Automatic Code Transformation of netCDF Code for I/O Optimisation

Jannek Squar, Anna Fuchs, Michael Kuhn and Thomas Ludwig

2023-09-14

Universität Hamburg
jannek.squar@uni-hamburg.de

Perspective of a Domain Expert

- Basic computer science skills [AC]
- HPC skills barely taught [MHS+20]
- Selfdescribing data formats (e.g. netCDF)
- Goal: Compute larger problem sizes
  - Advantage: Higher explanatory power
    - Larger area
    - Higher resolution
    - Additional dimensions
  - Disadvantage: Higher memory requirements
    - Limits imposed by memory
    - Longer I/O times

→ Easy prototyping using CATO

LLVM Infrastructure

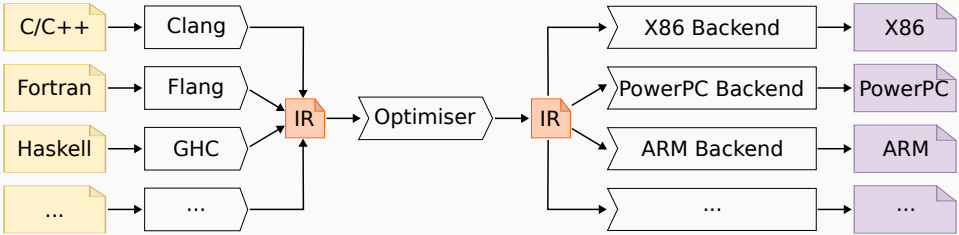

Figure 1: Modular LLVM infrastructure (based on [Lat]).

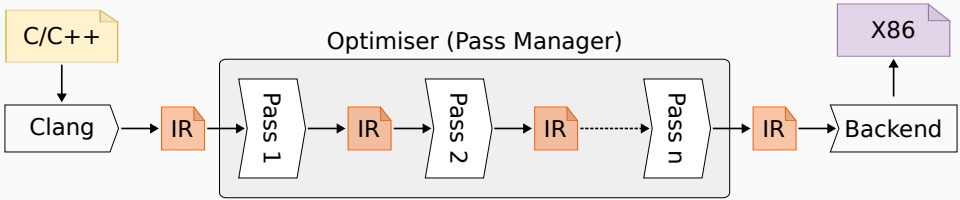

Figure 2: Insertion of CATO Pass (based on [Sam15]).

Usage:

- Use CATO wrapper instead of direct compiler call
- Optional environment variables

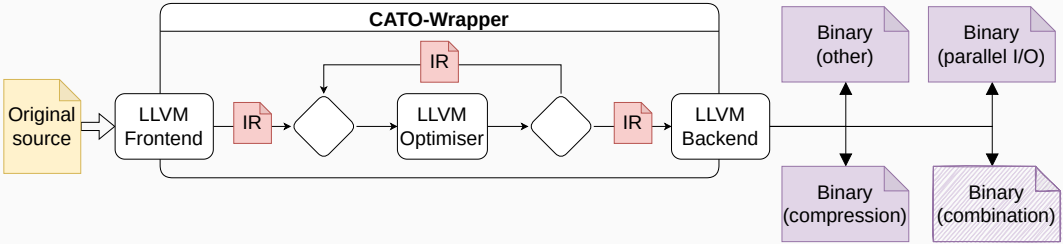

Figure 3: Workflow of CATO to create the modified binary

Fast prototyping using CATO [SJB+]

- Distribute heap variables
- Adjust netCDF code: parallel I/O and compression

Simple Example - Code

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <omp.h>
4
5  int main(int argc, char const *argv[])
6  {
7    double *p = malloc(sizeof(double) * 64);
8    double a = 0.0;
9    double b[2] = {1.0,1.0};
10   double d = 10.0;
11
12 #pragma omp parallel for shared(a,b,d)
13   for (size_t i = 0; i < 2; i++)
14   {
15     double c = 2.0 ;
16 #pragma omp critical
17       a = 3.0;
18 #pragma omp critical
19       b[i] = i*d;
20   }
21
22   free(p);
23   return 0;
24 }
```

Simple Example - Visualisation

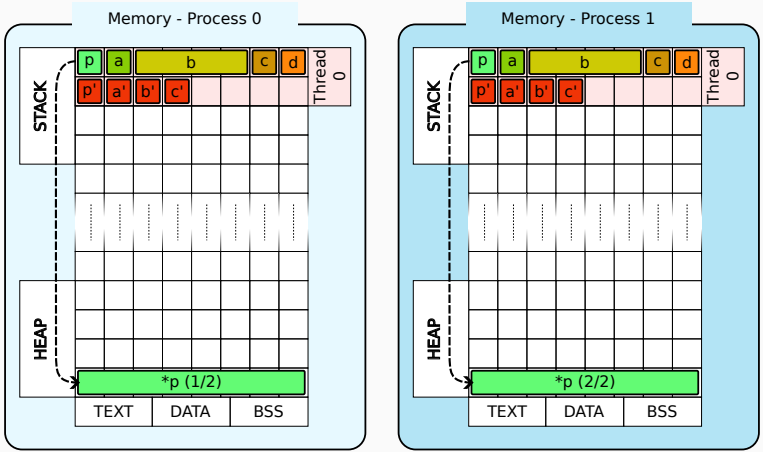

Figure 4: Handling/Distribution of stack and heap variables

Optimisation of Memory Usage

Memory Distribution:

- Focus on heap variables
  - Same input, larger runtime variables
  - Larger input, same runtime variables
- User chooses runtime parameters

$\rightarrow$ Improve memory and storage handling

Automatic parallelisation of netCDF I/O kernels

NetCDF on GitHub

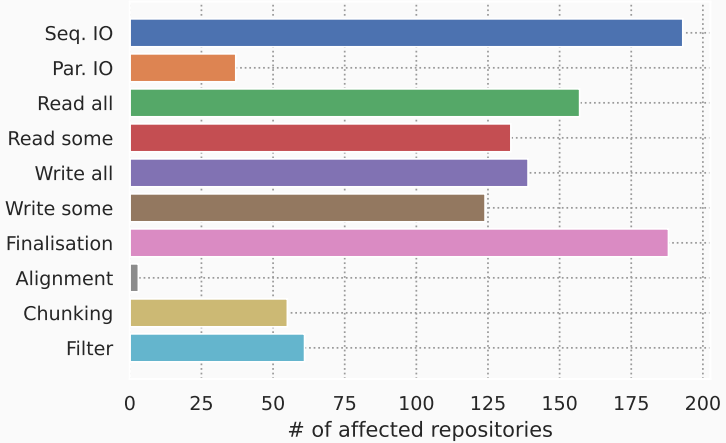

Figure 5: Popular netCDF functions (C/C++)

Unmodified netCDF kernel

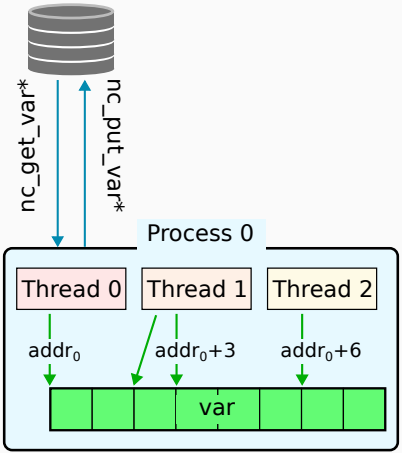

Figure 6: Serial access

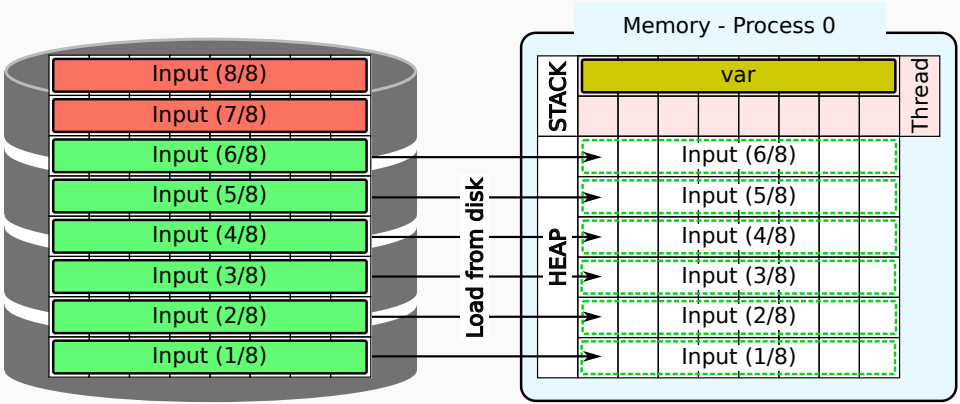

Figure 7: Memory limitation

Modified netCDF kernel

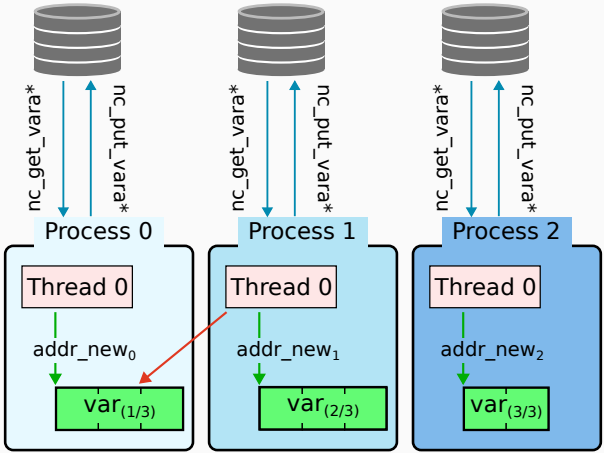

Figure 8: Distributed access using distributed I/O servers

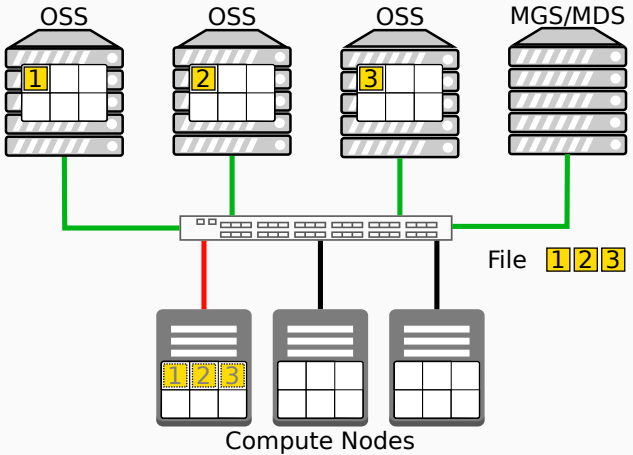

Figure 9: Stripes are loaded from single compute node

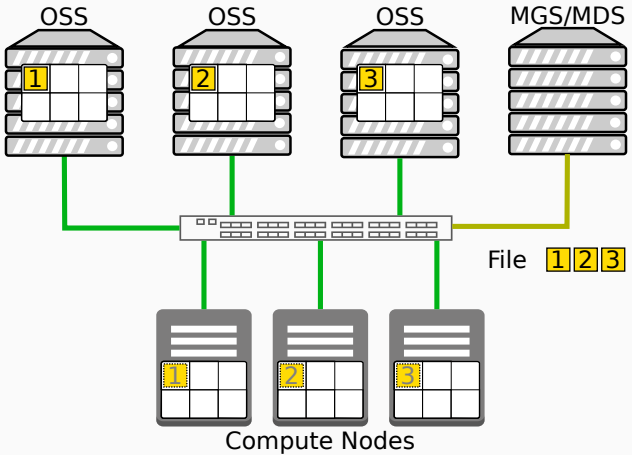

Figure 10: Stripes are loaded from multiple multiple compute nodes

Sequence within CATO

1. Automatic distribution of heap variable
2. Analysis of serial netCDF I/O calls
3. Replacement with parallel netCDF I/O calls
4. Adjustment of read/write range of netCDF calls

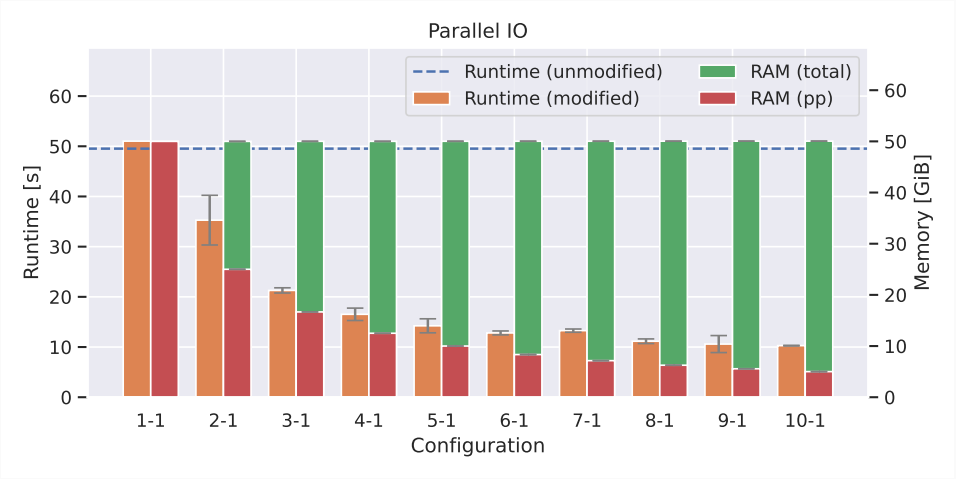

Figure 11: Runtime and memory consumption (DKRZ Levante, reading 50GiB file)

Automatic insertion of netCDF compression

Use Case: Spatial Access

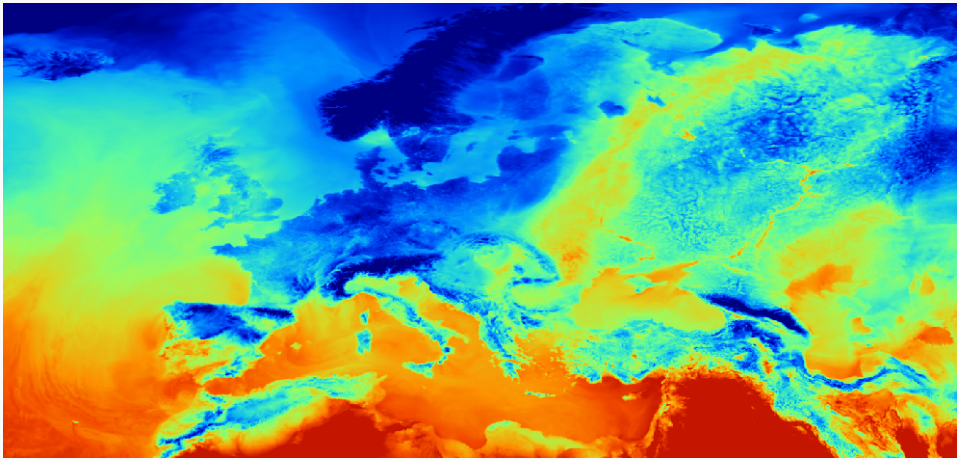

Figure 12: DWD ICON EU-nested 2m temperature (18.05.2023)

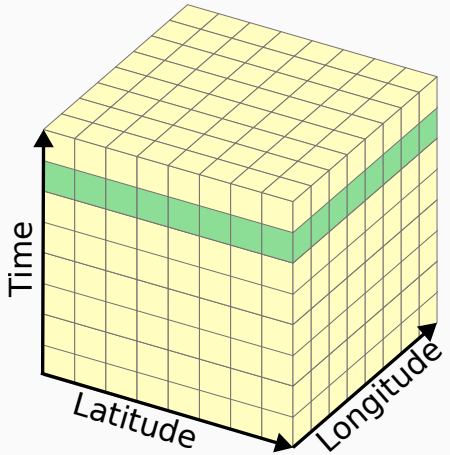

Figure 13: Spatial access

Use Case: Temporal Access

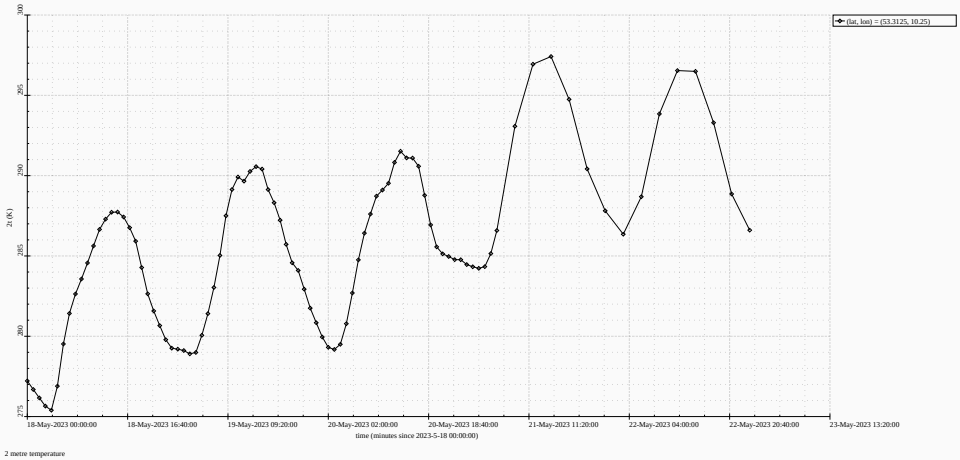

Figure 14: DWD ICON EU-nested 2m temperature time series (18.05.2023 (HH))

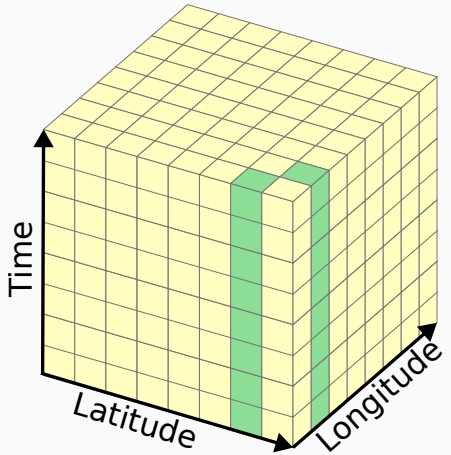

Figure 15: Time access

Improved chunking

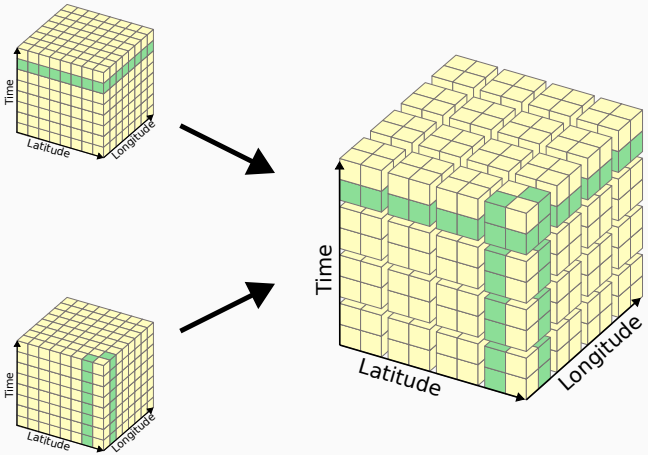

Figure 16: Combine different chunk orientations into one

Sequence within CATO

1. Analysis of variable definitions
2. Set chunk definitions
3. Set data alignment
4. Add compression filter

- Using parallel I/O and compression makes chunking mandatory
- Optimal chunking depends on use case

→ Domain expert may provide expert knowledge

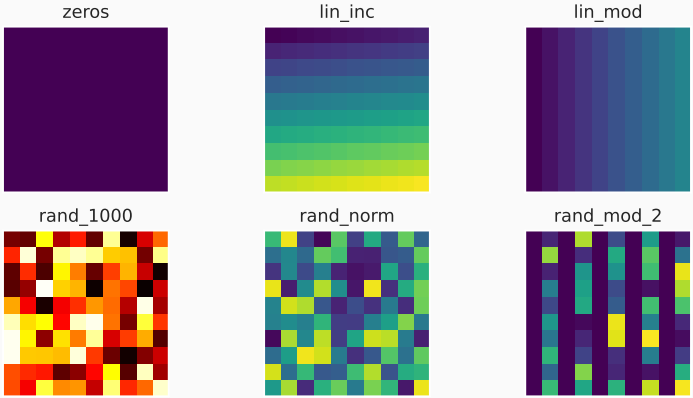

Figure 17: Visualisation of evaluated write patterns

Compressed Filesize

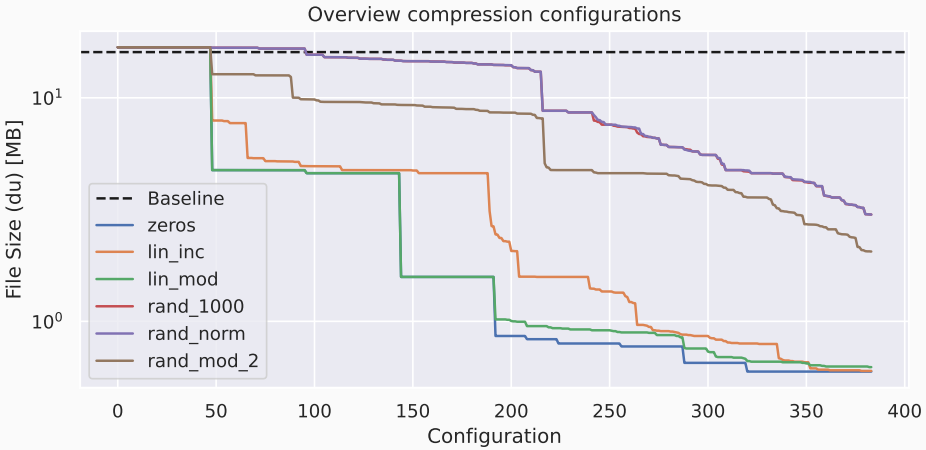

Figure 18: Configurations (compressor + level + chunking + alignment + preprocessing))

Conclusion

- Many data reduction techniques available
- Tool assistance
- Intel QuickAssist
- Lustre + ZFS
- Data formats

CATO:

- Easy usage
- No previous knowledge or code changes required
- Fast prototyping of diverse topics

Future Work

General:

- Reinsert OpenMP $\rightarrow$ less idle cores
- Improve memory management
- Pattern recognition
- Show code changes

netCDF:

- Consider all features
- Improve automatic determination of best configuration

Summary

- Fast moving HPC environment can be difficult for domain experts:
  - Many approches
  - Usually not trivial to apply
- CATO acts as a toolbox
  - Focus on earth sciences
  - Trivial usage within user space
  - Decelerated runtime during compute phase
  - Higher efficiency of memory usage

References

[AC] Ankit Agrawal and Alok Choudhary, Perspective: Materials informatics and big data: Realization of the “fourth paradigm” of science in materials science, no. 5, 053208.

[Lat] Chris Lattner, The architecture of open source applications, Online, Last accessed: 2023-05-12.

[MHS+20] Glen MacLachlan, Jason Hurlburt, Marco Suarez, Kai Leung Wong, William Burke, Terrence Lewis, Andrew Gallo, Jaroslav Flidr, Raoul Gabiam, Janis Nicholas, and Brian Ensorgauging, Building a shared resource HPC center across university schools and institutes: A case study, CoRR **abs/2003.13629** (2020).

[Sam15] Adrian Sampson, LLVM for Grad Students, Online, August 2015, Last accessed: 2023-05-12.

[SJB+] Jannek Squar, Tim Jammer, Michael Blesel, Michael Kuhn, and Thomas Ludwig, Compiler Assisted Source Transformation of OpenMP Kernels, 2020 19th International Symposium on Parallel and Distributed Computing (ISPDC), IEEE, pp. 44–51.