MIGRATING AN OLD VECTOR CODE TO MODERN VECTOR MACHINES

Hiroyuki Takizawa*, Kenta Yamaguchi†, Takashi Soga*, Thorsten Reimann‡, Kuzuhiko Komatsu†, Ryusuke Egawa*, Akihiro Musa*, and Hiroaki Kobayashi†

* Tohoku University, Sendai, Japan
‡ Technische Universität Darmstadt, Darmstadt, Germany
{takizawa@,yamaguchi@sc.cc.,soga@sc.cc.,komatsu@,egawa@,musa@,koba@}tohoku.ac.jp

Key words: Legacy code migration, vectorization, code optimization

Abstract. A legacy code means an important code for its application area and, because of the importance, it has been maintained for a long time. This paper presents a case study of migrating a legacy code to two modern vector systems, NEC SX-ACE and Intel Xeon Phi (Knights Landing, KNL). We first demonstrate that the legacy code written for old vector systems might not be appropriate for modern vector systems, and discusses how to maintain multiple code variants while keeping the application code as clean as possible.

1 INTRODUCTION

A legacy code in high-performance computing (HPC) is often optimized only for a particular system configuration because different systems – especially, different node architectures – require different code optimizations. Meanwhile, since the lifespan of an HPC application is usually much longer than that of one system generation, we often need to migrate such a legacy code to a new system whose node architecture could be totally different from that of the previous system. As a result, a legacy code gradually becomes difficult to maintain. Therefore, this paper shows a case study of migrating an old vector code to modern vector processors, and discusses the difficulties in legacy code migration.

2 AN INCOMPRESSIBLE FLOW SOLVER, FASTEST

The FASTEST code [1] has a long history of development, and was once optimized for classic vector systems. Some of its major kernels still have two versions, the default version optimized for standard x86 processors, and the vector version for old vector processors. The major differences between the two versions are 1) nested loops in the vector version are collapsed to be a longer single-nested loop, and 2) the hyperplane reordering is applied to the loop nest of the most time-consuming kernel, SIP SOL, for vectorization. If the vector version is used for a standard processor, the performance is lower than that of the default version.
3 A CASE STUDY OF LEGACY CODE MIGRATION

Since the vector version of the SIPSOL kernel in the original FASTEST code, called the hyperplane kernel, was written for old vector processors, it is not necessarily appropriate for modern vector processors. In this work, thus, another version of SIPSOL using the red-black reordering is implemented for performance comparison and discussions. The new SIPSOL kernel, called the red-black kernel, can increase the vector length and also avoid indirect memory access required by the hyperplane kernel. However, the red-black kernel increases the number of iterations needed until the convergence. Moreover, when the red-black kernel is executed on SX-ACE, approximately two times more floating-point operations are executed at each iteration because a mask table is used to alternately update red and black elements. Without performance comparison, therefore, it is unknown which kernel is more appropriate for modern vector processors.

Figure 1 shows the performance evaluation results. In the case of SX-ACE, the red-black kernel can run more efficiently than the other kernels because a higher average vector length is attained without using indirect memory access. It is also interesting that no code variant is appropriate for KNL, because the KNL performance heavily relies not only on the innermost loop parallelism for vectorization, but also on parallelism of outer loops for launching many threads. Yet another code variant involving more thread-level parallelism is obviously needed for KNL to achieve a reasonable performance. Therefore, it is clearly shown that these modern vector processors need different code optimization strategies, and we need a mechanism for easily testing multiple code variants for code optimization.

4 DISCUSSIONS

Unlike classic vector processors, the performance of SX-ACE could be sensitive to the memory access latency, and the lack of thread-level parallelism severely decreases the KNL performance. As shown in our case study, we need many code variants. Therefore, we have been developing the Xevolver framework [2] to transform one code in different ways for individual systems. By writing code transformation rules to generate all code variants used in the case study, it is clarified that our approach is helpful to maintain multiple code variants while keeping the application code as clean as possible.

REFERENCES
