EasyChair Preprint

№ 8259

Non-Blocking Atomic Snapshot Algorithms in MPI Remote Memory Access

Naveen, Alexey A. Paznikov and
Mujtaba Nazar Kadhim Al-Khaykanee

June 12, 2022

Non-blocking atomic snapshot algorithms in MPI remote memory access

Naveen
Informatics and Computer Engineering
St. Petersburg Electrotechnical University "LETI"
naveensheoran1000@gmail.com

Alexey A. Paznikov
Informatics and Computer Engineering
St. Petersburg Electrotechnical University "LETI"
apaznikov@gmail.com

Al-Khayakanee Mujtaba Nazar Kadhim
Informatics and Computer Engineering
St. Petersburg Electrotechnical University "LETI"
mujtabatoby2b@gmail.com

ABSTRACT

In this paper, we present the non-blocking atomic snapshot algorithms for High-Performance Computing in MPI RMA model. An Atomic Snapshot is useful for remote memory access where different processes have access to the concurrent data structure. Applications of atomic snapshot building multi-writer registers from single-writer registers, radar tracking system, counters, accumulators, check-pointing and concurrent backups, etc. Snapshots also useful for monitoring the parallel systems. An Atomic snapshot contains two operations update and scan.

In update processor writes the content to associated location and scan gives the linearizable view of all n segments. This paper presents non-blocking atomic snapshot in which update used MPI_Accumulate and MPI_Compare_and_swap atomic operations. In scan we used MPI_Get_accumulate operation for reading the register values atomically. In this paper, we proposed two non-blocking atomic snapshot algorithms. One algorithm for one snapshot and second algorithm for new and old snapshot.

KEYWORDS

Non-Blocking Atomic Snapshot, MPI RMA, Shared Memory, CAS

1 Introduction

An Atomic snapshot contains two operations update and scan [1]. In update processor writes the content to associated location and scan gives the linearizable view of all n segments [1]. Linearizability [4] is the fundamental requirement for designing the concurrent data structure. Past two decades authors expressed their view on atomic snapshot but mostly theoretical [2,3] work. In this paper we present the non-blocking atomic snapshot in MPI RMA model. In atomic snapshot memory, memory is partitioned into n parts, each partition for one processor, where each processor can write and all other processors can read the latest updated values. In the term of consistency guarantees 3 types of registers: Safe, Regular and Atomic [5].

In this algorithm, we used MPI_Accumulate and MPI_Compare_and_swap which are atomic operations. MPI_Accumulate performs the atomic update and in this algorithm, we used MPI_Op - MPI_REPLACE in MPI_Accumulate. In fig. 1, one-sided communication in MPI RMA processes can read and write on remote memory. In this paper, we design the single-writer multi-reader snapshot algorithm for MPI RMA model. Each process can update the value on the shared memory buffer also known as window buffer at only one position according to its rank as shown in fig. 2 and all processes can read this window buffer atomically. For update operation we used the MPI_Accumulate and MPI_Compare_and_swap atomic operations. MPI_Accumulate provides atomic read-and-update operations.

In MPI RMA two types of synchronization calls: 1. Active target communication, 2. Passive target communication. In this paper, we used Active target communication synchronization call. In active target communication when data is moved from one process to another, both processes are involved in the communication [6].

For synchronization we used MPI_Win_fence. MPI_Win_fence is the simplest Active target synchronization. In Window all processes collectively call fence for synchronization.

Window buffer is shared array for all processes. In update operation each process updates its value according to window buffer position, if process 0 wants to update the window buffer array, then it will update at 0^{th} position of the window buffer as shown in Fig. 2 and for process 1 window buffer's 1^{st} position and so on until all processes finished update operation.

Window buffer (W_b[0,1,2, …,n]):

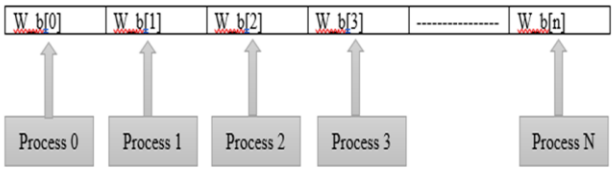


Figure 1: Update by Processes

Figure 2: Scan by Processes

2 Non-blocking Atomic Snapshot Algorithm for one Snapshot

update(rank,size, window_buffer1, window_buffer2,window)

begin

1. Integer: update=rank;
2. for i=0 to size-1
3. do
4. MPI_Accumulate(update , i , update*sizeof(int) , MPI_REPLACE, window);
5. If(window_buffer2[i] != window_buffer1[i])
6. MPI_Compare_and_swap(window_buffer2[i], window_buffer1[i]);
7. od;

end update;

scan(rank, size, window_buffer1, window_buffer2,window)

begin

1. for i=0 to size-1
2. do
3. MPI_Get_accumulate(window_buffer2[i], rank, i*sizeof(int), MPI_NO_OP, window);
4. od;

end scan

3 Non-blocking Atomic Snapshot Algorithm for old and new Snapshot

update(rank,size,window_buffer1,new_snap,old_snap,window)

begin

1. Integer: update=rank;
2. for i=0 to size-1
3. do
4. MPI_Fetch_and_op(&new_snap[i],&old_snap,MPI_INT,rank,i*sizeof(int),MPI_REPLACE,window);
5. MPI_Accumulate(update , i , update*sizeof(int) , MPI_REPLACE, window);
6. If(new_snap[i] != window_buffer1[i])
7. MPI_Compare_and_swap(new_snap[i], window_buffer1[i]);
8. od;

end update;

scan(rank, size, new_snap, old_snap, window)

begin

1. for i=0 to size-1
2. do
3. MPI_Get_accumulate(old_snap[i], rank, i*sizeof(int), MPI_NO_OP, window);
4. MPI_Get_accumulate(new_snap[i], rank, i*sizeof(int), MPI_NO_OP, window);
5. od;

end scan

here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here.

4 Comparison of snapshot algorithms

Table 1– Comparison According to Primitive used

Snapshot Algorithm	Primitive used
Lock Free	r/w register
Block Update	r/w register
Anderson	r/w register
Afek et al	r/w register
Aspnes and Herlihy	r/w register
Dwork et al	r/w register
Chandra Dwork	LL/SC
Rachman	Dyn. T&S
Attiya and Herlihy	T&S
Attiya and Rachman	r/w register
Kirousis et al	r/w register
Coordinated collect	LL/SC
Non-blocking Atomic Snapshot …	CAS

5 Benchmark

For testing these algorithms, we used HPE Cray EX Supercomputer and test the results of scan and update operations for different nodes.

Table 1 — Execution Time for update on 2 Nodes

No of Updates	**No of Nodes**	**Execution Time(seconds)**
512	2	50.2962
256	2	12.4448
128	2	2.8598
64	2	0.7223
32	2	0.2517
16	2	0.0805
8	2	0.0400
2	2	0.0222

As we can see in the table 1 when we used 2 updates on 2 nodes it's pretty fast. For this it took only 0.0222 s but when we increase the no of updates to 8 then it took more time as compare to 2 updates.

And when we increase the updates from 64 to 128 then execution time increase approximate 4 times.
But when we increase the updates 512 then on 2 nodes it so time consuming as we can see 50.2962 s. So now we need more nodes in parallel to reduce this execution time.

Table 2 — Execution Time for 128 Scans

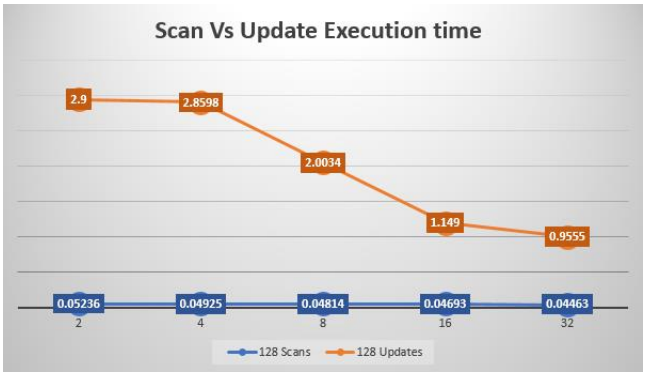
No of Scans	**No of Nodes**	**Execution Time(seconds)**
128	2	0.05236
128	4	0.04925
128	8	0.04814
128	16	0.04693
128	32	0.04463

Table 3 — Execution Time for 128 updates

No of Updates	**No of Nodes**	**Execution Time(seconds)**
128	2	2.9000
128	4	2.8598
128	8	2.0034
128	16	1.1490
128	32	0.9555

Table 4 —Execution Time for 256 updates

No of Updates	**No of Nodes**	**Execution Time(seconds)**
256	2	12.4448
256	4	11.6189
256	8	7.6956
256	16	4.2775
256	32	2.6203



Conclusion

In this paper we present non-blocking atomic snapshot algorithms for High Performance Computing in MPI RMA model. We designed the single-writer multi-reader algorithm where each process can update the assigned position for example Process 0 can update 0^{th} position of shared memory and Process 1 can update 1^{st} position of shared memory and so on. But all processes can read any position of the shared memory without locking. Using 2^{nd} algorithm, we can take the two non-blocking atomic snapshot as mentioned in algorithm old snap and new snap of the shared window buffer. In future work we will design the non-blocking multi-writer multi-reader atomic snapshot for High-Performance Computing in MPI RMA.

REFERENCES

[1] Riany, Yaron, Nir Shavit, and Dan Touitou. "Towards a practical snapshot algorithm." Theoretical Computer Science 269.1-2 (2001): 163-201.

[2] Afek, Yehuda, et al. "Atomic snapshots of shared memory." *Journal of the ACM (JACM)* 40.4 (1993): 873-890.

[3] Abrahamson, Karl. "On achieving consensus using a shared memory." Proceedings of the seventh annual ACM Symposium on Principles of distributed computing. 1988.

[4] Herlihy, Maurice P., and Jeannette M. Wing. "Linearizability: A correctness condition for concurrent objects." ACM Transactions on Programming Languages and Systems (TOPLAS) 12.3 (1990): 463-492.

[5] Israeli, Amos, and Amnon Shaham. "Optimal multi-writer multi-reader atomic register." Proceedings of the eleventh annual ACM symposium on Principles of distributed computing. 1992.

[6] MPI: A Message-Passing Interface Standard Version 4.0: page 593.