Praktikum Sekundärspeicheralgorithmen
Algorithmik II

Sanders, Dementiev, Schultes, Singler SS 06

http://algo2.iti.uka.de/ioprakt06.php

Experiment 6

Deadline: 13:00 — July 24, 2006

Semi-external Longest Common Prefix (LCP) computation and STXXL pipelining

Exercise 1

Read the following material:

- A semi-external LCP computation algorithm in [Kär03].
- STXXL pipelining in [DKS05] Section "Streaming Layer". See also Doxy documentation: `http://algo2.iti.uka.de/dementiev/stxxl/doxy/html/group__streampack.html` and examples `http://algo2.iti.uka.de/dementiev/stxxl/doxy/html/examples.html`

Exercise 2

Implement the semi-external LCP algorithm without pipelining using only the STL user layer. Your program must:

- use `char` as the character type. Assume $n < 2^{32}$, values stored in SA and LCP arrays are 32-bit `unsigned int`s.
- read the input string T and the suffix array SA from files and output the LCP array to a file. For this purpose `stxxl::vector`s can be mapped to files as in the previous experiment.
- be user-friendly, the file names of the input and output files must be given in the command line parameters.
- accept inputs without the special symbol $T[n]$ at the end of T. Assume you are given $T[0, n-1]$ and $SA[1, n]$ following the notation of [Kär03].
- use only 512MB of main memory for sorting. It is assumed that `char` array T fits in the main memory in the second stage of the algorithm. The caches of `stxxl::vector`s should take less than 16 MBytes each and have 4 pages.
- print the maximum LCP value.

Find the precise I/O volume function of your implementation. Assume that only one merge pass is needed in the external sorting.

Exercise 3

Draw a data flow graph of a pipelined implementation of the semi-external LCP algorithm similar to those in [DMKS05]. Using the data flow graph find the precise I/O volume function of your *pipelined* implementation. Assume that only one merge pass is needed in the external sorting. Implement a pipelined version of the semi-external LCP algorithm following the same design rules as in the previous exercise. Note, that for each pipelined sorter only 256 MB should be used, as two pipelined sorters will run simultaneously (merging and run formation) and share the main memory.

Exercise 4

Measurements:

- Measure the running time of the non-pipelined and the pipelined implementations for $n = 2^{20}, 2^{21}, \ldots, 2^{29}$ (powers of 2) characters. The input families:
  - Real-world instances (source code): prefixes of size n of the text in `/home/dementiev/courses/instances/sourcedat`. Run the measurements the whole file as well: $n = 547505710$. The suffix arrays of the prefixes are stored in files `sourcedat.`*`n`*`.sa`
  - Synthetic instances: concatenation of two identical random sequences of length $n/2$, stored in files `random_string.`*`n`*. Their suffix arrays are stored in files `random_string.`*`n`*`.sa`

  Copy the input files to your local disk for further usage.
- Besides the running time, measure the I/O volume and I/O wait time using `stxxl::stats` class (methods `get_read_volume` and `get_write_volume`).
- Try to tune your implementations changing the block sizes.

Write a report that includes *at least* the following:

- The analyses for Exercise 2 and 3.
- The flow graph from Exercise 3.
- The running time of implementations drawn in a single figure (plot, as usual, time per input item).
- The measured I/O volume and the I/O volume computed by the formulas of the implementations drawn in a single figure (bytes per input item).
- A plot and a table of the maximum LCP values for both of the input families.
- Discussion of the results:
  - Do the computed and measured I/O volumes match (closely)? Why?
  - Which implementation is faster? Why?
  - Which implementations are compute bound and I/O bound respectively? Why?

Send your source code and your report with figures to `dementiev@ira.uka.de` before the deadline. Also, make an appointment with Roman Dementiev for the defense of your work.

References

[DKS05] R. Dementiev, L. Kettner, and P. Sanders. Stxxl: Standard Template Library for XXL Data Sets. Technical Report 18, Fakultät für Informatik University of Karlsruhe, 2005. `http://algo2.iti.uka.de/dementiev/files/TRKA2005_18.pdf`.

[DMKS05] R. Dementiev, J. Mehnert, J. Kärkkäinen, and P. Sanders. Better External Memory Suffix Array Construction. In *Workshop on Algorithm Engineering & Experiments*, Vancouver, 2005. `http://i10www.ira.uka.de/dementiev/files/DKMS05.pdf`.

[Kär03] J. Kärkkäinen. `http://algo2.iti.uka.de/dementiev/courses/ioprakt06/lcp_juha.ps`, 2003.