High Performance Data Structures for Multicore Environments

Giuliano Laccetti
Department of Mathematics and Applications
Università degli Sudi di Napoli Federico II (Italy)
giuliano.laccetti@unina.it

Marco Lapegna
Department of Mathematics and Applications
Università degli Studi di Napoli Federico II (Italy)
marco.lapegna@unina.it

The rise of new multicore CPUs introduced new challenges in the design of concurrent data structures: in addition to traditional requirements like correctness and progress, the scalability is of paramount importance [1, 4, 5]. It is a common opinion that these two demands are partially in conflict each other, so that in these computational environments it is necessary to relax the requirements on a behavior identical to the corresponding sequential data structures to achieve high performances.

Our work joins a recent research trend based on a distributed approach in shared memory environments (e.g. [3]), with the aim to deal with a heap-based priority queue $\mathcal{S}$, that is a dynamical data structure where each node $s(k) \in \mathcal{S}$, $(k = 1, .., K)$ is tagged with a problem-dependent priority $e(k)$ and where the item $\hat{s}$ with highest priority $\hat{e} = \max_k e(k)$ is in the root.

In a multicore CPU, a suitable way to implement a priority queue in an iterative algorithm is to process several nodes simultaneously by threads running on different cores. In a centralized approach, where all threads access a single shared heap with a global synchronization, all the basic operations on the heap must be carried out in a critical section, with a strong efficiency degradation [2].

For these reasons, our first step in the development of a scalable heap, is to remove all global critical sections from the algorithm. To this aim, we give up the idea of a single centralized heap, and we reorganize the data structure $\mathcal{S}$ in N separate heaps $\mathcal{S}_i$, one for each thread p_i, each of them accessing a private data structure without synchronizations with other threads. Without global synchronization we have a pleasantly parallel algorithm, where it is easy to achieve very high efficiencies.

However also this approach has a side effect: because the complete disjunction of the heaps $\mathcal{S}_i$, the N items with the highest priority $\hat{s}_i$, resident in the roots of $\mathcal{S}_i$, are not those that globally have the highest priority, so that some threads can process unimportant items with risk of no significant progress for the whole application. Therefore it is important to observe that, in case of items with priority unevenly distributed among the local heaps, it should be desirable a periodical redistribution strategy for the highest priority items $\hat{s}_i$, in order to balance them among the threads.

In our relaxed approach, the N threads p_i are logically arranged according a 2-dimensional periodical mesh $\mathcal{M}_2$. This structure is a virtual grid of threads arranged along the points of a 2-dimensional space with integer non negative coordinates. In addition, the corresponding threads on the opposite faces of the mesh are connected too, so that the mesh is periodical. A shared buffer between each couple of connected nodes is established to allow sharing data according to a producer-consumer protocol. In this topology, at the iteration j, each thread p_i attempts to share its item $\hat{s}_i \in \mathcal{S}_i$, with highest priority $\hat{e}_i$ only with the next thread $p_{i+}^{(dir)}$ in the direction $dir = mod(j, 2)$ of $\mathcal{M}_2$, that is alternatively in the two directions of the mesh.

More precisely, in a fixed direction dir let $\hat{e}_i$ $\hat{e}_{i+}$ and $\hat{e}_{i-}$ be respectively the highest priority of the items in the heap root of the threads p_i, $p_{i+}^{(dir)}$ and $p_{i-}^{(dir)}$. If $\hat{e}_i > \hat{e}_{i+}$ then the item $\hat{s}_i \in \mathcal{S}_i$ with highest priority $\hat{e}_i$ is moved to the heap $\mathcal{S}_{i+}$ along the direction dir, using a producer-consumer protocol on the shared buffer. In the same way if $\hat{e}_{i-} > \hat{e}_i$ the item $\hat{s}_{i-} \in \mathcal{S}_{i-}$ with highest priority $\hat{e}_{i-}$ is moved to the heap $\mathcal{S}_i$. In this way, the critical items with highest priority are passed from thread in thread, an iteration after the other, through all the nodes of $\mathcal{M}_2$, with a synchronization overhead that does not depend on the number of threads N. It follows that the resulting algorithm can be considered scalable.

Our experiments are carried out on a system with 4 Intel Xeon E5-4610 v2 with 8-core (Harpertown) running at 2.33GHz, for a total of 32 cores. They show that for a scientific problem, the strategy is able to realize an effective compromise between parallel efficiency and numerical accuracy.

1. REFERENCES

[1] J. Dongarra, D. Gannon, G. Fox, K. Kennedy. The Impact of Multicore on Computational Science Software. in CTWatch Quarterly, Vol. 3 (2007)

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, A. White. Sourcebook of Parallel Computing. Morgan Kaufmann Publishers, 2003

[3] A. Haas, T. Henzinger, C. Kirsch, M. Lippautz, H. Payer, A. Sezgin, A. Sokolova. Distributed Queues in Shared Memory. in CF '13 Proceedings of the ACM International Conference on Computing Frontiers, ACM New York 2013, Article No. 17

[4] M.P. Herlihy, N. Shavit. The art of Multiprocessor

Programming. Rev. first ed., Morgan Kaufmann, 2012.
[5] N. Shavit. Data Structure in Multicore Age. Communication of the ACM, vol. 54 (march 2011), pp. 76-84.