gcc static checker update

Scott Snyder

Brookhaven National Laboratory, Upton, NY, USA

Nov 11, 2015

gcc static checker update

- The package `CheckerGccPlugins` adds additional static checking during compilation with gcc.
- New wiki page `CheckerGccPlugins` available describing the available checks. All warning messages will include a reference to this page.
- Only one currently enabled is gaudi inheritance checking:
  - Atlas algorithms/tools/services should derive from the classes in AthenaBaseComps rather than directly from the Gaudi classes.

divcheck: Redundant division checking

- Look for repeated divisions and division by constant that could be replaced by multiplication by the reciprocal.

```
float x = y/3.;
```

```
float x = y * (1/3.);
```

```
float xn = x / r;
float xn = y / r;
```

```
float inv_r = 1./ r;
float xn = x * inv_r;
float xn = y * inv_r;
```

```
for (int i=0; i < 10; i++)
  x[i] /= tot;
```

```
float inv_tot = 1./ tot;
for (int i=0; i < 10; i++)
  x[i] *= inv_tot;
```

- (Such transformations can change the results slightly, so the compiler won't do them with -ffast-math.)
- Decided not to enable this by default.

naming: Naming convention checker

Warn about violations of a few of the ATLAS naming conventions.

- Private/protected data members should start with `m_`, or `s_` if the member is static.
- `const static` members may also be written in all-caps.
- Do not use names starting with `m_` for anything other than data members.
- Do not start any identifiers with an underscore.
- Numerous exceptions. Try not to warn about external code. Allow ROOT-style conventions. Don't enforce member naming for persistent classes.
- Most of these warnings already cleaned up for AtlasCore.
- Could turn this on by default at any time, but worry that the number of warnings will be overwhelming.
  - For starters, maybe enable only in one nightly of devval (`rel_6` or `rel_0`)?

usingns: Check location of using directives/declarations

Warn about using directives/declarations in the global namespace:

- In a header file.
- In main source file before an `#include` directive.

Works fine, but requires a couple small patches to the compiler itself.

- Will try up upstream eventually, but not there yet.

thread: Thread-safety checking

Warn about potential thread-safety problems. (Work in progress.)

- For the forseeable future, we expect that only some fraction of our code will be considered thread-safe.
- Only want to generate warnings for code that is actually meant to be thread-safe.
- Explicitly mark such code, at different levels of granularity.
- Function or class: attribute.

```
void f1 [[ATLAS_THREAD_SAFE]] () { ... }
class [[ATLAS_THREAD_SAFE]] C { ... };
```

  - `ATLAS_THREAD_SAFE` is a macro:
  - Expands to real attribute name (currently `gnu::thread_safe`) when plugin is used.
  - Expand to nothing otherwise. (Syntax remains ok.)
- Single file: `#pragma ATLAS thread_safe`
- Entire package: Create a file `ATLAS_THREAD_SAFE` in the package's directory.

thread: Static checking

Warn about use of non-const static data:

```
static int x;
int f [[ATLAS_THREAD_SAFE]] () { return x; }
```

```
x.cc: In function int f():
x.cc:2:40: warning: Use of static expression x within
   thread-safe function int f() may not be thread-safe.
 int f [[ATLAS_THREAD_SAFE]] () { return x; }
                                        ^
x.cc:1:12: note: Declared here:
 static int x;
            ^
x.cc:2:40: note: See <https://twiki.cern.ch/twiki/bin/view/Atl
```

thread: Static checking

Also warn about taking pointer or reference to non-const static.

```
static int x;
int* f [[ATLAS_THREAD_SAFE]] () { return &x; }
```

```
x.cc: In function int* f():
x.cc:2:42: warning: Non-const pointer or reference bound
   to static expression x within thread-safe function
   int* f(); may not be thread-safe.
 int* f [[gnu::thread_safe]] () { return &x; }
                                          ^
x.cc:1:12: note: Declared here:
 static int x;
```

Suppress warning with an attribute. (Maybe should use a different name?)

```
static int x [[ATLAS_THREAD_SAFE]];
```

thread: const_cast checking

Warn about casting away const within thread-safe code.

```
int* f1 [[ATLAS_THREAD_SAFE]] (const int* y)
{
  return const_cast<int*>(y);
}
```

```
x.cc: In function int* f1(const int*):
x.cc:3:28: warning: Const discarded from expression y within
   thread-safe function int* f1(const int*); may not be threa
   return const_cast<int*>(y);
                           ^
```

Again can use an attribute on the LHS to suppress:

```
  int* yy [[ATLAS_THREAD_SAFE]] = (int*)y;
  return yy;
```

thread: mutable checking

Warn about assigning (or taking a non-const pointer/reference to) a `mutable` field of a `const` object within thread-safe code.

```
struct S { mutable int x; };
void f1 [[ATLAS_THREAD_SAFE]] (const S& s) { s.x = 10; }
```

```
z.cc: In function void f1(const S&):
z.cc:2:54: warning: Setting mutable field S::x within
  thread-safe function void f1(const S&); may not be thread-sa
 void f1 [[ATLAS_THREAD_SAFE]] (const S& s) { s.x = 10; }
```

Can suppress with:

```
struct S { mutable int x [[ATLAS_THREAD_SAFE]]; };
```

Also warn about declarations of `static`/`mutable` members within thread-safe classes.

thread: call checking

Warn about calls to thread-unsafe code from thread-safe code.

```
struct S { void s1(); };
void f1 [[ATLAS_THREAD_SAFE]] (S& s) { s.s1(); }
```

```
z.cc: In function void f1(S&):
z.cc:2:46: warning: Non-thread-safe function void S::s1()
   called from thread-safe function void f1(S&); may not be th
 void f1 [[ATLAS_THREAD_SAFE]] (S& s) { s.s1(); }
```

- In progress. Still need:
  - Whitelist external/stdlib functions.
  - Warn about thread-unsafe override of thread-safe virtual function.
  - Handle calls through function pointers?

Looking forward

- Finish call checks of thread-safety checker.
- What other things should we be checking for?
- Good bit of tuning likely needed before this is generally useful.
  - Start trying to use it to check some Core packages.
- Should we start enabling naming convention checker?