Performance Tuning of Ateles using Xevolver

Kazuhiko Komatsu
Tohoku University
11 October, 2017
WSSP26

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
  - TOP500-specialized system

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

Problems

- By effectively utilizing the features of each HPC system
  - Directly writing system-dependent information
    - C, Fortran, CUDA/OpenCL, AVX intrinsic, …
    - **Several versions** of an application code (or IF/DEF HELL)
      - Each version is optimized for an HPC system
    - **Low readability** and **maintainability**
      - System-dependent optimizations may be obstacle of application behavior

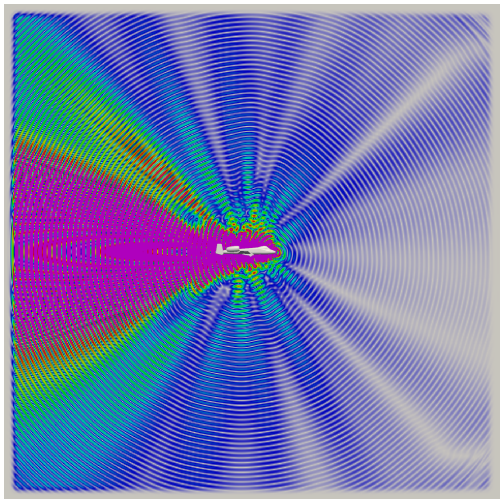
Objective and Approach

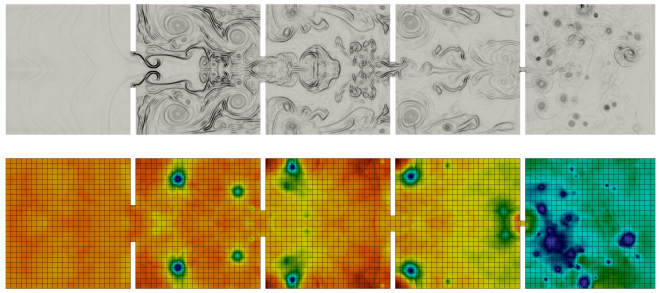
- Objective
  - System-dependent optimizations while keeping readability and maintainability

- Approach
  - Utilize code transformations for system-dependent optimizations
    - ***Xevolver***: Code transformation framework
    - ***Xevtgen***: Fortran-like rule generator
      - The original code as unchanged as possible

Case study: Tuning of Ateles for SX-ACE

- Overview of Ateles
  - One of solvers in the APES simulation suite
    - Fortran 2008/2003/95/90, python, OpenMP/MPI
      - ISO C interface
      - function pointer
      - allocatable variables in structures
  - Discontinuous Galerkin Solver
    - Fortran 100K lines
    - Python 3.5K lines





Preliminary Evaluation on SX-ACE

- Test data
  - testsuite/sx_testing/ateles.lua
- Parameter survey in the script

```
-- Parameters to vary --
--
-- The polynomial degree specifies the accuracy within each element and defines
-- the number of degrees of freedom per element that will be (degree+1)^3 for
-- a poly space of Q
degree = 7

-- ...the uniform refinement level for the periodic cube
-- (There will be 8^level elements in the mesh)
level = 2
```

  - “level” does not affect the performance
  - “degree” might affect the vector length

Single-node performance

	degree = 7 | level = 2	degree = 14 | level = 2
Real Time (sec)	83.176829	811.614073
User Time (sec)	82.835361	811.393898
Sys Time (sec)	0.043444	0.062352
Vector Time (sec)	12.100865	134.764311
Inst. Count	25562226844	542853655029
V. Inst. Count	799528202	8621373999
V. Element Count	61705955881	1262989794226
V. Load Element Count	7714451369	126354804785
FLOP Count	9218093212	207136982518
MOPS	1043.861625	2214.981009
MFLOPS	**111.282103**	**255.285359**
A. V. Length	**77.17796**	**146.495187**
V. Op. Ratio (%)	**71.362225**	**70.274554**
Memory Size (MB)	256.03125	320.03125
MIPS	308.590758	669.038375
I-Cache (sec)	0.266876	1.291097
O-Cache (sec)	56.223546	377.723006
Bank Conflict Time		
CPU Port Conf. (sec)	0.331255	8.065057
Memory Network Conf. (sec)	3.205328	45.384267
ADB Hit Element Ratio (%)	62.779898	51.393439

Detailed cost distribution

degree = 7 | level = 2

PROC. NAME	FREQ.	EXCL. TIME[sec]	%	AVER.TIME [msec]	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	BANK CONFLICT		ADB HIT
									CPU	NW PORT	ELEM. %
MODG_VOLTOFACE_Q	4584	69.65	**83.8**	15.194	**25.9**	**53.85**	151.1	1.572	0	0	99.99
MODG_PRJ_PFLUX3_Q_6	48896	3.009	3.6	0.062	297.7	98.96	38.5	2.986	0.05	1.671	84.29
MAXWELL_FLUX_CUBE_VEC	2292	2.403	2.9	1.048	214.9	80.25	39.1	2.181	0.009	0.043	63.24
MODG_PRJ_PFLUX2_Q_6	48896	1.594	1.9	0.033	562	98.86	38.6	1.563	0.061	0.254	81.99
MODG_PRJ_PFLUX1_Q_6	48896	1.518	1.8	0.031	589.8	98.87	38.8	1.476	0.03	0.165	79.93

degree = 14 | level = 2

PROC. NAME	FREQ.	EXCL. TIME[sec]	%	AVER.TIME [msec]	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	BANK CONFLICT		ADB HIT
									CPU	NW PORT	ELEM. %
MODG_VOLTOFACE_Q	16080	665.772	**82.2**	41.404	**62.6**	**48.56**	245.9	16.892	0	0	100
MODG_MAXWELL_PHYSFLUX_CONST	514560	23.831	2.9	0.046	0	95.6	238.3	23.027	3.559	17.651	0
MODG_PRJ_PFLUX3_Q_6	171520	23.159	2.9	0.135	1080.5	99.74	130.1	23.075	1.499	8.046	77.57
MODG_PRJ_PFLUX1_Q_6										401	70.25
MODG_PRJ_PFLUX2_Q_6										.96	79.31

- Cost distributions in both degrees are similar
- Low vector operation ratio in the highest routine

Analysis of the routine

MODG_VOLTOFACE_Q

```
  :
2929: +------>      do iAnsX = 1, maxPolyDegree+1
2930: |             ! get the face value of the ansatz function with fixed coordinate
2931: |             !!faceVal = faceValRightBndAns(iAnsX)
2932: |
2933: |+----->       do iVar = 1,nScalars
2934: ||+---->        do iElem=1,nElems
2936: |||V--->         do facepos = 1,mpd1_square
2937: ||||             ! get position of the current ansatz function
2938: ||||             pos = (facepos-1)*mpd1 + iAnsX
2939: ||||    A         faceState(iElem,facePos,iVar,leftOrRight) &
2940: ||||               &  = faceState(iElem,facePos,iVar,leftOrRight) &
2941: ||||               &  + volState(iElem,pos,iVar)
2942: |||V---          end do
2944: ||+----        end do
2945: |+-----       end do
2946: |
2947: +------      end do
  :
```

Loop Length= ~7

Loop Length= ~6

Loop Length= ~64

Loop Length= ~64

- Compile message

2936: vec(2): **Partially** vectorized loop.
2936: vec(24): Iteration count is assumed. Iteration count=5000.
2936: vec(25): Work vectors are used. Size=40000byte.

- The innermost loop is vectorized, but PARTIALLY
  - Data dependency cannot be solved by the compiler
- The length of the vectorized loop is SHORT

Optimizations effective for SX-ACE

```
MODG_VOLTOFACE_Q
  :
2930: +------> do iAnsX = 1, maxPolyDegree+1
2931: |            ! get the face value of the ansatz function with fixed coordinate
2934: |+----->    do iVar = 1,nScalars
2938: ||             m=nElems
2939: ||             n=mpd1_square
2940: ||             mn=m*n
2941: ||         !cdir nodep
2942: ||V---->       do ij= 0, mn - 1
2943: |||               iElem=ij / n + 1
2944: |||               facepos=mod(ij, n) + 1
2945: |||                ! get position of the current ansatz function
2946: |||                pos = (f
2947: |||    A            faceSta
2948: |||                 &  = fa
2949: |||                 &  + vo
2950: ||V----        end do
2953: |+-----      end do
2955: +------   end do
  :
```

Loop Length= ~7

Loop Length= ~6

Loop Length= ~64 * 64

- Compile message

2942: vec(1): Vectorized loop.
2947: vec(23): "NODEP" is specified. No dependency is assumed.: FACESTATE
2942: vec(29): ADB is used for array.: __REAL(KIND=8)

- The length of the vectorized loop increases
  - By collapsing both i loop and j loop
- Successfully vectorized by the “nodep” directive
- Applied for all the same loop nests in the routine

Motivation

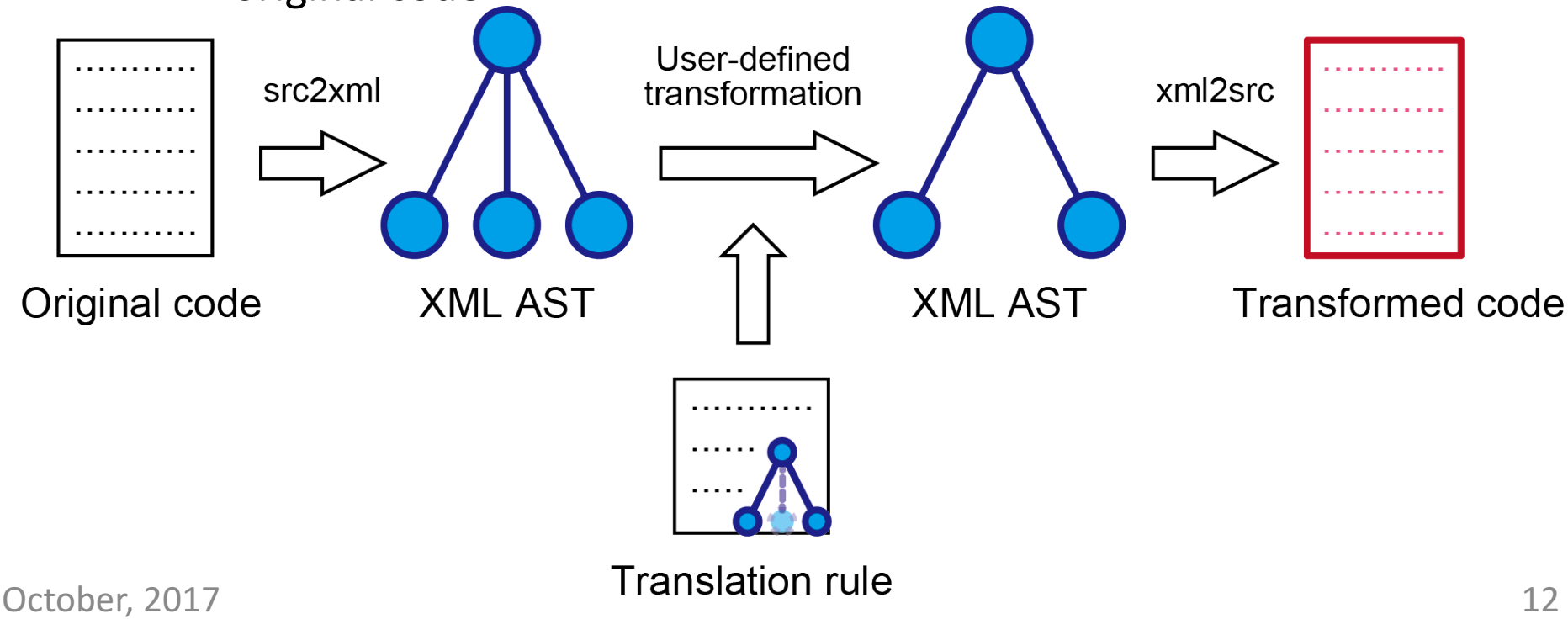
- System-aware optimization for SX-ACE
  - Loop collapse and compiler-specific directives
    - Degrade readability
  - Similar optimizations need to be REPEATEDLY applied
    - Loop collapse and insertion of the directive is applied all loop nests in whole code
    - Might lead human errors

The repetitive optimization can be replaced with transformation rules and code transformations

Xevolver: code transformation framework

- Xevolver [Takizawa, 2014]
  - Can separate system-awareness from a code
    - External custom translation rules and custom directives

    ➔ Minimize modifications and then keep maintainability of the original code



Xevtgen: Transformation Rule Generator

- Xevtgen [Suda, 2015]
  - Easily generation of transformation 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

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

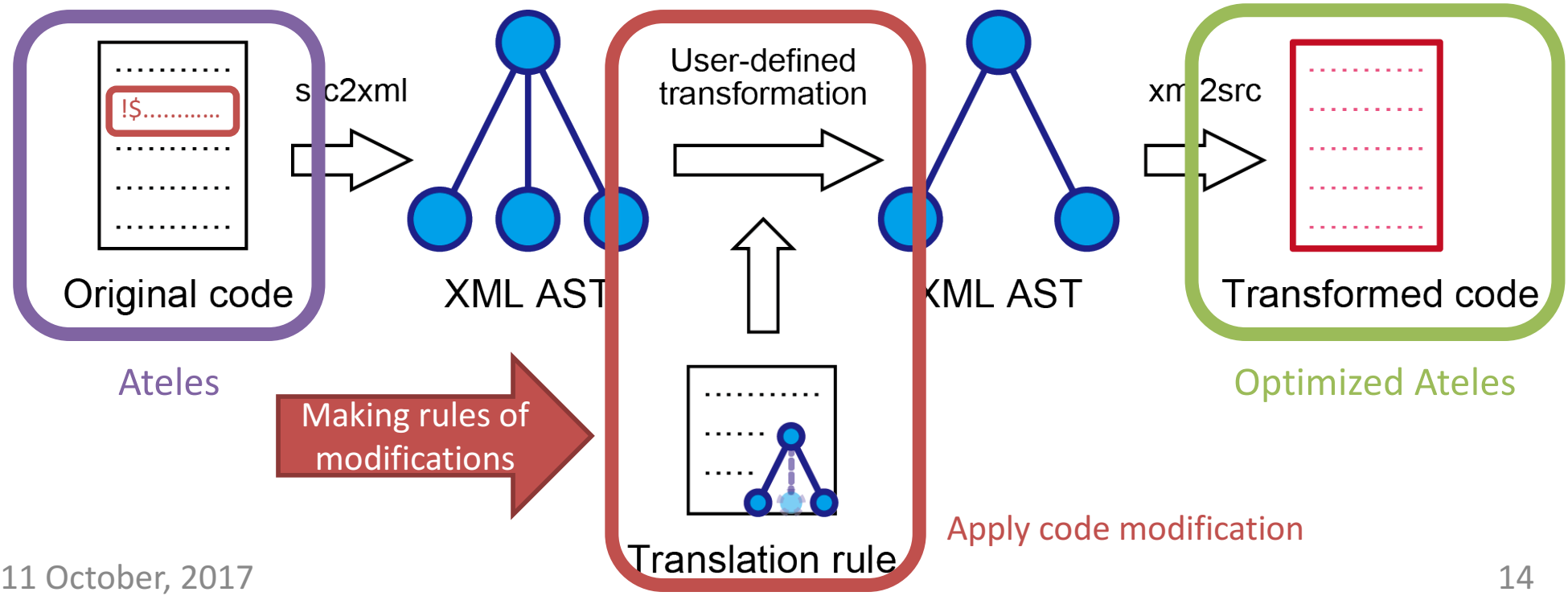
```
!$xev tgen dst begin
IF (I == 0) THEN
EXIT
END IF
!$xev tgen dst end
```

Destination pattern

Optimization using Xevolver

– Instead of directly modifying a code, custom rules and custom directives are defined using Xevolver

➜Minimize modifications and then keep maintainability



Loop collapse and NODEP directive

Original + xev directive

```
!$xev collapse
do iVar = 1,nScalars
  do iElem=1,nElems
    do facepos = 1,mpd1_square
      …
    end do
  end do
end do
```

```
do iVar = 1,nScalars
  m=nElems
  n=mpd1_square
  mn=m*n
!cdir nodep
  do ij= 0, mn - 1
    iElem=ij / n + 1
    facepos=mod(ij, n) + 1
    …
  end do
end do
```

- Code Transformation by Xevolver
  - Just inserting "!$xev collapse" into the original code
    - Can keep the maintainability of the original code as much as possible

Transformation rule

Source pattern

```
!$xev tgen src begin
!$xev collapse
do iElem=1,nElems
  do facepos = 1,mpd1_square
!$xev tgen stmt(body0)
  end do
end do
!$xev tgen src end
```

Destination pattern

```
!$xev tgen dst begin
m=nElems
n=mpd1_square
mn=m*n
!cdir nodep
do ij= 0, mn - 1
  iElem=ij / n + 1
  facepos=mod(ij, n) + 1
!$xev tgen stmt(body0)
end do
!$xev tgen dst end
```

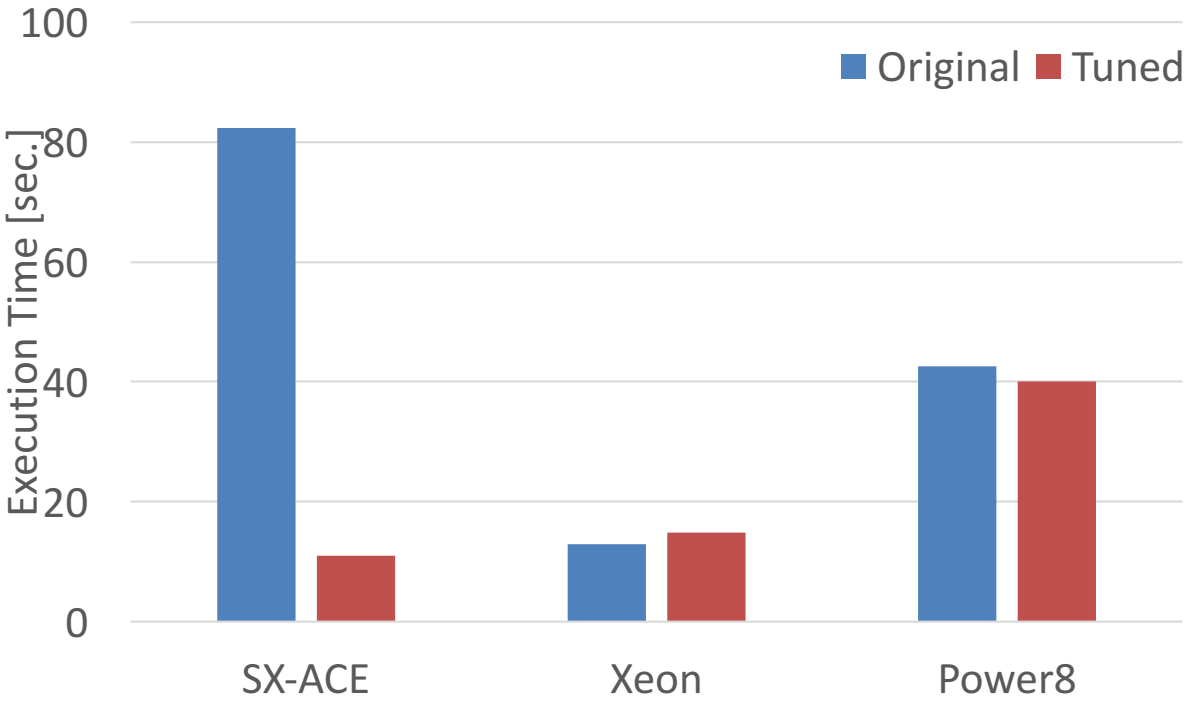
18 system-aware optimizations can be applied by code transformations

Experimental environment

- Comparison of performance of Ateles on various platforms
  - Original version
  - Tuned version for NEC SX-ACE
    - Loop collapse
    - Insertion of compiler-specific directives

Processor	Intel Xeon E5-2695v2	NEC SX-ACE	IBM Power8
#. of cores	2x 12 cores	4 cores	2x 8 cores
Mem. Capacity	128 GB	64 GB	512 GB
Compiler	Intel compiler 16.03	NEC SX 2003 Rev. 061	PGI 16.10 (openpower)

Results



- SX-ACE: x7.5 speedup to original, 18% to Intel original
- Xeon: Tuned version degrades 14%
- IBM: Tuned version accelerates 6%

Optimization for particular platform is not always effective.
Our approach that uses transformation is suitable!

Detailed analysis of loop collapse

- **degree=7**, level=2

	FREQ.	EXCL. TIME[sec]	%	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT		ADB HIT
												CPU	NW PORT	ELEM.%
BEFORE	4584	**69.650**	83.8	15.194	657.2	**25.9**	**53.85**	151.1	1.572	0.001	55.351	0	0	**99.99**
AFTER	4584	**2.117**	13.7	0.462	18677.2	**1135.1**	**99.83**	**256**	2.114	0.001	0.001	0.485	0.025	**73.51**

  - Vector operation ratios become more than 99%
  - Vector average lengths become more than 250

Conclusions

- Performance analysis and optimizations of Ateles on SX-ACE
  - Inline expansion
  - Loop collapse
  - ➔ Degrades the readability and maintainability
- For high maintainability and performance
  - Code transformation by Xevolver is employed
    - only small number of directives need to be inserted into the original code
- Future work
  - Increase of compatibility of Xevolver
    - The backend of compiler infrastructure does not fully support Fortran2003/2008
    - Need to modify Ateles to avoid the unsupported description