BRICS

Basic Research in Computer Science

BRICS RS-99-3 P. B. Miltersen: Two Notes on the Computational Complexity of One-Dimensional Sandpiles

Two Notes on the Computational Complexity of One-Dimensional Sandpiles

Peter Bro Miltersen

BRICS Report Series **RS-99-3**

ISSN 0909-0878 **February 1999**

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/99/3/`

Two notes on the computational complexity of one-dimensional sandpiles

Peter Bro Miltersen*

February, 1999

Abstract

We prove that the one-dimensional sandpile prediction problem is in $\mathbf{AC}^1$. The previously best known upper bound on the $\mathbf{AC}^i$-scale was $\mathbf{AC}^2$. We also prove that it is not in $\mathbf{AC}^{1-\epsilon}$ for any constant $\epsilon > 0$.

1 Introduction

In a recent paper, Moore and Nilsson [3] considered the following *one-dimensional sandpile prediction problem*, which can be described as follows (the description of Moore and Nilsson is slightly different but can be easily seen to be equivalent to the following). A one-dimensional sandpile is a map $h : \mathbf{Z} \to \{0, 1, 2, \ldots\}$. The value $h(i)$ is interpreted as the height of a pile of sand at position i. A pile of height 2 or more is *unstable*, and may *topple*, distributing sand evenly to the two neighboring positions. Formally, if map h' satisfies that for some i, $h'(i) = h(i) - 2, h'(i-1) = h(i-1) + 1, h'(i+1) = h(i+1) + 1$, and $h'(x) = h(x)$ for all $x \notin \{i-1, i, i+1\}$, we'll say that h' is a possible successor to h, or $h \to h'$. If there is no possible successor to h, we'll say that h is *stable*. If h is zero outside some interval I, say, $I = \{1, 2, \ldots, n\}$, it can be shown that there is a unique stable g, so that

*BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus, Denmark. Email: bromille@brics.dk.

$h \to^* g$. We will denote this unique g by h^*. The one-dimensional sandpile prediction problem is the following: Given $h : \{1, 2, \ldots, n\} \to \{0, 1, 2\}$, find h^*.

A similar definition can be made for the d-dimensional sandpile problem for any $d \geq 1$, but in this paper, we concentrate on the one-dimensional case.

The sandpile model is popular in the theory of complex dynamical systems. A general program by Moore and others seek to capture and measure the informal notion of the "complexity" of such a system by the *computational* complexity of the problem of predicting the behavior of the system. Following this program, Moore and Nilsson showed that the d-dimensional sandpile prediction problem is in $\mathbf{P}$ for all d, that it is $\mathbf{P}$-complete for $d \geq 3$ and that the 1-dimensional problem is in $\mathbf{AC}^1[\mathbf{NL}] \subseteq \mathbf{AC}^2 \subseteq \mathbf{NC}^3$. They also present an efficient sequential algorithm for the 1-dimensional problem with a time bound of $O(n \log n)$.

The purpose of this note is to present two pieces of information about the complexity of the one-dimensional sandpile prediction problem.

First we show the following improvement of the parallel upper bound of Moore and Nilsson: *The one-dimensional sandpile prediction problem is in* $\mathbf{AC}^1 \subseteq \mathbf{NC}^2$. Conceptually, the algorithm is simpler than the parallel algorithm of Moore and Nilsson. It is based on refining their *sequential* algorithm slightly, and then noticing that the refined algorithm can be carried out by a deterministic logspace Turing machine with access to an auxiliary pushdown store. It is known that the class of languages so computed is **LOGDCFL** [5], the class of languages logspace reducible to a deterministic context-free language, and as $\mathbf{LOGDCFL} \subseteq \mathbf{AC}^1$ [4], the result follows.

Second, we prove that *one-dimensional sandpile prediction is hard for* $\mathbf{TC}^0$ *by* $\mathbf{AC}^0$*-reductions.* That the one-dimensional sandpile prediction problem is hard for $\mathbf{TC}^0$ means that the one-dimensional sandpile is sufficiently complex to carry out at least some slightly non-rudimentary computation, such as computing the parity or majority of n bits. This provides a (weak) formal justification for the statement of Moore and Nilsson that the dynamics of the one-dimensional sandpile problem is "non-trivial". It also implies the following lower bound: *The one-dimensional sandpile prediction problem is* not *in* $\mathbf{AC}^{1-\epsilon}$ *for any constant* $\epsilon > 0$. As $\mathbf{AC}^i$ is the class of problems solvable by CRCW PRAMs with polynomially many processors in time $O((\log n)^i)$, we thus have fairly tight upper and lower bounds for solving the one-dimensional sandpile problem in this model of computation.

2 The upper bound

In their paper, Moore and Nilsson show that the one-dimensional sandpile problem can be solved by the following sequential algorithm. Given an input sandpile $h : \{1, \ldots, n\} \to \{0, 1, 2\}$, extend it to $\mathbf{Z}$ by making zero the values of h outside the interval from 1 to n. We maintain two subsets T and N of the integers. Initially T is the set $\{i \in \mathbf{Z} | h(i) = 2\}$ and N is the set $\{i \in \mathbf{Z} | h(i) = 0\}$. Note that while N is an infinite set, it has an obvious finite representation. Now, while T is not empty repeat the following two steps, a *round*:

1. Pick an $t \in T$. Find $z_1, z_2 \in N$ so that $z_1 \leq t < z_2$.
2. Let $N = (N - \{z_1, z_2\}) \cup \{z_1 + z_2 - t\}$. Let $T = T - \{t\}$.

When T is empty, h' defined to be 0 on N and 1 on $\mathbf{Z} - N$ is the unique stable successor of h. We refer the reader to the argument for this in Moore and Nilsson.

Moore and Nilsson suggest implementing the above algorithm directly by maintaining the "finite part" of the set N in sorted order, and, for each $t \in T$, do a binary search in N to find z_1 and z_2. Our first observation is that this binary search can be eliminated when going through the list T in increasing order. Indeed, we can maintain the following *invariant*. Between rounds, if $T = \{t_1 < t_2 < \ldots < t_m\}$, we can maintain a partition of N into N_1 and N_2 in such a way that the following properties are true:

1. All elements in N_1 are smaller than each element in N_2.
2. The elements z_1, z_2 so that $z_1 \leq t_1 < z_2$ are either
   (a) The largest element of N_1 and the smallest element of N_2 or
   (b) Both in N_2.

First note that we can easily establish the invariant at the beginning of the computation. Now given the invariant, we want to perform one round of the algorithm of Moore and Nilsson. We should find an appropriate z_1, z_2 so that $z_1 \leq t_1 < z_2$. If case (a) applies, we remove t_1 from T (so t_2 will be the "new" t_1 in the next round), z_1 from N_1 and z_2 from N_2. Note that $z_1 < z_1 + z_2 - t_1 \leq z_2$ and also note the t_2 is greater than the new largest element of N_1 (since t_1 was). Thus, if we insert $z_1 + z_2 - t_1$ as the new largest element of N_1 if $z_1 + z_2 - t_1 \leq t_2$ and we insert $z_1 + z_2 - t_1$ as the new smallest element of N_2 if $t_2 < z_1 + z_2 - t_1$, we have maintained the invariant. If case (b) applies, we move elements from N_2 to N_1 until case (a) applies, reducing it to this case. This completes the description of the algorithm.

Because of case (b), the reader may conclude that we have replaced binary search with linear search which does not sound like such a grand idea, but note that the size of N_2 is never increased during a round. Thus, the total cost of moving elements from N_2 during the entire execution of the algorithm is linear.

Concerning the complexity of the refined algorithm, viewed as a sequential algorithm we have to be somewhat careful about the exact model of computation. On a log cost RAM, the unrefined algorithm of Moore and Nilsson has complexity $O(n \log n)$ because of

1. The binary searches which cost $O(\log n)$ time each.

2. Adding and subtracting $O(\log n)$-bit integers, each operation costing $O(\log n)$ time.

The refined algorithm also has complexity $O(n \log n)$ in the log-cost model. However, it is common practice to *only* use the log-cost time measure when integers of bit length much bigger than the log of the input are involved (e.g., as in case of the problem of multiplying two n-bit integers). When only $O(\log n)$-bit length integers are involved, the *unit cost* RAM model is much more commonly used. In the unit cost RAM model, the refined algorithm has complexity $O(n)$ while the Moore-Nilsson version keeps having complexity $O(n \log n)$ because of the binary searches.

More important than its sequential complexity in various RAM models is the consequences of the refined algorithm for the parallel complexity of the sandpile prediction problem. Indeed, we next note that the refined algorithm can be implemented by a polynomial time, logspace Turing machine with access to an auxiliary pushdown store (i.e., an extra "free" tape, where a tape cell is erased when the head moves from the cell to its left neighbor). Because of the robustness of logarithmic space, we just have to argue that the algorithm can be implemented by a while-program using $O(1)$ variables, each holding an integer of $O(\log n)$ bits and an auxiliary object STACK where such $O(\log n)$ bit integers can be pushed and popped. We show how each variables in the refined algorithms can be represented using such objects.

N_1 will be represented by the STACK object and a single integer variable v. The invariant of the representation is the following: N_1 is exactly $\{\ldots, -v-2, -v-1, -v\} \cup \{\text{the elements in STACK}\}$ furthermore, the elements of STACK are sorted, with the largest at the top. With this representation, we can remove the largest element from N_1, and add an element to N_1 larger than the largest one. These are the only operations we need to perform on N_1 when running the refined algorithm. We can also easily initialize the representation to the correct content in the beginning of the refined algorithm.

N_2 will be represented by three integer variables l, i and w. The invariant of the representation is that $N_2 = \{l\} \cup \{j \geq i | H[i] = 0\} \cup \{j \in \mathbf{Z} | j \geq w\}$. Here $H[1..n]$ is the structure holding in the input, representing the map $h : \{1, \ldots, n\} \to \{0, 1, 2\}$. With this representation, we can replace the smallest element of N_2 by another element and remove the smallest element from N_2. These are the only two operations the refined algorithm uses.

T is represented in a way similar to N_2.

We have now shown that the refined algorithm for one-dimensional sandpile prediction can be implemented by a deterministic, polynomial time, logspace Turing machine with access to an auxiliary pushdown store. At the end of the algorithm, the output is contained in the representation of N_1 and N_2. If we are interested in the i'th bit of the output for some i, we can find it by either inspecting the structure for N_2 or the structure for N_1, in the last case popping a sufficient number of elements from the stack. Thus the language 1SANDPILE $= \{\langle h, i\rangle |$ the i'th bit in h^* is $1\}$ can be decided by a deterministic logspace Turing machine with an auxiliary pushdown store. By a result of Sudborough [5], this means that the language is in **LOGDCFL** which, by a result of Ruzzo [4] is a subset of $\mathbf{AC}^1$. The functional version of the problem, i.e. the map $h \to h^*$ itself is therefore computable in the same class.

3 The lower bound

Moore and Nilsson showed **P**-completeness of higher-dimensional versions of the sandpile prediction problem; here we show hardness for much lower complexity classes of the one-dimensional problem. When considering **P**-completeness, logspace reductions are most often used. However, these are not meaningful for classes below **L**, as those classes are not closed under logspace reductions. Here, we use a weaker notion of reductions, *DLOGTIME-uniform constant depth reductions* [1]. All classes considered here are closed under those reductions. A language π_1 reduces to an language to π_2 by such reductions if we can build DLOGTIME-uniform, constant depth, polynomial size circuit for π_1, using unbounded fan-in AND-gates, unbounded fan-in OR-gates, NEGATION-gates, and *oracle*-gates computing π_2.

Let MAJORITY be the problem of deciding whether a string of n input bits (n odd) have more 1's than zeros. We shall show that MAJORITY reduces to 1SANDPILE. Since MAJORITY is complete for $\mathbf{TC}^0$ by constant depth reductions (indeed, the *definition* of $\mathbf{TC}^0$ is that it is the closure of MAJORITY under constant depth reductions), it follows that 1SANDPILE is hard for $\mathbf{TC}^0$ under constant depth reductions.

Let PARITY be the problem of deciding whether a string of n input bits have an odd number of ones. As PARITY is in $\mathbf{TC}^0$, PARITY is not in $\mathbf{AC}^{1-\epsilon}$ for any $\epsilon > 0$ [2], and $\mathbf{AC}^{1-\epsilon}$ is closed under constant depth reductions, we get the corollary that 1SANDPILE is not in $\mathbf{AC}^{1-\epsilon}$ for any constant $\epsilon > 0$.

It remains to show that MAJORITY constant depth reduces to 1SANDPILE. We have to construct a uniform circuit for MAJORITY using unbounded fan-in AND gates, unbounded fan-in OR gates and 1SANDPILE oracle-gates.

Given an input $x_1 x_2 \dots x_n$ of the MAJORITY problem, we can assume that n is odd. Our circuit first constructs an instance $1 y_{11} y_{12} y_{13} y_{21} y_{22} y_{23} \dots y_{n1} y_{n2} y_{n3}$ of the sandpile problem where $y_{i1} = x_i$, $y_{i2} = 1 - x_i$ and $y_{i3} = 2$, except for $y_{n3} = 1$. For example, we reduce 0101011 to 1 012 102 012 102 012 101. Applying the Moore-Nilsson algorithm, we see that this instance reduces to a stable value with exactly one zero with an index between 1 and $3n$, this index being $n + 1 + \#\text{ones}$ in $x_1 x_2 \dots x_n$. Thus, to find the majority of $x_1 x_2 \dots x_n$, we just need to check if the index of the unique one in the reduced pile is bigger than $3n/2 + 1$. This is easily checked with a uniform constant depth circuit.

4 Discussion and open problems

In general, to carry out the program of making formal the informal notion of the "complexity" of a complex dynamical system by identifying "complexity" with the computational complexity of the prediction problem, it seems appropriate to use the finest scale available in the theory of computational complexity.

We have shown a $\mathbf{TC}^0$ lower bound and a **LOGDCFL** upper bound for the one-dimensional sandpile prediction problem. Obviously, getting tighter bounds would be desirable. In particular, is the one-dimensional problem hard for $\mathbf{NC}^1$? Is it in **L** or **NL**?

Moore and Nilsson classified the d-dimensional sandpile prediction problem as **P**-complete for $d \geq 3$ and left open whether the 2-dimensional sandpile prediction problem is also hard for **P**. We may note that the reduction used by Moore and Nilsson to show that the 3-dimensional problem is hard for **P** also establishes that the 2-dimensional problem is hard for $\mathbf{NC}^1$: Their reduction is a reduction from the monotone circuit value problem and the third dimension is only used when implementing crossovers of wires. Thus, the monotone *planar* circuit value problem reduces to 2-dimensional sandpile prediction, and this problem is hard for $\mathbf{NC}^1$. Getting a better lower bound than $\mathbf{NC}^1$ or a better upper bound than **P** for the 2-dimensional sandpile prediction problem would be most interesting.

5 Acknowledgement

I would like to thank Chris Moore for helpful discussions.

References

[1] D. A. M. Barrington, N. Immerman and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306.

[2] J. Håstad. *Computational Limitations of Small-Depth Circuits.* ACM doctoral dissertation award 1986. MIT Press, 1987.

[3] Christopher Moore and Martin Nilsson. The Computational Complexity of Sandpiles. *Journal of Statistical Physics*, to appear. Also available on `http://www.santafe.edu/ moore/`.

[4] W. Ruzzo. Tree-size bounded alternation. *J. Comp. System Sci.* 21:218-235, 1980.

[5] I. Sudborough. On the tape complexity of deterministic context-free languages. *J. Assoc. Comp. Mach.*, 25:405-414, 1978.

Recent BRICS Report Series Publications

RS-99-3 Peter Bro Miltersen. *Two Notes on the Computational Complexity of One-Dimensional Sandpiles*. February 1999. 8 pp.

RS-99-2 Ivan B. Damgård. *An Error in the Mixed Adversary Protocol by Fitzi, Hirt and Maurer*. February 1999. 4 pp.

RS-99-1 Marcin Jurdziński and Mogens Nielsen. *Hereditary History Preserving Simulation is Undecidable*. January 1999. 15 pp.

RS-98-55 Andrew D. Gordon, Paul D. Hankin, and Søren B. Lassen. *Compilation and Equivalence of Imperative Objects (Revised Report)*. December 1998. iv+75 pp. This is a revision of Technical Report 429, University of Cambridge Computer Laboratory, June 1997, and the earlier BRICS report RS-97-19, July 1997. Appears in Ramesh and Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science: 17th Conference*, FST&TCS '97 Proceedings, LNCS 1346, 1997, pages 74–87.

RS-98-54 Olivier Danvy and Ulrik P. Schultz. *Lambda-Dropping: Transforming Recursive Equations into Programs with Block Structure*. December 1998. 55 pp. To appear in *Theoretical Computer Science*.

RS-98-53 Julian C. Bradfield. *Fixpoint Alternation: Arithmetic, Transition Systems, and the Binary Tree*. December 1998. 20 pp.

RS-98-52 Josva Kleist and Davide Sangiorgi. *Imperative Objects and Mobile Processes*. December 1998. 22 pp. Appears in Gries and de Roever, editors, *IFIP Working Conference on Programming Concepts and Methods*, PROCOMET '98 Proceedings, 1998, pages 285–303.

RS-98-51 Peter Krogsgaard Jensen. *Automated Modeling of Real-Time Implementation*. December 1998. 9 pp. Appears in *The 13th IEEE Conference on Automated Software Engineering*, ASE '98 Doctoral Symposium Proceedings, 1998, pages 17–20.

RS-98-50 Luca Aceto and Anna Ingólfsdóttir. *Testing Hennessy-Milner Logic with Recursion*. December 1998. 15 pp. Appears in Thomas, editor, *Foundations of Software Science and Computation Structures: Second International Conference*, FoSSaCS '99 Proceedings, LNCS 1578, 1999, pages 41–55.