Large scale libc++ deployment

Evgenii Stepanov, Google
Ivan Krasin, Google
Containers of incomplete types

class A { std::deque<A> d; }

class B { std::set<B>::iterator p; }

Super-popular patterns. Some algorithms are much harder to write w/o this.

Supported in: libstdc++, stlport, boost.

hash_set & hash_map: done

deque: ABI-breaking change http://reviews.llvm.org/D10677
ABI stability

Libc++ generally avoids ABI-breaking changes.

- Helps shipping libc++ in production systems.
- Blocks several desired changes.
- Some users don’t care for ABI stability!

Solution: ABI versioning.

- `LIBCXX_ABI_VERSION=XX`
- `LIBCXX_ABI_UNSTABLE=ON`
always_inline

Control over which symbols are part of the ABI.

Almost 5900 uses in libc++.

Cons:

- Does not always work.
  - Current implementation does not inline unreachable call sites.
  - Incompatible function attributes prevent inlining.

- Breaks -O0.
  - Aggressive inlining w/o alloca merging => huge stack frames.
  - 15% testsuite speedup with always_inline removed!
internal_linkage

always_inline = internal linkage \texttt{(good)} + inlining \texttt{(bad)} \texttt{[if called directly]}

RFC: \texttt{\_attribute\_((internal\_linkage))}

Think C-style “static” on class methods. And even classes and namespaces.

\url{http://reviews.llvm.org/D13925}
Container assignment requirements

Allocator-aware container X: \( X\langle T \rangle \ a; \ a = t; \)

Standard & libc++: \( T \) is CopyInsertable into \( X \) and CopyAssignable.

libstdc++: \( T \) is CopyInsertable into \( X \)

struct A { A& operator=(const A&) = delete; };

struct A { const int x; };

libstdc++: PASS, libc++: FAIL
**Constexpr pair & initializer_list constructors**

```cpp
constexpr pair( const T1& x, const T2& y );
```

```cpp
constexpr initializer_list();
```

Standard & libc++: constexpr since C++14.

Libstdc++: constexpr since C++11.
Replacement for __gnu_cxx::random_sample

random_sample appears in libstdc++ & stlport, but not in libc++.

std::experimental::sample (library fundamentals TS), implemented in libc++. 
std::tuple extension

Constructor accepts less values than the number of tuple elements.

SFINAE issues: confusion between:

- Copy constructor
- Single element constructor

May end in infinite recursion via construction of tuple<tuple…> instead of a copy.

http://reviews.llvm.org/D12502 by EricWF
Misc differences

- `std::pow<float, float>` works in libstdc++ but not in libc++. Invalid code.
- `std::vector<bool>::const_reference` is **not** `bool` - libc++ not standard compliant.
- Different iteration order of hash-based containers.
Conclusions

- High quality implementation.
- Adopting a few extensions would make migration easier:
  - Container assignment should not require the element to be copy assignable. Const class members are very common.
  - Complete element type should not be a requirement for container instantiation.