GSB/RF
October 12, 2000

Project 2 — External Memory Libraries

The goal of the second project is twofold:

- To try out two existing software libraries for external memory computation.
- To implement and evaluate some of the algorithms from the lectures.

The libraries in question are the TPIE (Transparent Parallel I/O Environment) library developed at Duke University, and the LEDA-SM (Secondary Memory) extension package for the LEDA (Library of Efficient Data Types and Algorithms) library, developed at the Max-Planck-Institut für Informatik in Saarbrücken. The TPIE library is based in streams, while LEDA-SM is centered around I/O-efficient implementations of data structures.

For this project, the latest versions of TPIE (`www.cs.duke.edu/~tpie`) and LEDA-SM (`http://www.mpi-sb.mpg.de/~crauser/leda-sm.html`) should be downloaded and installed. The LEDA-SM package requires you to first download and install the LEDA library itself—for this, follow links starting at `www.mpi-sb.mpg.de/LEDA`. In all cases, the `.tgz`-archive retrieved contains installation instructions and a user manual.

Both TPIE and LEDA-SM are C++ libraries—consequently, this project must be programmed in C++. The report should be handed in by *Monday, November 6*.

Part A: Comparison of Libraries

The tasks of this part are the following:

1. Consider the following three implementations of streams:
   - The streams of TPIE.
   - Streams implemented by using the external queue data type of LEDA-SM.
   - The best of your own implementations from Project 1

   For each of these implementations, and for various (large) values of N, measure the time for writing the integers between 1 and N into a stream and reading them again. Compare the results.

2. Consider the following five implementations of MergeSort:
   - Using TPIE, as described in Section 4.6.1 of the TPIE manual.
   - Using the built-in sort function `AMI_sort()` of TPIE.
   - Using streams based on the external queue data type of LEDA-SM. The merging should be done by using an internal memory priority queue from the basic LEDA library.
   - Using the built-in external MergeSort algorithm from LEDA-SM, described in Section 6.2 of the LEDA-SM manual (or, alternatively, the `sort()` method of the external array data type).
   - The best of your own implementations from Project 1.

   For each of these implementations, and for various (large) values of N, measure the time for sorting N random integers. Compare the results.

Part B: Selection

The tasks of this part are the following:

1. Using TPIE, implement the classical "median of medians" linear time selection algorithm (see e.g. [1, Section 10.3]) such that the number of I/Os is $O(N/B)$. Experiment to find a good size (normally set at 5 or 7, but closer to M should be more I/O-efficient) for the basic sets, from each of which you find the median in the initial phase of an invocation of the algorithm. Also experiment to find a good way to find these medians of the basic sets—an internal version of the same algorithm, a randomized selection algorithm (e.g. the simple version using a random pivot element), or `qsort()`, are some of the possibilities.

2. Using TPIE, implement the selection algorithm of Sibeyn [4], in the simplified version described at the lecture [2]. Your should experiment to find a good value for s, and a good way to find the s samples of the basic sets of size M (see above for inspiration).

3. Using the best of the two MergeSort implementations based on TPIE from Part A, implement a selection algorithm based on sorting.

4. Compare the efficiency of the three implementations when finding the median of N random integers, for large values of N.

Part C: Random Permutation

The tasks of this part are the following:

1. Implement the random permutation algorithm of Sanders [3] using streams based on the external queue data type from LEDA-SM. The internal permutation in the base case of the recursion should be done using the `permute()` method of the array data type in the basic LEDA library.

2. Implement an algorithm for random permutations where each element is assigned a random integer in a scan of the input followed by sorting the elements with respect to the assigned integers using the best of the MergeSort implementations based on LEDA-SM from Part A. Ties should be handled by recursion (during a scan of the elements after the sorting).

3. Compare the efficiency of the two implementations when randomly permuting the integers from 1 to N, for large values of N.

Part D: Conclusion

Try to give an evaluation of the impact of using the two libraries with respect to:

- Efficiency of resulting programs.
- Easy of coding.

References

[1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[2] Rolf Fagerberg. Lecture notes for *External Selection.*
See `www.daimi.au.dk/~gerth/emF00/SelectionSlides.ps.gz`.

[3] Peter Sanders. *Random Permutations on Distributed, External and Hierarchical Memory.* Information Processing Letters 56:305–309, 1998.

[4] Jop F. Sibeyn. *External Selection.* Proceedings of the 16th Symposium Theoretical Aspects of Computer Science, Lecture Notes in Computer Science Vol. 1563, Springer-Verlag, 1999, 291-301.