Improving Support of MPI+OpenMP Applications

Geoffroy R. Vallée
Oak Ridge National Laboratory
Oak Ridge, Tennessee
valleegr@ornl.gov

David Bernholdt
Oak Ridge National Laboratory
Oak Ridge, Tennessee
bernholdtde@ornl.gov

ABSTRACT

The execution of hybrid applications, i.e., a combination of MPI for inter-node parallelism and a threading solution for on-node parallel, is perceived as a key option to achieve exascale. Unfortunately, such options have been historically developed by separate communities, resulting in challenges to deploy complex scientific hybrid applications on large scale systems. This work proposes a design for capabilities that would enable the precise definition and deployment of application layouts on compute node (i.e., placement of MPI ranks and threads). Our contributions are: (i) a new notation scheme that can be used to define complex "layouts"; (ii) a new runtime library for the coordination of the MPI and OpenMP runtimes; (iii) a set of new components for the MPI runtime in order to support our concept of layouts.

ACM Reference Format:
Geoffroy R. Vallée and David Bernholdt. 1997. Improving Support of MPI+OpenMP Applications. In *Proceedings of The EuroMPI 2018 Conference (EuroMPI 2018).* ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

MPI [5] is already widely used together with other runtime environments (the so-called MPI+X approach). Modern MPI implementations typically make internal use of threads to facilitate the progress of communications while computation proceeds on other threads, and the MPI standard includes features to manage some of the interactions between user threads and MPI's use of threads. Much more is needed but even within the current standard and implementation, there is room for significant improvements. We are investigating interoperability with user-facing thread-based runtime environments, OpenMP being the canonical example. We intend to develop and implement abstractions and APIs that allow better coordination between runtime environments that rely on threads. We are interested in the potential to coordinate threads and process placement between MPI and other runtimes, like OpenMP. Currently, these are very loosely coupled, in the interest of maintaining independence of the software stacks. However, as node architectures become more complex, it becomes harder to achieve the best placement, and the negative consequences of poor placement also increase [4, 7–10]. We propose abstractions and interfaces that allow runtimes to share information and coordinate process/thread placement.

2 RELATED WORK

In the context of this work, research on the optimization of hybrid applications can be categorized as follow. The first approach is to *improve existing standards*. The MPI forum currently has a hardware topology working group [6], which investigates what modifications to the standard could be done to best expose the underlying hardware to both MPI and hybrid applications: (i) by exposing the underlying hardware through hierarchical communicators, and (ii) through the standardization of mpirun/mpiexec to support portable mapping and binding directives. While these strandization efforts are important, we believe that the placement of MPI ranks and threads is not suitable for standardization since mainly a platform specific problem for which the resource managers and the runtimes needs to be involved.A second approach is to *modify the implementation of standards*. To the best of our knowledge, most research is focusing on optimizing runtime features to improve the execution of hybrid applications or to improve threading support within the context of MPI [1–3], and not on the coordination of existing runtimes. The final option is to investigate *resource managers and system tools* to facilitate and optimize the execution of hybrid applications, through the extensions of the resource/job managers. Home-made system tools have been developed over the years at leadership computing centers to help users express their requirements and translate their requirements to directives that the resource manager will understand. This type of tools support only fairly simple layouts (e.g., identical layouts for all MPI ranks) because of the limited control of a user-level tool over runtimes.

3 ARCHITECTURE

The placement of MPI ranks and threads on processor resources and/or on accelerators is a problem that is mainly application-specific, meaning that there is no one-fit-all solution. Figure 1 gives a schematic view of a layout. Because we need to support incompatible layouts, we decided to support the definition of layouts upon application execution. The first requirement is to let the user describe these layouts. We opted to define an explicit layout description method where the user precisely defines where ranks and threads are deployed. We propose a new notation to describe both

EuroMPI 2018, September 2018, Barcelona, Spain

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

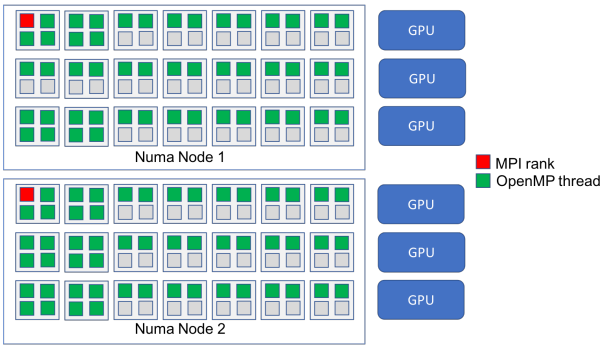

Figure 1: Example of a complex layout targeting the OLCF Summit system (the smaller squares represent hardware threads; groups of four small squares represent a core; groups of cores represent a NUMA node; 2 NUMA nodes represent an entire compute node; each NUMA node is linked to 3 GPUs)

the placement of MPI ranks and threads: *[runtime ID],[Parent context][target HW level][Map]*, where the *runtime ID* is a unique string representing a target runtime (MPI or OpenMP); *Parent context* is used to define dependencies between each entry. For example, the first entry for OpenMP defines MPI-0 as the *Parent context*, which means that the master thread is the first MPI rank on the node. The *target HW level* represents the target hardware components (e.g., Core). Finally, the *Map* describes where the rank/thread needs to be deployed and in which order. Note that the hierarchy of the underlying hardware needs to be reflected to avoid any possible confusion. While our layout description can seem cumbersome, we assume that this description is applicable to all compute node of the job, and that it is possible to develop tools to assist the users.

All Open MPI mappers are required to have two functions: one that maps the ranks on nodes that is performed before ranks are actually deployed, and one that maps the ranks on resources upon deployment on the node. Our new mapper, *explicit*, follows the same interfaces and explicitly maps all ranks based on the layout defined by the user. Once the layout is defined and submitted, via an argument to the mpirun command, the MPI implementation deploys first the MPI ranks, while in the context of OpenMP, the definition of places and policies applied during the creation of threads via environment variables can be used to precisely place threads. Practically, we provide a new Open MPI mapper that registers an MCA parameter for the definition of layouts. Our mapper parses the MCA parameter defining the layout and places all MPI ranks accordingly. In addition, it publishes the layout using PMIx to make it available to other libraries.

Once the MPI ranks are deployed on the nodes, it is necessary to constraint, for each MPI rank, where threads can be created. To avoid modifying the OpenMP runtime, we opted for the implementation of a helper library that retrieves the layout through PMIx and sets the OMP_PLACES environment variable. This library, named the MPI OpenMP Coordination library (MOC) also ensures that the environment variable is set based on the current MPI rank, allowing for rank specific layouts. To do so, MOC calculates and publishes through PMIx the actual rank but also the rank number on the node. Since MPI is the first runtime to act on a compute node, it virtually shares all published data with any PMIx compliant runtime or library. Practically, as PMIx can be viewed as a key/value store, we reserve a key name (MOC_LAYOUT) that is unique and available only in the scope of the job. This approach allows us to have a solution that let users specify a layout at the job level, facilitating the investigation of the performance impact of various layouts and for various computing platforms. Note that the MOC library needs to be performed before the initialization of the OpenMP runtime and therefore requires the application to call MOC_init() right after MPI_Init(). The MOC library also performs a few MPI calls to figure out the rank of the current process and the order of the ranks on the node (so we know that we are the nth rank on the node).

4 CONCLUSION

We presented the concept of *layouts* that let scientists precisely define the placement of MPI ranks and OpenMP threads, implemented via a new mapper for Open MPI and a new runtime helper library. We also propose to investigate the concept of *dynamic layouts*, i.e., layouts that can be changed during the execution of applications, which is especially of interest to applications that are composed of phases that potentially require different layouts. The support of such layouts will require more extensive runtime-level modifications (e.g., OpenMP runtimes are very static in nature).

REFERENCES

[1] Abdelhalim Amer, Huiwei Lu, Yanjie Wei, Pavan Balaji, and Satoshi Matsuoka. 2015. MPI+Threads: Runtime Contention and Remedies. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2015)*. ACM, New York, NY, USA, 239–248. https://doi.org/10.1145/2688500.2688522

[2] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. 2010. Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming. *Int. J. High Perform. Comput. Appl.* 24, 1 (Feb. 2010), 49–57. https://doi.org/10.1177/1094342009360206

[3] Alex Brooks, Hoang-Vu Dang, Nikoli Dryden, and Marc Snir. 2015. PPL: An Abstract Runtime System for Hybrid Parallel Programming. In *Proceedings of the First International Workshop on Extreme Scale Programming Models and Middleware (ESPM '15)*. ACM, New York, NY, USA, 2–9. https://doi.org/10.1145/2832241.2832246

[4] François Broquedis, Nathalie Furmento, Brice Goglin, Raymond Namyst, and Pierre-André Wacrenier. 2009. Dynamic Task and Data Placement over NUMA Architectures: An OpenMP Runtime Perspective. In *Proceedings of the 5th International Workshop on OpenMP: Evolving OpenMP in an Age of Extreme Parallelism (IWOMP '09)*. Springer-Verlag, Berlin, Heidelberg, 79–92. https://doi.org/10.1007/978-3-642-02303-3_7

[5] The Message Passing Forum. 1994. "MPI: a Message-Passing Interface Standard".

[6] MPI Forum Hardware Topology Working Group. [n. d.]. https://github.com/mpiwg-hw-topology.

[7] Emmanuel Jeannot and Guillaume Mercier. 2010. Near-optimal Placement of MPI Processes on Hierarchical NUMA Architectures. In *Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part II (Euro-Par'10)*. Springer-Verlag, Berlin, Heidelberg, 199–210. http://dl.acm.org/citation.cfm?id=1885276.1885299

[8] Haoqiang Jin, Dennis Jespersen, Piyush Mehrotra, Rupak Biswas, Lei Huang, and Barbara Chapman. 2011. High Performance Computing Using MPI and OpenMP on Multi-core Parallel Systems. *Parallel Comput.* 37, 9 (Sept. 2011), 562–575. https://doi.org/10.1016/j.parco.2011.02.002

[9] A. Mazouz, S. A. A. Touati, and D. Barthou. 2011. Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of SPEC OMP applications on intel architectures. In *2011 International Conference on High Performance Computing Simulation*. 273–279. https://doi.org/10.1109/HPCSim.2011.5999834

[10] Vishwanath Venkatesan, Rakhi Anand, Jaspal Subhlok, and Edgar Gabriel. 2013. Optimized Process Placement for Collective I/O Operations. In *Proceedings of the 20th European MPI Users' Group Meeting (EuroMPI '13)*. ACM, New York, NY, USA, 31–36. https://doi.org/10.1145/2488551.2488567