The Minimum Reservation Rate Problem in Digital Audio/Video Systems*

Dave Anderson † Nimrod Megiddo ‡ Moni Naor §

Abstract

The "Minimum Reservation Rate Problem" arises in distributed systems for handling digital audio and video data. The problem is to find the minimum rate at which data must be reserved on a shared storage system in order to provide continuous buffered playback of a variable-rate output schedule. The problem is equivalent to the minimum output rate: given input rates during various time periods, find the minimum output rate under which the buffer never overflows.

We present for these problems an $O(n \log n)$ expected time probabilistic algorithm and a deterministic one with complexity $O(n \log n \log \log n)$. The algorithms are obtained by applying a parametric search method due to Megiddo [6] that utilizes a parallel algorithm for evaluating a function in order to obtain an efficient sequential algorithm for solving an parametric equation of the function.

---

*Preliminary version appeared in Proc. 2nd Israeli Symp. on Theory of Computing and Systems, 1993, pp. 268–278.

†Sonic Solutions, 1891 East Fransc. Blvd., San Rafael CA 94901 USA. Work done while at the International Computer Science Institute, Berkeley.

‡IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel. Research supported in part by ONR contract N00014-91-C-0026. E-mail: megiddo@almaden.ibm.com.

§Incumbent of the Morris and Rose Goldman Career Development Chair, Dept. of Applied Mathematics and Computer Science, Weizmann Institute, Rehovot 76100, Israel. E-mail: naor@wisdom.weizmann.ac.il. Part of this work was done while at the IBM Almaden Research Center. Research supported by an Alon Fellowship and by a grant from the Israel Science Foundation administered by the Israel Academy of Sciences.

1 Introduction

In this paper we consider the *minimum reservation rate* problem which arises in distributed systems for handling digital audio and video data. This problem is formulated as follows. Given are consecutive *time intervals* $T_1, T_2, \ldots, T_n$ and *output rates* $O_1, O_2, \ldots, O_n$ (where O_i is the output rate during the ith time interval, $i = 1, \ldots, n$). Find the minimum rate R^* at which input can be reserved on a shared storage system such that the output flows continuously (with no "starvation").

The *minimum output rate* problem is very similar. Given are consecutive *time intervals* $T_1, T_2, \ldots, T_n$ and *input rates* $I_1, I_2, \ldots, I_n$ (where I_i is the input rate during the ith time interval, $i = 1, \ldots, n$), and a *buffer size* B. Find the minimum output rate R^* required to assure that the buffer never overflows.

We postpone explaining how the problem arises in digital audio/video systems to Section 6. Section 2 contains a precise definition of the problem and shows that the problem is equivalent to finding the minimum of some $O(n^2)$ values (thus providing a simple $O(n^2)$ algorithm). In Sections 3 and 4 we develop an $O(n \log n)$ randomized algorithm and an $O(n \log n \log\log n)$ deterministic one. These algorithms are based on the parametric search method of Megiddo [6]. Using this method, we apply a parallel algorithm for one problem (in this case, the "evaluation" problem) to obtain a fast sequential algorithm for another "parametric" problem. In Section 3 we describe an expected constant-time randomized parallel algorithm for the feasibility problem. In Section 4 we explain how to use this algorithm together with a sequential linear-time algorithm for feasibility, so as to solve the minimum rate problem efficiently. In Section 5 we describe an extension to the minimum rate problem where the input is cyclic. In Section 6 we discuss the background from digital audio/video systems of the minimum reservation rate problem. Finally, Section 7 proposes further research.

In order to make the paper more self-contained we now outline the essential steps of the parametric search method used in order to solve an optimization problem:

1. Find a monotone function F so that you can express the optimization problem as "find λ such that $F(\lambda) = 0$".

2. Provide a fast parallel algorithm $\mathcal{A}$ for evaluating $F(\lambda)$. The algorithm should be comparison based. Let P be the number of processors and T_A the run time of $\mathcal{A}$.

3. Provide a fast sequential algorithm $\mathcal{B}$ that enables determining whether a given λ' is smaller or larger than λ^* (where λ^* is the solution to $F(\lambda) = 0$). Let T_B be its run-time.

4. Show a way of simulating algorithm $\mathcal{A}$'s operations when executed on the optimal λ^*: at every round $\mathcal{A}$ makes several comparisons depending on the outcome of the previous rounds. Each such comparison (W_i, W_j) should have a *critical value* (or breakpoint) λ_{ij} so that $\lambda^* < \lambda_{ij}$ implies $W_i < W_j$ way and $\lambda^* > \lambda_{ij}$ implies $W_i > W_j$. A simulation of a round consists therefore of:

    (a) Find all the critical values of the comparisons of the round.

    (b) Find λ_m, the median of the critical values.

(c) Determine using algorithm $\mathcal{B}$ whether λ_m is smaller or larger than λ^*. You can now deduce for half the critical values their relationship with λ^*.

(d) Repeat steps 4b and 4c until $\mathcal{A}$'s decisions on all the comparisons at λ^* are known.

5. The result of the simulation is a segment $[\lambda_1, \lambda_2]$ such that for any $\lambda \in [\lambda_1, \lambda_2]$ algorithm $\mathcal{A}$ behaves as on λ^*. Find a way to extract λ^* from λ_1 and λ_2 (often $\lambda_1 = \lambda_2 = \lambda^*$).

The complexity of an algorithm designed by such a method is $O(T_A(P + T_B \log P))$ (assuming the bookkeeping can be done efficiently).

2 Preliminaries

Henceforth we consider the following problem:

Problem 2.1 (" Minimum Rate"): Given consecutive *time intervals* $T_1, T_2, \ldots, T_n$ and *input rates* $I_1, I_2, \ldots, I_n$ (where I_i is the input rate during the ith time interval, $i = 1, \ldots, n$), and a *buffer size* B. Find the minimum output rate R^* required to assure that the buffer does not overflow.

Lemma 2.2 *Let $a_i \equiv T_i I_i$ denote the total amount of data received during the ith interval ($i = 1, \ldots, n$). Then,*

$$R^* = \max_{1 \le i \le j \le n} \frac{a_i + \cdots + a_j - B}{T_i + \cdots + T_j} \ . \tag{1}$$

Proof: Let R be any feasible output rate, *i.e.*, the buffer never overflows when the output rate is R. For any pair (i, j) $(1 \le i \le j \le n)$, the total amount of data received during the intervals $T_i, \ldots, T_j$ is $a_i + \cdots + a_j$, whereas the total amount output during these intervals does not exceed $(T_i + \cdots + T_j)R$. It follows that

$$(a_i + \cdots + a_j) - (T_i + \cdots + T_j)R \ \le \ B$$

or, equivalently,

$$R \ \ge \ r_{ij} \equiv \frac{a_i + \cdots + a_j - B}{T_i + \cdots + T_j} \ .$$

Obviously, when the output rate is the minimum feasible one, R^*, the buffer must be full at least once. Moreover, during any interval, the amount in the buffer reaches its maximum level at one of the end points of the interval. It follows that the buffer must be full at the end of some interval T_j. Now, since the buffer is empty at the beginning of T_1, there exists a last time before the end of T_j when the buffer is empty. It is easy to see that the buffer is empty during closed time intervals, where the right end point of any such interval must coincide with an end point of one of the intervals T_k $(k = 1, \ldots, n)$. It follows that the last time before the end of T_j such that the buffer is empty must coincide with the starting point of an interval T_i $(1 \le i \le j)$. During the intervals $T_i \ldots, T_j$, the buffer is not empty, the output rate is R^*, and hence

$$(a_i + \cdots + a_j) - (T_i + \cdots + T_j)R^* \ = \ B \ .$$

This implies our claim. ∎

Corollary 2.3 *The minimum output rate R^* (see Problem 2.1) can be computed in $O(n^2)$ time.*

3 Feasibility of output rate

In this section we present algorithms for deciding feasibility of a given output rate. These algorithms turn out to be useful in the design of algorithms for the output rate minimization problem.

Denote

$$F(\lambda) = \max_{1 \le i \le j \le n} (a_i + \cdots + a_j) - (T_i + \cdots + T_j)\lambda \ .$$

Obviously, $F(\cdot)$ is strictly monotone decreasing, piecewise linear, and convex. Thus, there exists a unique λ^* such that $F(\lambda^*) = B$. It is easy to see that $\lambda^* = R^*$ (see Megiddo [4]).

In order to compute λ^*, we will follow the parametric search method with the use of parallel algorithms as proposed in Megiddo [6] and described in the Introduction above. To this end, we develop two algorithms for computing the value of $F(\lambda)$ at any given λ:

1. A sequential linear-time algorithm (this corresponds to $\mathcal{B}$ above).

2. A parallel constant-time randomized algorithm employing $n \log n$ processors under the comparisons model of Valiant [9] (this corresponds to $\mathcal{A}$ above). The algorithm should be such that deciding at every round which comparison should be made can be done in $O(n \log n)$ (sequential) time. A deterministic $O(\log \log n)$ algorithm employing $n \log n$ processors will also be suggested.

The problem of evaluating $F(\lambda)$ can rephrased as follows. Given λ, let $W_0 = 0$ and

$$W_i = \sum_{k=1}^{i} (a_k - T_k \lambda) \qquad 1 \le i \le n \ .$$

Problem 3.1 Given n real numbers $W_1, \ldots, W_n$, find a pair (i, j) $(0 \le i < j \le n)$ so as to maximize $W_j - W_i$.

Proposition 3.2 *Problem 3.1 can be solved in linear time.*

Proof: This can be done using a simple dynamic programming algorithm: For $k = 1, \ldots, n$, denote

$$V_k = \max\{W_j - W_i \mid 0 \le i < j \le k\}$$
$$m_k = \min\{W_i \mid 0 \le i \le k\} \ .$$

Obviously, $V_1 = W_1$ and $m_1 = \min\{0, W_1\}$. Furthermore, for $k = 1, \ldots, n-1$,

$$V_{k+1} = \max\{V_k, W_{k+1} - m_k\}$$
$$m_{k+1} = \min\{m_k, W_{k+1}\} \ .$$

Thus, the quantity we are interested in, V_n, can be found in $O(n)$ time. ∎

Proposition 3.3 *Problem 3.1 has a probabilistic expected constant time algorithm using* $n \log n$ *processors in parallel and an* $O(\log \log n)$ *deterministic algorithm using* $n \log n$ *processors in parallel (both under the comparison model). Deciding which comparisons to make can be done in* $O(n \log n)$ *(sequential) time.*

Proof: We assume, without loss of generality, that $n+1$ is a power of 2. For every $(1 \leq k \leq \log_2(n+1) - 1)$ and $1 \leq \ell \leq \frac{n+1}{2^{-k}} - 1$, let

$$M_{k\ell} = \max\{W_i \mid \ell 2^k \leq i < (\ell+1)2^k\}$$
$$m_{k\ell} = \min\{W_i \mid (\ell-1)2^k < i \leq \ell 2^k\} \ .$$

It is easy to verify that

$$\begin{aligned} & \max_{0 \leq i < j \leq n} \{W_j - W_i\} \\ = \quad & \max\{M_{k\ell} - m_{k\ell} \mid k = 1, \ldots, \log_2(n+1) - 1,\ \ell = 1, \ldots, (n+1)2^{-k} - 1\} \ . \end{aligned}$$

We first describe the probabilistic algorithm. It is now well known that the maximum (or minimum) of m elements can be found by m processors in expected constant time[1]. Furthermore, there is a constant time m processor parallel algorithm such that the probability of failure, *i.e.*, that the maximum is not found, is at most $\frac{1}{m}$ (it can actually be made to be smaller than e^{-m^c} for some $c > 0$). For every k and ℓ as above, we allocate 2^k processors to the problem of computing $M_{k\ell}$ and $m_{k\ell}$ using the constant time and linear number of processors maximum finding algorithm.The total number of processors is

$$\sum_{k=1}^{\log(n+1)-1} 2^k((n+1)2^{-k} - 1) = O(n \log n) \ .$$

After running the maximum finding algorithm in parallel on all these problems, some of the $M_{k\ell}$'s and $m_{k\ell}$'s may still not be known. We repeat this procedure (only for the unknown $M_{k\ell}$'s and $m_{k\ell}$'s) until

$$\sum_{k=1}^{\log(n+1)} 2^{2k} \cdot |\{\ell \mid \text{such that either } M_{k\ell} \text{ or } m_{k\ell} \text{ is unknown}\}| \leq n \log n.$$

When the latter is satisfied, we can allocate 2^{2k} processors to each unknown $M_{k\ell}$ or $m_{k\ell}$, a number that suffices to find them in one step (by allocating 2^{2k} processors to the respective problem). We argue that the expected number of times we have to repeat running the maximum finding algorithm is constant. To see this, note that since the probability of failure on $M_{k\ell}$ (or $m_{k\ell}$) is less than 2^{-k}, it follows that the expected value

$$E = \mathcal{E}[\sum_{k=1}^{\log(n+1)} 2^{2k} \cdot |\{\ell \mid \text{ such that } \text{ either } M_{k\ell} \text{ or } m_{k\ell} \text{ is unknown}\}|]$$

[1]The algorithm proceeds roughly as follows: a random samples of $\sqrt{m}$ elements is chosen; the maximum of the sample is found exhaustively; this maximum is compared to the remaining elements and only those larger are kept. With high probability the number of elements larger than the sample's maximum is not much greater than $\sqrt{m}$. Repeat this procedure until you have a set of $\sqrt{m}$ candidates, from which a maximum can be found exhaustively. See Reischuk [8] and Megiddo [5] for more details.

satisfies

$$E \leq \sum_{k,\ell} 2^{-k}2^{2k} \leq \sum_{k=1}^{\log(n+1)} \frac{n+1}{2^k} \cdot 2^{-k}2^{2k} = \sum_{k=1}^{\log(n+1)} (n+1)$$

which is $O(n \log n)$. This implies our claim. Finally, we compute the maximum of the differences $M_{k\ell} - m_{k\ell}$ in expected constant time.

The deterministic algorithm is similar, except that the for computing $M_{k\ell}$ and $m_{k\ell}$ we use Valiant's [9] m processor $\log\log m$ time algorithm for maximum finding in the comparison model.

Note that in both the randomized and the deterministic algorithms deciding which comparison to make is straightforward and can be done in linear (in the number of comparisons) time. ∎

We note that by Valiant's lower bound, there is no constant time deterministic algorithm for evaluating $F(\lambda)$ which employs substantially fewer than n^2 processors.

4 Finding the minimum rate

As outline in the Introduction, the algorithm for finding λ^* proceeds by simulating the parallel algorithm for evaluating $F(\lambda)$ at $\lambda = \lambda^*$ (without knowing λ^* in advance).

Proposition 4.1 *The minimum rate problem can be solved by a randomized algorithm in expected $O(n \log n)$ time and by a deterministic algorithm in $O(n \log n \log\log n)$ time.*

Proof: The parallel algorithm makes comparisons of the form: given $i < j$, is $W_i < W_j$? Here, W_i and W_j are linear functions of λ. Thus, there is a breakpoint λ_{ij} such that $\lambda^* > \lambda_{ij}$ if and only if $W_i < W_j$, namely,

$$\lambda_{ij} = \frac{\sum_{k=i+1}^{j} a_k}{\sum_{k=i+1}^{j} T_k} .$$

Note that once we have computed (in linear time) all the prefix sum $\sum_{k=1}^{i} a_k$ and $\sum_{k=1}^{i} T_k$, we can compute λ_{ij} for any given i and j in constant time.

In order to simulate one step of the parallel algorithm for evaluating $F(\lambda^*)$, we need to recognize for each of the $n \log n$ breakpoints λ_{ij} whether it is less than or greater than λ^*; these $O(n \log n)$ breakpoints are produced by the processors as they attempt to perform the comparisons they are responsible for in the simulated parallel algorithm. This task can be accomplished in $O(n \log n)$ time as follows. We first find the median λ' of the set of these breakpoints using the linear-time median-finding algorithm [2][2]. Next, we check (using the algorithm of Proposition 3.2) whether $\lambda' < \lambda^*$ (*i.e.*, whether $F(\lambda') > B$). Now we know for half the breakpoints their positions relative to λ^*. We continue by finding the median of the other half of the set of breakpoints, and so on. Altogether, there will be $O(\log n)$ calls to the algorithm for evaluating $F(\lambda)$, so the total time is $O(n \log n)$. The expected number of steps of the probabilistic parallel algorithm is constant. After

[2]In fact an approximate one would do. and in any case we can use the probabilistic algorithm of Floyd and Rivest [3].

simulating these steps at a cost of $O(n \log n)$, we know a pair i, j that maximizes $W_j - W_i$ at λ^*. It follows that

$$\lambda^* = \frac{\sum_{k=i+1}^{j} a_k - B}{\sum_{k=i+1}^{j} T_k} .$$

In the deterministic case we need to simulate $O(\log \log n)$ steps (corresponding to the steps in the deterministic maximum finding algorithm of Valiant [9]), and the rest is essentially the same. The result is an $O(n \log n \log \log n)$ algorithm. ∎

5 Extensions

We now describe two extensions of the problem discussed above. The first is when the buffer is not empty initially, but has an amount of $a_0 \leq B$. Based on our previous analysis, it is easy to see that the value of the optimal rate is obtained by substituting in (1) $a_0 + a_1$ for a_1.

The second extension is when the problem is cyclic, *i.e.*, we have the time intervals and $T_1, T_2, \ldots, T_n$ and the input rates $I_1, I_2, \ldots I_n$ repeating ad infinitum. In this case, for any feasible R we have

$$R \geq \frac{a_1 + a_2 + \cdots + a_n}{T_1 + \cdots + T_n} \tag{2}$$

since, otherwise, the buffer overflows eventually. Also, from Lemma 2.2 we have that for any feasible R, for all (i, j) $(1 \leq i, j \leq n)$, and any $k \geq 0$,

$$R \geq \frac{a_i + \cdots + a_n + k(a_1 + \cdots + a_n) + a_1 + \cdots + a_j - B}{T_i + \cdots + T_n + k(T_1 + \cdots + T_n) + T_1 + \cdots + T_j} .$$

However, this inequality is implied by

$$R \geq \frac{a_i + \cdots + a_n + a_1 + \cdots + a_j - B}{T_i + \cdots + T_n + T_1 + \cdots + T_j} .$$

and (2). Therefore, the optimal R^* for the cyclic problem is the maximum between (2) and the solution to the (non-cyclic) problem with time intervals $T_1, \ldots T_n, T_1, \ldots T_n$, and input rates $I_1, \ldots I_n, I_1, \ldots I_n$.

6 Background

A "digital audio editing system" allows users to record sound (music, dialog, special effects, etc.) on magnetic disks (see [7]). The audio encoding has a constant data rate, typically 88,200 bytes per second for each audio channel. Users can then assemble segments of these sound files into an "Edit Decision List" (EDL). An EDL has one or more "output channels," each consisting of sound segments arranged according to time. The segments in an output channel may overlap arbitrarily. The EDL can also specify "fade" functions that attenuate the volume of sound segments near their end points, so that they blend together smoothly.

The characteristics of EDL's vary over different types of usage, such as music production and dialogue editing. The number of segments in an output channel may vary from one to several thousands. The segments may be disjoint or overlapping. For example, the EDL for a movie soundtrack might have six output channels, with each channel containing a mixture of music, speech, and ambient sound.

Sound editing (*i.e.*, creating and modifying EDLs) is "non-destructive" in the sense that no sound files are created or modified during editing. To play an EDL, the system must read audio data from a disk in real time, "mix" overlapping segments together, apply the fade functions as needed, and send the result of each output channel to a different physical output device. This task typically requires the use of one or more digital signal processing (DSP) chips. Audio data is read from the disk in blocks (typically 64K bytes) and stored in memory buffers near the DSP's. Each DSP executes a loop during which it reads a sample for each active segment, performs the necessary scaling and mixing, and sends the results to the output interface. From there the sound goes to its final destination, *e.g.*, a compact-disk writer, a loudspeaker, or a radio transmitter.

Advances in technology allow streams of digital audio data to be sent through networks in real time. These advances have led to the development of distributed digital audio editing systems. In the most common configuration many users share common disks which they access through the network.

The hardware components of an editing system (*e.g.*, disks, networks, DSP's) have bounded performance. If the workload (*e.g.*, the number of overlapping segments in an EDL) is too high, then "dropouts" (*i.e.*, missing or incorrect output values) will occur. Dropouts must be avoided at all cost. Hence the system must use a scheme in which EDL's can be played only after the needed capacity on each hardware component has been "reserved." In a multi-user (distributed) system, this scheme may prevent one user from playing an EDL while another user is playing one. In a single-user system, the problem reduces to the question of whether or not an EDL can be played using all available hardware resources.

We now list our assumptions regarding the system:

1. Denote by $b(t)$ the amount of "data" held in the *buffer* at time t. Assuming the buffer size is B, the amount $b(t)$ varies continuously between 0 and B.

2. The data storage system (disk plus network) provides constant-rate data streams. Clients of the system can make reservation requests for fixed rates R; if a request is granted, the system fills the buffer at rate R, except when the buffer is full.

3. A given EDL defines a piecewise-constant "output rate function" $f(t)$ (*i.e.*, the rate at which the buffer is drained at time t) whose value is the number of audio segments active at time t times the audio data rate.

It follows that the function $b(t)$ is piecewise with right derivative b' such that $b'(t) = 0$ if $b(t) = B$ and $R > f(t)$, and $b'(t) = R - f(t)$ otherwise.

We say that *underflow* occurs if $b(t) < 0$ for some t. We say that a rate R is *admissible* if underflow never occurs. The following problems present themselves:

1. *Minimum Reservation Rate*: Assuming $b(0) = B$ (*i.e.*, the buffer is initially full), find the least admissible rate R^*.

2. *Minimum Buffer Pre-Load*: Given a value R that is admissible with $b(0) = B$, find the smallest value b^* such that R is admissible with $b(0) = b^*$.

The minimum-rate problem is interesting because with a solution to it we can maximize the number of EDL's that can be played concurrently. The minimum pre-load problem is interesting because the response time for EDL plays (*i.e.*, the time from the user clicking a "play" button to the beginning of audio output) should be minimized during interactive editing. For more details on these issues, see [1].

It is easy to see that a solution to Problem 2.1 is also a solution to the minimum reservation rate problem described above.

7 Conclusion and further research

The Minimum Reservation Rate problem and the Minimum Buffer Pre-Load problem are practical and important ones in the design of distributed systems for digital audio and video. We do not know whether $O(n \log n)$ is the optimal time. In particular, is it possible to prove an $\Omega(n \log n)$ lower bound in the comparison tree model? Alternatively, is it possible to improve the running time. Using the approach we took here, this requires (i) an n processors constant time algorithm for evaluating $F(\lambda)$ (ii) a way to amortize the sequential computation of $F(\lambda)$, so that $\log n$ calls would still require only linear time.

A different approach is to treat the problem as a linear program with n variables and $O(n)$ constraints: Let x_i be a variable corresponding to the amount output at interval T_i. Then R^* is the solution to the following:

$$\min R$$

s.t. $\forall 1 \leq k \leq n$

$$0 \leq a_1 + \cdots a_k - x_1 - \cdots x_k \leq B \quad \text{and} \quad x_k \leq T_i \cdot R$$

Can a linear program of this form be solved more quickly than $O(n \log n)$?

Acknowledgments

We thank Ed Reiss for bringing us together and Oded Goldreich for useful suggestions.

References

[1] D. P. Anderson, "Meta-scheduling for continuous media," *ACM Transactions on Computing Systems* **11**, (1993), 226–252.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, "Time bounds for selection," *J. Computer and System Sciences* **7** (1972) 451–455.

[3] R. W. Floyd and R. Rivest, "Expected Time Bounds for Selection", *C. ACM*, (1975).

[4] N. Megiddo, "Combinatorial optimization with rational objective functions," *Mathematics of Operations Research* **4** (1979) 414–424.

[5] N. Megiddo, "Parallel algorithms for finding the maximum and the median almost surely in constant-time," Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, October 1982.

[6] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *J. ACM* **30** (1983) 852–865.

[7] J. A. Moorer, "Hard-disk recording and editing of digital audio," in: *Proceedings of the 89th Convention of the Audio Engineering Society*, Los Angeles, 1990.

[8] R. Reischuk, "A fast probabilistic parallel sorting algorithm," in: *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Angeles, 1981, pp. 212–219.

[9] L. G. Valiant, "Parallelism in comparison problems," *SIAM J. Comput.* **4** (1975) 348–355.