Exercises on High Performance Computing
Block 1: Memory Hierarchy

S. Hunold, J. L. Träff, and F. Versaci

Parallel Computing Group
TU Wien

15 April, 2013

Generalities

Instructions. There will be four (or maybe five) blocks of exercises and you are required to solve *two blocks* out of them.

- To solve this block, solve *one of the two* following exercises (preferably in C/C++).
- To avoid penalizations, observe *exactly* the provided guidelines.
- The implementation will be judged based on correctness and performance.
- It is *strictly forbidden* to copy/take inspiration from code from other people or taken from internet. You are required to *write your own code*.
- You are required to hand in (to the contact email) your source code (including the Makefile, excluding any input file you might have used for testing purposes) in a tar.gz or zip file.
- You are also required to attach a document (5-15 pages, containing no source code, to be provided in pdf format) which should contain roughly the following sections:
  1. Broad description of the implemented algorithm (1-3 pages)
  2. Some implementation details (1-3 pages)
  3. Presentation of experimental results (2-6 pages)
  4. Analysis of the results (2-4 pages)

Contact. If you need some clarifications for this block, contact `versaci@par.tuwien.ac.at`.

Deadline. 24 June 2013, 24:00.

Exercises

1. We are interested in computing the number of misses incurred by the LRU eviction policy, given an input address trace and for any possible buffer size.

   - The address trace is made of 64-bit little-endian unsigned integers, stored in a file (create your own files for testing purposes).
   - The name of the input file should be passed as an argument in the command line.
   - The program should be named `lru-misses` and should either come with a Makefile or bash script for compiling it.

- The program should provide as output in the stdout the number of misses for buffers of sizes 2^k, with $k \in \{0, 1, \ldots, 64\}$.
- The number of misses should be printed in plain text, one per line.
- The expected behavior is thus the following:

```
$ make
$ ./lru-misses inputfile.bin
46842
23651
18454
...
```

- The algorithm you are required to implement is the one appearing in [1]

2. We want to sort an array of little-endian doubles, stored in a file, and save the sorted output into another file (both filenames to be passed as arguments in the command line).

   - The file might not fit into main memory (external sort) and should be mapped to virtual memory using `mmap` calls.
   - The program should be named `co-sort` and should either come with a Makefile or bash script for compiling it.
   - The expected behavior is thus the following:

   ```
   $ make
   $ ./co-sort inputfile.bin outputfile.bin
   ```

   - You are required to implement a cache oblivious sorting algorithm, choosing among the following ones
     (a) Distribution sorting [5]
     (b) Funnelsort [5]
     (c) Lazy funnelsort [2, 3]
     (d) Proximity mergesort [4]
   - You are also required to compare the performance against a standard quick sort implementation (you can use C `qsort` from stdlib or C++ STL `sort`)

References

[1] Almási, G., Caşcaval, C., and Padua, D. A. Calculating stack distances efficiently. *SIGPLAN Not. 38*, 2 supplement (June 2002), 37–43.

[2] Brodal, G. S., and Fagerberg, R. Cache oblivious distribution sweeping. In *ICALP* (2002), pp. 426–438.

[3] Brodal, G. S., Fagerberg, R., and Vinther, K. Engineering a cache-oblivious sorting algorithm. *ACM Journal of Experimental Algorithmics 12* (2007).

[4] Franceschini, G. Proximity mergesort: optimal in-place sorting in the cache-oblivious model. In *SODA* (2004), pp. 291–299.

[5] Frigo, M., Leiserson, C. E., Prokop, H., and Ramachandran, S. Cache-oblivious algorithms. *ACM Trans. Algorithms 8*, 1 (Jan. 2012), 4:1–4:22.