Xevolver for Performance Tuning of C Programs

Hiroyuki Takizawa, Shunpei Sugawara, Yoichi Shimomura, Keichi Takahashi and Ryusuke Egawa

Abstract We introduce a C interface for standard C programmers to define their own code transformation rules for performance tuning, mainly assuming loop transformations. The proposed C interface can support most of important features provided by the Fortran interface. As a result, performance concerns can be defined separately as user-defined code transformation rules, and thus the original application code can be kept unchanged as much as possible.

1 Introduction

High-Performance Computing (HPC) applications are often specialized for their target platforms to achieve reasonably high performance. Such code specialization is not only labor-intensive, but also makes it difficult to migrate the applications to other platforms. One idea to overcome this difficulty is separation of performance concerns, meaning that the information specific to a particular platform is expressed separately from the computation. However, in reality, one bad practice heavily used in HPC application development to achieve high performance on multiple platforms is a so-called "ifdef" approach that writes multiple code versions within a single file and uses C macro conditionals for the preprocessor to switch the code versions to

Hiroyuki Takizawa, Yoichi Shimomura and Keichi Takahashi
Cyberscience Center, Tohoku University,
e-mail: takizawa@tohoku.ac.jp, shimomura32@tohoku.ac.jp, keichi@tohoku.ac.jp

Shunpei Sugawara
Graduate School of Information Sciences, Tohoku University,
e-mail: shunpei@hpc.is.tohoku.ac.jp

Ryusuke Egawa
Tokyo Denki University, e-mail: egawa@mail.dendai.ac.jp

M. M. Resch et al. (eds.), *Sustained Simulation Performance 2021*,
https://doi.org/10.1007/978-3-031-18046-0_6

be used at the compilation, severely degrading the code maintainability. Therefore, we need an effective way of expressing platform-specific performance concerns separately from application codes.

In the Xevolver project [16], we have developed a programming framework for performance tuning with user-defined code transformations [6, 12]. A high-level programming interface for standard HPC programmers to describe their own code transformation rules has also been developed mainly for Fortran codes [10], because most of legacy HPC applications are written in Fortran. Lately, however, it is gradually becoming popular to use not only Fortran but also other programming languages such as C and C++, especially when new HPC applications are developed from scratch. Moreover, there are many tools such as CIVL [17] available for C and C++, but not for Fortran. If we need to use such a tool, there is no choice to use Fortran at the HPC application development. Therefore, we consider that Xevolver should provide a high-level interface not only for Fortran but also for C.

In this article, we introduce a C interface for standard C programmers to define their own code transformation rules for performance tuning, mainly assuming loop transformations. Through various case studies [5, 13, 15], Xevolver's approach has been proven to be effective in achieving high performance and code maintainability. The proposed C interface can support most of important features provided by the Fortran interface. As a result, performance concerns can be defined separately as user-defined code transformation rules, and thus the original application code can be kept unchanged as much as possible.

2 Related work

Software automatic performance tuning, or auto-tuning (AT) for short, is indispensable to exploit the performance of modern HPC systems by empirically exploring a parameter space relevant to performance [7]. To use AT techniques, an application code must be developed to be *auto-tunable* [14], and be able to change its behaviors according to parameter tuning and code version switching. One challenging issue is that there is no established way of developing a practical application while keeping it auto-tunable.

So far, several case studies have demonstrated that Xevolver's approach can enable standard HPC programmers to define their own code transformation rules without any special knowledge about compiler implementation technologies [5,13,15]. Although an HPC application code is directly modified by hand to adapt to its target platform in many cases, such manual code modifications can be replaced with code transformations, and thus the original HPC application code can remain almost unchanged if Xevolver can translate the original code to its optimized and/or auto-tunable version right before the compilation process.

Although the original Xevolver framework [12] has only low-level interfaces to manipulate internal code representation, Xevtgen [10] has been developed to provide a high-level interface for Fortran programmers to describe code transformation rules using Fortran syntax. Thanks to the high-level interface, Xevolver enables to develop a Fortran code without specializing it for any specific platform.

Egawa et al. [4] have presented a database of performance tuning expertise, called HPC refactoring catalog. Loop optimization techniques in the database are described along with Fortran sample codes, and the loop optimization is expressed as a code transformation rule. Sugawara et al. [11] demonstrated that most of those techniques are also effective for C programs running on recent platforms.

3 Xevolver for C

We are now designing and developing Xevolver for C (Xev-C) for performance tuning of C programs using user-defined code transformations.

In the original Xevolver framework [12], Abstract Syntax Trees (ASTs) are expressed in an XML format, called XML-AST. Then, AST-based code transformation rules are internally expressed also in another XML format, called XSLT, which is a standardized format to describe transformations of XML data, and hence can be used for transformation of XML-AST data. Since it is too painful for standard HPC programmers to describe XSLT rules to define code transformation, Xevtgen has been developed to define code transformation rules using Fortran syntax familiar to HPC programmers [10]. AST-based representation of code transformation is certainly useful for Xevolver to express a wide variety of code transformations. However, our case studies show that the high-level interface provided by Xevtgen can cover most cases where Xevolver's approach is required. Moreover, in the case where AST-based transformation is appropriate, there are many other tools to express such a code transformation. Therefore, Xev-C internally uses Clang AST, and implements only the high-level interface for HPC programmers to define their own code transformation rules required in practice on a case-by-case basis. Xev-C does not explicitly expose ASTs to users, and assumes to use Clang tools to develop AST-based code transformations if necessary.

Unlike the original Xevolver framework built on top of the ROSE compiler infrastructure [8], Xev-C is implemented using Clang [1]. Xev-C takes two C files for user-defined code transformations as shown in Figure 1. One of the two C files is an application code to be transformed, and the other is a code transformation rule written in C. As with Xevtgen, Xev-C assumes that users provide two versions of a code fragment to define a code transformation. One is the original version, and the other is its transformed version. Figure 2 shows an example of code transformation rule defined using Xev-C. If the rule in Figure 2 is applied to the code in Figure 3, the first loop is exactly the same as the original code version in the rule, and thus is

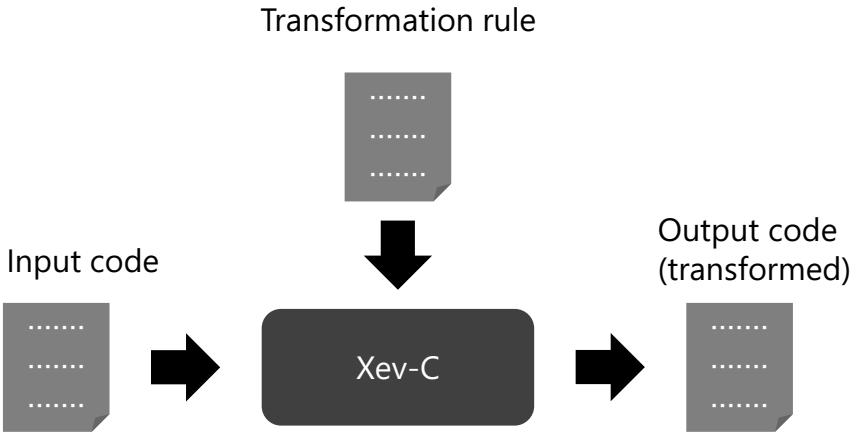

Fig. 1: Overview of Xevolver for C (Xev-C). Xev-C takes two C codes, input code and transformation rule, and then produces one code, a transformed version of the input code. All the codes are written in C.

```
1  #include "xev_defs.h"
2
3  int i,j;
4  double a[10][10], b[10], c[10];
5
6  int main()
7  {
8    xev_stmt_src("label1");
9    {
10     for(i=0;i<10;i++){
11       for(j=0;j<10;j++){
12         c[i] += a[i][j]*b[j];
13       }
14     }
15   }
16
17   xev_stmt_dst("label1");
18   {
19     for(int k=0;k<100;k++){
20       i = k/10;
21       j = k%10;
22       c[i] += a[i][j]*b[j];
23     }
24   }
25 }
```

Fig. 2: A simple transformation rule for loop collapse.

```
1  #include <stdio.h>
2
3  int i,j,n;
4  double a[10][10], b[10], c[10], d[10];
5
6  int main()
7  {
8    read_data_from_file(a,b,c,d);
9
10   for(i=0;i<10;i++){
11     for(j=0;j<10;j++){
12       c[i] += a[i][j]*b[j];
13     }
14   }
15
16   for(i=0;i<10;i++){
17     for(n=0;n<10;n++){
18       d[i] += a[i][n]*c[n];
19     }
20   }
21
22   write_data_to_file(a,b,c,d);
23
24   return 0;
25 }
```

Fig. 3: A simple code to be transformed.

replaced with the transformed version. As a result, in this particular example, loop collapse is applied to the first loop in Figure 3, but not to the second loop whose loop index of the innermost loop is `n`.

The transformation rule in Figure 2 is just text replacement and transforms a loop only if the loop is exactly identical to the original version in the rule. A large number of rules would be required if performance tuning is done only with such text replacement rules. To achieve performance tuning with as few rules as possible, Xev-C provides special variables, called Xev variables, so that a rule can be defined not for a particular code fragment but for a code pattern. Figure 4 shows a rule of loop collapse similar to the rule in Figure 2. In Figure 4, Xev variables `xi`, `xj`, and `stmt` are defined and used in the rule. Since `xi` and `xj` represent any expressions, they match any variables. Even if the loop index has a different name, the rule can be applied to the loop. Similarly, since `stmt` represents any statements, statements in the loop body do not affect to determine if the rule is applied. If the rule is applied, the loop body is unchanged and simply copied to the transformed version. As a result, in Figure 3, the second loop as well as the first one will be transformed by the rule.

```
1  #include "xev_defs.h"
2
3  int i,j;
4  xev_expr xi,xj;
5  xev_stmt* stmt;
6
7  int main()
8  {
9    xev_stmt_src("label1");
10   {
11     for(xi=0;xi<10;xi++){
12       for(xj=0;xj<10;xj++){
13         stmt;
14       }
15     }
16   }
17
18   xev_stmt_dst("label1");
19   {
20     for(int k=0;k<100;k++){
21       xi = k/10;
22       xj = k%10;
23       stmt;
24     }
25   }
26 }
```

Fig. 4: A simple transformation rule with Xev variables for loop collapse.

4 Evaluation and discussions

In this work, we have examined that the performance tuning expertise recorded in HPC refactoring catalog [4] can be expressed using the current design of Xev-C. As discussed in [11], Fortran codes in 28 out of 31 cases in the catalog can be translated into C. The three cases not translated into C use either of using built-in Fortran functions or libraries available only in Fortran. Most of the performance tuning techniques in the catalog are vectorization-aware loop optimizations mainly targeting the previous-generation vector systems, SX-9 [9] and SX-ACE [3]. In the following evaluation, the performance gains by the techniques are evaluated on the latest vector systems, two generations of SX-Aurora TSUBASA [2]. The system specifications are summarized in Table 1.

Xev-C does not support all the features provided by Xevtgen yet. However, we have confirmed that all the code transformation rules in the 28 C codes can be expressed as Xev-C rules. Therefore, we believe that the expressive ability of the current design of Xev-C is high enough at least for vectorization-aware loop optimizations.

Figure 5 shows the performance gains by the code transformations for SX-Aurora TSUBASA. The vertical axis shows the speedup ratio of the transformed code to the original code for each system. Each code is complied with either of `-O2` or `-O4`, to discuss how compiler optimization affects the performance. Overall, most of vectorization-aware loop optimizations for the previous-generation systems

Table 1: Specifications of the two generations of SX-Aurora TSUBASA used in the evaluation.

2nd generation SX-Aurora TSUBASA		
VE	Model	NEC Vector Engine Type 20B
	Core Count	8
	Peak Performance [TFLOPS]	2.45
	Memory Bandwidth [TB/s]	1.535
	Memory Capacity [GB]	48
	Compiler	ncc-3.4.0
VH	Model	Intel Xeon Silver 4208
	Core Count	8
	Memory Capacity [GB]	192

1st generation SX-Aurora TSUBASA		
VE	Model	NEC Vector Engine Type 10C
	Core Count	8
	Peak Performance [TFLOPS]	2.15
	Memory Bandwidth [TB/s]	0.750
	Memory Capacity [GB]	24
	Compiler	ncc-3.4.0
VH	Model	Intel Xeon Gold 6126
	Core Count	12
	Memory Capacity [GB]	96

are still effective even for SX-Aurora TSUBASA. However, because of advances in compiler technologies, it is worth mentioning that some performance tuning techniques are no longer effective or even harmful on performance, meaning that the compiler can perform the same or even better optimizations especially with higher-level optimization flag, `-O4`. For example, for Case No. 20, the loop optimization technique in the catalog is still effective if the code is compiled with the `-O2` flag, and thus the speedup ratio exceeds 1. However, when the `-O4` flag is used, the performance is degraded by applying the same loop optimization technique, because the compiler can optimize the loop better than the technique. This clearly indicates that a performance tuning technique should not directly be applied to an application code because it could become ineffective or even harmful in the future. Accordingly, Xevolver's approach to separation of performance concerns is promising to improve the code maintainability and make it possible to develop an application in a future-proof way.

Since Xev-C is designed for C programs, it can work together with other tools developed for C. For example, in [11], Xev-C is combined with a formal verification tool, CIVL [17], to check if a user-defined code transformation keeps the execution result of the transformed code unchanged. This code equivalence checking is an important feature for our code transformation framework, even though some technical issues remain unsolved. Therefore, user-defined code transformation with code equivalence checking will further be discussed in our future work. Combining Xev-C with other tools could also be interesting research topics.

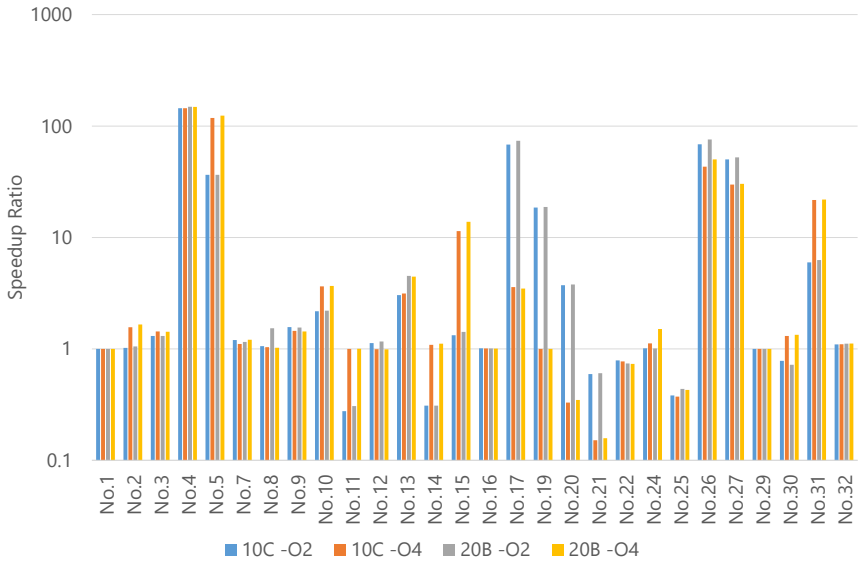

Fig. 5: Speedup ratio by code transformation, which is the performance ratio of the transformed code to the original code. A code transformation could degrade the performance because it represents a performance tuning technique for previous-generation vector systems.

5 Conclusions

This article has introduced Xev-C, which is a C interface to describe user-defined code transformation rules using C. Our evaluation results show that Xev-C can already express important features to express vectorization-aware loop optimizations, and achieve separation of performance concerns by defining code transformation rules separately from application codes. As compiler's optimization capability could change over time, a performance tuning technique could become ineffective or even harmful. Therefore, separation of performance concerns is important, and the case study in this article has demonstrated that Xevolver can contribute to the separation.

Acknowledgements This work is partially supported by MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program "R&D of a Quantum-Annealing-AssistedNext Generation HPC Infrastructure and its Applications," and JSPS KAKENHI Grant Numbers JP20H00593 and JP21H03449.

References

1. Clang https://clang.llvm.org/
2. R. Egawa, S. Fujimoto, T. Yamashita, D. Sasaki, Y. Isobe, Y. Shimomura and H. Takizawa. Exploiting the potentials of the second generation SX-Aurora TSUBASA. In: *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pp. 39–49 (2020).
3. R. Egawa, K. Komatsu, S. Momose, Y. Isobe, A. Musa, H. Takizawa and H. Kobayashi. Potential of a modern vector supercomputer for practical applications: performance evaluation of SX-ACE. *The Journal of Supercomputing* **73**, 3948–3976 (2017).
4. R. Egawa, K. Komatsu and H. Takizawa. Designing an open database of system-aware code optimizations. In: *CANDAR*, pp. 369–374 (2017).
5. K. Komatsu, R. Egawa, S. Hirasawa, H. Takizawa, K. Itakura and H. Kobayashi. Translation of large-scale simulation codes for an OpenACC platform using the Xevolver framework. *International Journal of Networking and Computing* **6**(2), 167–180 (2016).
6. K. Komatsu, A. Gomi, R. Egawa, D. Takahashi, R. Suda and H. Takizawa. Xevolver: A code transformation framework for separation of system-awareness from application codes. *Concurrency and Computation: Practice and Experience* **32**(7), 1–20 (2019).
7. K. Naono, K. Teranishi, J. Cavazos and R. Suda (eds.) *Software Automatic Tuning – From Concepts to State-of-the-Art Results*. Springer-Verlag, New York (2010).
8. ROSE Compiler, http://rosecompiler.org/
9. T. Soga, A. Musa, Y. Shimomura, R. Egawa, K. Itakura, H. Takizawa, K. Okabe and H. Kobayashi. Performance evaluation of NEC SX-9 using real science and engineering applications. In: *The Conference on High Performance Computing Networking, Storage and Analysis (SC09)* (2009).
10. R. Suda, H. Takizawa and S. Hirasawa. Xevtgen: Fortran code transformer generator for high performance scientific codes. *International Journal of Networking and Computing* **6**(2), 263–289 (2016).
11. S. Sugawara, Y. Shimomura, R. Egawa and H. Takziawa. Portability of vectorization-aware performance tuning expertise across system generations. In: *14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)* (2021).
12. H. Takizawa, S. Hirasawa, Y. Hayashi, R. Egawa and H. Kobayashi. Xevolver: An XML-based code translation framework for supporting HPC application migration. In: *The 21st annual IEEE International Conference on High Performance Computing (HiPC 2014)* (2014).
13. H. Takizawa, T. Reimann, K. Komatsu, T. Soga, R. Egawa, A. Musa and H. Kobayashi. Vectorization-aware loop optimization with user-defined code transformations. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)* (2017).
14. H. Takizawa, D. Sato, S. Hirasawa and H. Kobayashi. Making a legacy code auto-tunable without messing it up. In: *Poster presentation at ACM/IEEE Supercomputing Conference (SC16)*, pp. 1–2 (2016).
15. H. Takizawa, D. Sato, S. Hirasawa and D. Takahashi. A customizable auto-tuning scenario with user-defined code transformations. In: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1372–1378, IEEE (2017).
16. Xevolver: CREST. https://xev.sc.cc.tohoku.ac.jp/
17. M. Zheng, M.S. Rogers, Z. Luo, M.B. Dwyer and S.F. Siegel. CIVL: Formal verification of parallel programs. In: *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2015).