BITS & BYTES BULLETIN

MAX PLANCK
COMPUTING & DATA FACILITY

MPCDF

Bits & Bytes

No.206, April 2021

Computer Bulletin of the Max Planck Computing and Data Facility (MPCDF)*
https://docs.mpcdf.mpg.de/bnb

High-performance Computing

HPC System *Raven*: deployment of the final system

The *Raven*-interim HPC system is currently being dismantled to make way for the final system. The new machine will eventually comprise more than 1400 compute nodes with the brand new Intel Xeon IceLake-SP processor (72 cores per node arranged in 2 "packages"/NUMA domains, and 256 GB RAM per node). A subset of 64 nodes is equipped with 512 GB RAM and 4 nodes with 2 TB RAM. In addition, Raven will provide 192 GPU-accelerated compute nodes, each with 4 Nvidia A100 GPUs (4 x 40 GB HBM2 memory per node and Nvlink 3) connected to the IceLake host CPUs with PCIe Gen 4. All nodes are interconnected with a Mellanox HDR InfiniBand network (100 Gbit/s) using a pruned fat-tree topology with three non-blocking islands (2 CPU-only islands, 1 GPU island). The GPU nodes are interconnected with up to 200 Gbit/s. The first half of the final system will become operational at the beginning of May, the second half by July, 2021. The new IceLake CPU on *Raven* (Intel Xeon Platinum 8360Y[1]) is based on the familiar SkyLake core architecture and the corresponding software stack, which MPCDF users are already familiar with on the interim system as well as on *Cobra* and several clusters. Notably, the IceLake platform provides an increased memory bandwidth (8 memory channels per package, compared to 6 channels on Skylake and its sibling CascadeLake). First benchmarks have shown a STREAM Triad performance of about 320 GB/s for a new *Raven* node (compared to ca. 190 GB/s measured on *Cobra*). The IceLake CPU is based on 10 nm technology and shows a significant energy efficiency increase over the interim CPU. More information about *Raven* can be found on the MPCDF webpage[2] and in the technical documentation[3]. Details about the migration schedule and necessary user actions will be announced to all HPC users of MPCDF facilities in due time. Basically, users of the interim system will only be required to recompile and relink their codes and to adapt their Slurm submission scripts to match the new node-level resources. As the new machine will use the same file systems (/raven/u, /raven/ptmp, /raven/r) as the interim system, migration of user data is not needed.

Hermann Lederer, Markus Rampp

Charliecloud and Singularity containers supported on *Cobra* and *Raven*

The Singularity and Charliecloud container engines have been recently deployed on the HPC clusters of the MPCDF, offering additional opportunities to run scientific applications at our computing centre. Through containers, users have full control of the operating system and on the software stack included in their environment, so that applications, libraries and other dependencies can be packaged and transferred together. Containers also supply an operating system virtualization to run software. This level of isolation, provided via cgroups and namespaces of the Linux kernel, offers a logical mechanism to abstract applications from the environment in

*Editors: Dr. Renate Dohmen & Dr. Markus Rampp, MPCDF
[1]https://ark.intel.com/content/www/us/en/ark/products/212459/intel-xeon-platinum-8360y-processor-54m-cache-2-40-ghz.html
[2]https://www.mpcdf.mpg.de/
[3]https://docs.mpcdf.mpg.de/doc/computing

which they run, promoting software portability between different hosts.

Introducing just a small overhead with respect to bare metal runs, applications in containers have increased reproducibility, running identically regardless of where they are deployed. This makes the use of containers particularly compelling when porting software with complex dependencies or executing applications that require system libraries different from the ones available on the host system (or even a completely different operating system). Containers also provide an easy way to access and run pre-packaged applications that are available online, usually in the form of Docker containers that can be easily converted into a Singularity or Charliecloud container image.

Additional information on the use of Singularity and Charliecloud at MPCDF can be found at the technical documentation page of the MPCDF[4] and in Bits&Bytes No. 205[5].

Michele Compostella

Control and verification of the CPU affinity of processes and threads

Introduction

The correct mapping of processes and threads to processors is of paramount importance to get the best possible performance from the HPC hardware. That mapping is often called pinning and handled via CPU affinities. Likewise, wrong pinning is very often the cause for inferior performance, especially on systems one uses for the first time. In the worst cases of incorrect pinning, some processors would stay idle whereas other processors would be overloaded with more tasks than they are actually able to run simultaneously.

This article gives some information on how to check and control the pinning on MPCDF systems. The `pincheck` library and tool developed at MPCDF is introduced, before the article concludes with some technical background for those readers who are interested.

Checking CPU affinities at runtime

In practice it is unfortunately cumbersome to learn about the actual pinning of a job, as different batch systems, MPI libraries, and OpenMP runtimes offer different ways to turn on verbose output. For example, setting the environment variable `SLURM_CPU_BIND=verbose` will instruct Slurm's `srun` to print the process pinning it performs. Similarly, setting `I_MPI_DEBUG=4` will enable verbose output from the Intel MPI library that includes some pinning information. Third, for example, the variable `KMP_AFFINITY=verbose,compact` will enable pinning output for OpenMP codes compiled with the Intel compilers, but please be aware that `verbose` cannot be specified alone but always needs a type specifier (here `compact`), otherwise no thread pinning would be applied.

As each of these outputs depends on individual software they each need to be read and interpreted differently. To reduce that complexity, MPCDF has developed a simple library and tool that yields the pinning information of codes at runtime in a unified and human-readable fashion.

The `pincheck` library and tool

To give developers and HPC users the possibility to easily check the CPU affinities of the processes and threads of their actual HPC jobs at runtime, MPCDF has developed a lightweight C++ library and tool named `pincheck`[6]. It collects and returns the processor affinities from all MPI ranks in `MPI_COMM_WORLD` and from all the related OpenMP threads. The affinities are obtained in a portable way via system calls from the kernel, and no dependency on specific compilers or runtimes exists. `pincheck` is publicly available under a permissive MIT license.

For C++ codes, there is a header file ('pincheck.hpp') available that can be easily included and used from existing codes. In this case, the C++ header already includes the implementation, and no linking to a library is necessary. For C/C++ and Fortran codes, we will provide a library in combination with a C header file and a Fortran module with the next release in the near future. Alternatively, `pincheck` can be compiled and used as a stand-alone program to check the CPU affinities one gets based on certain batch scripts, environment variables, MPI and OpenMP runtimes, etc.

Detailed information on how to use `pincheck` from an existing code, and on how to compile and run it as a stand-alone program is available in the git repository.

Processor and thread affinities on Slurm-based systems at MPCDF

On the HPC systems and clusters at MPCDF, processes are typically started via the `srun` launcher of Slurm. Based on the resources requested for a batch job, Slurm takes care of the CPU affinities of *processes* (which are typically the MPI tasks) by applying useful defaults (i.e., the `block` distribution method).

For example, for a pure MPI job without threading, `srun` will pin the tasks to individual cores such that consecutive

[4] https://docs.mpcdf.mpg.de/doc/computing/software/containers.html
[5] https://docs.mpcdf.mpg.de/bnb/pdf/bits_and_bytes_issue_205.pdf
[6] https://gitlab.mpcdf.mpg.de/khr/pincheck

tasks share a socket. For hybrid (MPI/OpenMP) jobs that use one MPI task per socket and (per task) a number of threads equal to the number of cores per socket, `srun` will pin each MPI task to an individual socket. The *threads* spawned by these processes inherit the affinity mask, and the user has the option to further restrict the pinning of these individual threads. For OpenMP codes, this can be done by setting the environment variable `OMP_PLACES`, for example to the string `cores` which will pin each OpenMP thread to an individual core. Other threading models (e.g. pthreads) typically offer certain functions to achieve similar functionality.

The MPCDF documentation provides example submit scripts that already include proper settings for the pinning of MPI processes and OpenMP threads, see for example the section on Slurm scripts for the Raven system[7].

Technical background

The compute nodes of today's HPC systems typically contain two or more multi-core chips (sockets) where each chip consists of multiple individual processors (cores) – a design that implies a complex memory hierarchy: each core has its private caches (typically L1 and L2), but shares a last-level cache (typically L3) with a set of other cores that are linked via a fast on-die bus. That bus links to a memory controller to which DIMM modules are connected. Each socket contains one or more such sets of cores that are called NUMA (non-uniform memory access) domains for the following reason: a core may logically access any memory attached to the compute node, however, at different bandwidths and latencies depending on which NUMA domain a particular part of the memory is physically attached to. Different NUMA domains are connected via bus systems that are slower than the intra-domain buses.

On a NUMA system it is therefore desirable that a core accesses physical memory local to its NUMA domain. Memory allocation and use is managed by the Linux operating system in chunks (pages) that are typically 4 kB in size. A first-touch policy applies, and, if possible, memory pages are placed closest to the core on which they were first used. HPC developers must therefore write their threaded programs in a NUMA-aware fashion to optimize for caches and minimize inter-domain memory accesses, and make sure that a process or thread stays within the domain or on the particular core. For codes that implement non-ideal memory access patterns (e.g., thread 0 touches all memory first, and then other threads access that memory across NUMA domains), the automatic NUMA balancing of the Linux OS may improve the performance during the runtime.

By default, the scheduler of the Linux operating system may move processes and threads ("tasks") between the available processors. In case such moves occur within a NUMA domain, a task may suffer a temporary performance penalty when it is moved to a core which initially does not have relevant data cached in L1 or L2. In case a task is moved from one NUMA domain to another, there is in addition a more severe performance penalty caused by non-local memory accesses, i.e., when the moved task accesses memory pages physically located on another NUMA domain from where these pages had been touched first by the same task.

In most HPC scenarios it is advantageous to restrict that moving activity in order to improve the overall temporal and spatial locality of the caches and memory accesses. To enable programmers and users to control the placement of tasks relative to NUMA domains and cores, the operating system supports setting so-called affinity masks which are taken into account by the scheduler. Using such masks, tasks can be "pinned" to sets of cores (e.g. NUMA domains) or even to individual cores or hardware threads, such that they stay there and are not moved. On a low level these masks are actually bit masks, but fortunately users can mostly work on a higher level by using e.g. the variable `OMP_PLACES=cores` to instruct the OpenMP runtime to pin individual threads to individual cores.

References

- The `pincheck` library on MPCDF GitLab[8]
- Slurm `srun` manpage[9]
- Intel MPI Library Developer Guide for Linux OS[10]
- Intel C++ Compiler Classic Developer Guide and Reference, OpenMP Library Support[11]
- OpenMP Reference Guides[12]
- U. Drepper, What Every Programmer Should Know About Memory, 2007[13]
- C. Lameter, NUMA (Non-Uniform Memory Access): An Overview, 2013[14]
- Optimizing Applications for NUMA, Intel Corporation, 2011[15]
- Linux' automatic NUMA balancing[16]

Klaus Reuter

[7] https://docs.mpcdf.mpg.de/doc/computing/raven-user-guide.html#slurm-example-batch-scripts
[8] https://gitlab.mpcdf.mpg.de/khr/pincheck
[9] https://slurm.schedmd.com/srun.html
[10] https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top.html
[11] https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/optimization-and-programming-guide/openmp-support/openmp-library-support.html
[12] https://www.openmp.org/resources/refguides/
[13] https://www.akkadia.org/drepper/cpumemory.pdf
[14] https://doi.org/10.1145/2508834.2513149
[15] https://software.intel.com/content/www/us/en/develop/articles/optimizing-applications-for-numa.html
[16] https://documentation.suse.com/sles/15-SP2/html/SLES-all/cha-tuning-numactl.html

High-performance data analytics and AI software stack at MPCDF

In the last years we have been observing an ever-growing number of researchers who want to use institute clusters and the HPC systems at MPCDF for data analytics and especially for machine-learning and deep-learning projects. This wish stems from the fact that the extremely powerful resources of HPC servers, especially if equipped with high-end GPU devices, can substantially boost the performance of data-analytics and AI workloads. Furthermore, the possibility to use multi-node setups to parallelize the workflows can reduce the time to solution by orders of magnitude. However, for users it is a non-trivial task to obtain a software stack that really does exploit the hardware features of HPC systems (SIMD vectorisation, Tensor cores of the GPUs, high-bandwidth fabrics, to mention a few) and does run with a reasonable fraction of the theoretical performance of the systems. Especially the builds of frameworks which the users can obtain via the usual distribution ways of the ML/DL community, such as Python-based installation methods like "pip", usually do not run efficiently on HPC hardware.

In order to address the needs of its users for such workflows, MPCDF provides an HPC-optimized software stack for data-analytics and AI applications. Among other things, this software stack comprises

- basic ML and AI libraries such as Nvidia's and Intel's DNN implementations
  - Nvidia NCCL and cuDNN
  - Intel MKL-DNN
  - opencv
- popular frameworks
  - Tensorflow
  - Pytorch
  - Mxnet
  - scikit-learn
- parallelization frameworks
  - Horovod (for Tensorflow, Pytorch and MxNet)
  - Apache Spark
- tools for image and NLP processing

See the MPCDF documentation for a detailed list[17] and for some examples[18] of how to use the software together with Slurm.

Whenever possible, a CPU and a GPU variant of the software is provided, which gives the user the freedom of choice and allows a seamless migration between different nodes and even clusters. As usual, the software is provided on MPCDF systems via the module environment. Please note that MPCDF uses a hierachical software stack (see Bits & Bytes No. 198[19]) and not all software is always visible with the "module avail" command. We recommend to use the "find-modules" command, which will help users to find whether a software is available and which modules have to be loaded before the respective module will be visible.

Example:

```
user@cobra01:~> find-module tensorflow/gpu
tensorflow/gpu/1.14.0   (after loading
                           anaconda/3/2019.03)
tensorflow/gpu/2.1.0    (after loading
                           anaconda/3/2019.03)
tensorflow/gpu/2.1.0    (after loading
                           anaconda/3/2019.03)
tensorflow/gpu/2.2.0    (after loading
                           anaconda/3/2019.03)
tensorflow/gpu/2.2.0    (after loading
                           anaconda/3/2019.03)
tensorflow/gpu/2.3.0    (after loading
                           anaconda/3/2019.03)
```

After the desired modules have been loaded, the software can be used in the usual way and for example can be used with Jupyter Notebooks.

Further readings:

- Bits & Bytes No.203[20] for Jupyter Notebooks as a Service
- Bits & Bytes No.200[21] for Data Analytics at MPCDF

Andreas Marek

Decommissioning of AFS

After many years of acting as the central file system in MPCDF, the time has come to say goodbye to the Andrew File System (AFS). This does not mean that AFS will disappear immediately, but as a first step it is planned that home directories will no longer be set up in AFS and not all login nodes of the Linux clusters

[17] https://docs.mpcdf.mpg.de/doc/computing/software/data-analytics/list-of-supported-software.html
[18] https://docs.mpcdf.mpg.de/doc/computing/software/data-analytics/machine-learning-software.html
[19] https://docs.mpcdf.mpg.de/bnb/pdf/bits_and_bytes_issue_198.pdf
[20] https://docs.mpcdf.mpg.de/bnb/pdf/bits_and_bytes_issue_203.pdf
[21] https://docs.mpcdf.mpg.de/bnb/pdf/bits_and_bytes_issue_200.pdf

will provide access to AFS, as it is already the case on *gatezero*. The lack of support for Windows forces the use of alternatives. For most users the Sync&Share functionality provided by our datashare is a good solution. For experiment data and software distribution other ways are already established or still have to be determined. Thus, we kindly ask all our users to no longer consider AFS as the one and only filesystem for data exchange, but to implement alternatives and not to store new data in AFS home directories.

Andreas Schott

Relaunch of MPCDF website and new technical documentation platform

In March 2021, MPCDF relaunched its main website, adopting the corporate design of the Max Planck Society. The technical documentation for users of MPCDF services, including a comprehensive and continuously extended FAQ, as well as the MPCDF computer bulletin Bits&Bytes has been refurbished and is now available at https://docs.mpcdf.mpg.de/

Markus Rampp on behalf of the MPCDF Webteam

Events

New online introductory course for new users of MPCDF

The MPCDF has started offering a new online introductory course targeting new users. The first issue was held on April 13th with over 100 registered users from more than 30 Max Planck Institutes. In the future, it will be repeated on a semi-annual schedule. The 2.5 hour online course is given by application experts of MPCDF and includes an interactive chat option and concluding Q&A sessions. It provides a basic introduction to the essential compute and data services available at MPCDF, and is intended specifically for lowering the bar for their first-time usage. This course is the basis for more advanced courses such as the annual "Advanced HPC workshop" organised by MPCDF (next issue: autumn 2021, see below). Major topics of the online introductory course include an overview and practical hints for connecting to the HPC compute and storage facilities and using them via the Slurm batch system. The course material can be found at the MPCDF webpage.

Advanced HPC workshop: save the date

Our annual Advanced High-performance Computing Workshop is scheduled for November 22nd to 24th, 2021, with an additional day of hands-on sessions for accepted projects on the 25th. The main topics will be software engineering, debugging and profiling for CPU and GPU. The talks will be given by members of the application group and by experts from Intel and Nvidia. Further details and registration options will be announced in the next issue of Bits & Bytes.

Klaus Reuter, Sebastian Ohlmann, Tilman Dannert