Framework for Separation of Concerns Between Application Requirements and System Requirements

HIROYUKI TAKIZAWA, SHOICHI HIRASAWA, AND HIROAKI KOBAYASHI
(Tohoku University/JST)
BACKGROUND

• HPC programming = team work of programmers with different concerns

  Application developers ( = computational scientists)

  • write a program so as to get correct results

  → Main concern: relationship between simulation models and programs.

  Performance tuners ( = computer scientists/engineers)

  • write a program so as to get high performance

  → Main concern: relationship between programs and computing systems.
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?
How can we help legacy application migration?
It is difficult because system-specific optimizations are tightly interwoven with application codes.

- **System-aware Code Optimizations in Existing Applications**
  - Egawa@Tohoku-U
    - The code patterns should be refactored because they potentially (likely) degrade the performance portability across different systems.

- **System-aware Code Optimizations for “PostPeta” Systems**
  - Suda@U-Tokyo and Takahashi@U-Tsukuba
    - New optimization techniques and algorithms for future systems
      - Communication-avoiding algorithms, etc (Suda)
      - Highly-optimized implementations for GPU clusters, etc (Takahashi).

- **Representation of System-awareness**
  - Takizawa@Tohoku-U (PI)
    - How to separate system-awareness from application codes
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 vectorize the loop nests for GPUs.
  - Because of the same loop structure, all the loop nests need to be modified *in the same way* for GPUs.
REPETITIVE PATTERNS IN CODE MODIFICATIONS

Manual code modifications can be replaced with a smaller number of 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
An AST is a data tree and naturally represented as an XML document.

```latex
\{ 
  for(i=0; i<N; i++){
    for(j=0; j<M; j++){
      ...
    }
  }
\}
```
On top of the ROSE compiler infrastructure

- **Interconversion between ROSE ASTs and XML ASTs.**
- **XSLT** is employed to describe translation rules
  - XSLT rules can be written in a text format.
  - In the framework, other XML-related technologies are also available for translation, analysis, and visualization of ASTs.
- Apache Xerces and Xalan libraries are used for XML data representation and translation.
CUSTOM CODE TRANSFORMATION

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

```fortran
!!!xev loop_tag
!$xev loop_tag
!$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
```

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

```json
{
  "xev loop_tag":{
    "target":"SgFortranDo",
    "rules":[
      {"chill_unroll_jam":{"var":"k","max":4}},
      {"chill_unroll":{"var":"i","max":2}}
    ]
  }
}
```

- **Unroll and jam**
- **Loop Unroll**
CUSTOM CODE TRANSFORMATION

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

```fortran
!$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
```

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.
**RECENT PUBLICATION** *¹*

- 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)

*¹ Takizawa et al@HiPC2014.

---

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)

Different systems require different optimizations = importance of the separation for performance portability.

GPU-aware code optimizations are expressed as code translation rules in an external XML file.

→ The optimizations are enabled for GPU and disabled for SX-9 = High performance portability between GPU and SX-9.

Speedup due to code transformations (Main kernels of Numerical Turbine)
DEMONCARTIZING CODE TRANSFORMATIONS!

Program `loop_inv0`

```fortran
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
```

Automatic generation of translation rules

- A list variable catches multiple things
- Directive that drives transform
- Special form to catch arbitrary statement
- Loop is reversed
- The code pattern before transformation
- The code pattern after transformation
- Reproduces the caught statement

SIAM Computational Science & Engineering 2015

Ongoing
CONCLUSIONS

• **Xevolver Framework**
  
  = No new languages and models = incremental migration

  • AST is converted to a text format (XML) and exposed to programmers.

  • System-specific optimizations are separated from app.
    • Computational scientists can maintain the original code
    • Performance tuners describe system-specific optimizations in an external file

  → Helpful for long-term software evolution.

We need a standard way to describe system-awareness to fight against never-ending system changes.

  → Standardized Optimization Programming Interface
ACKNOWLEDGEMENTS

• We would like to thank Profs. Satoru YAMAMOTO, Masaya SHIGETA, and Keiko TAKAHASHI for allowing us to use their simulation codes for the performance evaluation.

• This work was partially supported by JST Post-Peta CREST and Grant-in-Aid for Scientific Research(B) #25280041.

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

Your feedbacks (and bug reports) are welcome!