COMMON TOOLS INTERFACE (CTI)

Scalable Tools Workshop

7/29/2019

andrewg@cray.com
agontarek

Agenda

- Overview of Cray debugging toolbelt
  - Learn about what is available on Cray systems
- MPIR discussion
  - Why a new solution is needed
- Brief intro to CTI

- Github link to CTI repo: https://github.com/common-tools-interface/cti

CRAY

PROGRAMMING ENVIRONMENT

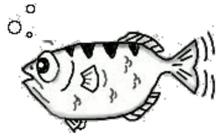
CRAY DEBUGGING TOOLS

From 30,000 feet

Cray debugging tools overview

- **Commercial third party debugging products**
  - Forge toolsuite from ARM/Allinea
  - Totalview debugger from RogueWave
  - Available on Cray systems

Cray debugging tools overview

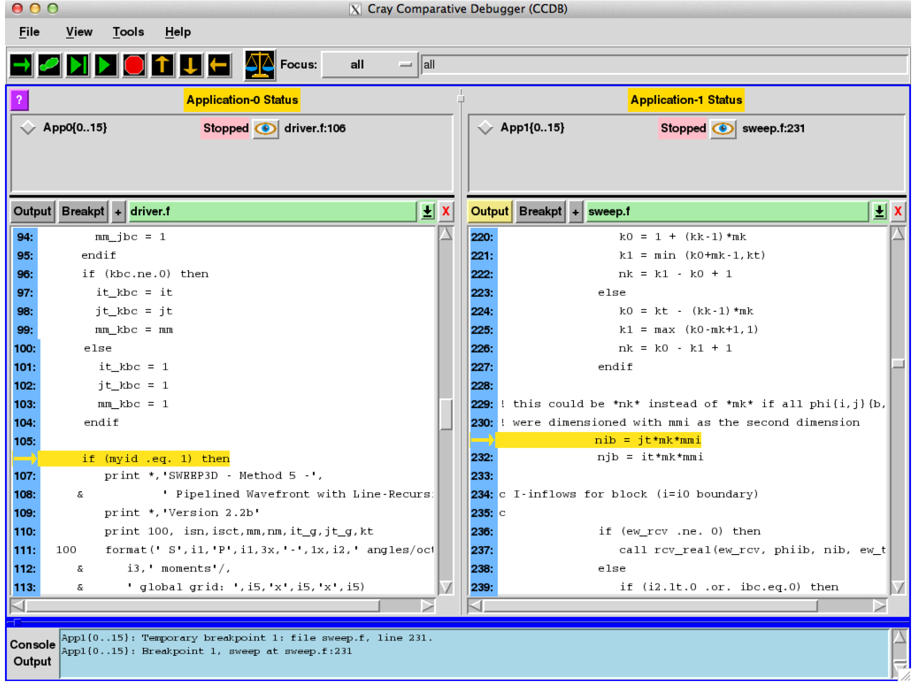
- **Gdb4hpc** 
  - Conventional CLI based interactive parallel debugger
  - Look and feel of gdb – syntax is inspired by gdb!
  - Debug your application at scale
- **Valgrind4hpc** 
  - Parallel valgrind based debugging tool (memcheck)
  - Aids in detection of memory leaks and errors in parallel applications
  - Aggregates like errors across PEs/threads

Cray debugging tools overview

- **STAT** (**S**tack **T**race **A**nalysis **T**ool)
  - Scalable generation of a single merged stack backtrace for the application
  - Open source tool from LLNL
  - GUI based tool (`stat-gui/stat-view`) along with cli tools (`stat-cl`)
  - Cray contributes code changes back upstream (ARM port)
  - Gain insight into application behavior at a function level
- **ATP** (**A**bnormal **T**ermination **P**rocessing)
  - Scalable core file generation and analysis when application crashes
  - Generates a merged stack backtrace akin to stat
  - Selection algorithm to dump unique core files

Cray debugging tools overview

- **CCDB** (**C**ray **C**omparative **D**e**B**ugger)
  - NOT a traditional debugger!
  - Compare two applications side-by-side
    - Focus on the data – not state and internal operations
  - GUI tool that interacts with gdb4hpc



Tool infrastructure

- **CTI** (**C**ommon **T**ools **I**nterface)
  - Single API to support tools across all Cray systems
  - WLM agnostic – write to API once, add WLM implementation in API
  - Application placement information, launch tool daemons on compute nodes
- **MRNET** (**M**ulticast **R**eduction **NET**work)
  - Scalable communication library for tools
- **Paradyn**
  - Dynamic instrumentation libraries

What's supported?

- **Shasta/XC**
  - All tools are supported on XC systems
  - These tools will continue to be supported on Shasta
  - User interaction will be the same on Shasta as it is today
- **Clusters**
  - Gdb4hpc, valgrind4hpc and CCDB only
- **Tools are compiler/network agnostic at their core**
  - Sockets for communication
  - Standard DWARF debug_info

WHY A COMMON TOOLS INTERFACE?

Towards a Common Tools Interface

- **Problem**: Cray needs to support several tools across diverse ecosystems
  - Multiple product lines
    - High end HPC systems like Shasta/XC/XK
    - Traditional cluster systems from the CS line
  - Multiple workload managers
    - ALPS based
      - PBS/Moab
    - SLURM based
    - Requests for many more!

What about MPIR?

- **State of the art**: MPIR
  - Good solution for its time!
- **Problematic in current era of HPC**
  - Required to attach onto the starter process
  - Multiple simultaneous tools?
  - Efficiency/scalability issues

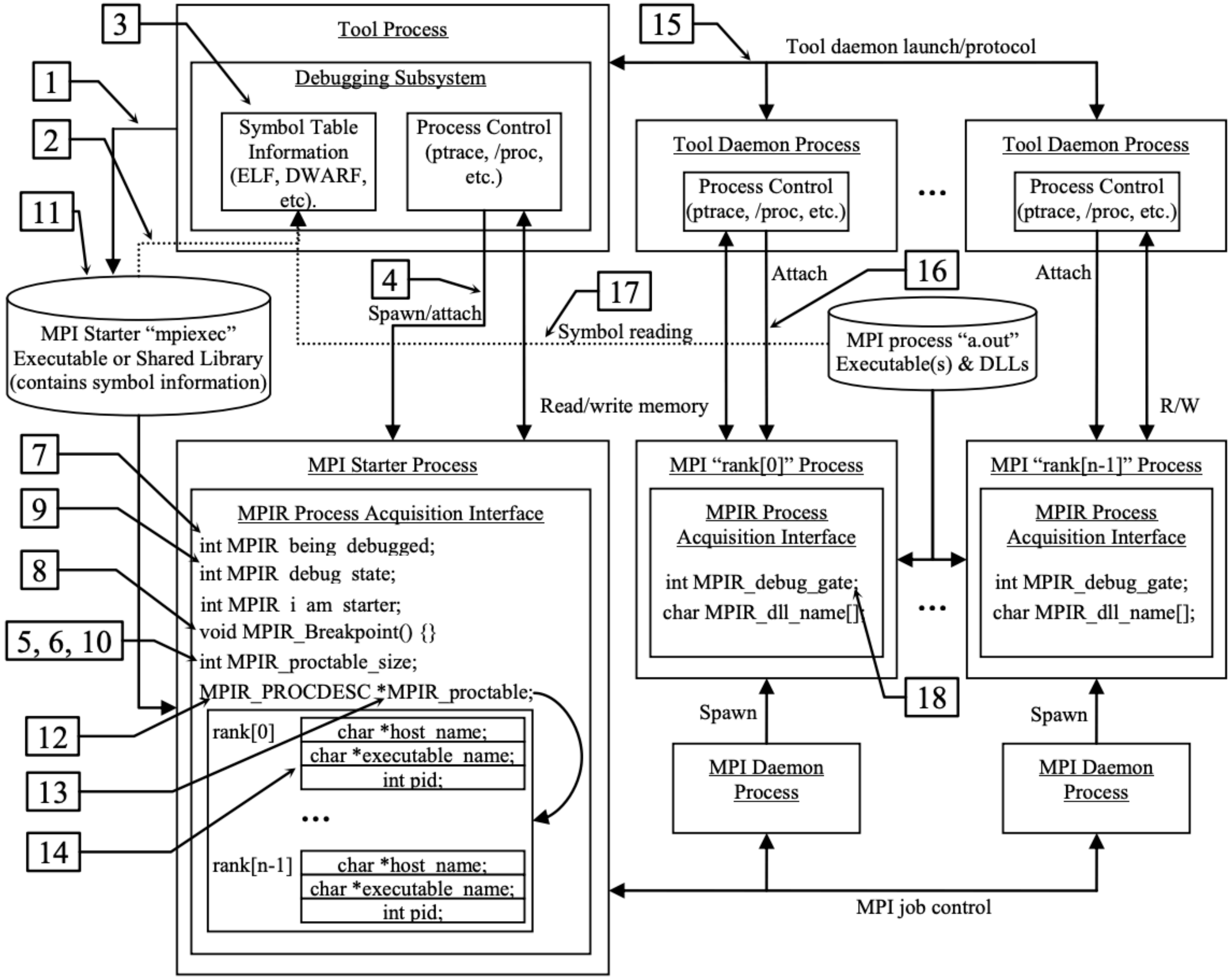


Figure 8.1: Example collaboration diagram for MPIR Process Acquisition Interface

What about MPIR?

- **Efficiency/scalability issues**
  - Requires placement data to be in-process

  - Read by Tool FE via ptrace (bad) or /proc/<pid>/mem (better)
  - Alternative: read directly from WLM DB instead
  - **Consider**
    - **Debug attach case**: Application is running in a batch script, attach onto application from another terminal
    - Need to connect to node where starter process is running in order to bootstrap!

What about MPIR?

- **Efficiency/scalability issues (cont.)**
  - Data requirements are unnecessary
    - Requires a `MPIR_PROCDESC` entry for each PE
    - ~22 MB of overhead at 1M PEs from struct alone
  - Efficient implementations will store only unique host_names/executable_names
    - What about thousands of simultaneous jobs?

```
typedef struct {
  char *host_name;
  char *executable_name;
  int   pid;
} MPIR_PROCDESC;
```

What about MPIR?

- **Efficiency/scalability issues (cont.)**
  - Cray tools care more about placement on the FE versus obtaining executable names/pids
    - Number of unique compute nodes
    - Hostname of each unique compute node
    - Number of PEs placed on that compute node
  - Obtain pid/exec path directly on compute node
  - **Approach**
    - Scale with number of compute nodes vs Pes
    - Better scaling with modern HPC systems

```
typedef struct {
  char *host_name;
  char *executable_name;
  int   pid;
} MPIR_PROCDESC;
```

```
typedef struct
{
    char *   hostname;
    int      numPes;
} cti_host_t;

typedef struct
{
    int              numHosts;
    cti_host_t *     hosts;
} cti_hostsList_t;
```

What about MPIR?

- **Tool daemon launching**
  - Extension exists to MPIR!
    - **OPTIONAL** 👎
  - Clunky interface
    - Write a variable via debugger to launch tool daemon
    - No generic interface for projecting files to computes
      - Use lustre/NFS??? Doesn’t scale for DSO lookup!
    - No way to control tool daemon specific environment
  - SSH directly to node vs srun vs alps toolhelper vs ???
    - Implement N different ways for launching tool daemons…

CRAY
PROGRAMMING
ENVIRONMENT

WHAT IS CTI?

What is CTI?

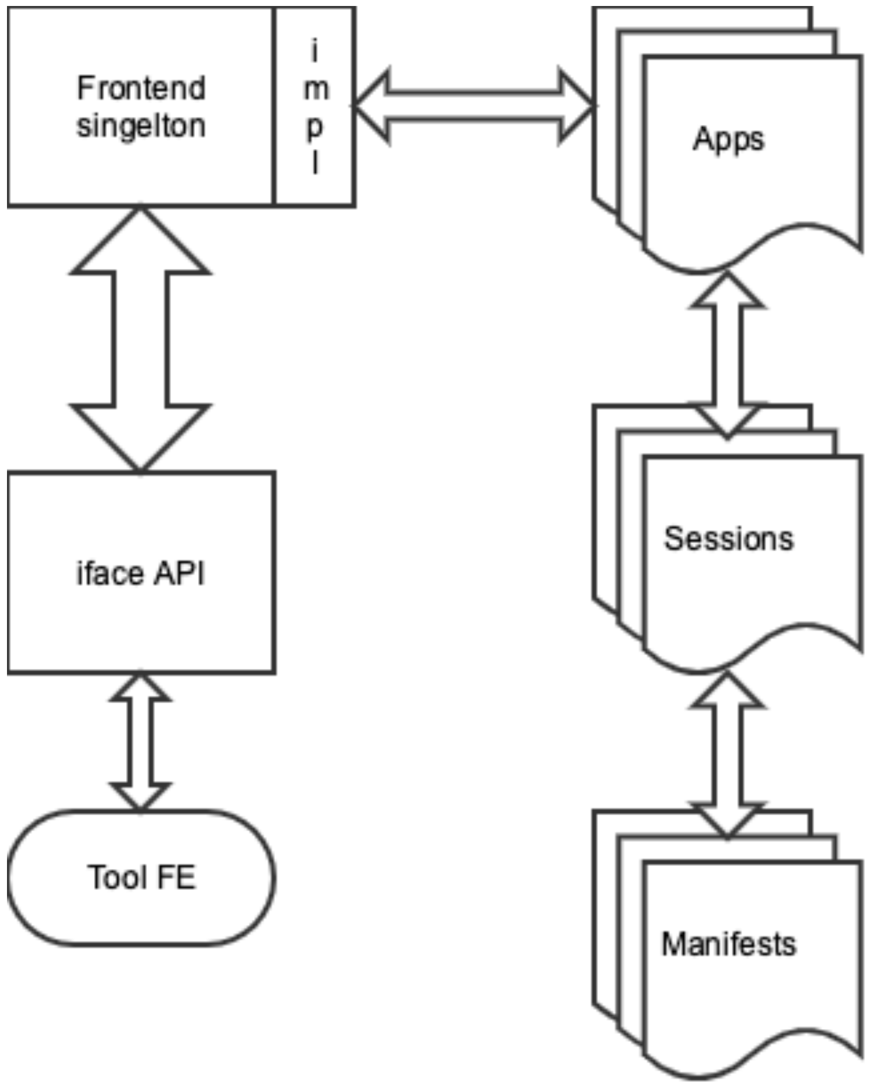
- **The Common Tools Interface aims to provide an API that provides…**
  - Way to query information about an application
  - Launch tool daemons alongside an application
    - Project binaries, libraries, and files alongside tool daemons
    - Setup tool daemon specific environment
    - Unique storage location for writing files (e.g. TMPDIR)
    - Allow for multiple tool daemons to co-exist
- **All while being implementation agnostic…**
  - Take care of common bootstrapping code that tool writers don’t necessarily care about

Towards a common interface

- CTI started as a side project in ~2011
  - **Goal**: Explore possibility of rapid prototyping of new tools
- Became CDST component in ~2013
  - Cray started supporting SLURM systems
  - Both ALPS and SLURM made things tricky for tools
  - Multiple code pathways – YUCK!
  - Porting our tools to use CTI saved development cycles and maintenance costs over time
- Stand alone cray-cti module in ~2015
- **TODAY**: Re-written in C++ on FE and open sourced (available on github)

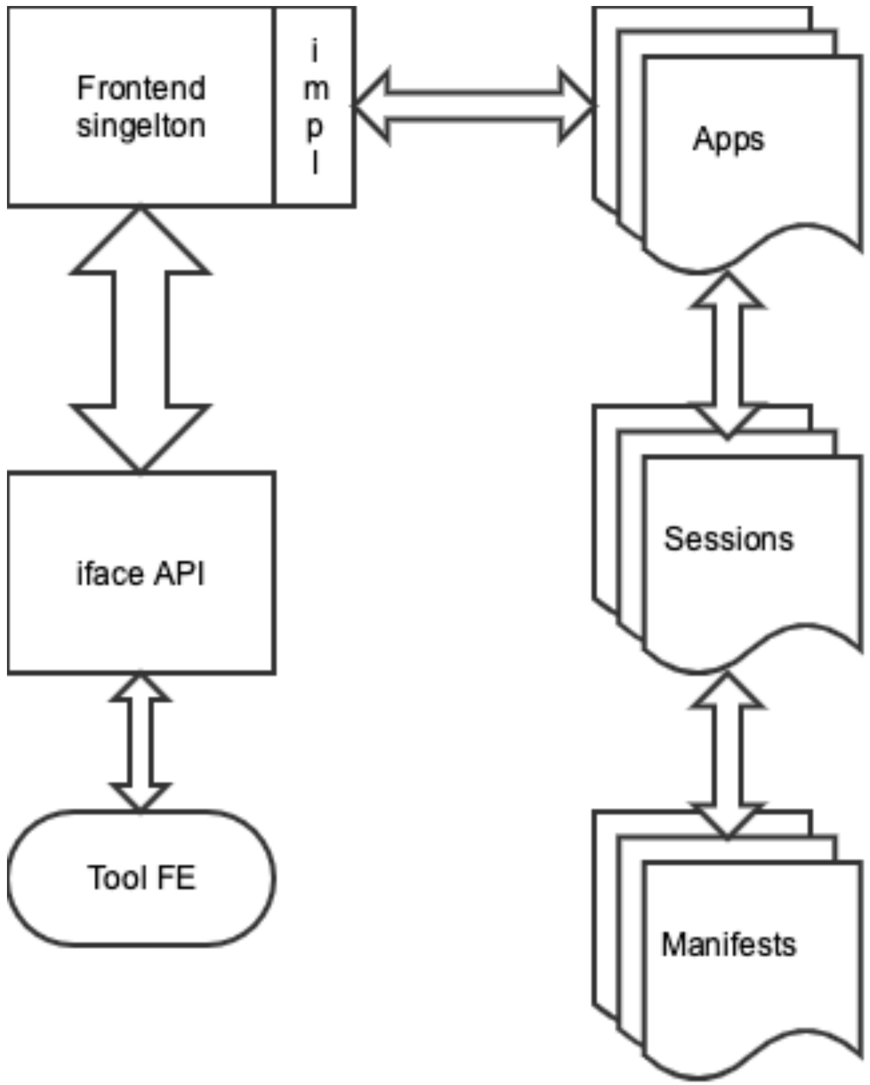
CTI FE architecture

- **Key object concepts**
  - **fe_iface**
    - C interface defining actual API
  - **Frontend**
    - Singleton object matching system type
    - Abstract base class requires real implementation to be provided
  - **App**
    - Represents application either registered or launched via API calls



CTI FE architecture

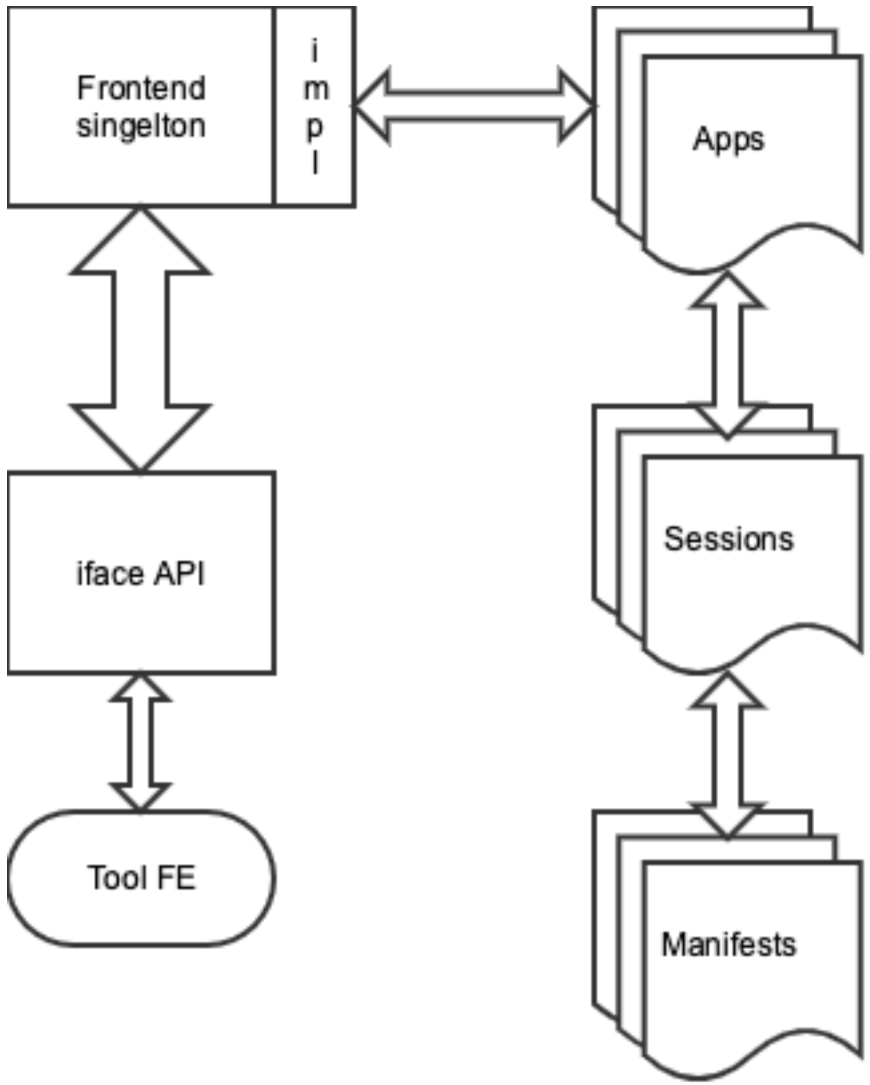
- **Key object concepts (cont.)**
  - **Session**
    - Represents a remote location on compute nodes
    - Stores state about what has previously been projected into session
    - Multiple tool daemons can share a session
  - **Manifest**
    - List of binaries, libraries, and files to be projected to compute nodes



CTI FE architecture

- **Key object concepts (cont.)**
  - **Ownership hierarchy**
    - Frontend owns associated apps
    - Apps own associated sessions
    - Sessions own associated manifests
    - When parent object destructed, all owned objects also destructed
  - **Iface mapping**

```
typedef int64_t cti_app_id_t;
typedef int64_t cti_session_id_t;
typedef int64_t cti_manifest_id_t;
```



CTI FE interface – Application launch/register

- A `cti_app_id_t` is required to do (almost) anything

```
cti_app_id_t cti_launchApp( const char * const  launcher_argv[],
                            int                 stdout_fd,
                            int                 stderr_fd,
                            const char *        inputFile,
                            const char *        chdirPath,
                            const char * const  env_list[]);
```

- Launch with barrier variant also provided
- WLM extensions provided via `cti_open_ops()` for registering an existing app

```
typedef struct {
    cti_srunProc_t* (*getJobInfo)(pid_t srunPid);
    cti_app_id_t    (*registerJobStep)(uint32_t job_id,uint32_t step_id);
    cti_srunProc_t* (*getSrunInfo)(cti_app_id_t appId);
} cti_slurm_ops_t;
```

CTI FE interface – query placement info

- Once a `cti_app_id_t` is obtained can query placement info

```
cti_hostsList_t * cti_getAppHostsPlacement(cti_app_id_t app_id);
```

```
typedef struct
{
    char *   hostname;
    int      numPes;
} cti_host_t;

typedef struct
{
    int             numHosts;
    cti_host_t *    hosts;
} cti_hostsList_t;
```

CTI FE interface – Session

- Need a session handle to launch tool daemons
  - No communication occurs in our implementation
  - API allows underlying implementations to do so if needed
  - `cti_createSession()` requires owning `cti_app_id_t`

```
cti_session_id_t cti_createSession(cti_app_id_t app_id);
```

CTI FE interface – Manifest

- Need a manifest to build a list of dependencies required by tool daemon
  - `cti_createManifest()` requires owning `cti_session_id_t`

```
cti_manifest_id_t cti_createManifest(cti_session_id_t sid);
```

- Once created, add dependencies to manifest

```
int cti_addManifestBinary(cti_manifest_id_t mid, const char *fstr);
```

```
int cti_addManifestLibrary(cti_manifest_id_t mid, const char *fstr);
```

```
int cti_addManifestLibDir(cti_manifest_id_t mid, const char *fstr);
```

```
int cti_addManifestFile(cti_manifest_id_t mid, const char *fstr);
```

- `cti_addManifestBinary()` adds dso dependencies by default

CTI FE interface – Tool Daemon Launch

- Use `cti_execToolDaemon()` to asynchronously launch a tool daemon alongside application

```
int cti_execToolDaemon( cti_manifest_id_t    mid,
                        const char *         fstr,
                        const char * const   args[],
                        const char * const   env[]);
```

- Requires a valid cti_manifest_id_t even if there are no additional dependencies
- `fstr` tool daemon binary implicitly added to manifest
- Synchronous launch mechanism for future work
  - Would allow error checking

CTI BE interface

- BE iface used to obtain information about that node
  - Link against tool daemon binary launched via `cti_execToolDaemon()`
  - Obtain application PIDs

```
cti_pidList_t * cti_be_findAppPids(void);
```

```
typedef struct
{
        pid_t                pid;     // This entries pid
        int                  rank;    // This entries rank
} cti_rankPidPair_t;

typedef struct
{
        int                      numPids;
        cti_rankPidPair_t *      pids;
} cti_pidList_t;
```

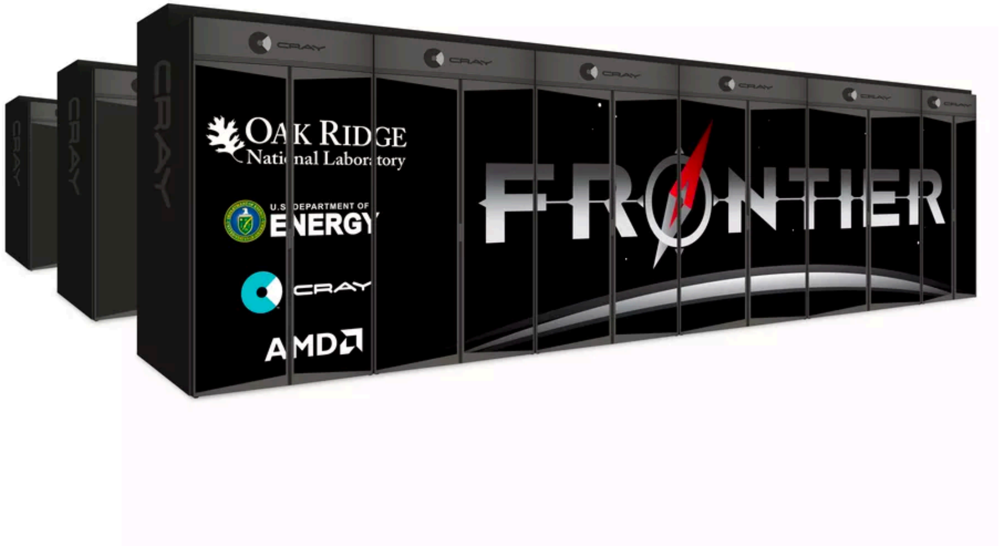
Future work

- **Community involvement as MPIR replacement**
  - If tools begin adopting this interface, better chance of new WLM implementors providing a reference CTI implementation for their ecosystem
  - Provides simple examples of tool requirements for bootstrapping
- **Add TBON communication layer**
  - Ability to gather error information
  - Scalable projection
  - Persistent TBON shared between tools?
- **Additional BE APIs to determine system capabilities**
  - Am I on a GPU node?

Wrap up

- Lots of material left uncovered!
  - Basic examples exist in tests/examples
- common-tools-interface org on github
- Reach out with any questions

- **Cray is hiring!**
  - Many positions open in PE



SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

QUESTIONS?

andrewg@cray.com
agontarek

cray.com
@cray_inc
linkedin.com/company/cray-inc-/