NVIDIA DRIVEWORKS 5.8

SYSTEM TASK MANAGER

September 20, 2022
Advance Information | Subject to Change
NVIDIA CONFIDENTIAL | Prepared and Provided under NDA

User Guide

Contents

1 Overview

Large-scale deployment of autonomous vehicles (AVs) requires the reconciliation of two competing goals: performance and safety. AV systems also need to be deterministic so that their worst-case behavior can be validated for satisfying timing guarantees. On a complex SoC like the NVIDIA DRIVE Orin™ SoC, the behavior exhibited by any software component depends on the state of the SoC as exhibited by two factors: the state of other software components (locks, memory contention, etc.), and the state of the hardware (caches, physical memory, etc.). To solve the problems associated with dynamic scheduling, we present a static, centrally monitored, OS-agnostic, non-preemptive scheduler that manages work across hardware engines on NVIDIA DRIVE Orin™ SoC.

Our scheduling framework also offers methodologies and tools to both characterize and tune the interactions between tasks on different hardware engines.

2 Getting Started

The NVIDIA System Task Manager (also known as STM) consists of the following components:

- Schedule Compiler
- Runtime
- Analytics
- Graph Visualizer
- Schedule Visualizer

The workflow for using these components is given below:

1. Application specifications and constraints are captured by the user in a compute graph.
2. The compute graph is fed to the Schedule Compiler, which processes it and generates a static schedule that is then passed over to the Runtime.
3. The executed static schedule can be switched by creating a Schedule Manager process which binds with STM's Schedule Manager Library. The Schedule Manager can stop and start execution of the provided schedules. All the static schedules must be provided to STM Master upfront, and all the runnables and resources must be registered during initialization.
4. During execution, the scheduling decisions and relevant events are logged in a log file.
5. This log file is then processed by the analytics tools to generate reports with execution statistics for the application. These can be used by developers to fine-tune the real-time behavior of the application by changing constraints in the compute graph or by making code changes.

6. To facilitate the ease of development, we also provide two tools: graph visualizer to visualize the input Directed Acyclic Graph and schedule visualizer to visualize the schedule table generated by schedule compiler.

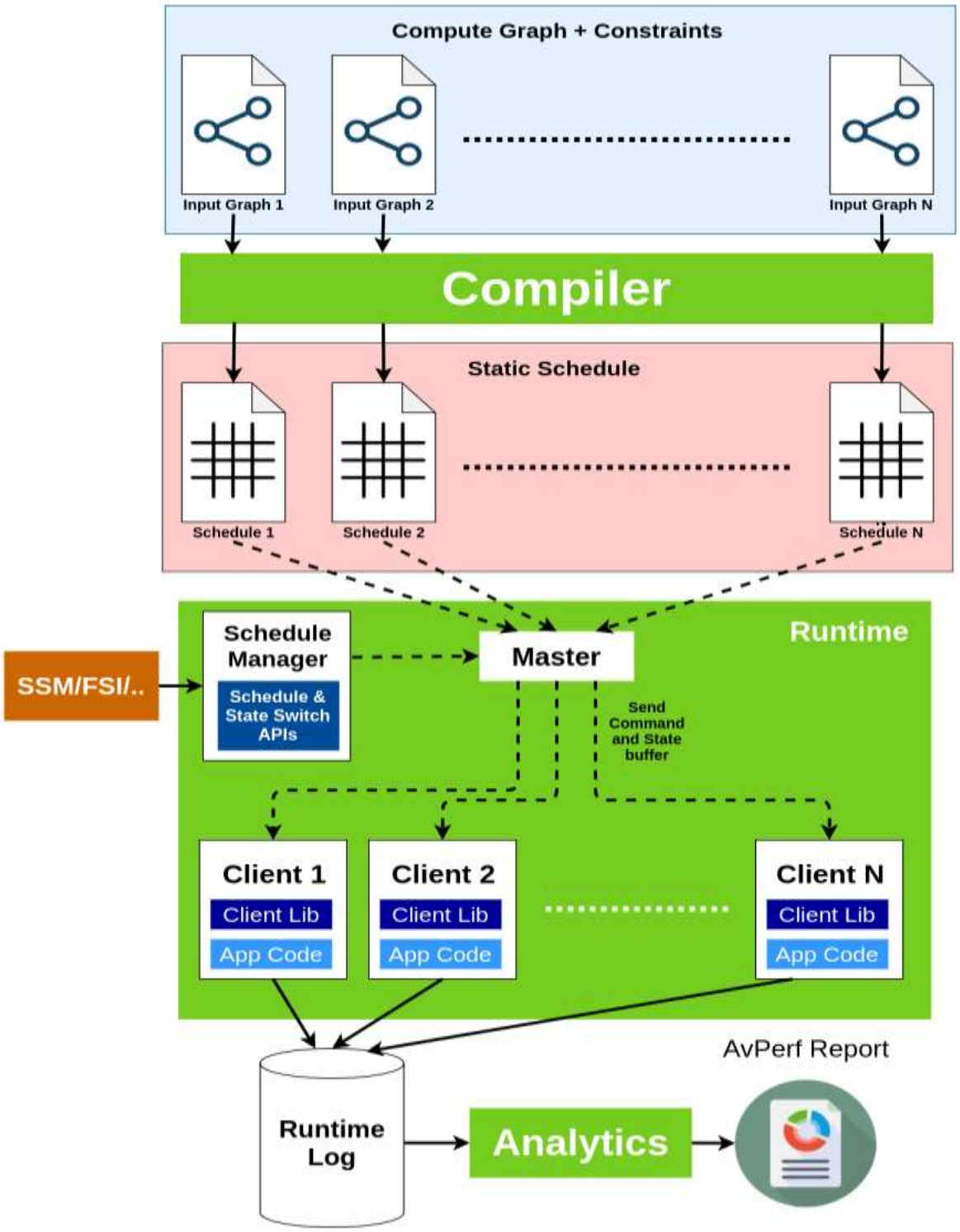

Figure 1 Overview of NVIDIA System Task Manager

2.1 Definitions

- Runnable: A runnable is an atomic unit of work, generally work that can be encompassed by a single function running on a single engine. Runnables can have dependencies on other runnables.
- Client: Clients are OS processes that contain runnables.
- Epoch: A periodic time base which dictates the rate at which a sub-graph of runnables is respawned.
- Resource: Any hardware engine (e.g., CPU, GPU, etc.) or software resources (e.g., CUDA streams, scheduling mutexes, etc.) shared by multiple runnables that need to be modeled by the STM compiler.

2.2 Supported Hardware Engines

STM provides the ability to schedule Runnables across the various hardware engines on the NVIDIA DRIVE Orin™ SoC as well as any discrete GPUs attached to NVIDIA DRIVE Orin™ SoC. Below is a short overview of the supported hardware engines and software libraries.

- CPU: STM can schedule Runnables across the different CPUs in NVIDIA DRIVE Orin™ SoC's CCPLEX. A CPU Runnable is represented as a callback function (C/C++) that is registered with the STM run-time. STM maintains a pool of worker threads that will call the clients registered functions at the appropriate offset in the schedule.
- GPU: STM can schedule GPU Runnables on NVIDIA DRIVE Orin™ SoC's iGPU as well as any discrete dGPUs attached to NVIDIA DRIVE Orin™ SoC. If running on an x86 machine, only dGPUs are supported. STM only supports scheduling of CUDA compute tasks and any higher-level libraries that are built on top of Cuda (TensorRT, CuDNN, CuBLAS, DriveWorks, etc.).
- NvMedia Engines: STM supports scheduling runnables on the DLA currently. NvMedia Runnables will be scheduled using the native synchronization primitives supported by the hardware, as abstracted by the DRIVE OS NvSciSync framework.
- PVA: STM can schedule PVA Runnables on NVIDIA DRIVE Orin™ SoC's PVA engine using the APIs in the wrappers provided with STM. The PVA 's capabilities are a good good match in the algorithmic domains where we need to have predictable processing, at low power and low latency.

3 Modules

3.1 STM Compiler

The schedule compiler processes the user input compute graph and outputs a static schedule. This section goes over the creation of the compute graph and outlines the mechanism that the compiler uses to create a static schedule from the compute graph.

3.1.1 Compute Graph and Constraints

The application requirements are captured by a compute graph, where each node in the compute graph specifies an atomic task on a single hardware engine. This section will walk you through the graph specification as captured by our YAML schema. Note that YAML treats everything as a list or a dictionary. Ordering of fields does not matter - the compiler is built to accept the fields in any order specified if the expected nesting hierarchy is satisfied. Note that IDs specified cannot contain a period (.) symbol as that symbol is used internally by the Framework.

3.1.1.1 Version

This field specifies the input specification version number. This is required to ensure that incompatible features are disabled/flagged. Ideally it should match the version of the provided compiler package. This field is mandatory.

```
Version: 2.0.0 # Input specification version - currently 2.0.0
```

3.1.1.2 Graph ID

The graph ID is the second top-level entry in the YAML (first being the Version). The graph definition is nested under the graph ID.

```
Version: 2.0.0 # Input specification version - currently 2.0.0

SimpleGraph: <..Graph Description..>
```

3.1.1.3 Schedule ID

This is the unique identifier used for schedule identification in multi-schedule runs. This value should be between 0-65535 (inclusive).

```
SimpleGraph: <..Graph Description..>

  Identifier: 1  # <Integer between 0 to 65,535>
```

3.1.1.4 Global Resources

Resources that are used system-wide are modeled under the global resources section. Hardware resources like CPUs or GPUs should go under this section. Any system-wide virtual scheduling mutexes can also be listed here. The compiler models each resource as a timeline on which only one runnable can execute at any time. A runnable can, however, use more than one resource. There are some limitations on the types of resources which can be simultaneously used by a runnable, which are covered in section 3.1.4 . Generally, resources are specified in the following format. Certain types of resources have additional features which are described in the respective

sections. Global resources are nested under a Resources section under the graph ID. To define a resource, a resource type needs to be specified. Resource instances are grouped under the appropriate resource type. YAML supports two ways of specifying lists, which are shown in the example below.

```
SimpleGraph:
Resources:
    Resource_Type0: [Rsrc_Type0_Instance0, Rsrc_Type0_InstanceN]
    Resource_Type1:
        - Rsrc_Type1_Instance0
        - Rsrc_Type1_Instance1
```

The CPU, GPU, DLA and VPU resource types are known resource types for the compiler, and it will take specialized scheduling steps for runnables scheduled on those resources. Other resource types are considered as scheduling mutexes, and they do not have any naming restrictions.

3.1.1.5 The CPU Resource Type

To specify CPUs in the system, the resource type should be set to CPU and the resource instances should be named as CPUX, where X is a valid CPU number.

```
SimpleGraph:
    Resources:
        CPU: [CPU0, CPU1, CPU2]
```

3.1.1.6 The GPU, DLA and VPU Resource Types

Graphical Processing Units (GPUs), Deep Learning Accelerators (DLAs) and Vector Processing Units (VPUs) are special hardware accelerators that can be used to offload computation from the CPUs. Work is submitted to these engines through CUDA Streams, DLA Queues and PVA Streams respectively. Supported device IDs are GPUX, DLAX and VPUX, where X is a valid instance number. While specifying the device instances, an optional limit can be specified to enforce a limitation on the number of streams/queues mapped to that device instance. To specify a Stream/Queue limit on an instance, append the instance ID with: Y, where Y is the limit. In the following example, instance GPU0 allows unlimited CUDA Streams, whereas GPU1 allows only 8 Streams. Similarly, DLA0 allows 4 Queues and DLA1 allows unlimited queues.

```
SimpleGraph:
Resources:
```

GPU:

- GPU0 # Unlimited Streams

- GPU1: 8 # 8 Streams

DLA:

- DLA0: 4 # 4 Queues

- DLA1 # Unlimited Queues

3.1.1.7 The Scheduling MUTEX Resource Type

Any resource type not known by the compiler is modeled as a scheduling mutex. There are no naming conventions associated with either the resource type or the resource ID for a scheduling mutex. Interfering runnables can specify a scheduling mutex as a resource requirement to prevent the compiler from scheduling them concurrently.

SimpleGraph:

Resources:

Can be used to mutually exclude memory-intensive tasks

MEMORY_BUS: [MEMORY_BUS0]

Scheduling mutexes

MUTEX: [SCHED_MUTEX0, SCHED_MUTEX1]

3.1.1.8 Hyperepochs

A hyperepoch is a resource partition that runs a fixed configuration of epochs that share the resources in that partition. It is periodic in nature, and it respawns the contained epochs at the specified period. This relationship between the hyperepoch and its member epochs will be covered in section 3.1.1.7.3. To define a hyperepoch, the required fields are Resources, Period and Epochs. In certain configurations, some fields can be omitted as specified in the respective sections. Hyperepochs are specified in a list under the 'Hyperepochs' keyword inside the Graph specification as shown below. Hyperepoch0 is the ID of the hyperepoch that is defined in the following graph.

SimpleGraph:

Hyperepochs:

- Hyperepoch0: # Hyperepoch ID

Period: 100ms

Resources:

- MEMORY_BUS0

- GPU0

- CPU0

- CPU1

Epochs:

3.1.1.8.1 Period

The period for a hyperepoch specifies the rate at which the contained epochs are spawned. This field can be omitted if the hyperepoch has only one epoch, and the periodicity of the hyperepoch is equal to that of the contained epoch. The period field is nested under the hyperepochs ID.

3.1.1.8.2 Resources

Each hyperepoch is associated with a mutually exclusive set of resources. Resources are mapped to hyperepochs by specifying the resource IDs in a list under the Resources heading inside the hyperepoch specification as shown in the example above. There, the resources MEMORY BUS0, GPU0, CPU0 and CPU1 are mapped to the hyperepoch Hyperepoch0. If there is only one hyperepoch in the system, this resource-mapping can be omitted and the hyperepoch is assumed to have access to all the resources in the system.

3.1.1.8.3 Epochs

Epochs are time bases at which rate constituent runnables spawn confined to the boundaries of the hyperepoch. Each epoch is a member of a hyperepoch, and has two attributes associated with it -Period and Frames. For specifying epochs, list epoch IDs under the Epochs heading in a hyperepoch as shown below.

Hyperepochs:

- Hyperepoch0:

Period: 100ms

Epochs:

- Epoch0:

Period: 10ms

Frames: 8

- Epoch1:

Period: 100ms

- Epoch2:

Period: 33ms

Frames: 3

Frames and Period for Epochs

The period specified for the epoch specifies the rate at which a frame of runnables is spawned, up to the number of frames specified, in the hyperepochs period. By default, if not specified, the number of frames is 1. In the example given above, Epoch0 spawns 8 frames in 100ms. Each frame is spawned 10ms apart. Epoch1 spawns once, as the number of frames defaults to 1, in the hyperepoch. Epoch2 spawns thrice, 33ms apart. If periodicity is not required at the epoch level, it can be omitted, and the number of frames would specify the number of times the epoch's set of runnables needs to be spawned. This can be used to figure out the number of frames that can fit inside the hyperepochs period. The following example shows how a system can use hyperepochs to define different frequency domains.

Version: 2.0.0

Drive: # Graph ID

Resources: # Global Resources

CPU: [CPU0, CPU1, CPU2]

GPU: [GPU0]

Hyperepochs:

- Perception: # Hyperepoch ID

Period: 100ms # Hyperepoch period

Resources: [CPU1, CPU2, GPU0] # Resource mapping

Epochs:

- Camera: # Epoch ID

Period: 33.33ms

Frames: 3

- Radar: # Epoch ID

Period: 100ms

Frames: 1

- Control: # Hyperepoch ID; Hyperepoch

Resources: [CPU0] # period inferred from epoch.

Epochs:

- VDC: # Epoch ID

Period: 10ms # Epoch frames = 1 (default)

This configuration has been visualized in the following figure. Note that Camera and Radar frames are synchronized with each other at the hyperepoch boundary, VDC frames are not aligned with either the Camera or Radar frames as they are in a separate hyperepoch with a different time base.

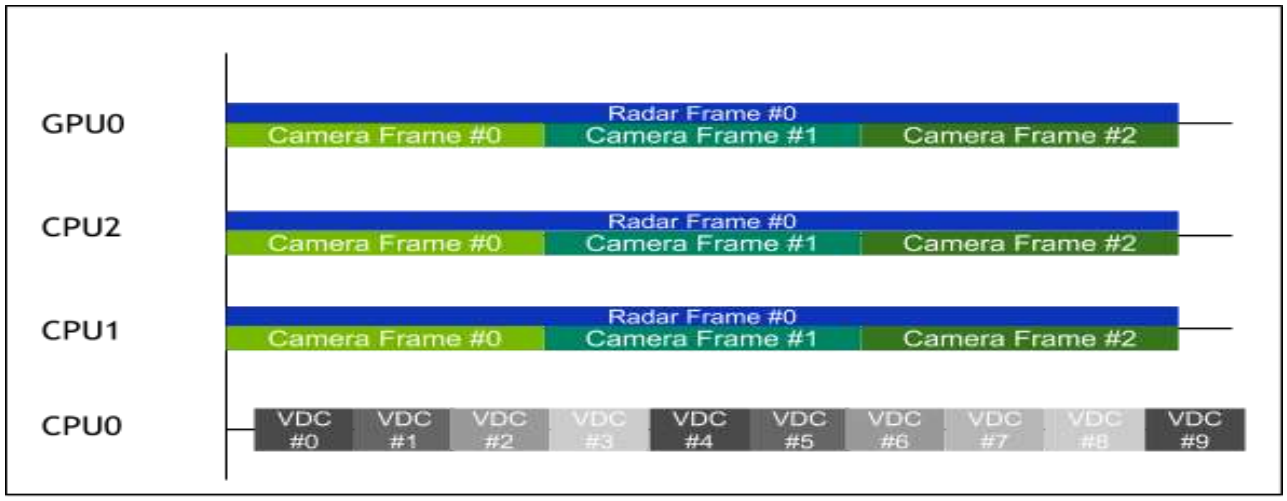

Figure 2 Visualization of the Hyperepoch configuration

3.1.2 Round Robinning of Runnables

The runnables inside an epoch can be round robinned, such that different runnables execute in the same slot for different frames.

Version: 2.0.0

Drive: # Graph ID

Resources: # Global Resources

```
CPU: [CPU0, CPU1, CPU2]
GPU: [GPU0]
Hyperepochs:
- Perception: # Hyperepoch ID
        Period: 100ms # Hyperepoch period
        Resources: [CPU1, CPU2, GPU0] # Resource mapping
        Epochs:
        - Camera: # Epoch ID
            AliasGroups:
                - parent_round_robin:
                        Steps: [client0.n2, client0.n4]
                - child_round_robin:
                        Steps: [client0.n3, client0.n5]
                Period: 33.33ms
                Frames: 3
        - Radar: # Epoch ID
                Period: 100ms
                Frames: 1
        - Control: # Hyperepoch ID; Hyperepoch
                Resources: [CPU0] # period inferred from epoch.
                Epochs:
                - VDC: # Epoch ID
                        Period: 10ms # Epoch frames = 1 (default)
...
```

In the above example, n2 and n3 will execute in even frames while n4 and n5 will execute in odd frames. These conditions must hold for steps:

1. The runnables must belong to the same client
2. The runnables must belong to same epoch

3. The runnables must have the same resources
4. Any other round robinned runnable that is a direct parent or child must have the same step length.

This section covered the skeleton for timing specification in the graph. In the following sections, we will cover the specification of tasks that adhere to these timing specs.

3.1.3 Clients

Hyperepochs and epochs define the timing boundaries for tasks (runnables). Clients define the data boundaries. A client is an operating system process that contains software resources (like CUDA streams) and runnables. Clients are specified in the graph specification section under the Clients header. Each client specifies contained software resources (if any). Clients also list the epochs contained in that client and runnables associated with each epoch. In general, a typical client would be specified as follows:

```
---
Version: 2.0.0
Drive: # Graph ID
Clients:
- Client0: # Client ID
    Resources: # Client0s internal resources
    # Resource Definition
    Epochs: # Epochs present in this client
    - Perception.Camera: # Epoch Global ID - <HyperepochID.EpochID>
        Runnables: # Runnables present in Perception.Camera
        - ReadCamera: # Runnable ID (Unique inside a client)
        # Runnable specification...
        - RunnableN:
        # Runnable specification...
    - Perception.Radar:
        Runnables: # Runnables present in Perception.Radar
        - ProcessRadar:
        # Runnable specification...
```

...

3.1.3.1 Client Resources

Clients can specify resources that are visible to runnables locally. These resources cannot be accessed by runnables in other clients. Global resources are visible to all runnables. Process-specific resources like CUDA streams, DLA handles, and PVA streams are some examples of client resources that cannot be shared across different clients. Also, internal scheduling mutexes can also be modeled here. These resources are specified in a format like that of Global Resources as specified in section 3.1.1.3.

Clients:

- Client1:

Resources:

ResourceType0:

- ResourceType0_Instance0

- ResourceType0_InstanceN

ResourceTypeN:

- ResourceTypeN_Instance0

- ResourceTypeN_InstanceN

Resource Type: CUDA Stream, DLA Handle, PVA Stream

CUDA streams, DLA handles, and PVA streams are client-specific software resources that are mapped to corresponding hardware engines (GPU, DLA and VPU respectively). To specify these resources, the resource types should be set to CUDA_STREAM, DLA_HANDLE, or PVA_STREAM respectively. The hardware engine mapping is conveyed to the compiler when specifying the resource instances as shown in the example below. The specified hardware resource instances should be specified under the corresponding hardware resource type in the Global Resources section. Note that the compiler will throw an error if the limits on the mapped resource (as specified in section 3.1.1.6) are violated.

Clients:

- Client0:

Resources:

CUDA_STREAM:

- CUDA_STREAM0: GPU0 # CUDA_STREAM0 mapped to GPU0

- CUDA_STREAM1: GPU0 # CUDA_STREAM1 mapped to GPU0

DLA_HANDLE:

- DLA_HANDLE0: DLA1 # DLA_HANDLE0 mapped to DLA1

PVA_STREAM: #A client can have one unique stream per VPU

- PVA_STREAM0: VPU0 # PVA_STREAM0 mapped to VPU0

Resource Type: Local Scheduling Mutex

Resource types other than those specified in section 3.1.1.3 above are treated as local scheduling mutexes. These cannot be mapped to a hardware resource.

Clients:

- Client0:

Resources:

LOCAL_SCHED_MUTEX:

- LOCAL_SCHED_MUTEX0

LOCAL_RESOURCE_MUTEX:

- RESOURCE_MUTEX0

3.1.4 Runnables

Tasks in the system executed on hardware engines are known as runnables. Ideally each runnable should only use a single hardware engine. Currently synchronous runnables (runnable that use multiple hardware engines simultaneously) are not supported. Runnables require resources for execution and can be dependent on other runnables. Care must be taken to ensure that dependencies do not introduce a cycle in the graph. Depending on the type of engine used, the compiler classifies each runnable into one of the three following classes:

1. **Submitter**: A runnable that runs on a CPU and submits another runnable that runs on a different hardware resource. E.g.: A CUDA kernel that launches work on the GPU.
2. **Submittee**: A runnable that is submitted by a submitter to be run on a particular hardware resource. Following the example above, the CUDA task launched on the GPU is termed as a Submittee runnable.
3. **Runnable**: Any task that cannot be classified as a Submitter or a Submittee.

For each runnable, the following parameters can be set:

1. **WCET (Worst Case Execution Time)**: The framework assumes that runnables have a bounded runtime. This runtime is captured by the WCET parameter. [Required Parameter]

2. **StartTime**: The start time of a runnable can be offset by this value from the beginning of its epoch.
3. **Deadline**: Suggested deadline to the compiler. The compiler will attempt to schedule this runnable before this deadline.
4. **Resources**: List of resources that a runnable needs. Currently, a runnable can request only one hardware resource. However, there are no limitations on the number of software resources that a runnable can ask for. The requirement can be specified as a resource type (e.g., GPU) if the developer does not care for a specific instance of a resource, or as a resource instance (e.g., GPU0). [Required Parameter]
5. **Dependencies**: List of runnables that need to be completed before this runnable can be scheduled. These runnables are specified as ClientID.RunnableID.
6. **Submits**: This field is present in Submitter runnable specification. It specifies the Submittee runnable ID. In a Submitter-Submittee pair, the Submits field should be populated. It is not necessary to add the Submitter runnable in the Submittees dependencies list.

```
Clients:
    - Client0:
        Resources:
        CUDA_STREAM:
        - CUDA_STREAM0: GPU0
        Epochs:
        - Perception.Camera: # Camera epoch in Perception Hyperepoch
            Runnables:
            - ReadCamera: # Normal runnable
                WCET: 10us
                Resources:
                - CPU # This runnable runs on a CPU
            - PreProcessImage: # Submitter runnable
                WCET: 20ms
                StartTime: 1ms # Starts 1ms after the camera epoch
                Resources: # GPU Submitter needs CPU0 and a stream
                - CPU0
```

```
            - CUDA_STREAM

            Dependencies: [Client0.ReadCamera] # Depends on ReadCamera

            Submits: Client0.PreProcessGPUWork # Mentions submittee

        - PreProcessGPUWork: # Submittee runnable

            WCET: 5000ns

            Deadline: 30ms # Hint to schedule this before 30ms

            Resources: [GPU]

            Dependencies:

                - Client0.PreProcessImage # Optional for submittees

            # Note: Inter-epoch dependencies are currently not

            # supported. Inter-client dependencies are supported.

- Perception.Radar: # Radar epoch in Perception Hyperepoch

    Runnables:

    - ProcessRadar:

    # Runnable specification...
```

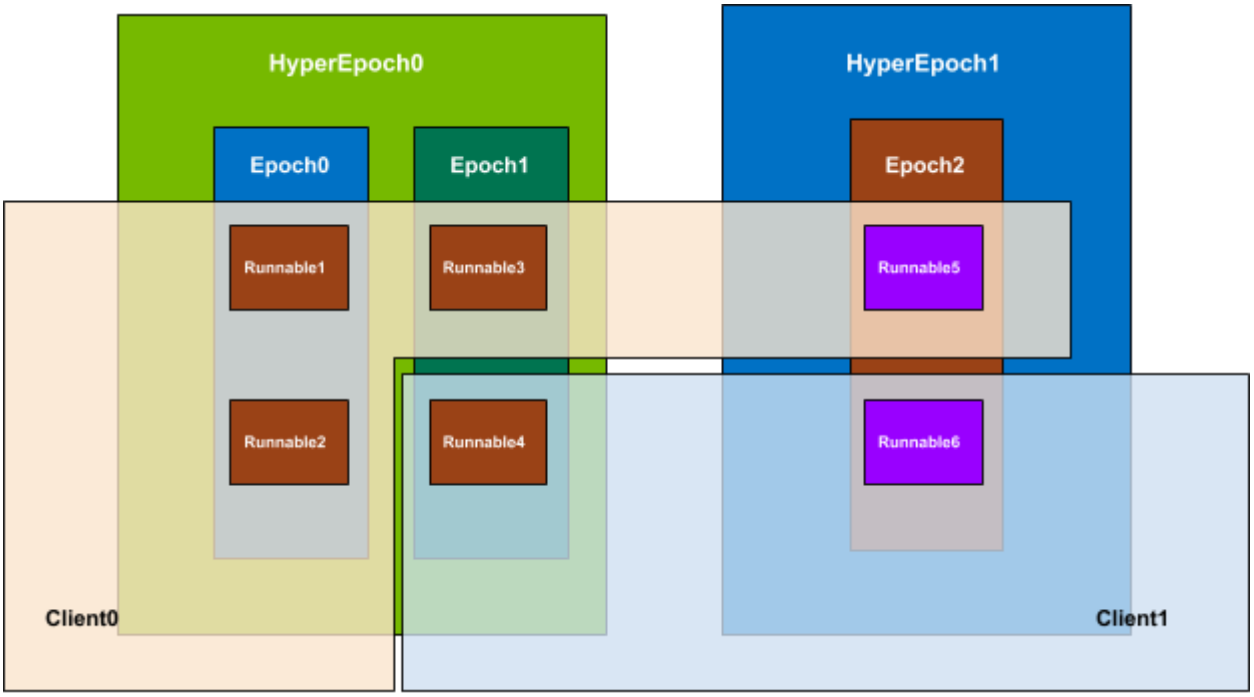

Figure 3 Boundaries in NVIDIA System Task Manager

3.1.5 Runnable Fusion

STM Compiler supports the fusion of CPU runnables. This helps in reducing the compilation time in the bigger graphs and reduces fences in cross CPU dependencies. This feature is enabled by default in the stmcompiler tool. To disable it in stmcompiler tool, use option nrf. Refer to the help section in stmcompiler tool for more.

3.1.6 Compiler Heuristics

The compiler currently supports two heuristics for flattening the graph into a schedule.

3.1.6.1 Branch and Bound (BNB)

This heuristic uses the branch and bound paradigm to find the solutions. With the default timeout of 0, the first found schedule is returned. This allows quick generation of schedules which are reproducible. Running again with same input yaml should produce the same schedule. For a longer and more extensive search, use a bigger timeout value. By default, one worker is spawned for each core in the system used for search. Use stmcompiler tool flags to change the number of workers (described in section 3.1.7).

3.1.6.2 One Active Runnable (OAR)

This heuristic is designed for isolated profiling of the workload. It creates a critical-path dominant sequence and schedules the sequence serially while putting in fences between sequential runnables. This ensures that only one runnable is active at any point in time on any hardware

engine. Note that profiling results from this heuristic will not uncover runnable-runnable performance interference in the system.

3.1.7 Stmcompiler tool

This script is present in x86 deb package in /usr/local/driveworks-5.0/tools/

The following are the flags which can be passed to stmcompiler

Variable	Flag	Description
ifile	-i	Input yml file
ofile	-o	Output .stm file
timeout	-t	Time-Out value for BNB (in seconds) per hyperepoch, default : 0. Note: Timeout Val 0 returns first schedule
num-workers	-j	Number of BNB workers searching in parallel, Default : Number of cores in system
max-contexts	-c	Max percentile threshold per hyperepoch for BNB to search schedule, default : 6 for timeout == 0
verbose	-v	Print verbose prints
export-kpis	-k	Export KPIs as a yaml file
version	-V	Print STM Compiler Version
no-runnable-fusion	-nrf	Disable Runnable Fusion in stmcompiler. Runnable fusion fuses the CPU runnables in the DAG if they are below the wcet_threshold parameter (default value is 500 us). Runnable fusion is enabled by default.
wcet-threshold	-wt	WCET threshold for Runnable fusion in the graph. STM compiler will fuse the CPU runnables in the DAG whose WCETs are less than the wcet threshold. Runnable fusion is enabled by default. The default wcet-threshold parameter is 500 us

Table 1 Stmcompiler command-line flags

3.2 Runtime

The STM runtime drives the execution of runnables across all NVIDIA DRIVE Orin™ SoC engines (including CPU) according to the schedule produced offline by the STM compiler. Unlike most schedulers, STM operates entirely in user space and is not part of any OS kernel. On x86, the engines are limited to CPU and dGPU.

3.2.1 Usage

3.2.1.1 Master

STM master is an executable that must be launched before any clients can return from a call to stmClientInit() and stmScheduleManagerInit(). The master executable gets passed the *.stm files produced by the compiler as mandatory command line arguments. It parses these files and creates internal schedule representation for each unique schedule ID. It then creates communication channels between the master process and the client processes for all the schedules. After all the clients yield to the scheduler by calling stmEnterScheduler(), the master process only becomes active when it receives a request to start/stop execution of a schedule. Only one schedule is allowed to execute at a time.

NOTE: Multiple schedule files passed as arguments to the STM Master must have unique Schedule Identifiers, otherwise STM will exit with an error.

3.2.1.2 Clients

A user can partition their application into multiple STM clients. Every STM client resides in a separate OS process and links to the shared library libstm runtime.so. Clients register resources and runnables with the libstm runtime.so, which creates a mapping between runnable/resource names and their process local addresses - function pointers for runnables and resource pointers (ex. CUDA streams, NvMedia device handles) for resources. After the registration phase, clients then yield to libstm runtime.so by calling stmEnterScheduler(). This function enters an infinite loop that schedules a process runnables via the previously registered function pointers periodically until termination.

To exit, clients can either pre-specify a maximum number of hyperepoch frames to execute as a parameter to stm master, or they can call stmExitScheduler() from within the application. For example, the application's SIGINT or SIGTERM handler might call stmExitScheduler() to gracefully exit from the blocking stmEnterScheduler() call.

STM clients follow the following initialization sequence:

- An external entity (the init system) starts the STM Master and all clients. STM does not require that the master and clients be started in any order.
- The STM Master will load and verify the schedule that was passed to it as a command line argument.
- Each client will initialize and register with the STM master. They each follow the following process:
    1. Once the client enters its main function, it performs any private initialization (e.g. allocating memory and loading configuration files.)
    2. The client initializes its STM client by calling stmClientInit().
    3. The client registers each of its runnables with STM by calling stmRegisterCpuRunnable(), stmRegisterCudaSubmitter(), stmRegisterDlaSubmitter() and stmRegisterVpuSubmitter().

4. If the client created references to hardware resources that STM interacts with (e.g., CUDAstreams, NvMedia handles) it registers those resources with STM by calling stmRegisterCudaResource(), stmRegisterDlaResource() and stmRegisterVpuResource().
5. Finally, the client calls stmEnterScheduler(). This hands over control to STM. After this call, the client may not allocate memory or new resources. This call will not return until the system is shut down.

- STM begins executing runnables according to the schedule.

3.2.1.3 Schedule Manager

The Schedule Manager resides in a single OS process and links to the shared library libstm_runtime.so. During system initialization, only one process can call stmScheduleManagerInit(). It controls which schedule is executed at a certain point of time. It provides two APIs to control the execution of the schedule:

1. stmStartSchedule(uint16 t scheduleId): This starts the execution of the schedule corresponding to the input scheduleId. If no matching schedule is found, it will result in an error and STM will exit.
2. stmStopSchedule(uint16 t scheduleId): This stops the execution of the schedule corresponding to the input scheduleId. If no matching schedule is found, or if the current executing schedule does not match the input scheduleId, it will result in an error and STM will exit.

To exit gracefully, the process calling stmScheduleManagerInit() can call stmScheduleManagerExit(). For example, the application's SIGINT or SIGTERM handler can call stmScheduleManagerExit() to gracefully exit.

3.2.1.4 Epoch Boundaries

All runnables in an epoch must complete before any runnable in the next epoch can begin. This is implemented via an additional, STM generated client called the framesync client. The framesync client waits on leaf node runnables to signal completion, and then sends a signal to root node runnables to begin executing in the next epoch. Given a schedule overrun, the STM runtime can operate in 2 modes: a frameskip mode and a free-running mode.

- Frameskip Mode

  In this mode, the framesync client will block the start of the next epoch until the next multiple of the epoch length. For example, if the epoch length is 100ms, and the last runnable in an epoch completes at t=350ms, the next epoch will start at t=400ms. If the epoch length is 100ms and the last runnable in an epoch completes at t=90ms, the next epoch will start at t=100ms.

- No-Frameskip-on-Overrun Mode

  In this mode, the next epoch will start as soon as all runnables in the previous epoch have finished, if the previous epoch over-ran its scheduled length.

3.2.1.5 Complete Swap Schedule Switch

STM implements complete schedule switch as depicted in the figure below. Upon receiving a request for stopping the current schedule, each hyperepoch finishes executing its current frame and does not start the execution of the next frame. Schedule execution comes to a halt after all the hyperepochs stop. After that, based on the input from the STM Schedule Manager, execution can restart with a different schedule.

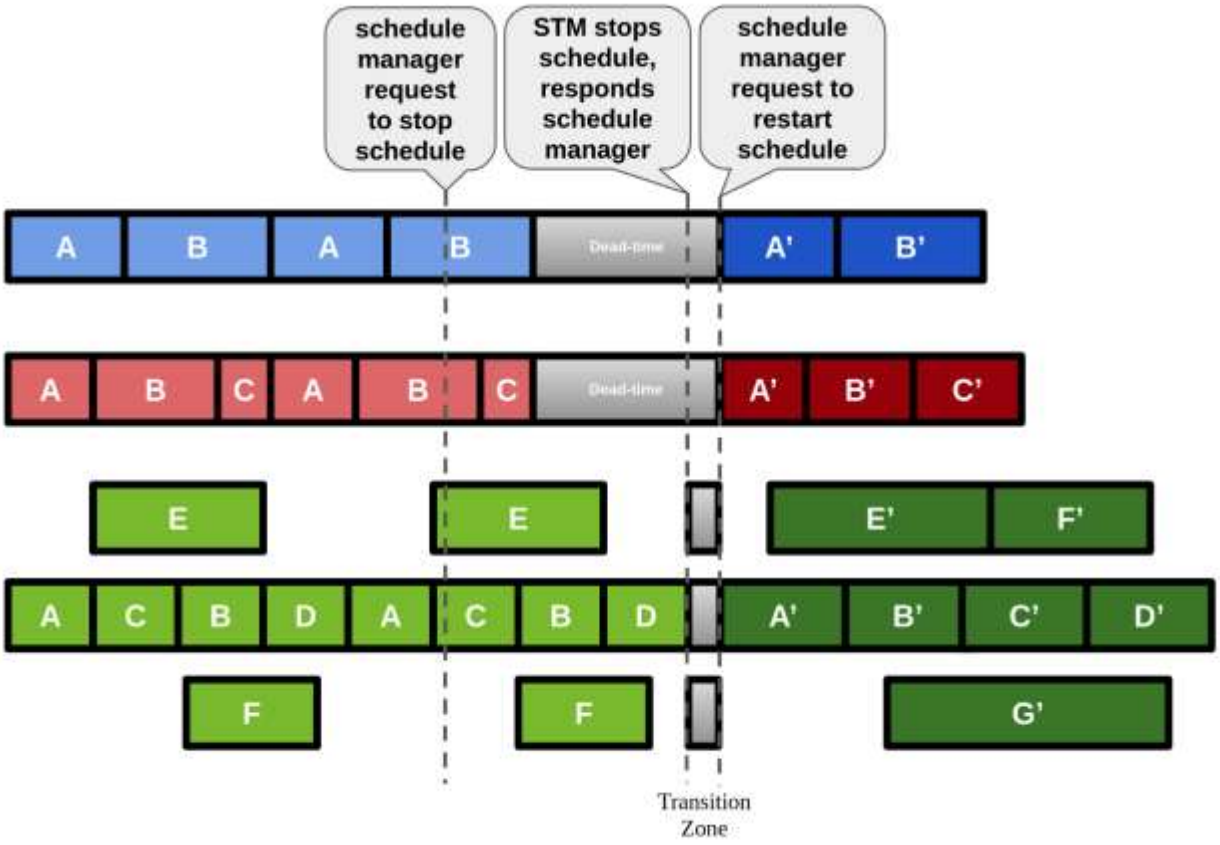

Figure 4 Overview of Schedule Switch

3.2.1.6 System Exit

The STM Schedule Manager is expected to stop the currently executing schedule using the schedule stop API. Upon receiving this request, each hyperepoch finishes executing its current frame and stops. Schedule execution comes to a halt after all the hyperepochs stop. Once schedule execution stops, the Schedule Manager can call the stmScheduleManagerExit() API for triggering

a system wide STM exit. After this, all the clients will exit the stmEnterScheduler() function and can start performing client exit tasks.

3.2.1.7 Public Interfaces (API)

The following APIs should only be accessed by clients. They can be accessed by including stm.h and linking to libstm runtime.so

- typedef void (*stmRunnable_t)(void* userdata);
  - Brief Function signature for STM CPU runnable.
  - stmRunnable_t functions should only contain CPU work. They should not call any blocking APIs or internally launch any threads.
- typedef void (*stmCudaSubmitter_t)(void* userdata, cudaStream_t stream);
  - Function signature for STM CUDA submitter runnable.
  - stmCudaSubmitter_t functions should contain some CPU work, followed by some number of asynchronous CUDA kernel launches. All kernels launched in submitter must be launched onto cudaStream_t stream. The use of stream 0 (the null stream) is prohibited due to CUDA's implicit synchronization semantics. This function should not internally launch any threads or call any blocking APIs, including cudaMemcpy(), cudaMemset(), cudaFree(), etc...
- STM_API void stmClientInit(const char* clientName);
  - Initialize STM client context.
  - This API must be called before any other STM APIs. This API will block until the STM master process is started. Client contexts should be cleaned up with stmClientExit() after STM has completed its execution.
- STM_API stmErrorCode_t stmRegisterCpuRunnable(stmRunnable_t func, const char* const runnableId, void* userdata);
  - Registers an stmRunnable_t function with the name provided by the user to the STM compiler.
  - param[in] func STM CPU runnable function pointer.o param[in] runnableId Name of CPU runnable provided to the STM compiler.o param[in] userdata Pointer to any data needed by the runnable.
  - return stmErrorCode_t, the completion code of the operation:
    - STM_ERROR_BAD_PARAMETER if runnableId exceeds STM_FUNC_NAME_MAX.
    - STM_ERROR_NOT_INITIALIZED if stmClientInit() has not been called
- STM_API stmErrorCode_t stmRegisterCudaSubmitter(stmCudaSubmitter_t func, const char* const runnableId, void* userdata);
  - Registers an stmCudaSubmitter_t function with the name provided by the user to the STM compiler

    - param[in] func STM CUDA submitter runnable function pointer
    - param[in] runnableId Name of CUDA submitter runnable provided to the STM compiler
    - param[in] userdata Pointer to any data needed by the runnable
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_BAD_PARAMETER if runnableId exceeds
        - STM_FUNC_NAME_MAX.§
        - STM_ERROR_NOT_INITIALIZED if stmClientInit() has not been called.
- STM_API stmErrorCode_t stmRegisterCudaResource(const char* resourceName, cudaStream_t stream);
    - Registers a CUDA stream resource.
    - param[in] resourceName Name of resource. Needs to match resource name passed to STM compiler
    - param[in] stream CUDA stream created by application
    - return stmErrorCode_t, the completion code of the operation
    - STM_ERROR_BAD_PARAMETER if any of the following occurs:
        - resourceName exceeds STM_RESOURCE_NAME_MAX
        - the number of resources registered exceeds STM_MAX_NUM_RESOURCES
        - stream is the NULL stream or has not been previously created by calling cudaStreamCreate()
        - the same cudaStream_t has already been registered (possibly with a different resourceName)
    - STM_ERROR_NOT_INITIALIZED if stmClientInit() has not been called.
- typedef void (*stmDlaSubmitter_t)(void* userdata, NvMediaDla* dla);
    - Function signature for STM DLA submitter runnable
    - stmDlaSumbitter_t functions should contain some CPU work, followed by some number of NvMediaDla submits. All submits must be launched on the provided NvMediaDla handle. These functions should not internally launch any threads or call any blocking APIs.
- STM_API stmErrorCode_t stmRegisterDlaResource(const char* resourceName, NvMediaDla* dla);
    - Registers a NvMediaDla resource
    - param[in] resourceName Name of resource. Needs to match resource name passed to STM compiler
    - param[in] stream DLA stream created by application
    - return stmErrorCode_t, the completion code of the operation:
    - STM_ERROR_BAD_PARAMETER if any of the following occurs:
        - resourceName exceeds STM_RESOURCE_NAME_MAX

        - the number of resources registered exceeds STM_MAX_NUM_RESOURCES
        - dla is NULL or has not been previously created by calling NvMediaDlaCreate()
        - the same NvMediaDla handle has already been registered (possibly with a different resourceName)
    - STM_ERROR_NOT_INITIALIZED if stmClientInit() has not been called.
- STM_API stmErrorCode_t stmRegisterDlaSubmitter(stmDlaSubmitter_t func, const char* const runnableId, void* userdata);
    - Registers an stmDlaSubmitter_t function with the name provided by the user to the STM compiler
    - param[in] func STM DLA submitter runnable function pointer
    - param[in] runnableId Name of DLA submitter runnable provided to the STM compiler
    - param[in] userdata Pointer to any data needed by the runnable
    - return stmErrorCode_t, the completion code of the operation
        - STM_ERROR_BAD_PARAMETER if runnableId exceeds STM_FUNC_NAME_MAX
        - STM_ERROR_NOT_INITIALIZED if stmClientInit() has not been called.
- typedef void (*stmVpuSubmitter_t)(void* userdata, cupvaStream_t vpu);
    - Function signature for STM VPU submitter runnable
    - stmVpuSubmitter_t functions should contain some CPU work, followed by some number of VPU submits. All submits must be launched on the provided VPU stream. These functions should not internally launch any threads or call any blocking APIs.
- STM_API stmErrorCode_t stmRegisterVpuResource(const char* resourceName, cupvastream_t vpu);
    - Registers a VPU resource
    - param[in] resourceName Name of resource. Needs to match resource name passed to STM compiler
    - param[in] VPU stream created by application
    - return stmErrorCode_t, the completion code of the operation:
    - STM_ERROR_BAD_PARAMETER if any of the following occurs:
        - resourceName exceeds STM_RESOURCE_NAME_MAX
        - the number of resources registered exceeds STM_MAX_NUM_RESOURCES
        - vpu is NULL or has not been previously created
        - the same vpu stream has already been registered (possibly with a different resourceName)
    - STM_ERROR_NOT_INITIALIZED if stmClientInit() has not been called.

- STM_API stmErrorCode_t stmRegisterVpuSubmitter(stmVpuSubmitter_t func, const char* const runnableId, void* userdata);
    - Registers an stmVpuSubmitter_t function with the name provided by the user to the STM compiler
    - param[in] func STM VPU submitter runnable function pointer
    - param[in] runnableId Name of VPU submitter runnable provided to the STM compiler
    - param[in] userdata Pointer to any data needed by the runnable
    - return stmErrorCode_t, the completion code of the operation
        - STM_ERROR_BAD_PARAMETER if runnableId exceeds STM_FUNC_NAME_MAX
        - STM_ERROR_NOT_INITIALIZED if stmClientInit() has not been called.
- STM_API stmErrorCode_t stmEnterScheduler(void);
    - Yield to STM scheduler, which will schedule all registered runnables in the STM schedule periodically until completion. This function will not return until a runnable calls stmExitScheduler()or the number of epochs specified in the STM master process have completed
    - All runnables and resources need to be registered prior to entering the schedule. This API will only begin scheduling work after all STM clients have started. This API can only be called once per call to stmClientInit()
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_TIMEOUT if any fence timed out.
        - STM_ERROR_GENERIC if there was another error.
        - STM_SUCCESS otherwise.
- STM_API stmErrorCode_t stmGetRuntimeInfo(stmRunnableInfo_t* info);
    - Get runtime information about executiono Must be called from within a CPU runnable
    - param[out] info Pointer to stmRunnableInfo_t whose fields will be populated
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_BAD_PARAMETER if info is NULL
        - STM_ERROR_NOT_FOUND if the current STM thread could not be found.
        - STM_ERROR_INVALID_STATE if the requisite information could not be obtained
        - STM_SUCCESS otherwise.
- STM_API stmErrorCode_t stmExitScheduler(void);
    - Return from stmEnterScheduler()
    - Called from within a runnable; causes all clients to return from their respective calls to stmEnterScheduler() at the next epoch boundary. Only needs to be called from one STM client; subsequent calls are ignored

    - return stmErrorCode_t, the completion code of the operation:
        - STM_SUCCESS if successful
        - STM_ERROR_GENERIC if any client process died or if any APIs in any STM client returned an error code.
- STM_API void stmClientExit(void);
    - Cleans up STM client context. No STM APIs can be called after this
    - This API can only be called once per call to stmClientInit(); doing so multiple times will cause undefined behavior.

The following APIs can be accessed by including stm_manager.h.

- STM_API stmErrorCode_t stmScheduleManagerInit(const char* scheduleManagerName);
    - This API must be called before any other STM Schedule Management APIs. This API will block until the STM master process is started. Manager's context should be cleaned up with stmScheduleManagerExit() after STM has completed its execution. There is only one schedule manager in the system.
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_INSUFFICIENT_MEMORY : failure of memory allocation of STM.
        - STM_ERROR_BAD_VALUE : unsupported value for discriminator.
        - STM_ERROR_NOT_SUPPORTED : connection to master not enabled. ignore this error for dual SOC case and soc is secondary
        - STM_ERROR_GENERIC : Failed to receive messages from master.
- STM_API stmErrorCode_t stmScheduleManagerInitWithDiscriminator(const char* scheduleManagerName, int32_t discriminator);
    - This API must be called before any other STM Schedule Management APIs. This API will block until the STM master process with this discriminator is started. Manager's context should be cleaned up with stmScheduleManagerExit() after STM has completed its execution. Manager must call at most one of the two: stmScheduleManagerInit() or stmScheduleManagerInitWithDiscriminator(),if both are called single or multiple times, it will cause undefined behavior. There is only one schedule manager in the system. All negative values of discriminator are equivalent and the same as calling stmScheduleManagerInit()
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_INSUFFICIENT_MEMORY : failure of memory allocation of STM.
        - STM_ERROR_BAD_VALUE : unsupported value for discriminator.
        - STM_ERROR_NOT_SUPPORTED : connection to master not enabled. ignore this error for dual SOC case and soc is secondary
        - STM_ERROR_GENERIC : Failed to receive messages from master.

- STM_API stmErrorCode_t stmScheduleManagerExit(void);
    - Cleans up STM schedule Manager context. No STM APIs can be called after this.
    - This API can only be called once per call to stmScheduleManagerInit(); doing so multiple times will cause undefined behavior.
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_NOT_INITIALIZED if not init api was called or called but failed.
        - STM_ERROR_GENERIC if fail to release some OS resources. check debug logs for further information
- STM_API stmErrorCode_t stmScheduleManagerExit(void);
    - This API must be called between stmScheduleManagerInit()/stmScheduleManagerInitWithDiscriminator() and stmScheduleManagerExit().
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_INSUFFICIENT_MEMORY : failure of memory allocation of STM.
        - STM_ERROR_BAD_STATE_TRANSITION : If current state is not READY
        - STM_ERROR_BAD_VALUE : unsupported value for discriminator.
        - STM_ERROR_NOT_SUPPORTED : connection to master not enabled. ignore this error for dual SOC case and soc is secondary
        - STM_ERROR_GENERIC : Failed to receive messages from master.
- STM_API stmErrorCode_t stmStopSchedule(uint16_t scheduleId);
    - This API must be called between stmScheduleManagerInit()/stmScheduleManagerInitWithDiscriminator() and stmScheduleManagerExit().
    - return stmErrorCode_t, the completion code of the operation:
        - STM_ERROR_INSUFFICIENT_MEMORY : failure of memory allocation of STM.
        - STM_ERROR_BAD_STATE_TRANSITION : If current state is not READY
        - STM_ERROR_BAD_VALUE : unsupported value for discriminator.
        - STM_ERROR_NOT_SUPPORTED : connection to master not enabled. ignore this error for dual SOC case and soc is secondary
        - STM_ERROR_GENERIC : Failed to receive messages from master.

3.2.1.8 stm_master

The following are the flags which can be passed to stm_master

Variable	Flag	Description
schedule	-s	Path to .stm schedule Files. Multiple Files separated by comma.
schedule-manager-name	-N	Name of the schedule manage which will connect to stm master
num-input-schedule	-i	Number of input schedule provided. This argument should be used if schedule manager is set
log	-l	Path to which the (binary) log file will be output. No log is generated otherwise.
master-forked-log-path	-L	Path to which any log files from master-forked processes (i.e. framesync and deadlock resolvers) will be outputted
soc	-x	The name of the SOC on which this stm_master is running (Must match SOC fields in yaml)
log-freq	-f	Number of times (per second) log should be written to (default 30/s)
timeout-us	-t	NvSciSync fence wait timeout in us (-1 for infinite timeout; default 5000000us timeout)
verbose	-v	Print verbose debug information. Note that this mode affects latency guarantees and should only be used when debugging.
no-skip-on-overrun	-m	Sets policy to not wait for next period trigger on overruns.
select-ptp-clk	-k	Selects which clock source to get timestamps from, during printing debug information.
epochs	-e	Number of hyperepochs for which to run schedule
core	-c	Index of CPU core on which to pin stm_master
discriminator	-D	Specify a discriminator if running multiple STM instances on a single machine. All clients connecting to a particular instance should have the same discriminator as specified in its corresponding Master.
Block-logging	-b	Block when logging buffer is full. Default is to drop events until buffer space is available
enable-memlock	-M	Page lock all client process memory. Default is false.
disable-cpu-affinity	-C	Disables pinning of STM threads to scheduled cores.
version	-V	Print STM version and exit

Table 2 stm_master command-line flags

3.3 Analytics

STM Analytics generates an analysis report from an execution of STM Runtime. It uses compiler generated schedule file (*.stm) and runtime generated system log file (filename set at runtime). STM Analytics generates a serialized data file which consists of statistics and timing information about all clients and runnables in the system.

3.3.1 stmanalyze

stmanalyze generates an internal representation of schedule and runtime logs. After that, it serializes the internal representation into a *pkl format. This format can be used to combine multiple *pkl files. It can be used by external visualizer for visualization.

3.3.1.1 Input

Variable	Flag	Description
schedule	-s	List of input schedules used during execution. Generated by stmcompiler.
log-file	-l	Input logger file. Generated from a STM Runtime run, logs from different schedules can be present.
pickled-results	-p	Input pickled results from a previous run. Use this option to re-run the verifier on already-processed data.
Out-file	-o	Generated output base file name.
Ram-limit	-R	The maximum amount of RAM to use when processing (in GB).
verbose	-v	Print verbose runtime information.
output-format	-f	The output format of the stm analytics script. Chose one or more of ”pickle”, ”html” or ”csv”.

Table 3 stmanalyze input command-line flags

3.3.1.2 Output

Variable	Flag	Description
out-file	-o	Generated output pickle (.pkl) file name.

Table 4 stmanalyze output command-line flags

4 Samples

The following are the instructions to run sample files:

4.1 Pre-requisites

1. Make sure that /etc/nvsciipc.cfg on target contains the entries in /usr/local/driveworks/targets/aarch64-Linux/config/nvsciipc.cfg (can append to existing /etc/nvsciipc.cfg file if they are not present). Please reboot the system after this step.
2. Mqueue length of at least 4096 needs to be supported. On Linux, do either of the following:
   i. Change the contents of file /proc/sys/fs/mqueue/msg_max to 4096 (does not persist across reboots).
   ii. Add fs.mqueue.msg_max = 4096 to /etc/sysctl.conf and restart (persists across reboot)
3. Besides the minimum mqueue number of messages, the total mqueue size (ulimit -q) needs to be increased since this build uses larger sized messages. Either run as sudo or add these lines to /etc/security/limits.conf

<user> soft msgqueue unlimited

<user> hard msgqueue unlimited

Allows the <user> (change it to appropriate name) to have unlimited sized mqueue

4.2 Run Sample Binaries and Sources

To run the sample binaries directly on x86:

- ps -ef | grep -e framesync -e stm_ | grep -v grep | awk '{print $2}' | xargs -rt sudo kill -s KILL || true
- sudo rm -rf /dev/shm/* /dev/mqueue/*
    - Note: The above command must be run if PDK < 6.0.5.0 only
- export CUDA_VISIBLE_DEVICES=1
- export LD_LIBRARY_PATH=/usr/local/driveworks/targets/x86_64-Linux/lib:/usr/local/cuda-11.4/lib:/usr/local/cuda-11.4/lib64:$LD_LIBRARY_PATH

Commands for each sample:

- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/bin/cpu_simple.stm -l x.log -e 50 & sudo /usr/local/driveworks/bin/stm_test_cpu_simple
- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/bin/gpu_multistream_multiprocess.stm -l x.log -e 50 & sudo /usr/local/driveworks/bin/stm_test_gpuX & sudo /usr/local/driveworks/bin/stm_test_gpuY

Commands for Schedule Switch Sample

- STM packages a sample schedule manager in the release. This sample schedule manager switches between schedule IDs 101 and 102 (cpu_gpu1.stm and cpu_gpu2.stm respectively) to demonstrate the schedule switch functionality. Execute the following commands in order and in different terminals:
    - Run the stm_master along with list of schedules:
        - sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu1.stm,/usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu2.stm -l x.log -e 500 -i 2 -N default
    - Run the test schedule manager binary:
        - sudo /usr/local/driveworks/bin/stm_sample_manager default -v
    - Run client binaries
        - sudo /usr/local/driveworks/bin/stm_sample_gpuX & sudo /usr/local/driveworks/bin/stm_sample_gpuY
- Each cycle of execution has 1 schedule switch (one switch between the two schedules passed as input to stm_master) and by default the schedules will switch with a time

period of 1000 milliseconds. There should be 10 cycles of execution for the above commands.The schedule switches can be seen in the logs of `stm_sample_manager`. Use `-v` with `stm_sample_manager` for verbose outputs.

To run the sample binaries directly on target:

- ps -ef | grep -e framesync -e stm_ | grep -v grep | awk '{print $2}' | xargs -rt sudo kill -s KILL || true
- sudo rm -rf /dev/shm/* /dev/mqueue/*
    - Note: The above command must be run if PDK < 6.0.5.0 only
- export CUDA_VISIBLE_DEVICES=1
- export LD_LIBRARY_PATH=/usr/local/driveworks/targets/aarch64-Linux/lib:/usr/local/cuda-11.4/lib:/usr/local/cuda-11.4/lib64:$LD_LIBRARY_PATH

Commands for each sample on target:

- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/bin/cpu_simple.stm -l x.log -e 50 & sudo /usr/local/driveworks/bin/stm_test_cpu_simple
- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/bin/gpu_multistream_multiprocess.stm -l x.log -e 50 & sudo /usr/local/driveworks/bin/stm_test_gpuX & sudo /usr/local/driveworks/bin/stm_test_gpuY
- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/bin/dla_simple.stm -l x.log -e 50 & sudo /usr/local/driveworks/bin/stm_test_dla
- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/bin/vpu_simple.stm -l x.log -e 50 & sudo /usr/local/driveworks/bin/stm_test_vpu
  (Note: The vpu_simple app is only available for PDKs 6.0.4.0+ and requires the presence of cuPVA SDK v2.0.0 libraries)

Commands for Schedule Switch Sample on target

- STM packages a sample schedule manager in the release. This sample schedule manager switches between schedule IDs 101 and 102 (cpu_gpu1.stm and cpu_gpu2.stm respectively) to demonstrate the schedule switch functionality. Execute the following commands in order and in different terminals:
    - Run the stm_master along with list of schedules :
        - sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu1.stm,/usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu2.stm -l x.log -e 500 -i 2 -N default
    - Run the test schedule manager binary:
        - sudo /usr/local/driveworks/bin/stm_sample_manager default -v

    - Run client binaries
        - sudo /usr/local/driveworks/bin/stm_sample_gpuX & sudo /usr/local/driveworks/bin/stm_sample_gpuY
- Each cycle of execution has 1 schedule switch (one switch between the two schedules passed as input to stm_master) and by default the schedules will switch with a time period of 1000 milliseconds. There should be 10 cycles of execution for the above commands.The schedule switches can be seen in the logs of `stm_sample_manager`. Use `-v` with `stm_sample_manager` for verbose outputs.

To use the tools given by STM on x86:

- STMCompiler : /usr/local/driveworks/tools/stmcompiler -i /path/to/input_file.yml -o /path/to/output_file.stm
- STMVizschedule : /usr/local/driveworks/tools/stmvizschedule -i /path/to/input_file.stm -o /path/to/output_file.html
- STMVizGraph : /usr/local/driveworks/tools/stmvizgraph -i /path/to/input_file.yml -o /path/to/output_file.svg

  NOTE: Needs GraphViz installed on the system (sudo apt install graphviz)

- STM Analytics: /usr/local/driveworks/tools/stmanalyze -s /path/to/input_file.stm -l /path/to/log_file -f html

  NOTE: The log file is obtained after running the sample binaries above.

To compile and run samples from src, follow these steps:

cd /usr/local/driveworks/samples/src/stm/src/

STM Compiler Step:

- /usr/local/driveworks/tools/stmcompiler -i test_cpu_gpu_simple/gpu_multistream_multiprocess.yml -o gpu_multistream_multiprocess.stm
- /usr/local/driveworks/tools/stmcompiler -i test_cpu_simple/cpu_simple.yml -o cpu_simple.stm
- /usr/local/driveworks/tools/stmcompiler -i test_dla_simple/dla_simple.yml -o dla_simple.stm
- /usr/local/driveworks/tools/stmcompiler -i test_vpu_simple/vpu_simple.yml -o vpu_simple.stm

STM Runtime Step:

NOTE: For cross compilation, ensure that driveworks_stm_cross.deb is installed

- cd /usr/local/driveworks/samples/src/stm/src/
- mkdir stm-build & cd stm-build
- To cross-compile: cmake -DCMAKE_BUILD_TYPE=Release .. -DCMAKE_TOOLCHAIN_FILE=cmake/Toolchain-V5L.cmake -DVIBRANTE_PDK:STRING=/drive/drive-t186ref-linux -DCUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda -DSTM_BASE_DIR=/usr/local/driveworks/targets/aarch64-Linux/ -DVIBRANTE_PDK_FOUNDATION:STRING=/drive/drive-t186ref-foundation
- For x86: cmake -DCMAKE_BUILD_TYPE=Release .. -DCUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda -DSTM_BASE_DIR=/usr/local/driveworks/targets/x86_64-Linux/
- make install -j <number of jobs>

To run the built samples on x86:

- ps -ef | grep -e framesync -e stm_ | grep -v grep | awk '{print $2}' | xargs -rt sudo kill -s KILL || true
- sudo rm -rf /dev/shm/* /dev/mqueue/*
    - Note: The above command must be run if PDK < 6.0.5.0 only
- export CUDA_VISIBLE_DEVICES=1
- export LD_LIBRARY_PATH=/usr/local/driveworks/targets/x86_64-Linux/lib:/usr/local/cuda-11.4/lib:/usr/local/cuda-11.4/lib64:$LD_LIBRARY_PATH

Commands for each sample on x86:

- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/cpu_simple.stm -l x.log -e 50 & sudo /usr/local/driveworks/samples/src/stm/src/stm-build/test_cpu_simple/client/stm_test_cpu_simple
- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/gpu_multistream_multiprocess.stm -l x.log -e 50 & sudo /usr/local/driveworks/samples/src/stm/src/stm-build/test_cpu_gpu_simple/clientX/stm_test_gpuX & sudo /usr/local/driveworks/samples/src/stm/src/stm-build/test_cpu_gpu_simple/clientY/stm_test_gpuY

Commands for Schedule Switch Sample on x86

- STM packages a sample schedule manager in the release. This sample schedule manager switches between schedule IDs 101 and 102 (cpu_gpu1.stm and cpu_gpu2.stm

respectively) to demonstrate the schedule switch functionality. Execute the following commands in order and in different terminals:

- Run the stm_master along with list of schedules :
  - sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu1.stm,/usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu2.stm -l x.log -e 500 -i 2 -N default
- Run the test schedule manager binary:
  - sudo /usr/local/driveworks/bin/stm_sample_manager default -v
- Run client binaries
  - sudo /usr/local/driveworks/bin/stm_sample_gpuX & sudo /usr/local/driveworks/bin/stm_sample_gpuY

- Each cycle of execution has 1 schedule switch (one switch between the two schedules passed as input to stm_master) and by default the schedules will switch with a time period of 1000 milliseconds. There should be 10 cycles of execution for the above commands.The schedule switches can be seen in the logs of `stm_sample_manager`. Use `-v` with `stm_sample_manager` for verbose outputs.

To run the built samples on Target:

NOTE: Rsync the built samples to the equivalent folder in Target

- ps -ef | grep -e framesync -e stm_ | grep -v grep | awk '{print $2}' | xargs -rt sudo kill -s KILL || true
- sudo rm -rf /dev/shm/* /dev/mqueue/*
- export CUDA_VISIBLE_DEVICES=1
- export LD_LIBRARY_PATH=/usr/local/driveworks/targets/aarch64-Linux/lib:/usr/local/cuda-11.4/lib:/usr/local/cuda-11.4/lib64:$LD_LIBRARY_PATH

Commands for each sample:

- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/cpu_simple.stm -l x.log -e 50 & sudo /usr/local/driveworks/samples/src/stm/src/stm-build/test_cpu_simple/client/stm_test_cpu_simple
- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/gpu_multistream_multiprocess.stm -l x.log -e 50 & sudo /usr/local/driveworks/samples/src/stm/src/stm-build/test_cpu_gpu_simple/clientX/stm_test_gpuX & sudo /usr/local/driveworks/samples/src/stm/src/stm-build/test_cpu_gpu_simple/clientY/stm_test_gpuY

- sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/dla_simple.stm -l x.log -e 50 & sudo /usr/local/driveworks/samples/src/stm/src/stm-build/test_dla_simple/dla_simple/stm_test_dla

Commands for Schedule Switch Sample on target

- STM packages a sample schedule manager in the release. This sample schedule manager switches between schedule IDs 101 and 102 (cpu_gpu1.stm and cpu_gpu2.stm respectively) to demonstrate the schedule switch functionality. Execute the following commands in order and in different terminals:
    - Run the stm_master along with list of schedules:
        - sudo /usr/local/driveworks/bin/stm_master -s /usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu1.stm,/usr/local/driveworks/samples/src/stm/src/sample_complete_swap/cpu_gpu2.stm -l x.log -e 500 -i 2 -N default
    - Run the test schedule manager binary:
        - sudo /usr/local/driveworks/bin/stm_sample_manager default -v
    - Run client binaries
        - sudo /usr/local/driveworks/bin/stm_sample_gpuX & sudo /usr/local/driveworks/bin/stm_sample_gpuY
- Each cycle of execution has 1 schedule switch (one switch between the two schedules passed as input to stm_master) and by default the schedules will switch with a time period of 1000 milliseconds. There should be 10 cycles of execution for the above commands.The schedule switches can be seen in the logs of `stm_sample_manager`. Use `-v` with `stm_sample_manager` for verbose outputs.

Use the stmanalyze tool given by STM on x86, to obtain the final performance of the logs produced by these steps.

4.3 Sample Schedule Manager App

This application can be used as a reference for implementing the schedule manager for different client applications. It can be configured to perform different schedule switch scenarios. It can go through each schedule in the following way.

- choose a schedule
- starts executing the schedule
- runs the schedule for provided interval
- stops the schedule
- switches to the next schedule

4.3.1 Schedule Manager Usage

Schedule Manager takes the following command line parameters:

- Mandatory Arguments:
    - The name of the Schedule Manager which needs to connect to the STM Master (The same name must be given to the STM Master as an input)
    - Path of the input yaml file for reading schedule Ids
- Optional Arguments:
    - -v : verbose
    - -p : Specify the period (ms) for which schedules are executed
    - -g : Specify the break/delta (ms) between the end of the previous execution and start of the next execution
    - -r : Maximum schedule index to reach before resetting
    - -c : Number of times the execution cycle is to be repeated
    - -x : allow exceptions during stopping and starting the schedule

Start STM Master , and then simply use:

./stm_test_manager <schedule manager name> <path/to/yaml1.yml>, <path/to/yaml2.yml> ....

5 Tools

There are two additional tools provided by STM.

5.1 VizGraph

VizGraph is used to visualize the graph in the yaml file in the form of svg. This tool can be found at: /usr/local/driveworks-5.0/tools/stmvizgraph . These are the options it supports:

Variable	Flag	Description
ifile	-i	Path to input yaml file
ofile	-o	Path to output svg file

Table 5 stmvizgraph command-line flags

NOTE: This needs the installation of Graphviz on the system.

5.2 VizSchedule

Vizschedule facilitates the visualization of the schedule generated by the STM Compiler (.stm file) in the html form. This tool can be found at: /usr/local/driveworks-5.0/tools/stmvizschedule . These are the options it supports:

Variable	Flag	Description
ifile	-i	Path to input stm file
ofile	-o	Path to output html file

Table 6 stmvizschedule command-line flags

6 Open-Source and Third-Party Licenses

6.1 Pyyaml

6.2 Bison

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

As a special exception, you may create a larger work that contains part or all of the Bison parser skeleton and distribute that work under terms of your choice, so long as that work isn't itself a parser generator using the skeleton or a modified version thereof as a parser skeleton. Alternatively, if you modify or redistribute the parser skeleton itself, you may (at your option) remove this special exception, which will cause the skeleton and the resulting Bison output files to be licensed under the GNU General Public License without this special exception.

This special exception was added by the Free Software Foundation in version 2.2 of Bison.

6.3 Bokeh

Copyright (c) 2012 - 2021, Anaconda, Inc., and Bokeh Contributors

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Anaconda nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.4 Flex

This code is derived from software contributed to Berkeley by Vern Paxson.

The United States Government has rights in this work pursuant to contract no. DE-AC03-6SF00098 between the United States Department of Energy and the University of California.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

6.5 GraphViz

Copyright (c) 2013-2021 Sebastian Bank

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software isfurnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.6 InfluxDB

Copyright (c) 2020 InfluxDB

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.7 IntervalTree

6.8 Jinja2

PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.9 Numpy

Copyright (c) 2005-2021, NumPy Developers.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the NumPy Developers nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ” AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.10 Orderedset

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of Ordered Set nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Copyright (c) 2009 Raymond Hettinger

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHE LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.11 Pandas

Copyright (c) 2008-2011, AQR Capital Management, LLC, Lambda Foundry, Inc. and PyData Development Team

All rights reserved.

Copyright (c) 2011-2021, Open-source contributors.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.12 Plotly

The MIT License (MIT)

Copyright (c) 2016-2018 Plotly, Inc

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR

OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.13 Ply

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the David Beazley or Dabeaz LLC may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.14 PyTest-Cov

The MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.15 PyTest

Copyright Holger Krekel and others, 2004-2021.

Distributed under the terms of the MIT license, pytest is free and open source software.

6.16 Tabulate

Copyright (c) 2011-2020 Sergey Astanin and contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.