8. Software Libraries

AlDaBi Praktikum

David Weese
WS 2010/11

Inhalt

- Bemerkungen zu den P-Aufgaben

- MCSTL
- STXXL
- SeqAn
- Boost

- Outro

BEMERKUNGEN ZU DEN P-AUFGABEN

Bemerkungen zu Aufgabe 7

- Eine Sequenz rückwärts iterieren:
  - So nicht:

```
string::iterator iter = sequence.end();
for (; iter >= sequence.begin(); iter--)
{
    // ...
}
```

  - Besser:

```
string::iterator iter = sequence.end();
for (; iter != sequence.begin();)
{
    --iter;
    // ...
}
```

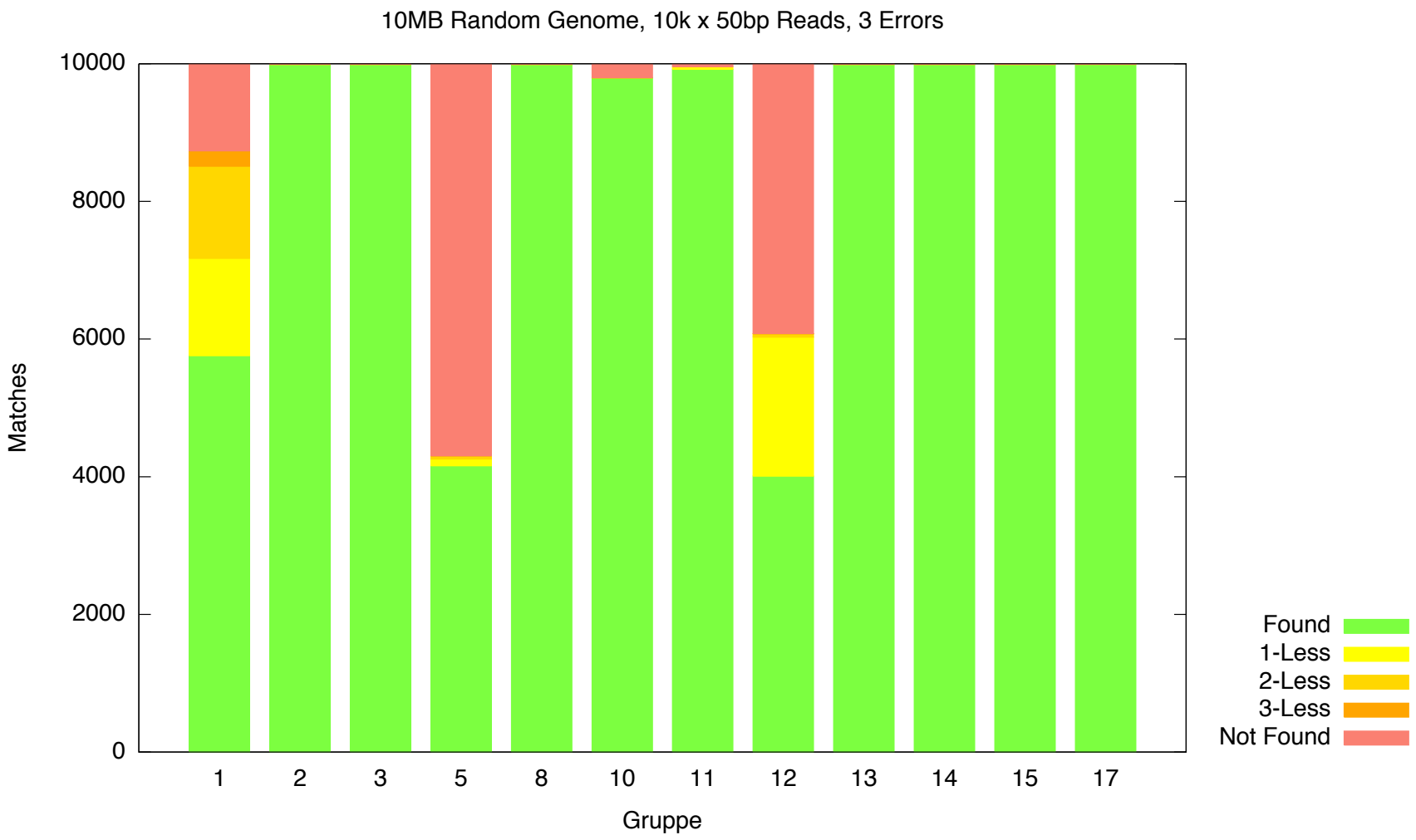
- Kommentarzeilen beachten:

```
...
>Sequenz 2
^@AAAGGCATTCTGTCCCGGCGCGCGGATTGCTGGGGGGGTGCGCCTTCCTGCCGCCTGG
CCACCTCCCGAGCTTCCTCTCCTATGAGGGGCCCGGCACTGGCCTGGGAGGTCCACTCAA
ACCACCAAGCGAGGAAAGTCAGGAGG eliezratnemmoK;B eliezratnemmoK;
```

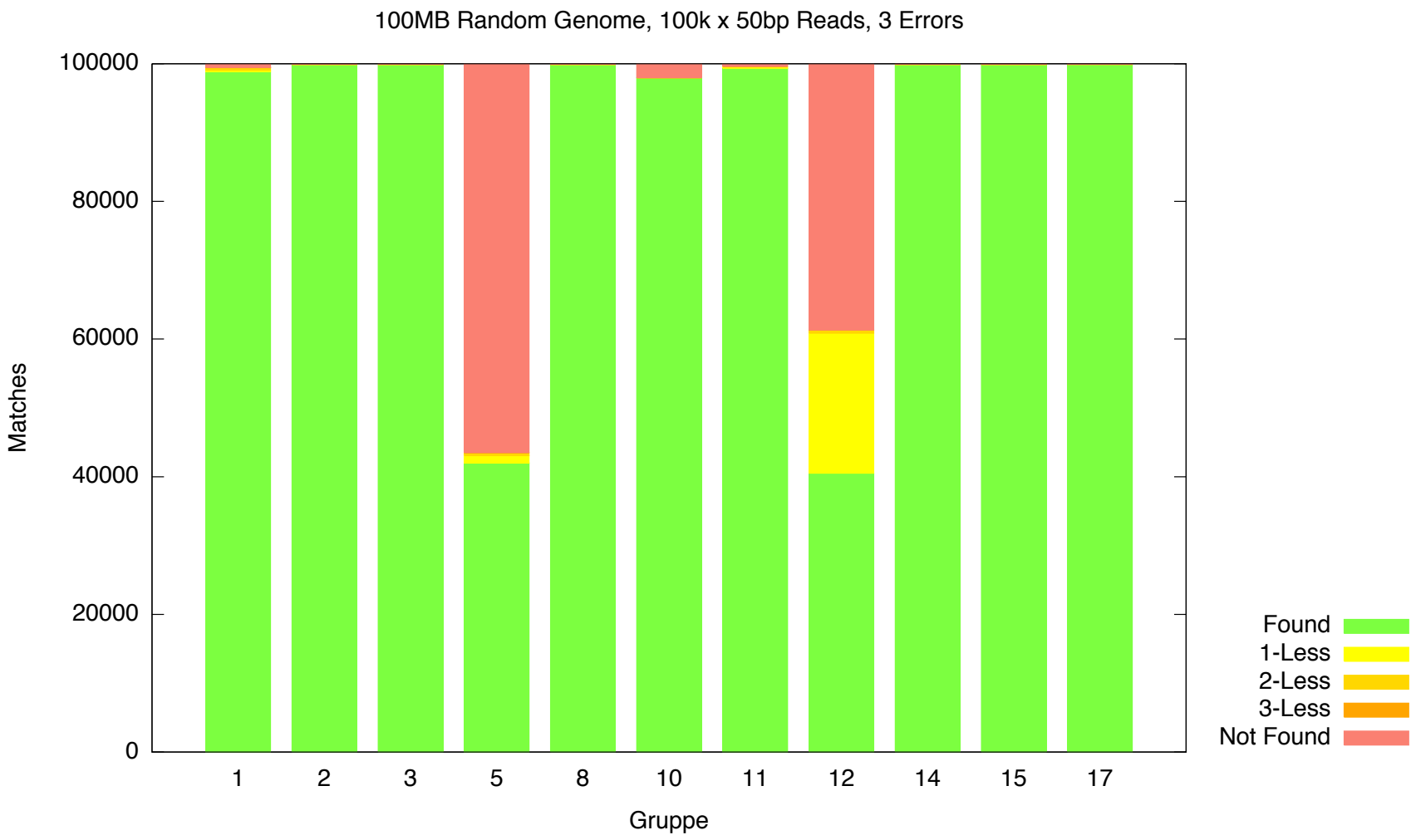
Bemerkungen zu Aufgabe 6

- C-optimale Schnittpositionen bestimmen
  - Es sollte ein Branch-And-Bound Mechanismus implementiert werden, keine volle Enumeration
  - Lässt man 0 oder $|S_i|$ als Cut-Positionen zu, sind die minimalen additional costs immer Null

Update Aufgabe 5 - Sensitivität



Update Aufgabe 5 – Sensitivität (II)



Update Aufgabe 5 - Ergebnisse

	Gruppe	Laufzeit (s)	Not-Found	Found	1-Less	2-Less	3-Less	4-Less
	1	122,53	1270	97160	549	959	62	0
1.	**2**	**43,58**	**96**	**99862**	**42**	**0**	**0**	**0**
	3	188,00	96	99862	42	0	0	0
	5	266,93	56493	41995	1075	437	0	0
3.	**8**	**115,17**	**96**	**99862**	**42**	**0**	**0**	**0**
	10	503,11	2036	97964	0	0	0	0
2.	**11**	**90,24**	**375**	**99323**	**302**	**0**	**0**	**0**
	12	166,55	38683	40520	20325	469	1	0
	14	538,18	96	99862	42	0	0	0
	15	514,86	96	99862	42	0	0	0
	17	551,83	82	99872	46	0	0	0

Punktestand

PA1	PA2	PA3	PA4	PA5	PA6	PA7	Summe
2	3	2	3	4	2	3	19
2	2	3	3	6	2	3	21
3	3	2	3	6	2	3	22
2	2	3	2	2	2	3	16
3	2	3	2	4	2	2	18
2	3	2	2	3	3	3	18
3	2	3	1	4	3	2	18
3	2	2	3	6	2	2	20
3	3	2	3	4	3	3	21
3	3	2	2	6	3	3	22
3	2	3	2	4	2	3	19
3	3	3	2	5	2	3	21
3	3	3	3	6	3	3	24
3	3	3	3	6	3	3	24
3	2	3					8
1	1	3	3	6	2	2	18

Gruppe	Prozent
1	**79**
2	**87**
3	**91**
4	**66**
5	**75**
6	**75**
7	**75**
8	**83**
10	**87**
11	**91**
12	**79**
13	**87**
14	**100**
15	**100**
16	**33**
17	**75**

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;
}
```

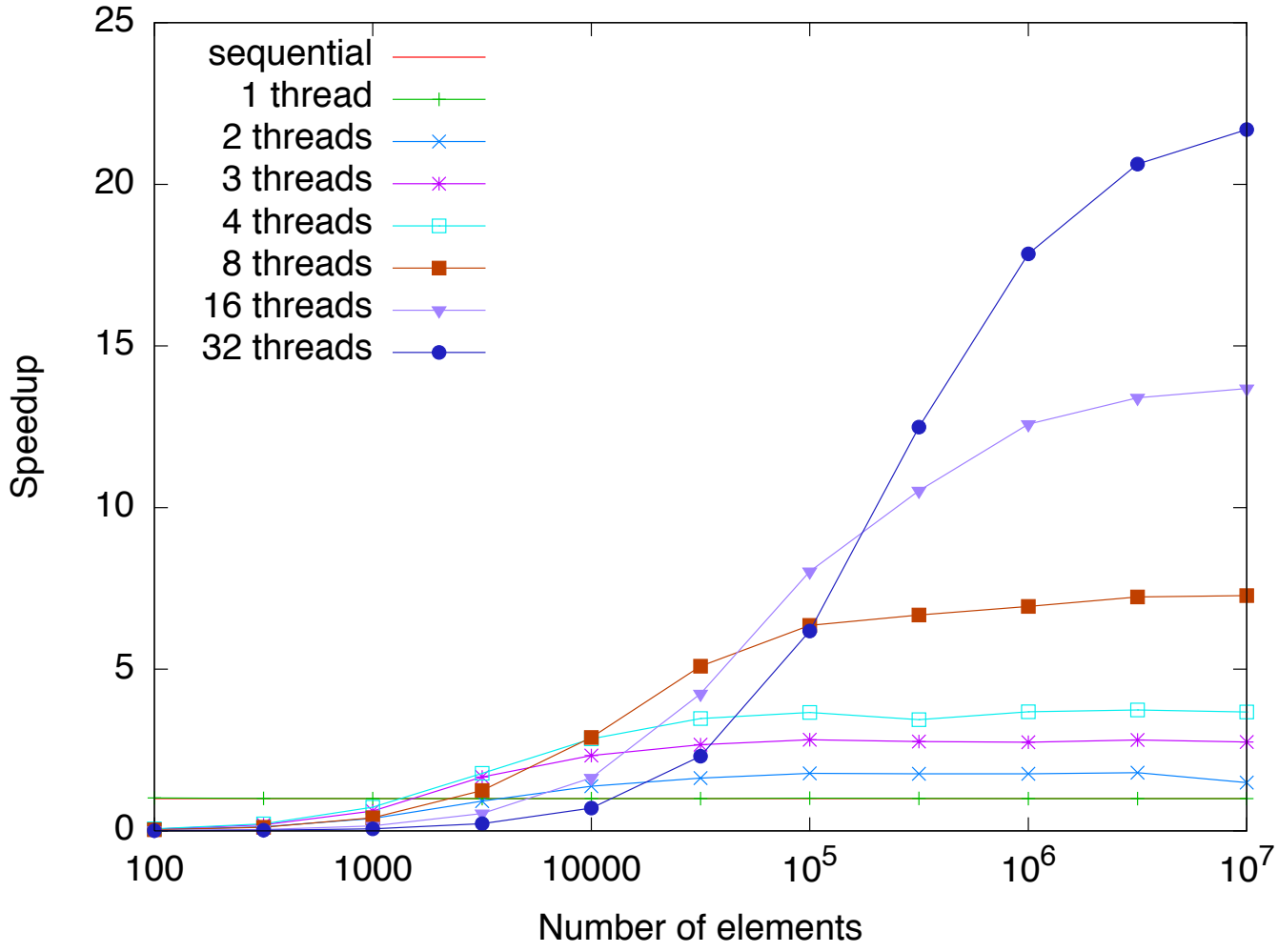
Benutzung (implizit)

- 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(), mcstl::sequential_tag()); // sequentiell
    return 0;
}
```

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

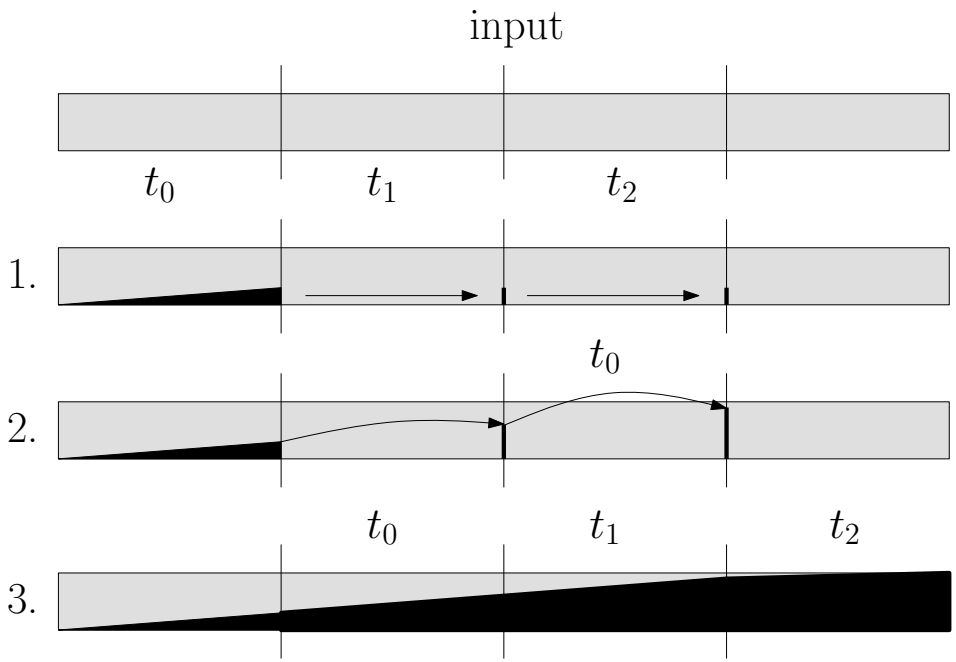
Paralleles Sortieren - Speedup



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

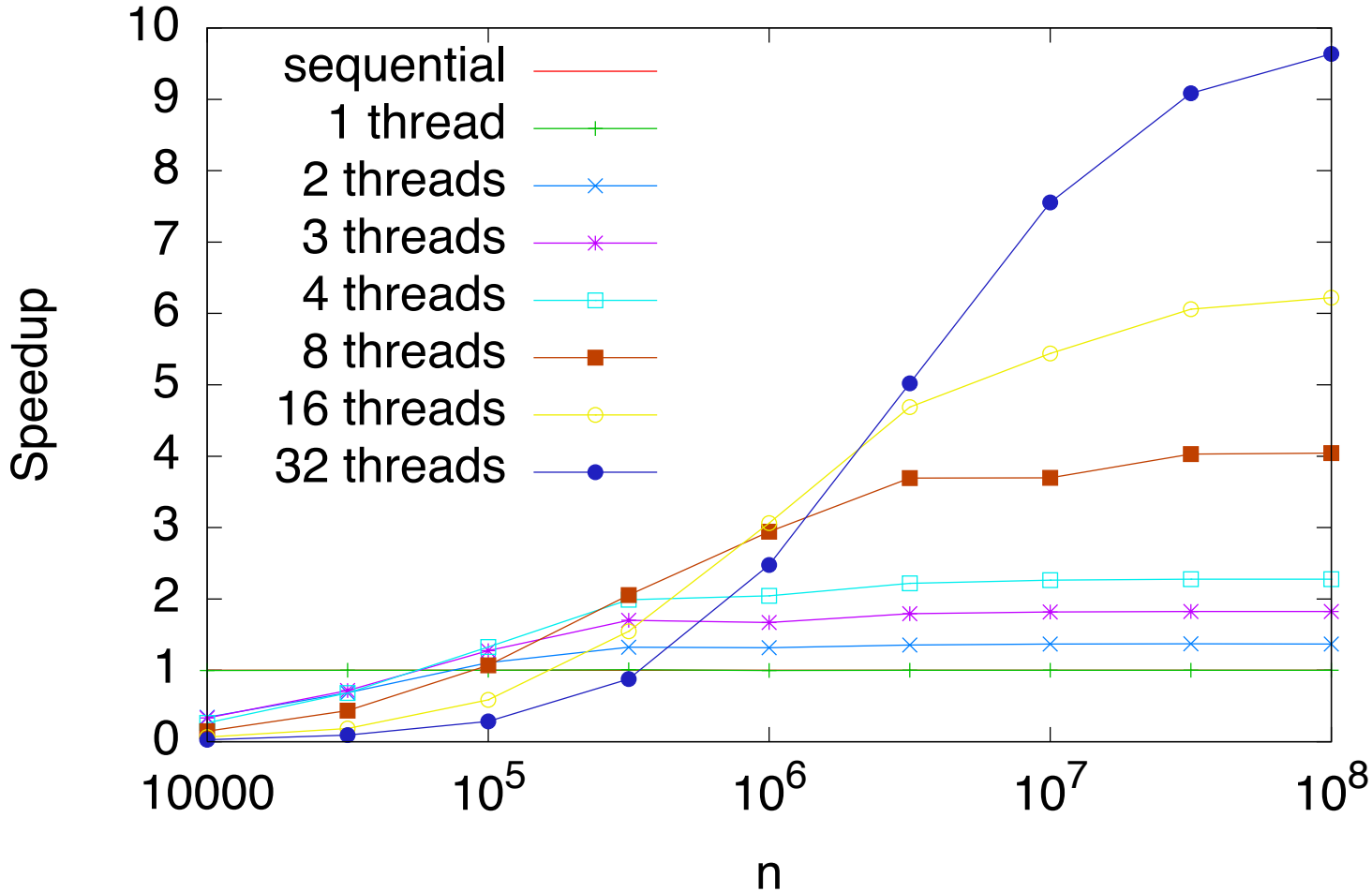
Kummulative Summe (parallel)

- Zerlege Feld f in p+1 gleiche Teile $f_0, \ldots, f_p$ (p = Anzahl Threads)
  1. Berechne parallel kumm. Summe für f_0 und die Summen $S_1, \ldots, S_{p-1}$ aller Elemente in $f_1, \ldots, f_{p-1}$
  2. Berechne $T_i := S_0 + \ldots + S_i$
  3. Berechne parallel die kumm. Summen für $f_1, \ldots, f_p$ beginnend mit $T_0, \ldots, T_{p-1}$



[3] P Sanders,J Singler. **VL Parallele Algorithmen**, KIT, 2008

Kummulative Summe - Speedup



[3] P Sanders, J Singler. **VL Parallele Algorithmen**, KIT, 2008

STXXL

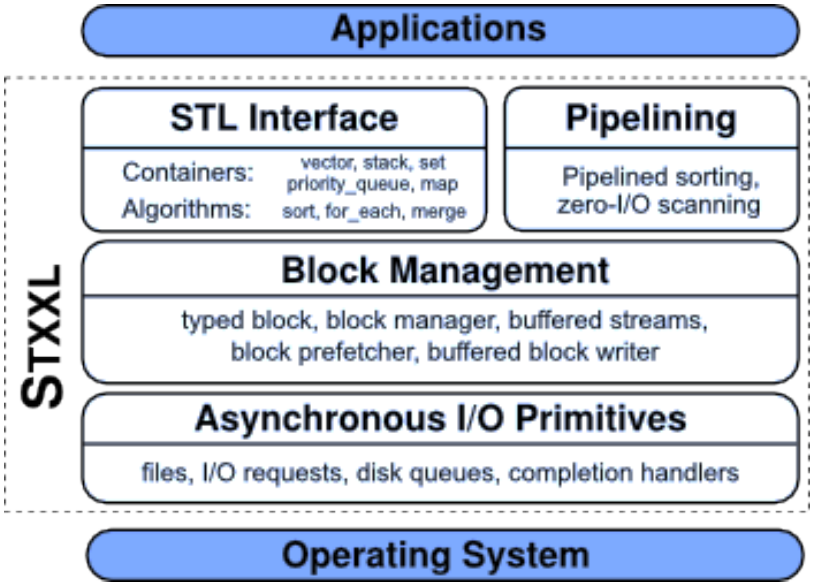
Standard Template Library for Extra Large Data Sets

STXXL

- Multi-Core Standard Template Library
  - 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 8-10)

- Lizenz
  - Boost Software License 1.0

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

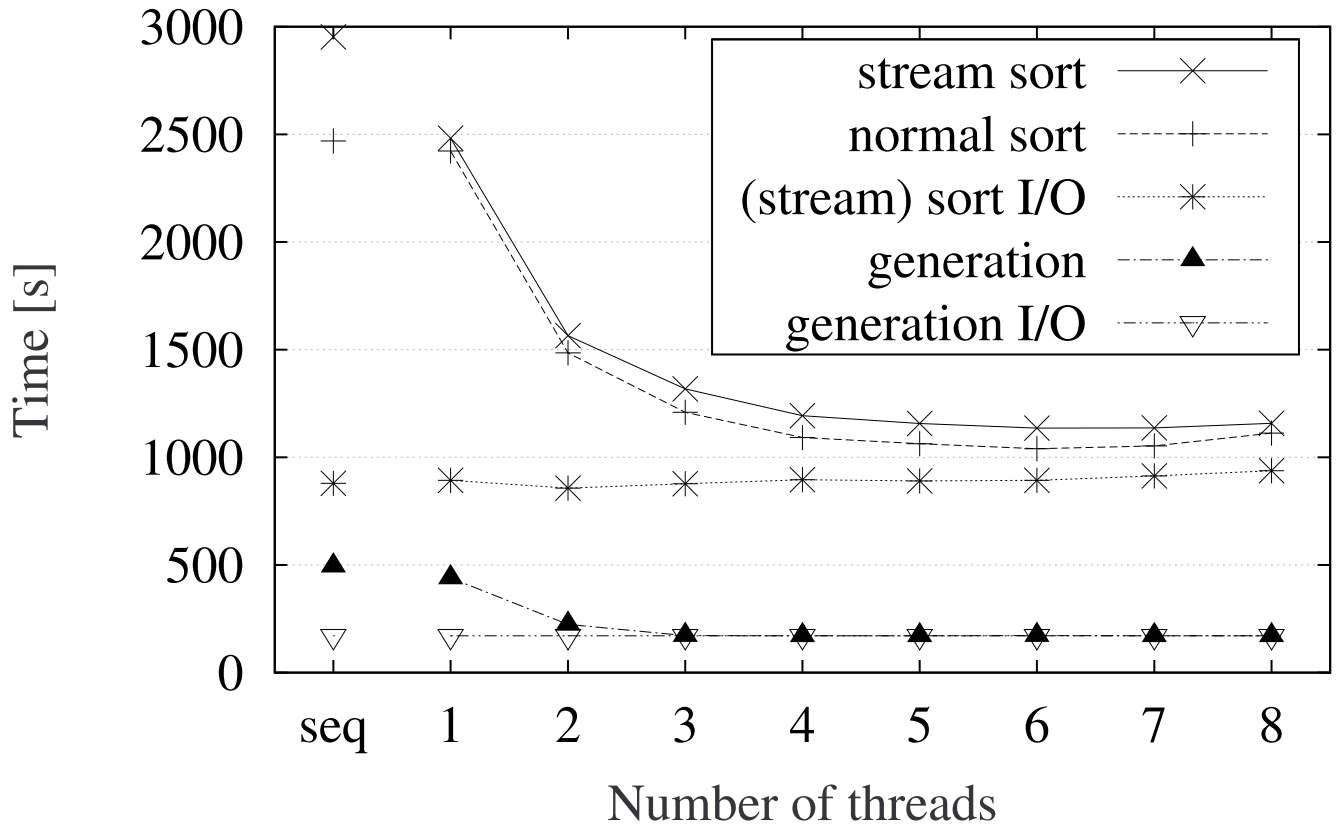
Benutzung

- 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 (P-VL 6)
  - 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 8-10, mingw)

- 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-A1)

```
#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 (P-A3)

```
#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>(0,-1,-1,-1), NeedlemanWunsch());
      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 P-A5)

```
#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 (P-A7)

```
#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, Upcase-String, …
- 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
  - **Iterval** – 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

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

- Lizenz
  - Boost License

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

OUTRO

Ausblick

- Wir bieten regelmäßig weitere Programmierkurse an
  - Softwarepraktika in den Sommersemestern:
    - **SeqAn** – Implementierung von Algorithmen zur Sequenzanalyse
    - **OpenMS** - Implementierung von Algorithmen zur Analyse massenspektrometrischer Daten
  - C++ Kurse:
    - **Programmierkurs C++** in den Wintersemestern
    - **C++ für Fortgeschrittene** in den Sommersemestern

- Betreute Bachelor/Masterarbeiten
  - http://www.inf.fu-berlin.de/en/groups/abi/theses
  - Weitere Themen in Absprache mit Prof. Reinert

- Bitte evaluiert diese und andere Veranstaltungen unter:
  - http://lehrevaluation.fu-berlin.de

ENDE

Danke für Eure Aufmerksamkeit!