MEMOCODE 2015 Design Contest: Continuous Skyline Computation

Peter Milder
Department of Electrical and Computer Engineering
Stony Brook University
Stony Brook, NY 11794–2350
peter.milder@stonybrook.edu

***Abstract*—The skyline query operation (also called the "maximum vector problem") is used to identify potentially interesting or useful data points in large sets of multi-dimensional data. When the data change over time (through addition and subtraction of points), this is called the "continuous skyline" query. The 2015 MEMOCODE Design Contest problem is to implement a system to efficiently compute the continuous skyline of dynamic data. Contestants were given one month to develop a system to perform the skyline query, aiming to maximize performance or cost-adjusted performance. Teams were encouraged to consider a variety of computational targets including CPUs, FPGAs, and GPGPUs. The two winning teams, which have been invited to contribute papers describing their techniques, combined careful algorithmic and implementation optimizations; both implemented the system on multicore CPUs.**

I. Introduction

Since 2007, the annual MEMOCODE design contests have presented a variety of problems, challenging teams from around the world to develop effective hardware/software solutions. Previous problems have included k-nearest neighbors [1], stereo matching [2], DNA sequence alignment [3], NoC simulation [4], packet inspection [5], rectangular-to-polar interpolation [6], sorting of encrypted data [7], and matrix-matrix multiplication [8].

This year's problem is to compute the continuous skyline of dynamic multidimensional data. The skyline operation is commonly used in applications such as data mining [9] to find points that are *Pareto optimal* across many dimensions.

II. Problem Overview

Skyline Computation. A common example to illustrate the idea behind the skyline operation is to consider the choice of a vacation hotel (as in [10]). Imagine you are choosing a hotel for vacation on a tropical island, and you are concerned only with each hotel's cost and its distance to the beach. We can visualize your options in the Figure 1.

If we consider only these two metrics (cost and distance) we can easily identify the hotels that you may want to consider. For example, we see that the hotel labeled "A" is both closer to the beach and less expensive than hotel "B". This means that A is said to *dominate* B. When we compare A and C, we see that A is better in one metric (distance) while C is better than A in the other (cost). For this reason, neither A nor C dominate each other. The skyline is the set of points which are not dominated by any other [10]. Figure 1 illustrates these points by connecting them with a solid line.

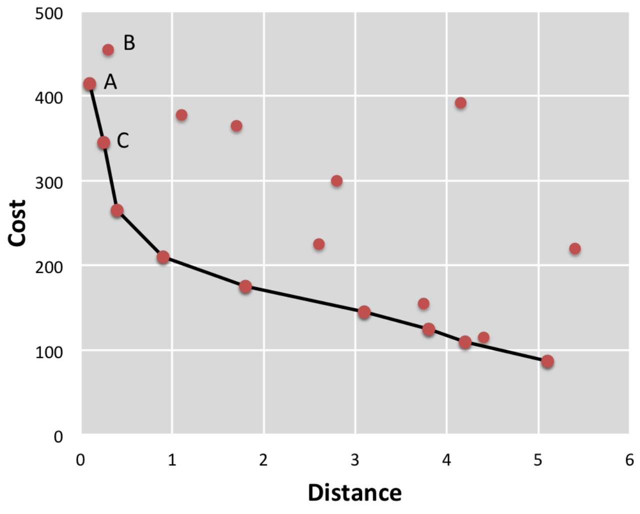

Fig. 1. Example Two-Dimensional Skyline Query

In this example, the set of data only contains two dimensions (cost and distance). If a vacationer cares about other aspects of the hotel (e.g., the hotel's quality or amenities), then more dimensions must be considered. We will use m to represent the dimensionality of the data.

Definition 1: Let D represent a set of n elements $d_0, \ldots, d_{n-1}$, each with m dimensions. The skyline of D is the set of all of its elements which are not dominated by any element of the set.

Definition 2: Element d_a dominates element d_b if and only if d_a is better than d_b in at least one dimension, and better than or equal to d_b in all m dimensions. We will say that a number x is "better" than another number y if and only if $x < y$.

Continuous Skyline. Our previous example assumed that the set of data was static. However, large complex datasets are often dynamic in nature. The continuous (or "continuous time-interval") skyline computation [11] is performed on data where entries are added or removed over time.

A naive approach to this is simply to re-calculate the entire skyline each time an element is added or removed. However, this can be quite inefficient; if only a portion of the active data has changed since the last computation, many of the necessary comparisons have already been performed.

We will model time as a series of discrete timesteps, and we will encode the dataset's changes with an activation time a and a deactivation time v for each of the n entries in the input set. At time-step t, our goal is to find the skyline among all entries in D that have activation time $a \leq t$ and deactivation time $v > t$. The outputs of our system will be the indices (in D) of the skyline elements, and the number of skyline entries found per time-step.

Problem Specification. The system takes as input: a dataset D consisting of n points in m-dimensions (where n and m are given), and two vectors of length n that correspond to the activation and deactivation times of each element in D. The data elements are sorted by activation time (that is, D is ordered from earliest to latest activation time).

For each timestep $t = 0, 1, \ldots, T$, the system must find the skyline across all data elements valid at time t. The system will store the skyline indices (indices of D) in memory, as well as the number of valid skyline points found. The maximum time-step T is the largest deactivation time in the set; the system may assume T is given. The timed portion of the solution must start with all input data in main system memory, and it must conclude with all output data in main memory.

III. Functional Reference Implementation

An unoptimized naive software implementation was provided to serve as the functional reference for the contestants' optimized implementations. The reference implementation reads input files for the dataset D, and the vectors which give the activation and deactivation time for each element. For each time step, it then finds the skyline, records the number of skyline entries found, and stores the resulting indices of the skyline elements. This functionality is summarized by:

```
initialize data;
// begin timing here
for t = 0 to T {
    for i = 0 to data_set_size-1 {
        if (i valid at time t) {
            for j = 0 to data_set_size-1 {
                isDominated = 0;
                if (j valid at time t) {
                    if (j dominates i) {
                        isDominated = 1
                        break
                    }
                }
            }
        }
        if (!isDominated)
            add i to skyline at time t
    }
    numFound[t] = num. elements found at t
}
// end timing here
sort and output results;
```

The `j dominates i` function was provided based on Definition 2.

Based on the problem definition, the order of the output values does not matter. For convenience, the functional reference sorts them by index; this will allow an easy use of `diff` to determine if two output files are equivalent. This sorting was not considered part of the contest implementation (and its runtime was not counted). Lastly, a `Makefile` was provided to compile and run the software, and to compare the results to a reference solution.

Datatype. Elements in D are given as unsigned integers. The range of the input values is limited such that every input element can be represented as a 16-bit unsigned value. The final output values are the indices of the skyline entries of D at each timestep (thus, unsigned integers between 0 and $n - 1$). Designers were allowed to optimize the datatype as appropriate. A solution is considered to be correct if all skyline indices match. There was no constraint on the order that the indices are produced during a given timestep.

Test Data. Three sets of data were provided to aid in validation and development. Each set contains an m-dimensional input dataset D, and the vectors which give the activation and deactivation time for each element. Each set is accompanied by a reference output that provides the sorted indices of the skyline elements. Details of the test data are given in Table I. All test data were generated synthetically to have the desired size and characteristics.

IV. The Contest

Participants were given one month to implement a solution using platforms such as FPGAs, GPUs, and CPUs. The solutions were validated using the supplied reference datasets. Performance was measured using the *large* dataset. The time taken to initialize data and to sort and read the final result from memory was excluded from the runtime measurement.

The submitted solutions were evaluated using two metrics: pure performance and cost-adjusted performance. The pure-performance metric was based solely on runtime, while the cost-adjusted metric was defined as the product of runtime and system cost. The system cost was determined based on the lowest listed commercial price; if no price were available, the system cost would be estimated by the judges. Contestants were encouraged to include their own estimate of system cost with their submissions.

V. Results

In order to access the reference code and data, contestants were asked to register by e-mail. Seven teams consisting of members from eight countries registered, and full working implementations were submitted by three teams. All three final submissions targeted CPU implementation.

Tables II and III summarize the results for performance and cost-adjusted performance, respectively. Note that some teams reported the runtime of their implementation on multiple processors; only the fastest or most cost-efficient solution from each team is included in the tables. The names and full affiliations of the participants of each of the three teams are included at the end of the paper.

As shown in the tables, the University of Tokyo team was the winner in both the pure performance and cost-normalized

TABLE I. TEST DATASET CHARACTERISTICS

Name	Entries	Dimensions	Time Steps	Max. Skyline Elem.	Appx. Input Size	Appx. Output Size
small	1,000	4	109	34	8 KB	15 KB
medium	50,000	5	10,998	310	500 KB	14 MB
large	800,000	7	40,998	2008	11 MB	329 MB

TABLE II. RUNTIMES FOR THE LARGE DATASET.

Team	Platform	Runtime (sec.)
University of Tokyo	Intel Core i7-4770K	0.407
IPM	AMD Opteron 6386 SE	1.4
Team DanTa	Intel Core 2 Duo P8400	517

TABLE III. COST-NORMALIZED RUNTIMES FOR THE LARGE DATASET.

Team	Platform	Cost (USD)	Runtime$\times$Cost
University of Tokyo	Intel Core i7-4770K	305	124
IPM	Intel Xeon X5650	80	280
Team DanTa	Intel Core 2 Duo P8400	59	30477

performance categories. This team included a variety of algorithmic and implementation-level optimizations to reach this performance. A key optimization strategy employed was to construct a tree structure capturing the dominance relationships at each point; this required careful procedures to update the tree as new data points become valid or invalid over time. However, once the tree is constructed, the dynamic updates to the database become relatively inexpensive. The University of Tokyo team also employed SIMD instructions to speed up the `dominates` function, and parallelized the algorithm across four cores.

A second place award was given to the team from the Institute for Research in Fundamental Sciences (IPM), Iran. The IPM team also targeted multicore CPUs, and evaluated their approach on a number of Intel and AMD processors. They reached their highest performance numbers on the AMD Opteron 6386 SE, a 16 core processor. The IPM team's most cost-efficient solution used the 6-core Intel Xeon X5650, which was only 2.5 times slower than their Opteron implementation in spite of its much lower cost. This team also optimized for the dynamic nature of the data, also focusing on efficient insertion and removal of points over time. In order to parallelize the system, this implementation parallelized over time, partitioning timesteps across the cores.

Congratulations to all of our participants. The first and second place teams have contributed invited papers to these proceedings detailing their techniques [12], [13].

VI. CONCLUSION

The 2015 MEMOCODE Design Contest targeted the problem of computing the continuous skyline of dynamic data. Working solutions were submitted by three teams. The winning teams performed effective optimizations involving algorithms, data structures, and parallelism.

Although the submissions to this year's contest focused solely on CPU-based implementation, a number of recent works have explored computing the skyline on GPU and FPGA. For example, [14] shows how the dominance operator can be computed in a branch-free manner to make it suitable for GPU and [15] details a partitioning scheme providing further improvements. Meanwhile, [16] presents a highly scalable parallel FPGA technique for computing the skyline; the authors later generalized this idea into a new computational structure called a shifter list [17].

ACKNOWLEDGMENTS

The 2015 MEMOCODE Design Contest is sponsored by Microsoft and Xilinx. We thank both sponsors for their support. Thanks to all of the contestants for their hard work and we congratulate them on their excellent results. Thank you also to Andreas Gerstlauer, Connie Heitmeyer, Elizabeth Leonard, Yi Deng, and Jean-Pierre Talpin for their help in planning and organizing this contest.

PARTICIPANTS

Thank you to all of the participants. The following teams submitted complete solutions (presented in the same order as Tables II and III):

- Kenichi Koizumi, Mary Inaba, Kei Hiraki
  Graduate School of Information Science and Technology, The University of Tokyo
- Ehsan Montahaie, Milad Ghafouri, Saied Rahmani, Hanie Ghasemi, Farzad Sharif Bakhtiar, Rashid Zamanshoar, Kianoush Jafari, Mohsen Gavahi, Reza Mirzaei, Armin Ahmadzadeh, Saeid Gorgin
  Institute for Research in Fundamental Sciences (IPM), Iran
- Daniele Tajolini

REFERENCES

[1] P. Milder, "MEMOCODE 2014 design contest: k-nearest neighbors with Mahalanobis distance metric," in *ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2014.

[2] E. Nurvitadhi, "MEMOCODE 2013 Hardware/Software Co-design Contest: Stereo Matching," in *IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2013.

[3] S. A. Edwards, "MEMOCODE 2012 Hardware/Software codesign contest: DNA sequence aligner," in *IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2012.

[4] D. Chiou, "MEMOCODE 2011 Hardware/Software CoDesign Contest: NoC simulator," in *IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2011.

[5] M. Pellauer, A. Agarwal, A. Khan, M. C. N. Ng, M. Vijayaraghavan, F. Brewer, and J. Emer, "Design Contest Overview: Combined Architecture for Network Stream Categorization and Intrusion Detection (CANSCID)," in *IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2010.

[6] F. Brewer and J. C. Hoe, "2009 MEMOCODE Co-Design Contest," in *IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2009.

[7] P. Schaumont, K. Asanovic, and J. C. Hoe, "MEMOCODE 2008 Co-Design Contest," in *ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2008.

[8] F. Brewer and J. Hoe, "MEMOCODE 2007 co-design contest," in *IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2007.

[9] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Elsevier, 2011.

[10] S. Börzsöny, D. Kossmann, and K. Stocker, "The skyline operator," in *Data Engineering, 17th International Conference on*, 2001, pp. 421–430.

[11] M. Morse, J. M. Patel, and W. I. Grosky, "Efficient continuous skyline computation," *Information Sciences*, vol. 177, no. 17, pp. 3411–3437, 2007.

[12] K. Koizumi, M. Inaba, and K. Hiraki, "Efficient implementation of continuous skyline computation on a multi-core processor," in *ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2015.

[13] E. Montahaie, M. Ghafouri, S. Rahmani, H. Ghasemi, F. S. Bakhtiar, R. Zamanshoar, K. Jafari, M. Gavahi, R. Mirzaei, A. Ahmadzadeh, and S. Gorgin, "Efficient continuous skyline computation on multi-core processors based on manhattan distance," in *ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2015.

[14] K. S. Bøgh, I. Assent, and M. Magnani, "Efficient GPU-based skyline computation," in *Proceedings of the Ninth International Workshop on Data Management on New Hardware*, 2013, pp. 5:1–5:6.

[15] K. S. Bøgh, S. Chester, and I. Assent, "Work-efficient parallel skyline computation for the GPU," *Proc. VLDB Endow.*, vol. 8, no. 9, pp. 962–973, May 2015.

[16] L. Woods, G. Alonso, and J. Teubner, "Parallel computation of skyline queries," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2013.

[17] L. Woods, G. Alonso, S. Group, and C. Science, "Parallelizing data processing on FPGAs with shifter lists," vol. 8, no. 2, 2015.