TaskWorks: A Task Engine for Empowering Asynchronous Operations in HPC Applications

Kaiyuan Hou[1,2], Quincey Koziol[2], and Suren Byna[2]

[1]Northwestern University, [2]Lawrence Berkeley National Laboratory

ABSTRACT

TaskWorks is a portable, high-level, task engine designed for HPC workloads. Applications can create tasks and define dependencies between them with the task engine. Once the task is defined and submitted to TaskWorks, the TaskWorks engine will execute it according to the specified dependencies, without additional input from the application. TaskWorks has an integrated event manager monitors files, sockets, timers, and MPI operations, and these events can be used as task dependencies. TaskWorks is compatible with MPI and is designed to work efficiently with HPC applications that perform inter-process (node) communication.

REQUIREMENTS

- Support asynchronous operations, and dependencies between operations
- Ease-of-use is important
- Must be HPC-friendly
- Portable to POSIX and Win32 systems

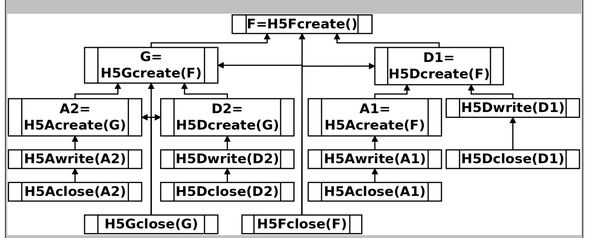

Figure 1: Dependencies in HDF5 operations. (For clarity, dependencies already covered by other dependent operations is not shown)

MOTIVATION

- Programming low-level thread APIs is tedious and error-prone
- Desire for enhanced task dependency logic
- Existing high-level task libraries have a complex API and programming model that is difficult to learn
- A standalone library can benefit many other applications

RELATED WORK

Feature	TaskWorks	Argobots	OpenMP	Pthreads
Task (high-level) API	Y	Y	Y	N
Thread (low-level) API	N	Y	Y	Y
Cross-Platform	Y	N	Y	N
Task Dependencies	Y	Y	Y	N
Custom Dependency Logic	**Y**	N	N*	N
Threadless (polled) Execution	**Y**	N	N	N
Event Loop Integration	**Y**	N	N	N
Task Priority	**Y**	N	Y	Y
MPI Compatibility	Y	Y	Y	Y
PMIx Integration	Y	N	Y	N

Table 1: Feature comparison of TaskWorks and other libraries. TaskWorks specific feature is highlighted in bold. * The OpenMP 5.1 standard expected to arrive in late 2020 will support custom dependency logic.

DESIGN GOALS

- User-friendly, easy to learn API
  - *Post-and-forget* task management
  - Hide complexity of low-level thread APIs
  - Support asynchronous operations in existing applications with minimal effort
- Portable and Adaptable
  - Cross-platform
  - Pluggable interface to interact with various backends
- Flexible
  - Custom task dependencies
  - Monitor events as well as run tasks
  - Fits a wide range of applications
- HPC-friendly
  - MPI-compatible
  - Coordinate with other HPC packages
  - Interaction of tasks between nodes

DESIGN

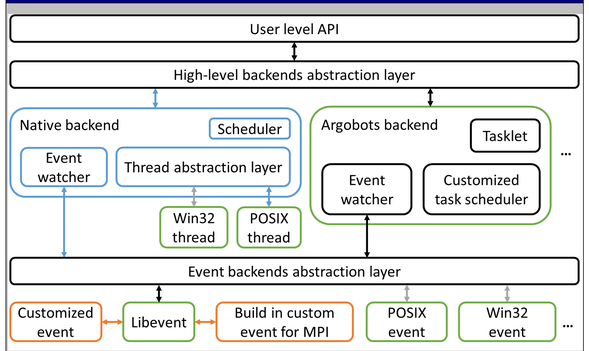

Figure 2: Architecture of TaskWorks. TaskWorks components are outlined in blue. Pluggable interfaces are highlighted in orange.

ENGINES

- Run tasks and events
- Manage workers (threads)
- Abstraction layer for both high -and low- level backends
  - OS threads, Argobots, ...
- Can be created without worker threads
  - Act as a task manager
- Multiple engines can run in parallel
  - Dedicate a different number of threads for different types of tasks

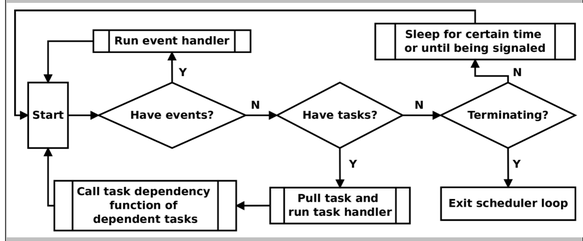

Figure 3: Task scheduler

TASKS

- Task function and metadata
- Dependency information
  - Form an implicit workflow graph
  - Can depend on tasks handled by other engines and other processes
- Barrier task - finishes when all tasks created before it finish
- Customized task dependency
  - User-defined dependency logic
  - Can start even when some dependent task hasn't completed
  - Allows cycles in the workflow graph

EVENTS

- Supported event types
  - File – Modify, delete, etc. operations
  - Socket – incoming traffic
  - Timer – once or recurring
  - MPI – incoming message
  - Customized – polling on a user defined function
- Interaction with tasks
  - Spawn tasks on events
  - Tasks can depend on events
- Share the same set of workers with tasks

USING TASKWORKS

```c
hid_t fid, gid;
int task1_fn (void *arg) {
    fid = H5Fopen ("example.h5", H5F_ACC_RDWR, H5P_DEFAULT);
    return (int)(fid < 0);
}
int task2_fn (void *arg) {
    gid = H5Gopen (fid, "group", H5P_DEFAULT);
    return (int)(gid < 0);
}
int task3_fn (void *arg) {
    H5Fclose (fid);
    H5Gclose (gid);
    return 0;
}
int main (int argc, char *argv[]) {
    int i;
    TW_Engine_handle_t engine;
    TW_Task_handle_t task[3];

    TW_Init (TW_Backend_native, TW_Event_backend_none, NULL, NULL);
    TW_Engine_create (NUM_THREADS, &engine);
    TW_Task_create (task1_fn, NULL, TW_TASK_DEP_NULL, 0, &task[0]);
    TW_Task_create (task2_fn, NULL, TW_TASK_DEP_NULL, 0, &task[1]);
    TW_Task_create (task3_fn, NULL, TW_TASK_DEP_NULL, 0, &task[2]);
    TW_Task_add_dep (task[1], task[0]);
    TW_Task_add_dep (task[2], task[1]);
    for (i = 0; i < 3; i++) TW_Task_commit (task[i], engine);
    for (i = 0; i < 3; i++) TW_Task_wait (task[i], TIMEOUT);
    for (i = 0; i < 3; i++) TW_Task_free (task[i]);
    TW_Engine_free (engine);
    TW_Finalize ();

    return 0;
}
```

Figure 4: Example usage of TaskWorks. Group can only be opened (task 2) after the file is opened (task 1). HDF5 objects can only be closed (task 3) after they are opened (task 1, 2).

COORDINATION

- MPI communication-aware task scheduling
- OpenMP compatible
  - Supports parallel sections in tasks

CONCLUSION

- HPC applications need a user-friendly, portable, flexible, and HPC-friendly task engine to empower asynchronous operations, such as I/O
- In this poster, we present TaskWorks
- Current status of TaskWorks
  - Argobots and Libevent (in Fig. 2) connectors are implemented
  - Native (in Fig. 2) connector using OS threads is completed
  - Support HDF5 async operation with TaskWorks is in the work
  - Optimization for multi-process environment is next

https://software.nersc.gov/exahdf5/taskworks

This research was supported by the Exascale Computing Project under Grant No. 17-SC-20-SC, a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative. This work is also supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract Nos. DE-AC02-05CH11231.