

An Oblivious Approach to Parallel Algorithms*

Francesco Silvestri

Department of Information Engineering
University of Padova, Italy
E-mail: `silvest1@dei.unipd.it`

Abstract

This extended abstract describes my dissertation work undertaken at the University of Padova under the supervision of Prof. A. Pietracaprina. My work has focused on the development of network-oblivious algorithms, that is, parallel algorithms that can run unchanged yet efficiently on a variety of machines characterized by different degrees of parallelism and communication capabilities. Also, I have studied the limitations of the cache-oblivious and network-oblivious approaches.

Keywords: *Network-oblivious, cache-oblivious, parallel algorithms, memory hierarchy.*

Introduction

Intuitively, a *cache-oblivious* algorithm [3] implements an adaptive strategy which runs efficiently on any memory hierarchy without requiring previous knowledge of the hierarchy parameters. For this reason, cache-oblivious algorithms are particularly attractive in a global computing environment, where software may be run on a variety of different platforms for load management purposes. In such a scenario the actual platform onto which an application is ultimately run, may be unknown at the time when the application is designed. In the recent years, a lot of effort in literature has been concentrated upon the development of efficient cache-oblivious algorithms for many problems, and some cache-oblivious techniques have also been commercialized (e.g., *Tokutek*,

*Some research papers cited in this extended abstract are available at <http://www.dei.unipd.it/~silvest1>. This work is supported in part by the EU/IST Project “AEOLUS”, and by MIUR of Italy under projects “MAINSTREAM”.

Inc. developed a storage technology based on cache-oblivious B-trees).

Since the advent of *Chip Multiprocessors* (CMPs), the design of efficient yet portable parallel algorithms has received more attention. While the issue of portability is less crucial for algorithms developed for special-purpose massively parallel platforms (e.g., *IBM BlueGene/L*), it becomes a primary concern when designing algorithms for CMPs which are likely to be used in a variety of application scenarios and with different (possibly unknown) machine configurations. In literature there are many models, like the *Decomposable Bulk Synchronous Parallel (D-BSP)* model [1], which efficiently describe many significant parallel platforms through few parameters. However, algorithms formulated in these models require the estimation of several architectural parameters, which may be hard and time consuming to do in practice.

Dissertation work

It is natural to wonder whether, at least for some problems, parallel algorithms can be designed that, while independent of any model parameters, are nevertheless efficient for a wide range of such parameters. My dissertation work has focused on this question, that is, on exploring the world of *network-oblivious* algorithms.

The explorative work [4] gives the first insight into the relations between parallel algorithms and cache-oblivious ones. This work provides a simulation technique through which efficient cache-oblivious algorithms are obtained from efficient parallel ones, thus reinforcing the known relation between parallelism and memory hierarchies. A natural question arises regarding the possibility of developing par-

allel algorithms which are oblivious to certain machine parameters. In [2] we addressed this issue by introducing the notion of network-oblivious algorithms and by defining a framework for their design and analysis. This framework is composed of three models of computation: the *algorithm specification model* $\mathcal{M}(n)$, which consists of a clique of processors whose number is a function of the input size n ; the *algorithm evaluation model* $\mathcal{M}(p, B)$, which is still a clique but defined by two parameters, i.e. the processor number p and the communication block-size B which capture parallelism and granularity of communication respectively; the *execution machine model*, which aims to describe the platforms where algorithms are actually carried out through few parameters. We adopted the D-BSP [1] as execution machine model. A network-oblivious algorithm is an algorithm formulated in the $\mathcal{M}(n)$ model, whose communication complexity is, roughly speaking, the number of communications performed by its execution on an $\mathcal{M}(p, B)$. More interesting, for a wide class of network-oblivious algorithms, optimality in the evaluation model for all values of the two parameters implies optimality on an arbitrary configuration of parameters of the execution model.

To help placing the network-oblivious framework in perspective, it may be useful to compare it with the well established cache-oblivious framework. In the latter, the specification model is the *Random Access Machine*; the evaluation model is the *Ideal Cache* model [3], which is composed of a slow unbounded memory and one level of cache described by its size and line length; the execution model is a machine with an arbitrary multilevel memory hierarchy. Moreover, optimality in the two-level model for all values of the two cache parameters translates into optimality on an arbitrary multilevel hierarchy.

In [2], optimal network-oblivious algorithms for matrix multiplication, matrix transposition, sorting, and Fast Fourier Transform are provided. It must be noticed that many of these network-oblivious algorithms are reminiscent of their cache-oblivious counterparts. Moreover, matrix transposition shows a similar negative result on both the cache-oblivious and network-oblivious frameworks. Specifically, many cache-oblivious algorithms reach optimality only in those caches where the cache size is at least the square of the line length (*tall-cache assumption*). In [6, 5] it is proved that there cannot exist

a cache-oblivious algorithm for performing any rational permutation for all values of the cache parameters. This result implies that a cache-oblivious algorithm for matrix transposition or the reverse of a vector, which are special cases of rational permutations, cannot be optimal in every cache. Similarly, in the network-oblivious scenario, many algorithms reach optimality only in those models which satisfy an assumption which requires input size, processor number and communication block length to verify a technical relation which is similar to the tall-cache assumption. In [2] it is proved that there cannot exist a network-oblivious algorithm for matrix transposition whose communication complexity is optimal for all values of the parameters of the evaluation model.

Further research

These results offer many new research opportunities. Naturally, the design of efficient network-oblivious algorithms can be considered for many key problems, beyond the few case studies provided in [2]. It would also be useful to identify other classes of machines for which network-oblivious optimality translates into optimal time, and to develop lower-bound techniques that limit the level of optimality that cache and network-oblivious algorithms can reach on certain classes of target platforms.

References

- [1] G. Bilardi, A. Pietracaprina, and G. Pucci. Decomposable BSP: A bandwidth-latency model for parallel and hierarchical computation. In J. Reif and S. Rajasekaran, editors, *Handbook of Parallel Computing*. Chapman & Hall/CRC, 2008.
- [2] G. Bilardi, A. Pietracaprina, G. Pucci, and F. Silvestri. Network-oblivious algorithms. In *Proc. of the 21st IPDPS*, 2007.
- [3] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. of the 40th FOCS*, pages 285–298, 1999.
- [4] A. Pietracaprina, G. Pucci, and F. Silvestri. Cache-oblivious simulation of parallel programs. In *Proc. of the 8th APDCM Workshop*, 2006.
- [5] F. Silvestri. On the limits of cache-oblivious matrix transposition. In *Proc. of the 2nd TGC*, LNCS 4661, pages 233–243, 2006.
- [6] F. Silvestri. On the limits of cache-oblivious rational permutations. *Special Issue of Theoretical Computer Science on the 2nd TGC*, to appear.

