8. Software Libraries

AlDaBi Praktikum

David Weese

René Rahn
WS 2014/15

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 benutzten OpenMP und atomare Operationen (`fetch_and_add`, `compare_and_swap`)

Parallel implementiert sind u.a.:

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

Verfügbarkeit

Platformen:

- 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

[1] **MCSTL: The Multi-Core Standard Template Library**, http://algo2.iti.kit.edu/singler/mcstl/

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)

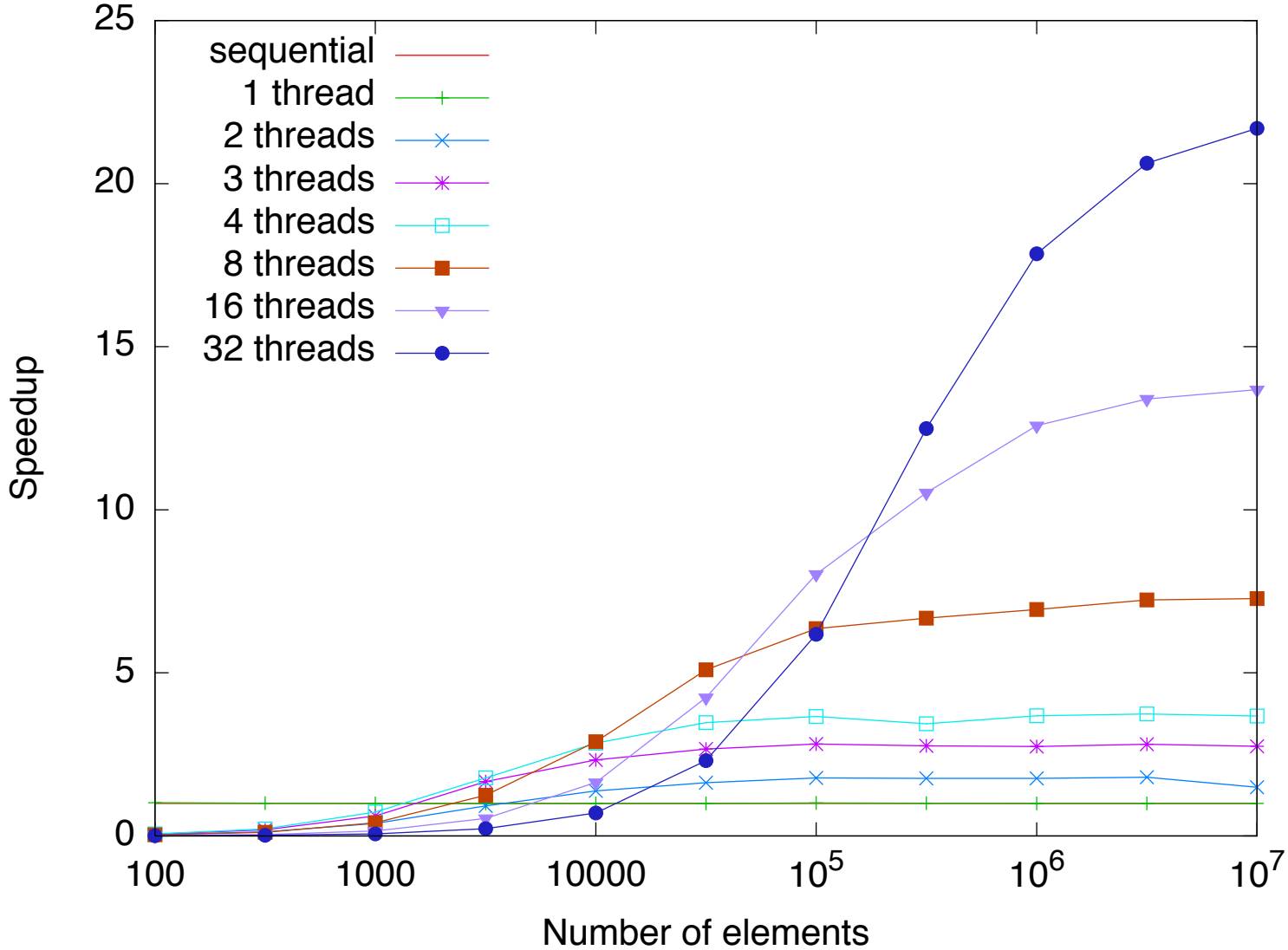
Es geht auch ohne den Quelltext zu verändern mit einer Definition:

- `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



[2] F Putze, P Sanders, J Singler. **The Multi-Core Standard Template Library (Extended Poster Abstract)**. PPoPP 2007

STXXL

Standard Template Library for Extra Large Data Sets

STXXL

Standard Template Library for Extra Large Data Sets

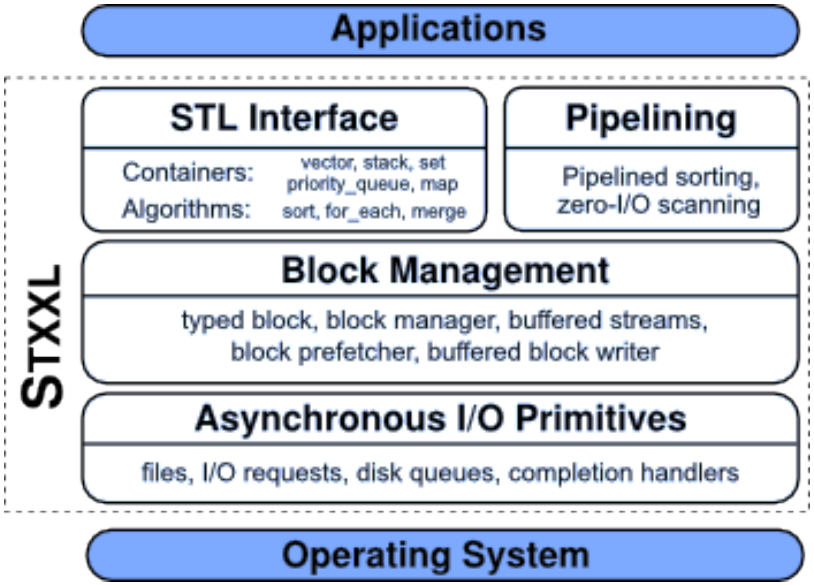
- Externspeicher-Datenstrukturen und Algorithmen
- STL-Interface für einfache Integration
- Transparentes Arbeiten mit Datenstrukturen, die nur noch auf der Festplatte Platz haben
- Unterstützt mehrere Festplatten parallel
- Benutzt optional die MCSTL

Container:

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

Algorithmen:

- `sort, for_each, merge`



Verfügbarkeit

Platformen:

- Linux, Mac OS X, FreeBSD (g++, clang++, icpc)
- Windows (VS 10 (requires Boost), VS 11, VS 12)

Lizenz

- Boost Software License 1.0

[4] **STXXL: Standard Template Library for Extra Large Data Sets**. http://stxxl.sourceforge.net/

Benutzung

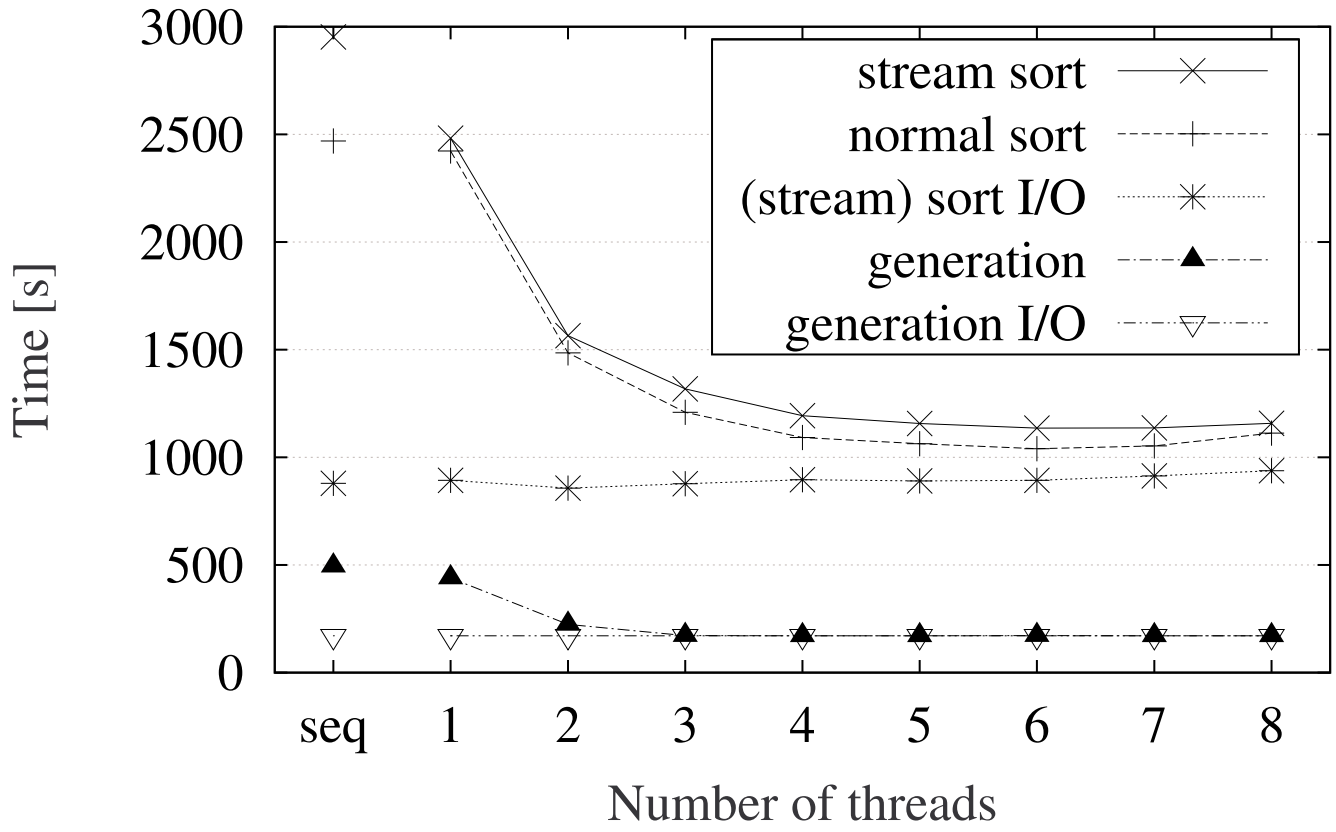
STXXL definiert eigenen Namensraum `stxxl`:

– **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(10000000000ull);

    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



[5] A Beckmann, R Dementiev, J Singler. **Building A Parallel Pipelined External Memory Algorithm Library** IPDPS 2009

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

Platformen:

- Linux, Mac OS X, FreeBSD (g++, clang++)
- Windows (VS 10-12)

Lizenz

- BSD License (3-clause)

[6] SeqAn: C++ Library for Sequence Analysis. http://www.seqan.de/

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,** FM-Index
- 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 <seqan/seq_io.h>

using namespace seqan;

int main() {
    SeqFileIn fin("input.fa");
    SeqFileOut fout("output.fa");
    Dna5String seq;
    CharString meta;
    while (!atEnd(SeqFileIn)) {
        readRecord(meta, seq, fin);
        reverseComplement(seq);
        writeRecord(fout, header, seq);
    }
    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, Upcase-String, …

Datenparallele Container & Algorithmen

- Journaled Strings, Journaled Srting Set, Journaled String Tree

Adaptionen bestehender Bibliotheken

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

boost

C++ LIBRARIES

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 (Enthalten im C++11 Standard)

Verfügbarkeit

Platformen:

- Sehr viele
- Inzwischen sind einige Bibliotheken in den TR1 (kommender C++ Standard) aufgenommen => Jetzt C++11/14

Lizenz

- Boost License

[7] Boost C++ Libraries. http://www.boost.org/

Boost Beispiel

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

Boost

```
#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;
}
```

C++11

```
#include <string>
#include <fstream>

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

    for (auto c : hello)
    {
          std::cout << c;
    }
    return 0;
}
```

Boost Beispiel (II)

- Lamda erspart Definitionen von Funktoren
  - Anwendbar z.B. beim Aufbau des Suffixarrays (P-Aufgabe 3)

```
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

```
std::vector<int> v(100);
// ...
std::sort(v.begin(), v.end(), [](int val1, int val2)
{
   return val1 > val2;
});
```

Boost Beispiel (III)

- Lamda erspart Definitionen von Funktoren

```
std::vector<int> v(100);
string text = "This is a large string";

std::sort(v.begin(), v.end(), [&](int val1, int val2)
{
   return text.substr(val1) > text.substr(val2);
});
```

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

Siegerehrung

Aufgabe 5: **Read Mapping**

1,90 s	1,90 s	3,11 s
30,77 s	20,50 s	41,61 s

ENDE

Danke für Eure Aufmerksamkeit!