Advanced Science and Technology Letters
Vol.52 (AICT 2014), pp.56-59
http://dx.doi.org/10.14257/astl.2014.52.09

Towards Changing Parallel Applications at Runtime

Dong Kwan Kim

Department of Computer Engineering
Mokpo National Maritime University, Jeonnam, 530-729, Korea
dongkwan@mmu.ac.kr

Abstract. Even though dynamic software updating (DSU) has addressed a variety of problems on software maintenance or evolution, there are a few outcomes in applying it to parallel applications. This paper explores the applicability of dynamic software updating for parallel applications using Message Passing Interface (MPI) which is a standardized message-passing system. As a validation, the proposed updating system dynamically changes an MPI program for matrix multiplication. During code replacement, the updating system can avoid unacceptable service interruption and keep intermediate computation results. The experimental result demonstrates that dynamic updates allow the MPI application to shorten its computation time and to enhance its operational flexibility.

Keywords: Message Passing Interface (MPI), parallel programming, dynamic changes

1 Introduction

Due to internal or external factors, most software systems will change over time after the initial release. The internal factor indicates program bugs, errors, and security flaws in the software system itself and the external factor means the change of users' requirements or computing environment. The inappropriate responses to such changes can cause software systems to be unstable, eventually crashed.

Dynamic software updating (DSU) [1, 2] can address software changes by allowing the programmer to change executing program code without rebooting or stopping. According to implementation strategies, dynamic update systems can support partial or whole code changes. While the partial update injects new code into a running program, the whole update replaces an old version with a new one.

This paper is intended to apply the DSU technique to Message Passing Interface (MPI) applications. MPI is a standardized library specification for message-passing and is used broadly among parallel software developers [3, 4]. MPI applications are usually designed for high performance on either massively parallel machines or workstation clusters. They are also characterized by long-running computations and high reliability. Thus, it is desirable not to reboot or stop the MPI application in order to reflect code changes. The proposed update system replaces the running MPI application with a new version while preserving intermediate computation results in

ISSN: 2287-1233 ASTL

order to shorten the total computation time. The system supports seamless computation– the computation of the MPI application can transfer to a new version as if it were executed on a single MPI program.

The rest of this paper is structured as follows. Section 2 gives an overview of the proposed system. Section 3 describes the proof-of-concept that changes an MPI application at runtime. Section 4 presents concluding remarks.

2 Approach Overview

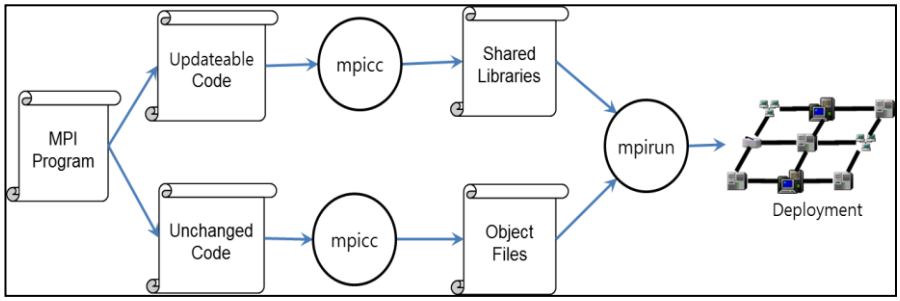

Fig. 1. Overall flow of making updateable MPI applications.

It is not common that software developers consider dynamic changes when they construct applications. Since the main role of software developer implements application logic, a special step adds the required functionality to enable dynamic updates. Therefore, an original MPI code needs to be written into updateable software without affecting its functionality. Figure 1 describes the overall procedures by which a parallel MPI application is enhanced to be dynamically updateable. At the first phase in Figure 1, the MPI program is divided into updateable and unchanged code. In this step, we can extract the updateable module which may be changed after deployment. Deciding on the updateable module depends on the expertise of software developer. The MPI compiler called *mpicc* compiles and produces shared libraries (.so files in Linux) from the updateable code. The unchanged code is compiled into object files. Finally, the *mpirun* command runs executable files on the compute cluster.

3 Proof-of-Concept

A preliminary experiment has been conducted to demonstrate MPI applications can benefit from being updated at runtime. During the experiment, a computer cluster has been used which consists of multiple CPU/GPU nodes. Each compute node runs two Intel Xenon processors with 8 cores each (a total of 16 cores), Four GPUs, 128GB RAM, and Red Hat Enterprise Linux 6.3. The computer nodes are connected by InfiniBand (40Gbit+).

Matrix multiplication is simple, but frequently used in the field of science and engineering. As the matrix size becomes larger, matrix multiplication needs to be parallelized across multiple compute nodes. In the experiment, an MPI application performs 2 x 2 matrix multiplication. The part of the matrix multiplication is made as a shared library. After the shared library is loaded, the matrix multiplication is computed. On receiving update requests, the shared library is unloaded and then a new one is loaded. The intermediate results of matrix multiplication are stored right before the old shared library is removed. The new shared library restarts the matrix multiplication at the update point where the intermediate results are preserved.

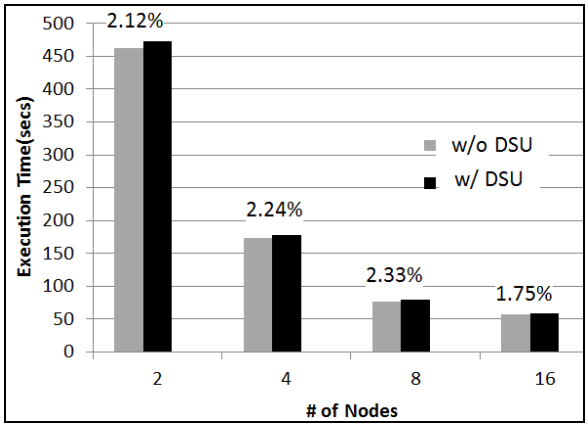

Fig. 2. Execution time of computing matrix multiplication in the compute cluster. x-axis: the number of nodes in the computer luster, y-axis: the total execution time.

Figure 2 shows the performance overhead imposed by dynamic updating. The total execution time of the original version is compared with that of the updatable version—the grey bars indicate the execution time of the original version which has no update management code and the black ones indicate the execution time of the updatable version. The label of each bar pair indicates the performance overhead between two versions. The execution time for four cases is measured by varying the number of participating cluster nodes. As expected, the execution time tends to speed up as the number of cluster nodes increases. It is meaningful that the total performance overhead never exceeds 3% even though the number of nodes increases.

4 Conclusions

It is obvious that the main concern of high performance computing is to produce a certain meaningful outcome under time constraints. Hence, parallel programming must be a necessary tool to achieve such a purpose. This paper has presented an alternative way that applies dynamic software updating for high performance computing. In the proposed approach, it is suggested that parallel MPI applications

are changed at runtime while avoiding unacceptable service interruption and supporting seamless computation. The preliminary result has demonstrated that dynamic updates can shorten the total computation time and enhance the operational flexibility of the MPI application at runtime. As parallel programming is growing rapidly under multicore processors, we can expect dynamic software updating can benefit software developers in maintaining parallel software systems.

Acknowledgments. 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).

References

1. Hicks, M., Nettles, S. M.: Dynamic software updating, ACM Transactions on Programming Languages and Systems. Vol. 27, issue 6, pp. 1049--1096. (2005)
2. Kim, D. K., Tilevich, E., Ribbens, C. J.: Dynamic Software Updates for Parallel High Performance Applications. Concurrency and Computation: Practice and Experience, vol. 23, issue 4, pp. 415--434. (2010)
3. The Message Passing Interface (MPI) standard, http://www.mcs.anl.gov/research/projects/mpi/.
4. Shafi, A., Manzoor, J., Hameed, K., Carpenter, B., Baker, M.: Multicore-enabling the MPJ Express Messaging Library, 8th International Conference on the Principles and Practice of Programming in Java (PPPJ), (2010)

