HiHAT C++API Proposal

Marcin Copik, Tal Ben-Nun

ETH Zürich

September 18, 2018

Why should we have C++ API?

- simplify interface (namespaces, RAII)
- human readable graph API
- high-level and user friendly (OpenCL versus SYCL)

Namespaces

Replace function prefixes

- `hhu`, `hhc`, `hhg`, `hhn`, `hhh`, `hhe`

With namespaces:

- `hh::`
- `hh::experimental::`

1. Would the interface benefit from nested namespaces such as `hh::graph::`?
2. Is there a need for C++ counterparts of low-level functions in common layer (`hhc`)

Wrapper classes

C API

```
hhTaskHndl CPU_invoke_fcn;
HIHAT_CHECK(hhnRegUserTask((void*)CPU_node,
    NULL, 0, sizeof(void*), CPU0_resrc, NULL,
     "CPU_invoke_fcn", HH_COMPLETED|
    HH_HAD_ERROR, HH_FIFO, &CPU_invoke_fcn));
```

C++ counterpart

```
hh::TaskHandle CPU_invoke_fcn(CPU_node,
 hh::Blob(nullptr, 0, sizeof(void*)),
 CPU0_resrc, nullptr, "CPU_invoke_fcn",
 hh::ActionState(HH_COMPLETED|HH_HAD_ERROR),
 hh::ExecOrder(HH_FIFO));
```

Wrapper classes

C++ wrapper

```
struct TaskHandle {
  hhTaskHndl handle;
  hhTaskHndl & get() { return handle; }
  const hhTaskHndl & get() const { return
   handle; }
}
```

- get() function provides underlying C handle for full functionality
- constructors and default parameters optimized for most common usage scenario

Graph API

Code example.

Graph API

- `AddEdge` and `AddNode` for flexible and dynamic graph creation
- operator overloading for more literate programming of static graphics

Enum-based flags

```
enum /* class */ ActionState {
  HH_NULL_STATE,
  HH_ON_HOLD,
  ...,
  HH_KILLED
}

void foo(int action_states) {}
// or?
void foo(ActionState action_states) {}
```

+ implicit scope of enum class
+ static type checking
− less robust interfaces

Error handling

Error codes returned in C API

```
HIHAT_CHECK(
 hheGraphTemplateCreate(
  &gD_template, 0
     )
);
```

Error handling

What about C++?

- functions return only error codes
  what if ctor fails?
- exceptions
  every function has to throw for consistency
- return both value and error codes
  requires expected-like implementation

Minor suggestions

Synchronous clean-up in C

```
HIHAT_CHECK(
 hheGraphInstanceDestroy(g_instance)
);
HIHAT_CHECK(hhnSyncAll(ahs));
free(CPU_addr);
```

Implicit synchronization and cleanup in destructor

```
ahs.postComplete(
  [CPU_addr]() {
    free(CPU_addr);
  }
);
```

Questions

1. What is the preferred way of handling errors?
2. Are enums preferred over integers in the interface?
3. Do we want a header-only implementation on top of HiHAT C library?
4. How can we make programming graphs easier?
5. Is it sufficient to target only common usage scenarios?