Design and Implementation of a Novel Message Logging Protocol for OpenFOAM

Xunyun Liu, Xiaoguang Ren, Yuhua Tang and Xinhai Xu
State Key Laboratory of High Performance Computing
National University of Defense Technology, Hunan, China 410073
Email: {liuxunyun, renxiaoguang, yhtang, xuxinhai}@nudt.edu.cn

***Abstract*—OpenFOAM is a free, open source CFD software package and has a large user base across most areas of engineering and science. Unfortunately, OpenFOAM itself faces severe reliability problems when running on the high performance computing platforms since the mean time between failures in such systems has become quite small. The existing fault-tolerance method in OpenFOAM typically tolerate fail-stop failures under the stop-and-rollback scheme, where even though there is only one processor failure, the whole system has to stop and roll back to the latest checkpoints, which indicates that the reliability has limited the scalability of parallel simulations on OpenFOAM. Inspired by the traditional sender-based message logging, we propose in this paper a novel message logging protocol which is seamlessly integrated in the framework of OpenFOAM. The proposed approach makes use of the snapshots of OpenFOAM as checkpoints and disables event logging mechanism completely due to the specific communication pattern of OpenFOAM. When a failure occurs during the execution, we do not stop the whole system, instead, we replace the failed process with the spawned substitution process and recover it by resending logged messages. We implement the protocol in Open MPI and evaluate it by molecular dynamics simulations on a subsystem of Tianhe-1A. Experimental results outline the advantage of our protocol on failure free performance and recovery time reduction.**

Keywords*—*message logging; fault tolerance; OpenFOAM

I. Introduction

OpenFOAM (Open Field Operation and Manipulation) is an object oriented numerical simulation toolkit for Computational Fluid Dynamics (CFD) and structural analysis [1], the prefix "Open" indicates the openness both in terms of source code and in its structure and hierarchical design. As a flexible set of C++ written modules, OpenFOAM builds various solvers, utilities and libraries, to simulate specific problems in continuum mechanics from fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetics [2]. OpenFOAM also provides an implicit, pressure-velocity, iterative solution framework, therefore users can implement physical models efficiently and flexibly by mimicking the forms of partial differential equations at the topmost hierarchy of the software, and those partial differential equations will be automatically solved by employing finite volume and finite element discretization on any structured or unstructured mesh [1]. With the help of the hierarchical framework, users can be freed from low level implementation details, such as numerical algorithms and parallel specific coding, and concentrate on the physical models [3]. In general, due to its wieldy, extendable and free-to-use features, OpenFOAM have been broadly-accepted by academia and industry.

However, OpenFOAM itself faces severe reliability challenge when aiming to achieve scalable performance. Since OpenFOAM is developed to solve enormous and complex scientific computation problems, it is usually running at the High Performance Computing (HPC) systems. Unfortunately those supercomputers have become less reliable as they grow in size [4], even under the most optimistic estimates, the mean time between failures in the next generation of exascale supercomputers will be reduced to a few hours [5]. This illustrates that OpenFOAM must handle a continuous stream of faults/errors/failures occurred in such HPC systems, we need to investigate efficient and reliable fault tolerance mechanisms and integrate them into OpenFOAM.

Generally speaking, OpenFOAM builds parallel simulation applications with the Message Passing Interface (MPI), and fault tolerance for message passing applications is usually achieved by rollback-recovery protocols. Among them Checkpoint/Restart is a widely used approach because of its simplicity of implementation and recovery [6]. In fact, the existing fault tolerant method in OpenFOAM is a typical Checkpoint/Restart mechanism: it coordinates processes at checkpoint time to ensure that the saved global state is consistent, and enforces all the process to roll back to the most recent checkpoint due to a failure of one process. However, the existing method does not scale well because of the global restoration process. Message logging protocols present a promising alternative to Checkpoint/Restart, it consists of only forcing the re-execution of crashed processes from their last checkpoint images to reach the state immediately preceding the crashing state, while the other processors may keep making progress or wait for the recovering processor in a low-power state [7].

To the best of our knowledge, this is the first work investigating message logging protocol in OpenFOAM. The contributions of this paper are summarized below:

- We introduce the basic idea behind message logging, aiming to reduce the recovery consumption.
- We design the integrated framework and its implementation of message logging on OpenFOAM.
- We demonstrate the validation of the protocol by making experiments on molecular dynamics simulations with a subsystem of TH-1A. Experimental results prove that our protocol has the great advantage of recovery time reduction and has little or no impact on failure-free performance.

The rest of the paper is organized as follows. Section 2

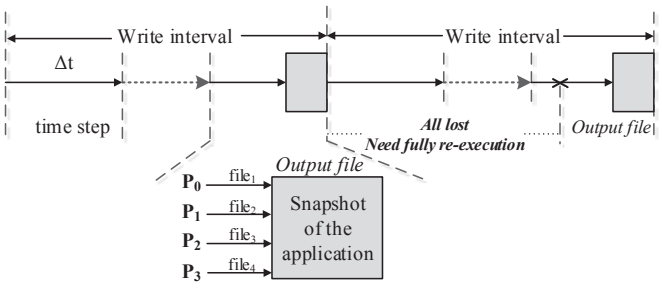

Fig. 1. The existing fault tolerance method in OpenFOAM, All the lost work need fully re-execution.

introduces the motivation and basic idea behind our message logging protocol. Section 3 describes the message logging protocol and its implementation in details. In section 4 we depict our evaluation methodology and demonstrate the superiority of our protocol over the existing fault-tolerance method by benchmarking. Finally, Section 5 concludes the paper.

II. Basic Idea

This section starts by analyzing the drawbacks of the existing fault tolerance method in OpenFOAM, then we introduce the basic idea behind our message logging protocol. As with most previous researches on message logging, we assume that the process execution is piecewise deterministic, and communications channels between processes are reliable and FIFO, Therefore, we will concentrate on the faults of computing processes with the assumption of fail-stop fault model.

A. The existing fault tolerance method in OpenFOAM

Since users often have requirements of visualizing the dynamic physical simulation procedure when finishing the computational process, OpenFOAM outputs the field data periodically during the execution, which is the so-called snapshot mechanism. As shown in Fig.1, we describe a parallel OpenFOAM program which is constituted by 4 processes, denoted by P_0, P_1, P_2 and P_3. OpenFOAM has defined the time step to solve the physical model iteratively, and it also introduces the concept of write interval to represent the number of time steps between two adjacent outputting procedures. When executing to the end of a write interval, each process outputs their field data to reliable storage respectively. Those output files compose the snapshot of the whole system, and the snapshot is enough sufficient to recover the computational process once a failure occurs.

However, despite of the simplicity of recovery, the drawbacks of the existing method is also significant. If the write interval is too long, all the operation efforts between the last file outputting point and the failure point will be lost, therefore the recovery program will spend as much time in re-executing. On the other hand, employing an overly short write interval will induce an unacceptable consumption of $I\backslash O$ bandwidth, which also limiting the practicality of fault tolerance.

B. Our message logging protocol

Our goal is to avoid the global restoration, thus we need a message logging protocol to help the substitution process to recover. In general, message logging must preserve message exchanges between application processes during failure free execution to be able to replay them in the same order after a failure, this is the so-called payload copy mechanism. Also, a classical message logging protocol needs the event logging mechanism to correct the inconsistencies induced by orphan messages and nondeterministic events, by adding the outcome of non-deterministic events to the recovery set. However, due to the specific communication pattern used in OpenFOAM (performing point-to-point communication with source and tag specified, and non-deterministic probe-matching statements are not used), event logging in our protocol can be safely disabled, which means the burden of message logging will be reduced drastically.

In this paper, we impose a message logging protocol at the parallel supporting hierarchy of OpenFOAM. The basic idea of our approach consists of 3 parts: 1. Each process records on-sending messages locally in its process memory; 2. Surviving processes detect the failures occurred on the communicator, and spawn a substitution process to replace the failure process; 3. Once the substitution is spawned, surviving processes resend the in-transit messages recorded in the message logger according to their transmission sequences. The word in-transit refers to messages which may be necessary for the progression of the recovered processes, but are not available anymore, as the corresponding send operation has been executed during failure-free procedure.

We illustrate our protocol by an example. As shown in Fig.2, $P_0 \sim P_3$ denote 4 processes executing the same OpenFOAM program in the previous section, and P_1 fails unexpectedly due to a process error. The failure recovery process is the procedure where surviving process spawn a substitution process P_1' and resend the in-transit messages, while the restart segment is introduced to make P_1' jump to the latest write interval point and get ready for the recovery procedure.

To be specific, at first the application executes normally, P_0 sends message m_1, m_2 to P_1 via MPI Send and then records them as message logs. Similarly,P_2 sends message m_3 to P_1. After finishing the reception of m_3 , P_1 fails unexpectedly, we can see how this happens on the middle part of the figure. P_3 detects the communicator failure when it is trying to send m_4 to P_1 , so it broadcasts the failure and moves into the failure recovery process automatically. During the recovery process, P_3 spawns P_1' as a substitution process, re-transmits the logged message m_4 to help P_1' recover. Meanwhile, P_0 and P_2 also turn into failure recovery process because of the failure notification, they resend the in-transit messages m_2 and m_3 to P_1'. After that, the surviving processes will jump out of the failure recovery process, continue execution or wait for any message from the other process. On the other side, P_1' takes the place of P_1 in the communicator, jumps to the checkpoint address, reads the live variables and enters the recovery procedure by means of the execution of the restart segment. During its recovery, all the external message P_1' needs will be resent by P_0, P_2 and P_3 with non-blocking send operations, and the recovery will consume much less time than the global re-execution.

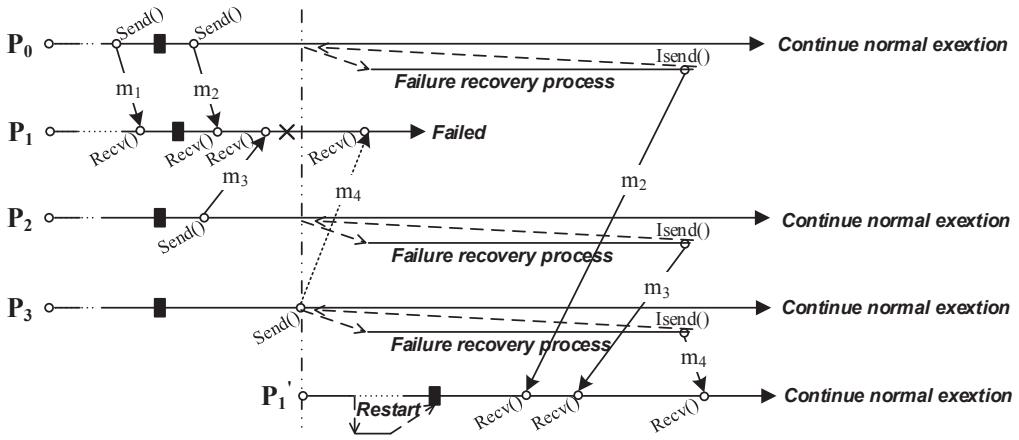

Fig. 2. An example execution of our message logging protocol. P_1' fails unexpectedly during the execution, we illustrate the procedure of spawning a substitution process and resending in-transit messages.

III. Message logging protocol in OpenFOAM

A. Data Structures

For each participating process, our message logging protocol requires the maintenance of the following items of system data.

- ***SSN*** (Send Sequence Number): SSN refers to the sequence number of a message sent by the process, In our protocol, messages sent to different destinations are preserved in different message log queues, thus each log queue maintains an independent send message sequence.
- ***HSR*** (Highest Sequence number Received): Each process holds a HSR array to record the highest message SSN received from the other processes, and different locations in the array correspond to different processes.
- ***HSD*** (Highest Sequence number Delivered): Each process also holds a HSD array to record the highest message SSN which has been delivered to the other processes successfully. It is worth noting that the HSD array will not be updated until a failure occurs, thus no acknowledge mechanism is required during the normal message exchanges, instead, it will be assigned at the beginning of the failure recovery protocol to suppress duplicate messages.
- ***Message log:*** a message log contains the entire message that was sent including the data, the identification of the destination process, the SSN and the MPI tag used for that message. After sending a message, its message log will be stored in the corresponding message log queue.

B. Forward Path Protocol

Assuming that process P_0 is going to send message m to process P_1 , the following steps are required to complete the transfer procedure and save application message in the senders memory.

1) P_0 determines whether the SSN used for the message m is smaller than the record stored in HSD. If so, the message m is a duplicate message and will be preserved without transfer; if not, continue the transfer process.
2) P_0 determines whether the destination of m is P_0 itself. If so, the message will be sent without recording, otherwise the message m will be preserved as a message log and then sent to the destination P_1.
3) after the reception of the message m, P_1 resolves the SSN carried in the message, then it update the HSR array by replacing the corresponding element with the received SSN. After that, the message passing procedure completes successfully.

C. Failure Recovery Protocol

When a failure is detected, the failed process is first restarted on an available processor from its most recent checkpoint. After that, the recovery protocol needs to consider two main activities, namely handling of lost messages and handing of duplicate messages.

Fig.3 illustrates the failure recovery protocol by an example. Assuming that a failure occurs during the execution procedure, the dashed line in grey represents the configuration of the entire application after the failed process P_1 has been reloaded from its checkpoint. We can see that they are 5 messages can cross the dashed line, among them m_1 and m_3 are called lost messages since they are still necessary for the recovery but are not available anymore. While m_2,m_4 and m_5 are referred to as duplicate messages because they are crossing the dashed line from future to the past. The failure recovery protocol must handle those message correctly to achieve a consistent system state.

For lost messages, all the process should figure out which messages have been properly delivered, thus an all-to-all operation is performed to build up the HSD arrays. Assuming that the notation for the j^{th} element in process i's HSR/HSD array is $HSR_i[j]/HSD_i[j]$, a simple formula reveals the relation between the HSR arrays and the HSD arrays at all processes.

$$HSR_i[j] = HSD_j[i] \qquad (i, j = 0, 1, ..., n-1)$$

After the formation of the HSD arrays, all the surviving processes should execute the algorithm of handling lost messages separately to resend the lost messages to the recovering process. If we assume that there are n processes running an OpenFOAM application, which are denoted by $P_0, P_1, ..., P_{n-1}$, and process f has been restarted from its latest checkpoint due to a process failure. Process $P_0, ..., P_{f-1}, P_{f+1}, ..., P_{n-1}$ need to perform the following algorithm.

Algorithm 1 Handling of lost messages

Initialization:

The process running the algorithm is denoted by P_t, its message log queues $logGue$ (the j^{th} message sent to P_i are logged in $logGue_i[j]$) and its HSD array HSD_t have been properly updated;

Traversal:

```
for i = 0; i < n − 1; i++ do
  if logGue_i.Lenth > HSD_t[i] + 1 then
    for all j ∈ logGue_i do
      if logGue_i[j].SSN > HSD_t[i] + 1 then
        MPI_Isend logGue_i[j] to P_f;
      end if
    end for
  end if
end for
```

Take the situation in Fig.3 as an example. As the HSR array of P_1 was been cleared because of the failure, the corresponding elements in other processes HSD arrays are set to 0. P_0 and P_2 knows that no message for P_1 has been properly delivered, and those lost messages will be resent in FIFO order by traversing their message loggers.

For duplicate messages, since the recovery process has maintained the SSN count in its HSD array, when it tries to replay any messages whose SSN count is less than or equal to the record in HSD, the message will be rejected as a duplicate in forward path protocol. Those duplicate messages need only to be preserved by the recovery process to rebuild the message logger.

D. Implementation in OpenFOAM

OpenFOAM is developed natively on Linux/UNIX platforms and usually ran at the High Performance Computing systems. Thus an MPI C++ compiler is required to build application, utilities and the other modules in OpenFOAM. We choose the Open MPI to implement our message logging framework and provide the parallelism support, for the reason that it is a typical example of the new generation MPI implementations, and it is developed in a true open source fashion. Our message logging framework will not affect the original OpenFOAM execution process like mesh generation, decomposition, numerical simulation and post-processing, instead, it encapsulates the default MPI communication functions to integrate message logging fault tolerance capabilities into OpenFOAM. Fig.4 shows the implementation of OpenFOAM in a Linux/UNIX environment, our message logging framework components are dashed.

Our implementation consists of 3 parts. First we modify the MPI communication primitives (e.g. MPI_Send/MPI_Recv)

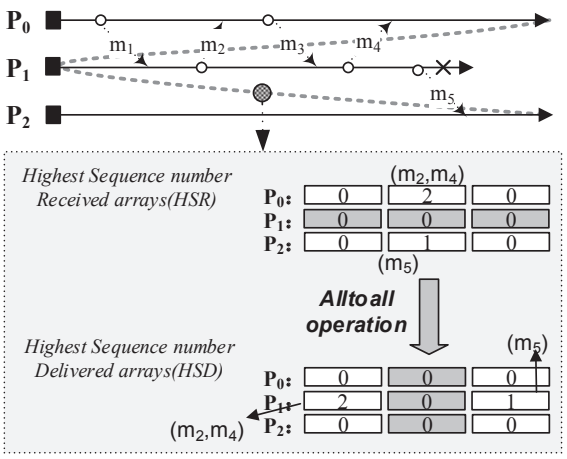

Fig. 3. An example execution of the failure recovery protocol. The upper part shows the communications procedure, the lower part shows the exchange process of the HSR arrays and the HSD arrays.

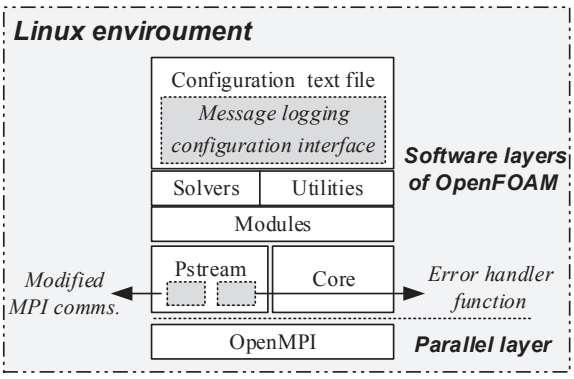

Fig. 4. The OpenFOAM implementation hierarchy chart, our message logging framework components are dashed and displayed in gray.

which are located at the $src\backslash Pstream\backslash Mpi$ directory of OpenFOAM, each of the modified functions is an implementation of a particular fault tolerant algorithm, and its goal is to extend the communication with message logging features and apply forward path protocol. After that, our error handler function is added to introduce the failure recovery protocol, the error handler function is called every time an MPI error is detected within the communicator. By replacing the predefined default error handler (*MPI_ERRORS_ARE_FATAL*), we can spawn the substitution process and recover it with redelivered messages, instead of aborting the whole parallel program. Finally, we offer the configuring interface to let users set the checkpoint interval, garbage collection interval and choose whether to enable the message logging features or not.

IV. EXPERIMENTS

In this section, we evaluate several different molecular dynamics simulations to understand the behavior of the protocol under different circumstances. We focus on the research of fault-free performance and fault recovery performance, in order to assess the superiority of our message logging protocol.

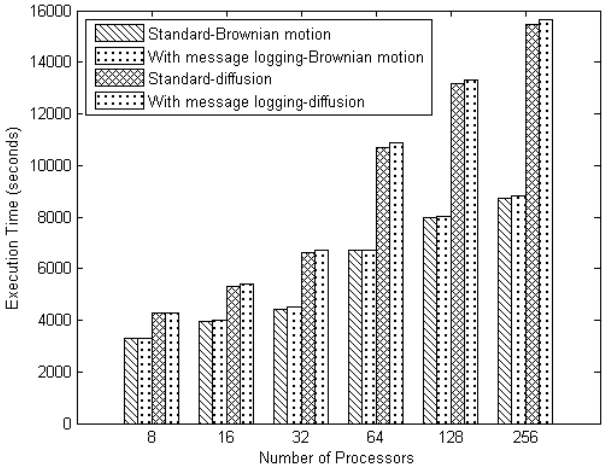

Fig. 5. Fault free performance evaluation on Brownian motion and the diffusion simulation.

A. Experiment Condition and Methodology

Our computer platform is a subsystem of Tianhe-1A, the system configuration and interconnection is the same as described in [8, 9], and the results presented are mean values over 5 executions of each test.

We choose molecular dynamics simulations as test cases, for the reason that the communication pattern used is quite simple. Due to its specific math-physical model, molecular dynamics simulation does not require the global reduction operation at the end of a iteration, which means a failure-free process should be able to keep making progress until it is waiting for messages from the failed process. But the existing fault-tolerance method in OpenFOAM forbids the surviving process to continue, it just rolls back all the processes to the latest checkpoints. Thus the molecular dynamics simulation should benefit from our message logging protocol greatly.

B. Fault Free Performance Evaluation

Suppose that there is a cuboid filled up with molecules, we use OpenFOAM to simulate the Brownian motion of those molecules in different parallelism degrees. First we set invariable parameters as follows: the temperature is $298K$, density is $1004kg/m^3$, cutoff is $10^{-9}m$ and each process hold an average of 8000 molecules. Then the physical model is simulated for 2000 time steps, and the result is illustrated in Fig.5. It is worth noting that our message logging protocol has only slightly impact on failure-free performance compared to the standard execution without fault-tolerance, and the overhead comes mainly from preserving message logs into the sender's volatile memory.

After that, we simulate the diffusion process under the same assumption. Initializing that all the molecules are located at one side of the cuboid, the physical model shows the spread procedure of molecules. The result in Fig.5 also proves that our message logging protocol is suitable for different communication patterns.

C. Recovery Performance Evaluation

We simulate a fault on a processor by sending SIGKILL to a process running on it, and the Brownian motion simulation running on 16 processes is chosen as our test case. First we checkpoint the system at the time step 1, then introduce a failure to process 3 at the time step 1000. Table 1 presents the elapsed wall clock time and CPU time consumed to recover the system. We find that the message logging protocol reduces the wall clock time by 17.5% and saving CPU time by 15.8%.

TABLE I. COMPARISON OF RECOVERY TIME-CONSUMPTION (SECONDS)

	Existing fault-tolerance method		Message logging	
	Failure free	Failure occurred	Failure free	Failure occurred
Wall time	3276	4951	3296	4083
CPU time	57416	74624	57447	62799

V. CONCLUSION

In this article, we introduce a novel message logging protocol to OpenFOAM. Compared to the former message logging researches, our protocol and its implementation have been seamlessly integrated into the software hierarchy of OpenFOAM, and the burden of event logging mechanism has been released. Also, our protocol exploits the snapshots of OpenFOAM as the checkpoints of message logging, thus the overhead of preserving checkpoints can be concealed completely. Moreover, our experimental results obtained on MD simulations demonstrate the validity and superiority of our method no matter whether errors occur or not. The new methodology proposed is simple yet effective, and it can be easily used by physicists who have no fault-tolerance relevant knowledge. To sum up, our work facilitates the adoption of message logging in computational fluid dynamics fields.

ACKNOWLEDGMENT

This work is supported by the funds (No.124200011) from Guangzhou Science and Information Technology Bureau, and the National Natural Science Foundation of China under Grant No.60921062 and 61120106005.

REFERENCES

[1] H. Jasak, A. Jemcov, and Z. Tukovic, "Openfoam: A c++ library for complex physics simulations," in *International Workshop on Coupled Methods in Numerical Dynamics, IUC, Dubrovnik, Croatia*, 2007, pp. 1–20.

[2] M. Beaudoin and H. Jasak, "Development of a generalized grid interface for turbomachinery simulations with openfoam," in *open Source CFD International Conference. Berlin, Germany*, 2008, pp. 4–5.

[3] C. Kunkelmann and P. Stephan, "Cfd simulation of boiling flows using the volume-of-fluid method within openfoam," *Numerical Heat Transfer, Part A: Applications*, vol. 56, no. 8, pp. 631–646, 2009.

[4] X. Yang, Z. Wang, J. Xue, and Y. Zhou, "The reliability wall for exascale supercomputing," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 767–779, Jun. 2012.

[5] A. Bouteiller, T. Herault, G. Bosilca, and J. J. Dongarra, "Correlated set coordination in fault tolerant message logging protocols," in *Proceedings of the 17th international conference on Parallel processing - Volume Part II*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 51–64.

[6] S. Chakravorty and L. Kale, "A fault tolerance protocol with fast fault recovery," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–10.

[7] E. Meneses, X. Ni, and L. Kale, "A message-logging protocol for multicore systems," in *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, 2012, pp. 1–6.

[8] X.-J. Yang, X.-K. Liao, K. Lu, Q.-F. Hu, J.-Q. Song, and J.-S. Su, "The tianhe-1a supercomputer: Its hardware and software," *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 344–351, 2011.

[9] X. Xu, X. Yang, and Y. Lin, "Wbc-alc: A weak blocking coordinated application-level checkpointing for mpi programs," *IEICE Transactions*, pp. 786–796, 2012.