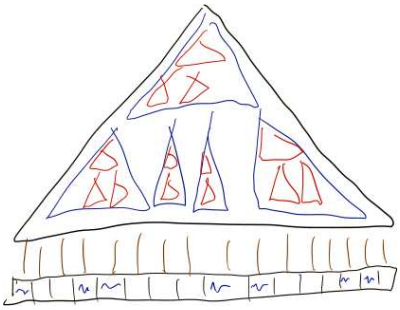
Université Libre de Bruxelles

Computer Science Department

INFO-F530 Computer science seminar

Report of

Seminar 1



Cache-Oblivious Algorithms & Data Structures

(by **John Iacono** – October 12, 2017 [Iac17])

Olivier Pirson – opi@opimedia.be
Master 2

June 4, 2018

ULB

Abstract of the presentation by John Iacono [Iac]

In the standard model of computation often taught in computer science courses one identifies elementary operations and counts them in order to obtain the runtime. However, given the complexity of computing hardware, this measure often does not correlate well with actual observed runtime on a computer; accessing n items sequentially or randomly typically have runtimes that differ by several orders of magnitude. In this talk I will present the cache-oblivious model of computation, a model that was introduced by Prokop *in 1999 and is relatively easy to reason with, by modeling the multilevel caches that are a defining feature of the cost of modern computation. After presenting the model, several data structure and algorithms that illustrate design techniques to develop cache-obliviously optimal structures will be presented. [Car+17]*

(Illustration of the title page from [Iac17].)

Contents

1 Provide an outline of the content of the seminar in your own words.

With the usual measure of the algorithmic complexity, the actual characteristics of our computers are abstracted. In particular for this topic the presence or not of a cache system between the CPU and the memory is ignored. The consequence is that two algorithms for the same problem may be have a significant difference of efficiency, even if they are the same usual measure of time complexity.

By example the visit of all the items of an array in a sequential scan or in a random order have both a linear cost in terms of the usual time complexity. But in practice the first one is really faster that the second one.

A way to understand this is to use the *disk access model*. This model counts the number cache misses, the number of blocks of size B transferred from the memory content of size N to a cache of size M. [1] A key point is that the move of a complete block takes the same time that the move of only one item. And because the cache is really faster than the memory, all the work done in the cache is considered as free. With this model we see that the sequential scan of an array minimizes the number of cache misses while the visit of all items in a random order maximizes it. [2]

Then John Iacono uses this model to analyse a B-tree, a search algorithm with the same complexity than a binary search but with a better use of the cache. [3] Instead an one dimensional array as used by the binary search, this is a tree, cut to subtrees of size B.

B-trees require to know B and M to build the structure. So to be efficient the implementation must be tuned for the computer used. And it is more difficult yet since our computers have a cache hierarchy instead a unique cache.

The idea of a *cache-oblivious algorithm* is to design it such that B and M are not used by the algorithm itself. They "*can be used for the analysis, but not by the algorithm.*" [Iac17]

Instead to cut to subtrees of size B, the Van Emde Boas tree data structure cuts the tree to subtrees of size $\sqrt{N}$ (corresponding to the half of the height) and recursively does the same on each subtrees. The beauty is that the number of cache misses is the same as the B-tree but the algorithm never used the parameters B and M.

Next John Iacono explains the funnelsort, an oblivious-cache version of the well known mergesort. The idea is also to recursively cut to subtrees of size $\sqrt{N}$ and to replace the merge of two subarrays of mergesort by a k-merger.

2 Which were the major points discussed by the speaker? List and explain their importance.

- Work in cache is free.

  The cache is really faster than the memory, so the most runtime is lost to transfer blocks from memory to the cache, and it is like all (simple) computations in the cache are free.

  This is the important concrete fact that makes the difference in efficiency. Two algorithms that seem equivalents, but with a number of different cache misses, have actually a different efficiency.

  Exploit the locality of data minimizes the number of visited blocks or from the opposite point of view maximizes the number of times we get access to the same block.

- Cache parameters "*can be used for the analysis, but not by the algorithm.*" [Iac17]

[1] We assume N bigger than M.
[2] The disk access model gives $\mathcal{O}\left(\left\lceil \frac{N}{B} \right\rceil\right)$ for the sequential scan and $\Theta(N)$ with the random order.
[3] $\mathcal{O}(\log_2(N))$ for the binary search with N enough big, and $\mathcal{O}(\log_B(N))$ for B-tree.

The algorithm takes advantage of the presence of the cache, but it must never use its parameters. Only the analysis can use them.
This is the cornerstone of the cache-oblivious algorithms. They take advantage of the presence of the cache hierarchy, but they do not have to be tuned for particular sizes and number of caches.

- Divide and conquer, recursion until to have subproblems small enough.

  The general principle used here to build cache-oblivious algorithms is the recursion to cut the problem to subproblems, and so forth until each subproblem can be resolved completely in the cache system, whatever it is.
  This is a way to obtain a good use of the cache hierarchy without having to tune the algorithms for each actual computer.

> "*The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.*"
>
> — Donald E. Knuth, [Knu74]

> "*Elegant recursion > overengineering*"
>
> — John Iacono, [Iac17]

3 For which discipline in computer science is the topic relevant and why (explain)?

It is a general algorithmic concept that can be used in all computer science disciplines to design programs running on actual computers, to take advantage of different kind of cache systems. This kind of cache system can be the cache hierarchy between the CPU and the memory, but it also can be between memory and disk, or between processes in a network.

4 Suggest two or three articles discussing similar work (not necessarily published by the speaker). Explain why they are related.

- *Cache-Oblivious Algorithms* [Pro99]
  It is the master's thesis of Harald Prokop mentioned several times during the presentation.
  This work gives a history of the cache-oblivious concept. It exposes an *ideal-cache model* to analyse ordinary algorithms and cache-oblivious algorithms. It presents optimal (in an asymptotically sens) cache-oblivious algorithms for some common tasks (for example the funnelsort). It shows that an optimal cache-oblivious algorithm for the ideal-cache model is also optimal for a more complicated cache hierarchy.
  Similar material in the shorter article co-signed by Prokop: *Cache-Oblivious Algorithms – Extended Abstract* [Fri+99].
  A more recent paper by Piyush Kumar presents again similar material, with some experimental results: *Cache Oblivious Algorithms* [Kum03].
- *The Cache Complexity of Multithreaded Cache Oblivious Algorithms* [FS09]
  After a presentation of different kinds of cache model in multithreads context, this paper introduces a model where each processor has a private cache: "*the ideal distributed cache model for parallel machines as an extension of the (sequential) ideal cache model*" [FS09]. It presents some parallel cache-oblivious algorithms.

5 Propose two scientific questions relevant for the topic discussed in this seminar.

- Recursion is the principle used to build the presented cache-oblivious algorithms. Can you evoke some cache-oblivious algorithms built without recursion? If yes, which are the principles used to build them? And if not, maybe some recursions used are tail recursive?
- In a context of parallel or distributed programs the communication between processes is a bottleneck (because of synchronisation problems and/or slow communication). Do you know some cache-oblivious algorithms built to minimize this communication?

6 Would you like to know more about this field? Do you have particular critiques about the scientific content? Provide a brief motivation for your answers.

John Iacono has started from the common knowledge of students and has explained us a new general theoretical concept that has practical profit. I found that interesting.

Now when I will need a really efficient algorithm for a problem I will search if there exists a cache-oblivious version.

It would be nice if I could find a cache-oblivious idea to make the parallelization of abstract interpreters for my master thesis [Pir18] faster. Maybe in a distributed context.

References

[Car+17] Jean Cardinal, Martine Labbé, Tom Lenaerts, and Olivier Markowitch. **Computer Science Seminar INFO-F-530**. 2017. URL: https://web.archive.org/web/20180601005154/https://www.ulb.ac.be/di/map/tlenaert/Home_Tom_Lenaerts/INFO-F-530.html (cit. on p. 1).

[Fri+99] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. **Cache-Oblivious Algorithms – Extended Abstract**. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039). 1999, pp. 285–297. DOI: 10.1109/SFFCS.1999.814600 (cit. on p. 4).

[FS09] Matteo Frigo and Volker Strumpen. **The Cache Complexity of Multithreaded Cache Oblivious Algorithms**. In: *Theory of Computing Systems* 45.2 (Aug. 1, 2009), pp. 203–233. ISSN: 1432-4350, 1433-0490. DOI: 10.1007/s00224-007-9098-2 (cit. on p. 4).

[Iac] John Iacono. **John Iacono**. URL: http://www.johniacono.com/ (cit. on p. 1).

[Iac17] John Iacono. **Cache-Oblivious Algorithms & Data Structures**. Université Libre de Bruxelles, Oct. 23, 2017. URL: https://web.archive.org/web/20180601125033/https://www.ulb.ac.be/di/map/tlenaert/Home_Tom_Lenaerts/INFO-F-530_files/DLS%20John%20Iacono.pdf (cit. on pp. 1, 3, 4).

[Knu74] Donald E. Knuth. **Computer Programming as an Art**. In: *ACM Turing Award Lectures*. New York, NY, USA: ACM, 1974. DOI: 10.1145/1283920.1283929 (cit. on p. 4).

[Kum03] Piyush Kumar. **Cache Oblivious Algorithms**. In: *Algorithms for Memory Hierarchies*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2003, pp. 193–212. DOI: 10.1007/3-540-36574-5_9 (cit. on p. 4).

[Pir18] Olivier Pirson. **An Efficient and Parallel Abstract Interpreter in Scala — Bitbucket**. An Efficient and Parallel Abstract Interpreter in Scala. 2018. URL: https://bitbucket.org/OPiMedia/efficient-parallel-abstract-interpreter-in-scala/ (cit. on p. 5).

[Pro99] Harald Prokop. **Cache-Oblivious Algorithms**. Master's thesis. Massachusetts Institute of Technology, June 1999. URL: http://supertech.csail.mit.edu/papers/Prokop99.pdf (cit. on p. 4).