Intel Threading Building Blocks (TBB)

Julius Adorf

26.10.2009
Seminar: Semantics of C++
TU München

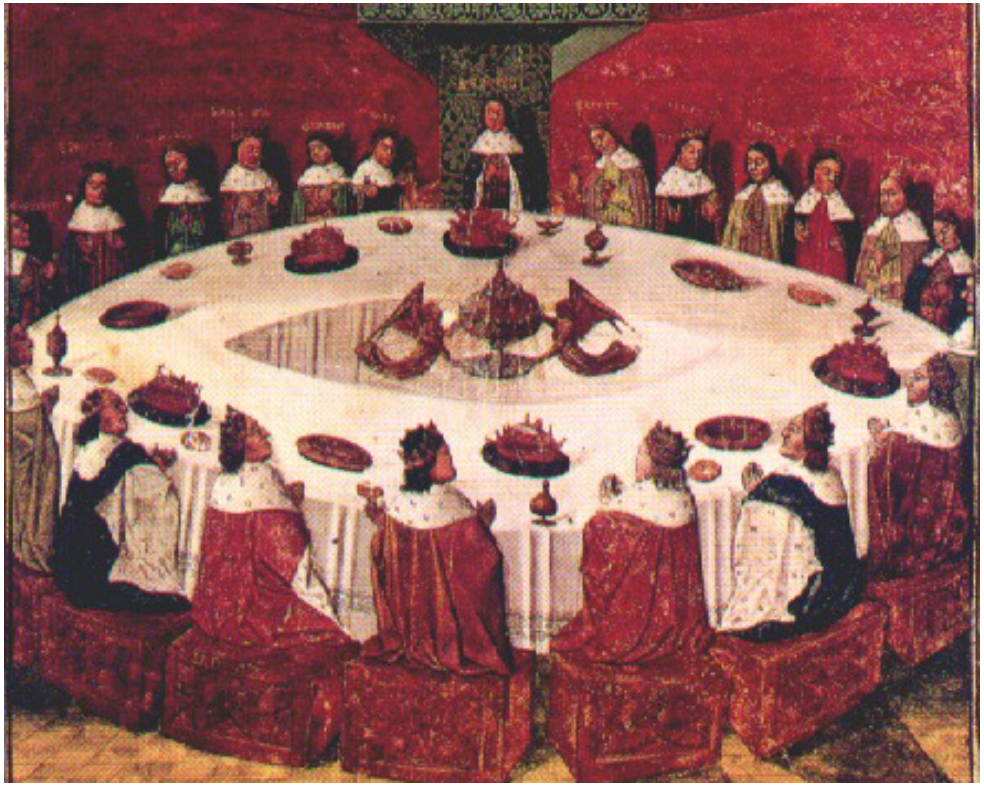
Tejas und Jayhawk?

Intel Threading Building Blocks (TBB)

- Parallelisierung für C++
- eine Bibliothek
- mittlerweile Open Source

Bildnachweis: tbb01

Shared-Memory



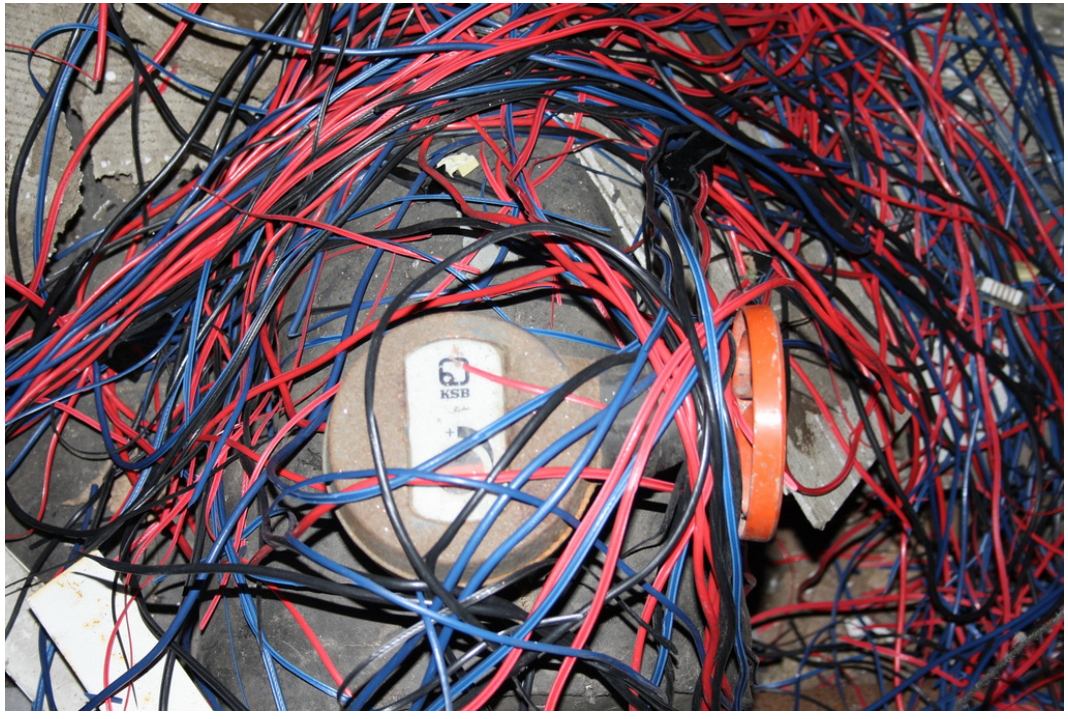
- TBB sind für Shared-Memory-Architektur ausgelegt

Überblick

- Konzeptuelles
- Schleifen parallel
- Quicksort parallel
- Mergesort parallel

Abstraktionsebene

Verwirklichung der Parallelisierung ...

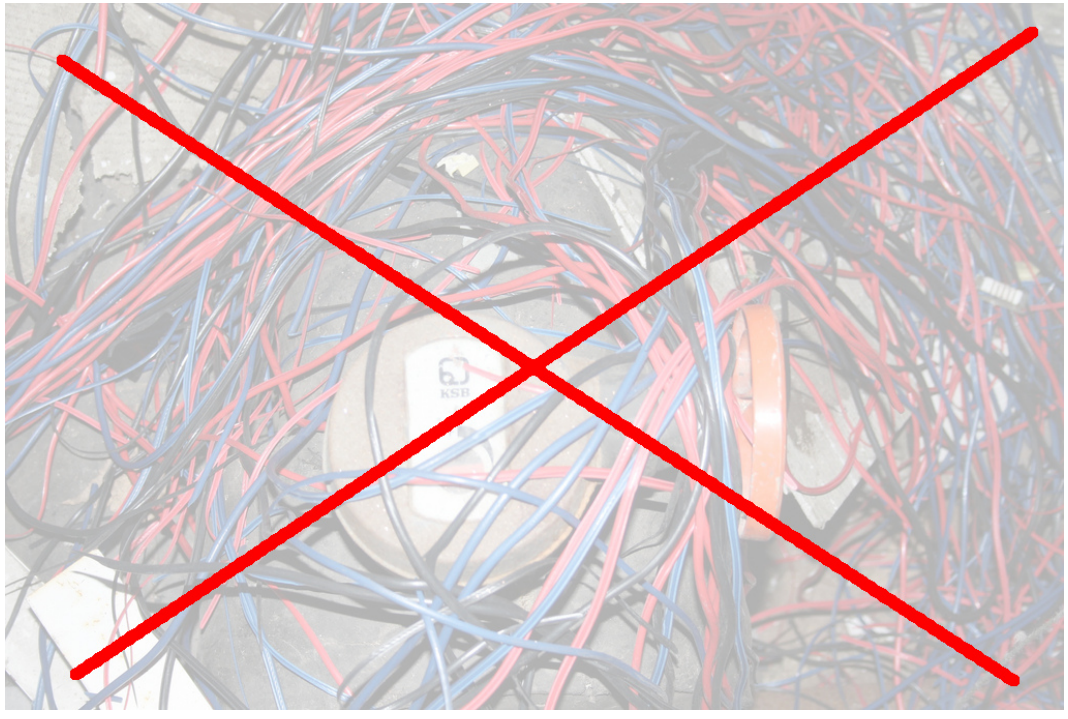


- mittels Threads

Bildnachweis: srt01

Abstraktionsebene

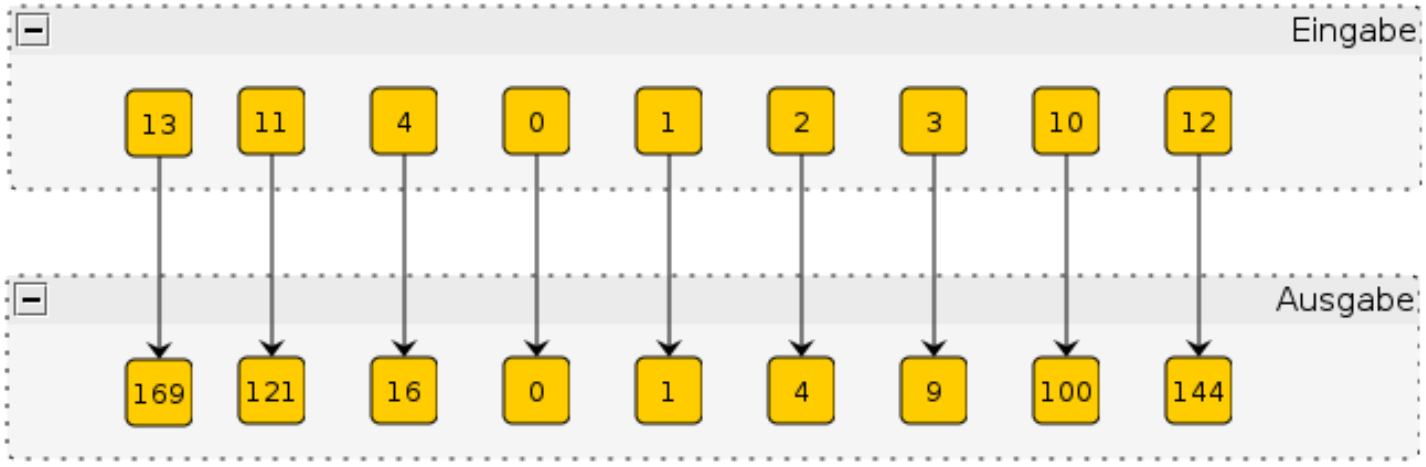
Parallelisierung aus Anwendersicht ...



- oberhalb der Thread-Ebene
- unterhalb der Compiler-Ebene

Bildnachweis: srt02

Eine Problemstellung

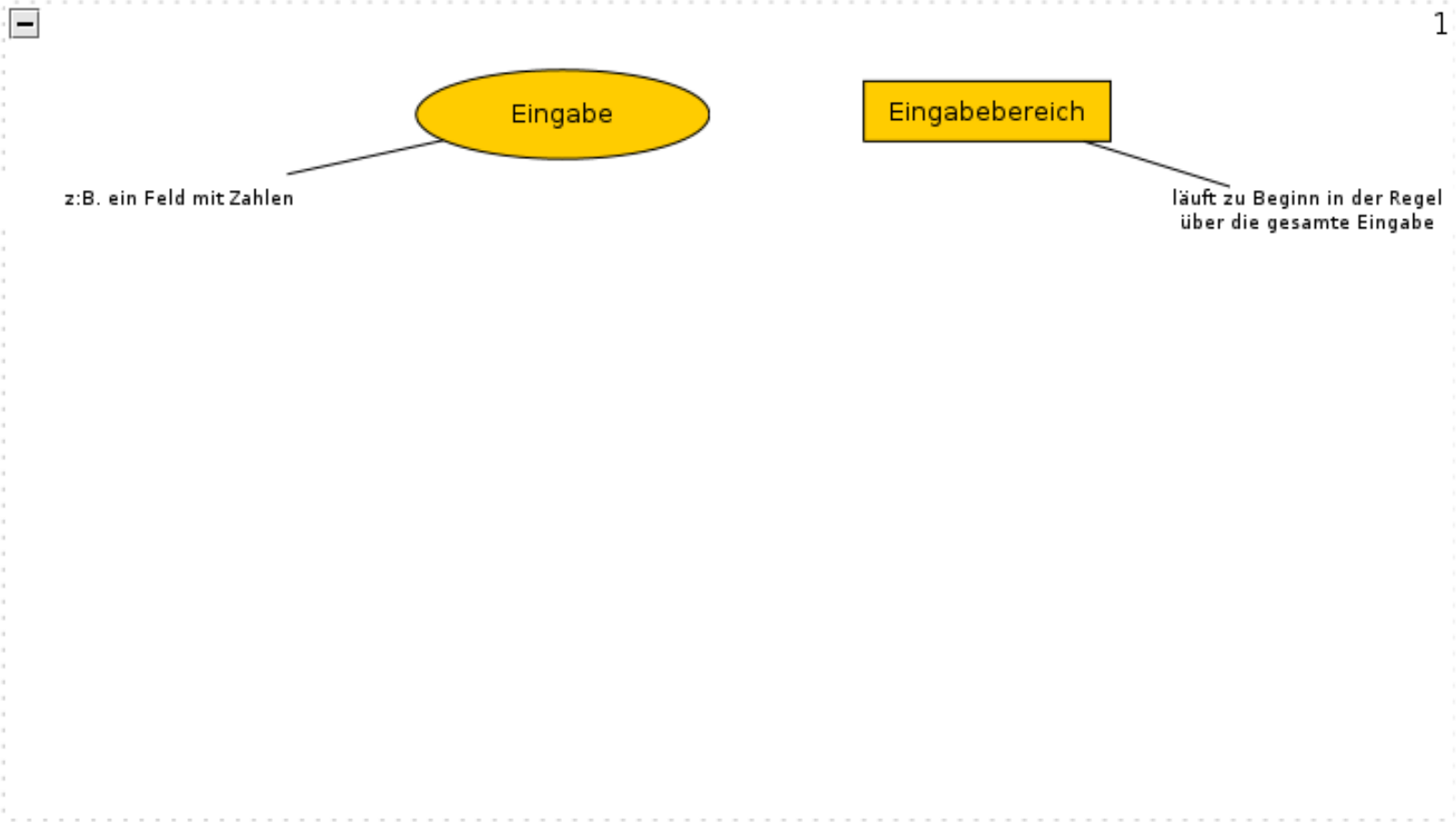


- Quadratzahlen berechnen
- einfache sequentielle Lösung
- parallele Lösung?

Hauptdarsteller

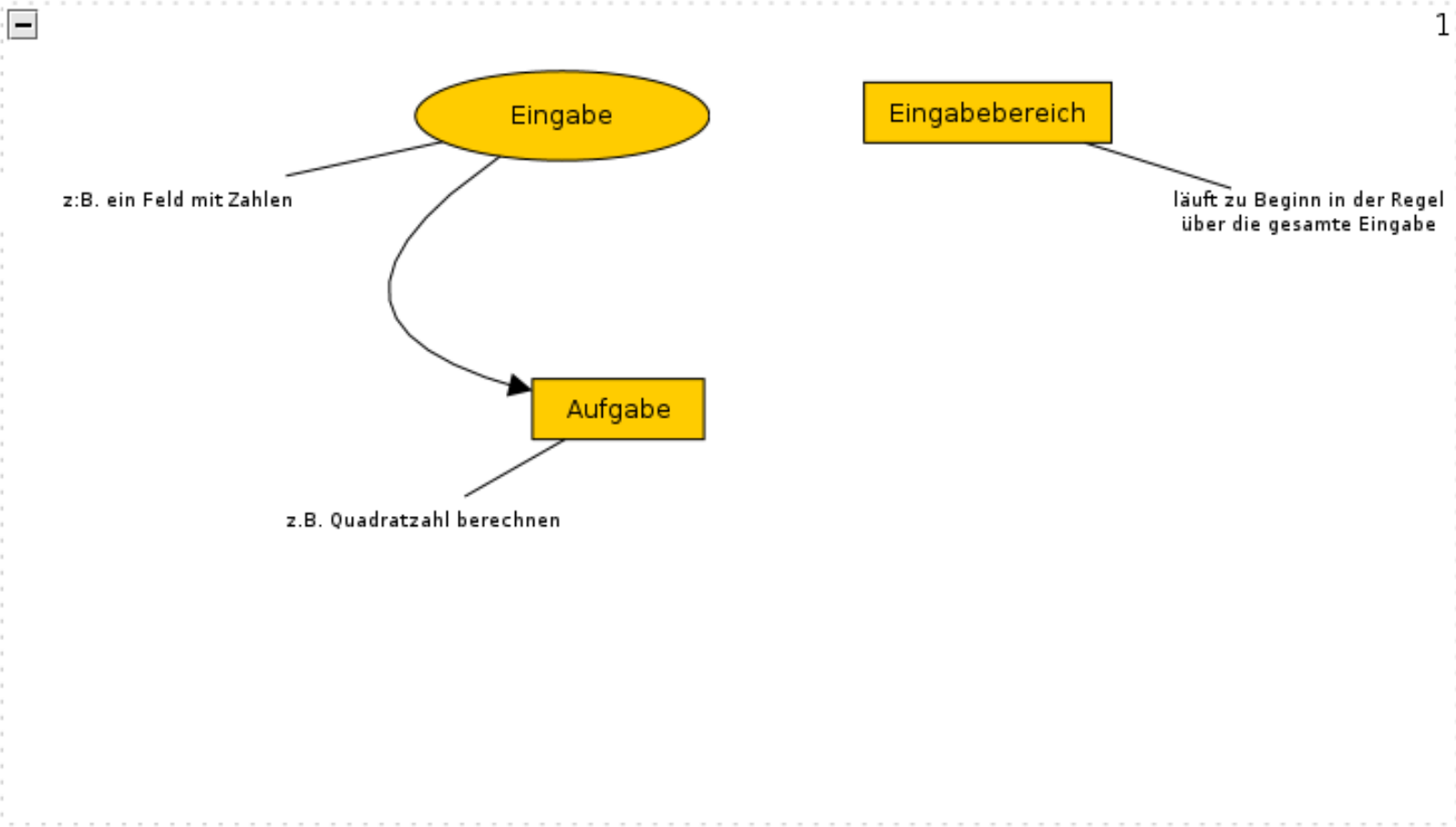
- Eingabebereich (= range)
- Aufgabe (= task)
- Algorithmus zur Parallelisierung (= algorithm)

Eingabebereich



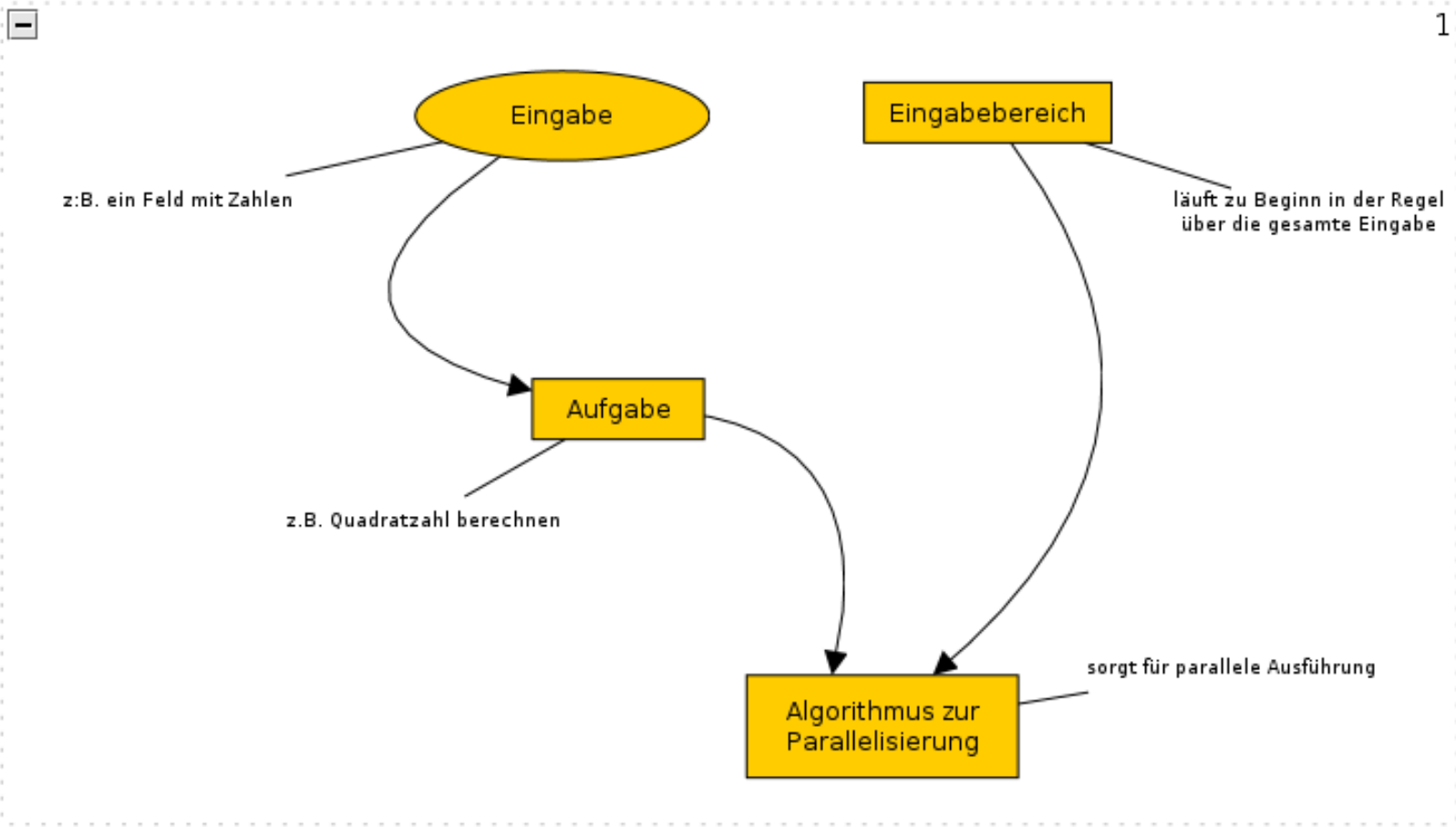
- z.B. *blocked_range* (ein Intervall)
- kann rekursiv geteilt werden
- eventuell selbst definieren

Aufgabe



- Aufgabe des Programmierers
- arbeitet sequentiell
- verarbeitet Eingabedaten in bestimmten Eingabebereich

Algorithmus

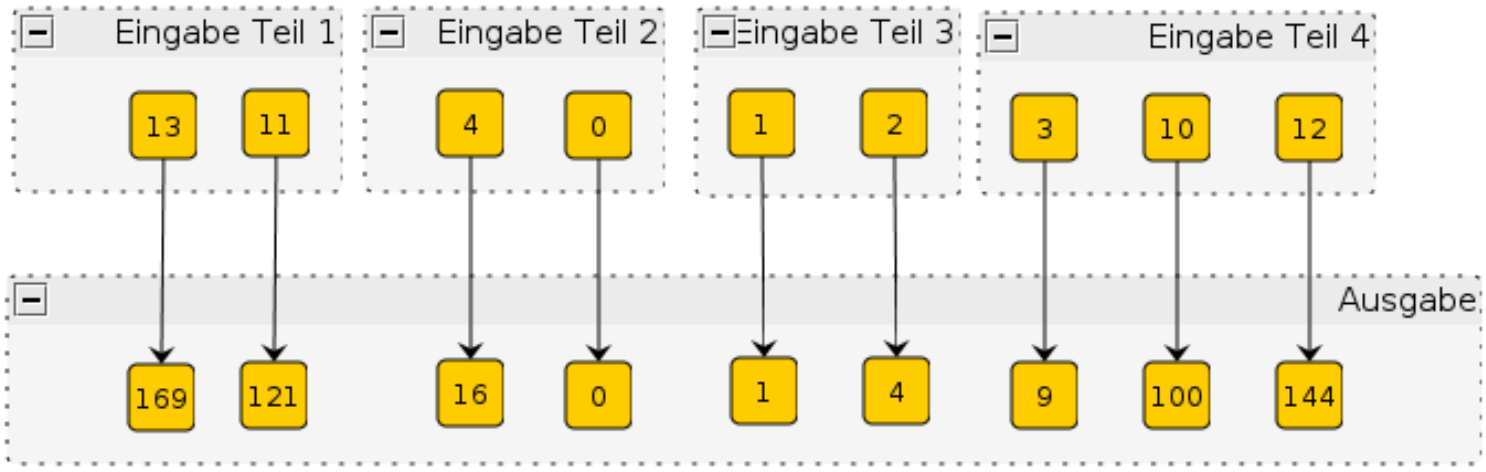


- sorgt für die parallele Ausführung
- nimmt Eingabedaten, Aufgabe, und Eingabebereich
- bereitgestellt von TBB

Überblick

- Konzeptuelles
- Schleifen parallel
- Quicksort parallel
- Mergesort parallel

Parallelisieren mit parallel_for - Lösungsansatz



- Datensatz partitionieren
- Lastverteilung durch *work stealing*

parallel_for - Vorbereitung

```
1 #include <iostream>
2 #include "tbb/task_scheduler_init.h"
3 #include "tbb/parallel_for.h"
4 #include "tbb/blocked_range.h"
5
6  using namespace std;
7  using namespace tbb;
```

parallel_for - Berechnung

```
 9  void print_square(int n) {
10    int n_squared = n * n;
11    printf("%3i squared is %3i\r\n", n, n_squared);
12  }
```

parallel_for - Aufgabe

```
14  class CalculateSquares {
15
16   int* const input;
17
18   public:
19   void operator()(const blocked_range<size_t>& r) const {
20     for (size_t i = r.begin(); i != r.end(); i++) {
21       print_square(input[i]);
22     }
23   }
24
25   CalculateSquares(int input[]) : input(input) { }
26  };
```

parallel_for - Aufruf

```
27
28  int main() {
29      task_scheduler_init init; // prior to v2.1
30      const int n = 10;
31      int input[n] = { 0, 1, 2, 3, 4, 10, 11, 12, 13, 21 };
32      parallel_for(
33              blocked_range<size_t>(0, n),
34              CalculateSquares(input));
35  }
```

TBB mit Lambda-Funktionen

```
14  // ... define print_square, include headers ...
15  int main() {
16    task_scheduler_init init(); // prior to v2.1
17    const int n = 10;
18    int input[n] = { 0, 1, 2, 3, 4, 10, 11, 12, 13, 21 };
19    parallel_for(
20      blocked_range<size_t>(0, n),
21      [=](const blocked_range<size_t>& r) {
22        for(size_t i = r.begin(); i != r.end(); i++)
23          print_square(input[i]);
24      });
25  }
```

- kommen 2010 mit C++0x-Standard
- Lambda-Funktionen unterstützt ab GCC 4.5

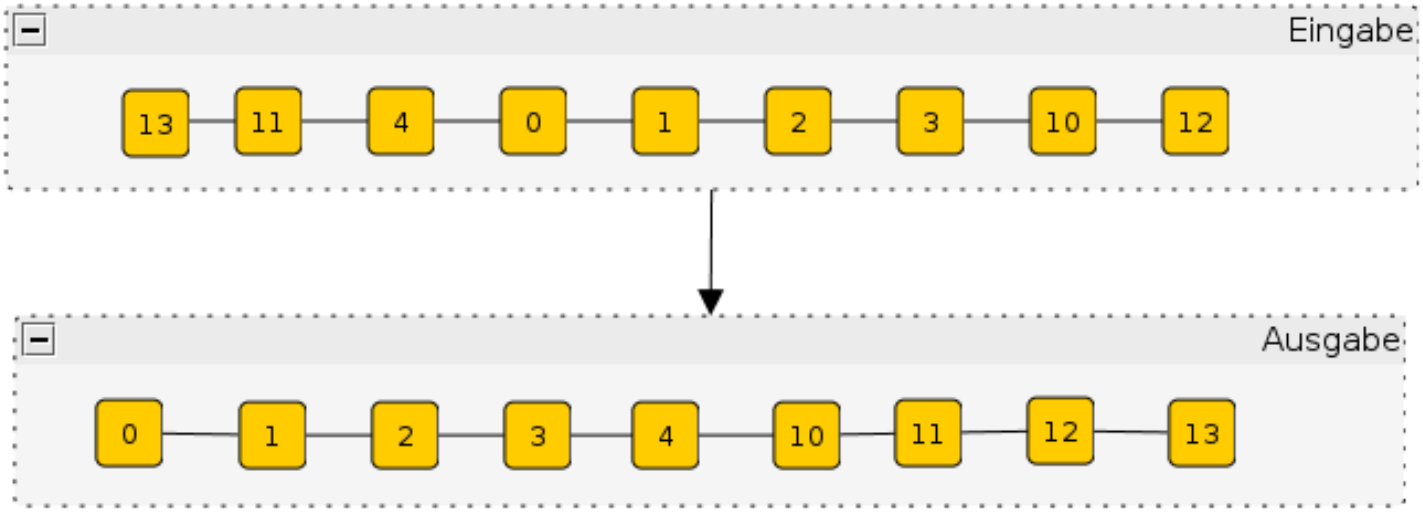
Algorithmen

- parallel_for - *Map* parallel, Rekursion, Schleifenparallelität
- parallel_reduce - *Reduce* parallel
- parallel_scan - Präfix-Scan parallel
- parallel_do - *Map* auf sequentiellem Iterator
- pipelining - Fließband parallel

Überblick

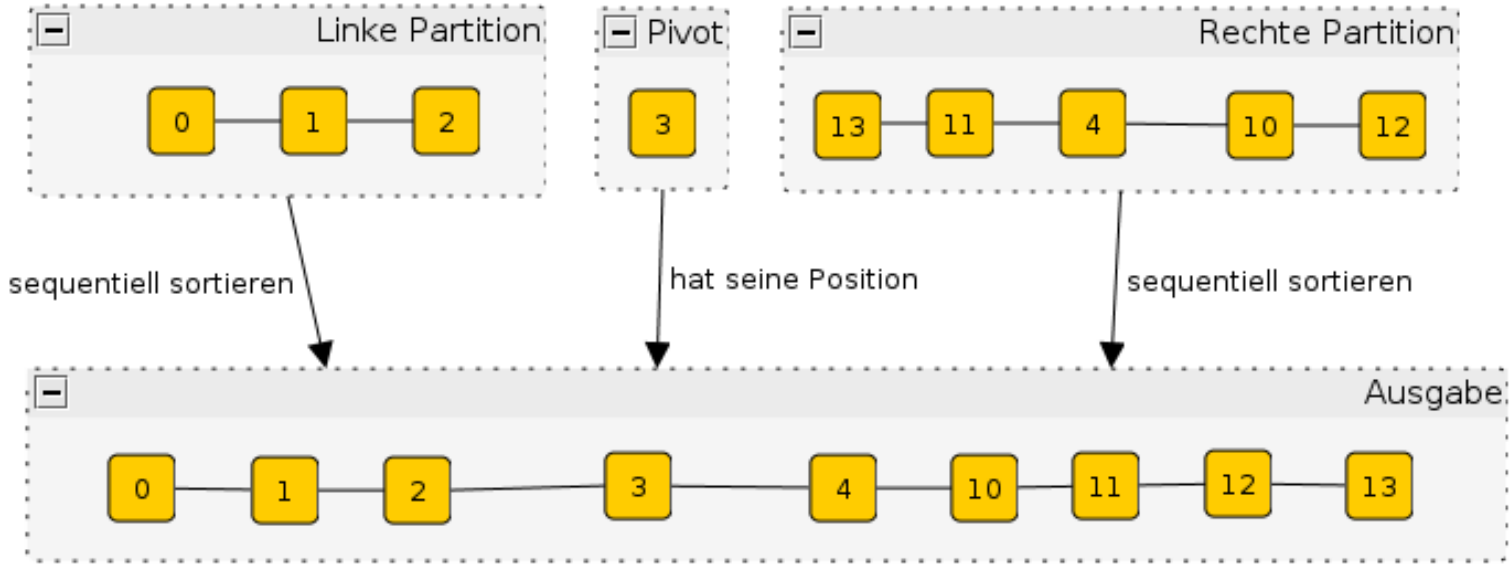
- Konzeptuelles
- Schleifen parallel
- Quicksort parallel
- Mergesort parallel

Quicksort parallelisieren



- Quicksort mit parallel_for
- left, right = partition(array)
- quicksort(left); quicksort(right)

Quicksort parallelisieren



- Arbeit beim rekursiven Abstieg
- Rekursiv partitionieren - eigene Range-Klasse einführen
- Teilbereiche sequentiell sortieren - Aufgabe definieren

Quicksort - eigene Range-Klasse einführen

```
 7  template<typename RandomAccessIterator,
 8            typename Compare>
 9  struct QuicksortRange {
10
11      RandomAccessIterator begin;
12      size_t size;
13      const Compare &comp;
14
15      bool empty() const;
16      bool is_divisible() const;
17      QuicksortRange(QuicksortRange&, split);
18      QuicksortRange(RandomAccessIterator _begin,
19                     size_t _size,
20                     Compare &comp_);
21
22  };
```

Quicksort - der Split-Konstruktor

```
37  // pseudo-C++
38  QuicksortRange(QuicksortRange& orig, split) {
39    int p = partition(orig.begin, orig.begin+orig.size);
40    orig.size = p;
41    begin = orig.begin + p + 1;
42    size = orig.size - p - 1;
43    comp = orig.comp;
44  }
```

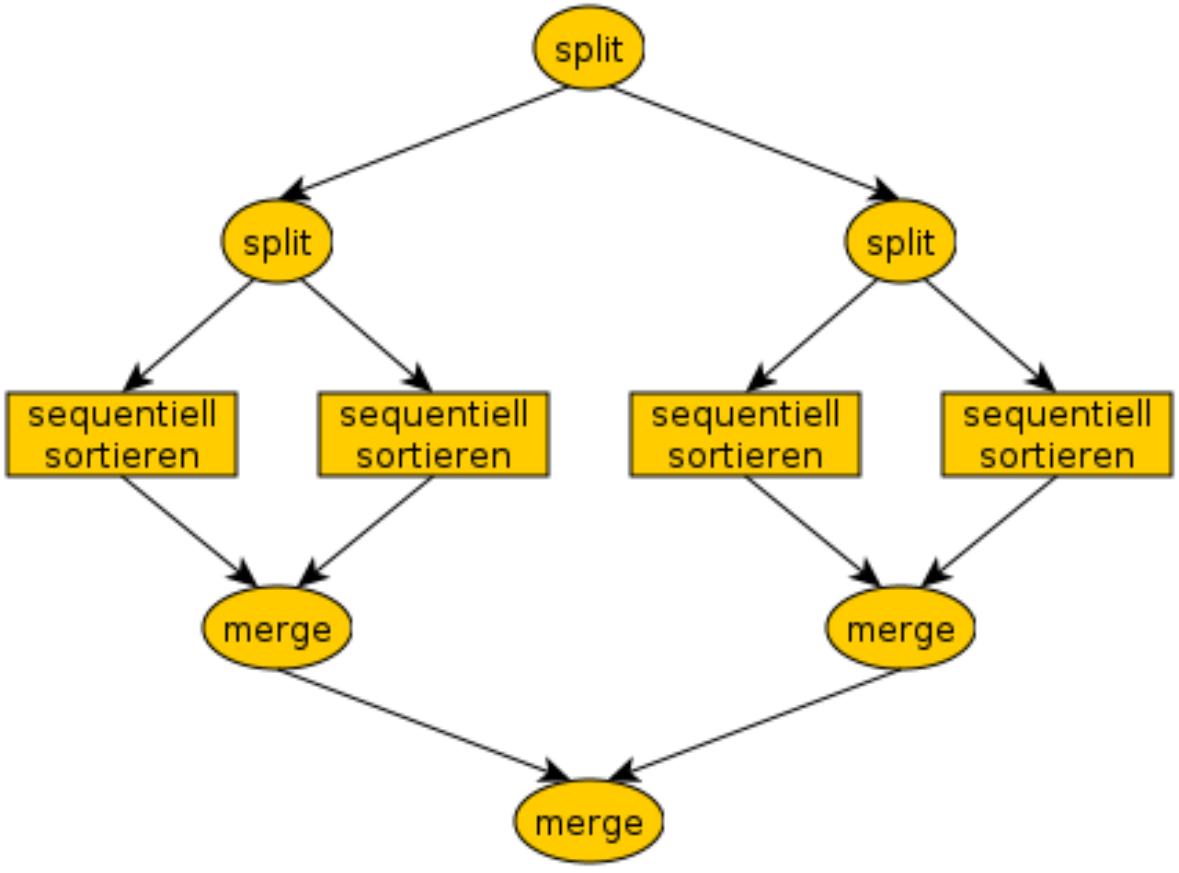
Quicksort - Aufgabe definieren

```
23
24  template<typename RandomAccessIterator,
25           typename Compare>
26  struct quicksort {
27
28    void operator()(
29        const QuicksortRange<RandomAccessIterator,
30                             Compare>& r) const {
31      sort(r.begin, r.begin + r.size, r.comp);
32    }
33
34  };
```

Überblick

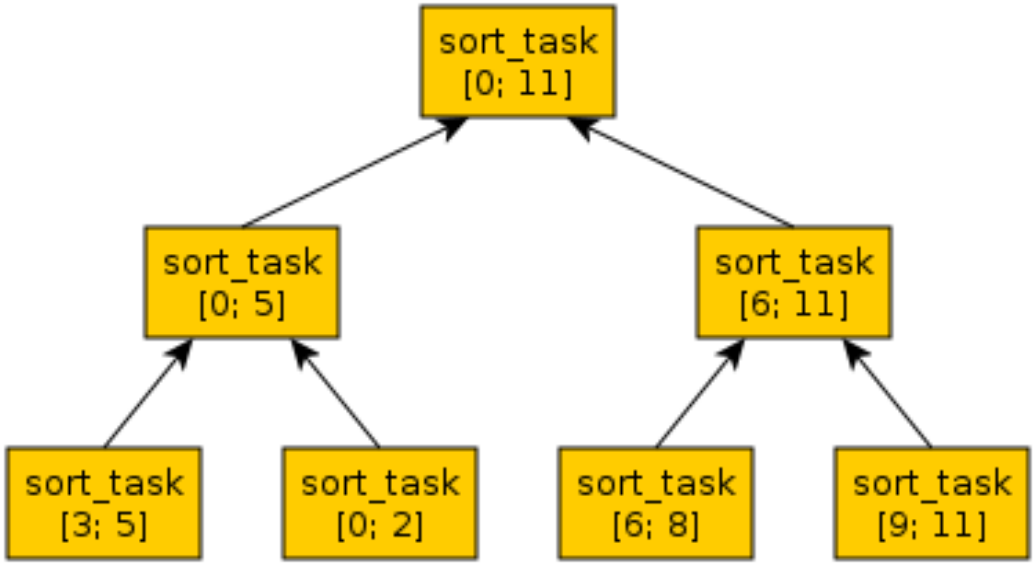
- Konzeptuelles
- Schleifen parallel
- Quicksort parallel
- Mergesort parallel

Mergesort parallel



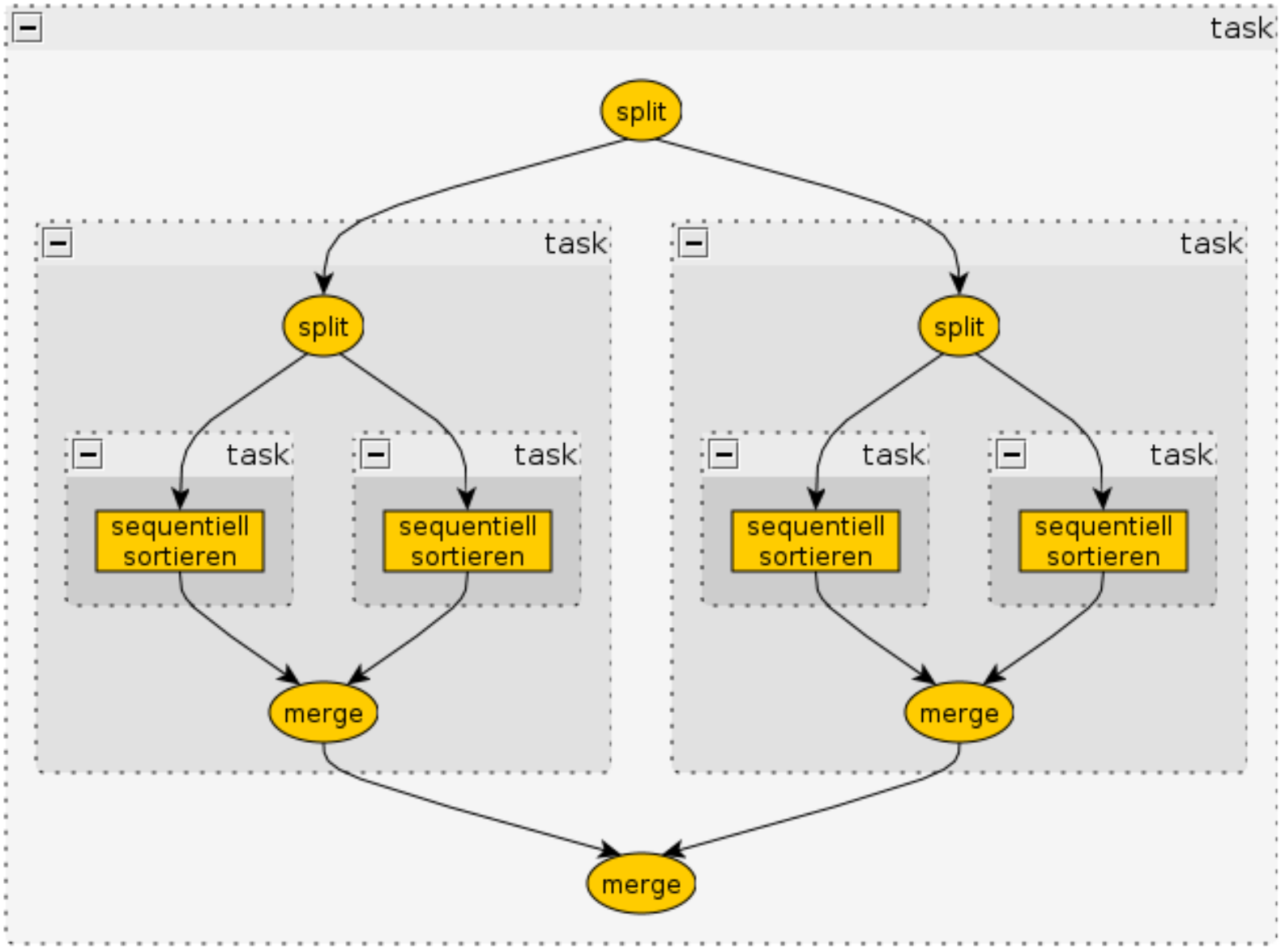
- Typisch Divide-and-Conquer
- Arbeit beim rekursiven Aufstieg
- Geht weder mit parallel_for noch mit parallel_reduce

Mergesort direkt mit Tasks



- Task sind kleinste Arbeitspakete
- Anordnung der Tasks in einem Baum
- Abarbeitung mit Tiefensuche

Mergesort mit Tasks



```
 9  struct SortTask: public task {
10    int *begin, *end;
11
12    SortTask(int *_begin, int *_end) :
13        begin(_begin), end(_end) { /* nop */ }
14
15    task* execute() {
16      size_t size = end - begin;
17      if (size <= 32) {
18        std::sort(begin, end);
19      } else {
20        int *median = begin + size / 2;
21        SortTask& sort_left = *new(allocate_child())
22                                  SortTask(begin, median);
23        SortTask& sort_right = *new(allocate_child())
24                                  SortTask(median, end);
25        set_ref_count(3);
26        spawn(sort_left);
27        spawn_and_wait_for_all(sort_right);
28        inplace_merge(begin, median, end);
29      }
30      return NULL;
```

Mergesort parallel

```
33
34  void mergesort(int *begin, int* end) {
35    SortTask& sort = *new(task::allocate_root())
36                              SortTask(begin, end);
37    task::spawn_root_and_wait(sort);
38  }
```

Mergesort parallel - Demo

Bilanz

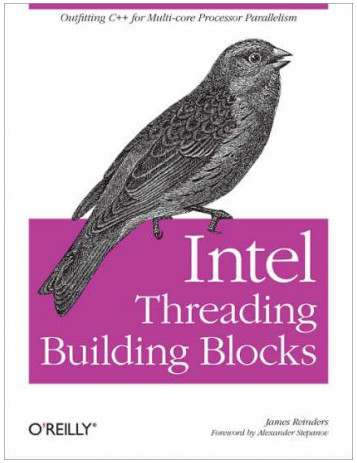
- Schleifen parallelisiert
- Quicksort parallelisiert
- Mergesort parallelisiert
- angekratzt: parallel_reduce, parallel_do, parallel_scan, pipeline

Fazit

- Verschachtelung von parallelen Abschnitten möglich
- Gut auch für nachträgliche Parallelisierung
- Programmierer behält Kontrolle
- Syntax gewöhnungsbedürftig
- Recht gute Dokumentation, viele Beispiele

Bildnachweis: tbb01

Quellenangaben - Intel Threading Building Blocks



- James Reinders: Intel Threading Building Block, O'Reilly, 2007
- Deilmann, Mario and Willhalm, Thomas: Intel Threading Building Blocks - ein templatebasiertes Programmiermodell für moderne Mehrkehrnarchitekturen, LinuxMagazin, Mai 2007
- Popovici, Nicolae und Willhalm, Thomas: Putting Intel Threading Building Blocks to Work, Proceedings of the 1st international workshop on Multicore software engineering, http://portal.acm.org/citation.cfm?id=1370085, 2008
- Helmut Erlenkötter: C++, Rowohlt, 2005
- http://software.intel.com/en-us/blogs/author/kevin-farnham/
- http://entwickler.de/zonen/portale/psecom,id,99,news,37123,p,0.html
- http://gcc.gnu.org/projects/cxx0x.html
- http://software.intel.com/en-us/blogs/2007/11/21/parallel_do-a-new-threading-building-blocks-component/
- http://software.intel.com/en-us/blogs/2009/08/03/parallel_for-is-easier-with-lambdas-intel-threading-building-blocks/
- http://software.intel.com/en-us/blogs/2009/08/03/hello-lambdas-c-0x-a-quick-guide-to-lambdas-in-c/
- http://software.intel.com/en-us/articles/intel-threading-building-blocks-openmp-or-native-threads/
- http://www.research.att.com/~bs/dne.html
- http://www.threadingbuildingblocks.org/

Quellenangaben - Grafiken

Alle Grafiken, die hier nicht in den Angaben auftauchen, sind selbst erstellt oder stehen zur freien Verfügung.

tbb01 Ausschnitt der Seite
http://www.threadingbuildingblocks.org/ (August 2009)

srt01 von Schrottie
http://www.flickr.com/photos/schrottie/3755566450/

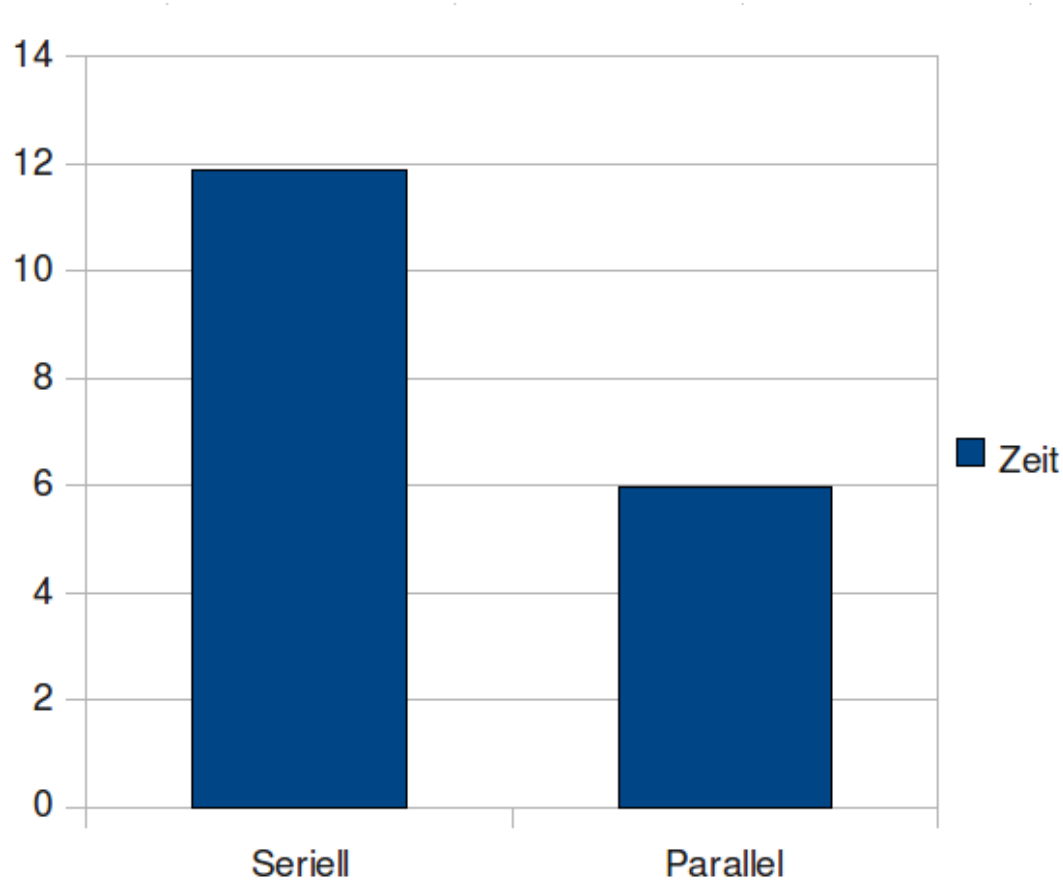
srt02 von Schrottie
http://www.flickr.com/photos/schrottie/3755566450/
(auf dieser Seite angepasst)

Zusätzliches Material

Benchmark

- Suche nach Substring
- Rechner mit Dual-Core-Prozessor

Benchmark



- Suche nach Substring
- Rechner mit Dual-Core-Prozessor