Making H5P Multi-Thread Safe:
A Sketch Design

John Mainzer
Lifeboat, LLC

10/16/22

Introduction

H5P provides support for the property lists used throughout HDF5. It must depend on H5I since property lists are accessed via IDs – instance of hid_t to be precise. Similarly, it must depend on H5E for error reporting. While in principle, there seems no reason why H5P proper should have any other dependencies, it should be no surprise at this point that this is not the case.

More generally, since H5P includes support for getting and setting the various properties associated with the various standard property list – FCPL (File Creation Property List), FAPL (File Access Property List), etc. – there must be at least minimal interaction with the packages configured with these properties. In principle, this should be limited to structure definitions, constants, and a bit of sanity checking – all items with little or no multi-thread safety significance. While this is certainly true in some cases, at present I don't know if it is true in general.

This is an extremely preliminary version of the sketch design for modifying H5P to be multi-thread safe. It is intended to map out the major dependencies and other issues, and to offer some initial thoughts on possible solutions. Expect all this to change as my detailed review of the code continues.

A Quick Overview of H5P

H5P exists to provide property list services to the HDF5 library proper, and to application programs. The basic services may be summarized as follows:

- Create, delete, copy, and modify classes of property lists. Here a "class" of property lists is best thought of as an archetype for newly created property lists of the specified class. Among other things, it specifies the properties that may appear in a new property list of the target class. New property list classes are created by duplicating an existing class, and then adding new properties.

- Create, delete, copy, compare for equality, encode, and decode property lists. Get and set the values of properties in a property list. Insert new properties into a given property list. Iterate through the entries in a property list.

Multi-Thread Issues in H5P

Like H5E and H5I, there appear to be no fundamental reason why H5P can't be made multi-thread safe[1]. Unlike H5E and H5P, one of the dependencies of H5P presents a major performance issue whose resolution has been on the to-do list for some time. It should be resolved in passing if at all practical.

In addition, there are other issues and known unknowns to be dealt with. These are outlined below.

Use of other HDF5 packages in H5P

The current implementation of H5P proper makes calls to the following packages in the HDF5 library:

- H5MM
- H5FL
- H5E
- H5I
- H5SL

In addition, the H5P code supporting the various predefined property list classes make calls into a variety of other packages in support of property value gets and sets. For example, the code supporting the metadata cache configuration property in the FAPL makes calls to H5AC (top level metadata cache) to run sanity checks on metadata cache configurations as part of the set operation. At least at first blush there seems to be no reason why there would be any multi-thread issues here, but experience indicates that I should expect at least one or two. I will find out as I proceed with my detailed review of the code.

As in H5E and H5I, H5MM and H5FL are easily avoided by using the C dynamic memory allocation functions directly, and by either not maintaining free lists, or maintaining them internally.

The dependencies on H5E and H5I are expected. With the proposed multi-thread safety modifications to these packages, they shouldn't be an issue.

---

1 But recall that this document is extremely preliminary. I haven't finished a detailed review of the core code, much less the property specific code.

H5SL implements skip lists, and was used extensively throughout the HDF5 library. Unfortunately, profiling exercises indicated that the shift to skip lists was the primary cause of the slowdown in HDF5 performance since HDF5 1.6. This was particularly an issue in H5P, since skip lists are a fairly heavy weight data structure, property lists seldom if ever exceed 25 entries, and (at the time) were looked up repeatedly. The introduction of H5CX largely mitigated this issue as it (among other things) caches the values of commonly accessed property list entries. However, property list operations remain expensive, and other optimization efforts have pushed H5P back up near the top of the optimization to-do list.

To date, optimizing skip lists out of H5P has not been attempted due to the complexity of that package. However, simplification will be necessary as part of multi-thread support – giving us a double incentive to redesign H5P to remove its dependency on H5SL.

Multi-thread thread issues in H5P proper

Leaving aside H5SL, and any issues presented by the code supporting individual properties, so far I haven't run across much. Each property list appears to be pretty much independent – but note that each property list maintains a pointer to its parent property list class. Some operations require access to the property list class and any property list classes it may descend from. This introduces a number of complexities whose full implications I am still in the process of investigating.

In addition to this, there are the usual potential public API race conditions. As before, this is an unsolvable problem from the perspective of H5P – all H5P can do is to execute operations in some order, and to keep its data structures in an internally consistent state. It will be the responsibility of the client to either avoid race conditions of the above type, or to handle them gracefully.

Solutions

While this is very preliminary, and should be expected to change as I continue my detailed review of the H5P code, my current thought is to rework both property list classes and property lists proper into flat structures. Further, since both the creation of new property list classes, and the addition of properties to either property list classes or property lists is rare, my thought is to optimize for the common cases, while accepting some additional overhead for the rare ones. Further, to simplify mutual exclusion, incorporate all property list class data into property lists so as to avoid the look-ups.

Let consider the property list class first. Implement it as a structure containing the fields from H5P_gen_class_t and H5P_gen_list_t (minus some fields that are no longer relevant), followed by array structures similar to H5P_gen_prop_t – one for each property. Conceptually at least,

whenever a new property is added, we realloc the structure to create room for the new entry, add it, and then sort the array by property name to allow binary search for properties. Each property in this array is set to its default value.

To create a new property list, we simply duplicate the parent property list class, set whatever flags are necessary to indicate that it is a property list and not a property list class, and insert it in the index. In the very rare event that a property is inserted into an individual property list, we handle it as per property list classes.

This makes property lists completely self-contained as least as far as H5P is concerned. Calls supporting the individual properties may complicate this – but I will not know until I complete my survey.

Ideally, we would make changes to the values of properties with atomics – thus avoiding the need for locking on property list. Where this is not possible, property lists would have to have read / write locks. Since property list are frequently initialized, and then never modified again, it may be useful to add an API to mark a property list as read only – thus avoiding the need for locking in the future.

Observe that this gets rid of skip lists while retaining the option of creating arbitrarily many new properties. I expect that this will increase the memory foot print of property lists. If this proves to be significant, a number of space optimizations are possible. Lookup times should be improved due to increased locality and the removal of skip list overhead. It may be useful to store the property names in adjacent locations to improve locality further.

In closing, I must emphasize that the above is nothing but initial musing on on how to modify H5P to support multi-thread, and to optimize out H5SL. We will see how much this sketch changes in the next version of this document.

Appendix 1 – H5P public API calls

<in progress>

Appendix 2 – H5P internal API calls

<in progress>