# Supporting Online Code Changes in Parallel Applications

Dong Kwan Kim

***Abstract*—** Dynamic software updating is one of the interesting software practices in terms of alleviating service interruption and performance degradation. It is obvious that dynamic updating can contribute to performance improvement in high-end computing. This paper presents useful guidelines and considerations when dynamic software updating is applied for parallel applications. According to the preliminary findings, dynamic software updates can be effectively used in parallel programs.

***Keywords*—**Dynamic software updating, DSU, Parallel application, Software change

## I. Introduction

PERFORMANCE improvement is one of the major concerns in parallel computing. Parallel computing has provided a solution for faster calculation in science and engineering applications that need a massive computation. In recent years, big data processing and mobile cloud computing are expanding into a new research area in parallel computing.

In the software development perspective, similar research has been conducted on how to improve the computation performance of a running program. Adaptive software makes it possible to fix program errors or add new features at runtime. Adaptive software can be implemented by dynamic software updating. Dynamic software updating can be applied for mobile computing and parallel computing as well as server applications.

The rest of this paper is structured as follows. Section 2 presents the background and related work on dynamic software updating. Section 3 describes the proposed guidelines and considerations for parallel dynamic updating. The preliminary experimental result is described in Section 4. Section 5 presents concluding remarks.

## II. Background

The goal of dynamic software updating is to change part or whole of a running program, thereby enabling developers to alter its behavior at runtime.

Figure 1 depicts how different static and dynamic updates are. Static updates in the upper part of Figure 1 are typical code maintenance which involves three tasks: system stop, code change, and system restart. When a parallel program is required to change at time t2, developers should stop the program and then fix or modify some part of the program. Obviously, the parallel program cannot continue its services or computation during updating. Therefore, such updates can result in service interruption and performance degradation. In contrast to the static update, dynamic updates are more effective and efficient in terms of saving computation time and providing seamless service. The lower part of Figure 1 shows that dynamic software updating helps the program complete a scientific computation at time t3.

Considerable research has been conducted on how to change a running program at runtime. In controlled conditions, some software systems are configurable and adaptable to a changing operation environment [1]. They are dependent on a special compiler, middleware, or runtime module.

As C, C++, and Java are getting more popular, DSU systems have been developed by utilizing prior research findings. The Rubah [2] system updates Java programs at runtime. It supports whole program updates and uses bytecode rewriting on a standard JVM. It also allows nearly arbitrary class changes. Unlike the Rubah system, the Jvolve [3] system relies on a dedicated JVM to implement the dynamic update service. In addition to these systems, the standard JVM provides a dynamic update facility called HotSwap with some limitations.

There are also many updating systems for C applications. The Ginseng system [4] realizes dynamic updates by adding patch files that contains differences between an old and new version. It is composed of a dedicated compiler, a patch generator, and a runtime system that links a patch file to a running program. In addition, POLUS [5] can modify binary code, not source code directly and the UpStare system [6] can change multi-threaded programs using stack reconstruction. Dynamic updates can be extended to wider application domains such as operating systems [7], database schema evolution, and virtualization.

Dong Kwan Kim is with the Mokpo National Maritime University, Mokpo-si, Jeollanam-do 530-729 South Korea (corresponding author's phone: +82-61-240-7271; e-mail: dongkwan@gmail.com ).

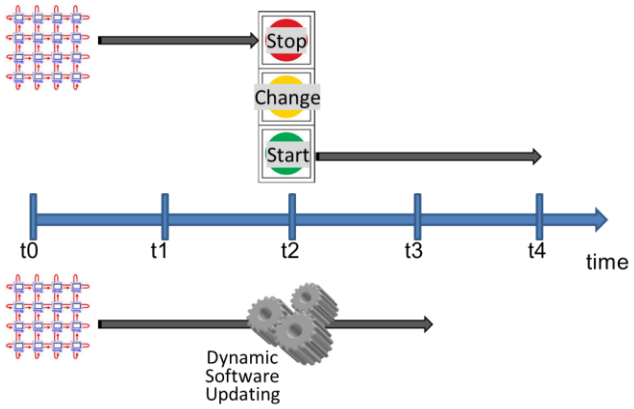


Fig. 1 Dynamic software updating

## III. Parallel Dynamic Update Systems

Figure 2 shows the structure of the proposed parallel dynamic software update system. The parallel DSU System consists of update preprocessor, code transformer, utility, update synchronizer, update point extractor, update recovery manager, and update runtime system. According to users' update requests, this update system changes a parallel application that is running on a computer cluster.

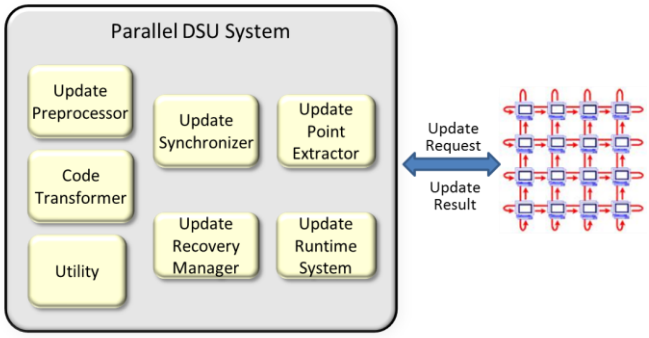


Fig. 2 Parallel dynamic update system

The update point extractor should select proper update positions in the program in order to perform safe code updates. Even if an external update request is arrived, the update will be delated until the program execution reaches the safe update point. Therefore, the update points affect the performance of the update system. The update system maintains the balance between safety and timeliness of the update.

Since multiple nodes participate in an update, the update synchronization is required to maintain the overall system-wide consistency. Without the synchronization mechanism, update nodes cannot communicate with non-updated nodes. The update synchronizer in Figure 2 deals with the consistency of the parallel application. Figure 3 depicts the synchronized updates for parallel applications. All participating processes perform their own jobs at the same time. When a user requests an update, the update manager in the DSU system notifies each worker process to start updating. All worker processes send back a signal to the update manager when they finish the update. Each worker process can resume its prior computation after all updates are completed.

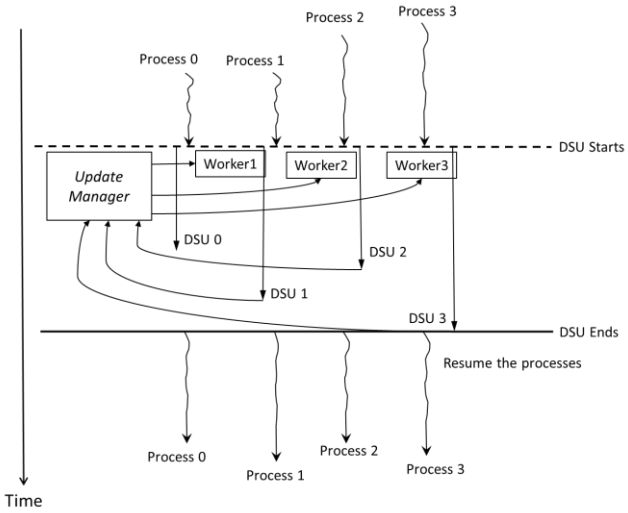


Fig. 3 Synchronizing parallel dynamic updates

The update system cannot avoid performance overhead of the target application. However, it should impose essentially little overhead due to updates. In particular, since parallel applications should complete a given computation as fast as possible, the computation performance cannot be compromised. The code transformer can insert an update code to the target application for the safe update. Such update code should not affect the functionality of the original code.

Even if updates have been applied, the behavior of the updated application may be working as unexpected for some reasons. In case of unexpected updates, the update recovery manager can nullify the applied updates. It enables the application to stay in a correct state by replacing the new version of the application with the old one. Consequently, the application rolls back to the state before updating.

## IV. Preliminary Results

Parallel applications often run on computer clusters where computer nodes are connected through a variety of network devices. A job scheduler in the cluster controls the execution order of jobs from cluster users. The initial version of a dynamic updating system has been implemented which follows the part of the proposed guidelines. Obviously, this system runs on a computer cluster and is controlled by its job scheduler. Simple but typical MPI applications have been updated at runtime including matrix multiplication and prime number calculator. The experimental results show the performance overhead imposed by dynamic updates could be acceptable. We can also use a computer cluster which consists of single-board computers such as Raspberry Pis [8]. The Raspberry Pi cluster can be built at a low cost and take up less space. Prior experiments show that the Raspberry Pi cluster can be a test bed for parallel applications. Dynamic update systems are dependent on programming languages and operating systems because they should change running applications loaded into memory. MPI programs running on the Linux operation system can use system library calls such as dlopen, dlclose, and dlsym.

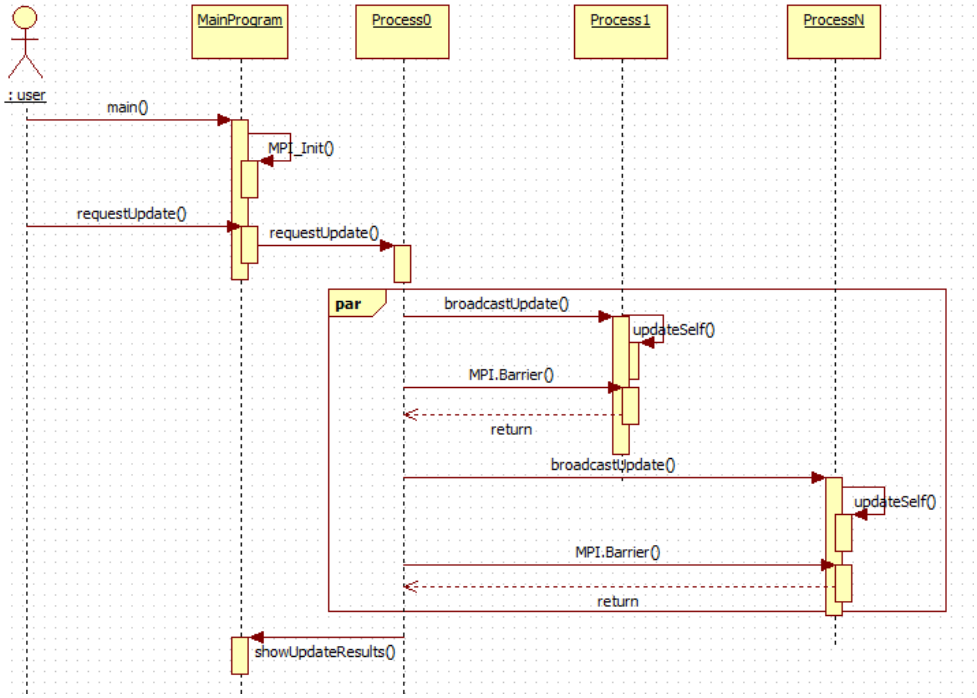


Fig. 4 The sequence diagram for dynamic updating

Figure 4 illustrates the flow of applying dynamic updating for parallel applications. The user starts the main function of the target program and then the MPI execution environment is initialized. Some processes will join a computation and perform the part of the computation. When the user sends an update request to the running parallel program, the root process broadcasts an update message to the participanting processes. For the update synchronization, the root process also sends a barrier message to the all worker processes. Even if a worker process completes the update, it is blocked until the others will do. After all updates are finished suceessfully, each worker process can resume the previous computation whose state information has been stored right before the update. The intermediate results of the old version are transferred to the new one. Therfoere, the old version's computation will be continued in the new one without loss of the intermediate results.

## V. Conclusion

In terms of shortening downtime, dynamic software updating can be effective in parallel application. This paper presents fundamental and useful strategies of allowing parallel applications to be changed at runtime. The initial implementation demonstrates the proposed suggestions on dynamic updating are feasible in parallel applications.

## Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A1006022, NRF-2010-0022080).

## References

[1] J. Kramer and J. Magee. Dynamic configuration for distributed systems. IEEE Trans. Software Engineering, 11(4):424–436, 1985. http://dx.doi.org/10.1109/TSE.1985.232231

[2] Lu´ıs Pina Lu´ıs Veiga, and Michael Hicks, Rubah: DSU for Java on a stock JVM, Proceedings of the ACM Conference on Object-Oriented Programming Languages, Systems, and Applications, Oct. 2014.

[3] Suriya Subramanian, Michael Hicks, and Kathryn S. McKinley. Dynamic software updates for Java: A VM-centric approach. ACM Conference on Programming Language Design and Implementation, June 2009.

[4] Iulian Neamtiu, Michael Hicks, Gareth Stoyle and Manuel Oriol, Practical Dynamic Software Updating for C, In Programming Language Design and Implementation 2006, June, Ottawa, Canada, 2006.

[5] Haibo Chen, Jie Yu, Chengqun Hang, Binyu Zang, and Pen-Chung Yew, Dynamic Software Updating Using a Relaxed Consistency Model, IEEE Transactions on Software Engineering, vol. 37, issue 5, September 2011.

[6] Kristis Makris and Rida A. Bazzi, Immediate Multi-Threaded Dynamic Software Updates Using Stack Reconstruction, In USENIX Annual Technical Conference 2009, June 2009.

[7] Andrew Baumann and Gernot Heiser, Providing Dynamic Update in an Operating System, Proc. USENIX, 2005.

[8] Raspberry Pi computer, http://www.raspberrypi.org