Fault Tolerance for OpenSHMEM

Pengfei Hao
University of Houston
phao@uh.edu

Pavel Shamis
Oak Ridge National Laboratory
shamisp@ornl.gov

Manjunath Gorentla Venkata
Oak Ridge National Laboratory
manjugv@ornl.gov

Swaroop Pophale
University of Houston
spophale@uh.edu

Aaron Welch
University of Houston
dawelch@uh.edu

Stephen Poole
Oak Ridge National Laboratory
spoole@ornl.gov

Barbara Chapman
University of Houston
bchapman@uh.edu

ABSTRACT

On today's supercomputing systems, faults are becoming a norm rather than an exception. Given the complexity required for achieving expected scalability and performance on future systems, this situation is expected to become worse. The systems are expected to function in a nearly constant presence of faults. To be productive on these systems, programming models will require both hardware and software to be resilient to faults. With the growing importance of PGAS programming model and OpenSHMEM, as a part of HPC software stack, a lack of a fault tolerance model may become a liability for its users. Towards this end, in this paper, we discuss the viability of using checkpoint/restart as a fault-tolerance method for OpenSHMEM, propose a selective checkpoint/restart fault-tolerance model, and discuss challenges associated with implementing the proposed model.

Categories and Subject Descriptors

C.2.6 [**COMPUTER-COMMUNICATION NETWORKS**]: Internetworking—*Standards*

1. INTRODUCTION

Modern high-performance computing (HPC) systems consist of an increasing number of computing resources, not only CPU cores but also hybrid architectures with GPUs and FPGAs. For example, the Oak Ridge Leadership Computing Facility (OLCF) hosts the TITAN supercomputer, one of the largest HPC systems in the world, which contains 18,688 nodes, a total of 299,008 CPU cores, and 18,688 Nvidia Tesla K20 GPUs (each of which having 2496 CUDA cores). In addition, HPC programs are also increasing in size and complexity and often perform high volume data processing; such applications require highly reliable systems with low failure rates. Achieving continuous running time without failures is a major issue on such large-scale systems, due to the superposition of failure possibilities across the computing resources.

Moreover, the Mean Time To Failure (MTTF) of individual components of the HPC system is not expected to increase. This assumption is based on a number of facts. First, the complexity, density, and sensitivity of individual components increases over time; hardware vendors increase the density of silicon chips, increase the number of transistors, and decrease power consumption of the components. This leads to a lower MTTF of system components. Second, according to the literature[3], the reliability of components in HPC systems has not improved in the last ten years. Since the reliability of individual components is not expected to increase while the number of components grows, the overall MTTF of the whole system is projected to decrease. Based on the above facts, there is a consensus among the HPC community that the future generation (Exascale) of systems will operate with a constant presence of faults.

The OpenSHMEM specification[4] currently lacks fault mitigation features, making OpenSHMEM programs vulnerable to system failures. The primary contribution of the this paper is to propose of a fault tolerance model in the OpenSHMEM specification. Although our discussion is focus on OpenSHMEM only, the proposed model can be applied to other libraries and languages of the PGAS programming model.

2. BACKGROUND

2.1 Related Work

Research in the area of fault tolerance for the PGAS programming model is sparse. Ali, et al.[1] proposed an application-specific fault tolerance mechanism. They achieved fault-tolerance using redundant communication and shadow copies. They evaluated the approach by implementing it as a part of Global Arrays, a shared-memory programming interface for distributed systems, and using NWChem [7], a framework specifically for computational chemistry problems. Our approach is more encompassing and not limited to a specific application or kernel.

Other researchers have proposed fault-aware and fault-tolerant models for message passing models, particularly MPI. Bland et al. have proposed and evaluated user level failure mitigation extensions to MPI. This provides an interface for han-

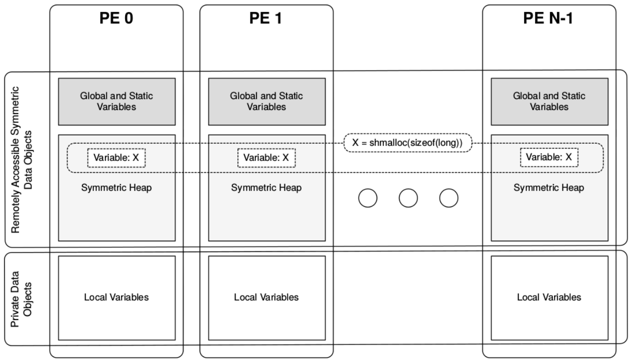
PE 0
PE 1
PE N-1
Remotely Accessible Symmetric Data Objects
Global and Static Variables
Variable: X
Symmetric Heap
X = shmalloc(sizeof(long))
Private Data Objects
Local Variables

Figure 1: OpenSHMEM Memory Model [4]

dling faults by the user (HPC application) without aborting the entire MPI job [2][3]. Graham et al. [5] have proposed similar models and evaluated MPI collective algorithms in the presence of faults [6].

In this paper, we explore the best practices in the fault tolerance domain to propose a solution addressing the fault tolerance challenge for OpenSHMEM.

2.2 OpenSHMEM Memory Model

OpenSHMEM is a PGAS library interface specification that provides an interpretation of the PGAS model. In OpenSHMEM, a Processing Element (PE) is an execution context which has access to local private memory and global memory. A partition of global memory is mapped to each PE. The PEs collectively manage the global memory. A PE can access the memory on remote PEs using the OpenSHMEM interfaces.

Figure 1 presents a programmer view of OpenSHMEM memory model. In OpenSHMEM the following kinds of data objects are symmetric:

- Fortran data objects in common blocks or with the SAVE attribute. These data objects must not be defined in a dynamic shared object (DSO).
- Global and static C and C++ variables. These data objects must not be defined in a DSO.
- Fortran arrays allocated with *shpalloc*
- C and C++ data allocated by *shmalloc*

3. FAULT TOLERANCE MODEL

3.1 Model introduction

Fault tolerance is the ability for a system to continue operating in the event of faults in its components. As a starting point we propose the checkpoint/restart mechanism for OpenSHMEM. We checkpoint the system state periodically and restart from the most recent checkpoint on detecting a fault.

Instead of saving the entire memory associated with a PE, we save only the symmetric memory data objects. We make this design choice because the symmetric data objects are accessible through the OpenSHMEM API by all PEs, so, most relevant information that needs to be communicated is stored there. Typically, distributed algorithms use this memory region to coordinate execution and share information.

In our model, we propagate the fault causing errors to the application (user), which means that the user needs to modify the application code to take advantage fault-tolerance capabilities.

3.2 Fault-tolerant Interfaces

In order to introduce explicit checkpoint functionality, we introduce the following interface:

int shmem_checkpoint_all(void)

This is a collective checkpoint operation requiring participation of all PEs. For each PE, the operation creates a shadow copy of the symmetric memory region on another PE (Figure 2). The operation gathers fault notifications that are inquired through the runtime environment. If any fault event was signaled between current and the previous checkpoint, the function returns a special error code.

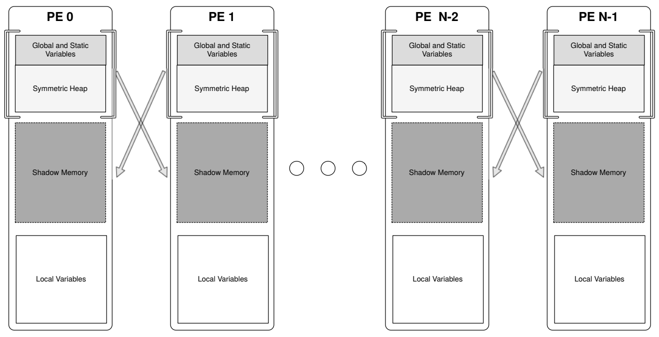

Figure 2: Shadow region for backup symmetric memory

If a fault occurs, then an error event is triggered, the application may use the following function in order to inquire details about the failure:

*shmem_query_fault(int **pes, int **pes_status, size_t *numpes)*

Once the information about the failed PEs is gathered, the application may use the following restart function:

*int shmem_restart_pes(int *pes, size_t numpes)*

This operation spawns new PEs using the underlying runtime environment.

The memory restore (rollback) interface is provided through the following interface:

int shmem_restore_all(void)

This function is a coordinated rollback operation that restores the symmetric memory region from the last checkpoint on all PEs. Listing. 1 is the pseudo that executes steps of a computation algorithm in a loop. This demonstrates how the proposed model is intended to be used by applications.

Listing 1: Pseudo Code

```
int step=0;
do{
   if(shmem_checkpoint_all()){
      /* Failure event was reported */
      int *pes;
      int *status;
      size_t num_pes;
      /* Query indexes of failed PEs */
      shmem_query_fault(&pes, &status, &
         num_pes);
      /* Restart failed PEs */
      shmem_restart_pes(pes, num_pes);
      /* Restore memory */
      shmem_restore_all();
      /* Reset the algorithm one step back
         */
      step--;
   }
   execute_computation(step);
   step++;
}while(step<MAX_STEPS);
```

3.3 Different restore options

Because this fault tolerance model is based on copying symmetric data objects on a partner PE, a node failure not only results in loss of local data values, but also the loss of backed up data stored in the shadow region. To tackle this situation, we give two options for the user to choose from, depending on the application characteristics.

The first scheme makes two copies of the symmetric memory - that is, along with the copy on a PE on a remote node, a local copy is made. If there is a roll back, PEs can use the local backup stored at the last checkpoint. Only PEs on the failed node need to do a copy from their partner PE. There is a trade-off between recovery time and memory usage while using this scheme. The effective memory usage is only 33%, but it could provide a speedy restore for PEs nodes that have not failed due to the local copy.

The second scheme is to ignore old data at failure and use the newest data to continue the algorithm. This is not a fully functional checkpoint solution because not all PEs can roll back to the previous checkpoint instance. Some algorithms, for example, iterative methods, can use this with no problem. The advantage of this scheme is that it allows for higher memory usage rate (50%).

4. CONCLUSION

As computational systems and application execution time grows, fault tolerance is going to become an important aspect for the HPC community. Since no prior research has been done for fault tolerance in OpenSHMEM, this paper aims at bringing fault tolerance to the attention of the OpenSHMEM community.
The OpenSHMEM memory model clearly distinguishes between private and globally accessible memory which allows us to design an efficient checkpoint/restart strategy. Our design for OpenSHMEM's fault tolerance model is based on the selective-in-memory checkpoint/restart algorithm combined with an application level fault notification mechanism. Moreover, since most PGAS languages and libraries are based on a similar memory model, the same design can be applied to them with minor changes.

5. ACKNOWLEDGMENTS

This work is supported by the United States Department of Defense and used resources of the Extreme Scale Systems Center located at the Oak Ridge National Laboratory.

6. REFERENCES

[1] N. Ali, S. Krishnamoorthy, N. Govind, and B. Palmer. A redundant communication approach to scalable fault tolerance in pgas programming models. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 24–31. IEEE, 2011.

[2] W. Bland, G. Bosilca, A. Bouteiller, T. Herault, and J. Dongarra. A proposal for user-level failure mitigation in the mpi-3 standard. *Department of Electrical Engineering and Computer Science, University of Tennessee*, 2012.

[3] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra. *An evaluation of user-level failure mitigation support in MPI*. Springer, 2012.

[4] T. Curtis and et al. Openshmem application programming interface, 2014.

[5] G. E. Fagg and J. J. Dongarra. Building and using a fault-tolerant mpi implementation. *International Journal of High Performance Computing Applications*, 18(3):353–361, 2004.

[6] J. Hursey and R. L. Graham. Preserving collective performance across process failure for a fault tolerant mpi. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1208–1215. IEEE, 2011.

[7] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. L. Windus, et al. Nwchem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010.