Performance Analysis of 2D-compatible 2.5D-PDGEMM on Knights Landing Cluster

Daichi Mukunoki[1,2(✉)] and Toshiyuki Imamura[1]

[1] RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan
{daichi.mukunoki,imamura.toshiyuki}@riken.jp
[2] Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan

Abstract. This paper discusses the performance of a parallel matrix multiplication routine (PDGEMM) that uses the 2.5D algorithm, which is a communication-reducing algorithm, on a cluster based on the Xeon Phi 7200-series (codenamed Knights Landing), Oakforest-PACS. Although the algorithm required a 2.5D matrix distribution instead of the conventional 2D distribution, it performed computations of 2D distributed matrices on a 2D process grid by redistributing the matrices (2D-compatible 2.5D-PDGEMM). Our use of up to 8192 nodes (8192 Xeon Phi processors) demonstrates that in terms of strong scaling, our implementation performs better than conventional 2D implementations.

Keywords: Parallel matrix multiplication · 2.5D algorithm
Xeon Phi · Knights landing

1 Introduction

Toward the Exa-scale computing era, the degree of parallelism (i.e., the numbers of nodes, cores, and processes) of HPC systems is increasing. On such highly parallel systems, computations can become communication-bound when the size of a problem is insufficiently large, even if the computation is a compute-intensive task, such as parallel matrix multiplication (the so-called PDGEMM in ScaLAPACK). Consequently, communication-avoiding techniques have been the focus of research to improve performance of computations in terms of strong scaling on highly parallel systems.

For PDGEMM, the 2.5D algorithm (2.5D-PDGEMM) has been proposed as a communication-avoiding algorithm [4] that assumes a 2D distribution of matrices stacked and duplicated vertically in a 3D process grid (the 2.5D distribution). The 2.5D algorithm decreases the number of the computational steps of the 2D algorithms (e.g., Cannon and SUMMA) by parallelizing the steps by utilizing the redundancy of the matrices in the 2.5D distribution, unlike the conventional

Y. Shi et al. (Eds.): ICCS 2018, LNCS 10862, pp. 853–858, 2018.
https://doi.org/10.1007/978-3-319-93713-7_85

PDGEMM, which uses 2D algorithms that are computed in parallel only by utilizing the 2D data parallelism of the matrices. Thus, if the 2.5D algorithm is used to compute matrices distributed in a 2D process grid with a 2D distribution, as executed by ScaLAPACK PDGEMM, the matrices must be redistributed from 2D to 2.5D. The implementation and performance of 2.5D-PDGEMM have been featured in several studies [1,3,4]; however, so far, these studies have not addressed the 2D compatibility.

We expect that such a 2D compatibility would be required so that, in the future, applications using the conventional PDGEMM would be able to achieve good strong scalability on highly parallel systems. Our previous study proposed a 2D-compatible 2.5D-PDGEMM implementation that computes matrices distributed on a 2D process grid with a 2D distribution and analyzed the performance of up to 16384 nodes on the K computer [2]. This implementation outperformed conventional 2D implementations, including the ScaLAPACK PDGEMM, in terms of strong scaling, even when the cost of the matrix redistribution between 2D and 2.5D was included.

This paper presents the results of our 2D-compatible 2.5D-PDGEMM implementation on the Oakforest-PACS system, which is a Xeon Phi 7200-series (codenamed Knights Landing) based cluster hosted by the Joint Center for Advanced High Performance Computing (JCAHPC), Japan. The system is equipped with 8208 Xeon Phi processors and was ranked numer six on the TOP500 list in November 2016. Our study used 8192 of the processors to assess the performance and effectiveness of the 2D-compatible 2.5D-PDGEMM on Xeon Phi-based supercomputers.

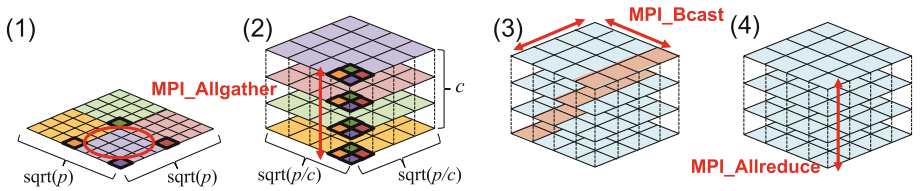

Fig. 1. Implementation of 2D-compatible 2.5D-PDGEMM

2 Implementation

Our 2D-compatible 2.5D-PDGEMM is based on the SUMMA algorithm [5] and computes $C = \alpha AB + \beta C$, where α and β are scalar values, and whereas A, B, and C are dense matrices distributed on a 2D process grid with a 2D block distribution. For simplicity, our current implementation only supports square matrices, a square process grid, and a 2D block distribution. Figure 1 summarizes our implementation. p corresponds to the total number of processes joining the computation and c corresponds to the stack size (duplication factor) of the 2.5D

algorithm. Thus, the matrices are initially distributed on a 2D process grid of $p = \sqrt{p} \times \sqrt{p}$ processes with a 2D distribution.

The computation is executed as follows: (1) the MPI_Comm_split creates the 3D logical process grid from a 2D process grid by dividing the original 2D process grid to form the levels of the 3D process grid; (2) the MPI_Allgather redistributes and duplicates matrices A and B; (3) the 2.5D algorithm executes $1/c$ of the SUMMA algorithm's steps at each level of the 3D process grid using DGEMM and MPI_Bcast; and (4) the MPI_Allreduce computes the final result for matrix C by reducing and redistributing the temporal results of the matrix on each level of the 3D process grid.

Table 1. Environment and conditions of evaluation

Processor	Intel Xeon Phi 7250 (Knights Landing, 1.4 GHz, 68 cores)
Memory	MCDRAM (16 GB) + DDR4 (96 GB)
Interconnect	Intel Omni-Path Architecture (100 Gbps, Full-bisection Fat Tree)
Compiler	Intel compiler 18.0.1
MKL	Intel MKL 2018.1
MPI	Intel MPI 2018.1.163
MPI options	OMP_NUM_THREADS = 16, I_MPI_PIN_DOMAIN = 64
	I_MPI_PIN_PROCESSOR_EXCLUDE_LIST = 0,1,68,69,136,137,204,205
	I_MPI_PERHOST = 4, KMP_AFFINITY = scatter,
	KMP_HW_SUBSET = 1t, I_MPI_FABRICS = tmi:tmi,
	HFI_NO_CPUAFFINITY = 1

3 Results

We evaluated the performance of our implementation using 8192 nodes on the Oakforest-PACS system. Table 1 summarizes the environment and conditions of the evaluation. Each node is equipped with one processor; therefore, the number of nodes is equal to the number of processors. The parallel execution model was a hybrid of the MPI and the OpenMP; however, hyperthreading was not used (1 OpenMP thread per core). We assigned 4 MPI processes per node when the number of the nodes was 256, 1024, and 4096 for performance reasons; however, due to a limitation in our implementation, we assigned 2 MPI processes per node when the number of the nodes was 128, 512, 2048, and 8192. On the Oakforest-PACS system, the tickless mode was set for the core number 0 only to receive timer interruptions. Therefore, the logical cores 0 and 1 were excluded to avoid the effects of OS jitter (the logical core 1 was also excluded, as both belong to the same core group).

We evaluated the performance of our implementation with stack sizes of $c = 1, 4$, and 16. Our implementation was designed to perform equivalently to the conventional 2D-SUMMA when $c = 1$. In addition, we measured the

performance of the ScaLAPACK PDGEMM for reference (the block size $nb = 512$ at maximum). We excluded the MPI sub-communicator setup cost from the execution time, because, on ScaLAPACK, such a communicator setup process becomes separated from the PDGEMM routine.

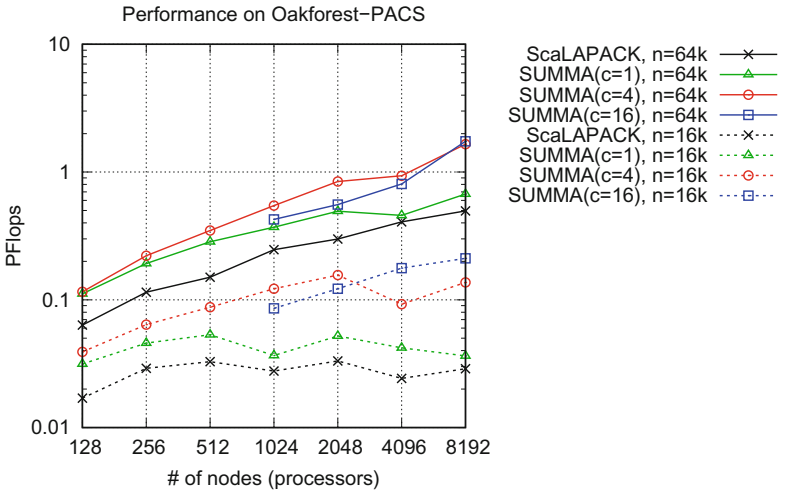

Fig. 2. Strong scaling performances for matrix sizes n = 16384 (16 k) and n = 65536 (64 k)

Figure 2 shows the strong scaling performances when the matrix size n = 16384 (16 k) and n = 65536 (64 k). Figure 3 shows the breakdown of the performances when the matrix size n = 65536. Bcast corresponds to the communication cost of using the SUMMA algorithm for the 2.5D matrix multiplication whereas Allgather and Allreduce correspond to the communication costs for redistribution and reduction.

Overall, the results show that the effectiveness of the 2.5D algorithm is higher than that we observed in our previous work on the K computer [2]. The factor that may most strongly cause the difference between the results of the Oakforest-PACS and the K computer is the ratio of the computation performance to the communication performance. Whereas the K computer has a relatively richer network compared with the floating-point performance per node, i.e., 20 [GB/s]/128 [GFlops] ≈ 0.16, the Oakforest-PACS system is a typical example of modern supercomputers, which has a relatively huge floating-point performance per node compared with the network's performance, i.e., 25 [GB/s]/3046.4 [GFlops] ≈ 0.0082. Thus, the Oakforest-PACS is approximately 20 times more a "Flops-oriented" system than the K computer is. The evaluation indicates that the 2.5D-PDGEMM is more effective as a communication-avoiding technique in such environments.

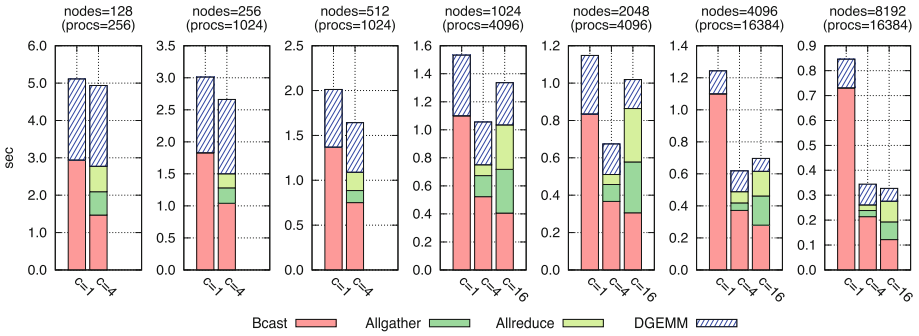

Fig. 3. Performance breakdown (matrix size n=65536)

4 Conclusions

This paper presented the evaluation of the performance of our 2D-compatible 2.5D-PDGEMM, which was designed to execute computations of 2D distributed matrices by using up to 8192 Xeon Phi Knights Landing processors on the Oakforest-PACS system. The results demonstrated that, on recent HPC systems, such as the Oakforest-PACS, which provide huge floating-point performance as compared with the network's performance, a 2D-compatible 2.5D-PDGEMM was quite effective as a substitute for the conventional 2D-PDGEMM. Beyond this study, we will further analyze and estimate the performance of the 2.5D-PDGEMM on future Exa-scale systems by creating a performance model and using a system simulator.

Acknowledgment. The computational resource of the Oakforest-PACS was awarded by the "Large-scale HPC Challenge" Project, Joint Center for Advanced High Performance Computing (JCAHPC). This study is supported by the FLAGSHIP2020 project.

References

1. Georganas, E., González-Domínguez, J., Solomonik, E., Zheng, Y., Touriño, J., Yelick, K.: Communication Avoiding and Overlapping for Numerical Linear Algebra. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2012), pp. 100:1–100:11 (2012)
2. Mukunoki, D., Imamura, T.: Implementation and performance analysis of 2.5D-PDGEMM on the K computer. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (eds.) PPAM 2017. LNCS, vol. 10777, pp. 348–358. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78024-5_31
3. Schatz, M., Van de Geijn, R.A., Poulson, J.: Parallel matrix multiplication: a systematic journey. SIAM J. Sci. Comput. **38**(6), C748–C781 (2016)

4. Solomonik, E., Demmel, J.: Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011. LNCS, vol. 6853, pp. 90–109. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23397-5_10
5. Van de Geijn, R.A., Watts, J.: SUMMA: Scalable Universal Matrix Multiplication Algorithm, Technical report, Department of Computer Science, University of Texas at Austin (1995)