On Limits on the Computational Power of Data-Accumulating Algorithms*

Stefan D. Bruda and Selim G. Akl
Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6
Email: {bruda,akl}@qucis.queensu.ca

March 28, 2004

Abstract

In the data-accumulating paradigm, inputs arrive continuously in real time, and the computation terminates when all the already received data are processed before another datum arrives. Previous research states that a constant upper bound on the running time of a successful algorithm within this paradigm exists only for particular forms of the data arrival law. This contradicts our recent conjecture that those problems that are solvable in real time are included in the class of logarithmic space-bounded computations. However, we prove that such an upper bound does exist in fact in both the parallel and sequential cases and for any polynomial arrival law, thus strengthening the mentioned conjecture. Then, we analyze an example of a non-continuous data arrival law. We find similar properties for the sorting algorithm under such a law, namely the existence of an upper bound on the running time, suggesting that such properties do not depend on the form of the arrival law.

Keywords: computational complexity, data-accumulating algorithms, sorting, parallel algorithms.

1 Introduction

In the classical study of algorithms, input data are considered to be available at the beginning of computation. The algorithm eventually terminates when the input data have been processed. What happens though when the data arrive *while* the computation is in progress? A paradigm considering such computations is the *data-accumulating paradigm* [4, 5, 6]. Here, data are considered as being an endless stream. An algorithm terminates when all the data that were already received have been processed.

In this paper we investigate the computational power of data-accumulating algorithms. As a case study we consider sorting algorithms, which are basic elements for various applications. It has been shown [5] that, if the data arrive *fast enough*, then a successful algorithm (i.e., one that terminates) must have a running time upper bounded by a constant;

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

when the running time exceeds that constant, the algorithm never terminates. Our results indicate that the qualifier "fast enough" is not necessary, and a constant upper bound for the running time exists for *any* polynomial data arrival law, and for any (parallel of sequential) d-algorithm. This is a negative, yet important result as it establishes a limit that was not previously known. Such a limit, however, is consistent with the conjectured inclusion of real-time problems in the class of logarithmic space-bounded computations (NLOGSPACE) [3].

Subsequently, we extend these results by considering another data arrival law which is significantly different from the polynomial one, being a non-continuous function. We find similar properties for the sorting algorithm, namely, the existence of an upper bound on the running time. Therefore, we conjecture that these properties are not dependent on the expression of the arrival law.

2 The Data-Accumulating Paradigm

In the data-accumulating paradigm [4, 5], input data arrive as time passes, conforming to a *data arrival law*. The computation terminates when all the data currently arrived have been processed. Algorithms pertaining to this class are called *data-accumulating algorithms* or, for short, *d-algorithms*. The form proposed in [5] for the data arrival law is

$$f(n,t) = n + kn^{\gamma}t^{\beta}, \tag{1}$$

where n is the size of the initial data set, and k, γ, and β are positive constants. Such a form of the arrival law is rather flexible, and its polynomial form eases the reasoning about algorithms that use it. In particular, note that, when $\gamma = 0$, the amount of data that arrive in one time unit is independent of the size of the initial data set. If $\beta = 1$, then the data flow is constant during the time, while in the case $\beta > 1$ the flow of data actually increases with time. Similarly, when $\beta < 1$, fewer and fewer data arrive as time increases.

A standard algorithm, working on a non-varying set of data, is referred to as a *static* algorithm. Consider a given problem Π. Let the best known static algorithm for Π be A'. Then, a d-algorithm A for Π working on a varying set of data of size N is *optimal* if and only if its running time $T(N)$ is asymptotically equal to the time $T'(N)$, where $T'(N)$ is the time required by A' working on the N data as if they were available at time $t = 0$.

The problems considered in [5] are solvable in polynomial time, i.e., $T'(N) = O(N^{\alpha})$, where α is a positive constant, with T' defined as above. This implies that an optimal d-algorithm must have a time complexity of $T(N) = cN^{\alpha}$, for some positive constant c[1]. If the d-algorithm terminates at time t, then we have $t = cN^{\alpha}$. Considering the data arrival law given by Relation (1), the termination time t can be expressed by the implicit function

[1]Note, however, that our study in the present paper covers d-algorithms that have a time complexity of cN^{α}, even if they are not optimal.

$$t = c(n + kn^{\gamma}t^{\beta})^{\alpha}. \tag{2}$$

It is shown in [5] that the d-algorithm will terminate for n in some interval that depends on the constants involved only for certain values of α, β, and γ. For example, it is claimed that, if $\gamma = 1$, for $\alpha\beta < 1$, then n grows as $(1/(kc^{1/\alpha}))t^{(1-\alpha\beta)/\alpha}$. That is, for any n, the d-algorithm will eventually terminate, even if its running time can be very large. For $\alpha\beta \geq 1$, on the other hand, the algorithm terminates only for bounded values of n; moreover, for $\alpha\beta > 1$, the running time itself is upper bounded by a constant (this bound not being present in the other cases).

The parallel d-algorithm is defined in the same manner as the sequential one, except that its execution time $T_p(N)$ is compared to the execution time $T'_p(N)$ of a parallel static algorithm that uses the same number of processors $P(N)$ as the parallel d-algorithm. The equation for the parallel optimal d-algorithm is similar to the one obtained for the sequential case:

$$t_p = \frac{c_p(n + kn^{\gamma}t_p^{\beta})^{\alpha}}{P}, \tag{3}$$

where the subscript p denotes the parallel case and P is the number of processors. It is assumed in [5] that P is a polynomial function of N.

We quote the following main results obtained in [5]:

Proposition 2.1 *Given a problem admitting an optimal sequential d-algorithm obeying Relation (2) and an optimal parallel d-algorithm obeying Relation (3), and for $c_p/P < c$, $\alpha\beta = 1$, $\gamma = 1$, and $P = \xi(n + kn^{\gamma}t_p^{\beta})^{\delta}$, with some constants ξ, $\xi > 0$, and δ, $0 \leq \delta \leq \alpha$, it holds that $\lim_{n \to 1/kc^{1/\alpha}} t/Pt_p = \infty$. In addition, $t/Pt_p > c/c_p$ for all values of α, β, γ.*

This result shows that the quantity t/Pt_p (and thus the speedup of the parallel algorithm) may become arbitrarily large. This implies that the normal bound for parallel speedup [2, 7] no longer holds.

3 Sorting

We now study sorting d-algorithms. Note that a similar problem is investigated in [5], but there the result is a search tree. Conceivably, there exist applications working in real time on large sequences of data for which the $O(\log n)$ access time to the elements in a search tree is not acceptable. Therefore, the sorting algorithms discussed here output an *array* of sorted elements (the access time is thus $O(1)$). Henceforth, we refer to such processing as *sorting on a linear structure* (or simply *sorting* when there is no ambiguity).

For comparison, we use an optimal static sorting algorithm whose running time is $\Theta(n \log n)$ [8].

3.1 A First Upper Bound on the Running Time

By definition, data accumulates as the computation proceeds. Generally, we consider that the incoming data are buffered until some amount q is reached, and then the buffered data are inserted into the already sorted sequence (for the non-buffered case simply set q to 1). The time complexity of such an operation is given by the following lemma.

Lemma 3.1 *Let the length of the already sorted sequence be l. Then, the time complexity of inserting q new elements into the sorted sequence is $\Theta(q \log q + l)$, for any q and l such that either $q = 1$ or $l \geq (q \log q)/(q-1)$.*

Proof. The upper bound is immediate. For the lower bound, we have the following: Supposing that the distance between the insertion point of some element and the end of the buffer is $\Omega(l)$ (the general case), the time required for inserting a datum into some sorted sequence is $\Omega(l)$. If the newly arrived data are not sorted, then one should insert them into the buffer one by one. Therefore, the insertion time for the whole sequence is $\Omega(ql)$. On the other hand, the number of operations needed to sort q data is $\Omega(q \log q)$ [8], while merging the sorted sequences requires $\Omega(q + l)$ operations [8]. $\square$

We did not consider the case in which more than q elements come before the processing of the already arrived data is finished. However, this case is equivalent to the one in which the data arrive too fast and the d-algorithm never stops. Indeed, as seen below, the necessary condition (4) for the algorithm to terminate in finite time covers this case.

Let t_q be the time in which a buffer of size q is filled. At some time t_x, when the buffer is filled and is ready to be inserted, the length of the already sorted sequence will be $l = n + kn^\gamma(t_x^\beta - t_q^\beta)$, because all the arrived data have been inserted, except the data which are in the buffer (otherwise, some elements are lost). We will consider for simplicity that $\beta = 1$, but similar results can be obtained for other values as well. In this context, the time interval between two arrivals is $dt = 1/kn^\gamma$. Suppose that the algorithm stops at time t. This means that, at time t, all the buffered data have been inserted before another datum arrives. Thus, the time required to insert the buffer (given by Lemma 3.1) should be no larger than dt, i.e.,

$$dt \geq q \log q + n + kn^\gamma(t - t_q). \tag{4}$$

Considering the data arrival law of Relation (1), the time t_q in which the buffer is filled is given by $q = kn^\gamma t_q^\beta$. Then, simple calculations let us derive the following bound on the computation time:

$$t \leq \frac{1}{(kn^\gamma)^2} + \frac{1}{kn^\gamma} q(1 - \log q) - \frac{n}{kn^\gamma}. \tag{5}$$

This imposes a limit on the running time of any sequential sorting d-algorithm that terminates for the polynomial data arrival law given by Relation (1). Henceforth, the right-hand side of Relation (5) will be denoted by t'_B[2]. We are now ready to determine the complexity of sorting on a linear structure.

Theorem 3.2 *Under the polynomial data arrival law given by Relation (1), the running time of any sequential d-algorithm for sorting on a linear structure is $\Theta(N^2)$. Thus, sorting on a linear structure does not admit an optimal d-algorithm.*

Proof. Assume first that q is constant with respect to the running time (however, if it is a function of n, then the proof is not affected). By Lemma 3.1, the time required to insert the q elements from the buffer is $\Theta(q \log q + l)$, where l is the number of already sorted elements. This time is $\Theta(l)$, since q is constant. Thus, the total time required for sorting N elements is $\Theta(\sum_{l=n}^{N/q} ql) = \Theta(N^2)$, as desired, since l increases by q each time a new buffer is inserted.

Assume now that the size q of the buffer varies with time. We have $\frac{\partial t'_B}{\partial t} = -\frac{1}{kn^\gamma} \log q \frac{\partial q}{\partial t}$. Obviously, $q \geq 1$ and $t'_B > 0$ (otherwise, the algorithm never terminates). If there is some constant (with respect to the time) Q such that $q \leq Q$ everywhere, then the relation derived for constant buffer size holds (as q may then be approximated by Q and the rate of growth does not change). Therefore, it is enough to consider the case of q being an increasing function. Then, $\frac{\partial q}{\partial t} > 0$, and this implies $\frac{\partial t'_B}{\partial t} < 0$, because $\log q > 0$ for $q > 1$. That is, t'_B is a decreasing function. Since $t'_B(0)$ is finite, there exists some Q such that $t'_B(q) \leq 0$, for all $q \geq Q$. By Relation (5), the termination time of the algorithm is less than t'_B. Therefore, when $q \geq Q$, the upper limit for the termination time is negative, which means that the algorithm never stops. Thus the values q for which the algorithm terminates are bounded again and we are in the case covered by constant buffer size. □

3.2 Extending the Upper Bound on the Running Time

In the previous section we established an upper bound on the running time only for proving the complexity of sorting d-algorithms. In this section, we analyze this bound. We begin by considering sorting d-algorithms, and then generalize our results for other d-algorithms.

First, note that a side consequence of the proof of Theorem 3.2 is that the best value for the buffer size q is the minimal possible, i.e., 1, because t'_B is a decreasing function with respect to q. In other words, there is no reason to buffer data; it is better to insert each arrived datum in linear time. Hence, we will consider $q = 1$. Second, the time required to insert one element ($q = 1$) into the already sorted sequence is cl, for some constant c. The expression for t'_B becomes in this case $t'_B = \frac{1}{c(kn^\gamma)^2} - \frac{n}{kn^\gamma}$.

[2] This notation was chosen in order to be consistent with the notation from [5]. There, t_B denotes an upper bound on the running time. Obviously, Relation (5) also defines an upper bound.

We have considered $\beta = 1$. This implies that the product $\alpha\beta$ is larger than 1, and the existence of a limit on the running time in this case was established in [5]. More interesting is the situation where $\beta \leq 1/2$, for $\alpha\beta \leq 1$. Under these conditions, no limit on the running time is known. We now study this case.

Theorem 3.3 *For the polynomial data arrival law given by Relation (1), if a sorting d-algorithm terminates, then its running time is upper bounded by a constant T that does not depend on n.*

Proof. We have the restriction $\gamma = 1$ for easier calculations. The time dt after which a new datum arrives is given by $1 = kn^{\gamma}((t + dt)^{\beta} - t^{\beta})$, for some moment t. That is, $(t + dt)^{\beta} - t^{\beta} = 1/kn^{\gamma}$. On the other hand, Relation (4) in the general case becomes $dt \geq q \log q + n + kn^{\gamma}(t^{\beta} - q/kn^{\gamma})$. From these two relations, $1/kn^{\gamma} \geq (q(\log q - 1) + n + kn^{\gamma}t^{\beta} + t)^{\beta} - t^{\beta}$. In particular, for $\gamma = 1$,

$$\frac{1}{kn} \geq (q(\log q - 1) + n + knt^{\beta} + t)^{\beta} - t^{\beta}. \tag{6}$$

The complexity of the sorting algorithm is $O(N^2)$ by Theorem 3.2. That is, for $\gamma = 1$, $n = t^{1/2}/c(1 + kt^{\beta})$. By substituting this value in Relation (6) and manipulating the obtained expression,

$$\frac{qc}{k} \geq b(t) \times a(t). \tag{7}$$

where $a(t) = (q(\log q - 1) + \frac{t^{1/2}}{c} + t)^{\beta} - t^{\beta}$ and $b(t) = t^{1/2}/(1 + kt^{\beta})$. Then, $\frac{\partial b(t)}{\partial t} = \frac{t^{-1/2}}{(1+kt^{\beta})^2}(1/2 + k(1/2 - \beta)t^{\beta})$, and hence, for $\beta \leq 1/2$, $\frac{\partial b(t)}{\partial t} > 0$. That is, $b(t)$ is an increasing function. Analogously, $a(t)$ is an increasing function as well:

$$\begin{aligned}
\frac{\partial a(t)}{\partial t} &= \beta\left(\frac{t^{-1/2}}{c} + 1\right)\left(q(\log q - 1) + \frac{t^{1/2}}{c} + t\right)^{\beta - 1} - \beta t^{\beta - 1} \\
&> \beta\left(\left(q(\log q - 1) + \frac{t^{1/2}}{c} + t\right)^{\beta - 1} - t^{\beta - 1}\right) \text{ [because } t^{-1/2}/c > 0] \\
&> 0 \text{ [because } q(\log q - 1) + t^{1/2}/c > 0 \text{ for large enough } t].
\end{aligned}$$

Therefore, $b(t) \times a(t)$ is increasing. Moreover, it is easy to see that, for $\beta < 1/2$, $\lim_{t\to\infty} b(t) = \infty$, and $\lim_{t\to\infty} a(t) > 0$. Therefore, $\lim_{t\to\infty} b(t) \times a(t) = \infty$ for $\beta < 1/2$. For $\beta = 1/2$, $lim_{t\to\infty} b(t) = 1/k$, and, for large enough t, $a(t) \geq t^{1/8}$ and thus $lim_{t\to\infty} a(t) = \infty$. Then again, $\lim_{t\to\infty} b(t) \times a(t) = \infty$. Since $b(t) \times a(t)$ is an increasing function and its limit is infinite, there exists some finite T such that $b(t) \times a(t) > \frac{qc}{k}$ for any $t > T$. Then, such a t larger than T will contradict the necessary condition for algorithm termination given by Relation (7). Hence, T is an upper bound for the running time and this completes the proof. □

Note that the theorem implicitly gives an upper bound for the maximum amount of data which can be processed, because this amount is given by $N = n + kn^{\gamma}t^{\beta}$ and its upper bound is obviously $n + kn^{\gamma}T^{\beta}$. We contradict by Theorem 3.3 the results derived in [5], where it is claimed that such a bound does not exist for $\alpha\beta < 1$.

In the case of sorting on a linear structure we found an upper bound on the running time for any data arrival law. Sorting is not the only case in which such a bound exists though.

Theorem 3.4 *For the polynomial data arrival law given by Relation (1), let A be any d-algorithm with time complexity $\Omega(N^{\alpha})$, $\alpha > 1$. If A terminates, then its running time is upper bounded by a constant T that does not depend on n.*

Proof. We consider only the case $\beta \leq 1/\alpha$, because the limit has been already found for $\alpha\beta > 1$ [5]. Let $\varepsilon = \alpha - 1$, $\varepsilon > 0$. If the algorithm terminates at some finite time t, then N data have been processed, $N = n + kn^{\gamma}t^{\beta}$. That is, the time for processing one datum is $cN^{\alpha}/N = cN^{\varepsilon}$ for some positive constant c. Following the same idea as the one used for deriving Relation (4), we obtain $dt \geq c(n + kn^{\gamma}t^{\beta})^{\varepsilon}$, which is similar to Relation (4). Therefore, analogously to the proof of Theorem 3.3, we obtain for $\gamma = 1$

$$\frac{qc}{k} \geq \frac{t^{1/2}}{(1 + kt^{\beta})}\left((c(n + knt^{\beta})^{\varepsilon} + t)^{\beta} - t^{\beta}\right). \tag{8}$$

The left-hand side of this relation is increasing, because $c(n + knt^{\beta})^{\varepsilon} > 0$, and it is immediate that the limit of the right-hand side is infinite. Hence, the limit T is derived in the same way as in the proof of Theorem 3.3. □

We now consider parallel d-algorithms. Recall that P is the number of processors in the parallel model. It is immediate that the process described in Lemma 3.1 admits linear speedup. Indeed, sorting q elements admits linear speedup [1] (page 179), and inserting the buffer into the previously sorted sequence may be achieved by using an optimal merging algorithm [1] (page 209). Thus, Relation (5) becomes in the parallel case $t \leq P/(kn^{\gamma})^2 + q(1 - \log q)/kn^{\gamma} - n/kn^{\gamma}$. As expected, this relation is similar to Relation (5) for the sequential case. Therefore, all the above sequential results hold for the parallel case as well. That is, the best value for q is 1 (buffering does not help), and a limit $t''_B(P)$, similar to t'_B, for the running time can be found, $t''_B(P) = \frac{P}{c_p(kn^{\gamma})^2} - \frac{n}{kn^{\gamma}}$. It is then easy to see that Theorem 3.4 holds for the parallel case as well.

Theorem 3.5 *For the polynomial data arrival law given by Relation (1), let A be any P-processor d-algorithm with time complexity $\Omega(N^{\alpha})$, $\alpha > 1$. If A terminates, then its running time is upper bounded by a constant T that does not depend on n but depends on P.*

Proof. It is enough to replace the first term from the right-hand side of Relation (8) in the proof of Theorem 3.4 by

$P \times t^{1/2}/(1+kt^{\beta})$. The proof is then analogous, as this replacement introduces a multiplicative constant, which does not change the sign of the derivative. As well, the appearance of P does not change the limit. $\square$

Finally, it is worth pointing out that a closer look at the proof of Proposition 2.1 given in [5] reveals that the optimality of the d-algorithm in question is not used to establish the result. Therefore:

Theorem 3.6 *Proposition 2.1 holds for any optimal or non-optimal d-algorithm with polynomial running time.*

3.3 Using Another Data Arrival Law

Up to this moment, we have considered a polynomial data arrival law for the sorting problem, but we expect similar results for other expressions. As an example, we consider here a totally different law: $N(t) = q(n) + q(n)\lfloor t/r(n) \rfloor$, for some fixed n, where $q, r : \mathbb{N} \to \mathbb{N}$. In plain English, data arrive in bundles of $q(n)$ elements each $r(n)$ time units. This law is interesting because it extends the analysis of d-algorithms to non-continuous functions. For such an arrival law, we first note that the minimum value for the size of the buffer q is $q(n)$. Indeed, $q(n)$ data arrive together, and all of them must be temporarily stored into some buffer until they are inserted into the sorted sequence. Second, the time interval dt between two data arrivals is $r(n)$. By the same method used to obtain t'_B we have $dt \geq q \log q + N(t - t_q)$. But t_q is null (since the whole bundle of data arrives at once) and $q = q(n)$. Hence, since sorting and merging admit linear speedup, the limits for the running time of the sequential and parallel P-processor algorithms are given by:

$$\begin{aligned}
\lfloor t/r(n) \rfloor &\leq \frac{r(n)}{q(n)} - (1 + \log q(n)) \\
\lfloor t/r(n) \rfloor &\leq P\frac{r(n)}{q(n)} - (1 + \log q(n))
\end{aligned}$$

The two relations above are very similar to the ones obtained for the polynomial arrival law. Therefore, a form of Theorems 3.4, 3.5, and 3.6 holds.

4 Conclusions

In our study of data-accumulating algorithms for sorting, we have taken a different approach than the one in [5]: we first obtained a bound for the running time and, based on this result, we were able to characterize sorting d-algorithms in terms of complexity. This bound also helped us find the optimal size of the input buffer. Note that a sorting (d-)algorithm updates the data structure in a time interval that depends on the number of already processed input data. This is the reason for which the upper bound on the running time t'_B exists. Therefore, we expected similar upper bounds for other problems with this property, such as the construction of search trees (problems 3 and 4 in [5]). Our expectations were justified as shown in Theorems 3.4 and 3.5.

Considering a data arrival law other than the polynomial one, we found that properties of sorting d-algorithms do not change significantly. Based on this, we conjecture that the general properties of such algorithms hold for any type of data arrival law.

Theorems 3.4 and 3.5 are the main results of this paper. They prove the existence of an upper bound on the running time for a large class of algorithms. It has been claimed [5] that the existence of such a limit depends on both α and β. That is, it depends on both the complexity of the d-algorithm and the data arrival law. We have shown that, in fact, the existence of such a bound depends only on the complexity of the d-algorithm. This is a serious limitation of d-algorithms. It is, however, consistent with the results in [3], where we conjecture that problems solvable in real time are included NLOGSPACE. Indeed, Theorems 3.4 and 3.5 restricts (successful) d-algorithms so that their static counterparts fall into this class.

References

[1] S. G. AKL, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[2] R. P. BRENT, *The parallel evaluation of general arithmetic expressions*, Journal of the ACM, 21 (1974), pp. 201–206.

[3] S. D. BRUDA AND S. G. AKL, *On the relation between parallel real-time computations and sublogarithmic space*, Tech. Rep. 2001-446, Department of Computing and Information Science, Queen's University, 2001. http://-www.cs.queensu.ca/TechReports/Reports/2001-446.pdf.

[4] F. LUCCIO AND L. PAGLI, *The p-shovelers problem (computing with time-varying data)*, in Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing, 1992, pp. 188–193.

[5] ——, *Computing with time–varying data: Sequential complexity and parallel speed–up*, Theory of Computing Systems, 31 (1998), pp. 5–26.

[6] F. LUCCIO, L. PAGLI, AND G. PUCCI, *Three non conventional paradigms of parallel computation*, in Parallel Architectures and Their Efficient Use, F. M. auf der Heide, B. Monien, and A. L. Rosenberg, eds., Springer Lecture Notes in Computer Science 678, 1992, pp. 166–175.

[7] J. R. SMITH, *The Design and Analysis of Parallel Algorithms*, Oxford University Press, 1993.

[8] J. D. ULLMAN, A. V. AHO, AND J. E. HOPCROFT, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.