Xevolver

TOHOKU UNIVERSITY

Vectorization-aware Loop Optimization with User-defined Code Transformations

Re-Emergence of Vector Architectures

(REV-A Workshop)

Sept 5, 2017@Honoluli, Hawaii

Hiroyuki Takizawa, Thorsten Reimann, Kazuhiko Komatsu,

Takashi Soga, Ryusuke Egawa, Akihiro Musa, and Hiroaki Kobayashi

Background

Computer Architecture Design **HPC Application Development**

Make Common Case Fast

- What is the **fast case** of vector architecture?

→ **Vectorized** long loops!

- Another system/compiler/programmer may require/prefer a different loop structure.

Separate vectorization-awareness from application codes

How Is Code Modified?

TOHOKU UNIVERSITY

- **Bad News -- Messy**
  - **Vectorization-aware code modifications are scattered over a code**
- **Good News -- Repetitive**
  - **Same (or similar) code modifications are required many times**

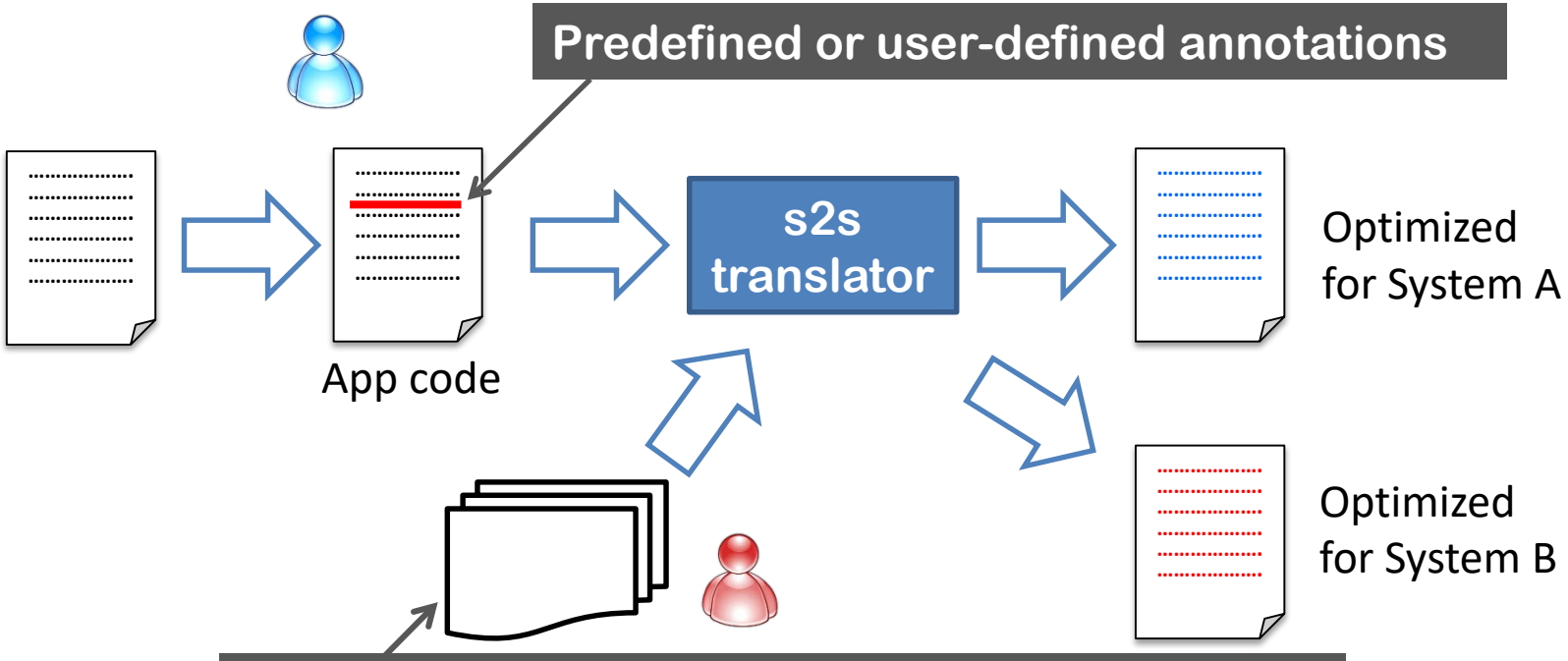
Manual code modifications can be replaced with a smaller number of mechanical code transformations.

→ Express application-specific and/or system-specific code modifications as mechanical code transformations

Xevolver Framework

Various transformations are required for replacing arbitrary code modifications.
= cannot be expressed by combining predefined transformations.

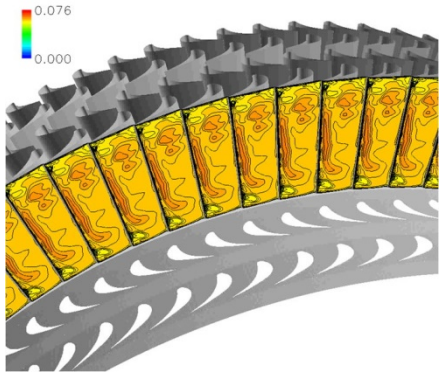
→ **Xevolver : a framework for custom code transformations**

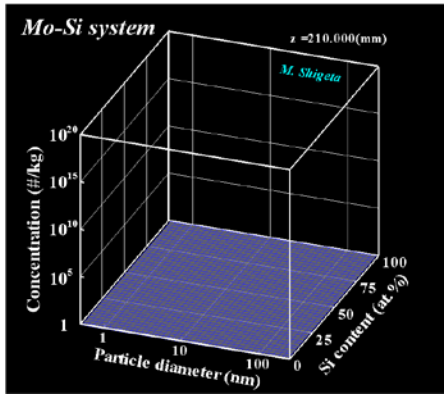


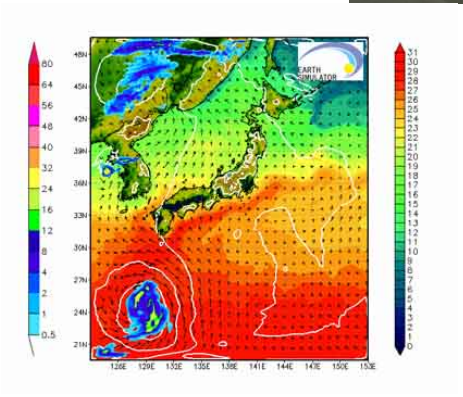
Takizawa et al@HiPC2014.

Case Studies with Real Applications

- Real-world applications originally developed for NEC SX-9 have been ported to OpenACC.
  - Numerical Turbine (Yamamoto et al@Tohoku-U)
  - Nano-Powder Growth Simulation (Shigeta@Osaka-U)
  - MSSG-A (Takahashi et al@JAMSTEC)





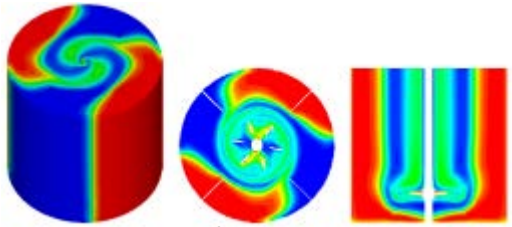


Is our approach effective for vectorization-aware loop optimizations?
- There are clear patterns in such code modifications!
  = Xevolver should be helpful for expressing vectorization-awareness.

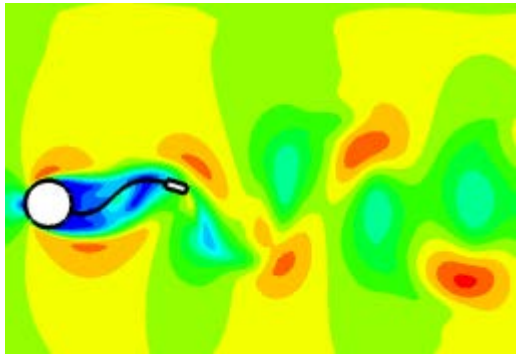
This Work

- Case Study: The FASTEST incompressible flow solver
  - Originally developed in the 1990's at FAU, Germany
    - SPPEXA ExaFSA version developed at Technical University Darmstadt
      - http://www.fnb.tu-darmstadt.de/forschung_fnb/software_fnb/software_fnb.en.jsp
    - Once written for old vector machines
      → Some kernels already have their default and vector versions

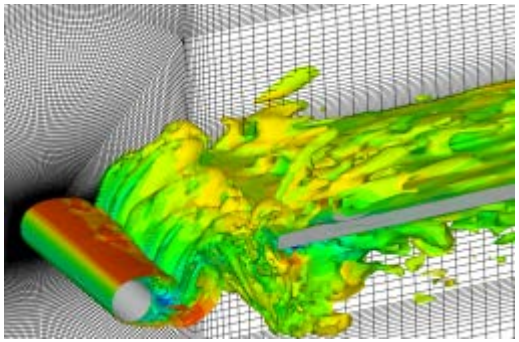
Turbulent flow around a cylinder



Mixing simulation



Fluid-structure interactions



3D flow around cylinder-plate configuration

What is the difference btwn default and vector versions?

- Loop collapse and its variants
  - Vector versions have different loop structures to enable vectorization and increase their loop lengths.

Standard version

```
1      real a(n,n,n),b(n,n,n)
2      do k=2,nkm
3        do i=2,nim
4          do j=2,njm
5            a(j,i,k) = a(j,i,k)+b(j,i,k)
6          end do
7        end do
8      end do
```

Vector version

```
1      real a(n,n,n),b(n,n,n)
2      do kij=0,(nke-1)*(nie-1)*(nje-1)-1
3        k=kij/((nie-1)*(nje-1))+2
4        i=mod(kij,(nie-1)*(nje-1))/(nje-1)+2
5        j=mod(kij,(nje-1))+2
6        a(j,i,k) = a(j,i,k)+b(j,i,k)
7      end do
```

What is the difference btwn default and vector versions?

- Loop collapse and its variants
  - Vector versions have different loop structures to enable vectorization and increase their loop lengths.

Transform

Standard version

```
eal a(n,n,n),b(n,n,n)
o k=2,nkm
 do i=2,nim
   do j=2,njm
     a(j,i,k) = a(j,i,k)+b(j,i,k)
   end do
 end do
end do
```

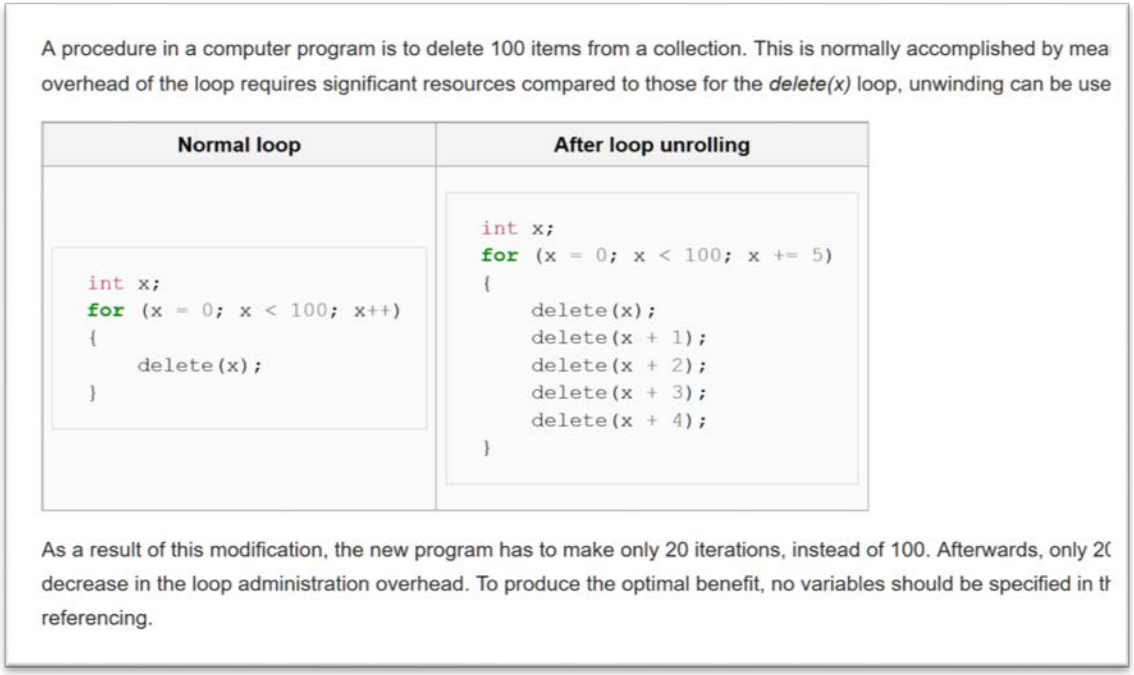
The compiler used in this case study does not collapse this loop structure.

Vector version

```
real a(n,n,n),b(n,n,n)
do kij=0,(nke-1)*(nie-1)*(nje-1)-1
  k=kij/((nie-1)*(nje-1))+2
  i=mod(kij,(nie-1)*(nje-1))/(nje-1)+2
  j=mod(kij,(nje-1))+2
  a(j,i,k) = a(j,i,k)+b(j,i,k)
end do
```

How to Describe Code Transformation

- One common way to explain a code transformation is to show **before-and-after versions**
  - e.g. Loop unrolling (screen capture of en.wikipedia.org)



A procedure in a computer program is to delete 100 items from a collection. This is normally accomplished by mea overhead of the loop requires significant resources compared to those for the *delete(x)* loop, unwinding can be use

Normal loop	After loop unrolling

```
int x;
for (x = 0; x < 100; x++)
{
    delete(x);
}
```

```
int x;
for (x = 0; x < 100; x += 5)
{
    delete(x);
    delete(x + 1);
    delete(x + 2);
    delete(x + 3);
    delete(x + 4);
}
```

As a result of this modification, the new program has to make only 20 iterations, instead of 100. Afterwards, only 2(decrease in the loop administration overhead. To produce the optimal benefit, no variables should be specified in tl referencing.

Users do not need any special knowledge (e.g. XML and AST) to describe a code transformation.

Simple Loop Collapse Rule


```
1  !$xev tgen var(nke,nje,nie) exp
2  !$xev tgen list(flxc_stl1) stmt
3  !$xev tgen trans stmt src begin
4  !$xev vector flxc begin
5      do k=2,nke
6        do i=2,nie
7          do j=2,nje
8  !$xev tgen stmt(flxc_stl1)
9          end do
10       end do
11     end do
12 !$xev vector fl
13 !$xev tgen trans stmt src end
14
15 !$xev tgen trans stmt dst begin
16 !$xev vector flxc begin
17 #include "vdircnodep.h"
18 do kij=0,(nke-1)*(nie-1)*(nje-1)-1
19   k=kij/((nie-1)*(nje-1))+2
20   i=mod(kij,(nie-1)*(nje-1))/(nje-1)+2
21   j=mod(kij,(nje-1))+2
22 !$xev tgen stmt(flxc_stl1)
23     end do
24 !$xev vector flxc end
25 !$xev tgen trans stmt dst end
```

Original Loop Structure

All statements in the loop body are copied.

Transformed Loop Structure

Code patterns are written in the rule → **easily customizable** for individual cases

Custom Loop Collapse Rule

- Collapsing imperfectly-nested loops

```
1  !$xev tgen list(vint_stl1) stmt
2  !$xev tgen trans src begin
3  !$xev vector vint begin
4    do  kcg=1,nkmg
5      kfg=2*kcg-1
6      do  icg=1,nimg
7        ifg=2*icg-1
8        do  jcg=1,njmg
9          jfg=2*jcg-1
10 !$xev tgen stmt(vint_stl1)
11       end do
12     end do
13   end do
14 !$xev vector vint end
15 !$xev tgen trans src end
16
17 !$xev tgen trans dst begin
18 !$xev vector vint begin
19 #include "vdirnodep.h"
20   do kijcg=0,nkmg*nimg*njmg-1
21     kcg=kijcg/(nimg*njmg)+1
22     icg=mod(kijcg,(nimg*njmg))/njmg+1
23     jcg=mod(kijcg,njmg)+1
24
25     kfg=2*kcg-1
26     ifg=2*icg-1
27     jfg=2*jcg-1
28 !$xev tgen stmt(vint_stl1)
29   end do
30 !$xev vector vint end
31 !$xev tgen trans dst end
```

Original Loop Structure

There is a statement between loops.

Transformed Loop Structure

Any statements can be inserted to avoid changing the code behaviors.

Experimental Setup

	NEC SX-ACE	Intel Xeon E5-2695v2
Peak Performance [Gflop/s]	256/socket, 64/core	230.4/socket, 19.2/core
Number of cores	4	12
Cache size	1MB/core	L2:256KB/core, L3:30MB/socket
Memory bandwidth [GB/s]	256	59.7
Vector length/ SIMD width (double)	256	4
Compiler	NEC SX/Fortran 111	Intel Compiler 16.0.3

User-defined Code Transformations

- 19 kernels of the original FASTEST code have their default and vector versions.
  - The difference between the two versions is expressed as a code transformation.
- 10 rules are defined for converting the 19 kernels
  - **Most of them are for collapsing loop nests.**
  - **A simple loop collapse rule is customized for defining other special rules.**
  - **Some rules can be reused for converting multiple loops.**

name	calcdp	calcp	celp2	flxc	vint	flxcoa	sipsol1	sipsol2	sipsol4	celuvw1
# loops	1	5	2	3	1	3	1	1	1	1

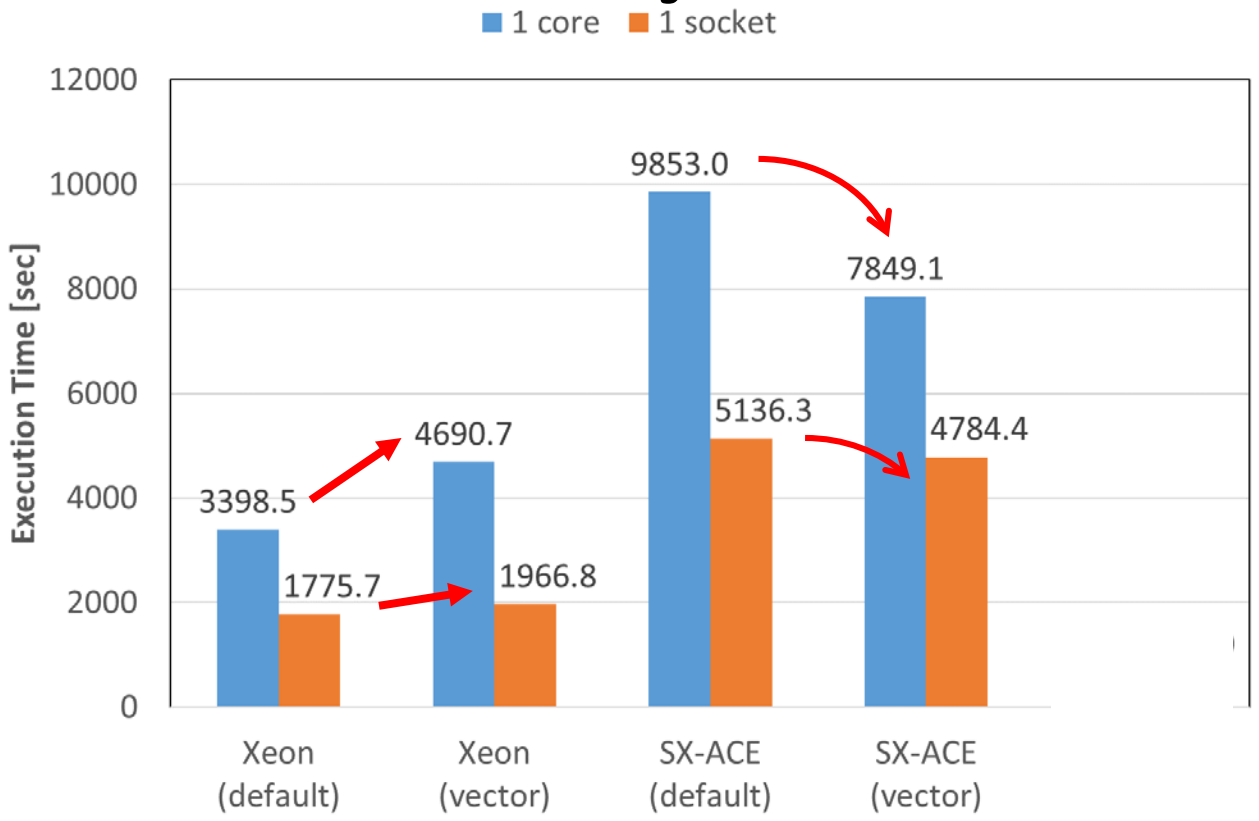
- Improving code maintainability
  - **Only the default version is kept, and the vector version is removed from the code.**
  - **The default version of each kernel is converted to its vector version using Xevolver when it is compiled for vector systems.**

Performance Evaluation Results

- **19** kernels already have their vector versions

  →The SX performance increases by using vector versions.

  = The other kernels may not be vectorized.



Obstacles for Vectorization


```
1   program main
2   ...
3   do i=1,ni
4     do j=1,nj
5     call foo(x,j,mode)
6     ...
7     enddo j
8   enddo i
9   ...
10  end program main
11
12  subroutine foo(x,itr,iflag)
13    integer x
14    integer itr
15    integer iflag
16
17    if (iflag == 1) then
18     x = itr
19    else if (iflag == 2) then
20     x = itr*2
21    end if
22  end subroutine foo
```

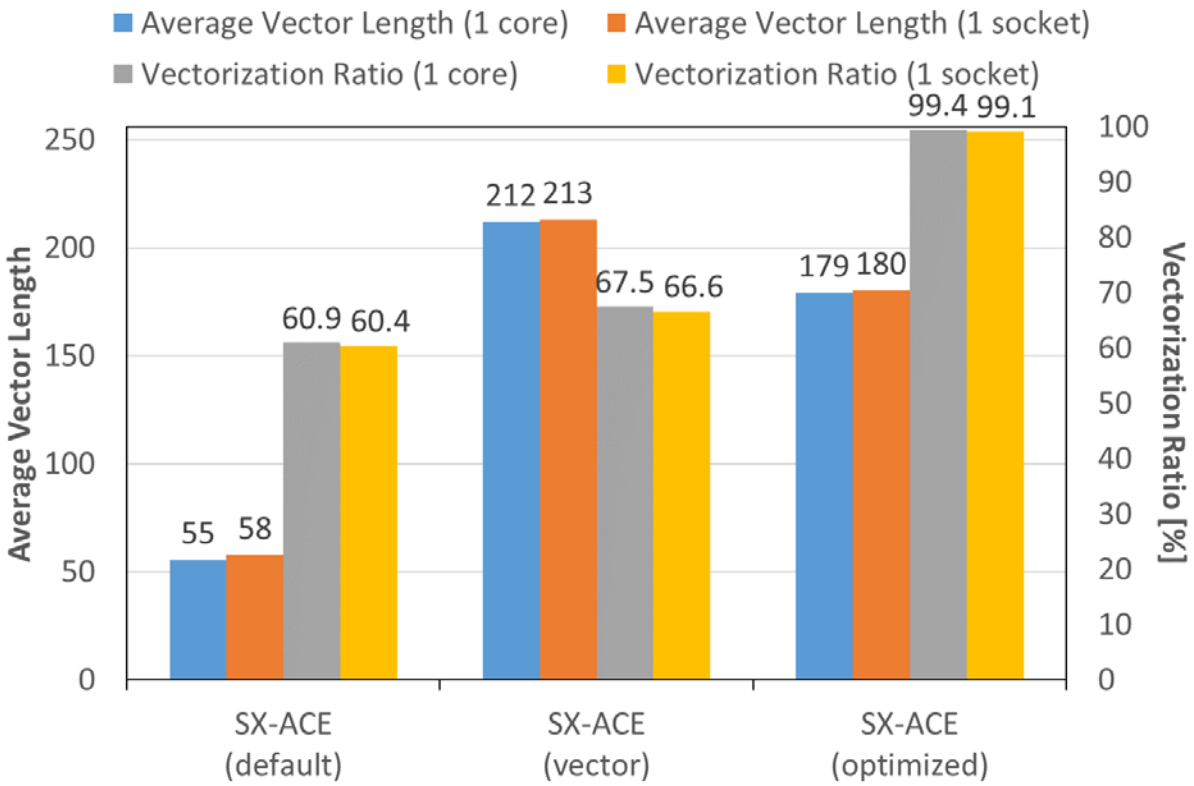
This loop nest is not vectorized.

The value of x is undefined if iflag is neither 1 nor 2.

This part is removed to prevent undefined variables.

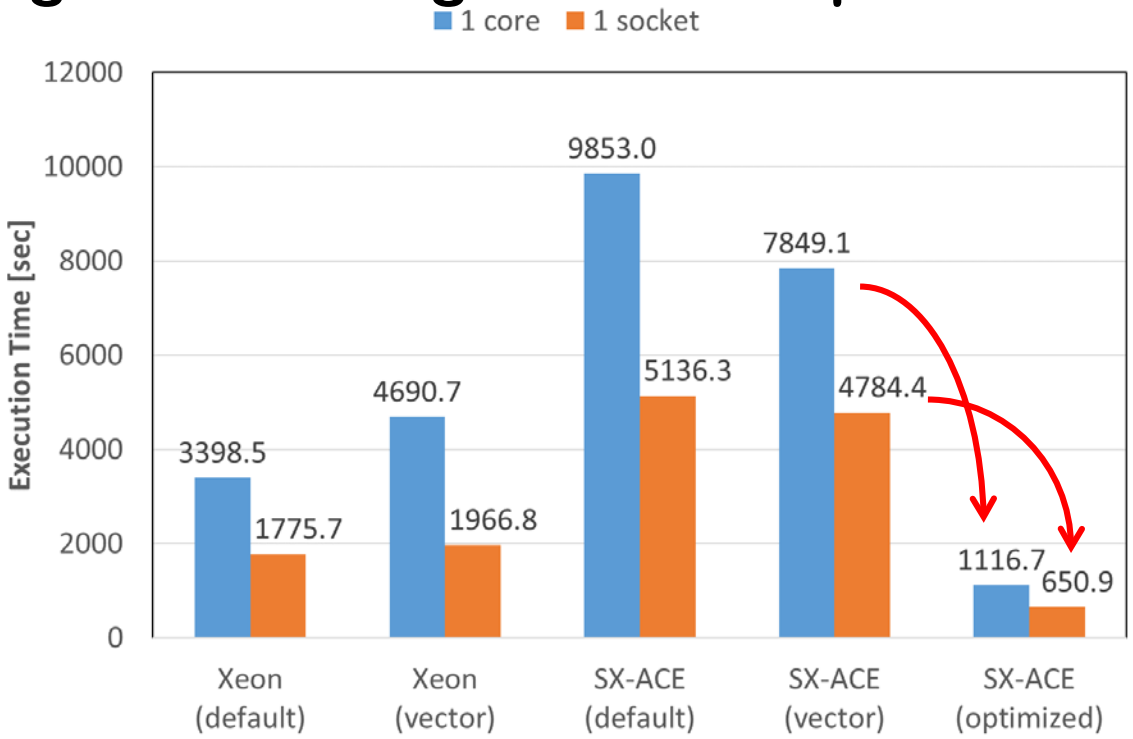
Vectorization Ratio

- Most loops are vectorized by minor modifications.
  - Obstacles for vectorization are simply removed.
  - System-specific optimizations are not applied to avoid degrading the maintainability.



Finally Achieved Performance

- **Most** kernels are now vectorized
  - = **Significant SX performance improvement**.
  - →Vectorization ratio is very important for SX-ACE.
  - →Average vector length is less important.



Conclusions

TOHOKU UNIVERSITY

- Xevolver framework
  - Vectorization-aware loop optimizations are separated from application codes.
    - Application developers can maintain the original code
    - System-specific optimizations are defined in an external file

- The right one of

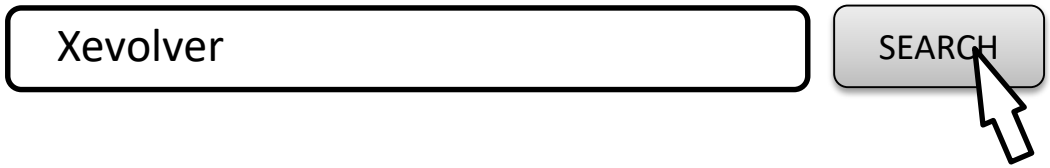
  code refactoring or code transformation

  should be used for the right purpose.
  - Xevolver provides the latter option.
  - An application code should be refactored if the refactoring could improve the maintainability.

→ We need to further explore the best practices of code optimizations with user-defined code transformations.

Acknowledgements

- We would like to thank Kenta Yamaguchi and Yoichi Shimomura for their contributions to this work .
- This work was partially supported by JST Post-Peta CREST, DFG SPPEXA ExaFSA project, and Grant-in-Aid for Scientific Research(B) 16H02822.



Xevolver with some sample translation rules is online available at http://xev.sc.cc.tohoku.ac.jp.

Your feedbacks (and bug reports) are welcome!