
Kazuhiko Komatsu, Ryusuke Egawa, Hiroyuki Takizawa, Hiroaki Kobayashi

Tohoku University
6/Dec/2016
Introduction

• Increase of variety of HPC systems
  – Scalar-type system
    • A large number of scalar processors with many cores and large cache
    • Massively parallel calculations
  – Accelerator-type system
    • Accelerators with lots of simple cores
    • Data parallel calculations
  – Vector-type system
    • Vector cores with a high memory bandwidth
    • calculates set of data elements at one time

System-dependent information has to be written in a code to exploit the potential of each HPC system
To Write System-Dependent Information

• By effectively utilizing the features of each HPC system
  – Directly writing system-dependent information
    • Low level programming, CUDA/OpenCL, AVX intrinsic, ...
    • Several versions of an application code (or IF/DEF HELL)
      – Each version is optimized for an HPC system
  – Directive-based programming
    • OpenMP, OpenACC, compiler-specific directives, etc
    • Only one version of a code
      – Multiple directive sets can be written in a code
Motivation

• Pitfalls: Directive-based approach is
  – High productivity
  – High maintainability
  ➔ Can it be REALLY true?

• Single version of a code for various HPC systems
  – Different kinds of directive sets has to be inserted
    • OpenMP, OpenACC, compiler-specific, etc
  – Different implementations using the same directive set
    • OpenMP parallel vs OpenMP task
    • ACC kernels vs ACC parallel, etc
Motivating Example: Himeno Kernel

• Three implementations in one code
  – OpenMP parallel
  – OpenACC kernels
  – OpenACC parallel
• Many lines spent for directives
  – 46% of code lines
    • 26 lines of the whole 56 lines
• NOT allowed different impl. using the same directive set
  – OpenACC kernels and parallel
Objective and Approach

• Objective
  – Easily utilize multiple kinds of directive sets

• Approach
  – Generation of various kinds of directive sets using user-defined rules and a special placeholder
    • Flexible and easy generation
      – Xevolver: Code translation framework
      – Xevtgen: Fortran-like rule generator
    • Keep maintainability
      – The original code as unchanged as possible
**Xevolver**: Code Translation Framework

- **Xevolver** [Takizawa, 2014]
  - Can separate system-awareness from a code as user-defined rules
  - Minimize modifications and then keep maintainability.

![Diagram of the Xevolver framework showing original code, src2xml, XML AST, user-defined transformation, XML AST, translation rules, and transformed code.](image-url)
**Xevtgen**: Translation Rule Generator

- **Xevtgen** [Suda, 2015]
  - Easily generation of translation rules for Xevolver
    - *Dummy Fortran code*
      - Fortran-like code with some special tgen directives
    - *source* and *destination patterns* of a dummy Fortran code

Standard Fortran programmers can easily learn and generate rules

**Source pattern**

```fortran
!$xev tgen src begin IF (I .EQ. 0) EXIT !$xev tgen src end
```

**Destination pattern**

```fortran
!$xev tgen dst begin IF (I == 0) THEN EXIT END IF !$xev tgen dst end
```
Proposed Directive Generation

• Key ideas
  – Directive generation by Xevolver framework
    • User-defined rules
      – Appropriate directives for individual HPC systems
    • Special placeholders
      – Specify a unique code pattern where several directives are inserted
      – Trigger directive generation

Multiple directive sets can be replaced into a smaller number of special placeholders
Overview of the Proposed Generation

Original code

Insertion a placeholder

Xev
!$xev par ...

To omp directives

OpenMP
!$omp par ...
!$omp end par

OpenACC
!$acc ker ...
!$acc end ker

Compiler
!DIR$ par ...
!DIR$ end par

To acc directives

To compiler pragmas

User-defined rules

Compiler ACC rule

OMP rule
!$xev to OpenMP

To compiler pragmas
Ex. Directive Generation of Himeno

• Translation triggered by "!$xev jacobi" placeholder

Original

```c
!$xev jacobi

do loop=1,nn
  gosa= 0.0
  do k=2,kmax-1
    do j=2,jmax-1
      do i=2,imax-1
        s0=a(I,J,K,1)*p(I+1,J,K) &
        ...
        wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
      enddo
    enddo
  enddo
enddo
```

OpenMP version

```c
!$OMP PARALLEL SHARED () &
!$OMP PRIVATE
  do loop=1,nn
    gosa= 0.0
    !$OMP DO
      do k=2,kmax-1
        do j=2,jmax-1
          do i=2,imax-1
            s0=a(I,J,K,1)*p(I+1,J,K) &
            ...
            wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
          enddo
        enddo
      enddo
    !$OMP END DO
  !$OMP END PARALLEL
```

ACC kernels version

```c
!$acc data present () &
!$acc present
  do loop=1,nn
    gosa= 0.0
    !$acc kernels
      !$acc loop vector(256) reduction(+:gosa)
      do i=2,imax-1
        s0=a(I,J,K,1)*p(I+1,J,K) &
        ...
        wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
      enddo
    !$acc end kernels
  !$acc end present
```

ACC parallel version

```c
!$acc data present () &
!$acc present
  do loop=1,nn
    gosa= 0.0
    !$acc parallel loop collapse(3)
      do k=2,kmax-1
        do j=2,jmax-1
          do i=2,imax-1
            s0=a(I,J,K,1)*p(I+1,J,K) &
            ...
            wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
          enddo
        enddo
      enddo
    !$acc end parallel loop
```

WSSP24
Easy Rule Generation by Xevtgen

Almost similar cost to writing directives into the original code
Experimental Environments

• Himeno benchmark
  – Original OpenMP
  – OpenMP parallel
  – OpenACC kernels
  – OpenACC parallel

• Environments
  – Intel Xeon E5-2630 x 2sockets
  – Nvidia Tesla K20
  – PGI compiler 16.4
  – Xevolver ver. 1.1.0-1.1.0 + Xevtgen gamma2
Comparisons of Modified Code Lines

<table>
<thead>
<tr>
<th>Himeno version</th>
<th>#. directives</th>
<th>#. code lines</th>
<th>#. rule lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP parallel</td>
<td>23</td>
<td>427</td>
<td>0</td>
</tr>
<tr>
<td>OpenACC kernels</td>
<td>19</td>
<td>423</td>
<td>0</td>
</tr>
<tr>
<td>OpenACC parallel</td>
<td>16</td>
<td>420</td>
<td>0</td>
</tr>
<tr>
<td>MP and ACC</td>
<td>51</td>
<td>455</td>
<td>0</td>
</tr>
<tr>
<td>Placeholder</td>
<td>3</td>
<td>407</td>
<td>118+114+111</td>
</tr>
</tbody>
</table>

- **Proposed approach**
  - Minimum number of directive lines in the code
  - Affordable additional lines of user-defined rules
    - The writing cost is the almost similar to the inserting cost of directives into the code
Performance comparisons

- CPU: MP and ACC versions are Faster than the original version
  - Aggressive optimization using MASTER thread and barrier leads degradation
- GPU: ACC parallel version is faster than the others
  - Suitable parameters for parallelization such as grid sizes are selected

The proposed approach can choose the best one!
Conclusions

• Multiple directive sets for exploiting the potential of various HPC systems
  – Decreases maintainability and productivity

• Directive generation approach by code translation
  – User-defined rules by Xevtgen rule generator
    ➔ Flexible and easy generation
  – A special placeholder
    ➔ Keep the original as unchanged as possible

• Future work
  – More detailed evaluation using practical applications
    • The number of placeholders and code lines of dummy codes
    • The positions of placeholders might be different among platforms