STXXL: Standard Template Library for XXL Data Sets

Roman Dementiev, Peter Sanders
Fakultät für Informatik, Universität Karlsruhe
[dementiev,sanders]@ira.uka.de

Lutz Kettner
Max Planck Institut für Informatik, Saarbrücken
kettner@mpi-sb.mpg.de
Large Data Sets are Growing

- Geographic Information Systems: detailed maps occupy many GBytes
- Texts: Google searches over 8 billion web pages
- Data warehouses: hundreds of terabytes
- Scientific computing: global climate simulations, aerodynamics, …
I/O Model

- Disk access involves a mechanical movement ⇒ slow
  - Random disk access: 1–20 ms
  - Random main memory access: 1–50 ns
  - Sequential reading/writing next elements is “for free”

- Aggarwal–Vitter I/O model
  - $N$ — size of input
  - $M$ — size of main memory
  - $B$ — size of transfer block
  - Cost measure – number of I/Os
Add disks to improve bandwidth:
- Parallel Disk Model (PDM) by Vitter–Shriver
  - $D$ independent disks
  - In an I/O step try transfer $D$ blocks between main memory and disks

```
CPU

M

DB

1 2 D
```

Add disks to improve bandwidth:
- Parallel Disk Model (PDM) by Vitter–Shriver
  - $D$ independent disks
  - In an I/O step try transfer $D$ blocks between main memory and disks

```
CPU

M

DB

1 2 D
```
What is **STXXL**?

- STL – C++ Standard Template Library, implements basic containers (maps, sets, priority queues, etc.) and algorithms (quicksort, mergesort, selection, etc.)
- **STXXL**: Standard Template Library for XXL Data Sets
  

  containers and algorithms that can process huge volumes of data that only fit on disks (I/O-efficient)
  
  - Compatible with **STL**
  - **Performance**–oriented
Related Work

- TPIE (1994–) Duke University, USA
  - Sorting, merging, matrix operations, geometric search data structures

  - Sorting, priority queues, search trees, suffix arrays

- FG (2002–) Dartmouth College, USA
  - Parallel design framework for clusters
  - Pipelining helps to mitigate disk and communication latency
STXXL Features

- **Parallel** disk support
- Handles very large problems (up to terabytes)
- **Pipelining** saves many I/Os
  - Feed output of EM alg. to input of another EM alg. directly
- **Explicitly** overlaps I/O and computation
- Avoids superfluous **copying**
  - in OS I/O subsystem and the library itself
- Compatible with **STL** – C++ Standard Template Library
  - Short development times
  - Reuse of STL code (e.g. selection alg.)
Applications

STL-user layer

Containers: vector, stack, set
Algorithms: priority_queue, map
sort, for_each, merge

Streaming layer

Pipelined sorting, zero-I/O scanning

Block management (BM) layer

typed block, block manager, buffered streams,
block prefetcher, buffered block writer

Asynchronous I/O primitives (AIO) layer

files, I/O requests, disk queues,
completion handlers

Operating System
- Hides details of **async. I/O** (portability)
- Implementations for POSIX/UNIX system
- Asynchrony provided by POSIX threads or Boost Threads
**STXXL**  Design: BM Layer

- Block abstraction
- Parallel disk model
- (randomized) striping and cycling
- parallel disk buffered writing and optimal prefetching

[HutchinsonSandersVitter01]
- **STXXL User Layers**

  - **STL-user layer**: compatible with STL
  - **Streaming layer**: programming with pipelining
STL-User Layer

- **EM containers:** `vector`, `priority_queue`, `stack`, `map`, `queue`
  - Can be **directly** used with algorithms from **STL**

- **EM algorithms:** `sort [DemSan03]`, `ksort`, `for_each`, ...

```cpp
1  stxxl::vector<edge> Edges(10000000000ULL);
2  std::generate(Edges.begin(), Edges.end(), random_edge);
3  stxxl::sort(Edges.begin(), Edges.end(), edge_cmp);
4  512*1024*1024);
5  stxxl::vector<edge>::iterator NewEnd =
6  std::unique(Edges.begin(), Edges.end());
7  Edges.resize(NewEnd - Edges.begin());
```
Streaming Layer and Pipelining

- EM algorithm $\implies$ data flow through a DAG
- Feed output data stream **directly** to the consumer algorithm
- Saves many (factor 2–3) I/Os in many EM algorithms
- A new *iterator-like* interface for EM algorithms
- Basic pipelined implementations (file, sorting nodes, etc.) provided by **STXXL**
## STXXL Performance: a Benchmark

- Maximal Independent Set (+input generation)
- I/O optimal [ZehPhd]: time forward processing, scanning, sorting, priority queue

```cpp
1    pq_type depend(PQ_PPOOL_MEM,PQ_WPOOL_MEM);
2    stxxl::vector<node_type> MIS; // output
3    for(;++!edges.empty();++edges) {
4        while (!depend.empty() && edges->src > depend.top())
5            depend.pop(); // delete old events
6        if(depend.empty() || edges->src != depend.top()) {
7            if(MIS.empty() || MIS.back() != edges->src )
8                MIS.push_back(edges->src);
9            depend.push(edges->dst);
10        }
11    }
```
MIS: Running Times

- **Debian Linux, g++ -O3**
- 2×Xeon 2GHz
- **single disk**
- \( N = 2000 \) MBytes
- \( M = 512 \) MBytes
- TPIE: only graph gen.
- **STXXL PQ is 3 times faster**
MIS: Larger Inputs

- Only graph generation
- single disk
- \( N = 16 \) GBytes
- \( M = 512 \) MBytes
- Scales well

![Graph](image)

- Merging+Writing
- Filling+Run form.
- Dup. removal
- Sorting
- Filling
MIS: More Disks

- 2,4 disks
- \( N = 2000 \) MBytes
- \( M = 512 \) MBytes
- Pipel. – CPU bound
- I/O-wait counters

Graph showing the time (s.) for different configurations:
- STL_1_disk
- STL_2_disks
- STL_4_disks
- Pipel_1_disk
- Pipel_2_disks
- Pipel_4_disks

Legend:
- Merging+MIS comp.
- Filling+Run form.
- MIS computation
- Dup. removal
- Sorting
- Filling
MIS: The Largest Graph

- The largest graph:
- $4.3 \cdot 10^9$ nodes, $13.4 \cdot 10^9$ edges = 100 GBytes
- Working space takes 4 hard disks
- Computation on an Opteron system took 3h 7min
STXXL Projects

Fast suffix array construction:
- index $4 \cdot 10^9$ characters in 11h on a PC [in ALENEX05]

Minimum spanning tree:
- 96 GByte input graph in 8h 40min on a PC [in TSC04]

BFS: $32 \cdot 10^6$ nodes, $128 \cdot 10^6$ edges
- grid graphs in less than a day
- random sparse graphs within an hour [in SODA06]

Network analysis:
- count triangles in WWW link graph
  $135 \cdot 10^6$ nodes, $1.2 \cdot 10^9$ edges in 4h 46min
Future Work

- **Windows port**: done
- More flexible error handling using C++ exceptions
- Inclusion in **Boost** libraries
- More algorithms and data structures: hash tables, strings, string sorting, ...
- Support of arbitrary variable-size objects