Charm++ - Bug #1545
Serialize std::vector with Custom Allocator

05/01/2017 05:13 PM - Nils Deppe

Status: New
Start date: 05/01/2017
Priority: Normal
Due date: 0%
Assignee: Phil Miller
% Done: 0%
Category: Estimated time: 0.00 hour
Target version: 6.8.1
Spent time: 0.00 hour

Description
It should be possible to serialize std::vector that has a custom allocator. What works for me right now is changing a few lines in pup_stl.hpp:

```cpp
#include <vector>

template <class T>
inline void operator|p(errno &p, typename std::vector<T> &v);
```

(~line 51) to

```cpp
template <class T, class A>
inline void operator|p(errno &p, typename std::vector<T, A> &v);
```

and line ~177 to

```cpp
template <class T, class A>
inline void operator|p(errno &p, typename std::vector<T, A> &v)
{ PUP_stl_container<std::vector<T, A>,T>(p,v); }
```

Do you think this could make it into v6.8.0?

History

#1 - 05/02/2017 10:48 AM - Phil Miller
- Assignee set to Phil Miller

I think we could do this, but I'm concerned about handling of instances in which the allocator isn't simply stateless and default-constructed. Would we also need to add p | v.get_allocator() on the packing side, and unpack the allocator on the receive side and then construct the new instance of the vector using it?

#2 - 05/02/2017 10:49 AM - Phil Miller
- Target version changed from 6.8.0 to 6.8.1
- Tags set to c++, pup, migration

For the moment, I'm going to target this at 6.8.1, since I don't think we're willing to hold the release to get this change done. If we have it done before the release, it should be included.

#3 - 05/02/2017 10:44 PM - Nils Deppe
That is a good question... This probably depends on the allocator itself. For some stateful allocators it'll make sense to serialize them, but probably for others it wouldn't, e.g. if they are working out of a node level memory pool or something. The correct solution is probably to pup the allocator if it has a pup function (I'll paste an implementation of a type trait that can be used for this below). Since many of these niceties require some C++11 STL support that could easily be implemented for the Charm++ 6.8.1 release should we do that? What is the time frame for 6.9?

```cpp
template <typename T, typename = void>
struct has_pup_member : std::false_type {};

template <typename T>
struct has_pup_member<T, cpp17::void_t<decltype(std::declval<T>().pup(
    std::declval<PUP::er &>()))>> : std::true_type {};
```

If you want implementations of void_t, and enable_if let me know and I can provide them. The only issue with the above is that `decltype` is a C++11 language feature, so that cannot be written up...

---

#4 - 05/04/2017 11:04 AM - Phil Miller

We're not going to require any more C++11 support in a patch/bug-fix release like 6.8.1 than we do in the feature release 6.8.0. 6.9 would ideally come soon after, since we'll be able to make a lot of improvements by adopting C++11 whole-heartedly, but our gears turn slowly.

The full support can appear in our mainline development branch as soon as we branch/tag for 6.8, so users won't have to wait for 6.9 to be released to get the fully-working code.

We could go ahead with adding naive support for stateless allocators now, with a note in the documentation that stateful allocators are not yet handled correctly. I don't think there's any automatic way to detect that, though, so it could lead to some 'interesting' failures if someone doesn't see that note. I can think of a terrible hack involving statically testing sizeof(A) and failing if it's big enough to hold a pointer, so that people can't do this by accident.

I think my preference is to just wait to after the 6.8 series, and implement the right code from the start.