Geosci. Model Dev. Discuss., 7, C3327–C3333, 2015
www.geosci-model-dev-discuss.net/7/C3327/2015/

Geoscientific Model Development Discussions

Interactive comment on "The generic simulation cell method for developing extensible, efficient and readable parallel computational models" *by* I. Honkonen

I. Honkonen

ilja.j.honkonen@nasa.gov

Received and published: 10 February 2015

General remarks

I thank the referees for insightful comments and suggestions. I have made major changes to the manuscript based on their input and the lessons learned from using and futher developing the code.

Point-by-point replies

Anonymous referee

Point of criticism 1 - ...more details on the presented approach ... memory layout: I have added a note on the memory layout to introduction and a paragraph to discussion section about the performance implications.

Point of criticism 2 - ...suggest replacing figs. 2-4 by parallel example with fewer external dependecies...: I reworked the examples, now the first 4 do not require external dependencies besides MPI, the game of life programs are parallel and use MPI directly.

Technical comments

P 4581, l 26: missing reference to Conway's Game of Life: added

P 4582, ll 3-5: This is very specific ... misleading on several locations.: I removed references to copy pasting and now use incorporate. I do not think without modification is misleading as model code does not have to be modified in order to be combined with other models. In a real program e.g. calls to advection functions in the new figure 5 would be hidden behind another layer of indirection so that even copy pasting would not be required for figure 6. For example each model could be given as a template parameter to a simulation or time loop class which would automatically do the same thing that figure 6 is showing. However showing this would be out of scope for this work.

P 4583, l 1: from from typo?: fixed

P 4585, l 1: boost::indetermined would be clearer: fixed

P 4591, l 1: requerd typo: fixed

Sylwester Arabas

Main remark 1 - ...begin the paper with presentation of two complete simple examples implemented (i) using the idiom/library and (ii) without using it...: Figures 2 and 3 now show parallel games of life with and without using the cell class and use MPI directly. I added another figure (4 in new version) showing the difference between using and not using the presented cell class in a program and added related text to several sections.

Main remark 2 - ...important to address at the very beginning of the paper, is to define clearly the elemental assumptions of the library design and its implications...: I added discussion about the memory layout of variables and its potential effect on program speed as well as discussion about the effect of using the cell class on threading and vectorization of code. An important added point was the comparison of generic cell class with std::tuple as with C++14 the behavior of both is very close to each other, except that the cell class provides nicer syntax and support for MPI programs.

Main remark 3 - ...evade discussion on how the introduced library design limits one in applying some constructs that would intuitively be expected to be attainable with an object-oriented library...: I do not see how gensimcell limits the constructs presented by the referee and also in some cases I think the missing functionality is not applicable to gensimcell. For example referee's listing B uses a struct with two integer members as the cell type but a generic simulation cell with two integer variables and a transfer policy of either Always_Transfer or Never_Transfer (or std::tuple even without C++14 as the example uses integers to refer to variables) would work just as well since the memory layout of variables is the same as in the struct. This probably shows best that some of the supposedly missing functionality belongs in my opinion into another layer of abstraction, i.e. a grid library of one Blitz++ matrix or nested std::arrays, and would seem to render rest of the remark not applicable to gensimcell. Naturally e.g. the loop-free construct shown in listing B, perhaps better known as pattern matching which is separate from object oriented features of a language, can be written for con-

tainers other than Blitz++ array. In C++11 this abstraction exists for single containers (std::transform) and writing a version that supports nested containers seems easy. Using std::transform does require that cell data be the inner most container of simulation data, e.g. std::array<std::tuple<...>, N> instead of std::tuple<std::array<..., N>, ...>, but again writing a version for a dedicated container (grid library) also seems easy but would be out of scope of this paper. Using only standard containers and language constructs as much as possible in the presented examples was done to minimize the number of external dependencies and make the code easier to learn/understand. Examples in the corrected paper use even fewer external dependencies and new figure 3 is a parallel 1-dimensional version of referee's listing A.

Other comments

I suggest changing "The generic ..." into "A generic ...": fixed

Many GMD papers mention the program name in the title...: The title is already quite long and is more about the method than my implementation of it but I added a distinction between these two to the abstract.

I suggest mentioning domain decomposition and the geoscientific context in the abstract: added

Offering a snapshot of the code repository as an electronic supplement to the paper would make long-term archival of the paper viable: I'll add a snapshot of the source code repository as a supplement.

I suggest summarising the library API in one place...: fixed

Why not use "Listing" instead of "Figure"?: fixed

...simulation results from the first example...: output of 1st example is now in ASCII format, added to text.

Subsections 1.1 and 4.1 are the only subsections within their parent sections...: fixed

...reference to game of life...: fixed

...listing all library dependencies...: fixed

The statement in line 18 on page 7/4583 ... discuss that using it implies loop-based syntax: While this is a feature of the cell container and not individual cells the reworked examples do not state this so no changes.

All mentions of "parallel computational model" seem to assume domain decomposition - this is misleading as parallelism may take also other forms: It is true that in this paper domain decomposition is used but I do not think this is mandatory as this is again a feature of the cell container, i.e. grid library, and not single cells. This is discussed briefly in the section "combination of three simulations".

...the [] operator is chosen arbitrarily...: It was not because operator [] can only take one argument so it seemed logical for returning a reference to data of one variable. The () operator could perhaps be used for returning references to data of several variables inside an std::tuple in which case, if given only one argument, returning one reference in a tuple would be the least surprising choice. Also many STL containers use [] and not () for accessing data. I added a footnote to the text about this.

The listing in Figure 7 is syntactically incorrect...: fixed

...would prevent understanding "indeterminate" as a part of the sentence...: fixed

...private/public distinction may be omitted from the listings...: fixed

Describing the advection example, it is worth mentioning which algorithm it implements: fixed

...presented examples feature numerous constructs or choices that seem not in line with the embraced object-orientation and modern C++11 standard...: Standard containers were used to minimize the number of external dependencies and the asso-

ciated learning curve. DCCRG does not use boost::mpi as we found that unnecessary copies of data were created by serialization used by boost::mpi when transferring data. When simulation data already fills most of the available memory (as e.g. in http://dx.doi.org/10.1016/j.jastp.2014.08.012) creating extra copies is not possible. Granted we might have been using boost::mpi incorrectly but in the end it was significantly easier to use MPI's C API instead. The rand function from C was used because using the C++ one would add several unnecessary lines of code and good quality random numbers were not required. Neither arrays nor random numbers are used in the reworked examples so these changes are no longer required.

The sentence "no changes to existing code are required for combining models" concerns models that where written in C++ using the very same library aimed at coupling models. It is misleading: The entire sentence is "The changes required for combining and coupling models are minimal and in the presented examples no changes to existing code are required for combining models." Note the "in the presented examples" part, I do not think it is misleading.

The reference to Stroustrup 1999 in line 23 on page 4579 seems not a best match for the sentence...: Yes unfortunately the reference doesn't mention templates by name, I added your suggestion also as a reference.

Why #include "" and not #include <> is used for system headers?: No particular reason but it turns out that the functionality of neither one is specified by the standard, it is only guaranteed that if the "" version fails then the search is repeated as if <> had been used. The relevant discussion on stackoverflow has more details and references to the C standard (piCookie's answer): https://stackoverflow.com/questions/21593

...suggest skipping the comments on copy-pasting code: The new version uses the term incorporate.

...calls for using the "-march=native" flag...: I tested with "-march=native -mtune=native" with GCC on another machine (not supported on my Mac at the moment) and the

performance difference was about 0.2 %, on the hardware described in the paper the performance difference with Clang was about 3%. I did not test with an Intel compiler as my trial period expired but the difference probably would not be much larger than with other compilers. I noted this in the new version.

The vague statement in section 5 that gensimcell...: I clarified that gensimcell shouldn't affect performance of neither serial nor parallel programs.

I suggest skipping "return 0"...: fixed

I suggest not naming something a "traditional scientific code"...: fixed

typo: varible: fixed

typo: requred: fixed

markup leftover: fixed

underscores are not readable in several places in the listings...: I switched to a different font in listings, now underscores are better visible.