No. 183
November 2010

Bits & Bytes

Garching Computing Center of the Max Planck Society and the Institute for Plasma Physics*
Boltzmannstraße 2, D-85748 Garching bei München

New Linux cluster for remote visualization

Klaus Reuter

Introduction

A Linux cluster with powerful graphics hardware was installed in the computing center in order to provide interactive remote visualization services to scientists of the Max Planck society. The production phase started in the middle of October, 2010.

Scientific visualization has significantly gained in importance over the last couple of years. This is partly due to the strong increase in the size and complexity of typical data sets produced by massively parallel simulation codes. Visual impression is an efficient and intuitive way to extract information from such huge datasets. In addition, visualization tools usually support quantitative analysis of multidimensional data, e.g. contour plots on slices through the data set. Moreover, complementing the scientific content of a publication or talk with illustrative images and animations often helps a scientist to communicate results.

Cluster specification

The new cluster (delivered by Hewlett Packard) comprises 6 visualization nodes with 2 NVidia FX5800 GPUs each. 5 of the nodes are equipped with 2 Intel W5580 quad core CPUs and 144 GB RAM. For especially demanding tasks, a large node with 4 Intel X7542 hexa core CPUs and 256 GB RAM is available. A machine with two Intel E5540 quad core CPUs and 144 GB RAM serves as login node. The nodes are connected with a fast InfiniBand network. A 30 TB scratch file system is dedicated to the cluster. In addition, the scratch file system (/ptmp) of the Power6 supercomputer and the DEISA file system are accessible from the visualization nodes. Thereby, HPC users can investigate their simulation results directly.

To allow for interactive use of the visualization cluster which is a requirement quite different from non-interactive simulation runs handled by batch systems, a special reservation and allocation system (VSRT) has been developed. Users can access VSRT via a web browser, reserve resources on a calendar in advance, or start a visualization session immediately. A single GPU, a single node, or several nodes can be reserved and used from within a session. At maximum, 12 visualization sessions are supported on the cluster in parallel.

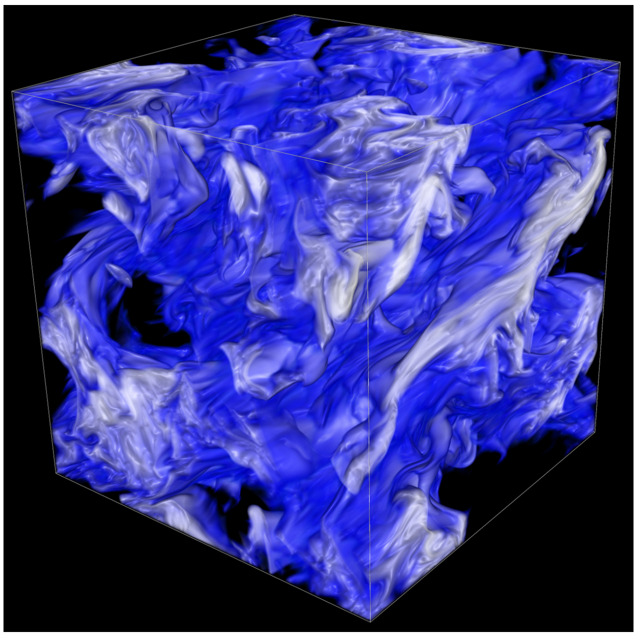

Figure 1: Temperature fluctuations in turbulent MHD convection. Parallel ray casting on 4 nodes (32 CPUs) using VisIt. Simulation: J. Pratt, W.-C. Müller (Max-Planck-Institute for Plasma Physics), Visualization: RZG

Technical details

Technically, remote visualization relies on server-side rendering and efficient transport of the generated images to the client (which is, in general, an arbitrary workstation connected to the Internet). This method has the advantage that the client does not require special graphics hardware and that raw data does not need to be copied to the client. RZG has decided for the open-source solution VirtualGL to transport image data. The generally recommended mode of operation is to start a TurboVNC session via VSRT which launches a preconfigured remote desktop on a visualization node. Detailed information on how to

*Tel.: +49(89) 3299-01, e-mail: benutzerberatung@rzg.mpg.de, URL: http://www.rzg.mpg.de/

Editor: T.Dannert & M.Rampp, Rechenzentrum Garching am IPP, Boltzmannstr. 2, 85748 Garching

connect is given by VSRT. The user can connect to the remote desktop using an arbitrary VNC client, however, the TurboVNC client is recommended for performance reasons. It is freely available for all major operating systems. Users of Unix-like operating systems locally connected at Garching may alternatively use VirtualGL image transport in combination with X forwarding (restriction to Garching due to firewall settings). Thereby, a visualization application running on the remote server seamlessly integrates into the user's desktop environment, without having to use a remote desktop environment. Installation packages for VirtualGL and TurboVNC can be obtained from the download section in VSRT.

Visualization software and support

RZG provides a wide selection of software on the visualization cluster, only few of which can be addressed in the following. The versatile general purpose tools VisIt and ParaView should satisfy most requirements. These applications provide a plethora of methods for data analysis (e.g. cuts, volume rendering, iso surfaces, ...), are designed for massive parallelism (MPI), and support non-interactive rendering via Python APIs. To interactively visualize data on Cartesian or spherical grids, the VAPOR visualization platform is especially useful. A wavelet-based multi-level data compression allows for excellent interactivity. Apart from these tools, special purpose software for applications from astrophysics, chemistry, and biology is provided. Visit the RZG web pages to get a comprehensive list of the available visualization software. Applications can either be started via icons on the VNC desktop or via wrapper scripts provided by RZG, e.g. 'rzg-visit' to start VisIt. Visualization support is provided by the application support group at RZG. A selection of on-going and completed projects is presented on the RZG visualization webpage.

How to access the cluster

Detailed information on how to access and use the cluster is available at `http://www.rzg.mpg.de/visualization`. Send email to visualization@rzg.mpg.de to get an account on the visualization cluster.

Code validation tools

Andreas Marek

Given the ever-growing complexity of scientific codes and also the target platforms (hardware and software environments, runtimes etc.), automated analysis and validation of the source code and its runtime behaviour has become an indispensable step in the process of validating an application and its results. The following article introduces some basic tools that assist the programmer in checking and debugging it's code and it focuses on (parallel) codes written in FORTRAN or C/C++.

The compiler

Careful interpretation of compiler output should always be a precursor to employing more sophisticated code validation tools. Experience shows that compiler warnings and messages quite often indicate problems and thus should not lightheadedly be ignored. Of course, not every warning indicates a real problem, but it is good practice to verify this by examining each warning individually and to decide from case to case whether the compiler indicates a real problem or not. It is also recommended to compile a code with different compilers (if available) and with appropriate debug options ('array bounds' checking, 'division by zero signals', only to name a few) and to investigate the warnings and error messages produced: it is well known that some compilers are more strict than others and a compiler comparison can already make some coding errors visible . Validation with the help of the compiler can be complemented with employing one or more of the following tools.

FORCHECK: A Fortran Verifier and Programming Aid

On Linux systems RZG provides a static syntax checker and code analyzer for Fortran programs. Forcheck is a commercial tool that can analyze individual program units and/or the entire program for conformance to some Fortran standard (FORTRAN 77, 90, and 95), for the use of undeclared or uninitialized variables, obsolete language features and many more.
On Linux systems Forcheck can be enabled via a module

```
module load forcheck
```

and be invoked with the command

```
forchk [options] source-files.
```

One might start with the following set of options

```
forchk -decl -f90 -ff -ancmpl -anprg -anref
       -shcom -shinc -l forcheck.out *.f90
```

which will analyse all source files (*.f90, supposed to be in free format) and write a report to a file (here: forcheck.out). Specifically, in this example the following actions resp. checks are performed:

```
- Warnings for variables that have not explicitly declared in a type statement (-decl)
- Validate the syntax for conformance to FORTRAN 90 standard (-f90)
- Flag unreferenced objects, such as unreferenced common blocks, procedures,
  modules etc. (-ancmpl)
- Analyze the reference structure (-anref),  show cross-reference listings (- shcom),
  show include files(-shinc)
```

In the report output file Forcheck will display informative, warning or error messages.

Note that Forcheck provides an extensive set of options to analyze a code, which cannot be explained in detail here. The reader is referred to the manual (see http://www.forcheck.nl/downloads.htm). and is advised to select and apply options and checks which are most appropriate for the particular application.

MARMOT: MPI Correctness Tool

Marmot, developed at the HLRS Stuttgart and the TU Dresden (see http://www.hlrs.de/organization/av/amt/projects/marmot/), is a non-commercial tool which aims at checking the correct MPI usage in a code at runtime. It includes checking of conformance to the MPI standard, the calling of MPI procedures, and monitors usage of MPI resources.

At RZG, Marmot is provided on Linux systems (Intel compiler, Intel-MPI), both for purely MPI and hybrid MPI and OpenMP codes.

It is available with the command

```
module load marmot
```

or

```
module load marmot-mt
```

, respectively.

MARMOT is provided as C++ library, which has to be linked to the code in the correct order with the MPI libraries, a task which is easily accomplished by compiling the code with the available wrapper scripts, e.g. one uses the commands marmotf77 / marmotf90 for Fortran or marmotcc / marmotcxx for c/c++ to compile the code.

In order to check the code one has to set the LD_LIBRARY_PATH enviroment variable

```
export LD_LIBRARY_PATH=$MARMOT_LIBDIR: \
       $LD_LIBRARY_PATH (bash syntax)
```

and simply execute the newly compiled executable with one additional MPI-task – a Marmot "bookkeeper task" – , e.g.

```
mpiexec -n 9 ... instead of mpiexec -n 8 ...
```

for an MPI job which would normally use 8 tasks.

If one cannot spend an additional MPI task for bookkeeping, the test will run as well but execution might be slower than expected. After completion of the run an output file provides information with different severity levels ranging from "note", "information," to "warning" or "error".

In the above example, Marmot will write its analysis in an ASCII format file (default), a behavior that one can change by setting the environment variable MARMOT_LOGFILE_TYPE to another value prior to the running of the job. Possible values are MARMOT_LOGFILE_TYPE=0,1,2 for summary output in ASCII, HTML, or CUBE format (which can be viewed with the GUI provided with SCALASCA, see previous issue (182) of Bites and Bytes), respectively.

VALGRIND: a memory debugging and profiling tool

On Linux systems RZG provides the free GNU-software VALGRIND toolbox (see http://valgrind.org/), which can be used for analyzing and debugging memory usage of a code. Valgrind is best suited for C and C++ codes, but can be used with Fortran codes as well. Valgrind comprises a core utility (called valgrind) and different tools for different purposes:

MEMCHECK: a memory debugger, which checks all read and writes to memory and can detect memory problems like memory leaks, reading and writing to memory after it has been free'd, and many more.

CACHEGRIND or CALLGRIND: cache profilers, which can pinpoint the source of cache misses

MASSIF: a heap profiler, which can give you information about memory usage

HELGRIND: detects synchronisation errors in programs which use the POSIX pthread parallelization method. Note that OpenMP might be supported, depending on the compiler used to build the executable. Some vendors use their own threading primitives, which cannot be analyzed with Helgrind.

Using any of these tools requires preparing the code by compiling with debug symbols (-g option) and it is not recommended to use compiler optimizations higher than -O0 or -O1.

Valgrind is made available by the commands

```
module load valgrind
```

or

```
module load valgrind-mpi
```

for MPI programs, respectively.

In general valgrind is used like

```
valgrind --tool={memcheck,cachegrind, \
                 callgrind,massif,helgrind} \
         [valgrind-options] \
         your-prog [your-prog-options]
```

As an example binary where no errors are found by Valgrind we show the analysis of the standard listing "ls -l" command, which is done by

```
valgrind --tool=memcheck --leak-check=yes ls -l
```

and obtain, see Output 1, shows that no error has been found in the "ls" command.

An example error report (taken from http://www.valgrind.org/docs/manual/manual-core.html#manual-core.example) is shown in Output 2. One should be aware that Valgrind might find errors or warnings in libraries one uses, but which one can not fix (like the GNU C library, MPI library, etc.) and thus one does not want to see these errors in the analysis. This can be achieved by writting a personal suppression file (see the Valgrind manual for more information).

Of course Valgrind provides much more possibilities for memory analysis than what could be covered in this introductory summary. We would like to motivate the reader to experiment with it and get familiar with its debugging options.

```
==27134==
==27134== HEAP SUMMARY:
==27134==     in use at exit: 20,050 bytes in 48 blocks
==27134==   total heap usage: 646 allocs, 598 frees, 98,333 bytes allocated
==27134==
==27134== LEAK SUMMARY:
==27134==    definitely lost: 0 bytes in 0 blocks
==27134==    indirectly lost: 0 bytes in 0 blocks
==27134==      possibly lost: 0 bytes in 0 blocks
==27134==    still reachable: 20,050 bytes in 48 blocks
==27134==         suppressed: 0 bytes in 0 blocks
==27134== Reachable blocks (those to which a pointer was found) are not shown.
==27134== To see them, rerun with: --leak-check=full --show-reachable=yes
==27134==
==27134== For counts of detected and suppressed errors, rerun with: -v
==27134== Use --track-origins=yes to see where uninitialised values come from
==27134== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Output 1: The output of a Valgrind checking the "ls" command.

```
==25832== Valgrind 0.10, a memory error detector for x86 RedHat 7.1.
==25832== Copyright (C) 2000-2001, and GNU GPL'd, by Julian Seward.
==25832== Startup, with flags:
==25832== --suppressions=/home/sewardj/Valgrind/redhat71.supp
==25832== reading syms from /lib/ld-linux.so.2
==25832== reading syms from /lib/libc.so.6
==25832== reading syms from /mnt/pima/jrs/Inst/lib/libgcc_s.so.0
==25832== reading syms from /lib/libm.so.6
==25832== reading syms from /mnt/pima/jrs/Inst/lib/libstdc++.so.3
==25832== reading syms from /home/sewardj/Valgrind/valgrind.so
==25832== reading syms from /proc/self/exe
==25832==
==25832== Invalid read of size 4
==25832==    at 0x8048724: BandMatrix::ReSize(int,int,int) (bogon.cpp:45)
==25832==    by 0x80487AF: main (bogon.cpp:66)
==25832==  Address 0xBFFFF74C is not stack'd, malloc'd or free'd
==25832==
==25832== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==25832== malloc/free: in use at exit: 0 bytes in 0 blocks.
==25832== malloc/free: 0 allocs, 0 frees, 0 bytes allocated.
==25832== For a detailed leak analysis, rerun with: --leak-check=yes
```

Output 2: An example error output of Valgrind. It shows that 4 bytes at address 0x8048724 are illegally read. This happens at line 45 of the source bogon.cpp, which was called from line 66 in bogon.cpp.

Power6 software upgrade

Ingeborg Weidl, Renate Dohmen

On the Power6 cluster 'vip', there was an upgrade of the AIX operating system, of the LoadLeveler batch system, of the parallel environment (poe, MPI) and of the Fortran and C compilers at the beginning of November 2010.

The default compilers on 'vip' are now xlf 13.1.0.3 and C/C++ 11.1.0.0, and OBJECT_MODE=64 is the default. With the latter setting, the -q64, -b64, -X64 switches of the compiler, linker, archiver, respectively, are no more required.

The new version of the LoadLeveler batch system shows improved scalablility and performance during job scheduling and job dispatching.

The new version of poe has a number of new features. Among others subtasking is supported, which allows users to start concurrent parallel MPI jobs on a given number of processors allocated via LoadLeveler.