GammaLib - Action #811

Feature # 810 (Closed): Optimize computations

Add sin() and cos() cache to GSkyDir

03/28/2013 03:26 AM - Knödlseder Jürgen

Status:	Closed	**Start date:**	03/28/2013
Priority:	Normal	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.0.0		

Description

The dist() method is one of the most used function of the GSkyDir class. When called repeatedly, the dist() method recomputes for the moment the sin() and cos() of the coordinates. This can be prevented by adding a sin() and cos() cache that stores the values once computed.

The drawback is that the cache will considerably increase the size of the GSkyDir class, as there are 4 coordinates (RA, DEC, GLON, GLAT) and (at least) two functions (sin, cos), requiring 8 double precision values. In addition, we may want to store the tan() results that are used in posang(). And we need flags that signal that a cache has been set (this could eventually be done by Bit checking, yet Bit checking is slightly slower than bool checking ... at least on my Mac OS X).

History

#1 - 03/28/2013 08:01 AM - Knödlseder Jürgen

To avoid a large amount of memory for the pre computation cache, we can try a logic where the pre computation cache is allocated dynamically. We may simply define a structure

```
struct cache {
   double m_sin_lat;
   double m_cos_lat;
};
```

that is dynamically allocated when necessary. We then just have to add two pointers

```
mutable cache* m_cel_cache;
mutable cache* m_gal_cache;
```

for the cache for celestial and galactic coordinates. By default, the pointers will be set to NULL, which is used as a flag for having the cache set or not. We then add methods

```
void set_cel_cache(void) const
{
   if (m_cel_cache == NULL) {
      if (!m_has_radec && m_has_lb) { // Make sure that RA, DEC are valid
         gal2equ();
      }
      m_cel_cache = new cache;
      m_cel_cache->m_sin_lat = std::cos(m_dec);
      m_cel_cache->m_cos_lat = std::cos(m_dec);
   }
   return;
}
```

to set the cache values. In addition, we can add methods

```
double sin_dec(void) const
{
   set_cel_cache();
   return m_cel_cache->m_sin_lat;
}
```

etc. for returning the cache values. When values are set, we simply deallocate the cache to make sure it will be recomputed the next time when it is needed

```
if (m_cel_cache != NULL) delete m_cel_cache;
```

This will bring a speed penalty due to memory allocation and deallocation, yet I guess that this is easily recovered after using the cache a few times. To be tested.

#2 - 03/28/2013 04:15 PM - Knödlseder Jürgen

- File GSkyDir.hpp added

- File GSkyDir.cpp added

I did some timing tests, the relevant code is attachment:"GSkyDir.hpp" and attachment:"GSkyDir.cpp". For class variants have been implemented:

- GSkyDir implements the actual dist computation
- GSkyDir2 implements a precomputation cache and the methods sin_dec and cos_dec
- GSkyDir3 implements a precomputation cache and the cache computations are directly performed in the relevant class
- GSkyDir4 implements a dynamic allocation of a precomputation cache, and the cache computations are directly performed in the relevant class

The main difference between GSkyDir2 and GSkyDir3 is that no overhead in calling sin_dec and cos_dec is needed in the latter, while the former needs function calls, although the sin_dec and cos_dec methods have been declared inline.

Computations have been done using the nested loop

```
// Loop
double sum = 0.0;
for (int outer = 0; outer < nouter; ++outer) {
    GSkyDir crab( 83.6331,  22.0145);
    GSkyDir vela(128.8361, -45.1764);
    for (int inner = 0; inner < ninner; ++inner) {
        double dist = crab.dist(vela);
        sum += dist;
    }
}
```

where nouter and ninner are the number of iterations in the outer and inner loop, respectively. This allows to study the speed gain when reusing the cache values.

Here a CPU benchmark on Mac OS X for 10000000 / ninner iterations, where ninner was selected to be 1, 2, 3, 10, 100 and 1000:

Class	**1**	**2**	**3**	**10**	**100**	**1000**
GSkyDir	1.41949	1.36537	1.32523	1.21473	1.22217	1.19108
GSkyDir2	1.6772	1.21985	1.06055	0.846352	0.771483	0.761635
GSkyDir3	1.47498	1.07373	0.917423	0.632076	0.552024	0.542549
GSkyDir4	3.15931	1.9054	1.45324	0.838957	0.577843	0.604065

GSkyDir is obviously fastest when the dist computation is never reused (ninner=1). GSkyDir3 comes closets to the results, since only little overhead is involved with testing the existence of precomputed values. GSkyDir4 is worst, as the dynamic allocation of the cache memory takes some time. The dynamic allocation penalty is only recovered after a few reuses of the values.

If the precomputed values are reused (ninner>1), GSkyDir3 works best, taking only 45% of the time needed for the original method for (ninner>100). The drawback of this method is that the memory held by the class is essentially doubled with respect to the original class. If precomputed values are

dynamically allocated, about 50% of the time is needed for (ninner>100).

For comparison, here the same table on **galileo** (32 Bit Linux system):

Class	**1**	**2**	**3**	**10**	**100**	**1000**
GSkyDir	7.01	6.87	6.83	6.57	6.58	6.61
GSkyDir2	7.34	5.24	4.57	3.63	3.27	3.24
GSkyDir3	7.1	4.99	4.29	3.3	2.91	2.88
GSkyDir4	8.85	5.9	4.88	3.48	2.94	2.89

The general behavior is similar, yet the dynamic memory allocation penalty is considerably smaller. With a single reuse of the cache, the GSkyDir4::dist method becomes faster than the GSkyDir::dist method. For (ninner>100), the speed penalty for dynamic memory allocation completely vanished.

And finally the same table for **kepler** (64 Bit Linux system):

Class	**1**	**2**	**3**	**10**	**100**	**1000**
GSkyDir	4.35	4.31	4.16	4.01	3.96	3.95
GSkyDir2	4.9	3.36	2.93	2.14	1.85	1.82
GSkyDir3	4.52	3.02	2.61	1.85	1.57	1.52
GSkyDir4	6.05	3.86	3.19	2.04	1.61	1.57

Here the dynamic memory allocation penalty is larger, but the code is still faster than the original code for (ninner>1). The computation time for large ninner is 40% of the original time.

Conclusion: The implementation of a cache should be considered. Using additional members roughly doubles the size of GSkyDir, but reduces the computing time to 40-45% of the actual duration on all tested systems. Using a dynamically allocated cache will slow down significantly the computations if the values are never reused, hence this approach should not be pursued.

#3 - 03/28/2013 04:16 PM - Knödlseder Jürgen

- Status changed from New to In Progress

- % Done changed from 0 to 20

#4 - 03/29/2013 01:03 AM - Knödlseder Jürgen

- Target version set to 00-08-00

#5 - 03/29/2013 03:17 AM - Knödlseder Jürgen

- Assigned To set to Knödlseder Jürgen

- % Done changed from 20 to 90

The pre computation cache has been implemented. Still need to check the impact on the result.

#6 - 03/29/2013 03:48 AM - Knödlseder Jürgen

- Status changed from In Progress to Feedback

- % Done changed from 90 to 100

- Remaining (hours) set to 0.0

Here the actual speed change (including also modifications in the matrix stack, vector and model access):

Method	**Old (/usr/local/gamma)**	**New**
Binned	32.5 sec	18.4 sec
Unbinned	0.31 sec	0.22 sec

#7 - 12/03/2013 09:59 AM - Knödlseder Jürgen

- Status changed from Feedback to Closed

- Estimated time set to 0.00

#8 - 12/10/2013 12:16 AM - Knödlseder Jürgen

- Target version deleted (00-08-00)

#9 - 07/11/2014 04:12 PM - Knödlseder Jürgen

- Target version set to 1.0.0

Files

GSkyDir.hpp	2.32 KB	03/28/2013	Knödlseder Jürgen
GSkyDir.cpp	7.26 KB	03/28/2013	Knödlseder Jürgen