No. 184
July 2011

Bits & Bytes

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

Next Generation Supercomputer

Hermann Lederer

The current Max Planck supercomputers operated at RZG are the IBM BlueGene/P and Power6 systems which were installed in September 2007, and in May 2008, respectively. In the 2012/2013 time frame these systems shall be replaced by a more powerful next-generation system in the Petaflop range. The plans for such a replacement, supported by 25 Max Planck Institutes, have been approved by the Max Planck advisory council BAR ('Beratender Ausschuss für Rechenanlagen') in March 2011. In April, a European procurement has been started by the administrative headquarters of the Max Planck Society. The procurement is ongoing, results are expected still in 2011.

Software updates on HPC systems

Francois Thomas (IBM), Renate Dohmen, Markus Rampp

Updated LAPACK and SCALAPACK libraries

Earlier this year we have updated our software installations for the Power6 and BlueGene/P platforms to provide the latest versions of the numerical libraries LAPACK (available versions: 3.3, 3.2, 3.1) and SCALAPACK (available versions: 1.8). Both libraries are available in different versions as modules (check the output of command `module avail`). This installation supersedes the existing libraries installed under `/usr/local/lib`, which are outdated. HPC users are encouraged to point their build scripts, makefiles, etc. to the new LAPACK and SCALAPACK installations.

On Power6 and BlueGene/P, IBM's *Engineering and Scientific Subroutine Library* (ESSL) provides the platform-optimized BLAS routines required by LAPACK (and SCALAPACK). In addition, ESSL also contains optimized equivalents for a number of (but not all) LAPACK routines. When linking an application with ESSL *and* LAPACK it is therefore essential to force the linker to pick ESSL symbols *before* their LAPACK equivalents in order to get the high-performance equivalents from ESSL. See the output of the commands `module help lapack` or `module help scalapack` for specific advice on proper link lines to use.

Note that for historic reasons there is a well-known incompatibility between a small subset of LAPACK routines and their ESSL counterparts. If an application uses any of the routines `(S,D,C,Z)GEEV`, `(S,D)SPEV`, `(C,Z)HPEV`, `(S,D)SPSV`, `(C,Z)HPSV`, `(S,D)GEGV`, `(S,D)SYGV`, special care must be taken as ESSL and LAPACK do *not* use the same argument lists for these routines (see the ESSL documentation for details; a forthcoming ESSL release is expected to finally cure this inconsistency and provide full LAPACK compatibility). Since developers often prefer to adhere to the LAPACK quasi-standard rather than adapting their applications to match the ESSL variant of the argument list, RZG packaged a library named 'lapack-compat' which contains only the LAPACK variants of the aforementioned routines. This allows to construct a link line which picks optimized equivalents (excluding the 'incompatible routines') from ESSL and everything else from LAPACK. For details, see the output of the command `module help lapack`.

NAG SMP library for Power6

The RZG traditionally supports different sequential versions of the NAG Fortran77 and Fortran90 library on the Power6 machine. The licensed product comprises a collection of about 1.600 numerical and statistical routines. NAG libraries are generally available for many compilers and for many platforms and operating systems. Since

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

Editor: H.Lederer & M.Rampp, Rechenzentrum Garching am IPP, Boltzmannstr. 2, 85748 Garching

February 2011, also an SMP parallel version of the latest Mark22 is installed on the Power6 under a trial license for one year. This version is especially tuned for multi-core processors and shared-memory systems. For more specific product information and library contents, please visit NAG's website. Users are kindly encouraged to try this new installation; any feedback will be welcome. All versions of the NAG library on the Power6 are provided as modules.

New Scalable Eigenvalue Solver

Hermann Lederer

The computation of selected eigenvalues and eigenvectors of a symmetric (Hermitian) matrix has relevance for various science disciplines. For the calculation of a significant portion of the eigensystem typically direct eigensolvers are used. For large problems, the eigensystem calculations can become the computational bottleneck. The eigenvalue solver for the symmetric eigenproblem, as taken from the ScaLAPACK library - the state-of-the-art for direct solvers for the calculation of a big part or all eigenvectors - shows limited scalability, even for large problem sizes. As a consequence, a project called ELPA was initiated, supported by the German Federal government, to develop and implement an efficient eigenvalue solver for petaflop applications (BMBF Grant 01IH08007 from Dec 2008 to Nov 2011). The ELPA consortium of University of Wuppertal (BUW), Technical University of Munich (TUM), Fritz-Haber-Institute (FHI), MPI for Mathematics in the Natural Sciences (MIS), IBM and RZG has now developed new one-step and two-step procedures for a direct solver, showing scaling behaviours highly superior to that of using ScaLAPACK routines. The new one-step and two-step solvers have been made publicly available with a LGPL license so that the new solvers can be used in own programs. On RZG's Power6 and BlueGene/P the libraries are available as modules. To obtain the source code, please see `http://elpa.rzg.mpg.de/software` and the corresponding WIKI page. The technical details are described in the following paper: T. Auckentaler et al: Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, in press 2011 by Parallel Computing doi:10.1016/j.parco.2011.05.002.

The new solver is already used in production in the simulation package FHI-aims of the Fritz-Haber-Institute of the Max-Planck-Society. In the meantime the new solvers have been tested on the BlueGene/P PetaFlop system JUGENE at FZ Juelich up to the full machine (294.912 cores). The figure shows good scalability up to the full PetaFlop machine of 72 racks with 294.912 cores using a matrix size of 260.000. Findings are described in the corresponding Technical Report of FZ Juelich, April 2011 (see pages 27-30). Additional independent tests have been successfully performed by an US scientist on the Cray XE6 system 'Hopper' at NERSC, Berkeley.

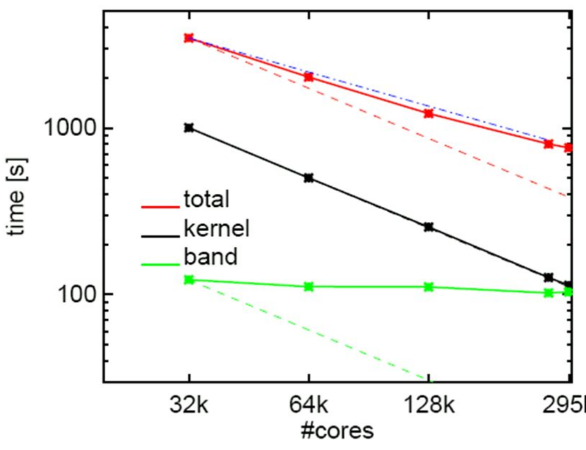

Figure 1: Strong scaling of the eigenvalue solver for a matrix of size 260000 on BlueGene/P at FZJ. Shown are the total solver time (red), the time of the kernel (black), and the time for transformation of a band matrix to tri-diagonal form (green).The dashed lines give the theoretical perfect linear scaling. Note that the kernel part scales perfectly. As a guideline the blue dashed-dotted line shows the limit for a scaling of time proportional to $1/1.6^n$. (Figure by A. Marek, Juelich Scaling Workshop Report, 2011)

HPSS - High Performance Storage System

Andreas Schott

At RZG hierarchical storage systems have a long tradition. Starting with the migrating file system on the Cray YMP in 1991 and later Cray T3E to the IBM Power4 and Power6 HPC systems, and in addition the m-tree in AFS, there has always been a mechanism to provide an automatic migration of large files to and retrieval back from the tape storage. The initial implementation was based on the Cray (later SGI) DMF (Data Migration Facility).

Currently the migration software is using the HSM (Hierarchical Storage Management) extension of IBM's TSM (Tivoli Storage Management).

To cope with the growing demand for data storage, RZG is now switching to HPSS for the hierarchical storage management. As in the past this underlying software is in principle transparent to the end user. Features of this software are extreme scalability and throughput, which leverages the limitations already seen in the current software. HPSS is used in world-wide leading computer centres, where huge amounts of data need to be stored. The new system will provide faster access to tape-migrated data; this refers both to the r-tree of the HPC system and the m-tree of AFS.

The data currently located in either of the file-system trees will be migrated to the new system without any user action required; it will be triggered and supervised by RZG staff. Only the very few users who still save data with the arc command are requested to contact RZG on how to transfer and archive that data.

The new HPSS system is providing a future-proof path to the long-term storage of data, both for the automatically migrating hierarchical file systems, and the long-term data archives in general. The start of operation is planned for October 2011.

Performance Analysis with IBM HPC toolkit

Markus Rampp

The IBM High Performance Computing Toolkit (*HPCT*) is a collection of tools for analyzing the parallel and serial performance of applications with a particular focus on collecting hardware-specific performance counters. The package comprises libraries and executables for instrumenting and profiling applications and for the analysis of collected profiling data. HPCT is available on RZG's HPC systems, Power6 and BlueGene/P as a module. Note that due to different software levels, the HPCT versions and hence some details of usage and functionality of the toolkit differ on Power6 and BlueGene/P (see the RZG webpages for details).

This article presents a brief introduction to HPCT and its basic usage and provides some practical hints for performance analysis of actual applications. We shall focus on compute performance here and will not address HPCT's capabilities for profiling I/O. See the documentation on RZG's webpages for detailed usage instructions (example link lines etc.) and for further documentation and references.

Overview

The IBM High Performance Computing Toolkit comprises the following major components:

- **hpccount**

The hpccount command collects and reports to stdout a number of basic performance characteristics of the application like the execution wall clock time, resource utilization statistics, hardware performance counters information and derived metrics. Similar to the basic unix command `time`, the invocation of an executable is simply prefixed with the command `hpccount` (see Example 1, below).

- **libmpitrace**

The mpitrace library allows to profile MPI function calls in an MPI application written in C or FORTRAN, and to create a trace (timeline) of those MPI calls. The traces can be used to visualize and analyze MPI communication patterns and timelines. In order to use the mpitrace library applications simply need to be relinked. Note that libmpitrace should only be used in single threaded applications or applications in which MPI function calls are made only on a single thread.

- **libhpm**

The hpm library provides routines for manually instrumenting individual sections of a source code written in C or FORTRAN, by inserting special subroutine calls, like, e.g. f_hpmstart() and f_hpmstop() around regions of interest. For instrumenting threaded regions, e.g. code inside an OpenMP-parallel loop, thread-safe variants of these subroutines (e.g. f_hpmtstart(), f_hpmtstop()) must be used.

- **hpctInst**

The hpctInst command allows to instrument subsections (can be subroutines, OpenMP regions, MPI routine calls, or ranges of lines of source code) of an application written in C or FORTRAN in order to gather performance characteristics similar to those reported by hpccount, but on a user-defined, typically more fine-grained level. Instrumentation is performed without recompilation, provided debug symbols were created in the compilation step (-g).

- **peekperf** and **peekview**

Peekperf is the main graphical frontend to HPCT. Peekview is a graphical utility to visualize MPI trace information gathered by libmpitrace.

Comparison with other tools

The distinguishing feature of HPCT is its ability to collect *hardware-specific performance counters*. In that respect, in-depth analysis with HPCT should be considered as a second step, namely trying to understand *why* overall application performance might be limited by certain parts of the code, after having localized *where* this happens by applying simpler tools like tprof, gprof, RZG's perflib, scalasca or alike. The performance tool scalasca (see Bits and Bytes no. 182) in particular allows to conveniently analyze also the communication and scalability properties (MPI routine breakdown, load-balance analysis, ...) in an integrated and graphical way. However, scalasca's capabilities concerning hardware-performance counters (available via the PAPI interface), and in particular derived metrics like Flop rates, are not as comprehensive as those provided by HPCT.

For comparison, RZG's performance library perflib delivers a concise overview of runtimes and Flop rates for all user-instrumented sections of an application. As opposed to HPCT, perflib and scalasca are available also on the linux clusters operated by RZG..

Hints and basic profiling strategies

- different profiling approaches with HPCT should not be mixed, i.e. an executable already instrumented with libhpm should not be instrumented or profiled with a another tool, like, e.g., hpctInst or hpccount.

- while the overhead by running an executable with hpccount is virtually negligible we recommend to always assess potential profiling overhead introduced by libhpm, libmpitrace or hpctInst by comparing runtimes with those of the uninstrumented binary. Profiling overhead may become non-negligible, for example, if huge numbers of MPI or subroutine calls get instrumented in an application.

- due to its negligible overhead hpccount can be used routinely in production runs. This allows to document the performance history of an application and can help identifying unexpected performance variations due to code changes or system updates.

- in order to assess and optimize the performance of an application with HPCT the following steps are recommended to be taken in the given order:

1.) measure overall program performance with hpccount.

Example 1 (see below) shows typical hpccount invocation and sample output. Besides the numbers given in section named '`Resource Usage Statistics`' metrics like '`Utilization rate`', '`Instructions per run cycle`' and '`percent of peak performance`' should be considered in order to judge the overall efficiency of the application. Using the command `hpccount -x` a more verbose output is produced for the last paragraph which explains the definition of derived metrics like 'Utilization rate'.

2.) measure MPI communication characteristics and performance with libmpitrace

The output shown below in Example 2 contains a basic MPI routine breakdown and information of message-size distributions. The mpi_profile for task 0, specifically, in addition shows a communication summary for all tasks and network mapping topology information (BlueGene/P version only). Besides the mpi_profile.nn textfiles also files named mpi_profile.nn.viz (nn denotes the MPI rank) and a file named single_trace are produced. The latter two types of output files can be viewed with the `peekperf` or `peekview` utility, respectively.

Setting TRACE_ALL_TASKS = yes enables tracing of all mpi ranks instead of only the first 256 ranks (which is the default). OUTPUT_ALL_RANKS = yes overrides the default which collects profile information for only 4 MPI ranks: rank 0, and the ranks with the maximum, minimum, and medium MPI times.

3.) identify hot spots at the subroutine level using hpctInst or peekperf

This step can possibly be preceded or replaced by analysis with tprof, gprof, scalasca or alike.

4.) manually instrument individual code sections with libhpm or peekperf

The output from both, hpctInst or libhpm-instrumented applications is much similar to the one produced by hpccount (see below), but with multiple sections corresponding to the instrumented subroutines or regions. Only this last step requires modification of the source code. If performance counters are not of primary interest, perflib instrumentation may be considered as an alternative.

```
~>module load hpct
~>poe hpccount ./a.out
 hpccount v3.2.4 (IHPCT v2.2.0) summary

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode            : 0.456731 seconds
 Total amount of time in system mode          : 0.008140 seconds
 Maximum resident set size                    : 12796 Kbytes

[...]
 #######  End of Resource Statistics  ########

 Execution time (wall clock time)     : 0.467167809605598 seconds
[...]
```

```
  PM_RUN_INST_CMPL (Run instructions completed)                        :       3027444339
  PM_RUN_CYC (Run cycles)                                              :       2178381383

  Utilization rate                                 :         99.127 %
  Instructions per run cycle                       :          1.390
  Total scalar floating point operations           :       2040.000 M
  Scalar flop rate (flops / WCT)                   :       4366.740 Mflop/s
  Scalar flops / user time                         :       4405.180 Mflop/s
  Algebraic floating point operations              :       2040.000 M
  Algebraic flop rate (flops / WCT)                :       4366.740 Mflop/s
  Algebraic flops / user time                      :       4405.180 Mflop/s
  Scalar FMA percentage                            :         99.314 %
  Scalar % of peak performance                     :         23.412 %
```

Example 1: hpccount invocation and sample output

```
~>module load hpct
~>mpxlf90_r my_prog.F -L$IHPCT_HOME/lib -lmpitrace
~>export TRACE_ALL_TASKS = yes
~>export OUTPUT_ALL_RANKS = yes
~>poe ./a.out
~>cat mpi_profile.0
elapsed time from clock-cycles using freq = 850.0 MHz
-----------------------------------------------------------------
MPI Routine                  #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                     9            0.0          0.000
MPI_Comm_rank                    11            0.0          0.000
MPI_Isend                   1292901         3167.7          4.129
MPI_Irecv                   1292901         3309.4          1.382
MPI_Waitall                   40485            0.0         28.012
MPI_Bcast                       848            4.0          0.017
MPI_Barrier                       6            0.0          0.126
MPI_Gatherv                     847         6936.0         14.415
MPI_Scatterv                     22         6936.0          0.381
MPI_Allgatherv                   44        77528.0          1.065
MPI_Allreduce                893206         1530.7         85.489
MPI_Alltoall                     46         2704.7          0.967
-----------------------------------------------------------------
total communication time = 135.984 seconds.
total elapsed time       = 2162.378 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Isend                 #calls    avg. bytes      time(sec)
                           58789          48.0          0.205
                          117578         100.0          0.321
[...]
MPI_Allreduce             #calls    avg. bytes      time(sec)
                           87483           8.0          5.984
                            1950          16.0          0.886
[...]
MPI_Alltoall              #calls    avg. bytes      time(sec)
                              44        2592.0          0.853
                               2        5184.0          0.115

-----------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 135.984 sec for task 0
  median  communication time = 867.370 sec for task 746
  maximum communication time = 896.198 sec for task 47

taskid  xcoord  ycoord  zcoord  procid   total_comm(sec)   avg_hops
     0       0       0       0       0       135.984   41170680.00
     1       1       0       0       0       887.305   41170680.00
     2       2       0       0       0       889.404   41170680.00
[...]
```

Example 2: Linking a FORTRAN program with libmpitrace and example output for MPI task 0