ICON DSL: A Domain-Specific Language for climate modeling

Raul Torres, Leonidas Linardakis, Julian Kunkel, Thomas Ludwig

WOLFHPC 2013

18-11-2013

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

Introduction

Climate simulation models

- Global climate simulations are one of the “Grand Challenges” of computing
- Composed by several hundreds of thousands of code lines in a general-purpose language
- Code complexity increases to simulate additional physical processes
- Modelers have to equilibrate efficiency and portability (not an easy task)
- Debugging and maintenance are difficult
- Common high level approaches are the usage of backend libraries or template-based operators, both being awkward expressions of mathematical operators

ICON Climate Model

- An initiative from Max Planck Institute for Meteorology and The German Weather Service (http://www.mpimet.mpg.de/en/science/models/icon.html)
- Its goal is to integrate circulation models for the atmosphere and the ocean in a unified framework
- It is being written in Fortran for several years
- It exhibits several explicit machine-dependent optimizations, i.e:
  - Nested Do loops were written originally to exploit vectorization on a vector machine
  - But for cache-based architectures, the order of the loops should be changed
  - The change was achieved by using preprocessing directives
  - What about the index order? and the memory layout?

Our proposal

We aim to provide an abstraction framework for the ICON model in the form of a Domain-Specific Language (DSL)

- It is an extension of Fortran
- New keywords hide memory dimension and layout of variables with specific model semantics
- A source-to-Source translator converts DSL code into fully compatible Fortran code, where the computation details are expressed
- It uses an Intermediate Representation (IR) suitable for simplification and high level optimizations
- It has the ability to express climate mathematical operators in an easy and natural way.
- And the capability to adapt the implementation of these operators to different architectures and parallel levels

The current implementation is preliminary, but demonstrates a great potential for adaptivity and user-friendliness.

ICON Domain-Specific Language

Keyword specification

Keywords of the DSL and their corresponding behavior are defined in a separated platform-specific file. Each new keyword is defined as 3-field tuple separated by spaces, as follows:

<keyword_name> <platform_specific_settings> <keyword_type>

- *keyword_name* : new keyword
- *platform_specific_settings* : keyword feature
- *keyword_type* : where in Fortran

Example:

Platform A: BASIC_ARRAY 1,0 declare

Platform B: BASIC_ARRAY 0,1 declare

Array declarations

Configuration:

```
ON_CELLS {1,2,0,3} declare
```

Usage of the keyword:

```
REAL, ON_CELLS, POINTER ::  my_variable
my_variable( i , j , k, l) = 2
```

Generated Fortran code:

```
REAL, DIMENSION(:,:,:,:), POINTER ::  my_variable
my_variable( j , k , i, l) = 2
```

Array initialization

Configuration:

```
SHAPE_4D {1,2,0,3} initialize
```

Usage of the keyword:

```
my_variable = SHAPE_4D( a, b, c, d )
```

Generated Fortran code:

```
my_variable = (/ b , c , a, d /)
```

Optimizers

Configuration:

```
INLINE inline optimize
```

Usage of the keyword:

```
INLINE SUBROUTINE example_subroutine(...)

INLINE CALL example_subroutine(...)
```

Design of the translation infrastructure

First approach: ANTLR Parser Generator

ANTLR has capabilities for designing of parsers for grammars, specially for DSLs. However, we encountered several burden that made development harder.

- The symbol table must be built and managed by the programmer itself
- AST usage is cumbersome
- Recovery of ignored tokens might be difficult
- The implementation of the inlining mechanism required the support of an external text replacement tool

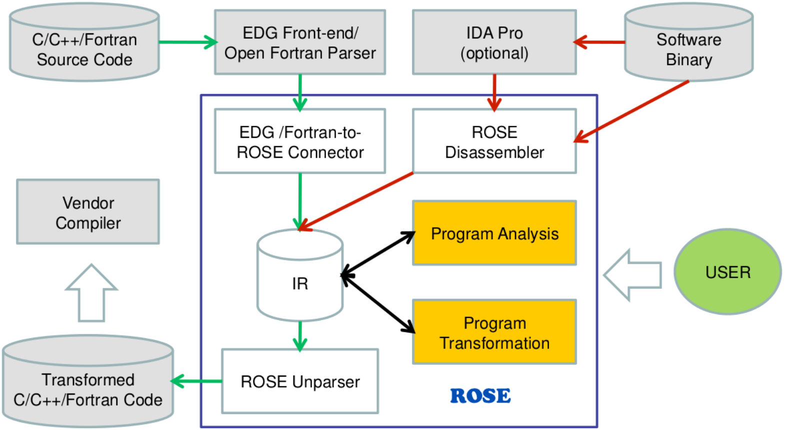
We recommend ANTLR for:

- Design of simple grammars and translators
- Implementation of parsers
- Construction of translators between different languages

Rose Compiler (http://rosecompiler.org/)

- Source-to-source translation infrastructure developed at Lawrence Livermore National Laboratory
- Open source project
- Targets expert and non-expert audience
- Works as a library and is written in C++ mostly
- Supports C, C++, Fortran and UPC
  - Front-end that converts a given language to an AST
  - Back-end that generates Fortran code
- The AST preserves all the information of the code
- Comes with some generic analyses, transformations and optimizations at the AST level
  - Loop optimization
  - Inlining
  - Outlining
  - Auto-parallelization

Rose overview



Taken from: Semantic-Aware Automatic Parallelization of Modern Applications Using High-Level Abstractions. Liao et al.

Issues with Rose

- Rose Compiler provides no interface to design a language extension
- A few correctly parsed Fortran statements have no corresponding action to build nodes on the AST
- Pragma annotations of the kind of Open MP are given nodes in C or C++ codes, but not in Fortran codes
- Same for Inlining mechanism
- Rose creates a sort of header files for Fortran modules, but they do not store the semantics of the our extension

Translation infrastructure

The translation of extended Fortran code into native Fortran works as follows:

1. A machine-dependent configuration file is parsed, where the particular details of the platform are specified.
2. The DSL enriched Fortran code is parsed, the symbol table and the intermediate representation, called Abstract Syntax Tree (AST), are constructed, without losing any information about the source code.
3. Before unparsing, the tree is modified to transform the provided abstractions according to those particularities of the platform.
4. As a final step, native Fortran code is generated by traversing the modified tree.

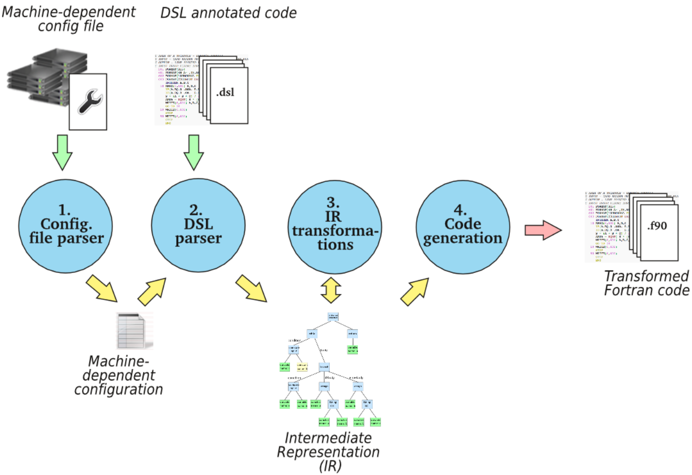


Figure: Translation infrastructure

Evaluation

Considerations

- Original code was optimized initially for a vector machine (NEC)
- A memory bandwidth bottleneck on current cache-based machines was detected
- We utilized an optimized memory layout for IBM Power6 and Intel Westmere architectures
- It was determined manually to make a better use of the available cache levels
- The DSL abstractions were applied on the ICON testbed code
- A synthetic test data was used with a configuration of 20480 cells x 78 levels
- The DSL keyword for inlining was not used
- Generated Fortran code was compiled and executed on the mentioned architectures

IBM Power6

With the appropriate machine-specific configuration the efficiency of central data structures of ICON could be improved, obtaining up to 17% of speedup

Cores	32	64	128	192
NO_DSL iterations/sec	635479	1426037	2798150	3601217
DSL iterations/sec	719527	1664402	3096318	3993947
Speedup	13%	17%	11%	11%

Table: Achieved iterations per cells per sec for different number of cores on an IBM Power6 architecture

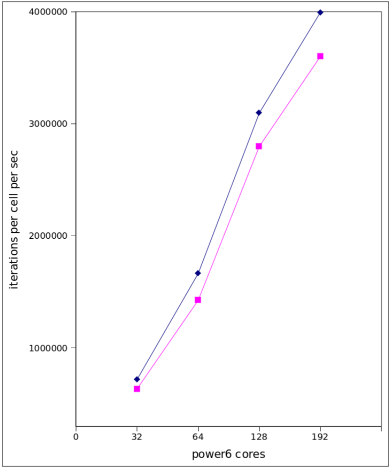


Figure: Performance comparison between code with and without DSL keywords for IBM Power 6 architecture

Intel Westmere

For the case of Westmere, up to 16% of speedup was obtained

Cores	2	4	8	12
NO_DSL iterations/sec	41914	65937	61292	55209
DSL iterations/sec	48574	75521	68908	60927
Speedup	16%	14%	12%	10%

Table: Achieved iterations per cells per sec for different number of cores on an Intel Westmere architecture

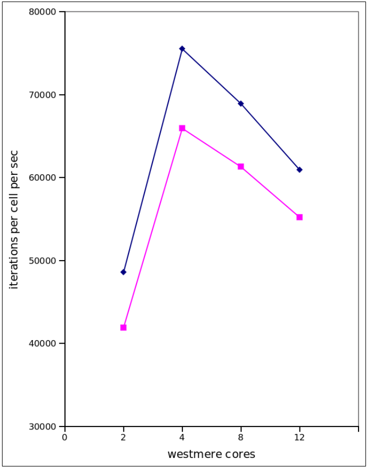


Figure: Performance comparison between code with and without DSL keywords on a Intel Westmere architecture

Performance Counter	NO DSL	DSL	Improvement
Retired instructions	1.68322e+12	1.5579e+12	7% reduction
Cycles per instruction	0.546809	0.514415	6% reduction
L1 cache misses rate	0.0170913	0.00532005	68% reduction
L2 cache misses rate	0.00518718	0.00410406	20% reduction
Memory bandwidth (MB/sec)	1221.44	1422.61	14% increase

Table: Performance counters on a Intel Westmere architecture

On going and Future work

On going work: Loop abstraction

```
type(t_int_state), intent(in)            :: ptr_int
real(wp), EDGES_3D, intent(in)       :: vec_e
intent(wp), CELLS_3D, intent(inout) :: div_vec_c
SUBSET, CELLS_3D, intent(in)     :: cells_subset
ELEMENT, CELLS_3D                :: cell
ELEMENT, EDGES_OF_CELL           :: edge

FOR cell in cells_subset DO
  div_vec_c(cell) = 0.0_wp
  FOR edge in cell%edges DO
    div_vec_c(cell) =  div_vec_c(cell) + &
    & vec_e(edge) * ptr_int%geofac_div(edge)
  END FOR
END FOR
```

```
type(t_int_state), type(in) :: ptr_int
real(wp), intent(in)        :: vec_e(:,:,:)
real(wp), intent(inout)     :: div_vec_c(:,:,:)
type(t_subset_range_3D)     :: cells_subset
type(t_grid_cells), pointer :: cell_cells
integer                     :: cell_idx_start, cell_idx_end, ...
integer                     :: edge_cell_idx, edge_idx, ...

cell_cells =>  cells_subset%cells

DO cell_block = cells_subset%start_block, &
& cells_subset%end_block
  ...
  DO cell_idx = cell_idx_start, cell_idx_end
    ...
    DO cell_level = cells_subset%start_level, &
    & cells_subset%end_level
      ...
      div_vec_c(cell_level, cell_idx, cell_block) = 0.0_wp
      ...
      DO edge_cell_idx = 1, cell_cells%num_edges(cell_idx, &
      & cell_block)
        ...
        div_vec_c(cell_level, cell_idx, cell_block) = ...
        ...
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

Future work

- Opportunities for automatic parallelization
- Emerging architectures based on accelerators or heterogeneous hardware can be targeted
- Different levels of parallelism (blocks, thread groups, threads, vectors, etc.) can be exploited
- Usage of different memory layouts on a single architecture
- Outlining can be used to build kernels

Conclusion

ICON DSL as a Fortran extension:

- It eases the modeling process for the climate expert
- It allows code portability and facilitates performance improvement
- There is no need to learn a new language
- Array declarations and initializers can take advantage of memory layout abstractions
- Subroutine calls can be easily optimized by being inlined

Automatically generated code exhibited a significant improvement on IBM Power6 and Intel Westmere architectures when the appropriate set of index interchanges were expressed in the configuration file of the DSL

Thanks!
Danke!
Gracias!