Status Report: University of Hamburg

Nabeeh Jum’ah, Anastasiia Novikova, Julian Kunkel

Scientific Computing
Department of Informatics
University of Hamburg

2017-03-20
Outline

1. Contributions to WP1
2. Contributions to WP2
3. Contributions to WP3
4. Goals in 2017
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

![Diagram of triangular and hexagonal grids](image.png)
DSL development

- GGDML basic concepts
GGDML impact

**LOC statistics**

<table>
<thead>
<tr>
<th>Model, kernel</th>
<th>lines (LOC)</th>
<th>words</th>
<th>characters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>before DSL</td>
<td>with DSL</td>
<td>before DSL</td>
</tr>
<tr>
<td>ICON 1</td>
<td>13</td>
<td>7</td>
<td>238</td>
</tr>
<tr>
<td>ICON 2</td>
<td>53</td>
<td>24</td>
<td>163</td>
</tr>
<tr>
<td>NICAM 1</td>
<td>7</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>NICAM 2</td>
<td>90</td>
<td>11</td>
<td>344</td>
</tr>
<tr>
<td>DYNAMICO 1</td>
<td>7</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>DYNAMICO 2</td>
<td>13</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>total</td>
<td>183</td>
<td>55</td>
<td>911</td>
</tr>
<tr>
<td>percentage</td>
<td>30.05%</td>
<td>47.20%</td>
<td>45.04%</td>
</tr>
</tbody>
</table>

**COCOMO estimations**

<table>
<thead>
<tr>
<th>Software project</th>
<th>DSL?</th>
<th>Effort Applied</th>
<th>Dev. Time (months)</th>
<th>People require</th>
<th>dev. costs (M€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-detached</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without</td>
<td>2462</td>
<td>38.5</td>
<td>64</td>
<td>12.3</td>
<td></td>
</tr>
<tr>
<td>with</td>
<td>1133</td>
<td>29.3</td>
<td>39</td>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>Organic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without</td>
<td>1295</td>
<td>38.1</td>
<td>34</td>
<td>6.5</td>
<td></td>
</tr>
<tr>
<td>with</td>
<td>625</td>
<td>28.9</td>
<td>22</td>
<td>3.1</td>
<td></td>
</tr>
</tbody>
</table>
Tool structure and embedding into code

Contributions to WP1

Contributions to WP2

Contributions to WP3

Goals in 2017

Julian M. Kunkel
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
- Source-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 (plugged 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 defining 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ratio</th>
<th>Throughput [MiB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Compr.</td>
</tr>
<tr>
<td>sigbits</td>
<td>0.45</td>
<td>464.4</td>
</tr>
<tr>
<td>sigbits,lz4</td>
<td>0.23</td>
<td>401.6</td>
</tr>
<tr>
<td>zfp-precision</td>
<td>0.3</td>
<td>194.8</td>
</tr>
</tbody>
</table>

Table: For ECHAM data files

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ratio</th>
<th>Throughput [MiB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Compr.</td>
</tr>
<tr>
<td>sigbits</td>
<td>0.39</td>
<td>520.9</td>
</tr>
<tr>
<td>sigbits,lz4</td>
<td>0.39</td>
<td>499.8</td>
</tr>
<tr>
<td>zfp-precision</td>
<td>0.32</td>
<td>170.1</td>
</tr>
</tbody>
</table>

Table: For 5 different random patterns
WP2: Results for Absolute Tolerance

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

Data file from ECHAM (sorted by ratio of abstol,iz4)
WP2: Results for Absolute Tolerance
### WP2: Results for Absolute Tolerance

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ratio</th>
<th>Throughput [MiB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Compr.</td>
</tr>
<tr>
<td>abstol</td>
<td>0.22</td>
<td>269.3</td>
</tr>
<tr>
<td>abstol,lz4</td>
<td>0.08</td>
<td>253.4</td>
</tr>
<tr>
<td>sz</td>
<td>0.08</td>
<td>66.0</td>
</tr>
<tr>
<td>zfp-abstol</td>
<td>0.24</td>
<td>263.9</td>
</tr>
</tbody>
</table>

**Table:** For ECHAM data files

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ratio</th>
<th>Throughput [MiB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Compr.</td>
</tr>
<tr>
<td>abstol</td>
<td>0.229</td>
<td>260.1</td>
</tr>
<tr>
<td>abstol,lz4</td>
<td>0.230</td>
<td>257.9</td>
</tr>
<tr>
<td>sz</td>
<td>0.252</td>
<td>46.4</td>
</tr>
<tr>
<td>zfp-abstol</td>
<td>0.387</td>
<td>163.1</td>
</tr>
</tbody>
</table>

**Table:** For 5 different random patterns
WP2: Experiments

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

- **Lossless compression**
  - Algorithms: memcpy 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: Nabech: 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)