GUST: a C++ based simulation framework

Pekka Janhunen and Arto Sandroos

Finnish Meteorological Institute, Space Research, Helsinki, Finland,
pekka.janhunen@fmi.fi,
WWW home page: http://www.space.fmi.fi/~pjanhune

Abstract. We present a new C++-based simulation framework GUST (General-Use Simulation Templates) whose development work is ongoing at our institute. GUST is mainly intended for plasma simulation and will support at least fluid (magnetohydrodynamic, MHD), particle and hybrid simulations. It makes heavy use of the C++ template mechanism and provides automatic grid adaptation and parallelisation. A graphical user interface for generating GUST applications without coding in C++ is part of the system and enables GUST to be used by non-programmers as well. This presentation will be a status report of GUST.

1 Introduction

As a general background, in our space plasma physics research group we have written codes for the three main types of plasma simulation: MHD (fluid) [1, 2], particle [3] and hybrid simulation [4]. The codes work well, but they are not very flexible. Only the particle codes are parallelised. The codes share some common visualisation tools, but the sharing has not been formalised. The MHD and hybrid codes use adaptive grid techniques so that these codes are rather competitive even without parallelisation.

A clear need in the group has arised to make the codes more flexible, i.e. more easily adaptable to new physical situations. There is also a need for scientists without programming skills to run the codes as well. Finally, there is the need to parallelise the MHD and hybrid codes. These needs basically arise from the increasing power of computers, which means that software and manpower are more and more of a bottleneck. Although we know well the situation only in our research group, we expect that a similar situation exists at the moment in many other groups as well.

2 Design criteria

To address these needs we decided to build a new unified simulation framework “GUST” (General Use Simulation Templates), to which we assigned the following design criteria:

1. It must contain everything that one needs to do simulations: design phase, running and visualisation.

2. Automatic grid adaptation and parallelisation support.
3. Support at least fluid, particle and hybrid simulations, but it should also be as generic as possible.
4. Extensible by programming but possible to use also by non-programmers.
5. Nearly top speed execution (accepting $\sim$ 50% reduction).

Automatic grid adaptation and parallelisation means that the system provides nontrivial added value for the user: if the user would write the simulation for his/her particular application from scratch, it could take years of programming.

To achieve as much genericity as possible we selected the C++ language and its template mechanism as the tool for expressing generic algorithms. Our earlier simulation codes are written in C++, although they do not make full use of the language's present standardised capabilities. The first C++ ISO standard first appeared in 1998 and nearly standard-obeying compilers are now widely available. We think that the time is now right to employ the C++ template mechanism seriously for scientific computation. By using templates we achieve top speed (at least in theory) since we largely avoid the runtime overheads that would otherwise follow from generic coding (e.g., if one would use non-inlinable virtual functions instead of templates). A small penalty we have to accept is that recompilation of the main program is needed every time the user changes some aspect of it. On modern personal computers the compilation time is according to our measurements only 5-10 seconds, however.

3 Overall structure of GUST

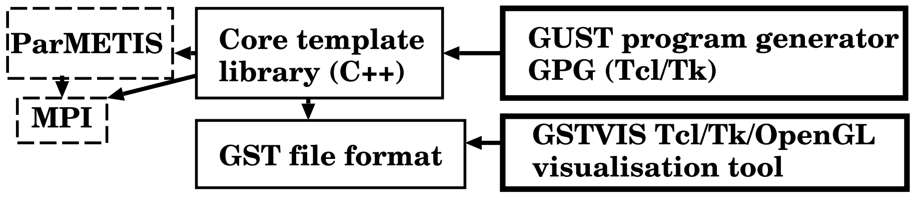
ParMETIS
MPI
Core template library (C++)
GUST program generator GPG (Tcl/Tk)
GST file format
GSTVIS Tcl/Tk/OpenGL visualisation tool

Fig. 1. Schematic representation of the components of GUST.

GUST has four main parts (Figure 1):

1. The core template library. Because of the heavy use of templates most of its appears in C++ header files. The library implements e.g. general grid classes. It uses the Message Passing Interface (MPI) library [5] for parallelisation and the ParMETIS [6] general graph partitioning library (which in turn also makes MPI calls) for distributing the adaptive grid elements optimally to different processors in case of parallel execution.

2. A GUST-specific binary "GST" file format definition. All simulation results are stored in GST files. As the need arises, we intend to provide (one-way) conversion routines to commonly used formats such as CDF [11], HDF or HDF5 [12].
3. A GUST program generator (GPG) having a graphical user interface (GUI) which enables non-programmer users to automatically generate a GUST C++ main program which uses the GST template library, as well as manage GUST runs. The GPG is written in the Tcl/Tk language [7]. All details about the simulation are kept in a "document" file which the user edits with the GPG and from which the GPG generates C++ code when needed. The document file is stored in a Tcl/Tk-specific textual format. The document model means that using the GPG is in many ways similar to using common office software.
4. A visualisation program GSTVIS for viewing the contents of GST files and generating plots out of them. At the moment of this writing, GSTVIS coding has only been started. Except for being more general, GSTVIS will roughly follow our earlier tool "hcvis". A file translator from GST format to HCVIS format exists so GST files can already now be viewed with hcvis. Hcvis and GSTVIS are written in Tcl/Tk for the GUI part and in C++/OpenGL [8] for the data processing and drawing part. The Togl module [10] is used to enable OpenGL drawing in Tcl/Tk widgets.

To keep porting and maintenance of the system easy GUST depends on as few libraries as possible. The MPI library is the de facto and also rigorously defined parallelisation standard. The data distribution of a given adapted grid is a complex problem which is however solved by the ParMETIS library. The ParMETIS library depends only on the MPI library and is in our experience easy to compile and port. For the GPG and GSTVIS, a GUI support toolset is needed. We selected Tcl/Tk because Tcl/Tk code is compact and easy to understand. Being an interpreted language, Tcl/Tk is inherently slow compared to C/C++, but here we use it only for parts where speed is completely uncritical. For graphics rendering we use the Mesa implementation [9] of OpenGL because it is fast, widely used, supports offscreen rendering and can interface to Tcl/Tk widgets through the Togl module. In one phase we considered using a more complete toolset such as VTK [13] or QT [14], but finally decided that using them would add too much complexity (and thus potential future maintenance and porting problems).

The GPG is the component of GUST that the user mainly interacts with. Even if a user wants to extend the system by writing pieces of C++ code, these code fragments can in most cases be part of GPG documents, from which they are spliced in the generated C++ code automatically. On the other hand, it is also possible and not difficult to skip GPG and instead write directly the C++ main program which usies the underlying template library. GSTVIS is (or will be) another component which is very frequently used. For those visualisation and postprocessing purposes for which GSTVIS is not enough, simple commandline utilities for interpolating values from the stored GST grid files will also be provided.

4 Conclusions

GUST will provide (within its scope which will include at least the most common types of plasma physics simulations) also for non-programmer scientists a fast and reliable way of defining and running simulations under different dimensionalities and geometries. The generated simulations use state of the art numerical techniques and will run in parallel if wanted so that they scale from desktop to supercomputers with no changes at the user level. Their performance can compete with hand-written simulations (and much exceed it, in case the hand-written simulation does not use adaptive gridding and parallelisation). We emphasise that GUST is work in progress and that here we only gave some rather rough ideas about the ways the work has been proceeding up to now.

References

1. Janhunen, P., GUMICS-3 – a global ionosphere-magnetosphere coupling simulation with high ionospheric resolution, ESA Symposium Proceedings on 'Environmental Modelling for Space-based applications', ESTEC, Noordwijk, NL, 18–20 Sep 1996, ESA SP–392, 1996 (our first MHD publication).
2. Laitinen, T.V., T.I. Pulkkinen, M. Palmroth, P. Janhunen, and H.E.J. Koskinen, The magnetotail reocnnection region in a global MHD simulation, Ann. Geophysicae, 23, 3753–3764, 2005 (one of our latest MHD publications).
3. For example, Janhunen, P., A. Olsson, A. Vaivads and W.K. Peterson, Generation of Bernstein waves by ion shell distributions in the auroral region, Ann. Geophysicae, 21, 881–891, 2003.
4. For example, Janhunen, P. and E. Kallio, Surface conductivity of Mercury provides current closure and may affect magnetospheric symmetry, Ann. Geophys., 22, 1829–1837, 2004.
5. The Message Passing Interface (MPI), `http://www-unix.mcs.anl.gov/mpi/`
6. The ParMETIS C-library, `http://www-users.cs.umn.edu/~karypis/metis/parmetis/`
7. The Tcl/Tk language, `http://wiki.tcl.tk/`
8. The OpenGL graphics library standard, `http://www.opengl.org`
9. The Mesa implementation of OpenGL, `http://www.mesa3d.org`
10. The Togl Tcl/Tk OpenGL widget, `http://togl.sourceforge.net`
11. The NASA Common Data Format (CDF) library, `http://cdf.gsfc.nasa.gov/`
12. The NCSA Hierarchical Data Format (HDF) library, `http://hdf.ncsa.uiuc.edu/`
13. The Visualization Toolkit, `http://public.kitware.com/VTK/`
14. The QT library, `http://www.trolltech.com/`