Making the Case for Portable MPI Process Pinning

Balazs Gerofi
RIKEN R-CCS
JAPAN
bgerofi@riken.jp

Rolf Riesen
Intel Corporation
USA
rolf.riesen@intel.com

Yutaka Ishikawa
RIKEN R-CCS
JAPAN
yutaka.ishikawa@riken.jp

ABSTRACT

As architectural complexity of node resources in high-performance computing (HPC) keeps increasing, topology aware process placement becomes utmost important for efficiently utilizing the underlying hardware. Although most MPI implementations provide interfaces to control process placement, existing APIs are fully implementation specific and non-standard solutions, leading to non-portable job scripts among different MPI environments. Furthermore, most of the existing APIs provide overly intricate and often redundant process pinning mechanisms.

In this poster we propose `mpipin`, an MPI implementation agnostic process pinning tool that provides a simple, intuitive interface for deterministic resource assignment. We describe `mpipin`'s API, its topology aware design and implementation. Through experiments, we demonstrate its ability to provide identical process pinning behaviour in Intel MPI, MVAPICH and Open MPI environments using the same command line invocation.

ACM Reference format:
Balazs Gerofi, Rolf Riesen, and Yutaka Ishikawa. 2018. Making the Case for Portable MPI Process Pinning. In *Proceedings of EuroMPI '18, Barcelona, Spain, Sep, 2018,* 2 pages.
DOI: http://dx.doi.org/10.1145/3095770.3095777

1 INTRODUCTION AND MOTIVATION

Complexity of node architecture in supercomputing environments has increased significantly during the past decade. With the advent of many-core CPUs and the prevalence of non-uniform memory access (NUMA) architectures the presence of large number of CPU cores with sophisticated hardware topology has become commonplace. For example, Intel's Xeon Phi Knights Landing CPU in SNC-4 memory configuration can provide up to 272 CPU cores organized around 8 NUMA domains [4].

In such environments, MPI jobs are typically executed with multiple ranks inside compute nodes. Indeed, in our previous study we found that most of the CORAL benchmarks [2] perform best when run using 32 or 64 ranks per node on a large-scale, many-core based supercomputer [3]. To efficiently utilize complex hardware topologies, locality information must be carefully considered and thus correctly placing MPI ranks on appropriate CPU cores has become utmost important.

EuroMPI '18, Barcelona, Spain

DOI: http://dx.doi.org/10.1145/3095770.3095777

Most MPI implementations provide interfaces for process pinning. For example, Intel MPI relies on the `I_MPI_PIN` environment variable, together with `I_MPI_PIN_ORDER` and `I_MPI_PIN_DOMAIN`, etc., to control process placement. MVAPICH supports process binding when compiled with `hwloc` [1]. Some of the environment variables to control process placement are: `MV2_ENABLE_AFFINITY` toggles binding, and `MV2_CPU_BINDING_POLICY` can be used together with `MV2_THREADS_PER_PROCESS` to control pinning.

Open MPI, on the other hand, expects various command line arguments to `mpirun` (e.g., the `-bind-to core`, or `-bind-to socket`, etc.) so that processes are pinned to specific CPU cores. It also provides a long list of other options to enable fine grained control over how exactly processes are mapped to CPUs.

Recognizing the chaotic landscape of available process placement mechanisms among MPI implementations, certain batch job systems provide their own native interfaces. For example the `srun` command of the SLURM cluster resource manager [5] provides the `--cpu-bind` argument with a wide variety of options to control process binding. Needless to say, SLURM's options are also different than existing MPI based solutions.

In summary, there are multiple issues with the existing pinning APIs and the lack of a unified interface thereof:

- Jobs scripts that rely on MPI implementation specific pinning options are not portable among platforms using different MPI implementations.
- Comparing performance between different MPI implementations is difficult, because one needs to ensure that process pinning is performed exactly the same way in different environments.
- Some of the existing APIs provide non-deterministic process placement in subsequent executions with the same pinning option which can lead to reproducibility problems.

To overcome these issues, we propose `mpipin`, an MPI implementation agnostic process pinning tool that provides a simple, intuitive interface for resource assignment. Description of design and implementation, together with a simple usage example compared to Intel MPI, MVAPICH and OpenMPI are provided in the rest of this document.

2 DESIGN AND IMPLEMENTATION

`mpipin` is a simple process pinning tool that provides an intuitive interface and is completely independent from MPI implementations. `mpipin` merely relys on the fact that MPI ranks inside a node are spawned by a common parent process (usually by the MPI proxy process). A general usage example of `mpipin` is shown in the following invocation:

```
mpirun -hostfile ~/hosts -n <N> -ppn <PPN> \
mpipin --ranks-per-node <PPN> app
```

As seen, `mpipin` is invoked before the application binary and information on the desired process pinning policy is passed as command line arguments to the tool itself. Internally, `mpipin` processes running on the same node synchronize at job startup time, elect a leader process that creates a shared memory region which is then mapped by all `mpipin` processes. The leader process collects topology information and determines where each rank will have to be placed. Processes are ordered by creation time and process ID (i.e., the OS `pid`), which ensures that subsequent executions of the same invocation places the same local rank to the same set of CPUs. `mpipin` currently obtains topology information directly by parsing the Linux `sysfs` file system where topology information related to NUMA nodes, CPUs and caches is exposed. However, future usage of an `hwloc` [1] based backend is being considered.

Once resources are partitioned by the leader, all processes are woken up and each rank sets its processor affinity (simply by calling the `sched_setaffinity()` system call) to its corresponding CPU partition. From an MPI implementation's point of view `mpipin` is simply the application to be executed and thus it is easy to see how it remains MPI implementation agnostic. The tool is still in its early development phase and currently supports the following options:

- **–processes-per-node, –ranks-per-node, –ppn**: Specifies the number of MPI processes per node.
- **–threads-per-process, –cores-per-process, –tpp**: Specifies the number of threads (i.e., logical CPUs) per MPI process.
- **–compact**: Follow a compact process layout (default).
- **–scatter**: Follow a scattered process layout.
- **–exclude-cpus, –exclude-cores**: Specifies a list of logical CPUs to be excluded from resource partitioning.

We have determined these options based on our experience with running a substantial number of mini- and actual applications during a large scale evaluation [3]. The current default pinning policy of `mpipin` is *compact* process layout, where each rank is assigned a group of logical CPUs with the most local resources shared (i.e., CPU cores that share resources in the order of caches starting from L1 going higher, via sockets, and finally located in the same NUMA node). The tool also supports *scatter* layout as well as explicitly leaving out CPU cores that may be allocated for other purposes (e.g., for OS activities).

3 DEMONSTRATION

We provide a comparative demonstration of process placement using Intel MPI, MVAPICH and Open MPI. Consider the following scenario where compute nodes consist of a two socket Intel Xeon system with 14 CPU cores on each socket and 2 HW threads per CPU core (i.e., an E5-2690 v4 system). When running a hybrid MPI + OpenMP program one may intend to run four ranks per node and thus assign 7 CPU cores (i.e., 14 HW threads to each rank) laying out processes next to each other. Using Intel MPI one would invoke the following command:

```
mpirun -env I_MPI_PIN_DOMAIN=14 \
       -env I_MPI_PIN_ORDER=compact \
       -n 4 -ppn 4 -host <host> app
```

To achieve the same mapping in MVAPICH, the following command line is required:

```
mpirun -env MV2_ENABLE_AFFINITY=1 \
       -env MV2_CPU_BINDING_POLICY=hybrid \
       -env MV2_THREADS_PER_PROCESS=14 \
       -env MV2_HYBRID_BINDING_POLICY=linear\
       -n 4 -ppn 4 -host <host> app
```

Finally, in Open MPI one would need to use:

```
mpirun -map-by ppr:2:socket:pe=7 \
       -np 4 -host <host>:4 app
```

Using `mpipin`, all the above examples could be reduced to the same invocation as listed in Section 2 by simply replacing N and PPN by 4. We intend to provide more representative examples in the final poster.

4 RELATED WORK

`hwloc` [1] provides a library for exposing hardware locality information in a portable manner. It also provides command line tools for topology information processing and process pinning. Both Open MPI and MVAPICH can utilize `hwloc` as their process binding backend. `hwloc` command line tools, however, can not be used directly for MPI process pinning because each process is provided its own private view of the topology.

One of the most similar frameworks to our proposal is LIKWID [6]. In particular, `likwid-mpirun` provides a dedicated pinning API that is translated by wrapper scripts to MPI implementation specific pinning options. The drawback of this approach is that unless LIKWID provides support to the underlying MPI implementation, the tool is not applicable. On the contrary, `mpipin` is executed by the MPI job itself and thus it remains transparent to the MPI implementation.

ACKNOWLEDGMENT

This work has been partially funded by MEXT's program for the Development and Improvement for the Next Generation Ultra High-Speed Computer System, under its Subsidies for Operating the Specific Advanced Large Research Facilities.

REFERENCES

[1] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. 2010. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, IEEE (Ed.). Pisa, Italy. DOI: https://doi.org/10.1109/PDP.2010.67

[2] CORAL. 2013. Benchmark Codes. https://asc.llnl.gov/CORAL-benchmarks/. (Nov. 2013).

[3] Balazs Gerofi, Rolf Riesen, Masamichi Takagi, Taisuke Boku, Yutaka Ishikawa, and Robert W. Wisniewski. 2018. Performance and Scalability of Lightweight Multi-Kernel based Operating Systems. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

[4] James Jeffers, James Reinders, and Avinash Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition 2Nd Edition* (2nd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[5] Morris A. Jette, Andy B. Yoo, and Mark Grondona. 2002. SLURM: Simple Linux Utility for Resource Management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 44–60.

[6] Jan Treibig, Georg Hager, and Gerhard Wellein. 2010. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops (ICPPW '10)*. IEEE Computer Society, Washington, DC, USA, 207–216. DOI: https://doi.org/10.1109/ICPPW.2010.38