CFIO2: Overlapping Communications and I/O with Computations Using RDMA Technology

Cheng Zhang, Xiaomeng Huang, Yong Hu, Shizhen Xu, Haohuan Fu,
and Guangwen Yang

Ministry of Education Key Laboratory for Earth System Modeling,
Center for Earth System Science, Tsinghua University, 100084,
and Joint Center for Global Change Studies, Beijing, 100875, China
zhangcheng12@mails.tsinghua.edu.cn

Abstract. The output data produced by numerical climate model simulations have increased greatly in complexity and size. The exploding volume of climate data is becoming a challenge for climate scientists. Our previous work, Climate Fast Input/Output (CFIO) library, implemented a two-phase I/O method to overlap I/O with computations, and achieved high throughput. In this paper, we present CFIO2, which can overlap communications with computations using Remote Direct Memory Access(RDMA) technology. We design a simple communication interactions model to implement asynchronous and concurrent data transfer. The experimental results show that CFIO2 can provide higher throughput than CFIO and can shorten the overall simulation time for climate model significantly.

1 System Architecture

CFIO [1] [2] utilizes a two-phase I/O method to offload its I/O tasks. MPI processes in CFIO are divided into two parts: I/O clients and I/O servers. When using CFIO in climate models, the clients are assigned for computation, and servers for I/O. After completion of several simulation steps, clients forward all output data to servers, and continue their next simulation step immediately. The I/O tasks are offload to servers, and servers execute I/O tasks by invoking PnetCDF interface separately. Thus, CFIO can overlap I/O with computations.

The data transfer between clients and servers, which is called communication phase as well in this paper, takes considerable time in overall simulation. The data transfer method of CFIO is to call MPI interfaces of send and receive correspondingly. It's obviously a synchronous way so this phase can not be overlapped. CFIO had also attempted to communicate asynchronously using multi-threads, but great impair is encountered due to competition of resources including CPU and network. In this work, our target is to overlap the communication phase with the computation phase as well.

To eliminate communication overhead on client side, we introduce a RDMA communication layer to CFIO, constructing CFIO2 as figure 1 illustrates. Using RDMA write/read operations, connected client and server could access the

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 542–545, 2014.
© IFIP International Federation for Information Processing 2014

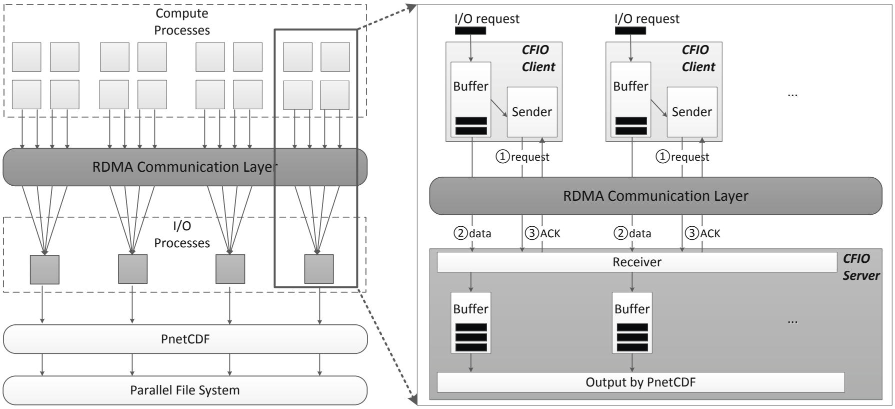

Fig. 1. The system architecture of CFIO2

memory of each other directly. What important is those access actions will not interrupt the process or occupy the CPU of the peer side. So asynchronous communication can be implemented.

Overlapping of communications and I/O with computations can be achieved via this asynchronous communication. Client just asynchronously posts I/O request by writing it's memory descriptor to server, and turns back to next computation phase immediately, without waiting for the completion of later communication and I/O. Server fetches output data from client's memory according to that descriptor, and acknowledges client about the completion of data transfer, both with asynchronous RDMA operations. Therefore, communication phase and I/O phase are both invisible at client's view.

2 Communication Interactions

To support RDMA communications, two kinds of memory region are registered at each process. One is used to buffer the output data, named data buffer. Another one is used to record key addresses of data buffer, start address as an example. We call this memory region as address buffer or descriptor of data buffer.

Figure 2 illustrates the scenario of communication interactions between one client and one server. Client (1) products some output data after one computation phase, which is stored in an area ranging from address *used* to address *free* of data buffer; (2) updates address buffer using the key addresses of data buffer, and importantly, changes flag to *1*; (3) copies the content of its address buffer to server's address buffer with RDMA write; and (4) turns to next computation phase immediately. Server (3) finds that flag of its address buffer is changed from *0* to *1*, recognizing that there are output data prepared for I/O. Then, it (5)

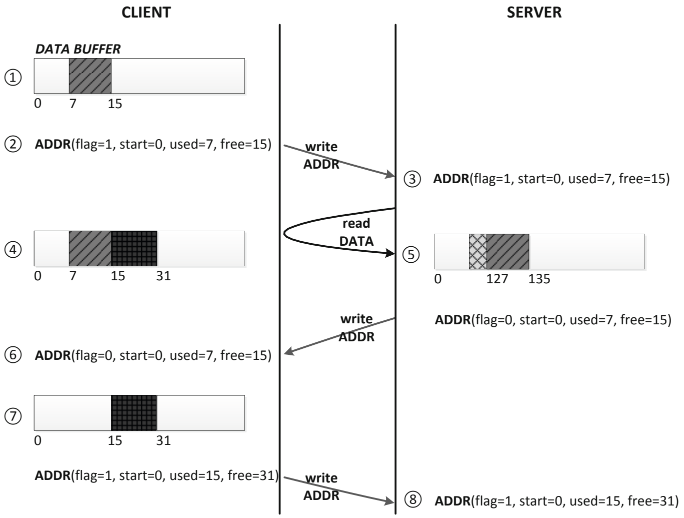
CLIENT
SERVER
DATA BUFFER
①
0 7 15
② ADDR(flag=1, start=0, used=7, free=15)
write ADDR
③ ADDR(flag=1, start=0, used=7, free=15)
read DATA
④
0 7 15 31
⑤
0 127 135
write ADDR
ADDR(flag=0, start=0, used=7, free=15)
⑥ ADDR(flag=0, start=0, used=7, free=15)
⑦
0 15 31
ADDR(flag=1, start=0, used=15, free=31)
write ADDR
⑧ ADDR(flag=1, start=0, used=15, free=31)

Fig. 2. An example of communication interactions between client and server

posts requests of RDMA read to get data from client according to the content of address buffer. Once all output data is fetched or local data buffer is full, (6) server updates its address buffer, changes its flag back to *0*, and writes content of address buffer to client. This address buffer describes which area of client's data buffer has been read. Client (7) updates its arguments of data buffer according to newly updated address buffer, noticed the change of flag. It then (8) updates address buffer and posts a write request again.

3 Results

We performed experiments on the *Explore*100 cluster of Tsinghua University. Each cluster node contains 2 2.93GHz Intel Xeon X5670 6-core processers, and 32G or 48G main memory. Those nodes are interconnect with InfiniBand, which provides a peak bandwidth of 40 Gb/s. The file system of *Explore*100 is Lustre, which consists of 1 Meta-Data server(MDS) and 40 Object Storage Targets(OST).

3.1 Write Performance

To measure the raw output performance, we tested CFIO2 with different proportions of client and server. This test case contains 20 iterations, and in each iteration the clients forward 3.2GB output data to servers. The number of clients is fixed to 256.

Figure 3 shows the write throughput of PnetCDF, CFIO, and CFIO2. The X-axis stands for the number of processes that call PnetCDF interface to execute parallel I/O. As expectation, write throughput of CFIO and CFIO2 are both lower than PnetCDF. CFIO2 reaches its peak throughput of 1.09 GB/s when running with 128 servers. CFIO2 Performs better than CFIO, because it takes advantage of event-driven method and achieve higher throughput of communication.

3.2 Overlapping Evaluation

To confirm the advantage of overlapping communication with computation, we imitated a climate model with typical I/O pattern. This experiment executes 20 iterations of computation and I/O. Each computation iteration costs 4.5 seconds and produce 3.2GB output data. The number of servers is 64.

The result is illustrated in figure 4. PnetCDF takes more total time than CFIO and CFIO2, because no overlapping occurs. CFIO can overlap I/O phase with computation phase, so it outperforms PnetCDF significantly. The iteration time of CFIO2 clients is about 91 seconds, which is very close to the pure computation time. And on the server side, the total time for communication and I/O of each iteration is less than 4.5 seconds. So, CFIO2 manages to overlap the communication phase with computation phase. As to both the CFIO and CFIO2, the gap of server and client is the data transfer time plus I/O time of the last iteration.

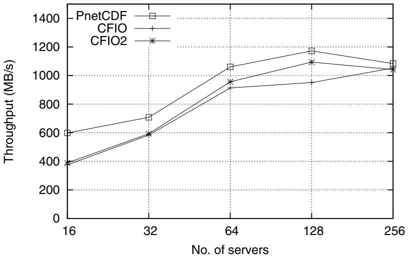

Fig. 3. I/O throughput

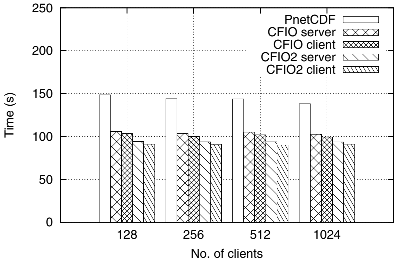

Fig. 4. Overall simulation time

References

1. Wang, W., Huang, X., Fu, H., et al.: CFIO: A Fast I/O Library for Climate Models. In: 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 911–918. IEEE (2013)
2. Huang, X., et al.: A fast input/output library for high resolution climate models. Geoscientific Model Development 7(1), 93–103 (2014)