GROMACS - Task #869

Make analysis data histogramming and multipoint data easier to use in parallel

01/26/2012 07:38 PM - Teemu Murtola

Status:	In Progress	
Priority:	Normal	
Assignee:	Teemu Murtola	
Category:	analysis tools	
Target version:		
Difficulty:	uncategorized	

Description

The current parallelization design in src/gromacs/analysisdata/ is such that when frames are added to AnalysisDataStorage (in any order, within implementation limitations), it internally stores them and calls notification methods in AnalysisDataModuleInterface such that they are always called in the same order. But this is not currently implemented for multipoint data. This makes in particular histogramming difficult in parallel scenarios, since the input data is often multipoint in nature.

There are three issues that could be done to improve the situation:

- Implement full support for storage of older frames of multipoint data. This would allow multipoint data to be treated the same as simple data in AnalysisDataStorage. This is not ideal for histogramming, where the input data may contain a lot of points, which would not need to be stored otherwise. But as a fallback, this would still be useful to implement.
- Implement parallel-enabled modules, that allow notification methods to be called with out-of-order frames. These modules could use AnalysisDataStorage internally to make the frames sequential later in the processing chain.
- For histograms in particular, it could be worth refactoring them such that in addition to AnalysisDataModuleInterface subclasses, there would be implementations that only derive from AbstractAnalysisData. The latter would provide explicit methods to add points instead of getting them through the module interface. Nearly all of implementation could be shared between the two, so this wouldn't probably be much work. This would simplify tools that mainly calculate histograms from large set of values (e.g., RDF calculation) in that they wouldn't need to add passthrough AnalysisData object, whose output might not be very useful outside the tool.

Related issues:

Related to GROMACS - Feature #868: Implement parallelization support to analy...	**In Progress**
Related to GROMACS - Task #1010: Better support for multiple AnalysisData dat...	**In Progress**

Associated revisions

Revision 64bc3746 - 06/12/2013 07:30 AM - Teemu Murtola

Better averaging for analysis data modules.

Added an AnalysisDataFrameAverager for functionality common to analysis
data modules that compute averages over frames. Currently, it only
accumulates the average and variance in double precision (using better
formula than before), but provides a useful base for implementing other
functionality:
- Block averaging or other methods of better estimating the error.
- Parallelization support to allow these modules to work in parallel
after #869 is implemented.
For all of these, after the implementation is done in this common class,
it is easy to have that functionality in all the modules. Some
interface changes may be required for some of the above, though.

Also some improvements to AnalysisDataAverageModule documentation.

Change-Id: I06ae7a92d36a0ee7e2fc0a1602ac40e6b8212d1d

Revision dc8edac5 - 06/13/2013 09:45 AM - Teemu Murtola

Better averaging for analysis data modules.

Added an AnalysisDataFrameAverager for functionality common to analysis
data modules that compute averages over frames. Currently, it only
accumulates the average and variance in double precision (using better
formula than before), but provides a useful base for implementing other
functionality:
- Block averaging or other methods of better estimating the error.

- Parallelization support to allow these modules to work in parallel
after #869 is implemented.
For all of these, after the implementation is done in this common class,
it is easy to have that functionality in all the modules. Some
interface changes may be required for some of the above, though.

Also some improvements to AnalysisDataAverageModule documentation.

Change-Id: I06ae7a92d36a0ee7e2fc0a1602ac40e6b8212d1d

Revision 54919371 - 06/24/2013 07:11 AM - Teemu Murtola

More flexible input data for analysisdata tests.

Instead of somewhat inflexible parsing from a static real[] array,
construct the input data objects by explicit method calls to add frames
and point sets. Convert the existing tests to construct the input data
this way.

Increase coverage of multipoint tests for AnalysisData by using
point sets that do not cover the full range of columns.
Improve test error messages and fix issues found while doing this.

Prerequisite for unit tests for #869 and #1010.

Change-Id: Idd0831d9bbf8e59b6edfab758cd53881133e1f3a

Revision d7fb6e07 - 06/24/2013 07:15 AM - Teemu Murtola

Restructure AnalysisData unit tests.

Add test fixtures and other machinery that allows using the same test
logic for multiple different input data using typed tests from Google
Test.

Avoids code duplication in unit tests for #869 and #1010.

Change-Id: I74962aa4ed0741329b3dbcd40663fc94a3bc96b5

Revision 06a0c8a5 - 07/03/2013 07:54 PM - Teemu Murtola

Split AnalysisDataStorageFrame into two.

Main changes:
- Move parts of AnalysisDataStorageFrame that are only used internally
by the data storage into a separate class, which is internal to
datastorage.cpp. The remaining public interface now only contains
methods that are required to construct the in-progress frame/point
set.
- Make AnalysisDataStorageFrame have a separate vector of column values,
which then get transfered into the actual storage when necessary.
This part is required for handling multipoint storage.
- Only keep AnalysisDataStorageFrame objects for in-progress frames.
Without this part, the memory usage of full storage would double; now
the memory usage only increases proportional to the number of
concurrent data frames.
- Recycle the AnalysisDataStorageFrame objects for subsequent frames.
Avoids memory allocation, as the temporary column values don't need to
be reallocated for each frame.

These changes clarify the responsibilities in the code, and allow for a
much cleaner implementation of multipoint storage (subsequent commit).

The implementation may not be as clean as possible, but it will see
further changes in subsequent commits.

Refactoring in preparation for #869 and #1010, no functional change.

Change-Id: I40928ff10c5aded3b094604b37349b9fd174d267

Revision 5692d555 - 07/17/2013 08:49 PM - Teemu Murtola

Support for storing multipoint analysis data.

Main changes:

- The storage frames now store a vector of "point set info" objects,
which partition the vector of values into point sets, and annotate
those sets with extra information (currently, the index of the first
column).
- The implementation of the frame access wrappers in dataframe.* are
adapted similarly, and methods are added to allow access to multiple
point sets within a single frame.
- Added tests for multipoint storage.
- Other changes are adapting to these interface changes.

Still doesn't work perfectly; full support for addColumnModule()
requires reworking the logic in dataproxy.cpp.

Part of #869.

Change-Id: I21141bad83b3f2bb72ece9146aaec02792e6cd0e

Revision 7a55aae0 - 09/05/2013 09:51 PM - Teemu Murtola

AbstractAnalysisData module handling into a separate class.

Move all logic related to maintaining the list of attached modules,
checking their compatibility with the data, and notifying them from
AbstractAnalysisData into a separate AnalysisDataModuleManager class.
AbstractAnalysisData now contains an instance of this class, and
provires a moduleManager() protected method for derived classes to
access it.

Module notification methods are no longer responsible of keeping the
frame count. Instead, made the frameCount() method pure virtual and
added a method in AnalysisDataStorage to conveniently implement this
method.

Moved the responsibility of calling all data notification methods into
AnalysisDataStorage (including notifyDataStart() and
notifyDataFinish()). This allows modules that use AnalysisDataStorage to
not even know of the implementation details of the module manager.

Improved the semantics of setColumnCount() etc. such that they now
immediately check the compatibility of data modules and throw if there
is a problem.

Rationale:
- Remove implementation details from the installed abstractdata.h
header: AbstractAnalysisData is one of the main public interfaces in
the module, and it exposed unnecessary complexity in the header. Now
only the subclasses that actually need it see these details.
In particular with #869, the module notification interface is only
going to get more complex.
- Reduce the responsibilities of AbstractAnalysisData: now it only
keeps track of the dimensionality of the data (and notifies the
module manager of relevant changes). Now it is closer to a pure
interface, which is possibly should be.
- Makes it possible to remove friendship between AbstractAnalysisData
and AnalysisDataStorage. This allows further simplification in
datastorage.cpp.

Related to #869.

Change-Id: Ic42d4b27bb6d1e445e3f74aa06ae52b354bf4403

Revision 08b94180 - 09/06/2013 05:37 AM - Teemu Murtola

Support for parallel data modules.

Main changes:
- Add parallelDataStarted() to AnalysisDataModuleInterface to
initialize modules for parallel processing of data. If this method
is called, and the module returns true, then the module accepts that
frame notification methods can be called in relaxed order: the frames
can be started in any order, and multiple frames may be active
simultaneously.
- Implement this method in the existing modules, either explicitly or
by using convenience base classes.
- Add notification methods to AnalysisDataModuleManager to notify

parallel frames. These methods are called as soon as the data is available to notify those modules that accept parallel data. Existing methods were changed to only notify the serial modules.
- Make AnalysisDataStorage call the new notification methods appropriately. In most cases, this absorbs the complexity that these new methods bring.
- Update (part of) the unit tests to make it possible to test this relaxed frame ordering.

Part of #869.

Change-Id: I3cccf78a09221b8181c3d8b217a793c996d0c522

History

#1 - 12/29/2012 02:19 PM - Teemu Murtola

- Target version set to 5.0

If we want to have parallelization support in 5.0 for the tools, then this would be good to do for that as well.

#2 - 03/27/2013 07:38 PM - Roland Schulz

Has anyone started any work on this? (I know it isn't marked as in progress but just want to be sure)

#3 - 03/27/2013 07:54 PM - Teemu Murtola

I'm not aware of anything else than those e-mails with Anders where you have also been involved in. I have thought about the design a bit, but not on a very concrete level, and have not done any implementation. Are you planning to start? Just let me know if you want me to share my thoughts on the design, either before you do anything or after you have done an initial design yourself. The three bullets are more or less independent of each other.

#4 - 04/04/2013 01:16 AM - Roland Schulz

What we really need is a sparse storage for analysis data. We wrote a contact counting module and for most pairs the number of contacts is 0. So storing this using as many columns as the number of pairs takes much more memory than necessary. An alternative would be to use multipoint and use 3 columns (molA,molB,#contacts) and as many points as non-zero pairs. For that we would only need the first bullet point. But maybe we should add directly to datastorage an option for sparse data, and it would automatically store only non-zero columns. Do you think this would be useful for other modules and thus a useful addition?

To implement the first bullet point, should one AnalysisDataValue::value_ change from real to vector<real> and then simply adding the support in datastorage to write also if it is multipoint? Or is it less obvious than that?

#5 - 04/04/2013 05:57 AM - Teemu Murtola

Sparse storage could be an option as well, but as you say, it is also possible to use multipoint data for the same purpose; it depends a bit on the way you want to write out the results. But sparse storage shares a lot in common with the storage of multipoint data, so it could be relatively straightforward to implement both at the same time.

I think that rather than changing AnalysisDataValue, it would be better to add support for directly storing multiple sets of points, i.e., store the multipoint data as it is coming in, so that AnalysisDataPointSetRef doesn't need to be changed. What I had in mind was that instead of storing a single std::vector<AnalysisDataValue>, one could conceptually store an std::vector<AnalysisDataPointSet>, where each PointSet contains a std::vector<AnalysisDataValue>. This should make it easier to understand. As an implementation detail, it could be more efficient to still store a single std::vector<AnalysisDataValue> and an additional std::vector<PointSetIndex>, where PointSetIndex would allow one to find the indices in the first vector that correspond to a particular point set.

With this approach, sparse storage should be essentially the same as multipoint storage, simply representing the non-zero columns as distinct point sets. There would only need to be extra checks that the same column does not appear more than once. I would also suggest to instead of making the storage sparse on the zero/non-zero aspect, instead make it sparse on the isPresent() aspect.

#6 - 04/26/2013 05:47 AM - Teemu Murtola

- Status changed from New to Accepted

#7 - 05/11/2013 12:07 PM - Teemu Murtola

Roland: Have you done something on the first point/on the sparse storage? I've been thinking of implementing the first option in #1010, and that would most likely be able to reuse exactly the same changes to the storage as the first point here. I just want to know whether doing that will conflict with something that you have been working on, and if so, whether that is OK and/or how soon could you have your changes merged to the main branch?

#8 - 05/30/2013 08:11 PM - Teemu Murtola

Roland: ping. Do you have any status update on this? Or is it OK (not conflicting with and not duplicating work you've done) if I go on and implement #1010 and the first point from here?

#9 - 05/31/2013 06:15 AM - Roland Schulz

Sorry! I forgot to reply to your previous message. I currently don't have any time on working on this and only John worked on parallel analysis. But as far as he told me 2 weeks ago he also didn't have time to work on this aspect. So it will be great if you are doing point 1.

#10 - 05/31/2013 12:48 PM - Teemu Murtola

- Status changed from Accepted to In Progress

- Assignee set to Teemu Murtola

#11 - 06/09/2013 12:09 PM - David van der Spoel

Just wanted to mention I intend to work on parallel analysis in the near future (have hired a programmer for the summer). If we get one or two tools to work in parallel we intend to write a paper about it. More later.

#12 - 06/09/2013 12:13 PM - Roland Schulz

Are you planning to use threads or MPI? If MPI are you going to base it on https://gerrit.gromacs.org/#/c/1316/ ?

#13 - 06/11/2013 07:44 PM - Teemu Murtola

https://gerrit.gromacs.org/#/c/2441/ now implements the first point from here. Will look at the others after #1010.

#14 - 06/27/2013 07:56 PM - Teemu Murtola

The second point (the most important part of this task) is now implemented in https://gerrit.gromacs.org/#/c/2462/.

#15 - 06/20/2014 10:45 AM - Erik Lindahl

- Target version changed from 5.0 to 5.x

#16 - 07/14/2014 11:51 AM - Teemu Murtola

- Project changed from Next-generation analysis tools to GROMACS

- Category set to analysis tools

#17 - 07/11/2016 08:28 PM - Mark Abraham

- Target version deleted (5.x)