Status Report: University of Hamburg

Nabeeh Jum'ah, Anastasiia Novikova, Julian Kunkel

Scientific Computing
Department of Informatics
University of Hamburg

2017-03-20

AIMES

Outline

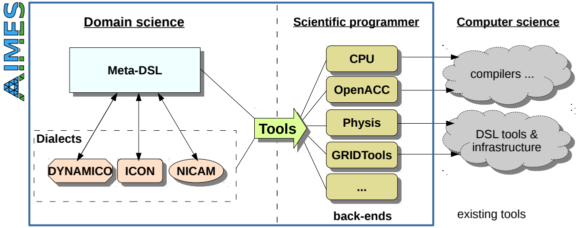
WP1: Towards Higher-Level Code Design

Recap: Goals of the WP

- Bypass shortcomings of general-purpose languages
  - Offer performance-portability
  - Enhance source repositories maintainability
  - Get rid of complexity in optimized-code development
  - Enhance code readability and scientists productivity
- Extend modelling programming language
  - Based on domain science concepts
  - Free of any lower level (e.g., architecture) details

Approach

- Re-arrange model development workload
  - Domain scientists develop domain logic in source code
  - Scientific programmers write hardware configurations
- Source code written with extended language
  - Closer to domain scientists logic
  - Scientists do not need to learn optimization
  - Write code once, get performance for various configurations
- Hardware configurations define software performance
  - Written by programmers with more experience in platform
  - Comprise information on target run environment



Pursued Approach

DSL

- Co-Design approach
- Scientists agreement on a unified set of constructs
- Model-specific dialect support

Technical ...

- Prototyped the lightweight source-to-source translation tool
- Developed a strategy how to embed it into build systems (to-discuss in the round!)
- Investigated build optimization(LLVM conference poster)
- Investigated various approaches for iterating HEVI ICO data...

Pursued Approach

Artefacts

- Repository of code kernels
- Code for prototype
- D1.1 started (but ahead of time)
- Posters

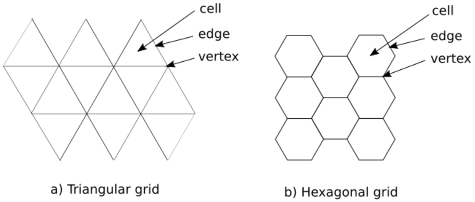
DSL development

Co-Design of DSL

- Extracted relevant codes from the models
- Reformulated them as DSL variants
- Identified commonalities and abstracted them
- Discussed with scientists the formulation
- Specified the DSL and converted code into DSLized version

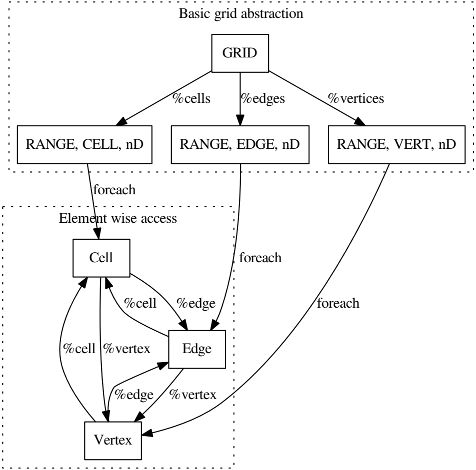
DSL development

- Developed DSL (**GGDML**) constructs
  - GGDML: *General grid definition and manipulation language*
  - Grid definition
  - Grid-bound variable declaration
  - Grid-bound variable access/update
  - Stencil operations
- Hide memory access details
- Abstract higher concepts of grids, hiding connectivity details



DSL development

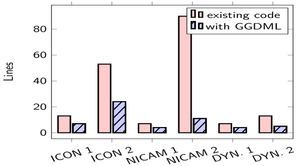
- GGDML basic concepts



GGDML impact

- LOC statistics

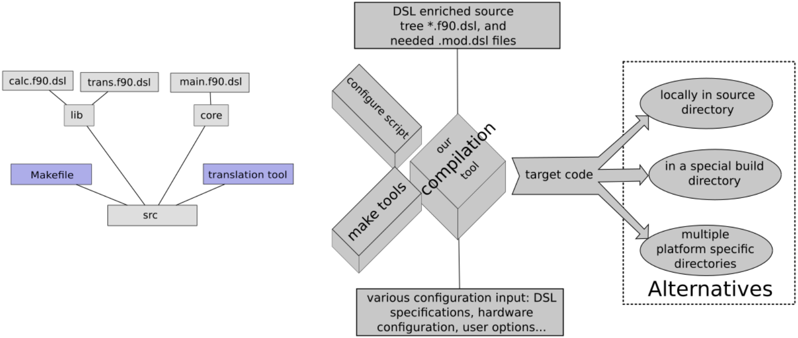
Model, kernel	lines (LOC) before DSL	lines (LOC) with DSL	words before DSL	words with DSL	characters before DSL	characters with DSL
ICON 1	13	7	238	174	317	258
ICON 2	53	24	163	83	2002	916
NICAM 1	7	4	40	27	76	86
NICAM 2	90	11	344	53	1487	363
DYNAMICO 1	7	4	96	73	137	150
DYNAMICO 2	13	5	30	20	402	218
total	183	55	911	430	4421	1991
percentage	30.05%		47.20%		45.04%	



- COCOMO estimations

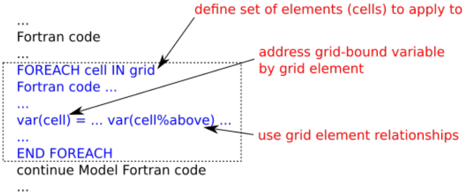
Software project	DSL?	Effort Applied	Dev. Time (months)	People require	dev. costs (M€)
Semi-detached	without	2462	38.5	64	12.3
	with	1133	29.3	39	5.7
Organic	without	1295	38.1	34	6.5
	with	625	28.9	22	3.1

Tool structure and embedding into code



Perspective: Scientist

- Use Fortran to write model
- Write kernels with GGDML iterator
- Within iterator: write Fortran code body
- Access an element with respect to grid (not memory)

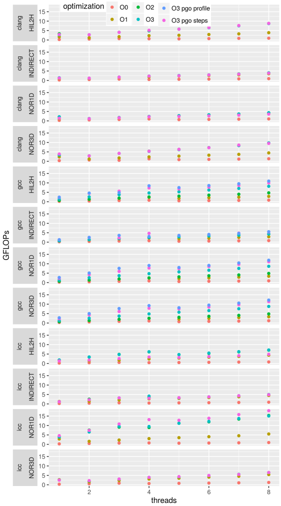


Perspective: Scientific Programmer

- Handle code translation workload
- Find best options for run environment to optimize software
    - Parallelization details
    - Caching options
    - Vectorization considerations
    - General optimizations options
    - ...
- Prepare configuration files necessary to generate code

Results of Performance Analysis

- Tools developed for
  - Source-to-source translation
  - Performance analysis
  - Model build optimization
- Explored Optimization options
- Explored memory-layouts
  - 3D and 1D transformation
  - Hilbert filling curves & HEVI
- With various compilers
  - Intel
  - GCC
  - CLang



WP1: Towards Higher-Level Code Design

- Higher-level code (i.e. using DSL extensions) is translated
  - within a configuration driven translation procedure
  - into machine-optimized general-purpose language code
- Soure-to-source translation tools are used
  - Lightweight tools for high maintainability and ease of use
  - Easily integrate into build systems (e.g. make)
  - High flexibility and extensibility
    - Different general-purpose languages can be used (pluged in)
    - Model-specific dialects can be handled
    - DSL extension/modification is possible with little effort

WP1: Towards Higher-Level Code Design

- Tools to improve compilation of optimized code
  - To harness power of general purpose programming language compilers
  - Improve usage of compiler options to build repositories
- Learn optimal compilation procedure at a repository build
- Use learned information for next builds
- Less compile time
- Optimized compilation to get performance of optimized code

WP2: Massive I/O

Recap: Goals of the WP2

- Optimization of I/O middleware for icosahedral data
  - Throughput, metadata handling
- Design of domain-specific compression (c. ratio $> 10:1$)
  - Investigate metrics allowing to define accuracy per variable
  - User-interfaces for specifying accuracy
  - Methodology for identifying the required accuracy
  - Compression schemes exploiting this knowledge

WP2: Supported quantities

Quantities defining the residual (error):

absolute tolerance: compressed can become true value $\pm$ absolute tolerance

relative tolerance: percentage the compressed value can deviate from true value

relative error finest tolerance: value definining the absolute tolerable error for relative compression for values around 0

significant digits: number of significant decimal digits

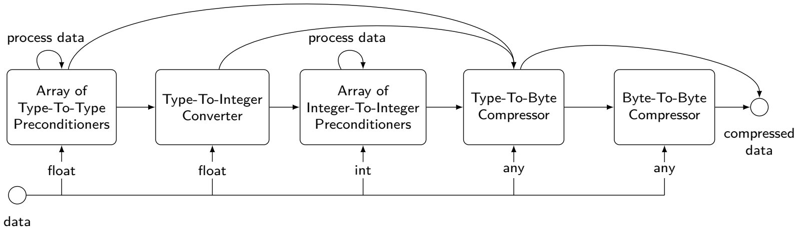
significant bits: number of significant decimals in bits

Quantities defining the performance behavior:

- absolute throughput in MiB or GiB
- relative to network or storage speed

The system's performance must be trained initially.

WP2: Architecture of SCIL



Compression chain

WP2: Tools

- Creating several relevant multi-dimensional data patterns of any size
- Adding random noise based on the hint set to existing data
- To evaluate compression on existing CSV and NetCDF data files

WP2: Implemented compression methods

- GZIP (GNU ZIP)
- ZFP
- SZ
- FPZIP
- LZ4fast
- WAVELET
- Wavetrisk
- Abstol
- Sigbits

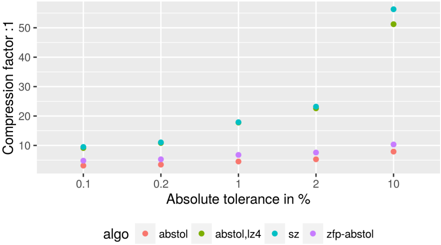
Abstol and Sigbits were developed alongside SCIL at the Universität Hamburg.

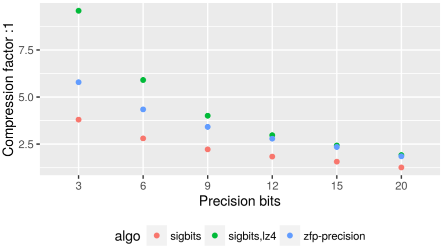
WP2: File Formats

- CSV
- HDF5
- NetCDF4

WP2: Tolerance-Based Results

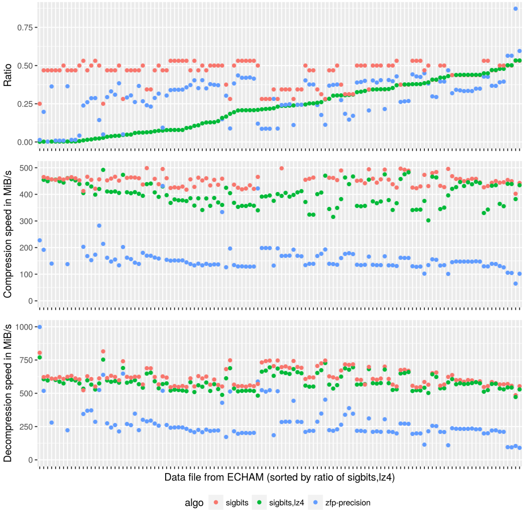
The mean compression factor is computed based on the sum of the data size: factor of 50:1 means the space is reduced to 2% of the original size.



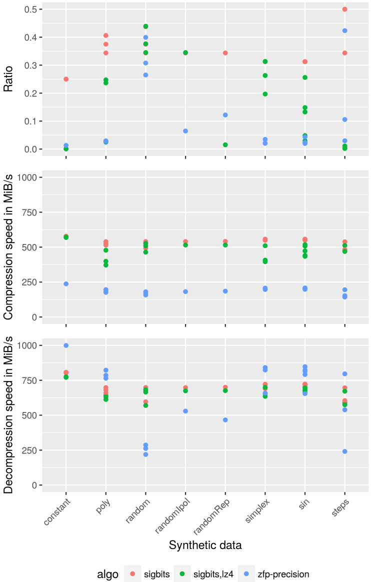


WP2: Results for Precision Bits

Comparing algorithms using 9 precision bits for the mantissa.



WP2: Results for Precision Bits



WP2: Results for Precision Bits

		Throughput [MiB/s]	
Algorithm	Ratio	Compr.	Decomp.
sigbits	0.45	464.4	620.7
sigbits,lz4	0.23	401.6	585.2
zfp-precision	0.3	194.8	418.3

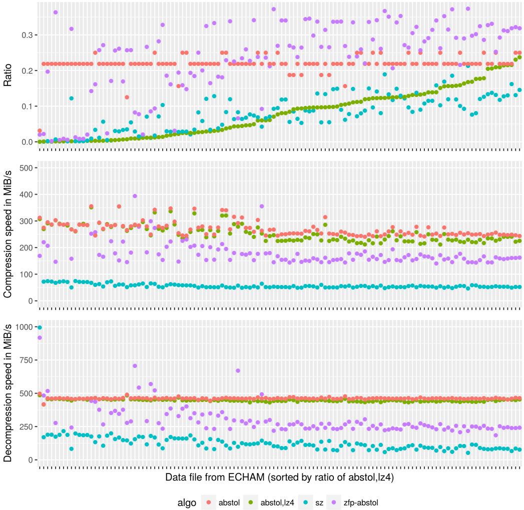
Table: For ECHAM data files

		Throughput [MiB/s]	
Algorithm	Ratio	Compr.	Decomp.
sigbits	0.39	520.9	658.0
sigbits,lz4	0.39	499.8	639.3
zfp-precision	0.32	170.1	255.8

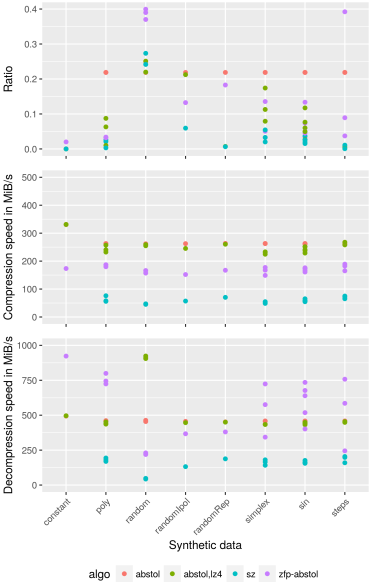
Table: For 5 different random patterns

WP2: Results for Absolute Tolerance

Comparing algorithms using an absolute tolerance of 1% of the maximum value



WP2: Results for Absolute Tolerance



WP2: Results for Absolute Tolerance

		Throughput [MiB/s]	
Algorithm	Ratio	Compr.	Decomp.
abstol	0.22	269.3	461.7
abstol,lz4	0.08	253.4	446.8
sz	0.08	66.0	127.4
zfp-abstol	0.24	263.9	492.9

Table: For ECHAM data files

		Throughput [MiB/s]	
Algorithm	Ratio	Compr.	Decomp.
abstol	0.229	260.1	457.8
abstol,lz4	0.230	257.9	913.2
sz	0.252	46.4	46.0
zfp-abstol	0.387	163.1	223.7

Table: For 5 different random patterns

WP2: Experiments

For each test data (CSV or NetCDF format), the following setups are run:

- Lossless compression
  - Algorithms: memcopy and lz4
- Lossy compression with significant bits
  - Tolerance: 3, 6, 9, 15, 20 bits
  - Algorithms: zfp, sigbits, sigbits+lz4
- Lossy compression with absolute tolerance
  - Tolerance: 10%, 2%, 1%, 0.2%, 0.1% of the data maximum value
  - Algorithms: zfp, sz, abstol, abstol+lz4

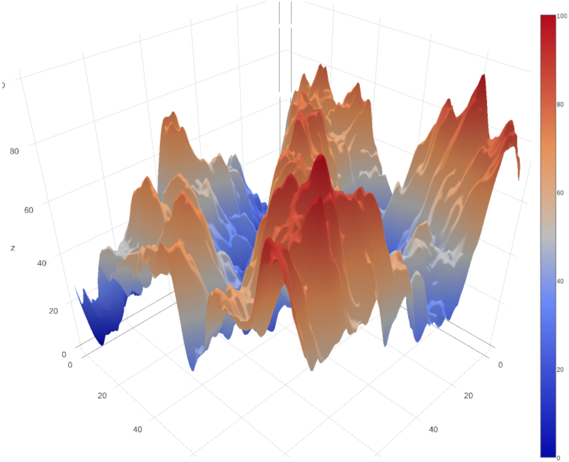
In the test, one thread of the system is used for the (de-)compression. A configuration is run 10x measuring (de-)compression time and compression ratio.

WP2: Experiments

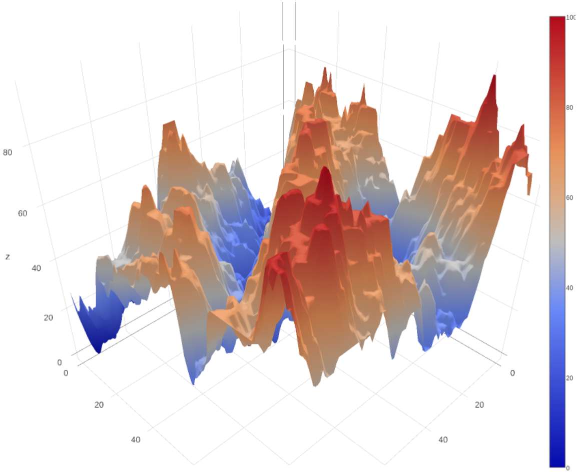
Single precision floating point test data is build upon:

- Synthetic, generated by SCIL's pattern lib.
    - e.g., Random, Steps, Sinus, Simplex
- Data of the variables created by ECHAM
    - The climate model creates 123 vars

WP2: Example synthetic data: simplex 206 in 2D



WP2: Compressed with Sigbits 3bits (ratio 11.3:1)



WP2: Summary

Results:

- novel algorithms can compete with ZFP/SZ when setting the absolute tolerance or precision bits
- SZ compresses better than abstol (in some cases)

Future work:

- A single algorithm honoring all quantities
- Automatic choose for the fitting algorithm

Goals in 2017

Deliverables

- M15 (May 2017) D1.1 Model-specific dialect formulations
- M18 (Aug 2017) D2.3 Report and code: compression API (+ test apps)
- M24 (Feb 2018) D1.2 Report and whitepaper: DSL concepts for ico models
- M24 (Feb 2018) D2.4 Report and code: best-practices to det. var accuracy
- M24 (Feb 2018) D2.5 Report: Optimization potential in ICO formats (?)

WP1

- Translation tool ready for Fortran ICON, NICAM and DYNAMICO kernels to:
  - OpenMP, GPU w. OpenACC, Xeon Phi
  - Target kernels as part of the testbed (WP3)
  - Some optimization (not all necessary optimizations)
- Evaluation of YASK

Goals in 2017

WP2

- Task 2.1: Investigate file formats for ICO data: 2 Month
  - Workshop: Exascale I/O for Unstructured Grids (EIUG)
  - September Monday 25th
- Establish collaboration with externals
  - Developers of: CubismZ, SZ, ZFP
- Stabilize APIs and memory handling of SCIL
- Decision algorithm for SCIL
- AllQuant compression: Honoring all quantities

Goals in 2017

WP3

- HDF5 / NetCDF benchmarking for compression
  - Synthetic using NetCDF bench (developed at DKRZ)
    - Extension of the benchmark for SCIL patterns
  - Problem: Parallel compression patch not yet included in HDF5
  - Embedded into a real application... But which?
  - Alternative: Application writes another format / we change it?
- Testbed with DSL-ized kernels and workflows to test

Roadmap for WP1

- May: Paper (DSL) to Journal: more on SWE / costs, writing D4.1 (scientific report)
- April: Writing D1.1, translation tool, (vacation)
- May: Deliver D1.1 (Model-specific dialects), trivial cases on Xeon Phi/GPGPUs, translation tool
- June: ISC-HPC attendance, convert ICON code to DSL, translation tool
- July: Support converting WP3 testbed to DSL, preliminary testing on Xeon Phi, GPGPUs
- Aug:
- Sept: Support converting WP3 testbed to DSL
- Oct: Support converting WP3 testbed to DSL, alternative memory layouts
- Nov: Performance testing of DSL
- Dec: Paper writing
- Jan: (vacation)
- Feb: Deliver 1.2 (DSL concepts)

Roadmap for WP2

- April: Investigate file formats + Start D2.3
- May: Write D2.3, structure for D2.4, D2.5, (*vacation*)
- June: SCIL extension: API + memory handling, ISC conference
- July: SCIL extension: AllQuant
  - Experiment with some file formats
  - Install them, observe access patterns on Mistral
- Aug: Deliver D2.3, prepare presentation for EIUG workshop
- Sept: EIUG workshop, Ideas for D2.4
- Oct: Write D2.5, Investigate file formats (*vacation*)
- Nov: Write D2.4, experiments for D2.4
- Dec: Experiments for D2.4
- Jan: Incorporate feedback into D2.4 / D2.5, decision algorithm
- Feb: Deliver D2.4, D2.5, paper for results of WP2

Collaboration

WP1 / Nabeeh

- March-May: Reviews/Contributions to D1.1 Model-specific dialect
  - Providing domain scientists perspective on formulation (one page per model / add your terminology)
  - Checking code snippets
- Sept-Feb: Reviews/Contributions to D1.2 DSL concepts
- July: Input needed: Formulation of WP3 testbed
  - Prototype kernels / applications
- Oct: Porting some kernels to the DSL (with our help)
  - We must do performance tests

Embedded documentation Deliverable: D1.1 that is Next

TODOs (as suggestions) are provided within the deliverables where actions are needed.
This way documents become self managed in terms of actions.

2.2.1 DYNAMICO

TODO: Thomas Dubos: Please add the typical "terms" used in Dynamico and give examples

2.2.2 ICON

TODO: Nabeeh: Check with Günther: Please add the typical "terms" used in Dynamico and give examples

2.2.3 NICAM

TODO: Hisashi Yashior: Please add the typical "terms" used in Dynamico and give examples

2.3 Functional Requirements

Collaboration

WP2 / Anastasiia

- June: Information about used file formats (D2.5)
  - What file formats are you using?
  - Describe issues and problems you have for these formats
  - Contribution to D2.5
- July: Review of D2.3
- Oct: Contribution to D2.4 (identify accuracy)
  - How can a scientist identify the accuracy?
  - We have to test it on real code?
  - Visualization / checkpoint restart requirements...
- Dec: Review of D2.4
- Dec: Review of D2.5

WP4

- April, Contributions to D4.1 (scientific report)