Bibliotheken

Hochleistungs-Ein-/Ausgabe

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Michael Kuhn

2019-05-07

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

Bibliotheken

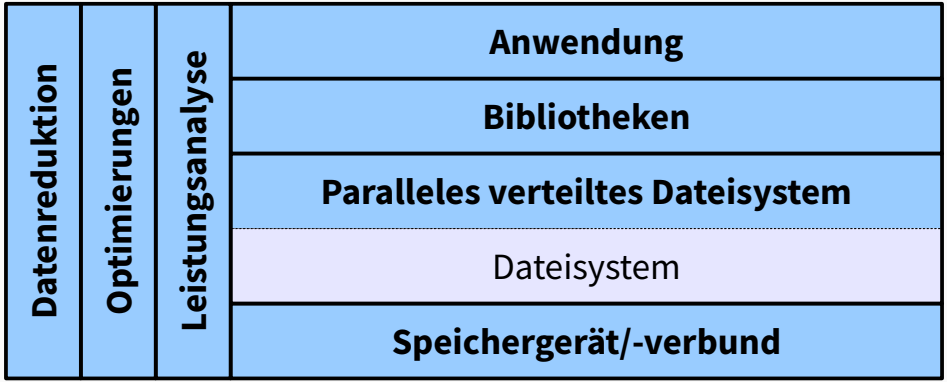

Abbildung 1: E/A-Schichten und orthogonale Themen

Motivation

- POSIX und MPI-IO können für parallele E/A genutzt werden
  - Beide Schnittstellen sind allerdings nicht sehr komfortabel
- Benutzung in wissenschaftlichen Anwendungen ist problematisch
  - Austauschbarkeit von Daten
    - Eingeschränkte Datenportabilität
  - Aufwendige Programmierung
    - Byte- bzw. element-orientierter Zugriff
  - Leistungsprobleme
    - Genaue Kenntnis des Speichersystems notwendig

Motivation...

- Bibliotheken stellen zusätzliche Funktionalität bereit
  - Selbstbeschreibende Daten
    - Daten können ohne Vorkenntnisse gelesen werden
  - Interne Strukturierung
    - Es können beispielsweise mehrere Variablen gespeichert werden
  - Abstrakte E/A-Definition
    - Keine manuellen Funktionsaufrufe notwendig
- Außerdem Umgehung von Leistungsproblemen
  - Verursacht beispielsweise durch zu strikte Semantik

Motivation...

- Vorteile
  - Portabilität der Daten
  - Komfortable Benutzung
  - Einfachheit der Programmierung
- Nachteile
  - Zusätzliche Schichten benötigt
  - Komplexes Zusammenspiel

Ansätze

- Leistungssteigerung
  - SIONlib
- Selbstbeschreibende Datenformate
  - NetCDF (Network Common Data Form)
  - HDF (Hierarchical Data Format)
- Abstrakte E/A-Definition
  - ADIOS (Adaptable IO System)

- SIONlib bietet effizienten Zugriff auf prozess-lokale Dateien
- Zugriffe werden auf eine oder wenige physikalische Dateien abgebildet
  - Je nachdem was optimale Leistung verspricht
  - Ausrichtung an Dateisystemblöcken/-streifen
    - Ziel: Reduzierung/Eliminierung von Sperrenoverhead
- Möglichst rückwärtskompatibel
  - Wrapper für `fread` und `fwrite`
  - Öffnen und Schließen benötigen spezielle Funktionen

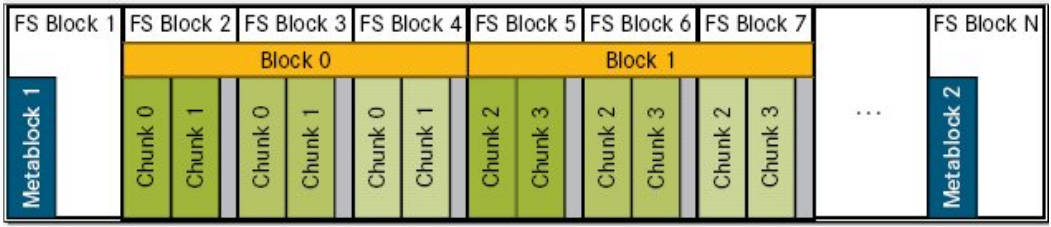

Abbildung 2: SIONlib-Dateiformat [2]

- Daten eines Prozesses innerhalb eines Dateisystemblocks
  - Zusätzliche Unterteilung in Chunks
  - Mehrere Dateisystemblöcke zu einem Block zusammengefasst
- Metablock 1: Statische Metadaten (beim Öffnen geschrieben)
- Metablock 2: Dynamische Metadaten (beim Schließen geschrieben)

```
1  int fd;
2  FILE* fp;
3
4  fd = sion_paropen_mpi(..., &numfiles, ..., &chunksize, &fsblocksize,
5                        ..., &fp, ...);
6  for (...) {
7      fwrite(..., fp);
8  }
9  sion_parclose_mpi(fd);
```

Listing 1: SIONlib-Beispiel für parallelen Zugriff

- `numfiles`: Anzahl der Dateien (-1 für automatische Bestimmung)
- `chunksize`: Maximale Größe eines Schreibaufrufs
- `fsblocksize`: Größe eines Dateisystemblocks (-1 für automatische Bestimmung)

- Mehrere Zugriffsmodi
  - Kollektives Öffnen mit `sion_paropen_mpi`
    - Analog zu `MPI_File_open`
  - Nicht-kollektives Öffnen mit `sion_open_rank`
    - Erlaubt Zugriff auf Chunks eines bestimmten Prozesses
  - Serieller Zugriff via `sion_open` und `sion_close`
- Intelligente Abbildung und Ausrichtung
  - Exklusive Nutzung einzelner Dateisystemblöcke durch Prozesse
    - Führt aufgrund von Padding u. U. zu nicht genutztem Speicherplatz
  - Abbildung auf internes Dateilayout
    - Weniger Arbeit für Anwendungsprogrammierer

Zusammenfassung

- SIONlib primär zur Umgehung von Unzulänglichkeiten existierender Dateisysteme
  - Einerseits Probleme mit vielen Dateien
    - Benötigt ausreichende Metadatenleistung zur Erstellung und zum Öffnen
    - Dateisysteme üblicherweise nicht für extrem viele Dateien und Verzeichnisse ausgelegt
  - Andererseits gemeinsame Dateien häufig langsam
    - Korrektes Zugriffsmuster sehr wichtig, da POSIX Sperren notwendig macht

- Entwickelt vom Unidata Program Center
  - University Corporation for Atmospheric Research
- Projekt wurde 1989 gestartet
  - Basiert auf dem Common Data Format der NASA
- Hauptsächlich in wissenschaftlichen Anwendungen
  - Insbesondere in den Klimawissenschaften, der Meteorologie und der Ozeanographie
- Besteht aus Bibliotheken und Datenformaten

- Es exisitieren drei Formate
  1. Klassisches Format (CDF-1)
  2. Klassisches Format mit 64-Bit-Offsets (CDF-2)
  3. Klassisches Format mit vollständiger 64-Bit-Unterstützung (CDF-5)
  4. NetCDF-4-Format
- Datenformate sind offene Standards
  - CDF-1 und CDF-2 sind internationale Standards des Open Geospatial Consortiums
- CDF-1 und CDF-2 sind eigenständige Formate
  - NetCDF-4 nutzt HDF5

- Zuerst keine Unterstützung für parallele E/A in CDF-1 und CDF-2
  - Daher Entwicklung von Parallel-NetCDF mit inkompatibler Schnittstelle
- Ab NetCDF-4 Unterstützung für parallele E/A via HDF5
  - Allerdings nur für das NetCDF-4-Format
- Mit aktuellen NetCDF-Versionen parallele E/A für alle Formate möglich
  - Unterstützung für NetCDF-4-Dateien mittels HDF5
  - Unterstützung für CDF-1, CDF-2 und CDF-5 mittels Parallel-NetCDF

- Schnittstellen für viele Sprachen
  - C, Fortran, C++, Java, R, Perl, Python, Ruby etc.
- Datenformat ist architekturunabhängig
  - Endianness-Konvertierung
- NetCDF unterstützt Gruppen und Variablen
  - Gruppen enthalten Variablen
  - Variablen enthalten Daten
- Zusätzliche Attribute für Variablen

Funktionalität...

- Unterstützung für mehrdimensionale Arrays
  - `char`, `byte`, `short`, `int`, `float` und `double`
    - Seit CDF-5 zusätzlich ubyte, ushort, uint, int64 und uint64
  - NetCDF-4: `ubyte`, `ushort`, `uint`, `int64`, `uint64` und `string`
- Größe der Dimensionen ist beliebig
  - Bei CDF-1, CDF-2 und CDF-5 nur eine unbeschränkt
    - Beispiel: Matrix kann nur in der Zeitdimension wachsen
  - Bei NetCDF-4 beliebig viele unbeschränkt
    - Unterstützung nur mit komplexerem Datenformat möglich

Funktionalität...

- Zusätzliche Funktionen
  - Transparente Kompression
- Diverse Werkzeuge verfügbar
  - Beispielsweise `ncdump`, um Daten auszugeben
  - NetCDF Operators (NCO)
- Darauf aufbauende Standards
  - Das Climate Data Interface unterstützt u. a. NetCDF

Beispiel

```
1  netcdf ... {
2  dimensions:
3      time = UNLIMITED ; // (8760 currently)
4  variables:
5      double time(time) ;
6          string time:units = "days" ;
7          string time:long_name = "Julian_date" ;
8
9  // global attributes:
10         string :Conventions = "None" ;
11         string :creation_date = "Wed Jul 16 12:52:44 CEST 2014" ;
12 }
```

Listing 2: ncdump-Ausgabe

1. Datei anlegen mit `nc_create`
   - Paralleler Zugriff mit `nc_create_par`
     - Backend kann mit `NC_MPIIO` bzw. `NC_NETCDF4` festgelegt werden
2. Dimensionen definieren mit `nc_def_dim`
3. Gruppen definieren mit `nc_def_grp`
4. Variablen definieren mit `nc_def_var`
5. Attribute schreiben mit `nc_put_att_*`
6. Variablen schreiben mit `nc_put_var_*`
7. Datei schließen mit `nc_close`

Funktionsweise...

- Lesen unterscheidet zwei Fälle
  1. Dateistruktur ist unbekannt
     - Verfügbare Gruppen und Variablen müssen zuerst bestimmt werden
  2. Dateistruktur ist bekannt
     - Zugriff über Gruppen- und Variablennamen

1. Öffnen der Datei mit `nc_open`
   - Paralleler Zugriff mit `nc_open_par`
2. Gruppen-IDs auslesen mit `nc_inq_ncid`
3. Variablen-IDs auslesen mit `nc_inq_varid`
4. Variablen auslesen mit `nc_get_var`
5. Datei schließen mit `nc_close`

Funktionsweise...

- Unterschiedliche Modi
  - Nach dem Anlegen einer neuen Datei im Define Mode
  - Nach dem Öffnen einer existierenden Datei im Data Mode
- Define Mode erlaubt das Ändern der Dateistruktur
  - Hinzufügen von Dimensionen, Attributen und Variablen
  - Einige Einstellungen nur direkt nach Definition änderbar
    - Unter anderem Kompression, Byte-Reihenfolge, Fehlerkorrektur und Füllwert
- Data Mode erlaubt das Speichern von Daten
- Bei NetCDF-4 automatischer Moduswechsel
  - Ansonsten `nc_redef` bzw. `nc_enddef` notwendig

- Alternativer Ansatz für parallele E/A
  - Unterstützt CDF-1, CDF-2 und CDF-5
- Entwickelt durch Northwestern University und Argonne National Laboratory
  - Teilweise dieselben Entwickler wie MPI-IO und OrangeFS
- Schnittstelle ist inkompatibel
  - NetCDF-4 kann aber Parallel-NetCDF nutzen

Überblick

- Besteht aus Dateiformaten und Bibliotheken
  - Erlaubt Verwaltung selbstbeschreibender Daten
- Aktuelle Version ist HDF5
  - HDF4 wird immer noch aktiv unterstützt
- Probleme mit Vorversionen
  - Komplizierte API
  - Einschränkungen wie z. B. 32-Bit-Adressierung

Überblick

- Unterstützt Gruppen und Datensätze
  - Datensätze speichern Daten
  - Gruppen strukturieren den Namensraum
  - Analog zu Dateien und Verzeichnissen
- Gruppen können Datensätze und Gruppen enthalten
  - Hierarchischer Namensraum
- Attribute für Datensätze und Gruppen
  - Beispielsweise Minimum und Maximum

- Objekte werden über POSIX-ähnliche Pfade zugegriffen
  - Beispielsweise `/path/to/dataset`
  - Pfad kann Informationen über Daten enthalten
- HDF-Dateien sind selbstbeschreibend
  - Können ohne vorheriges Wissen über Struktur und Inhalt geöffnet werden
  - Annotationen erlauben Interpretation der Daten

- Datensätze können mehrdimensionale Arrays eines Basisdatentypen speichern
  - Integer, Float, Character, Bitfield, Opaque, Enumeration, Reference, Array, Variable-length und Compound
- Datensätze haben Eigenschaften
  - Größe, Genauigkeit, Byte-Reihenfolge etc.
- Beliebig viele unbeschränkte Dimensionen

```
HDF5 "..." {
GROUP "/" {
   ATTRIBUTE "creation_date" {
      DATATYPE  H5T_STRING {
         STRSIZE H5T_VARIABLE;
         STRPAD H5T_STR_NULLTERM;
         CSET H5T_CSET_ASCII;
         CTYPE H5T_C_S1;
      }
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
}
}
```

Listing 3: h5dump-Ausgabe

- Sprachspezifische Datenspeicherung
  - Daten werden nach C-Konvention zeilenweise gespeichert
  - Fortran-Daten werden automatisch umgewandelt

1	2	3
4	5	6
7	8	9

Abbildung 3: 3x3-Matrix

1	2	3	4	5	6	7	8	9

(a) C-Speicherlayout

1	4	7	2	5	8	3	6	9

(b) Fortran-Speicherlayout

- Mithilfe von Chunking werden Datensätze in kleinere Stücke aufgeteilt
  - U. a. notwendig für Kompression und andere Filter
- Chunking erlaubt Daten in allen Dimensionen zu erweitern
  - Bei zusammenhängender Speicherung nicht möglich
- Mögliche Optimierungen
  - Anpassung an Streifenbreite oder effizienter spaltenweiser Zugriff
- Zusatzaufwand
  - Verwaltung mit Chunk-Index
  - Üblicherweise geringere Leistung

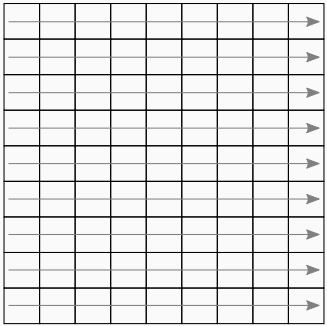

Abbildung 4: Zusammenhängender Datensatz

- Datensätze werden zusammenhängend in der Datei gespeichert

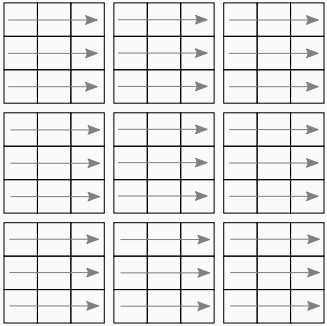

Abbildung 5: Datensatz mit Chunking

- Datensätze werden in mehrere Chunks aufgeteilt
  - Es werden immer komplette Chunks gelesen bzw. geschrieben

- Unterstützung für mehrere Backends (VFL)
  - Aktuell POSIX und MPI-IO
    - MPI-IO erlaubt parallelen Zugriff auf gemeinsame HDF-Dateien
- Diverse Werkzeuge verfügbar
  - Beispielsweise `h5dump`, um Daten auszugeben
- Zusätzliche Funktionen
  - Transparente Kompression und benutzerdefinierte Filter
- Wird aktiv weiterentwickelt
  - Virtual Object Layer (VOL) erlaubt alternative Speicheransätze
    - Im Gegensatz zu VFL auf Basis von HDF5-Objekten
  - Weitere Funktionen für Exascale

- ADIOS ist stark abstrahiert
  - Kein byte- oder element-orientierter Zugriff
  - Direkte Unterstützung für Anwendungsdatenstrukturen
- Entworfen für hohe Leistung
  - Insbesondere von wissenschaftlichen Anwendungen
  - Caching, Zusammenfassen von Operationen etc.

- E/A-Konfiguration wird in XML-Datei ausgelagert
  - Beschreibt relevante Datenstrukturen
  - Wird benutzt, um automatisch Code zu erzeugen
- Entwickler spezifizieren E/A auf hoher Abstraktionsstufe
  - Kein Kontakt mit Middleware oder Dateisystem
  - Einige Änderungen ohne Neukompilation möglich

```
<adios-config host-language="C">
  <adios-group name="checkpoint">
    <var name="rows" type="integer"/>
    <var name="columns" type="integer"/>
    <var name="matrix" type="double" dimensions="rows,columns"/>
  </adios-group>
  <method group="checkpoint" method="MPI"/>
  <buffer size-MB="100" allocate-time="now"/>
</adios-config>
```

Listing 4: ADIOS-XML-Konfiguration

- Variablen werden in Gruppen zusammengefasst
  - Pro Gruppe Festlegung der E/A-Methode
- Puffergrößen können gesetzt werden

```
1 adios_open(&adios_fd, "checkpoint", "checkpoint.bp", "w",
      ↪ MPI_COMM_WORLD);
2 #include "gwrite_checkpoint.ch"
3 adios_close(adios_fd);
```

Listing 5: ADIOS-Code

- Code wird mit `gpp.py` generiert
  - `gread_checkpoint.ch` und `gwrite_checkpoint.ch`
- Entwickler müssen nur entsprechenden Header einbinden
  - Und ein wenig auf Variablennamen etc. achten

```
1 adios_groupsize = 4 \
2                 + 4 \
3                 + 8 * (rows) * (columns);
4 adios_group_size (adios_handle, adios_groupsize, &adios_totalsize);
5 adios_write (adios_handle, "rows", &rows);
6 adios_write (adios_handle, "columns", &columns);
7 adios_write (adios_handle, "matrix", matrix);
```

Listing 6: ADIOS-Header für das Schreiben

- Gruppengröße wird automatisch berechnet und gesetzt
- `adios_write`-Aufrufe schreiben Daten
  - Schreibvorgänge werden möglichst im Cache abgewickelt

```
1 s = adios_selection_writeblock (rank);
2 adios_schedule_read (fp, s, "matrix", 1, 1, matrix);
3 adios_perform_reads (fp, 1);
4 adios_selection_delete (s);
```

Listing 7: ADIOS-Header für das Lesen

- Lesen komplexer als Schreiben
  - Bietet aber auch zusätzliche Funktionalität
- Ausschnitte der Daten können selektiert werden
  - ADIOS bestimmt selbstständig optimale Lesestrategie
- Mehrere Leseoperationen können geplant werden
  - Spezifizierung von Schritten, anschließend eigentliche Ausführung

Funktionalität

- Eigenes Dateiformat (BP)
  - Kann in HDF5, NetCDF und ASCII konvertiert werden
- Unterstützt Datentransformationen
  - Unter anderem Kompression
- Read-Scheduling für effiziente Ausführung
  - Erlaubt beispielsweise Staging von Daten
- Zusätzliche Funktionalität
  - `adios_{start,stop}_calculation`: Markierung der Berechungsphasen, um E/A parallel zu Berechnungen ausführen zu können
  - `adios_end_iteration`: Stellt Timinginformationen bereit

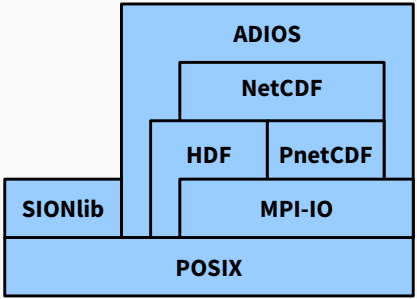

Abbildung 6: Interaktion zwischen Bibliotheken

- Bibliotheken bauen mitunter auf mehreren anderen Bibliotheken auf
  - Unter Umständen Kenntnis aller Abhängigkeiten und der Optimierungen notwendig

Disjoint [1]

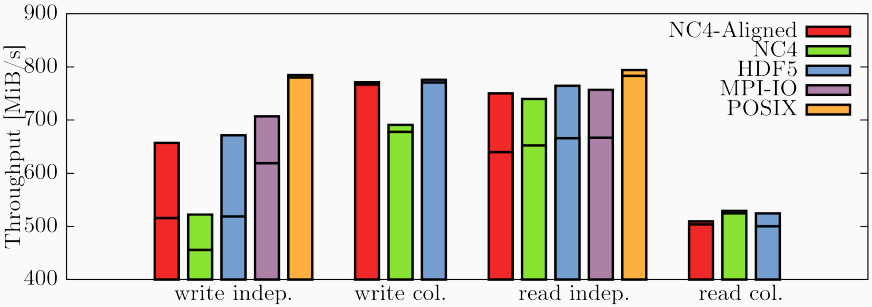

Fig. 5. Disjoint Pattern

- Clients für zusammenhängende Datenblöcke verantwortlich
  - Keine Überlappungen, um Konflikte zu vermeiden
- Jeder Client kommuniziert potentiell mit allen Servern

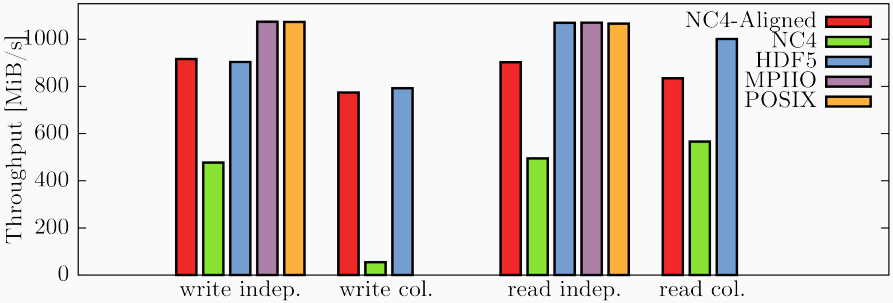

Fig. 6. 1-OST Pattern

- Jeder Client kommuniziert mit genau einem Server
  - Weniger Kommunikationsoverhead und potentielle Konflikte

Zugriffs- und Chunk-Größen [1]

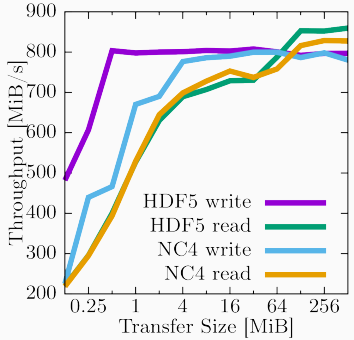

Fig. 7. Varying Transfer Size

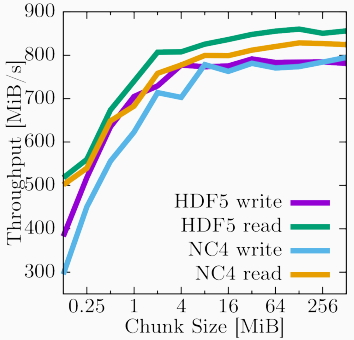

Fig. 8. Chunked Layout

- Leistung stark abhängig von Zugriffs- und Chunk-Größen

1- bis 10-OST [1]

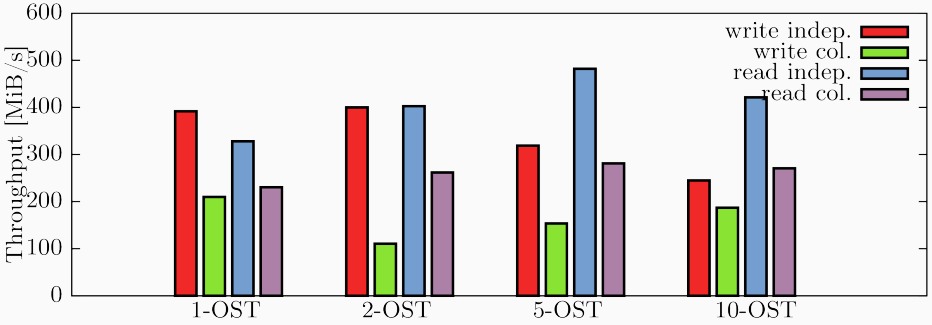

Fig. 9. 1- to 10-OST Pattern, HDF5 with Chunked I/O

- Leistung variiert je nach Anzahl der kontaktierten Server

- Interaktion zwischen Bibliotheken ist komplex
  - Erreichbare Leistung kann nicht einfach vorhergesagt werden
  - Mögliche Leistungsprobleme und Optimierungen auf mehreren Schichten
  - Analyse ist schwierig, da alle Schichten erfasst werden müssen
- Optimierung für vorhandenes System notwendig
  - HDF5 und SIONlib erlauben Anpassung an Dateisystemgrenzen
    - NetCDF hat dafür nur beschränkte Unterstützung

Zusammenfassung

- E/A-Schnittstellen sind häufig nicht komfortabel zu nutzen
  - E/A-Bibliotheken erlauben u. a. strukturierten Zugriff
    - Zusätzliche Annotationen und Metadaten erlauben einfachen Austausch
- Zoo von Bibliotheken für unterschiedliche Anwendungszwecke
  - Analyse von Fehlern und Leistungsproblemen schwierig
  - SIONlib umgeht Leistungsprobleme aktueller Dateisysteme
  - NetCDF und HDF bieten ähnliche Funktionalität
    - Beide erlauben parallele E/A

Quellen

[1] Christopher Bartz, Konstantinos Chasapis, Michael Kuhn, Petra Nerge, and Thomas Ludwig. **A Best Practice Analysis of HDF5 and NetCDF-4 Using Lustre.** In Julian Martin Kunkel and Thomas Ludwig, editors, *High Performance Computing*, number 9137 in Lecture Notes in Computer Science, pages 274–281, Switzerland, 06 2015. Springer International Publishing.

[2] SIONlib. **File Format.** `https://apps.fz-juelich.de/jsc/sionlib/docu/fileformat_page.html`.