Program Transformation Techniques Applied to Languages Used in High Performance Computing

Songqing Yue (syue@cs.ua.edu)
Advisor: Dr. Jeff Gray

THE UNIVERSITY OF ALABAMA
COMPUTER SCIENCE

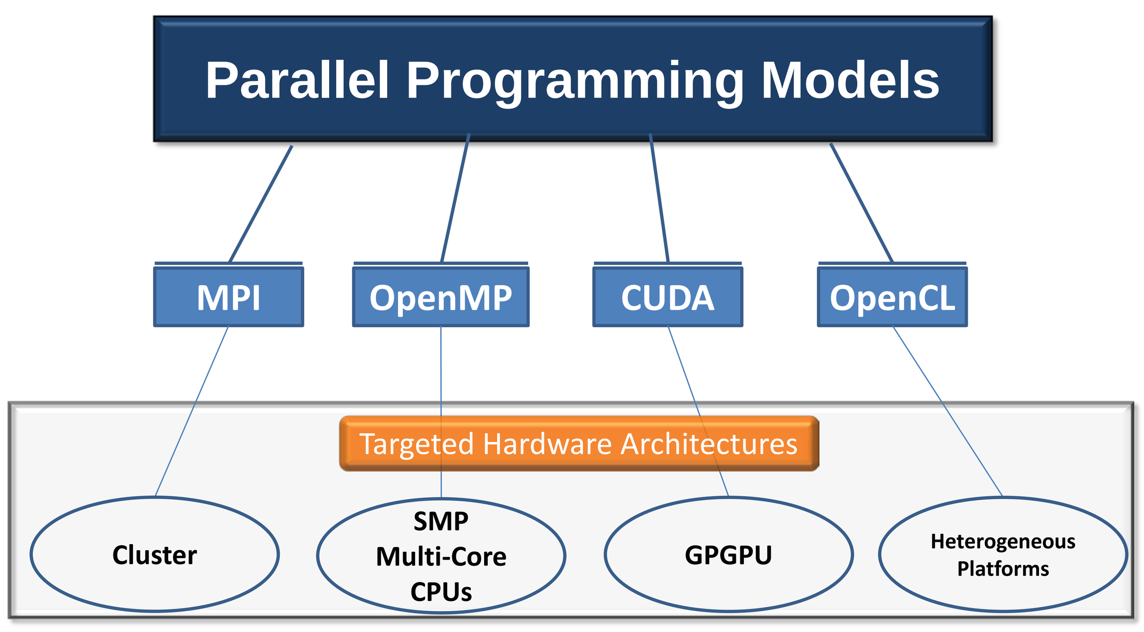
Background

Sequential Computing

Traditionally, software has been written for ***serial*** computation…

Parallel Computing

…the simultaneous use of multiple compute resources to solve a computational problem

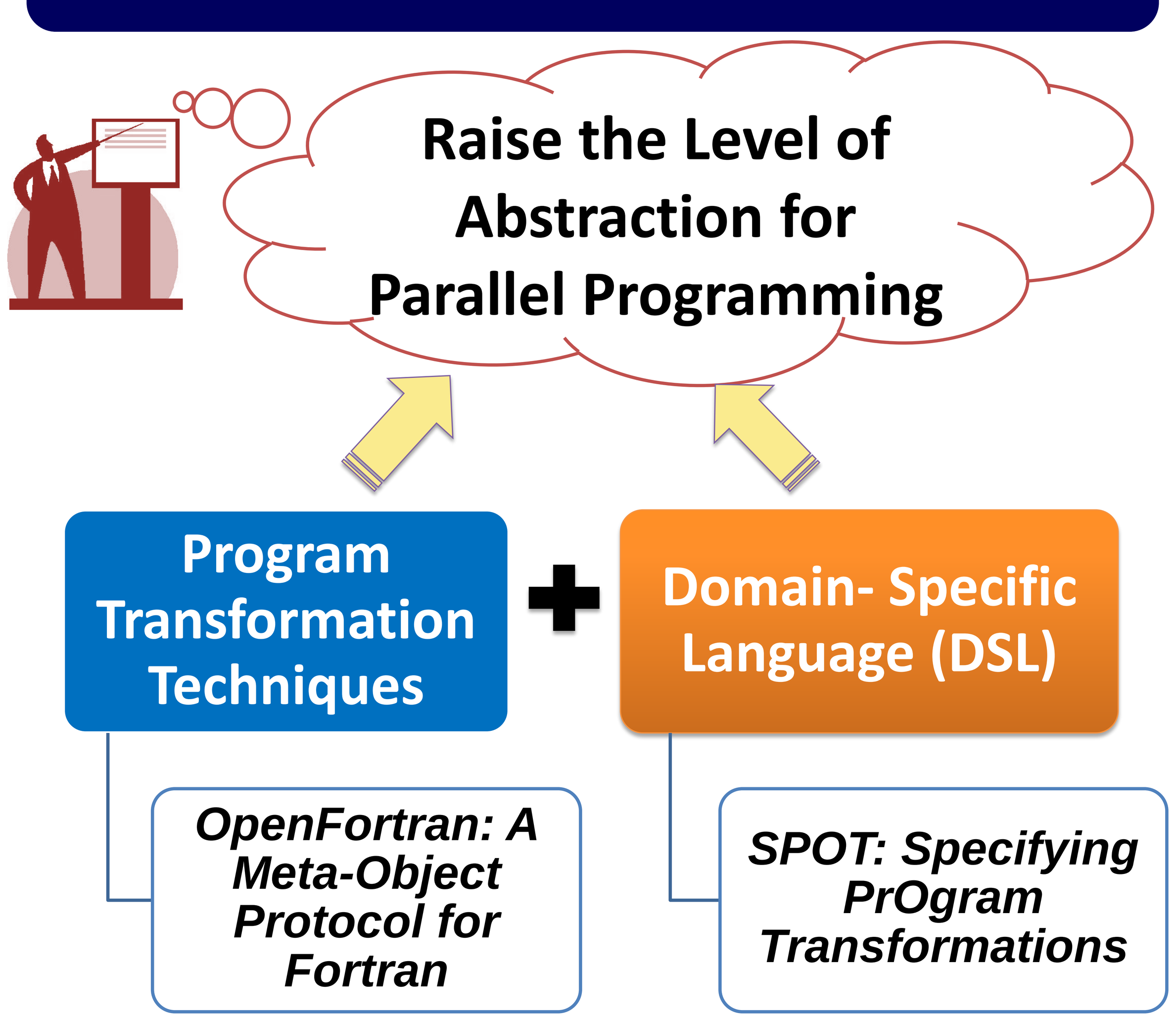


Problems

Using parallel programming models to convert serial programs into parallel programs presents several new challenges:

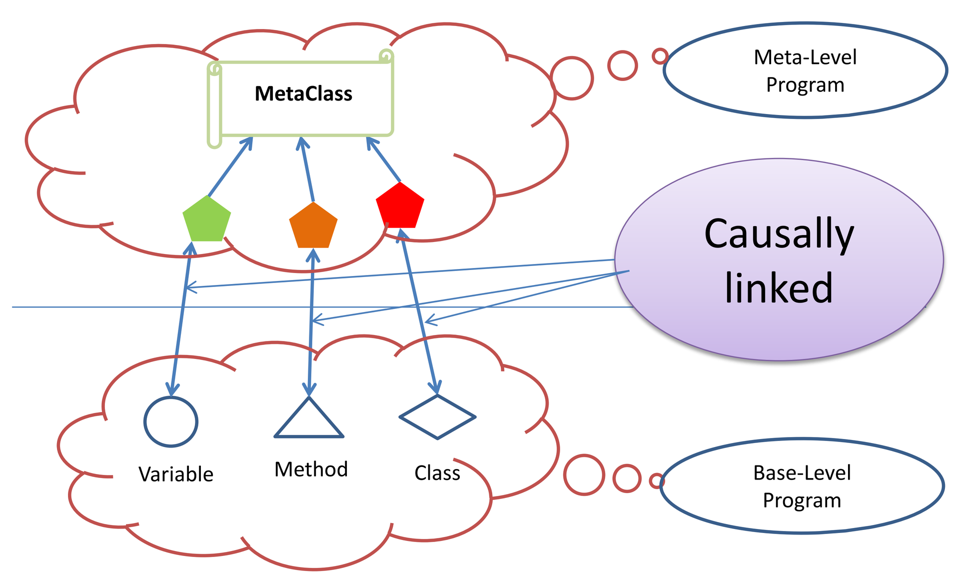
- It is a tedious and error-prone process to manually parallelize sequential code. The programmers are typically responsible for identifying parallelism.
- These models necessitate invasive reengineering of existing programs for inserting parallel code, typically with compiler directives and API invocations. The process can be intrusive and thus makes software maintenance extremely challenging.
- Various tools, such as parallelizing compilers or pre-processors are available to assist with the process of parallelization. However, most of these tools are either concentrated on a particular device or language, or limited to a subset of code (mostly loops). Furthermore, most of the available solutions are invasive and require source code changes.

Research Goal

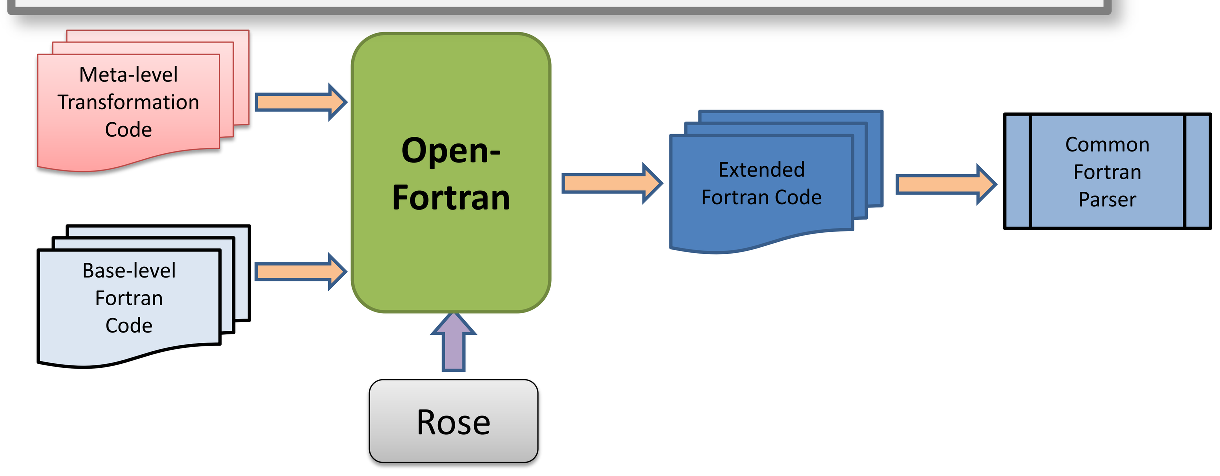


Our Approach

Meta-Object Protocol (MOP)



Transformation Process with OpenFortran



OpenFortran

+	−
A powerful tool to provide computational reflection • It can be used to programmatically affect basic language mechanisms • It offers control over compilation to avoid performance downgrade	Present learning and usage challenges to user • There is a steep learning curve • It is difficult for average programmers to understand the complex details of meta-programming and program transformation

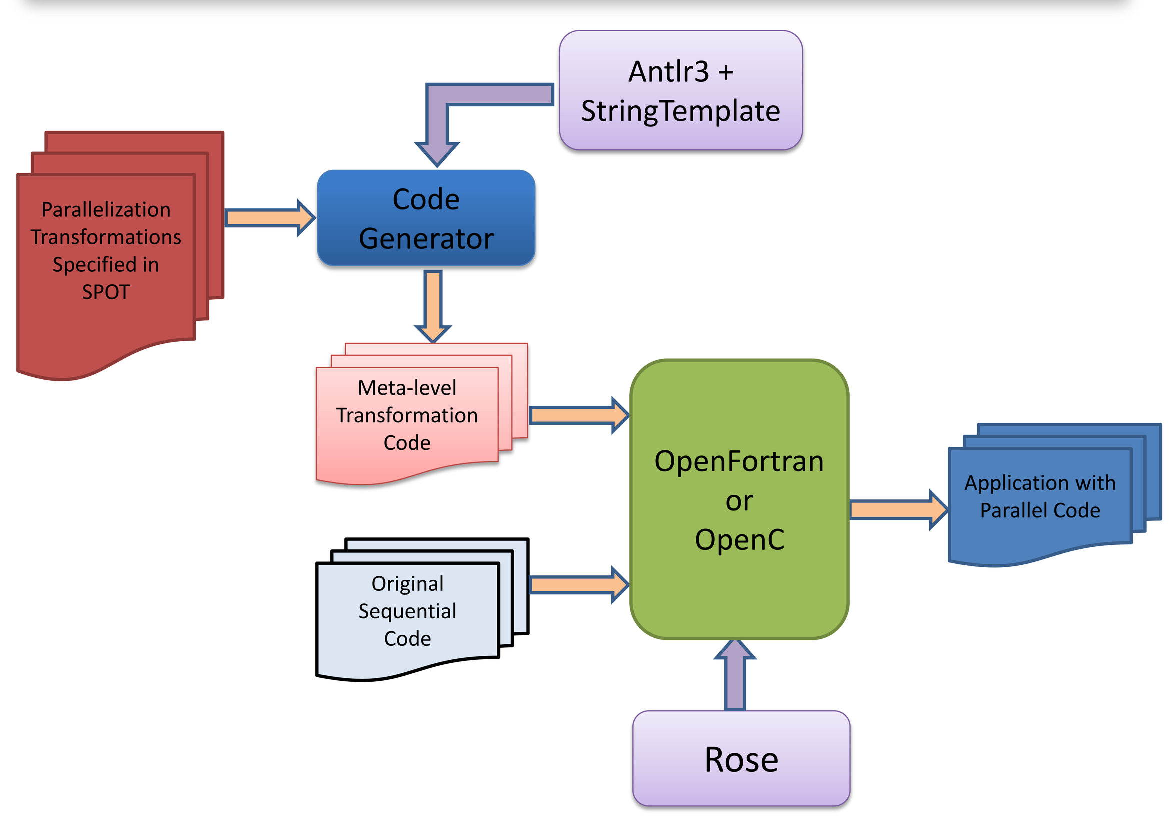
SPOT: A Domain-Specific Language

Design Goal: To provide language constructs that allow developers to perform direct manipulation on programs and hide the accidental complexities

Design Decisions:

- High-level programming concepts, e.g., functions, variables, statements and classes as the building constructs of SPOT
- Rule-based patterns allow systematic transformations, such as add, delete, and modify a programming concept
- AspectJ-style to express locations and scope of transformation

Overview of the parallelization process



Case Study

```
PROGRAM TEST
…
DO I=1,1000
A(I) = A(I) + B
ENDDO
END
```

Parallelize Fortran Code with OpenMP

```
PROGRAM TEST
…
!$OMP PARALLEL DO
DO I=1,1000
A(I) = A(I) + B
ENDDO
!$OMP END PARALLEL DO
END
```

```
Transformer paraDoExe
{
  Program pgm = FindProgram(fileName.TEST)
  LoopStatement firstLoop = FindLoop(pgm, lname)
  Around(LoopStatement firstLoop)
  {
    AddCommentBefore($OMP PARALLEL DO)
    AddCommentAfter($OMP END PARALLEL DO)
  }
}
```

Contributions

- Our approach allows expression of program transformations in terms of design intent rather than the underlying implementation
- The transformation process is at compile-time, so there is reduced run-time penalty
- Higher-level abstraction enables generating to different languages of implementation from the transformation library written in SPOT
- Our approach is non-invasive by generating a new copy of code for transformation and keeping the original code intact

Future Work

- Extend and implement new constructs for SPOT in an incremental manner to support more parallel programming models
- Empirical Evaluation
  - ✓ Productivity
  - ✓ Accuracy
  - ✓ Adaptability
- Implement a generalized framework named OpenFoo, suitable for extending an arbitrary programming language by creating a MOP for the language
- Possible Application Areas

Profiling
- collect and analyze performance information
- help developers obtain an overview of system performance
- Provide run-time feedback to help with parameters of parallelization

Checkpointing
- add fault tolerance into computing systems
- stores a snapshot of the current application state, and later on, use it for restarting the execution in case of failure

Code Coverage Tool
- determine the degree to which the source code of a program has been tested