Implementation and Evaluation of NAS Parallel CG Benchmark on GPU Cluster with Proprietary Interconnect TCA

Kazuya Matsumoto[1], Norihisa Fujita[2],
Toshihiro Hanawa[3], and Taisuke Boku[1,2]

[1] Center for Computational Sciences, University of Tsukuba
[2] Graduate School of Systems and Information Engineering, University of Tsukuba
[3] Information Technology Center, The University of Tokyo

Abstract. We have been developing a proprietary interconnect technology called Tightly Coupled Accelerators (TCA) architecture to improve communication latency and bandwidth between accelerators (GPUs) over different nodes. This paper presents a Conjugate Gradient (CG) benchmark implementation using the TCA and results of performance evaluation on the HA-PACS/TCA system, which is a proof-of-concept GPU cluster based on the TCA concept. The implementation is based on the CG benchmark in NAS Parallel Benchmarks, and its parallelization is achieved by a two-dimensional decomposition of matrix data. The TCA utilization improves the communication performance compared with the implementation with MPI/InfiniBand utilization for small size benchmark classes. This study also shows that the CG implementation with the two-dimensional decomposition is more suitable for the TCA utilization than a CG implementation with a one-dimensional decomposition to make use of the interconnect.

1 Introduction

Currently, GPU clusters are widely used as high performance computing systems. A problem of GPU clusters is that the communication speed between multiple compute nodes is not fast enough compared to its high computation speed. In order to address this problem, we have been researching the Tightly Coupled Accelerators (TCA) architecture [2]. The TCA is a technology on a proprietary interconnect network to enable direct communication between accelerators over different nodes.

We have conducted the basic performance evaluation of the TCA [4, 1]. In [4], a Conjugate Gradient (CG) method has been implemented by utilizing allgather and allreduce collective communications with TCA's communication functions. The parallelization of the CG implementation is accomplished by a one-dimensional decomposition of matrix data. Results of the performance evaluation shows that the CG method implementation using TCA outperforms the implementation using MPI/InfiniBand for sparse matrices whose matrix (problem) size is nine thousand or smaller; however, the communication performance of TCA tends to become lower when either or both of the target matrix sizes and the number of utilizing processes are larger.

In the present study, we evaluate a different CG method implementation with the TCA utilization. The CG implementation is based on the CG benchmark in NAS Parallel Benchmarks. We apply a two-dimensional decomposition of matrix as the enhanced implementation from [4], and present results of performance evaluation on the HA-PACS/TCA GPU cluster. Additionally, this study presents communication performance differences between the CG implementation with the two-dimensional decomposition and the one-dimensional decomposition.

2 Tightly Coupled Accelerators Architecture

This section briefly explains the Tightly Coupled Accelerators (TCA) architecture (see [2, 1] for detailed information on the TCA). The TCA is a novel technology of proprietary interconnect for PC clusters. The PCI Express Adaptive Communication Hub ver. 2 (PEACH2) is a prototype implementation of the TCA architecture. We can construct a cluster system by connecting the PEACH2 boards with each other. The communication using the PEACH2 is conducted only with the PCIe protocol, and the overhead time for protocol conversion, which is required in the InfiniBand, is eliminated. Consequently, the PEACH2 enables data communication with extremely low latency. The PEACH2 provides two types of data communication functions: PIO and DMA. In the PIO communication, the data is transferred to a remote node by the CPU's remote write operation. The latency of PIO is very small, and, as a result, the PIO is useful to transfer short messages. The DMA function is achieved by the DMA controller, which has four DMA channels. While the latency of DMA is larger than that of PIO, the DMA demonstrates higher maximum bandwidth performance.

The HA-PACS (Highly Accelerated Parallel Advanced system for Computational Sciences) is a GPU cluster system at the Center for Computational Sciences, University of Tsukuba. The HA-PACS/TCA is a proof-of-concept system of TCA architecture concept and a performance evaluation test-bed of PEACH2 board. The HA-PACS/TCA contains the PEACH2 as an interconnect adapter in addition to a commodity InfiniBand interconnect. Table 1 shows the specification of HA-PACS/TCA. The HA-PACS/TCA consists of four sub-clusters. Each sub-cluster is composed of 16 compute nodes. The 16 nodes are connected by the PEACH2 and configure a 2×8 torus network. Note that the 64 nodes of HA-PACS/TCA are connected also by two ports of InfiniBand QDR in a fat-tree configuration with full bisection bandwidth.

3 Implementation

The CG benchmark in NAS Parallel Benchmarks (NPB) is originally written in Fortran. Though several CG benchmark studies on GPUs have been conducted [3, 8], there is no available implementations written in MPI and CUDA to our knowledge. Because of the fact, in the present study, we modify the MPI version

Table 1. HA-PACS/TCA system configuration

Node configuration	
Motherboard	SuperMicro X9DRG-QF
CPU	Intel Xeon E5-2680 v2 2.8 GHz × 2 (IvyBridge 10 cores / CPU)
Memory	DDR3 1866 MHz × 4 ch, 128 GB (=8 × 16 GB)
Peak performance	224 Gflops / CPU
GPU	NVIDIA Tesla K20X 732 MHz × 4 (Kepler GK110 2688 cores / GPU)
Memory	GDDR5 6 GB / GPU
Peak performance	1.31 Tflops / GPU
Interconnect	InfiniBand: Mellanox Connect-X3 Dual-port QDR TCA: PEACH2 board (Altera Stratix-IV GX 530 FPGA)
System configuration	
Number of nodes	64
Interconnect	InfiniBand QDR 108 ports switch × 2 ch
Peak performance	364 Tflops

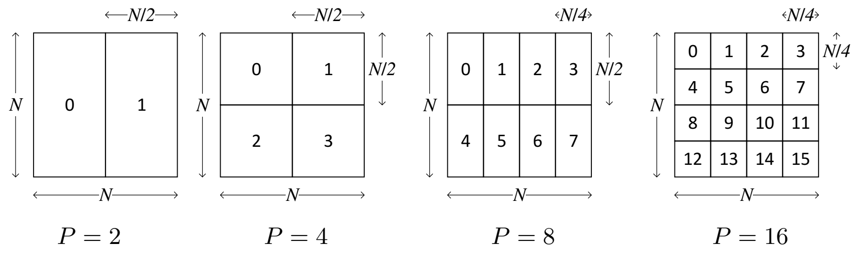

Fig. 1. Process distribution by two-dimensional decomposition of matrix data. The number in each rectangular represents its process rank id.

of CG benchmark in NAS 3.3.1 such that the main computation part (`conj_grad` function) is written in C language and CUDA, and its inter-node data communications are conducted with the TCA/PEACH2. Let us denote a linear equation system as $Ax = b$, where A is an $N \times N$ symmetric positive definite matrix, and both x and b are a vector with N elements in the following.

The modified implementation uses the identical data distribution and communication pattern to the original MPI version. The parallelization of CG benchmark is achieved by a two-dimensional decomposition of the matrix A and conformable distribution of vectors. When we define the number of processes as P, the matrix A is two-dimensionally decomposed by $P = P_r \times P_c$ processes. Based on the decomposition, each process contains $(N/P_r) \times (N/P_c)$ sub-matrix of A and vectors with N/P_c elements. Figure 1 shows the process distribution by two-dimensional decomposition of matrix A data for $P = 2, 4, 8, 16$.

Almost all of computations in the implementation are carried out by GPUs. While we implement the sparse-matrix vector multiplication (SpMV) by ourselves, our CG implementation utilizes the NVIDIA's CUBLAS library for vector dot product (DOT) and vector addition (AXPY) operations. Note that the computation part is not tuned so deeply since this study focuses on performance evaluation of data communication.

Three kinds of data communication are required in the implementation. The first required communication is to send vector data for obtaining the product of SpMV after the local SpMV computation on a GPU in each node. The communication is conducted among P_c processes in the same process row in a binary tree fashion, and thus $\sqrt{P_c}$ communication steps are needed (each step needs to send $8N/P_c$ Bytes of vector data in double precision). The second communication is to send a scalar (8 Bytes) value to compute the sum of local product by the DOT computation. This communication is also made among the P_c processes and $\sqrt{P_c}$ steps are required. The third communication is to send $8N/P_c$ Bytes of a vector for data exchange. In the following, let us name the first, second and third communication as COMM_SpMV, COMM_DOT and COMM_EXCH, respectively.

The COMM_DOT is scalar data communications between CPU memories of different processes and the latency for issuing communication operations occupies almost all of its communication time. The COMM_SpMV and COMM_EXCH are block data communications of $8N/P_c$ Bytes between different GPU memories and a communication bandwidth is important for high performance communication as well as the issue latency. Considering the communication characteristics, we implement the COMM_SpMV and COMM_EXCH with the DMA communication function of TCA/PEACH2 and implement the COMM_DOT with the PIO function. Note that, as shown in Figure 2, the DMA communication is faster than the PIO communication when message sizes are larger than 128 Bytes, which are smaller than sizes for the required block communications in any problem classes of CG benchmark.

The TCA/PEACH2 configures a 2×8 torus network on a sub-cluster of HA-PACS/TCA; thus, a way of process (node) mapping also affects the communication performance. As can be seen from Fig. 1, the CG implementation with two-dimensional decomposition requires communication among P_c processes (4 processes at most when we utilize up to 16 nodes). We use a node mapping shown in Figure 3. This mapping does not cause message data contentions and collisions within the TCA/PEACH2's communication network on a sub-cluster even when $P = 16$ cases.

4 Performance Evaluation

We conduct performance measurements on a sub-cluster of the HA-PACS/TCA. A single GPU and a single CPU socket are utilized per node[4]. For comparison with the implementation using TCA/PEACH2, this section also presents the

[4] This is because using two or more sub-clusters entails a hybrid utilization of the TCA/PEACH2 and MPI/IB, and because two or more GPUs usage requires additional considerations to use the TCA/PEACH2 effectively. Both of the hybrid utilization and the multi GPU usage are among our future work.

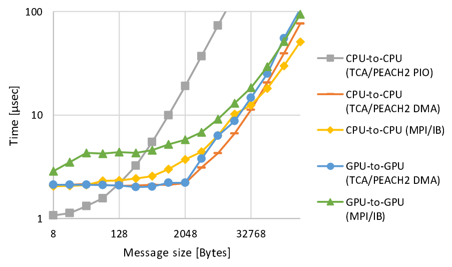

Fig. 2. Ping-pong communication performance between two neighboring nodes on the HA-PACS/TCA.

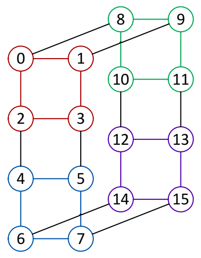

Fig. 3. Node mapping optimized for CG benchmark implementation on a sub-cluster of HA-PACS/TCA. Circles represent compute nodes, lines between circles represent data links, and the number corresponds to its process rank id.

performance of an implementation using MPI/InfiniBand (MPI/IB) for inter-node communications. We use the MVAPICH2 GDR 2.1a (MV2GDR) MPI library implementation [6]. As with the TCA/PEACH2, the MV2GDR utilizes the GPUDirect for RDMA (GDR) technology [5] for direct communication between GPUs. Note that the theoretical peak bandwidth of TCA/PEACH2 (PCIe Gen2 x8) is twice lower than that of MPI/IB (dual-rail InfiniBand QDR[5]); therefore, the implementation using TCA/PEACH2 is outperformed when message sizes become large. On the condition where the program is compiled by Intel C compiler 15.0.2 with MV2GDR 2.1a and CUDA 6.5 usage, the TCA/PEACH2 is faster for message sizes up to 64 KB than the MPI/IB in terms of the ping-pong communication performance as shown in Fig. 2.

In the CG benchmark, the problem sizes (CLASS) and the number of processes (P) can be designated. This section presents results of performance evaluations for CLASS=S, W, A, B and $P = 2, 4, 8, 16$. The problem (matrix/vector) size N is 1,400 for CLASS=S, 7,000 for CLASS=W, 14,000 for CLASS=A, and 75,000 for CLASS=B. We measure the consuming time for each computation/communication operation in the `conj_grad` program function. Figure 4 shows the measured time breakdown on average time of ten times calls to the function. Note that this is the breakdown of process rank 0 and the communication time is the communication wait time. A single call of the `conj_grad` function includes $26\sqrt{P_c}$ of COMM_SpMV, $52\sqrt{P_c}$ of COMM_DOT, 26 of COMM_EXCH, 26 of SpMV, 52 of DOT, and $76 + 26\sqrt{P_c}$ of AXPY operations. In Fig. 4, the performance is shown for the implementation in Fortran (original NPB-MPI

[5] The theoretical peak bandwidth of the dual-rail InfiniBand QDR is 8 GB/s, which is equivalent to that of PCIe Gen3 x8.

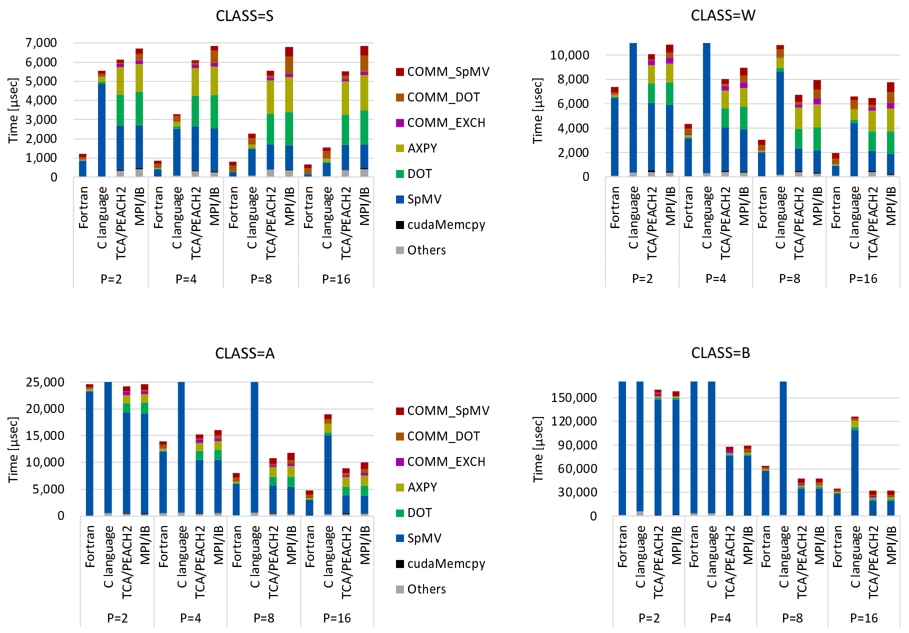

Fig. 4. Time breakdown on a single `conj_grad` function call of CG benchmark. In each figure, the performance (time in microsecond) is shown for the implementation in Fortran (original NPB-MPI code), C language, CUDA with TCA/PEACH2 communication, and CUDA with MPI/IB communication. The upper plot results for several results in Fortran and C, such as CLASS=B & P=2 case, are cut for simplicity.

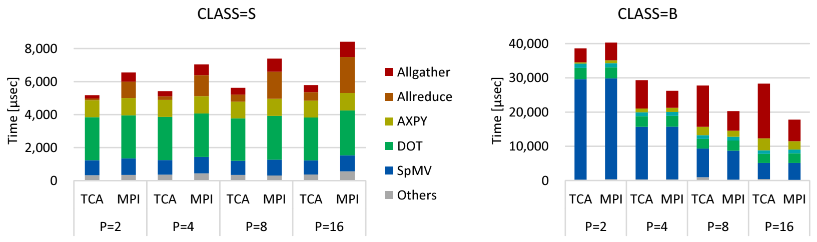

Fig. 5. Time breakdown of CG implementation with one-dimensional decomposition

code), C language, CUDA with TCA/PEACH2 communication, and CUDA with MPI/IB communication.

The Fortran implementation is faster than the C implementation (similar performance results were reported for the NPB-LU benchmark by Pennycook et al. [7]). Compared with the original Fortran implementation, the GPU/CUDA usage for NPB implementation deteriorates the performance in several cases (particularly in cases of smaller size class and larger number of processes utilization). This is mainly due to that the GPU usage brings bigger overheads for the CUDA kernel invocations[6] and for the inter-node communication call between GPUs.

The TCA/PEACH2 utilization contributes the performance improvement for the three small size classes (CLASS=S, CLASS=W, and CLASS=A) compared with the MPI/IB utilization. Especially, the communication performance is 3.00 times higher in the case CLASS=S with $P = 16$, and the overall performance including the computation time and communication time is 1.24 times higher. The large performance improvement by TCA/PEACH2 is derived from improvements of COMM_SpMV and COMM_DOT. The communication time is 2.42 times shorter for COMM_SpMV and 4.72 times shorter for COMM_DOT in this case. In the case CLASS=A with $P = 16$, the communication performance using the TCA/PEACH2 is 1.44 times higher and the overall performance is 1.12 times higher. The performance of COMM_DOT with TCA/PEACH2 is higher in all four size cases since the communication latency mostly decides the performance for the scalar data communication. However, the TCA/PEACH2 is not always effective. The performance for CLASS=B is almost same. In the CLASS=B case, the COMM_EXCH is the main performance bottleneck.

To see how the performance is different between the CG benchmark implementation with two-dimensional decomposition (2D-CG) in the present study and the CG implementation with one-dimensional decomposition (1D-CG) in our previous study [4], we additionally measure the performance of 1D-CG implementation on the equivalent condition and cases (1D-CG is implemented by ourselves and not in the NAS Parallel Benchmarks). Figure 5 shows the measured time breakdown of the 1D-CG for matrices corresponding CLASS=S and CLASS=B. The 1D-CG utilizes allgather and allreduce collective communications (see [4] for implementation details). All P processes are involved for both the collective communications in 1D-CG, whereas P_c processes are involved at most in 2D-CG. Since the network topology of TCA/PEACH2 is 2×8 torus network, collisions within the communication network cannot be avoided for $P = 16$ cases in 1D-CG. As shown in Fig. 5, the communication time in 1D-CG becomes larger when P increases. In addition, the largest message size of 1D-CG is larger than that of 2D-CG for $P = 8, 16$ cases (the size is $8N/2$ Bytes in 1D-CG and $8N/P_c$ in 2D-CG). This fact is disadvantage for large matrix sizes (CLASS=B) and relative performance differences between TCA/PEACH2 and MPI/IB is large compared with 2D-CG implementation. In general, the message size of each communication in 2D-CG is shorter than or equal to that in 1D-CG for corresponding communication. The performance degradation with shorter message size is serious in MPI/IB while TCA/PEACH2 provides a good performance thanks to its very small latency. Thus, the combination of such short messages and avoidance of message collision on the torus network of TCA/PEACH2 leads this performance improvement on 2D-CG benchmark.

[6] A CUDA kernel invocation at least takes 9.7 μsec, including the CUDA stream synchronization time, in our measurement.

5 Conclusion

The present study has utilized the TCA/PEACH2 for an implementation of NAS Parallel CG benchmark and conducted its performance evaluation on the HA-PACS/TCA GPU cluster. Results of the performance evaluation show that the CG implementation with the parallelization by a two-dimensional decomposition of matrix data does not cause message data collisions within the communication network of TCA/PEACH when processes are properly mapped to nodes; and the present CG implementation is considered to be better suited for the TCA/PEACH2 utilization than the previous CG method implementation with a one-dimensional decomposition [4]. The performance improvement over MPI/IB utilization is due to the very small latency of TCA/PEACH2. We will continue researches on the TCA with the view that reducing the latency between accelerators by direct communication is important for strong-scaling computing.

Acknowledgements. The present study was supported by the Japan Science and Technology Agency's CREST program entitled "Research and Development of Unified Environment on Accelerated Computing and Interconnection for Post-Petascale Era." The authors would like to thank the Center for Computational Sciences, University of Tsukuba for allowing us to use the HA-PACS/TCA system as part of the interdisciplinary Collaborative Research Program.

References

1. Hanawa, T., Fujii, H., Fujita, N., Odajima, T., Matsumoto, K., Boku, T.: Evaluation of FFT for GPU Cluster Using Tightly Coupled Accelerators Architecture. In: Proc. Cluster 2015. pp. 635–641. IEEE (2015)
2. Hanawa, T., Kodama, Y., Boku, T., Sato, M.: Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators. In: Proc. IPDPSW 2013. pp. 1030–1039. IEEE (2013)
3. Lee, S., Vetter, J.S.: Early evaluation of directive-based GPU programming models for productive exascale computing. In: Proc. SC '12 (2012)
4. Matsumoto, K., Hanawa, T., Kodama, Y., Fujii, H., Boku, T.: Implementation of CG Method on GPU Cluster with Proprietary Interconnect TCA for GPU Direct Communication. In: Proc. IPDPSW 2015. pp. 647–655. IEEE (2015)
5. NVIDIA: NVIDIA GPUDirect (Accessed April 25, 2016), https://developer.nvidia.com/gpudirect
6. Panda, D.K.: MVAPICH2-GDR (MVAPICH2 with GPUDirect RDMA) (Accessed April 25, 2016), http://mvapich.cse.ohio-state.edu/overview/
7. Pennycook, S.J., Hammond, S.D., Jarvis, S.A., Mudalige, G.R.: Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark. SIGMETRICS Perform. Eval. Rev. 38(4), 23–29 (2011), http://doi.acm.org/10.1145/1964218.1964223
8. Xu, R., Tian, X., Chadrasekaran, S., Yan, Y., Chapman, B.: NAS Parallel Benchmarks for GPGPUs Using a Directive-Based Programming Model. In: Languages and Compilers for Parallel Computing. LNCS, vol. 8967, pp. 67–81. Springer (2015)