Two-Dimensional Closest Pair Algorithms in the VAT-Model

Fabian Dütsch*

Abstract

Recently, Jurkiewicz and Mehlhorn [10] observed that the cost of virtual address translation affects the practical runtime behavior of several fundamental algorithms on modern computers. We extend their results to two dimensions by analyzing and experimentally evaluating algorithms for the closest pair problem regarding the impact of address translation.

1 Introduction

Modern computer systems feature a multi-level memory hierarchy and virtual memory, which cannot be modeled adequately with the RAM-model. Usually, the number of cache misses in the presence of a memory hierarchy is examined in the EM- [1] and CO-model [6]. A machine modeled in these models consists of a slow main memory of unlimited size and a fast cache of size M. Data are moved between these levels in blocks of size B. In contrast to EM-algorithms, cache-oblivious algorithms assume the optimal cache replacement strategy and must not refer to the parameters M and B in the code.

However, these models do not cover the cost of virtual address translation. Running processes access their own linear address spaces via virtual addresses. The operating system transparently maps virtual to physical addresses, typically by walking a root-to-leaf path in a process-specific translation tree. Jurkiewicz and Mehlhorn observed impacts of the cost of virtual address translation on the practical runtime of several fundamental algorithms: the observed runtime of scanning an array in random order, binary searching, and sorting by heapsort exceeds the predicted RAM- and I/O-complexities by a factor of $\mathcal{O}(\log n)$ [10]. As these results cannot be explained by any of the former models, Jurkiewicz and Mehlhorn developed a new model of computation, that considers the cost of virtual address translation: the Virtual Address Translation (VAT) model. Their analyses show that the VAT-complexities match the measured runtime behaviors.

In this note, we revisit the two-dimensional closest pair problem in the VAT-model. We analyze and experimentally evaluate the algorithms by Bentley and Shamos [4], Hinrichs *et al.* [9] and Golin *et al.* [7].

*Department of Computer Science, Westfälische Wilhelms-Universität Münster, Germany, f.duetsch@uni-muenster.de

2 The Model

The *VAT-model* [10] extends the EM-model by modeling virtual addresses and their translation cost. A machine modeled in the VAT-model features two memory levels, both of which are partitioned into pages of size P (corresponding to B). The single processor cannot directly access the main memory, but first has to load the concerned page into the *translation cache* (TC) of the size of W pages (corresponding to M/B blocks). The cost of a cache fault is τ times the cost of a RAM-operation, where τ is some positive machine parameter.

The program accesses the main memory via virtual addresses. An address is a $(d+1)$-tuple in $\{0,\dots,K-1\}^d \times \{0,\dots,P-1\}$; the first part, called the *index*, determines the page and the second part the offset within the page. To translate a virtual address, one has to traverse a root-to-leaf path in a K-ary *translation tree* of height $d := \lceil \log_K(\text{max used page}) \rceil$. Leaves correspond to data pages, i.e., physical pages storing the data. The translation nodes of the *translation path* are determined by the components of the address index and are accessed successively. The internal nodes of size P as well as leaves are stored in the physical main memory and have to be present in the TC when being accessed.

The VAT-complexity is given by the number of cache faults incurred by accesses to data pages and translation nodes. Hence, it may exceed the I/O-complexity by a factor of $\mathcal{O}(d) = \mathcal{O}(\log_K(m/P))$ and the RAM-complexity by a factor of $\mathcal{O}(\tau d)$, with m being the program's memory consumption. The *asymptotic order relations* assume, among others, that following relations hold: [10]

(A1) $1 \le \tau d \le P$, i.e., loading a translation path can be amortized over P RAM-operations.

(A2) $d \le W < m^\theta$, for $\theta \in (0,1)$, i.e., the memory consumption is much bigger than the cache size.

Furthermore, we assume that the root of the translation tree is always present in the TC.

2.1 Previous results

To analyze cache-oblivious algorithms in the VAT-model, Jurkiewicz and Mehlhorn interpreted the translation tree as a $(d+1)$-level memory hierarchy: for $0 \le i \le d$, the translation nodes of height i correspond to blocks of size $K^i P$ and form a memory level.

By uniformly allocating the TC to these memory levels, the following statement results.

Theorem 1 ([10]) *A cache-oblivious algorithm with I/O-complexity $C(M, B, n)$, where M is the size of the cache and B is the size of a block, incurs maximally $\sum_{i=0}^{d-1} C(\lfloor \frac{W}{d} \rfloor K^i P, K^i P, n)$ cache faults in the VAT-model with optimal replacement strategy.*

Thus, linear scanning with I/O-complexity $1 + \lceil \frac{n}{B} \rceil$ causes at most $2d + \frac{K}{K-1}\frac{n}{P}$ faults in the VAT-model. However, this theorem cannot be applied to many cache-oblivious algorithms, as the tall-cache assumption $M \in \Omega(B^2)$ does not hold on each level of the corresponding memory hierarchy. To address this problem, Jurkiewicz *et al.* derived the upper bound $4d \cdot C(\frac{PW}{4}, dP, n)$ for a larger class of algorithms [11]. It implies that I/O-optimal sorting algorithms sort with at most $\mathcal{O}(\frac{n}{P}\lceil \frac{\log n/(PW)}{\log W/d} \rceil)$ faults in the VAT-model. For realistic cache sizes $W \in \Omega(d^2)$, this complexity equals the optimal I/O-complexity. Furthermore, taking into account $\frac{\tau}{P} \overset{(A1)}{\leq} \frac{1}{d}$ shows, that the VAT-cost of sorting is dominated by the RAM-cost $\mathcal{O}(n \log n)$.

3 Preliminaries

We start by analyzing several building blocks used in the closest pair algorithms in the VAT-model.

3.1 Search Data Structures

When searching for a random element, at least one random memory access is necessary. In this case, the best strategy is to cache the upper nodes of the translation tree. We can prove:

Proposition 2 *The average number of cache faults incurred by a search for a random and uniformly distributed element in a data structure storing n elements is at least $\lceil \log_K \frac{n}{P(W+1)} \rceil - 1$.*

This implies that the VAT-complexity of hashing, unlike its RAM- and I/O-complexity, is not constant. At the same time, the amortized number of faults caused by perfect hashing with constant amortized RAM-complexity does not exceed $\Theta(\log_K \frac{n}{PW})$.

The VAT-complexity of accessing and updating a search tree depends on its memory layout and the cache replacement strategy. A search in a balanced binary tree with a random layout incurs $\mathcal{O}(\log \frac{nd}{W} \log_K \frac{n}{PW}) \overset{(A2)}{=} \mathcal{O}(\log \frac{n}{W} \log_K \frac{n}{PW})$ cache faults, if the upper nodes of the search tree and of the translation tree are cached. Searching in a B-tree [2] causes $\mathcal{O}(\log_P \frac{n}{W} \log_K \frac{n}{PW})$ faults. The multi-level blocking regarding blocks of sizes $K^i P$, for $0 \leq i \leq d$, generates a static, cache-aware layout with search cost of $\sum_{i=0}^{d-1} \lceil \log_{K^i P} n \rceil \leq d + \log_P n + \log_K n \ln \log_P n$ cache faults, if searches start with an empty TC. Otherwise, the number of cache faults is $\mathcal{O}(\log_P \frac{n}{W} + \log_K \frac{n}{W} \log \log_P \frac{n}{W})$, if $W \geq 2d$. Applying Theorem 1 to the search in the cache-oblivious van Emde Boas layout [12] results in the same VAT-complexity [10]. Furthermore, the cache-oblivious layout can be dynamized to support searches and updates in the same worst-case complexity (see, e.g., [3]). As the following theorem shows, this is optimal.

Theorem 3 *The average- and worst-case complexities of the number of faults incurred by comparison-based searching is $\Theta(\log_P \frac{n}{W} + \log_K \frac{n}{W} \log \log_P \frac{n}{W})$.*

Proof Sketch. To establish a lower bound, we separately consider a problem with lower I/O-complexity $C(M, B, n)$ on the levels of the translation tree. Interpreting the levels as different levels of a memory hierarchy shows that $\sum_{i=0}^{d} C(WK^i P, K^i P, n)$ is a lower bound for the number of cache faults in the VAT-model. In both cases, plugging in the lower I/O-bound $\Omega(\log_B \frac{n}{M})$ of comparison-based searching results in the claimed lower VAT-complexity. □

The proof carries over to each problem for which an optimal cache-oblivious algorithm whose I/O-complexity does not depend on M is known. In this case, the stated lower bound matches the upper bound from Theorem 1.

In summary, it can be stated that the cost of virtual address translation usually dominates the RAM-complexity of searching.

3.2 Divide and Conquer

In the RAM-model, the complexity of divide-and-conquer algorithms is often determined using the master theorem [5]. However, it is not suitable for the VAT-model, as it may eliminate additional parameters, such as P and W. Additionally, it does not consider that cache faults can be prevented at deep recursive levels in certain cases. We thus adapt the master theorem to the VAT-model.

Theorem 4 *Let $a \in \mathbb{N}$ and $b \in \mathbb{R}$ be constants, with $b > 1$, and let $V \in \mathbb{N}$. Let C, f, and g be asymptotic positive functions. Consider an in-place, divide-and-conquer algorithm incurring $C(W, P, K, d, n) \leq a \cdot C(W, P, K, d, \lceil \frac{n}{b} \rceil) + g(W, P, K, d) \cdot f(n)$ faults. Then, the number of cache faults incurred by an asymptotically optimal replacement strategy on a TC of size $W \geq V \geq 2d + \Omega(d)$ is at most*

$$\begin{cases} \mathcal{O}((g(W,P,K,d) \cdot f(PV) + V)(\frac{n}{PV})^{\log_b a}), \\ \quad \text{if } \exists c > 1 \exists n_0 \in \mathbb{N} \forall n \geq n_0 : a \cdot f(\frac{n}{b}) \geq c \cdot f(n). \\ \mathcal{O}(g(W,P,K,d) \cdot f(n) \cdot \log \frac{n}{PV} + (\frac{n}{PV})^{\log_b a} V), \\ \quad \text{if } \exists k \geq 0 : f(n) \in \Theta(n^{\log_b a} \log^k n). \\ \mathcal{O}(g(W,P,K,d) \cdot f(n)), \\ \quad \text{if } \exists c < 1 \exists n_0 \in \mathbb{N} \forall n \geq n_0 : a \cdot f(\frac{n}{b}) \leq c \cdot f(n). \end{cases}$$

Proof Sketch. $g(W, P, K, d)$ can be factored out, so that the additional parameters are not being eliminated. In cases 1 and 2, cache faults can be prevented from level of recursion $\mathcal{O}(\log_b \frac{n}{PV})$ to the base cases by caching the translation paths to all particular input data in V pages of the TC. As a result, the factors differ from the corresponding factors of the master theorem. Furthermore, accessing the input data of the recursive calls of level $\mathcal{O}(\log_b \frac{n}{PV})$ and deeper incurs at most $\mathcal{O}((\frac{n}{PV})^{\log_b a} V)$ faults. □

The theorem also applies for out-of-place algorithms, if the used memory areas fulfill the *inclusion property*, i.e., if each memory area accessed by a recursive call is a contiguous subset of the corresponding memory area of the particular recursive caller. Details will appear in a full version of this note.

A sequential accesses pattern within a divide-and-conquer algorithm causes up to $\mathcal{O}(a^i d + \frac{m}{P})$ cache faults on the ith level of recursion, where m is the size of the memory area storing the data accessed. The following statement shows that the number decreases to $\mathcal{O}(d + \frac{m}{P})$, if a translation path of a "nearby" page is cached at the beginning of each recursive call.

Proposition 5 *Let $a \in \mathbb{N}$ and S be a sequential subsequence of the accesses of a divide-and-conquer algorithm. Consider the memory areas accessed by S at a fixed level of recursion. If*

- *the recursive calls' memory areas are disjoint, each contiguous, and contained in a contiguous memory area of overall size m,*
- *the translation path of the previous access of S is still present when the particular next access of S within the same recursive call occurs, and*
- *a translation path of any address between the currently accessed page and the first page of the $(a-1)$th previous memory area (ordered by their addresses) is present when the first access of S within any recursive call except for the one corresponding to the first memory area occurs,*

then S maximally incurs $(a+1)d + a\frac{K}{K-1}\frac{m}{P}$ cache faults on that level of recursion.

For all but the first memory accesses within a call of a recursive algorithm, an address of the current memory area may already be present. Hence, if we strengthen the third condition to require the presence of a translation path of an address of the current memory area, the upper bound amounts to $2d + 2\frac{K}{K-1}\frac{m}{P}$.

In summary, it can be stated that divide and conquer is an important design tool for VAT-algorithms.

4 Closest Pair Algorithms

In this section, we investigate the VAT-efficiency for algorithms representing different design paradigms.

4.1 Divide and Conquer

The divide-and-conquer algorithm by Bentley and Shamos [4] divides the point set by the median x-coordinate, recurses on both subsets, merges the subsets by y-coordinates, and, determines the closest pair within a certain vertical stripe around the median x-coordinate. To efficiently compute the median, implementations usually presort the input points. We can implement the out-of-place merging by five sequential scans (three interleaved scans to merge the points in an additional memory area and two interleaved scans to copy them back to the input area). In parallel to copying the points back, the points of the vertical stripe can be extracted to the beginning of the additional memory by interleaved scanning. Finally, the closest pair of points within the stripe can be found with the cost of a single scan, as each point from the stripe has do be compared to maximally seven subsequent points.

By Proposition 5, each of the seven scans incurs $\mathcal{O}(d + \frac{n}{P})$ cache faults on every level of recursion. As both memory areas used obey the inclusion property when being allocated appropriately, the calls at level of recursion $\mathcal{O}(\log \frac{n}{PW})$ and deeper maximally cause $\mathcal{O}((\frac{n}{PW})^{\log_2 2} W) = O(\frac{n}{P})$ faults (Thm. 4(2), $V \in \Theta(W)$). Therefore, the algorithm incurs $\mathcal{O}(\frac{n}{P}\log\frac{n}{PW})$ cache faults, provided that $W \geq 4d + \Omega(d)$. By (A1), their costs amount to $\mathcal{O}(\frac{n}{d}\log\frac{n}{PW}) \subseteq \mathcal{O}(n \log K)$. Consequently, the overall VAT-complexity $\mathcal{O}(n \log n)$ is dominated by the RAM-complexity. The number of cache faults can be decreased further to the translation cost of sorting $\mathcal{O}(\frac{n}{P}\lceil\frac{\log n/(PW)}{\log W/d}\rceil)$ by increasing the branching factor to $\Theta(W/d)$, similar to distribution sweeping [8].

4.2 Plane-Sweep

The plane-sweep algorithm by Hinrichs *et al.* [9] incrementally computes the closest pair by iteratively processing the input points in x-order. It maintains points already considered but still relevant ordered by their y-coordinates in a search tree. When advancing to the next point p, points not relevant anymore are deleted from the search tree, p is inserted, and is then compared to a constant number of adjacent points in the tree.

In the VAT-model, the cost of the initial sorting step is $\mathcal{O}(n \log n)$ [10]. Accessing the respectively current point and the points potentially not relevant anymore within the sorted array overall incurs $\mathcal{O}(d + \frac{n}{P})$ cache faults. Assuming a VAT-optimal search data structure, the insert, delete, and search operations cause $\mathcal{O}(n\log_P\frac{n}{W} + n\log_K\frac{n}{W}\log\log_P\frac{n}{W})$ cache faults. The overall VAT-complexity $\Theta(n\log n + \tau n(\log_P\frac{n}{W} + \log_K\frac{n}{W}\log\log_P\frac{n}{W}))$ is dominated by the translation cost of searching, provided that $\tau \in \Omega(\log K/\log\log_P\frac{n}{W})$.

4.3 Randomized Incremental Construction

The algorithm by Golin *et al.* [7] uses randomized incremental construction. Initially, the points are randomly permuted. Even a naive implementation with RAM-complexity $\Theta(n)$ incurs $\mathcal{O}(n \log_K \frac{n}{PW})$ cache faults, whereas sorting increases the RAM-complexity. Next, the algorithm iteratively inserts the points into a grid of mesh size equal to the distance of the closest pair by then. In this way, a constant number of queries to the grid and, if necessary, a rebuild suffice to process each point. As the probability of a rebuild in the ith iteration amounts to $\mathcal{O}(1/i)$, the expected overall RAM-complexity is $\Theta(n)$, if dynamic perfect hashing is used. In that case, by Proposition 2, the expected number of cache faults amounts to $\Theta(n \log_K \frac{n}{PW})$. Therefore, the translation cost of random permuting and hashing dominate the VAT-complexity.

5 Experimental Evaluation

To evaluate the practical impact of translation cost, we implemented the above closest pair algorithms in C++. The experiments were run on a single core of an Intel Core i5-4210 CPU clocked at 2.7 GHz with 3 MiB cache and 8 GiB main memory running a 64-bit Ubuntu 14.04.2 OS. The code was compiled with g++ 4.8.2 and optimization level -O3. The runtimes were averaged over 100 measurements.

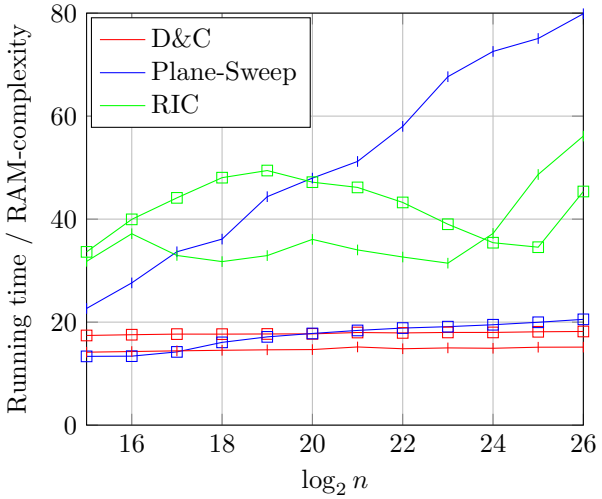

Figure 1: Normalized operation time in logarithmic scale of closest pair algorithms

The above figure illustrates the *normalized operation time*, i.e., the measured running time divided by the RAM-complexity [10]. The algorithms were applied to random point sets distributed in two different ways. The points are uniformly distributed in the unit square (□) and in $\{0\} \times [0, 1)$ (depicted as |) respectively. In both cases, the normalized operation time of the algorithm by Bentley and Shamos is constant. Therefore, the practical runtime behavior matches the RAM-complexity, as predicted by the VAT-model. The implementation of the algorithm by Hinrichs *et al.* uses `std::set`, i.e., a balanced search tree without an optimal memory layout and with overall VAT-complexity $\mathcal{O}(\tau n \log \frac{n}{W} \log_K \frac{n}{PW})$. In case of the first distribution, it is not clear if the normalized operation time is bounded. In the second case, the normalized operation time seems to grow approximately linear in the logarithm of the input size. Thus, the worst-case runtime behavior $\mathcal{O}(n \log^2 n)$ seems to exceed the RAM-complexity and match the VAT-complexity.

We evaluated implementations of the algorithm by Golin *et al.* using hash maps of different libraries and several different hash functions. In all cases, the shape of the graph is just as irregular as the depicted (downscaled) normalized operation time of the implementation using `std::unordered_multimap`. Because of that, the runtime behavior does not entirely conform with the VAT-complexity. It seems, however, to exceed the RAM-complexity. We leave the explanation of this phenomenon as an open problem.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, Sept. 1988.

[2] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1(3):173–189, 1972.

[3] M. A. Bender, R. Cole, and R. Raman. Exponential structures for efficient cache-oblivious algorithms. In *Proc. 29th ICALP*, pages 195–207, 2002.

[4] J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proc. 8th ACM STOC*, pages 220–230, 1976.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2009.

[6] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th FOCS*, pages 285–298, 1999.

[7] M. Golin, R. Raman, C. Schwarz, and M. Smid. Simple randomized algorithms for closest pair problems. *Nordic Journal of Computing*, 2(1):3–27, Mar. 1995.

[8] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proc. 34th FOCS*, pages 714–723, 1993.

[9] K. Hinrichs, J. Nievergelt, and P. Schorn. Plane-sweep solves the closest pair problem elegantly. *Information Processing Letters*, 26(5):255–261, Jan. 1988.

[10] T. Jurkiewicz and K. Mehlhorn. The cost of address translation. In *Proc. ALENEX*, pages 148–162, 2013.

[11] T. Jurkiewicz, K. Mehlhorn, and P. K. Nicholson. Cache-oblivious VAT-algorithms. *Computing Research Repository*, abs/1404.3577, 2014.

[12] H. Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, June 1999.