Information Systems 36 (2011) 808–823

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Parallel skyline computation on multicore architectures ☆

Hyeonseung Im, Jonghyun Park, Sungwoo Park *

Pohang University of Science and Technology (POSTECH), Republic of Korea

ARTICLE INFO

Article history:
Received 19 May 2010
Received in revised form
8 September 2010
Accepted 6 October 2010
Recommended by: L. Wong
Available online 5 January 2011

Keywords:
Skyline computation
Multicore architecture
Parallel computation

ABSTRACT

With the advent of multicore processors, it has become imperative to write parallel programs if one wishes to exploit the next generation of processors. This paper deals with skyline computation as a case study of parallelizing database operations on multicore architectures. First we parallelize three sequential skyline algorithms, BBS, SFS, and SSkyline, to see if the design principles of sequential skyline computation also extend to parallel skyline computation. Then we develop a new parallel skyline algorithm PSkyline based on the divide-and-conquer strategy. Experimental results show that all the algorithms successfully utilize multiple cores to achieve a reasonable speedup. In particular, PSkyline achieves a speedup approximately proportional to the number of cores when it needs a parallel computation the most.

1. Introduction

Multicore processors are going mainstream [26]. As a response to the problem of excessive power consumption and the lack of new optimization techniques, the industry has adopted a new strategy for boosting processor performance by integrating multiple cores into a single processor instead of increasing clock frequency. In upcoming years, we will see processors with eight, sixteen, or more cores, but not with much higher clock frequency.

The advent of multicore processors is making a profound impact on software development [27]. As there is little performance gain when running sequential programs on multicore processors, it is imperative to write parallel programs in order to exploit the next generation of processors. Due to simpler design and lower clock frequency in individual cores, sequential programs may even experience performance loss on tomorrow's multicore processors.

This radical change in processor architectures begs an important question for the database community: *how can we exploit multicore architectures in implementing database operations*? Since multicore architectures combine multiple independent cores sharing common input/output (I/O) devices, this question is particularly relevant if database operations under consideration are computationally intensive, but not I/O intensive. In such cases, multicore architectures offer an added advantage of negligible or low overhead for communications between parallel threads, which we can implement as reads and writes to the main memory or disk.

This paper deals with *skyline computation* [2] as a case study of parallelizing database operations on multicore architectures. Given a multi-dimensional dataset of tuples, a skyline computation returns a subset of tuples, called *skyline tuples*, that are no worse than, or not dominated by, any other tuples when all dimensions are considered together. Because of its potential applications in decision making, skyline computation has drawn a lot of attention in the database community [14,3,9,19,16,1,35].

The computationally intensive nature of skyline computation makes it a good candidate for parallelization especially on multicore architectures. Typically the cost of skyline computation depends heavily on the number of comparisons between tuples, called *dominance tests*,

☆ This is a significantly extended version of the paper that appeared in the 22nd IEEE International Conference on Data Engineering, 2009 [20]. This work was supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2008-313-D00969).

* Corresponding author.
E-mail addresses: genilhs@postech.ac.kr (H. Im), parjong@postech.ac.kr (J. Park), gla@postech.ac.kr (S. Park).

0306-4379/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2010.10.005

which involve only integer or floating-point number comparisons and no I/O. Since a large number of dominance tests can often be performed independently, skyline computation has a good potential to exploit multicore architectures. So far, however, its parallelization has been considered mainly on distributed architectures [4,31–33]; only Selke et al. recently consider its parallelization on multicore architectures [24].

We parallelize three sequential skyline algorithms of different kinds to see if the design principles of sequential skyline computation also extend to parallel skyline computation. From index-based skyline algorithms, we choose the branch-and-bound skyline (BBS) algorithm [19] which uses R-trees [10] to eliminate from dominance tests a block of tuples at once. From sorting-based skyline algorithms, we choose the sort-filter-skyline (SFS) algorithm [3] which presorts a dataset according to a monotone preference function so that no tuple is dominated by succeeding tuples in the sorted dataset. We also parallelize a new nested-loop skyline algorithm, called SSkyline (Simple Skyline), which neither uses index structures nor presorts a dataset. In addition to parallelizing three sequential skyline algorithms, we also develop a new skyline algorithm, called PSkyline (Parallel Skyline), which is based on the divide-and-conquer strategy and designed specifically for parallel skyline computation. PSkyline is remarkably simple because it uses no index structures and divides a dataset linearly into smaller blocks of the same size (unlike existing divide-and-conquer skyline algorithms which exploit geometric properties of datasets).

We test the four parallel skyline algorithms on a sixteen-core machine (with four quad-core CPUs). Experimental results show that all the algorithms successfully utilize multiple cores to achieve reasonable speedups except on low-dimensional datasets and datasets with a low density of skyline tuples on which the sequential algorithms already run fast enough. In particular, the comparison between parallel BBS and PSkyline suggests that for the efficiency of a parallel skyline algorithm, a simple organization of candidate skyline tuples may be a better choice than a clever organization that eliminates from dominance tests a block of tuples at once but favors only sequential skyline computation.

Although the main topic of this paper is parallel skyline computation on multicore architectures, its main contribution also lies in providing evidence that the time is ripe for a marriage between database operations and multicore architectures. In order to exploit multicore architectures to their fullest, we may have to devise new index structures or reimplement database operations accordingly. In fact, researches along this line are already producing highly promising (and even surprising) results in the database community [17,11–13,29,6]. Certainly we do not want to find ourselves struggling to squeeze performance out of just a single core while all other 31 cores remain idle!

This paper is organized as follows. Section 2 introduces skyline computation and the parallel programming environment OpenMP [5] for implementing the parallel skyline algorithms, and discusses related work. Sections 3 and 4 explain how we parallelize BBS and SFS, respectively. Section 5 presents SSkyline and its parallelization. Section 6 presents the design and implementation of our parallel skyline algorithm PSkyline. Section 7 gives experimental results and Section 8 concludes.

2. Preliminaries

This section reviews basic properties of skyline computation and gives a brief introduction to OpenMP. Then it discusses related work.

2.1. Skyline computation

Given a dataset, a skyline query retrieves a subset of tuples, called a skyline set, that are not dominated by any other tuples. Under the assumption that smaller values are better, a tuple p dominates another tuple q if all elements of p are smaller than or equal to their corresponding elements of q and there exists at least one element of p that is strictly smaller than its corresponding element of q. Thus the skyline set consists of those tuples that are no worse than any other tuples when all dimensions are considered together.

Let us formally define the skyline set of a d-dimensional dataset D. We write $p[i]$ for the i-th element of tuple p where $1 \le i \le d$. We write $p \prec q$ to mean that tuple p dominates tuple q, *i.e.*, $p[i] \le q[i]$ holds for $1 \le i \le d$ and there exists a dimension k such that $p[k] < q[k]$. We also write $p \not\prec q$ to mean that p does not dominate q, and $p \prec\succ q$ to mean that p and q are incomparable ($p \not\prec q$ and $q \not\prec p$). Then the skyline set $\mathcal{S}(D)$ of D is defined as

$$\mathcal{S}(D) = \{p \in D | q \not\prec p \text{ if } q \in D\}$$

Note that $\mathcal{S}(D) \subset D$ and $\mathcal{S}(\mathcal{S}(D)) = \mathcal{S}(D)$ hold. We refer to those tuples in the skyline set as skyline tuples.

The computational cost of a skyline query mainly depends on the number of dominance tests performed to identify skyline tuples. A dominance test between two tuples p and q determines whether p dominates q ($p \prec q$), q dominates p ($q \prec p$), or p and q are incomparable ($p \prec\succ q$). The computational cost of a single dominance test increases with the dimensionality of the dataset.

Usually a skyline algorithm reduces the number of dominance tests by exploiting specific properties of skyline tuples. For example, transitivity of $\prec$ allows us to eliminate from further consideration any tuple as soon as we find that it is dominated by another tuple:

Proposition 2.1 (*Transitivity of* $\prec$). *If* $p \prec q$ *and* $q \prec r$, *then* $p \prec r$.

Another useful property is that we may consider incomparable datasets independently of each other. We say that two datasets D_1 and D_2 are incomparable, written $D_1 \prec\succ D_2$, if $p \prec\succ q$ holds for every pair of tuples $p \in D_1$ and $q \in D_2$.

Proposition 2.2 (*Incomparability*). *If* $D_1 \prec\succ D_2$, *then* $\mathcal{S}(D_1 \cup D_2) = \mathcal{S}(D_1) \cup \mathcal{S}(D_2)$.