International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 7, Number 2, pages 405–418, July 2017

A Directive Generation Approach to High Code-Maintainability for Various HPC Systems

Kazuhiko Komatsu, Ryusuke Egawa, Hiroyuki Takizawa
Cyberscience Center, Tohoku University
6-3, Aramaki Aza Aoba, Aoba-ku, Sendai, Miyagi, 980-8578, Japan

Hiroaki Kobayashi
Graduate School of Information Sciences, Tohoku University
6-6-01, Aramaki Aza Aoba, Aoba-ku, Sendai, Miyagi, 980-8578, Japan

Received: February 15, 2017
Revised: May 6, 2017
Accepted: June 5, 2017
Communicated by José Gracia

Abstract

The emergence of various high-performance computing (HPC) systems compels users to write a code considering the characteristic of each HPC system. To describe the system-dependent information without drastic code modifications, the directive sets such as the OpenMP directive set and the OpenACC directive set are proofed to be useful. However, the code becomes complex to achieve high performance on various HPC systems because different directive sets are required for various HPC systems. Thus, the code-maintainability and readability are degraded. This paper proposes a directive generation approach that generates various kinds of directive sets using user-defined rules. Instead of using several kinds of directive sets, users only have to write special placeholders that are utilized to specify a unique code pattern where several directives are inserted. Then, the special placeholders trigger the generation of appropriate directives for each system using a user-defined rule with a code transformation framework *Xevolver*. Because only special placeholders are inserted in the code, the proposed approach can keep the code-maintainability and readability. From the performance evaluations of directive-based implementations on various HPC systems, it is shown that the best implementation is different among the HPC systems. Then, through the demonstration of transformation into multiple kinds of implementations, the proposed approach can successfully generate directives from a smaller number of special placeholders. Therefore, it is clarified that the proposed directive generation approach is effective to keep the maintainability of a code to be executed on various HPC systems.

Keywords: Maintainability, Performance-Portability, Code Transformation Framework

1 Introduction

Recent statistical data on the TOP 500 list [1] indicate that the variety of HPC systems has been increasing. Generally, these HPC systems can be classified by their equipped processors; a scalar processor, an accelerator, and a vector processor. Scalar-type systems are equipped with a large number of scalar processors that have multiple cores and large cache memories. The scalar-type systems are suitable for highly parallel calculations for massively parallel applications [2]. Accelerator-type systems are equipped with accelerators such as GPUs (Graphics Processing Units) and Intel Xeon Phi processors that have a lot of small simple computational cores. These accelerator-type systems can efficiently perform data parallel calculations [3][4]. Vector-type systems are equipped with vector processors that have several cores specialized for vector calculations with high memory bandwidth. The vector-type systems can calculate sets of data elements at the same time [5][6].

To exploit the potential of these various HPC systems, system-dependent information for each HPC system as well as application behavior have to be written in a code. The application code should effectively utilize the features of each HPC system.

The most widely-used approach in writing system-dependent information is to write several versions of an application code, each of which is optimized for a specific HPC system. However, this approach tends to drastically modify a code for each optimization. For example, optimizations for an accelerator using CUDA or OpenCL [7] require major code modification to the original code [8][9]. As a result, the code-maintainability degrades because an application developer has to maintain different versions of the code.

The other approach is to use directive sets that can describe system-dependent information while keeping the code-maintainability. For example, the OpenMP directive set enables a serial code to be executed on a shared memory system and/or an accelerator [10], while the OpenACC directive set allows a code to be executed on an accelerator [11]. Various features of HPC systems can be utilized by just inserting directives to a code.

However, even in the directive-based approach, a code for various HPC systems becomes complicated. Different directive sets have to be inserted into one code by considering individual systems. The OpenMP directive set is often used for scalar-type and vector-type systems while the OpenACC directive set is used for an accelerator-type system. Thus, different kinds of directive sets have to be maintained in one code for two types of system; both OpenMP and OpenACC directives are inserted into one application code. As a result, a large number of code lines would be spent for directives, which do not express the application behavior and are used only for performance. Thus, the maintainability and readability of such a code decrease even with the directive-based approach.

This paper proposes a directive generation approach that generates various kinds of directive sets using user-defined rules triggered by special placeholders. Instead of writing several kinds of directive sets, users only have to write the special placeholders that are utilized to specify a unique code pattern where several kinds of directive sets are inserted. Then, the special placeholders trigger the generation of appropriate directives for each system using a user-defined rule with a code transformation framework *Xevolver*. Thus, a single version of an application code with special placeholders can be used for various HPC systems, which can keep the maintainability and readability of the original code.

The rest of this paper is organized as follows. Section 2 briefly describes the related work on code-maintainability for various HPC systems. Section 3 proposes a directive generation approach that generates various kinds of directives by using the Xevolver framework. Section 4 evaluates the four kind of directive-based implementations to show that the best implementation is different among HPC systems. Then, directive generation using user-defined rules is demonstrated to clarify the effectiveness of the proposed approach. Section 5 gives concluding remarks and future work.

2 Code-Maintainability of the Directive-Based Approach

The directive sets are often used in the field of HPC. One of the reasons is that various functions offered by different directive sets are easily utilized by just inserting directives into an existing code. Thus, users can easily try using directives. Another reason is that the use of directive sets can avoid drastic code modifications. Simply inserting directives into a code does not require a modification of the structure of the original code, and therefore the code-maintainability can be kept. Thus, an application developer can easily continue to use the original code with directives.

However, an effective directive set depends on the target HPC system executing the application. Thus, various kinds of directive sets need to be used in one application code in order to maintain

```
!!$omp target data map(to: a, b, c, p, bnd, wrk1) map(from: wrk2)
!$omp parallel shared (kmax,jmax,imax,nn,a,p,b,c,bnd,wrk1,wrk2) &
!$omp private (k,j,i,s0,ss,loop)
!$acc data present(a,b,c, p, bnd) &
!$acc present      (wrk1,wrk2)
  do loop=1,nn
     gosa= 0.0
!$omp do reduction(+:gosa)
!$acc kernels
!$acc loop gang
!!$acc parallel loop collapse(3) reduction(+:gosa)
     do k=2,kmax-1
        do j=2,jmax-1
!$omp simd
!$acc loop vector(256) reduction(+:gosa)
           do i=2,imax-1
              s0=a(I,J,K,1)*p(I+1,J,K) &
...
              ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
              gosa=gosa+ss*ss
              wrk2(I,J,K)=p(I,J,K)+OMEGA *ss
           enddo
!$omp end simd
        enddo
     enddo
!!$acc end parallel loop
!$omp end do nowait
!$omp barrier
!$omp do
!$acc loop gang
!!$acc parallel vector_length(128)
!!$acc loop collapse(3)
     do k=2,kmax-1
        do j=2,jmax-1
!$acc loop vector(256)
!$omp simd
          do i=2,imax-1
              p(I,J,K)=wrk2(I,J,K)
           enddo
!$omp end simd
        enddo
     enddo
!!$acc end parallel
!$acc end kernels
!$omp end do nowait
  enddo
!$acc end data
!$omp end parallel
!!$omp end target data
```

Figure 1: The Himeno benchmark with multiple directive sets.

one unified version of the code for various HPC systems.

Figure 1 shows the Himeno benchmark [12], in which four kinds of the directive-based implementations using the OpenMP and the OpenACC directive sets are applied; OpenMP parallel version, OpenMP target version, OpenACC kernels version, and OpenACC parallel version. OpenMP parallel version can perform multithread execution to use multiple cores of CPU. OpenMP target, OpenACC kernels, and OpenACC parallel versions can offload executions to an accelerator. By the target constructs of OpenMP, the offload region can be specified. In the same way, OpenACC kernels and OpenACC parallel constructs specify the offload region. While the kernels construct only specify the region of the code, the parallel construct needs to specify the mapping of the offload region to the accelerator as well as the offload region.

From this figure, it is obvious that many code lines are spent for these directives shown in the red color. 28 code lines out of the total 58 code lines in the loop body are spent for OpenMP or OpenACC directives, which is more than 48% of the total lines of the kernel code. Moreover, these directive sets do not represent application behavior, but are used only for the performance. Furthermore, two different implementations using the same directive set are not allowed. In this figure, two of the implementations are commented out; one is the implementation using OpenMP target directives, and the other is that using OpenACC parallel directives. Thus, in this example,

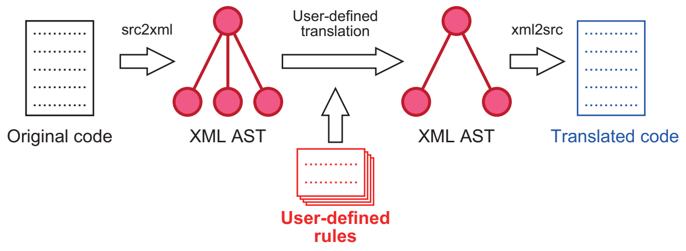

Figure 2: An overview of transformation using the Xevolver framework.

users have to manually enable appropriate OpenMP target directives, switch the implementations from OpenACC kernels directives to OpenACC parallel directives, or vice versa. Even with high functional editors or IDEs (Integrated Development Environments), it is hard to easily enable or switch them.

In addition, directives need to be described from the first row in the case of the fixed format of Fortran. As the indent style of directive sets is different from that of Fortran, the directives degrade the code readability. In this way, simultaneous use of various kinds of directive sets drastically increases the number of code lines, and spoils the code-maintainability and readability. Thus, there is a demand for helpful tools that can keep the original code unchanged even in the directive-based implementations.

Code transformation is one of the ways to keep code-maintainability. Instead of directly modifying the original code, the original code is transformed into an appropriate code for a target HPC system. Code modification of the original code is replaced with code transformation. Many code transformation frameworks have been proposed such as ChiLL [13], POET [14], and Xevolver [15]. These code transformation frameworks can perform code transformation according to custom transformation rules. In other words, optimizations for each system can be separated from the original code as transformation rules. ChiLL and POET are, in particular, specialized for combinations of pre-defined loop transformation rules. On the other hand, in addition to combinations of pre-defined rules, Xevolver and its tool-set can easily define custom code transformation rules. Moreover, the transformation rules generated by the knowledge of optimizations of practical applications are collected and opened for potential users [16]. Flexible code transformations can be easily achieved by Xevolver.

Figure 2 shows an overview of transformation by the Xevolver framework. By preparing a transformation rule in advance, an original code is transformed using the rule in the framework. First, an application code is parsed by using the ROSE compiler infrastructure [17], and then its output AST (Abstract Syntax Tree) is converted to an XML document [18] by the *src2xml* command of the Xevolver framework. An AST is represented as an XML document called an *XML AST*. The XML AST is transformed into a new XML AST based on a user-defined transformation rule. After the transformation of the original XML AST, the new XML AST is converted by the *xml2src* command of the Xevolver framework. Finally, a transformed version of the application code is generated [15].

Since XSLT (XML Stylesheet Language Transformations) [19] is a standard XML data conversion format, XSLT is utilized to describe an XML AST transformation rule for the Xevolver framework. Compiler experts can describe various code transformation rules by using XSLT [15][20]. Figure 3 shows an example of the AST rule written in XSLT to transform the custom directive *!$xev directive_gen* into *OpenMP do* and *OpenMP end do* directives. Line 5 declares the variable *xev_transform* with "*!$xev directive_gen*". Lines 7 to 9 specify that the templates defined from Lines 11 to 33 are applied to whole AST. From Lines 12 to 17, all nodes in AST are copied by the *copy* and *copy-of* elements. Lines 19 to 33 insert *OpenMP do* and *OpenMP end do* directives. As a directive is represented as an XML element of *PreprocessingInfo*, the template from Lines 20 to 33 is applied to the *PreprocessingInfo* element. Line 22 searches for the *PreprocessingInfo* whose text starts with "*!$xev directive_gen*." Only when the target *PreprocessingInfo* is found, the directives *"!$omp parallel do"* and *"!$omp end parallel do"* are inserted in Lines 23 and 24, respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:variable name="xev_transform" select="'!$xev directive_gen'"/>

  <xsl:template match="/">
    <xsl:apply-templates />
  </xsl:template>

  <!-- copy nodes -->
  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>

  <!-- replace to openmp directive -->
  <xsl:template match="PreprocessingInfo">
    <xsl:choose>
      <xsl:when test="starts-with(text(),$xev_transform)">
        <PreprocessingInfo type="3" pos="2">!$omp parallel do</PreprocessingInfo>
        <PreprocessingInfo type="3" pos="3">!$omp end parallel do</PreprocessingInfo>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy>
          <xsl:copy-of select="@*" />
          <xsl:apply-templates />
        </xsl:copy>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

Figure 3: The XSLT transformation rule for inserting OpenMP parallel directives.

However, for standard users who develop and optimize an HPC application code using high-level languages such as Fortran and C, it is too low-level to describe such transformation rules in XSLT. Thus, the Xevolver framework also provides a high-level way to generate user-defined transformation rules. *Xevtgen* is a high-level tool that can easily generate an XSLT transformation rule [21]. A user-defined transformation rule can be described in the conventional Fortran with some special Xevtgen directives. Thus, standard Fortran programmers can easily learn and generate a user-defined transformation rule.

This paper focuses on the flexibility and easiness of the Xevolver framework for keeping code-maintainability for multiple HPC systems. Due to the diversity of HPC systems, multiple versions of directive sets need to be inserted into a code to exploit the potential of each HPC system. By using the Xevolver framework, system-dependent information such as directive sets are separated from the original code.

3 Directive Generation Using User-defined Rules with *Xevtgen*

This section proposes a directive generation approach that generates different directives using user-defined rules. The key idea of the proposed approach is to generate appropriate directives for individual HPC systems triggered by special placeholders using the *Xevolver* framework and its user-defined rule generator *Xevtgen*. For directive generation using a user-defined rule, a special placeholder is utilized to identify a unique code pattern. Then, programmers define the corresponding code transformation rule to the special placeholder. The correctness of the transformation has to be guaranteed by the programmers. By using the transformation rule, the special placeholder triggers the generation of directives. As a result, the proposed approach can keep the maintainability of the original code because the programmers write only special placeholders to an application code instead of writing several kinds of directive sets for various HPC systems.

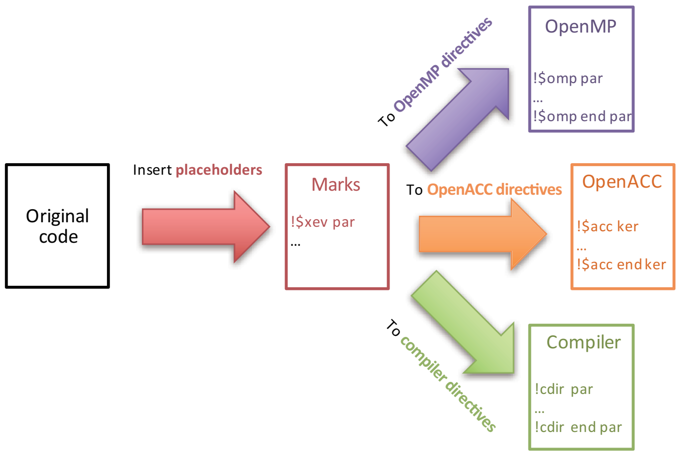

Figure 4: An overview of the proposed directive generation.

```
358:!$xev jacobi
359:  do loop=1,nn
360:     gosa= 0.0
361:     do k=2,kmax-1
362:        do j=2,jmax-1
363:           do i=2,imax-1
364:              s0=a(I,J,K,1)*p(I+1,J,K) &
...
376:              ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
377:              GOSA=GOSA+SS*SS
378:              wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
379:           enddo
380:        enddo
381:     enddo
382:     do k=2,kmax-1
383:        do j=2,jmax-1
384:          do i=2,imax-1
385:              p(I,J,K)=wrk2(I,J,K)
386:           enddo
387:        enddo
388:     enddo
389:  enddo
```

Figure 5: The Himeno benchmark with a special placeholder.

Figure 4 shows an overview of the proposed directive transformation approach. First, instead of OpenMP, OpenACC, and/or compiler-specific directives, a special placeholder is inserted into an original application code. In this figure, *!$xev directive_gen* is used as an example, and users can use any keyword as a placeholder, as long as it is unique. Then, the Xevolver framework takes responsibility for generating appropriate directives triggered by the special placeholder. Considering the target HPC system, the directive generation is carried out based on user-defined transformation rules. By defining multiple transformation rules in advance, appropriate directive generation for individual systems can be realized. The special placeholder *!$xev directive_gen* triggers the generation of OpenMP, OpenACC, or compiler-specific directives. As a result, a single version of the application code with special placeholders can be utilized for various HPC systems. Instead of maintaining multiple versions of an application code, users only have to maintain a single version of the code with special placeholders and transformation rules.

Figure 5 shows an example of the kernel of the Himeno benchmark, into which a special placeholder is inserted. The placeholder is inserted to specify the code line where OpenMP and/or OpenACC directives are inserted as in Figure 1. In Line 358, the placeholder *"!$xev jacobi"* is inserted into the main loop body of the benchmark. Compared with Figure 1, the code is very simple and it is easier to concentrate on understanding the application behavior.

By using the inserted placeholder, various directives are generated according to user-defined rules made by Xevtgen. Instead of writing AST transformation rules in XSLT, programmers write a pair of Fortran-like codes called *a dummy Fortran code*; one is a *source pattern* of the original code and the other is a *destination pattern* of the transformed code. Then, from these patterns, Xevtgen automatically generates an XSLT transformation rule from the original code to the transformed code.

Figure 6 shows a dummy Fortran code that generates OpenMP directives from the special placeholder in the jacobi subroutine of the Himeno benchmark. Note that other dummy Fortran codes for the initialization subroutine that are corresponding to Lines 8 to 65 are omitted. The dummy Fortran codes become inputs of Xevtgen. Then, the XSLT transformation rules are automatically generated. The conventional Fortran code and some special *!$xev tgen* directives appear in the dummy Fortran code. The source and destination patterns are defined in *!$xev tgen src* and *!$xev tgen dst*, respectively. The source pattern is defined from Lines 66 to 85 and the destination pattern is defined from Lines 87 to 117. In the source pattern of Line 67, the same placeholder *!$xev jacobi* as the Line 358 of Figure 5 is described. It detects the unique code pattern.

In the dummy Fortran code, special variables are utilized to define these patterns such as variables ii, ib, ie, is, and so on. These special variables declared from Lines 4 to 7 can represent variables of any name. In the source pattern, such special variables are used to represent loop indices. Hence, if the same special variable appears in the destination pattern, it is replaced with the corresponding loop index at code transformation. Note that those special variables are used only for representing variables in Fortran codes; they cannot represent variables in directives. Even if a special variable appears in a part of a directive line such as private, shared, and reduction clauses, it is not replaced with any variable and thus appears in the transformed code as it is. Therefore, programmers have to directly write concrete variable names in directive lines, carefully considering the transformed code. Furthermore, more special variables are utilized such as variables $body1$ and $body2$ declared in Line 3. Variables $body1$ and $body2$ are represented as the loop bodies in *!$xev tgen stmt (body1)* in Line 73 and *!$xev tgen stmt (body2)* in Line 80, respectively.

Application-dependent descriptions are used in Lines 88, 89, 91, and 92 in order to simply and easily write the dummy code. The variable names used in the Himeno benchmark are used for *shared*, *private*, and *reduction* clauses. In addition, Line 91 assumes that the particular statement, gosa=0.0, exists in the source code. Thus, this dummy Fortran code is useful only for the Himeno benchmark shown in Figure 5. If the variable names are changed, the dummy code has to be changed accordingly. In return for not pursuing such generality, our approach can simplify the syntax of dummy Fortran codes, and hence facilitate the definition of custom code translation rules for individual applications.

This is a trade-off between generality and easiness of writing the dummy code. If programmers can obtain these information from the source code, the application-dependent descriptions in the dummy Fortran code can be avoided, which increases the generality of the dummy code.

In addition to the dummy Fortran code in Figure 6, by preparing dummy Fortran codes for OpenMP target, OpenACC kernels, and OpenACC parallel directives, the Himeno benchmark with a special placeholder can be transformed into any of four implementations. The directives using two kinds of directive sets such as OpenMP and OpenACC can also be generated from one dummy Fortran code.

The maintenance effort of these dummy Fortran codes is small because the dummy codes need to be modified only when a loop structure of the original code is changed. As the modified loop structure in the original code can be used in the dummy codes, the maintenance effort is minimized.

From a dummy Fortran code with several *!$xev tgen* directives, a transformation rule from the source pattern to the destination one can be generated. Only knowledge of Fortran and several *!$xev tgen* directives is necessary to generate a transformation rule. Any knowledge of XML, XSLT, and AST is not necessary at all. Therefore, programmers can easily and quickly learn how to describe a user-defined rule by using Xevtgen.

```fortran
program OpenMP_directive_generation

!$xev tgen list(body1,body2) stmt
!$xev tgen var(ii,ib,ie,is) exp
!$xev tgen var(jj,jb,je,js) exp
!$xev tgen var(kk,kb,ke,ks) exp
!$xev tgen var(ll,lb,le,ls) exp
...
!$xev tgen src begin
!$xev jacobi
do ll = lb, le, ls
gosa=0.0
do kk = kb, ke, ks
do jj = jb, je, js
do ii = ib, ie, is
!$xev tgen stmt(body1)
end do
end do
end do
do kk = kb, ke, ks
do jj = jb, je, js
do ii = ib, ie, is
!$xev tgen stmt(body2)
end do
end do
end do
end do
!$xev tgen src end

!$xev tgen dst begin
!$omp parallel shared (kmax,jmax,imax,nn,a,p,b,c,bnd,wrk1,wrk2) &
!$omp private (k,j,i,s0,ss,loop)
do ll = lb, le, ls
gosa=0.0
!$omp do reduction(+:gosa)
do kk = kb, ke, ks
do jj = jb, je, js
!$omp simd
do ii = ib, ie, is
!$xev tgen stmt(body1)
end do
!$omp end simd
end do
end do
!$omp end do nowait
!$omp barrier
!$omp do
do kk = kb, ke, ks
do jj = jb, je, js
!$omp simd
do ii = ib, ie, is
!$xev tgen stmt(body2)
end do
!$omp end simd
end do
end do
!$omp end do nowait
end do
!$omp end parallel
!$xev tgen dst end

end program OpenMP_directive_generation
```

Figure 6: A dummy Fortran code to generate a transformation rule by Xevtgen.

Table 1: The specifications of computing systems.

	System #1	System #2	System #3	System #4
CPU	2× Intel Xeon E5-2630	Intel Xeon E5-2680v4	2× IBM Power8	SX-ACE
#. cores	2× 6	14	2× 8	4
Capacity	64 GB	128 GB	512 GB	64GB
GPU	Nvidia Tesla K20X	Nvidia Tesla P100	Nvidia Tesla P100	N/A
#. CUDA cores	2688	3584	3584	N/A
VRAM	6 GB	16 GB	16 GB	N/A
OS	CentOS 6.2	CentOS 6.8	Ubuntu 16.04.1	SUPER-UX
Compiler	PGI compiler 16.10 gcc 6.3.0 (OpenMP4.5)	PGI compiler 16.10 gcc 6.3.0 (OpenMP4.5)	PGI compiler 16.10	SX Fortran90 Rev.533

4 Evaluation

In this section, we demonstrate directive generations from special placeholders for various implementations to clarify the effectiveness of the proposed approach.

4.1 Experimental environment

In the evaluation, the Himeno benchmark and a atmospheric simulation code are used. For the Himeno benchmark, four different code versions, i.e., *OpenMP parallel*, *OpenMP target*, *OpenACC kernels*, and *OpenACC parallel*, are manually generated from the serial version of the benchmark written in Fortran 90. Then, four dummy Fortran codes for Xevtgen are prepared to generate the XSLT transformation rules. Furthermore, the parallel version of the Himeno benchmark written in Fortran 90 and OpenMP, called *OpenMP original*, is also used for the performance evaluation.

The atmospheric simulation code called MSSG (Multi-scale Simulator for the Geo-environment) has been developed for a global geo-environment simulation that incorporates non-hydrostatic atmosphere, ocean, and sea-ice components [22][23]. As the code is originally developed on a SX vector supercomputing system, the compiler-specific directives for SX systems are utilized. In this experiments, *OpenMP parallel* and *OpenACC kernels* versions are manually generated from one of the dominant kernels in the MSSG code. Then, in the same way of the Himeno benchmark, three dummy codes for Xevtgen are prepared to generate the XSLT transformation rules.

The performance-portability of these versions on a CPU and a GPU is also discussed. For the performance evaluation, three hybrid computing systems and one supercomputing system are used. Table 1 shows the specifications of the hybrid computing systems and the supercomputing system. The first system is equipped with Intel Xeon E5-2630 connected to Nvidia Tesla K20 via PCI express. The second system is equipped with Intel Xeon E5-2680v4 connected to Nvidia Tesla P100 via PCI express. The third system is equipped with Intel Xeon E5-2680v4 connected to Nvidia Tesla P100 via Nvidia NVLink, which has a $5 times$ higher bandwidth than PCI express. The last system is the SX-ACE vector supercomputing system. It is equipped with the SX-ACE processor.

For the hybrid computing systems, the PGI compiler 16.10 is used for compiling all implementations except the OpenMP target implementation. For the OpenMP target implementation, gcc 6.3.0 is used because OpenMP version 4.0 or above is necessary for the OpenMP target directives. For the SX-ACE system, the SX Fortran 90 compiler are used.

4.2 Evaluation and discussions

Figure 7 shows the performance differences among various versions of the Himeno benchmark. The horizontal-axis indicates the versions of the benchmark using various directive sets. The *Auto-optimization* indicates that the code is automatically optimized by the compiler. The *OpenMP*

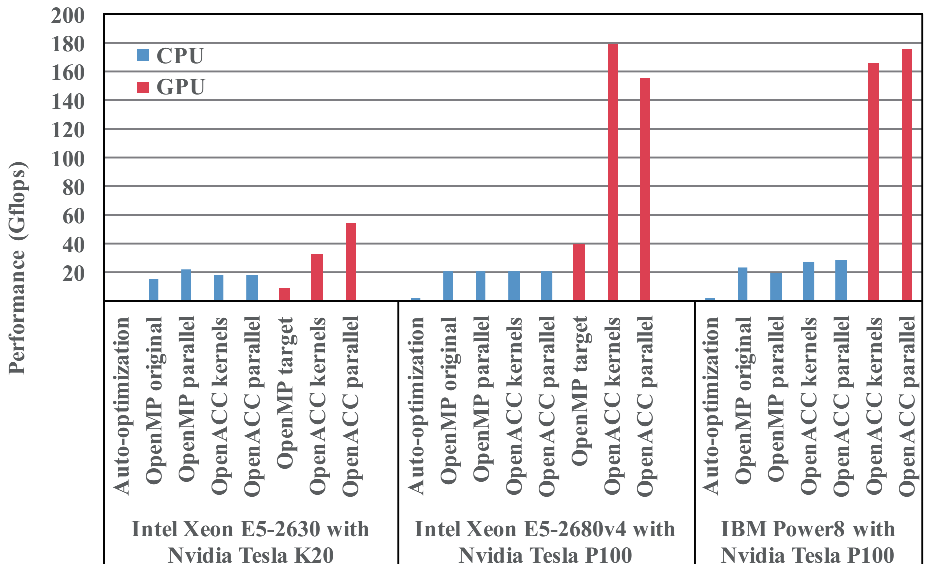

Figure 7: Performance comparisons of the Himeno benchmark with various directive sets.

Table 2: The numbers of code lines in the Himeno benchmark.

Himeno version	#. directives	#. code lines	#. rule lines
OpenMP parallel	23	427	0
OpenMP target	27	431	0
OpenACC kernels	19	423	0
OpenACC parallel	16	420	0
MP and ACC	55	459	0
Placeholder	3	407	118+122+114+111

original indicates the parallel version of the Himeno benchmark. The *OpenMP parallel*, *OpenMP target*, *OpenACC kernels*, and *OpenACC parallel* versions are generated by the proposed approach. The vertical-axis indicates the sustained performance. The blue and red bars indicate that the codes are executed on a CPU and a GPU, respectively.

First, taking look at the performances of three CPUs, the best version is different among CPUs. In E5-2630, E5-2680v4, and Power8, the best versions are *OpenMP parallel*, *OpenMP original*, and *OpenACC parallel*, respectively. When the number of threads is small, the naive implementation such as *OpenMP parallel* can achieve the highest performance. In E5-2630, *OpenMP parallel* version using 8 threads achieved the best performance. On the other hand, as the number of threads increases, especially in the case of Power8, *OpenMP original*, *OpenACC kernels*, and *OpenACC parallel* versions achieve higher performance than *OpenMP parallel* version. This is because these versions carefully consider the overhead of executing a number of threads. Especially in the *OpenMP original* version, the overhead of synchronization among threads is reduced by using asynchronous barriers with the MASTER threads. In addition, as a large number of threads assumes to be used for executions in OpenACC environment, it might handle the execution using many threads compared with *OpenACC parallel.* As a result, these implementation such as *OpenMP original*, *OpenACC kernels*, and *OpenACC parallel* versions can successfully reduce the overhead of synchronization of a large number of threads.

Furthermore, compared with *Automatic-optimization* version, the other OpenMP and OpenACC versions achieve much higher performance. This means that the directive-based approach is still effective because the HPC systems are getting more complicated so that more information are

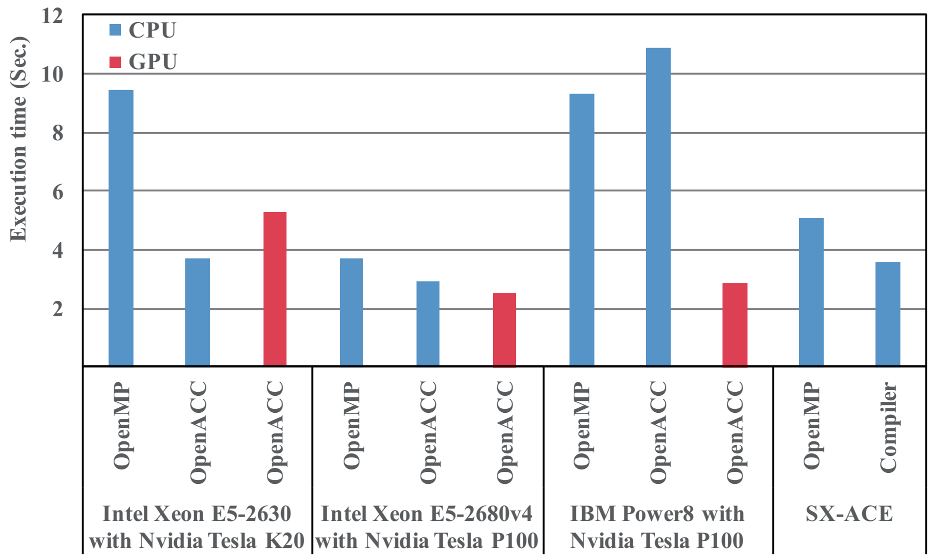

Figure 8: Performance comparisons of the MSSG kernel with various directive sets.

required from directives to exploit the high potential of such HPC systems.

In the cases of execution on GPUs, the best version is also different among GPUs. In Tesla K20 with E5-2630, Tesla P100 with E5-2680v4, and Tesla P100 with Power8, the best versions are *OpenACC parallel*, *OpenACC kernels*, and *OpenACC parallel*, respectively. In the case of Tesla K20 with E5-2630, the performance of the *OpenACC parallel* version is higher than that of the *OpenACC kernels* version. This is because the parameters for parallelization such as grid sizes are different. As a result, the times for data transfer and kernel execution using *OpenACC kernels* become shorter than that using *OpenACC parallel*. In the case of Tesla P100, the best versions are different. As the architectures of CPUs and the interconnects between a CPU and a GPU are different, the best versions become different even for the same GPU.

Taking a look at the *OpenMP target* version, the performance is much lower than the two OpenACC versions. The reason is that the implementation of *OpenMP target* is under developing. However, it is expected that the performance of *OpenMP target* becomes improved because compiler vendors and open-source community have been developing a novel feature of *OpenMP target*.

From these results, it is clarified that the best version for each processor is different even in the directive-based programming. Thus, to keep high performance-portability among various HPC systems, different directive sets need to be appropriately selected.

The proposed directive approach can realize high performance-portability by flexibly generating directives considering the HPC system. Table 2 shows the numbers of directives, total code lines, and code lines for Xevtgen dummy codes in different versions of the Himeno benchmark; *OpenMP parallel*, *OpenMP target*, *OpenACC kernels*, *OpenACC parallel*, multiple directive sets, and placeholder versions. From this table, it is clarified that the proposed approach using placeholders for directive generation can minimize the modifications of the original code. Only three special placeholders to be transformed into the four versions of implementations are necessary. Therefore, the proposed approach can keep the original code unchanged as much as possible.

However, in the proposed approach, four dummy Fortran codes for Xevtgen have to be described to generate the XSLT transformation rules. The code lines of the dummy Fortran codes for *OpenMP parallel*, *OpenMP target*, *OpenACC kernels*, and *OpenACC parallel* directive sets are 118, 122, 114, and 111, respectively. However, we believe that these numbers of lines are affordable because these dummy Fortran codes are similar to each other and easy to be described from the original code in the Fortran-like description with only five kinds of *!$xev tgen* directives.

Figure 8 shows the performance differences among various versions of the kernel of MSSG.

Table 3: The numbers of code lines in the MSSG kernel.

Himeno version	#. directives	#. code lines	#. rule lines
OpenMP parallel	3	105	0
OpenACC kernels	6	108	0
Compiler directives	4	106	0
MP, ACC and Compiler	13	115	0
Placeholder	1	103	22+41+39

The horizontal-axis indicates the versions of the kernel using various directive sets. The *OpenMP*, *OpenACC*, and *Compiler-specific directive* versions are generated by the proposed approach. The vertical-axis indicates the execution time of the kernel. The blue and red bars indicate that the codes are executed on a CPU and a GPU, respectively.

Even in the case of the practical application, the best version is different among four systems. The best version of E5-2630 with Tesla K20 is the *OpenACC* version executed on the CPU. The best versions of E5-2680v4 with Tesla P100 and Power8 with Tesla P100 are the *OpenACC* version executed on the GPU. The best version of SX-ACE is *Compiler-specific directive*. From these results, it is shown that the best version for each system is different even in the case of the practical application. Thus, to keep high performance-portability among various HPC systems, appropriate directive sets need to be selected. From this aspect, the proposed approach that can generate appropriate directives considering the target system is useful.

Table 3 shows the numbers of directives, total code lines, and code lines of Xevtgen dummy codes in different versions of the MSSG kernel; *OpenMP*, *OpenACC*, *Compiler-specific*, multiple directive sets, and placeholder versions. From this table, it is clarified that the proposed directive generation approach can minimize the modifications even in case of the MSSG kernel. Only a special placeholder is necessary to transform the original code into three versions of implementations. Therefore, the proposed approach can keep the maintainability of the original code with an affordable cost for dummy Fortran codes.

5 Conclusions

This paper proposes a directive generation approach that can generate directives for various HPC systems using the Xevolver framework. To exploit the potential of various HPC systems by one application code, various kinds of multiple directive sets are inserted because the appropriate way of using directives for each HPC system is different. To avoid decreasing the maintainability and readability due to multiple kinds of directive sets, the proposed approach inserts special placeholders instead of inserting actual directives. Then, the placeholders are used to trigger the directive generation by using user-defined rules. By generating multiple transformation rules in a programmer-friendly way using Xevtgen, each of which is corresponding to one HPC system, appropriate directives for any target HPC system can be generated.

From the experiments, it is clarified that the best implementation depends on the HPC system even in the directive-based programming. In order to exploit the potential of various HPC systems, an application code should be optimized using different directive sets by considering the target HPC system. Then, by the demonstration of the directive generation, it is clarified that the proposed approach can generate appropriate directives with minimal code modification to the original code. As a result, the proposed directive generation approach is effective to keep the high maintainability of a code to be executed on various HPC systems.

Future work is to confirm the feasibility of the proposed directive generation approach. As the number of placeholders and their corresponding transformation rules greatly depend on applications and target HPC systems, it is important to discuss feasibility and effective of the proposed approach through the evaluation using more practical applications.

Acknowledgments

This research was partially supported by Core Research of Evolutional Science and Technology of Japan Science and Technology Agency (JST CREST) "An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Heterogeneous Systems" and DFG Software for Exascale Computing (SPPEXA) "Exascale Simulation of Fluid-Structure-Acoustics Interactions." The authors would like to thank Prof. Suda for the development of Xevtgen and fruitful discussions and supports.

References

[1] Top500 supercomputer sites. [Online]. Available: http://www.top500.org/

[2] Y. Hasegawa, J.-I. Iwata, M. Tsuji, D. Takahashi, A. Oshiyama, K. Minami, T. Boku, F. Shoji, A. Uno, M. Kurokawa, H. Inoue, I. Miyoshi, and M. Yokokawa, "First-principles calculations of electron states of a silicon nanowire with 100,000 atoms on the k computer," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011, pp. 1:1–1:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063386

[3] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros, "Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.42

[4] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, and S. Matsuoka, "Peta-scale phase-field simulation for dendritic solidification on the tsubame 2.0 supercomputer," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011, pp. 3:1–3:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063388

[5] T. H. Dunigan Jr., J. S. Vetter, J. B. White III, and P. H. Worley, "Performance evaluation of the cray x1 distributed shared-memory architecture," *IEEE Micro*, vol. 25, no. 1, pp. 30–40, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1109/MM.2005.20

[6] 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 *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, 2009, pp. 28:1–28:12. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654088

[7] OpenCL - the open standard for parallel programming of heterogeneous systems. [Online]. Available: https://www.khronos.org/opencl/

[8] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[9] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi, "Evaluating performance and portability of opencl programs," in *The Fifth International Workshop on Automatic Performance Tuning*, June 2010.

[10] The OpenMP API specification for parallel programming. [Online]. Available: http://openmp.org/

[11] OpenACC directives for accelerometers. [Online]. Available: http://www.openacc-standard.org/

[12] "Himeno benchmark," http://accc.riken.jp/2444.htm.

[13] C. Chen, J. Chame, and M. Hall, "Chill: A framework for composing high-level loop transformations," University of Southern California, Tech. Rep., 2008.

[14] Q. Yi, "Poet: A scripting language for applying parameterized source-to-source program transformations," *Journal of Software - Practice & Expererience*, vol. 42, no. 6, pp. 675–706, Jun. 2012. [Online]. Available: http://dx.doi.org/10.1002/spe.1089

[15] H. Takizawa, S. Hirasawa, Y. Hayashi, R. Egawa, and H. Kobayashi, "Xevolver: An xml-based code translation framework for supporting hpc application migration," in *21st International Conference on High Performance Computing (HiPC 2014)*, Dec 2014, pp. 1–11.

[16] *Designing an HPC Refactoring Catalog Toward the Exa-scale Computing Era.*

[17] D. Quinlan and C. Liao, "The rose source-to-source compiler infrastructure," in *Cetus Users and Compiler Infrastructure Workshop, in conjunction with PACT 2011*, October 2011.

[18] "Extensible markup language (XML) 1.1 (second edition)," https://www.w3.org/TR/xml11/.

[19] "XSL transformations (XSLT) version 2.0," https://www.w3.org/TR/xslt20/.

[20] K. Komatsu, R. Egawa, S. Hirasawa, H. Takizawa, K. Itakurayz, and H. Kobayashi, "Migration of an atmospheric simulation code to an openacc platform using the xevolver framework," in *Proceedings of the Third International Symposium on Computing and Networking*, Dec 2015, pp. 528–534.

[21] R. Suda, H. Takizawa, and S. Hirasawa, "Xevtgen: fortran code transformer generator for high performance scientific codes," in *Proceedings of the Third International Symposium on Computing and Networking*, Dec 2015, pp. 528–534.

[22] K. Takahashi, R. Onishi, T. Sugimura, Y. Baba, K. Goto, and H. Fuchigami, "Seamless simulations in climate variability and hpc," in *High Performance Computing on Vector Systems 2009*, M. Resch, S. Roller, K. Benkert, M. Galle, W. Bez, and H. Kobayashi, Eds. Springer Berlin Heidelberg, 2010, pp. 199–219.

[23] K. Takahashi, A. Azami, Y. Tochihara, Y. Kubo, K. Itakura, K. Goto, K. Kataumi, H. Takahara, Y. Isobe, S. Okura, H. Fuchigami, J. Yamamoto, T. Takei, Y. Tsuda, and K. Watanabe, "World-highest resolution global atmospheric model and its performance on the earth simulator," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2011*, Nov 2011, pp. 1–12.