Performance Evaluation of NICAM-DC-MINI using XcalableACC on Accelerated Cluster

Masahiro Nakao
RIKEN Advanced Institute
for Computational Science
Hyogo, Japan
masahiro.nakao@riken.jp

Hitoshi Murai
RIKEN Advanced Institute
for Computational Science
Hyogo, Japan

Akihiro Tabuchi
Graduate School of Systems and
Information Engineering
University of Tsukuba
Ibaraki, Japan

Taisuke Boku
Center for Computational Sciences
University of Tsukuba
Ibaraki, Japan

Mitsuhisa Sato
RIKEN Advanced Institute
for Computational Science
Hyogo, Japan

ABSTRACT

Cluster systems equipped with accelerators, commonly known as “accelerated clusters”, have entered widespread use in various fields. In order to develop applications in accelerated clusters, programmers often use a combination of CUDA and MPI, or a combination of OpenACC and MPI (OpenACC + MPI). However, these combinations face programming complexity issues due to their inclusion of MPI functions. This paper introduces the XcalableACC (XACC) language, which is a combination of OpenACC and XcalableMP (XMP). XMP is a Partitioned Global Address Space (PGAS) language for distributed memory systems, and XMP enables programmers to develop applications more easily than MPI. To evaluate the performance and productivity of XACC, we implemented the NICAM-DC-MINI application using XACC. The results show that while the performance of XACC was slightly inferior to that of OpenACC+MPI, the implementation of XACC has a much better outlook.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference Format:
Masahiro Nakao, Hitoshi Murai, Akihiro Tabuchi, Taisuke Boku, and Mitsuhisa Sato. 2018. Performance Evaluation of NICAM-DC-MINI using XcalableACC on Accelerated Cluster. In *Proceedings of HPC Asia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BACKGROUND

Cluster systems equipped with accelerators, commonly known as “accelerated clusters”, are widely used in various fields. To develop applications in accelerated clusters, a combination of CUDA and MPI is often used due to their ability to exploit the system performance. In addition, a combination of OpenACC and MPI (OpenACC+MPI) has emerged [13] because OpenACC can be used to develop applications using accelerators at reduced programming costs and has higher code portability than CUDA. However, even with these combinations, the low productivity of MPI cannot be overcome.

Partitioned Global Address Space (PGAS) languages, which offer higher productivity than MPI, have been proposed for distributed memory systems. Thus, a combination of OpenACC and a PGAS language is proving to be useful in accelerated clusters. Examples of PGAS languages include XcalableMP (XMP) [7, 12], Coarray Fortran (CAF) [17], PCJ [16], Unified Parallel C (UPC) [1], UPC++ [18], HabaneroUPC++ [10], X10 [4], Chapel [3], and DASH [5].

Previously, we designed the XcalableACC (XACC) [6, 11] language as an XMP extension that uses OpenACC for accelerated clusters. XMP is a PGAS language that provides two parallelization features. One is a typical parallelization using directives, the other is a flexible parallelization using coarray features. Since programmers can use both XMP features and OpenACC directives in XACC, they can develop applications on accelerated clusters with ease. XACC consists of extensions of C and Fortran, and the Fortran version of XACC is backward compatible with Fortran 2008.

In this paper, we report on the development of the NICAM-DC-MINI application [14] using the Fortran version of XACC, along with our evaluation of the performance and productivity of XACC on our accelerated cluster. NICAM-DC-MINI, which is a subset of the NICAM-DC application[15], contains the minimum computational procedures to run a baroclinic wave test case[9].

2 OMNI COMPILER

Omni compiler is a reference implementation for an XACC language that has been developed as an open-source project by the University of Tsukuba and the RIKEN on GitHub[1].

Figure 1 shows that the compile process used in Omni compiler. First, the compiler translates an original file that has been prepared by a programmer into an intermediate file. The XMP and XACC directives in the original file, along with their coarray features, are

[1]https://github.com/omni-compiler

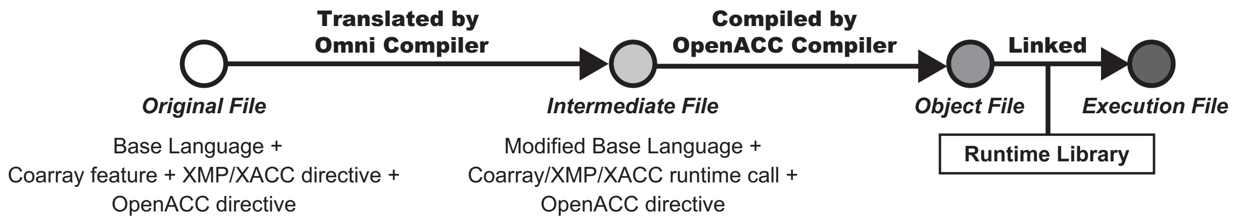
Translated by Omni Compiler
Compiled by OpenACC Compiler
Linked
Original File
Intermediate File
Object File
Execution File
Runtime Library
Base Language + Coarray feature + XMP/XACC directive + OpenACC directive
Modified Base Language + Coarray/XMP/XACC runtime call + OpenACC directive

Figure 1: Compile process in Omni compiler

Table 1: Evaluation environment

CPU	Intel Xeon-E5 2680v2 2.8 GHz x 2 Sockets
Memory	DDR3 1866MHz 59.7GB/s 128GB
GPU	NVIDIA Tesla K20X (GDDR5 250GB/s 6GB) x 4
Network	InfiniBand FDR 7GB/s
Software	Omni compiler 1.2.1, PGI compiler 17.3, CUDA 8.0.44, MVAPICH2 2.2

translated into runtime calls in the intermediate file. If necessary, the code except for the original file XMP and XACC directives, as well as their coarray features, are also modified. At this point, the intermediate file still has general OpenACC directives. Next, an OpenACC compiler compiles the intermediate file and creates an execution file with a link to the Omni compiler's runtime library. This allows Omni compiler to use any OpenACC compiler as a backend compiler.

3 IMPLEMENTATION OF NICAM-DC-MINI

We implemented NICAM-DC-MINI using coarray features and OpenACC directives in XACC based on the OpenACC+MPI version of NICAM-DC-MINI. To accomplish a coarray-based implementation, the MPI functions in the based code must be replaced with their corresponding coarray notations. We changed the code using the following rules[8] basically.

- MPI_Send/Isend $\longrightarrow$ coarray assignment.
- MPI_Recv/Irecv $\longrightarrow$ (be deleted).
- MPI collective communication $\longrightarrow$ intrinsic subroutine (e.g. co_broadcast).
- MPI_Wait and MPI_Barrier $\longrightarrow$ sync all statement.

Figure 2 shows a code modification portion performed using XACC. Note that an additional **sync all** statement is required before coarray operation in line 1 of Fig. 2b in order to ensure that the array *recvbuf* on all images can be used.

4 EVALUATION

We evaluated the NICAM-DC-MINI using XACC on HA-PACS/TCA system, the specification of which is shown in Table 1. We used a PGI compiler as a backend compiler for OpenACC directives. The performance of the original NICAM-DC-MINI using OpenACC+MPI was also evaluated for comparison. In this experiment, an image is mapped to an MPI process at the Omni compiler's runtime library, and an MPI process is assigned to a single CPU core.

We executed both NICAM-DC-MINI applications in the target data "gl06rl01z80pe10" and "gl06rl01z80pe20" during 100 steps with

```
 1  do ro=1,romax(halo)
 2    call mpi_irecv(recvbuf(1,ro), &
 3                   rsize(ro,halo)*cmax, &
 4                   mpi_double_precision, &
 5                   sourcerank(ro,halo), &
 6                   recvtag(ro,halo), &
 7                   ADM_comm_run_world, &
 8                   areq(ro), &
 9                   ierr)
10  end do
11
12  do so=1,somax(halo)
13    call mpi_isend(sendbuf(1,so), &
14                   ssize(so,halo)*cmax, &
15                   mpi_double_precision, &
16                   destrank(so,halo), &
17                   sendtag(so,halo), &
18                   ADM_comm_run_world, &
19                   areq(so+romax(halo)), &
20                   ierr)
21  end do
22
23  call mpi_waitall(acount,areq,stat,ierr)
```

(a) Based code

```
1  sync all
2  do so=1,somax(halo)
3    recvbuf(1:ssize(so,halo)*cmax, dstimg(so)[destrank(so,halo)+1] =
           sendbuf(1:ssize(so,halo)*cmax,so)
4  end do
5  sync all
```

(b) New code

Figure 2: A code modification portion performed using XACC

strong scaling. The "gl06rl01z80pe10" is executed in 10 MPI processes, and the "gl06rl01z80pe20" is executed in 20 MPI processes. Fig. 3 shows the results where the performances of XACC are slightly inferior to those of OpenACC+MPI due to the additional communication described in Section 3. However, in terms of productivity, Fig. 2 shows that the coarray features can express communication more intuitively than MPI functions.

5 SUMMARY

In this paper, we report on the implantation of the NICAM-DC-MINI using the XACC coarray feature. While our evaluation showed that the performance of the XACC implementation was slightly inferior to the original OpenACC+MPI implementation, we believe

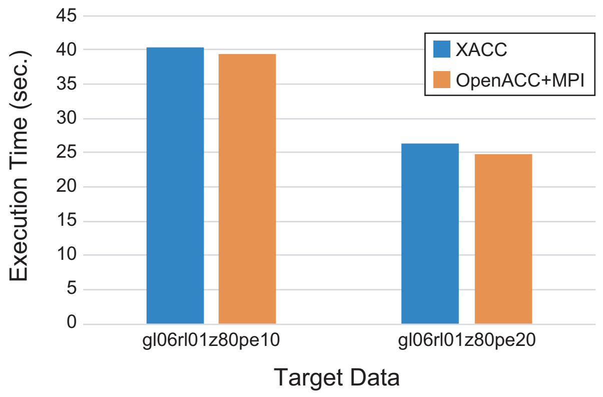

Figure 3: Performance results

the outlook of the XACC implementation could be improved considerably.

In our future work, we will seek to further improve on the performance we have achieved thus far. In the current implementation of Omni compiler, coarray communication is a blocking operation. Thus, it is considered likely that the performance could be improved by positively using asynchronous communication via coarray[2].

ACKNOWLEDGEMENTS

This research used the HA-PACS/TCA system provided by Interdisciplinary Computational Science Program in the Center for Computational Sciences, University of Tsukuba. The work was supported by the Japan Science and Technology Agency, Core Research for Evolutional Science and Technology program entitled "Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era" in the research area of "Development of System Software Technologies for Post-Peta Scale High Performance Computing."

REFERENCES

[1] A publication of the UPC Consortium. 2013. (November 2013). https://upc-lang.org/assets/Uploads/spec/upc-lang-spec-1.3.pdf.
[2] Akihiro Tabuchi et al. 2017. Implementation and Evaluation of One-sided PGAS Communication in XcalableACC for Accelerated Clusters. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (CCGrid '17)*.
[3] Chamberlain B.L. and Callahan D. and Zima H.P. 2007. Parallel Programmability and the Chapel Language. *Int. J. High Perform. Comput. Appl.* 21, 3 (Aug. 2007), 291–312.
[4] Charles Philippe and Grothoff Christian and Saraswat Vijay and Donawa Christopher and Kielstra Allan and Ebcioglu Kemal and von Praun Christoph and Sarkar Vivek. 2005. X10: An Object-oriented Approach to Non-uniform Cluster Computing. *SIGPLAN Not.* 40, 10 (Oct. 2005), 519–538.
[5] Karl Fürlinger, Tobias Fuchs, and Roger Kowalewski. 2016. DASH: A C++ PGAS Library for Distributed Data Structures and Parallel Algorithms. *CoRR* abs/1610.01482 (2016). arXiv:1610.01482 http://arxiv.org/abs/1610.01482
[6] XcalableACC Specification Working Group. 2017. XcalableACC Specification. (2017). `http://xcalablemp.org/XACC.html`.
[7] XcalableMP Specification Working Group. 2017. XcalableMP Specification. (2017). `http://xcalablemp.org/specification`.
[8] H. Murai and M. Nakao and H. Iwashita and M. Sato. 2017. Preliminary Performance Evaluation of Coarray-based Implementation of Fiber Miniapp Suite using XcalableMP PGAS Language. In *PGAS Applications Workshop*.
[9] Christiane Jablonowski and David L. Williamson. 2006. A baroclinic instability test case for atmospheric model dynamical cores. *Quarterly Journal of the Royal Meteorological Society* 132, 621C (2006), 2943–2975. https://doi.org/10.1256/qj.06.12
[10] Kumar Vivek and Zheng Yili and Cavé Vincent and Budimlić Zoran and Sarkar Vivek. 2014. HabaneroUPC++: A Compiler-free PGAS Library. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models (PGAS '14)*. Article 5, 10 pages.
[11] M. Nakao and H. Murai and H. Iwashita and A. Tabuchi and T. Boku and M. Sato. 2017. Implementing Lattice QCD Application with XcalableACC Language on Accelerated Cluster. In *2017 IEEE International Conference on Cluster Computing*. 429–438. https://doi.org/10.1109/CLUSTER.2017.58
[12] M. Nakao and H. Murai and H. Iwashita and T. Boku and M. Sato. 2017. Implementation and evaluation of the HPC challenge benchmark in the XcalableMP PGAS language. *The International Journal of High Performance Computing Applications* (2017), 1–14. https://doi.org/10.1177/1094342017698214
[13] Matthew Otten et al. 2016. An MPI/OpenACC implementation of a high-order electromagnetics solver with GPUDirect communication. *The International Journal of High Performance Computing Applications* 30, 3 (2016), 320–334.
[14] NICAM developers. 2014. NICAM-DC mini-application. (2014). https://github.com/fiber-miniapp/nicam-dc-mini.
[15] NICAM developers. 2015. NICAM dynamical core package. (2015). https://scale.aics.riken.jp/nicamdc/.
[16] Nowicki M. and Gorski L. and Grabrczyk P. and Bala P. 2014. PCJ - Java library for high performance computing in PGAS model. In *High Performance Computing Simulation (HPCS), 2014 International Conference on*. 202–209.
[17] Numrich Robert W. and Reid John. 1998. Co-array Fortran for parallel programming. *SIGPLAN Fortran Forum* 17, 2 (Aug. 1998), 1–31.
[18] Yili Zheng and Kamil, A. and Driscoll, M.B. and Hongzhang Shan and Yelick, K. 2014. UPC++: A PGAS Extension for C++. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. 1105–1114. https://doi.org/10.1109/IPDPS.2014.115