ctools - Feature #2421

make cslightcrv, csspec ... parallelizable

03/27/2018 10:40 AM - Tibaldo Luigi

Status:	Closed	**Start date:**	03/27/2018
Priority:	Normal	**Due date:**	
Assigned To:	Tibaldo Luigi	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.6.0		

Description

All scripts that make multiple analysis iterations (over energy bins, time bins ...)

History

#1 - 03/27/2018 12:57 PM - Knödlseder Jürgen

The parallelization should be done using the multiprocessing module that is available from Python 2.6 on. Ideally the code should be written in a way so that it is still working even if the multiprocessing module is not available.

#2 - 05/11/2018 05:00 PM - Tibaldo Luigi

Add cssens to the list

#3 - 06/15/2018 06:25 PM - Tibaldo Luigi

- Assigned To set to Tibaldo Luigi

#4 - 06/20/2018 10:29 AM - Tibaldo Luigi

- Status changed from New to In Progress

The main complication turns out to be that multiprocessing only works on functions with arguments that are accepted by pickle. This excludes all the Python objects created via swig. There are some indications on how to make a swig object pickalable here
https://stackoverflow.com/questions/9310053/how-to-make-my-swig-extension-module-work-with-pickle#9325185
It is not clear if this approach can ever work with respect to the parent class logger.

#5 - 06/20/2018 12:01 PM - Tibaldo Luigi

First attempt: make cslightcrv class itself pickalable
(accepted solution at https://stackoverflow.com/questions/9310053/how-to-make-my-swig-extension-module-work-with-pickle#9325185)
so that I can pass it with all its attributes (observations, times, etc) to an external function that carries out the operation to be parallelised. Fails for unclear reasons: the process hangs for a while and then ends with Killed: 9.

#6 - 06/21/2018 09:40 AM - Knödlseder Jürgen

- % Done changed from 0 to 10

Is there no way to circumvent the pickle issue by parallelizing functions that do not have GammaLib objects as argument?

#7 - 06/21/2018 11:41 AM - Tibaldo Luigi

Jürgen pointed out this thread
https://stackoverflow.com/questions/27318290/why-can-i-pass-an-instance-method-to-multiprocessing-process-but-not-a-multipro
with some possible solutions to parallelize a class member rather than an external function.
Answer 4, the easiest, fails, because the wrapper function contains a new instance of the class, not the one with all the attributes etc.

#8 - 06/22/2018 01:02 PM - Tibaldo Luigi

Jürgen started working on the pickle-ability of gammalib/ctools objects (#1938), this issue will be on hold until that effort is completed.

#9 - 07/02/2018 06:22 PM - Tibaldo Luigi

- % Done changed from 10 to 20

With the upgrade described in #1938 it was easy to parallelize cslightcrv. The only open issue remains the logging: the log file currently gets mangled by the multiple processes running in parallel.

#10 - 07/03/2018 12:46 PM - Knödlseder Jürgen

I did not notice a mangling of the log file on Python 2, but I had to change the logic of the multiprocessing since in Python 2 class methods cannot be pickled. Here's the code that I implemented:

```
# ========================================== #
# Global functions for multiprocessing support #
# ========================================== #
def _multiprocessing_func_wrapper(args):
    return _multiprocessing_func(*args)
def _multiprocessing_func(cls, i):
    return cls._timebin(i)
```

and

```
# Create pool of workers
from multiprocessing import Pool
pool = Pool(processes = self._nthreads)

# Run time bin analysis in parallel with map
args    = [(self, i) for i in range(self._tbins.size())]
results = pool.map(_multiprocessing_func_wrapper, args)

# Close pool and join
pool.close()
pool.join()
```

I also did some timing of a 30 min Crab observation for which 5 time bins were requested:

nthreads	Wall clock (sec)	CPU clock (sec)
1	15	12.3
2	12	3.1
3	10	3.2
4	9	3.3
5	9	3.6
6	9	3.5
7	9	3.4
8	9	3.5

Note that the CPU clock seems to measure only the time spent outside the _timebin method in case of multiprocessing (nthreads != 1). The wall clock is of course subject to fluctuations, but decreases with increasing number of threads. There is a small increase of the CPU clock with number of threads, which may be related to the overhead of pickling the cslightcrv instance.

#11 - 07/03/2018 01:31 PM - Tibaldo Luigi

I tested in Python 3 with your modifications:

- still works fine
- log file is same as before

Do you really need the following piece?

```
# Close pool and join
pool.close()
pool.join()
```

I thought this was not needed for the map method.

#12 - 07/03/2018 02:32 PM - Knödlseder Jürgen

user#266 wrote:

> I tested in Python 3 with your modifications:
>
> - still works fine
> - log file is same as before
>
> Do you really need the following piece?
> [...]
>
> I thought this was not needed for the map method.

I'm not a multiprocessing expert, but it seems to be good to do this (see https://stackoverflow.com/questions/38271547/when-should-we-call-multiprocessing-pool-join)

#13 - 07/03/2018 03:34 PM - Knödlseder Jürgen

I added a GLog::buffer() method to recover the buffer of a logger into a string, and I used this method in cslightcrv (see below). Before calling the _timebin method the logger is cleared and the buffer size is set to a large value so that normally the buffer will never be flushed into a file (I guess even that the clearing unsets the link to the log file). After _timebin the output of the buffer is recovered in a string, which is then passed together with other computing information to the client.

```
# ============================================ #
# Global functions for multiprocessing support #
# ============================================ #
def _multiprocessing_func_wrapper(args):
  return _multiprocessing_func(*args)
def _multiprocessing_func(cls, i):

  # Initialise thread logger
  cls._log.clear()
  cls._log.buffer_size(100000)

  # Compute light curve bin
  cstart  = cls.celapse()
  result  = cls._timebin(i)
  celapse = cls.celapse() - cstart
  buffer  = cls._log.buffer()

  # Close logger
  cls._log.close()

  # Collect thread information
  info = {'celapse': celapse, 'log': buffer}

  # Return light curve bin result and thread information
  return result, info
```

In the client, I added the following code to demangle the output and to insert the thread logger into the main logging stream:

```
# Create pool of workers
from multiprocessing import Pool
pool = Pool(processes = self._nthreads)

# Run time bin analysis in parallel with map
args        = [(self, i) for i in range(self._tbins.size())]
poolresults = pool.map(_multiprocessing_func_wrapper, args)

# Close pool and join
pool.close()
pool.join()

# Construct results
results = []
for i in range(self._tbins.size()):
  results.append(poolresults[i][0])
  self._log_string(gammalib.TERSE, poolresults[i][1]['log'], False)
```

#14 - 07/03/2018 03:34 PM - Knödlseder Jürgen

Forgot: all code is now in devel (GammaLib/ctools), hence you can start a fresh branch from devel to continue your developments.

#15 - 07/04/2018 11:11 AM - Tibaldo Luigi

- % Done changed from 20 to 30

cslightcrv is now fully parallelized. There is still an issue with log files that sometimes get mangled (when the buffer is almost full at the moment the parallelization starts) that Jürgen is investigating.

In the mean time I'll move on to parallelize the following scripts:

- csphasecrv
- csspec
- cssens
- csphagen

#16 - 07/04/2018 01:47 PM - Knödlseder Jürgen

I fixed the log file mangling issue. The code is in the devel branch.

#17 - 07/04/2018 03:23 PM - Tibaldo Luigi

Working on the parallelization of csspec I discovered a typo in gammalib GEbounds.i
The piece of code

```
state = tuple([self.emin(i) for i in range(self.size())]), \
        tuple([self.emin(i) for i in range(self.size())])
```

has to be replaced by

```
state = tuple([self.emin(i) for i in range(self.size())]), \
        tuple([self.emax(i) for i in range(self.size())])
```

The change is committed in my branch 2421-parallelise-cscripts

#18 - 07/04/2018 04:47 PM - Knödlseder Jürgen

Thanks for catching that one. The fix is merged into devel.

#19 - 07/04/2018 05:41 PM - Tibaldo Luigi

- % Done changed from 30 to 60

Scripts parallelised:

- cslightcrv
- csphasecrv
- csspec
- csphagen
  Still to be done: cssens, cross-references between scripts (make sure mp settings are passed down properly), documentation.

#20 - 07/05/2018 03:59 PM - Tibaldo Luigi

- % Done changed from 60 to 80

Scripts parallelised:

- cslightcrv
- csphasecrv
- csspec (only for method SLICE)
- cssens
- csphagen
- cspull
- cstsdist

I checked for cross-references, and the only parallelised script used in other scripts is csphagen, which is used in cslightcrv and csphasecrv (trhough obsutils.get_onoff_obs). If things are left as they are (nthreads not specified in obsutils) for the default value if we have n CPUs there will be n^2 processes spawned when the method is called. This may not be desirable. How about we set nthreads = 1 in obsutils.get_onoff_obs, so that the number of total processes will stay the same as the number of CPUs or whatever the user requested?

Working on the documentation now. I will update the reference manual. We may think of a general introduction to multiprocessing in ctools/cscripts to put upstream somewhere.

#21 - 07/05/2018 04:43 PM - Tibaldo Luigi

- % Done changed from 80 to 90

Reference manual updated for all scripts.
Remaining things to sort out:

- csphagen cross-references (cf. previous update)
- general introduction to multiprocessing in documentation?

#22 - 07/05/2018 05:09 PM - Tibaldo Luigi

I implemented a way to handle the usage of csphagen through obsutils which seems reasonable to me. obsutils.get_onoff_obs has an additional parameter nthreads set by default to 0. This implies the default behavior for csphagen w.r.t. multiprocessing. However, in cslightsrv and csphagen I pass nthreads=1, so that the original number of processes set in those scripts is preserved.
For the future this is flexible enough that other possibilites are open/easy to implement.

#23 - 07/06/2018 11:31 AM - Knödlseder Jürgen

user#266 wrote:

> I implemented a way to handle the usage of csphagen through obsutils which seems reasonable to me. obsutils.get_onoff_obs has an additional parameter nthreads set by default to 0. This implies the default behavior for csphagen w.r.t. multiprocessing. However, in cslightsrv and csphagen I pass nthreads=1, so that the original number of processes set in those scripts is preserved.
> For the future this is flexible enough that other possibilites are open/easy to implement.

That sounds like a good solution.

#24 - 07/06/2018 11:58 AM - Tibaldo Luigi

Good, I implemented the same logic for obsutils.sim, which is used in cspull (and, formally, in cssens and cstsdist, but there the On/Off analysis is not implemented).
I also updated the reference manual for all tools. Do you think we need a general introduction to multiprocessing in ctolls/cscripts?

#25 - 07/06/2018 01:55 PM - Knödlseder Jürgen

Since for the end user things are pretty transparent I'm not sure that an "introduction" is needed. But maybe a page describing how multiprocessing is handled in ctools and cscripts would probably be valuable.

The question is whether before doing this we should add a nthreads parameter to all OpenMP parallelised ctools so that they behave from the User perspective exactly like cscripts. We should also note that OpenMP supports depends on the compiler that is used, and that currently, no OpenMP support comes by default on Mac OS X. We could even describe how to make OpenMP available on a Mac.

#26 - 07/06/2018 02:06 PM - Tibaldo Luigi

- Status changed from In Progress to Pull request

- % Done changed from 90 to 100

Okay, then as discussed in person let's consider this issue completed, and defer to another one the harmonization between ctools and cscripts and the general documentation.

#27 - 07/17/2018 03:00 PM - Knödlseder Jürgen

- Status changed from Pull request to Closed

Merged into devel.

#28 - 07/18/2018 07:25 PM - Knödlseder Jürgen

- Status changed from Closed to Feedback

The ctools make check seems to hang in the cslightcrv pickling test for Python 3. See https://cta-jenkins.irap.omp.eu/job/ctools-python/670/.

Have you tried a unit test with Python 3?

#29 - 07/19/2018 06:18 AM - Tibaldo Luigi

Yes, it runs successfully in about 10 minutes in my Python 3.5.4 installation. I just run it again: here you are the results:

```
PASS: test_python_ctools.sh
PASS: test_python_cscripts.sh
PASS: test_examples.py
============================================================================
Testsuite summary for ctools 1.6.0.dev1
============================================================================
# TOTAL: 3
# PASS:  3
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
```

#30 - 07/19/2018 03:39 PM - Knödlseder Jürgen

The code hangs in the following loop in GObservations::likelihood::eval

```
// Loop over all observations. The omp for directive will deal
// with the iterations on the differents threads.
#pragma omp for
for (int i = 0; i < m_this->size(); ++i) {

  // Compute likelihood
  *cpy_value += m_this->m_obs[i]->likelihood(cpy_model,
                                             cpy_gradient,
                                             cpy_curvature,
                                             cpy_npred);

} // endfor: looped over observations
```

which implies that there is a conflict between the Python multiprocessing and OpenMP. This also explains why this does not show up on Mac OS X, since Mac OS X does not support OpenMP.

#31 - 07/19/2018 11:54 PM - Knödlseder Jürgen

Looks like OpenMP is not compatible with fork(): https://stackoverflow.com/questions/49049388/understanding-openmp-shortcomings-regarding-fork

#32 - 07/20/2018 10:25 AM - Knödlseder Jürgen

- Status changed from Feedback to In Progress

- % Done changed from 100 to 90

It turns out that when setting the maximum number of threads to 1 using the omp_set_num_threads() function the tests run through.

I therefore added nthreads parameters to ctlike and cterror, and some internal mechanism that sets the number of threads through the omp_set_num_threads() function in case that nthreads>0. At the same time, I set the ctlike nthreads parameter to 1 in the parallelised cscripts that use ctlike, so that no conflict occurs when multiprocessing is used.

It's still a surprise to me that no issue occurs when simulating events in ctobssim. Let's see if the current fix resolves the issue.

#33 - 07/20/2018 04:49 PM - Knödlseder Jürgen

It looks like the modifications did it, but I will wait until a full night continuous integrations runs through. In addition, I will add unit tests for pickling for all tools, as they exists now for cslightcrv (since this was the unit test that posed problems).

#34 - 07/25/2018 10:21 PM - Knödlseder Jürgen

I found a bug in the main loop of cspull and cstsdist which was first executed in parallel, and then another time in series, which made the overall execution time longer. An else statement was in fact missing for the case that nthreads is not larger than 1.

I merged the bug fix into devel.

#35 - 07/28/2018 12:10 AM - Knödlseder Jürgen

- Status changed from In Progress to Closed

- % Done changed from 90 to 100

Things seem to work now. Close the issue.