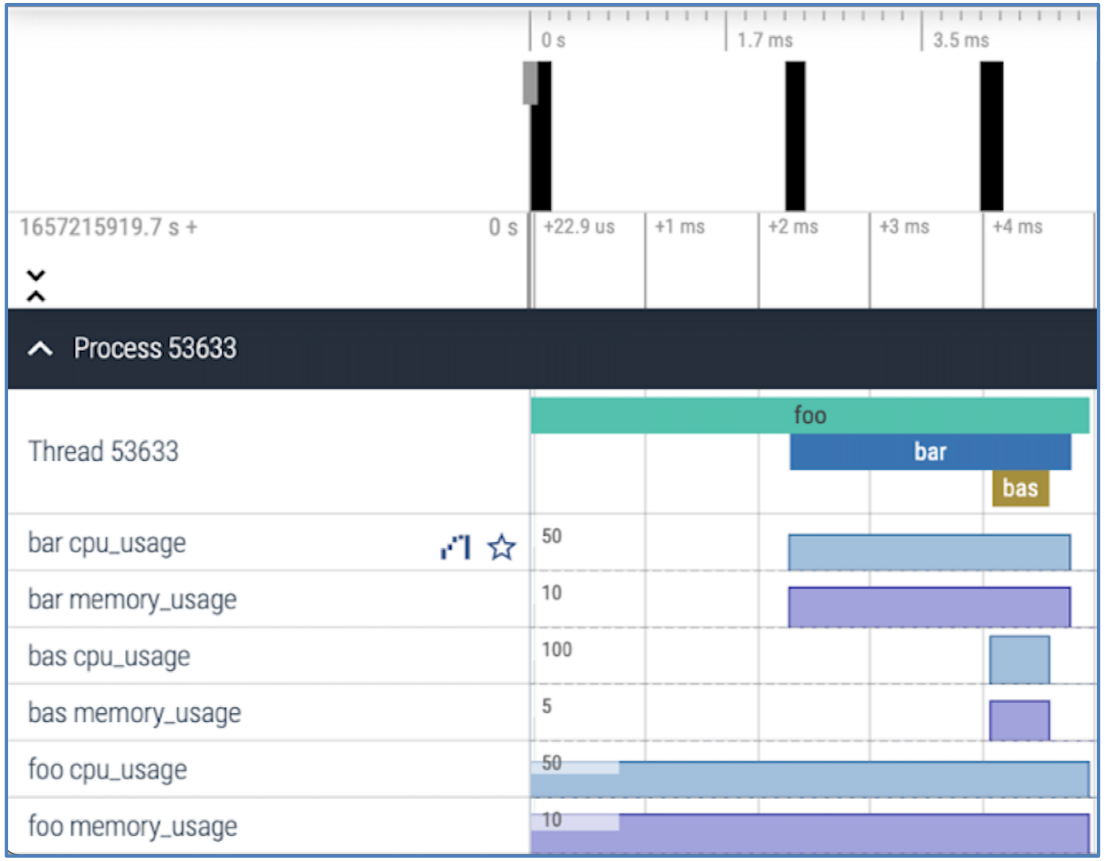


Visualizing traces using the Perfetto UI: AMS Example



PerfFlowAspect: Work in Progress

- Better support for multi-process, multi-thread visualizations
- Improve code quality, test coverage, and documentation
- Add performance metrics such as GPU usage and other performance counters, add Hatchet support
- Integrate PerfFlowAspect with AMS workflow in order to analyze its performance
- Identify other workflows of interest



CASC

Center for Applied
Scientific Computing

Lawrence Livermore
National Laboratory

Thank you!