

# 8. Software Libraries

ALDaBi Praktikum

# Inhalt

## Software Libraries

- MCSTL (Performance)
- STXXL (Large Data)
- SeqAn (Sequenzanalyse)
- Boost

## Outro

# MCSTL

Multi-Core Standard Template Library

# MCSTL

## Multi-Core Standard Template Library

- Besteht aus **parallelisierten** Algorithmen und Datenstrukturen der STL
- Implementierungen benutzen OpenMP und atomare Operationen (`fetch_and_add`, `compare_and_swap`)

## Parallel implementiert sind u.a.:

- `find`, `find_if`, `mismatch`, ...
- `partial_sum` (**kumulative Summe**)
- `partition`
- `nth_element`/`partial_sort`
- `merge`
- `sort`, `stable_sort`
- `random_shuffle`
- `for_each`, `transform` (**embarrassingly parallel**)

# Verfügbarkeit

## Plattformen:

- Linux, Mac OS X
- Windows (prinzipiell möglich)

## Lizenz:

- Boost Software License 1.0 und GPL v2

## MCSTL ist Teil des g++ 4.3 (und höher)

- Heisst seitdem *libstdc++ parallel mode*
- Für g++ 4.2 (und vorherige) kann sie separat installiert werden

# Benutzung (explizit)

MCSTL-Algorithmen sind definiert

- in Header-Dateien <parallel/...> im Namensraum `__gnu_parallel`

```
#include <vector>
#include <parallel/algorithm>

int main() {
    std::vector<int> v(100);
    // ...
    std::sort(v.begin(), v.end());           // sequentiell
    __gnu_parallel::sort(v.begin(), v.end()); // parallel
    return 0;
}
```

# Benutzung (implizit)

Es geht auch ohne den Quelltext zu verändern mit einer Definition:

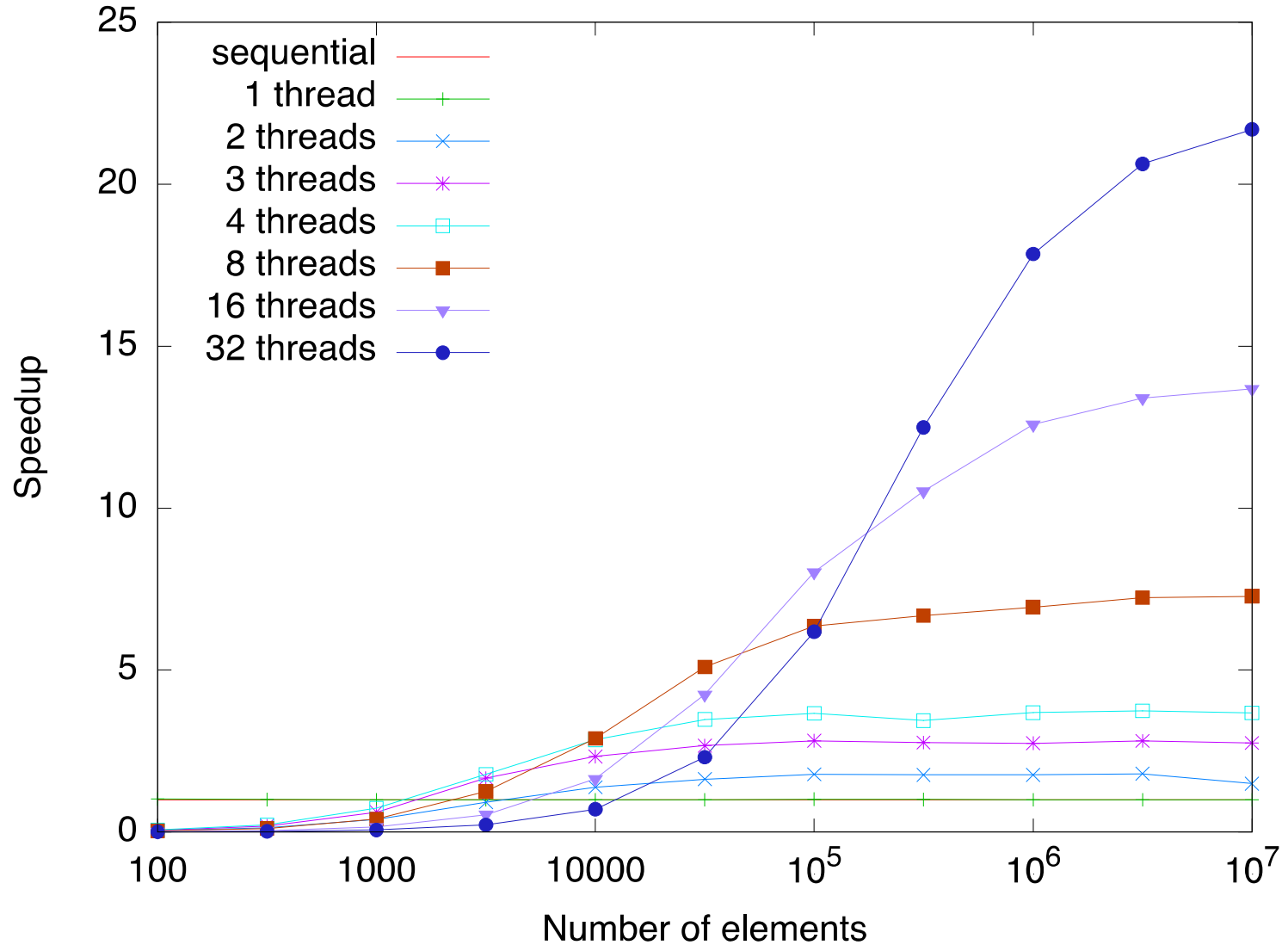
– `g++ -D_GLIBCXX_PARALLEL beispiel.cpp`

```
// #define _GLIBCXX_PARALLEL // <-- kann auch hier definiert werden
#include <vector>

int main() {
    std::vector<int> v(100);
    // ...
    std::sort(v.begin(), v.end()); // parallel
    std::sort(v.begin(), v.end(), //
              __gnu_parallel::sequential_tag()); //
    sequentiell
    return 0;
}
```

Sequentielle Algorithmen können noch mit dem Parameter  
`__gnu_parallel::sequential_tag()` ausgewählt werden

# Paralleles Sortieren - Speedup





# STXXL

Standard Template Library for Extra Large Data Sets

# STXXL

## Standard Template Library for Extra Large Data Sets

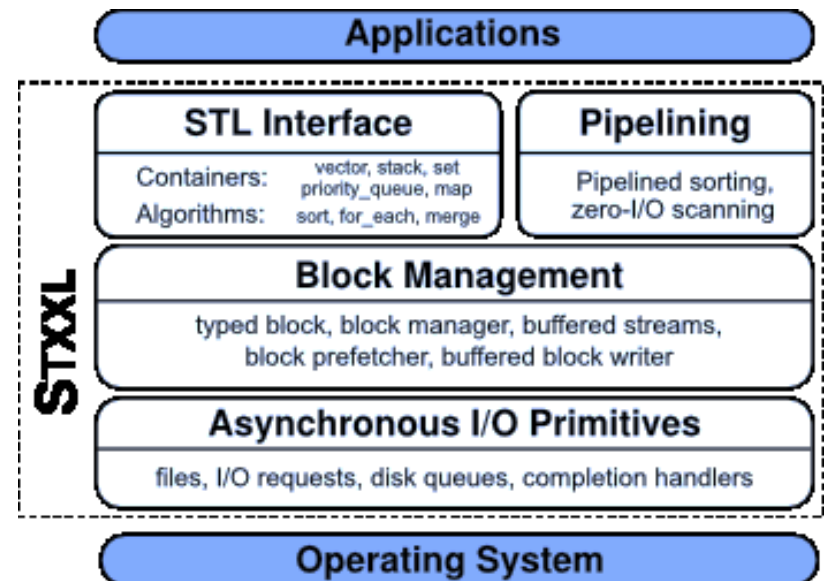
- Externspeicher-Datenstrukturen und Algorithmen
- STL-Interface für einfache Integration
- Transparentes Arbeiten mit Datenstrukturen, die nur noch auf der Festplatte Platz haben
- Unterstützt mehrere Festplatten parallel
- Benutzt optional die MCSTL

### Container:

- `vector`, `stack`, `set`
- `priority_queue`, `map`

### Algorithmen:

- `sort`, `for_each`, `merge`



# Verfügbarkeit

## Plattformen:

- Linux, Mac OS X, FreeBSD (g++, clang++, icpc)
- Windows (VS 8-10)

## Lizenz

- Boost Software License 1.0

# Benutzung

STXXL definiert eigenen Namensraum `stxxl`:

- **Beispiel:** Externspeicher-Vektor mit Zufallszahlen füllen und sortieren

```
#include <stxxl/mng>
#include <stxxl/sort>
#include <stxxl/vector>

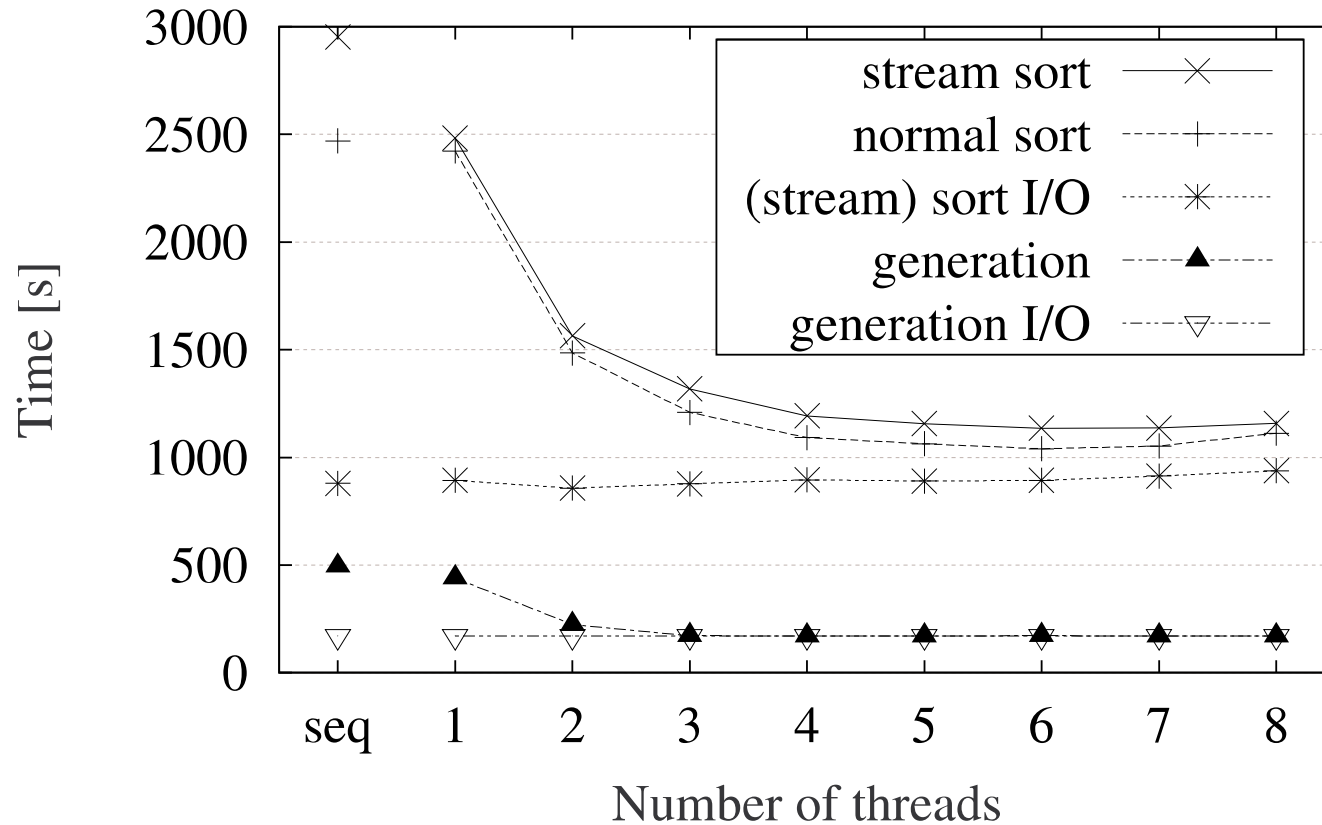
int main() {
    typedef stxxl::vector<int> TMyVector;
    unsigned memory_to_use = 128 * 1024 * 1024;
    TMyVector v(1000000000ull);

    stxxl::random_number32 rnd;
    for (TMyVector::size_type i = 0; i < v.size(); i++)
        v[i] = rnd() % 0xffffffff;

    stxxl::sort(v.begin(), v.end(), std::less<int>(), memory_to_use);
}
```

# STXXL + MCSTL

Sortieren von 100GB 64bit Ganzzahlen mit STXXL und MCSTL





C++ Library for Sequence Analysis

# SeqAn

## C++ Library for Sequence Analysis

- Datenstrukturen und Algorithmen für die Sequenzanalyse
- Schwerpunkt liegt auf Effizienz, Generik und Erweiterbarkeit
- Benutzt Techniken wie Template Subclassing, Metafunctions
- Verschiedene Implementierungen einer Datenstruktur können mit Tags ausgewählt werden

## Container:

- `String`, `StringSet`, `Modifier`
- `Align`, `Graph`, `Index`
- `Finder`, `Pattern`, ...

## Algorithmen:

- `reverseComplement`, `globalAlignment`, `find`, ...

# Verfügbarkeit

## Plattformen:

- Linux, Mac OS X, FreeBSD (g++, clang++)
- Windows (VS 8-10, mingw)

## Lizenz

- BSD License (3-clause)



# Benutzung

Alle Datenstrukturen sind im Namensraum `seqan` definiert

- In SeqAn lassen sich eigene Alphabete definieren
- Vordefiniert sind u.a.: `Dna`, `Dna5`, `AminoAcid`
- **Beispiel:** Alle Zeichen des `Dna5`-Alphabets aufzählen und ausgeben

```
#include <seqan/sequence.h>
#include <seqan/basic.h>
#include <iostream>

using namespace seqan;

int main() {
    int alphSize = ValueSize<Dna5>::VALUE;
    for (int i = 0; i < alphSize; ++i)
        std::cout << (Dna5)i << ' '; // Ausgabe: A C G T N
    return 0;
}
```

# String Matching

SeqAn definiert ein allgemeines Interface zum Suchen von Strings

- **Finder** und **Pattern** kapseln Text und Suchmuster
- 2.Template-Argument ist optional und wählt den Suchalgorithmus
- **Beispiel**: Exaktes Matching mit Horspool (P-Aufgabe 1)

```
#include <iostream>
#include <seqan/find.h>
using namespace seqan;
int main() {
    CharString haystack = "Simon, send more money!";
    CharString needle = "mo";

    Finder<CharString> finder(haystack);
    Pattern<CharString, Horspool> pattern(needle);
    while (find(finder, pattern))
        std::cout << beginPosition(finder) << std::endl;
    return 0;
}
```

# String Matching (II)

Algorithmen werden durch Tags ausgewählt

- Horspool, **ShiftAnd**, ShiftOr, Bfam, ... (exaktes Matching)
- SetHorspool, AhoCorasick, MultiBfam (multiples exaktes Matching)
- DPSearch, Pex, Myers (approximatives Matching)

```
#include <iostream>
#include <seqan/find.h>
using namespace seqan;
int main() {
    CharString haystack = "Simon, send more money!";
    CharString needle = "mo";

    Finder<CharString> finder(haystack);
    Pattern<CharString, ShiftAnd> pattern(needle);
    while (find(finder, pattern))
        std::cout << beginPosition(finder) << std::endl;
    return 0;
}
```

# Alignments

Es gibt verschiedene Spezialisierungen zum Berechnen von Alignments:

- Lokales oder (semi-)globales Alignment, Lineare oder affine Gapkosten
- Banded oder unbanded
- **Beispiel:** Globales Alignment mit Levenshtein-Scoring (P3)

```
#include <iostream>
#include <seqan/align.h>
using namespace seqan;
int main() {
    Align<CharString, ArrayGaps> align;
    resize(rows(align), 2);
    assignSource(row(align,0), "IMISSMISSISSIPPI");
    assignSource(row(align,1), "MYMISSISAHIPPIE");

    int score = globalAlignment(align, Score<int, Simple>(0,-1,-1));
    std::cout << align;
    std::cout << "Score:" << score << std::endl;
    return 0;
}
```

# Indizes

Es gibt verschiedene Textindizes:

- (Enhanced) Suffix Arrays, (Lazy) Suffix Trees, **q-gram Indizes**
- Alle unterstützen das Pattern/Finder Interface für exaktes String Matching
- Einige Indizes lassen sich benutzen wie Suffix Trees
- **Beispiel:** Bestimme alle Vorkommen eines q-grams im Text (Teil von P5)

```
#include <iostream>
#include <seqan/index.h>
using namespace seqan;
int main () {
    typedef Index<DnaString, IndexQGram< UngappedShape<3> > > TIndex;
    TIndex index("CATGATTACATA");

    hash(indexShape(index), "CAT");
    for (unsigned i = 0; i < countOccurrences(index, indexShape(index)); ++i)
        std::cout << getOccurrences(index, indexShape(index))[i] << std::endl;
    return 0;
}
```

# File I/O

SeqAn unterstützt:

- gängige Fileformate: Fasta, Fastq, GFF, GTF, Amos, SAM, ...
- Datenstrukturen (String<...>, External<> >) und Algorithmen für Externspeicher
- **Beispiel:** Reverskomplemente bilden

```
#include <iostream>
#include <fstream>
#include <seqan/file.h>
using namespace seqan;
int main() {
    std::ifstream fin("input.fa");
    std::ofstream fout("output.fa");
    Dna5String seq;
    CharString header;
    while (!_streamEOF(fin)) {
        readMeta(fin, header, Fasta());
        read(fin, seq, Fasta());
        reverseComplement(seq);
        write(fout, seq, header, Fasta());
    }
    return 0;
}
```

# Außerdem enthalten ...

## Graphen

- Gerichtet/ungerichtet
- Bäume, Tries, Automaten, HMMs, Alignmentgraphen

## Seeds

- Seed Datenstrukturen
- Seed Extension und Chaining Algorithmen

## Filter

- SWIFT q-gram Filter

## Modifiers

- Views von Containern, die deren Inhalt nicht verändern
  - Komplement, Reverses, Reversekomplement, Uppcase-String, ...

## Adaptionen bestehender Bibliotheken

- STL, Pizza'n'Chili (komprimierte Textindizes)



Portable C++ Source Libraries



# Boost

## Boost enthält

- 104 einzelne Bibliotheken
- Davon 88 reine Headerbibliotheken (wie die STL)

## Beispiele:

- **Rational** – Klasse zum Rechnen mit rationalen Zahlen
- **Interval** – Rechnen mit Intervallen
- **Multi-Array** – Generisches n-dimensionales Feld
- **Regex** und **Spirit** – Parser für reg. Ausdrücke bzw. EBNF Grammatiken
- **Concept Check** – Interfaces in generischer Programmierung
- **Foreach** und **Lambda** – Einfaches Iterieren über Sequenzen und Definieren von lambda-Funktionen

# Verfügbarkeit

## Plattformen:

- Sehr viele
- Inzwischen sind einige Bibliotheken in den TR1 (kommender C++ Standard) aufgenommen

## Lizenz

- Boost License

# Boost Beispiel

- Makros zur Erleichterung des Programmierens, wie z.B. BOOST\_FOREACH
  - Erspart Notation von Iteratoren o.Ä.

```
#include <string>
#include <fstream>
#include <boost/foreach.hpp>

int main() {
    std::string hello("Hello World!");

    BOOST_FOREACH (char c, hello)
    {
        std::cout << c;
    }
    return 0;
}
```

# Boost Beispiel (II)

- Lamda erspart Definitionen von Funktoren
  - Anwendbar z.B. beim Aufbau des Suffixarrays (P-Aufgabe 2)

```
std::vector<int> v(100);  
// ...  
std::sort(v.begin(), v.end(), *_1 > *_2);
```

- Hier: Funktion „größer als“ in der Zeile definiert
- Mehr Beispiele und ausführliche Dokumentation unter [http://www.boost.org/doc/libs/1\\_55\\_0/doc/html/lambda.html](http://www.boost.org/doc/libs/1_55_0/doc/html/lambda.html)

**OUTRO**

# Ausblick

## Interesse an C++ gefunden?

- Softwarepraktika nach dem 5. Semester:
  - **SeqAn** – Implementierung von Algorithmen zur Sequenzanalyse
  - **OpenMS** - Implementierung von Algorithmen zur Analyse massenspektrometrischer Daten

## Betreute Bachelor/Masterarbeiten

- <http://www.mi.fu-berlin.de/w/ABI/ThesesHome>
- Weitere Themen in Absprache mit Prof. Reinert

## Bitte evaluiert diese und andere Veranstaltungen:

- <http://lehrevaluation.fu-berlin.de>
- Aufruf zur Evaluation kommt (normalerweise) per Mail

**ENDE**

Danke für Eure Aufmerksamkeit!