

Xevolver

Please visit our booth **#2315** (Tohoku University)
or **#2921** (The Japan Science and Technology agency)

Expressing system-awareness as code transformations
for performance portability across diverse HPC systems

SC15 workshop on Portability among HPC architecture for Scientific Applications

Nov 15, 2015@Hilton Austin

**Hiroyuki Takizawa, Shoichi Hirasawa, Kazuhiko Komatsu,
Ryusuke Egawa and Hiroaki Kobayashi
(Tohoku University/JST)**

BACKGROUND

- HPC application development = team work of programmers with different concerns



Application developers (= computational scientists)

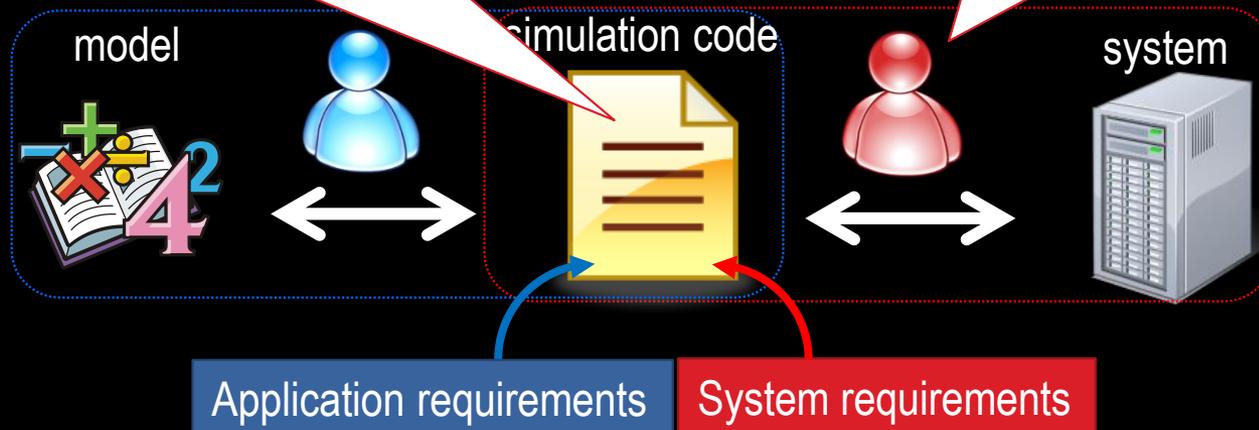
- write a program so as to get correct results

Assumption: Legacy Code

Application code written in C or Fortran already exists. **It is not allowed to rewrite the code** because application developers have to maintain the code.

Assumption: Oracle Tuner

Oracle Tuner knows how to adapt the code to a new target system.



WHAT'S THE PROBLEM?

- **System complexity is increasing**

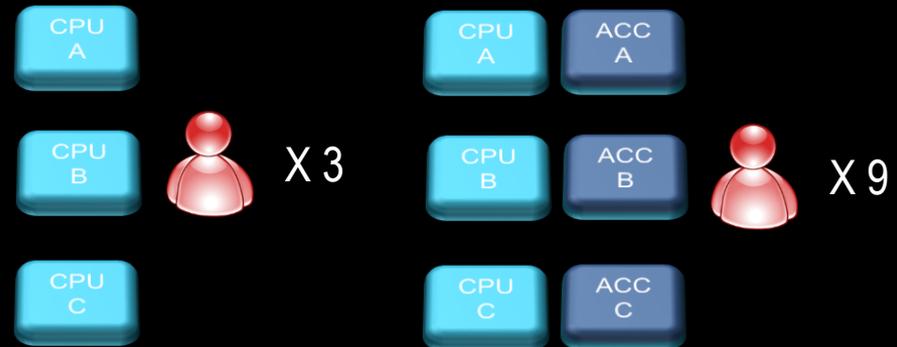
- Need to consider both parallelism and heterogeneity
- Also need to manage deeper memory/storage hierarchy, power, fault tolerance, ...

System-aware performance optimizations are needed for high performance

→ An HPC application is **specialized** for a particular system

- **System diversity is also increasing**

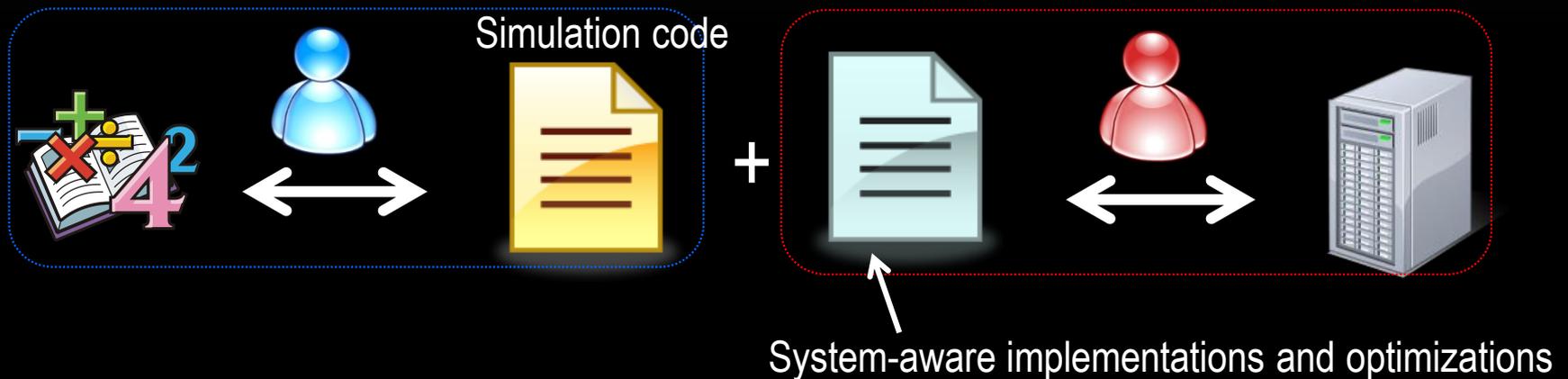
- Different processor combinations
- Different system scales
- Different interconnect network topologies
- Different system operation policies



What can we do to achieve high performance on various systems?

OUR GOAL = APPROPRIATE DIVISION OF LABOR

- **Separation of system-awareness from application programs**



There are many approaches to abstraction of system-awareness

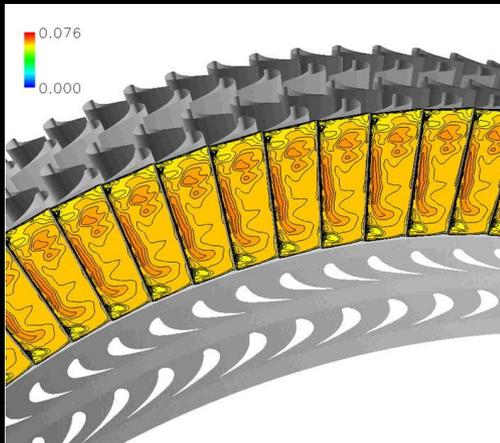
- System-aware implementations with a common interface = Numerical libraries
- Standardized programming models and languages = MPI, OpenMP, OpenACC ...

In reality, we still need to manually modify a code for high performance.

→ **How can we abstract such code modifications?**

A MOTIVATING EXAMPLE

- **Numerical Turbine (NT)**
 - Developed by Prof. Yamamoto (Tohoku U.)
 - 44 loop nests of the code have the same loop structure.
 - The loop nests are optimized for NEC SX-9 system.
 - OpenACC compiler cannot achieve high performance on GPUs.
 - Because of the same loop structure, all the loop nests need to be modified **in the same way** for GPUs.



HOW IS CODE MODIFIED?

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

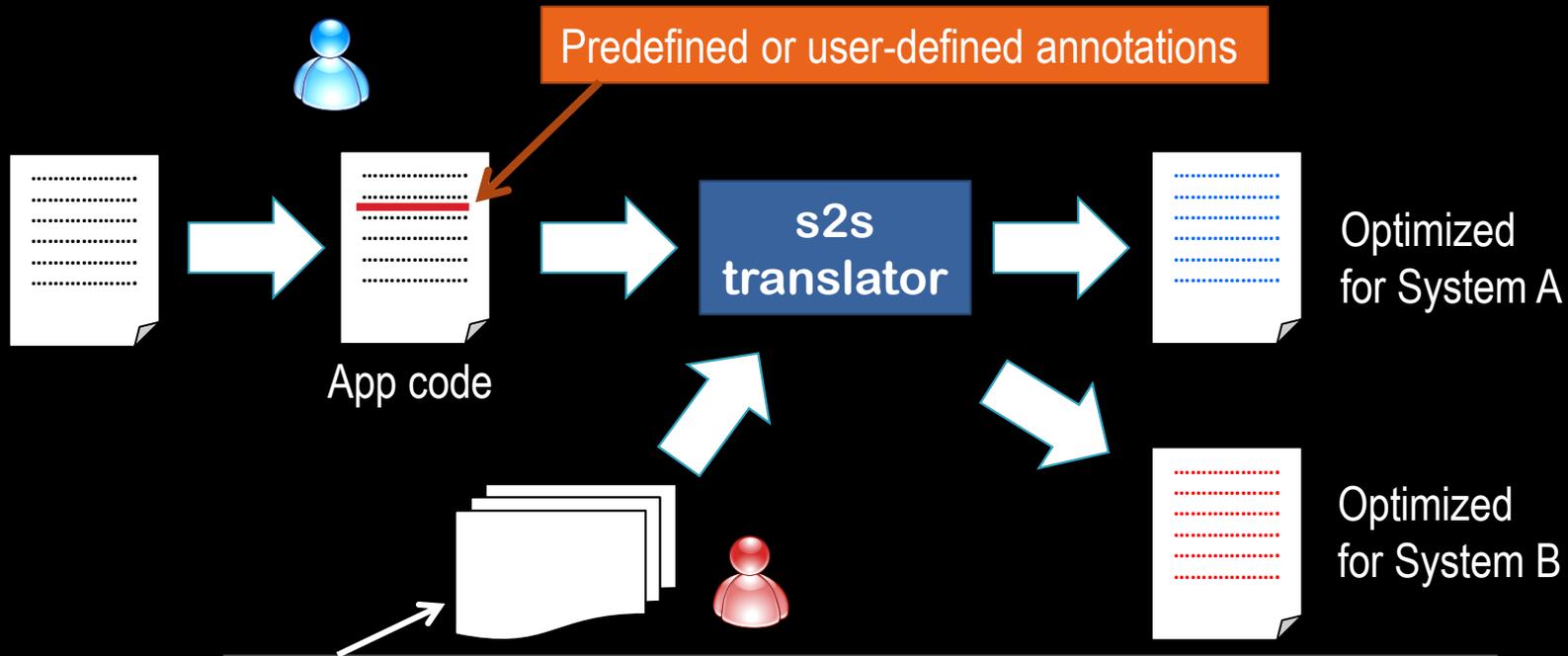
Manual code modifications can be replaced with a smaller number of mechanical code transformations.

→ 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.

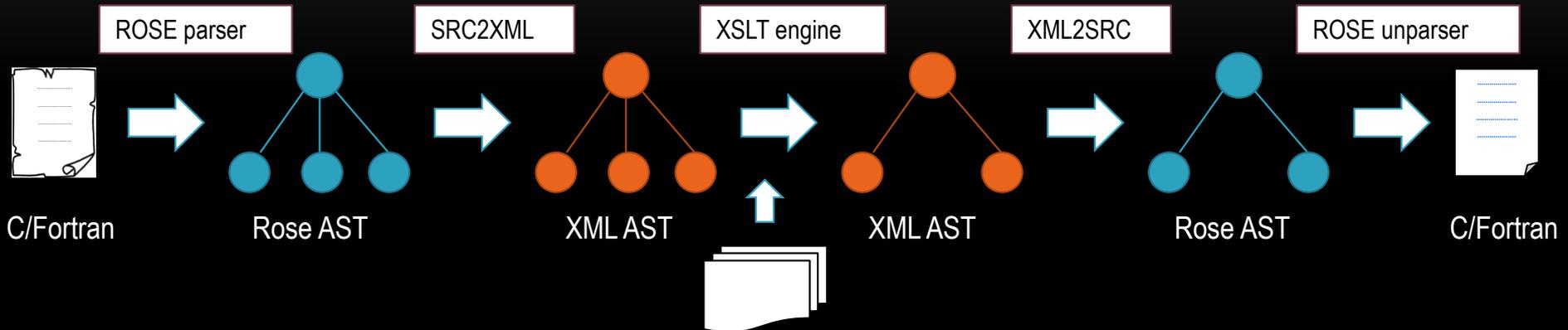
→ **Xevolver : a framework for custom code transformations**



Translation rules

- Define the code transformation of each annotation
- Different systems can use different rules
- **Users can define their own code transformations**

PROOF-OF-CONCEPT IMPLEMENTATION



- On top of the **ROSE compiler infrastructure**
 - ✓ **Interconversion between ROSE ASTs and XML ASTs.**
- **XSLT** is employed to describe transformation rules
 - ✓ XSLT rules can be written in a text format.
 - ✓ In the framework, other XML-related technologies are also available for transformation, analysis, and visualization of ASTs.
- Xerces and Xalan libraries are used for XML data representation and transformation.

CUSTOM CODE TRANSFORMATION

Application code is just annotated with a user-defined mark (directive/comment).

```
!$xev loop_tag
do k=1,n-1
  do j=1,n-1
    do i=1,n-1
      B(i,j,k)=A(i,j,k)
    end do
  end do
end do
```

Application code

```
<xsl:template match="SgFortranDo">
  <xsl:choose>
    <xsl:when test="preceding-sibling::*[1]/SgPragma/@pragma = 'xev loop_tag'">
      <xsl:comment>
        test-3.xsl xev loop_tag
      </xsl:comment>
      <xsl:variable name="step1">
        <xsl:apply-templates select="." mode="chill_unroll_jam">
          <xsl:with-param name="max" select="4" />
          <xsl:with-param name="var" select="'k'" />
        </xsl:apply-templates>
      </xsl:variable>
      <xsl:apply-templates select="exslt:node-set($step1)"
        mode="find_loop_and_unroll" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy>
        <xsl:copy-of select="@*" />
      </xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Unroll and jam

Loop unrolling

Every translation rule is written declaratively in XML (XSLT).
Users can customize it without developing their own code translators.

The translation rule is defined in an **external** file

AUTOMATIC GENERATION OF TRANSLATION RULES *1

```
program loop_inv0
```

```
!$xev tgen variable(i_, i0_, i1_)
!$xev tgen list(stmt_)
```

```
!$xev tgen src begin
```

```
!$xev(.) loop inv
```

```
do i_ = i0_, i1_
```

```
  call xev_exec(stmt_)
```

```
end do
```

```
!$xev tgen src end
```

```
!$xev tgen dst begin
```

```
do i_ = i1_, i0_, -1
```

```
  call xev_exec(stmt_)
```

```
end do
```

```
!$xev tgen dst end
```

```
end program loop_inv0
```

Two code versions : original and translated codes

A list variable catches multiple things

Directive that drives transform

The code pattern **before** transformation

Special form to catch arbitrary statement

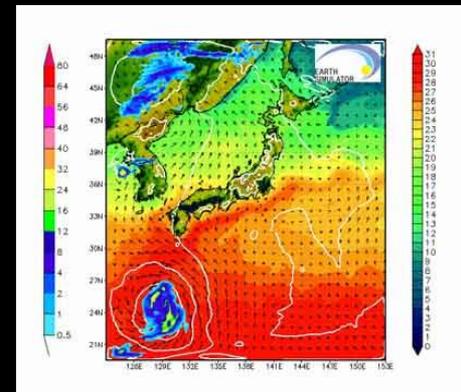
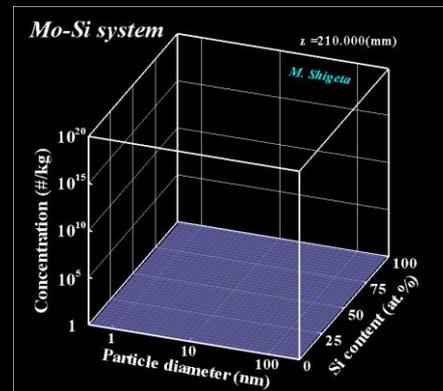
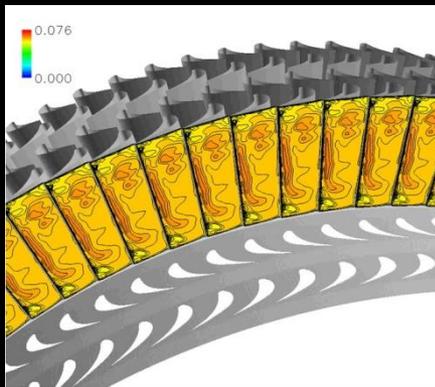
Loop is reversed

The code pattern **after** transformation

Reproduces the caught statement

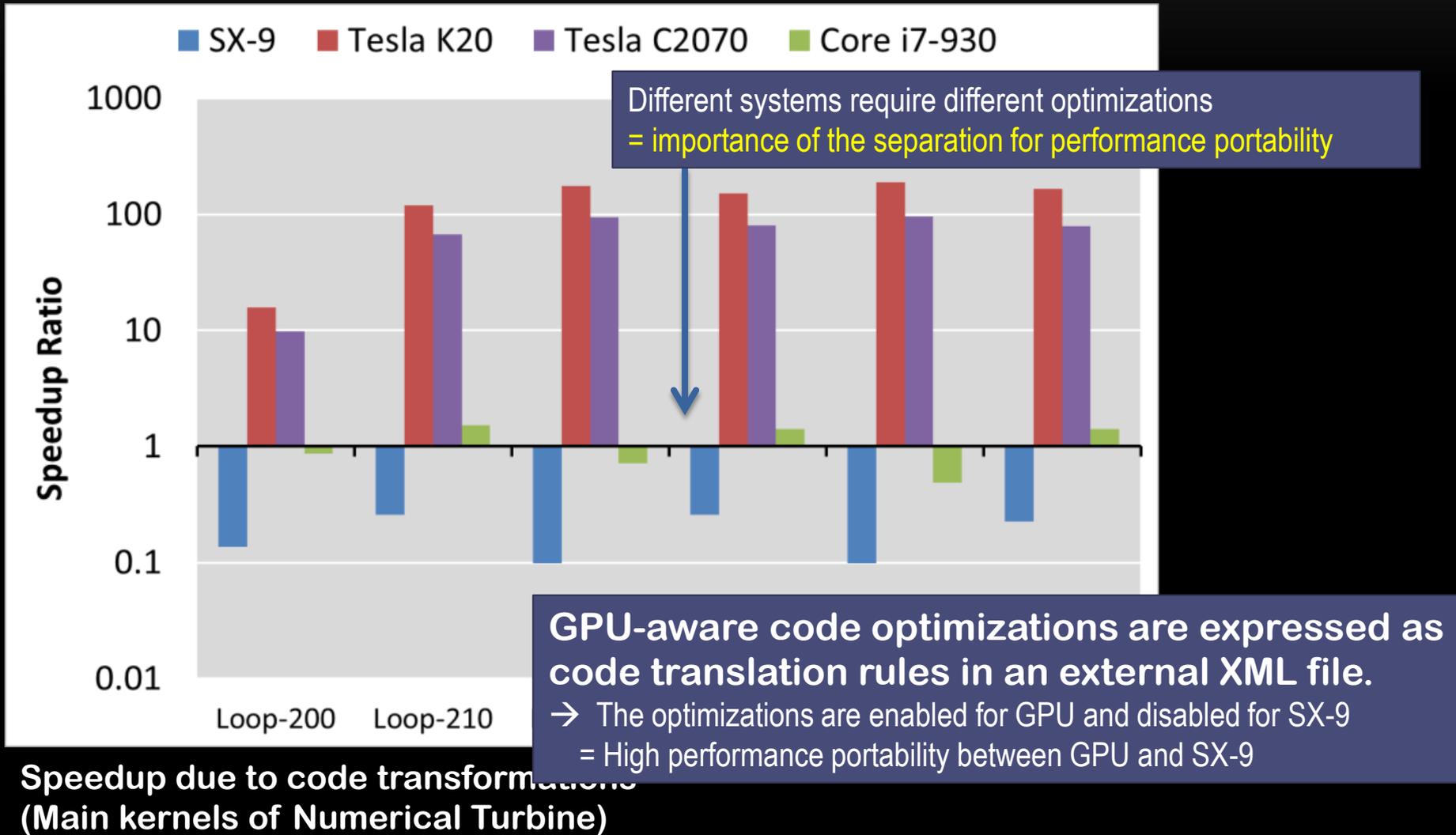
CASE STUDIES WITH REAL APPLICATIONS *2

- **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)



Xevolver can express system-awareness in an XML data format for migrating all the applications to OpenACC platform without major modifications.

PERFORMANCE EVALUATION RESULTS (NT)



AOS-TO-SOA CONVERSION

```
struct aos {
    double x;
    double y;
    double z;
} point[N];

void init(){
    for(i=0;i<N;i++)
        point[i].x = point[i].y = point[i].z = 0;
}
```

```
struct soa {
    double x[N];
    double y[N];
    double z[N];
} point;

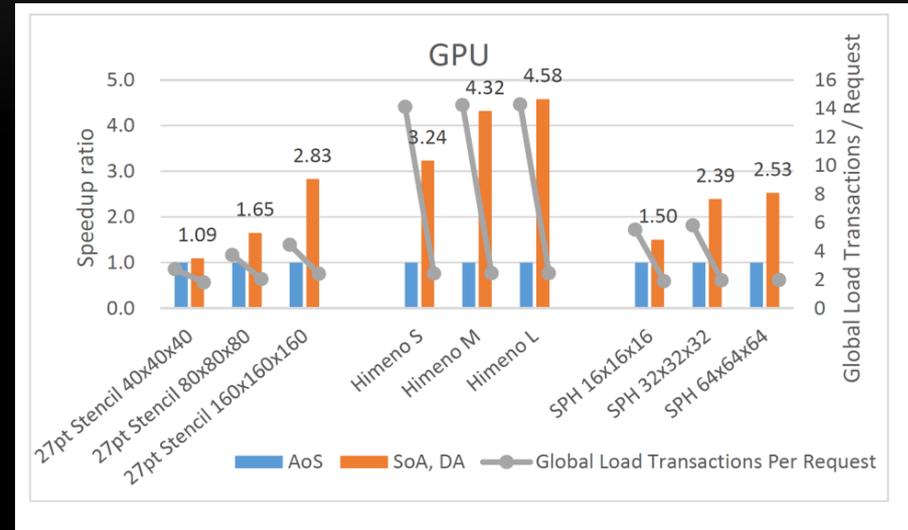
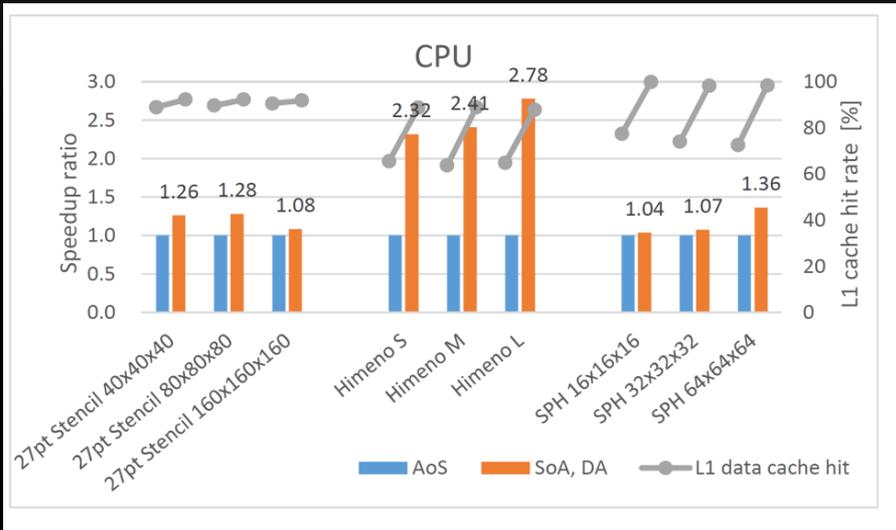
void init(){
    for(i=0;i<N;i++)
        point.x[i] = point.y[i] = point.z[i] = 0;
}
```

`point[A].B` to `point.B[A]`

A: any integer expression
B: any member of the struct

For AoS-to-SoA conversion
(1) Convert the declaration
(2) Convert every reference

IMPACT OF DATA LAYOUT OPTIMIZATION *3



- **Data layout optimizations can improve the performance of both CPU and GPU**
 - The GPU performance is more sensitive to the data layout.
 - The CPU performance also improves if the data size exceeds the cache capacity.
 - **The transformation rule is reusable** if customized for individual systems and applications

CONCLUSIONS

- **Xevolver Framework**

- AST is converted to a text format (XML) and exposed to programmers.
- System-specific optimizations are separated from application codes.
 - **Application developers** can maintain the original code
 - **Performance tuners** describe system-specific optimizations in an external file
- Helpful for an appropriate division of labor.

→ We need **a standard way to express system-awareness** to fight against system diversity in the future.

VISIT OUR BOOTH!

- Tohoku University (#2315)

SC15
November 16-19, 2015
 Austin Convention Center - Level 1
 Austin, Texas

