Brief Announcement: Relations Between Space-Bounded and Adaptive Massively Parallel Computations

Michael Chen
Iowa State University, Ames, IA, USA

A. Pavan
Iowa State University, Ames, IA, USA

N. V. Vinodchandran
University of Nebraska–Lincoln, NE, USA

Abstract

In this work, we study the class of problems solvable by (deterministic) Adaptive Massively Parallel Computations in constant rounds from a computational complexity theory perspective. A language L is in the class AMPC^0 if, for every $\varepsilon > 0$, there is a deterministic AMPC algorithm running in constant rounds with a polynomial number of processors, where the local memory of each machine $s = O(N^\varepsilon)$. We prove that the space-bounded complexity class $\mathsf{ReachUL}$ is a *proper subclass* of AMPC^0. The complexity class $\mathsf{ReachUL}$ lies between the well-known space-bounded complexity classes Deterministic Logspace (DLOG) and Nondeterministic Logspace (NLOG). In contrast, we establish that it is unlikely that PSPACE admits AMPC algorithms, even with polynomially many rounds. We also establish that showing PSPACE is a subclass of *nonuniform*-AMPC with polynomially many rounds leads to a significant separation result in complexity theory, namely PSPACE is a proper subclass of $\mathsf{EXP}^{\Sigma_2^{\mathsf{P}}}$.

2012 ACM Subject Classification Computing methodologies → Massively parallel algorithms; Theory of computation → Complexity classes

Keywords and phrases Massively Parallel Computation, AMPC, Complexity Classes, LogSpace, NL, PSPACE

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.37

Funding This work is supported in part by NSF grants 1934884, 2130536, and 2130608.

Acknowledgements The authors thank Sriram Pemmaraju and Meena Mahajan for their helpful discussions. We thank anonymous reviewers for their valuable comments and pointers to some critical references.

1 Introduction

The *Massively Parallel Computation* (MPC) model is widely accepted as the standard theoretical model for distributed computation frameworks such as MapReduce, Spark, Hadoop, FlumeJava, Beame, Pregel, and Gigraph [7, 9]. It was defined in [5], and it captures computation on large data: data is adversarially distributed to processors, and each processor has local memory $s = O(N^\varepsilon)$ $(0 < \varepsilon < 1$ where N is the input size. Computation occurs in rounds, and in each round, every machine performs computation based on its local data and then communicates with other machines with the constraint that the amount of communication by a process is equal to that of its local memory s. A salient feature of the MPC model is that no computational restriction is placed on the processor, except that each processor has local memory s, and a key objective is to minimize the number of rounds.

Ideally, one would like to design an algorithm with constant rounds with a small number of processors. The MPC model has been extensively studied in the context of designing algorithms as well as its relationship with complexity classes [2, 3, 4, 7, 5, 6, 16].

Recent work of [7] introduced an adaptive extension of the MPC model called *Adaptive Massively Parallel Computation* model (AMPC). In the AMPC model, the processors communicate via a shared memory called Distributed Data Stores (DDS) by reading from and writing to the DDS. In a single round, a machine can adaptively query the DDS to obtain s words and write at most s words, and as in the case of MPC, s is $O(N^{\varepsilon})$. In [7], authors designed a constant round randomized AMPC algorithm for 1v2-Cycle as well as a few other graph problems.

In this work, we report progress on the power and limitation of the AMPC model from a computational complexity theory viewpoint. Towards this, we define a robust complexity class which we denote by AMPC^0. A language L is in AMPC^0 if for every ε there is a constant round (depending on ε) AMPC algorithm with $P = p(N)$ many processors (where $p(\cdot)$ is a polynomial) each with $s = O(N^{\varepsilon})$ memory. We define a similar complexity class $\mathsf{AMPC}^{\mathsf{poly}}$ where the number of rounds is polynomial. We study the relationship of these AMPC complexity classes with respect to the standard space-bounded complexity classes DLOG, NLOG, and PSPACE. The starting point of our work is that the ideas from the randomized AMPC algorithm for 1v2-Cycle from [7] can be used to show that the complexity class DLOG is a subset of (uniform) AMPC^0. Motivated by this, we explore whether NLOG is a subset of AMPC^0. We make progress toward this question by studying a complexity class ReachUL [8, 1, 11]. This is a natural complexity class that lies between DLOG and NLOG and has been studied earlier in the context of designing space-efficient algorithms for reachability that beat the Savitch's bound [1]. We prove that ReachUL is a subset of (uniform) AMPC^0, More interestingly, we show that ReachUL is a *proper subset* of (uniform) AMPC^0. On the contrary, we observe that it is unlikely that the whole of PSPACE (or even NP) can be solved even in $\mathsf{AMPC}^{\mathsf{poly}}$. This is because every language that admits (uniform) $\mathsf{AMPC}^{\mathsf{poly}}$ algorithm can be solved in subexponential time. Since we do not believe that PSPACE can be solved in subexponential time, we obtain that it is unlikely that $\mathsf{PSPACE} \subseteq \mathsf{AMPC}^{\mathsf{poly}}$. We also consider the limitation of nonuniform $\mathsf{AMPC}^{\mathsf{poly}}$. We unconditionally show that there exist languages in $\mathsf{E}^{\Sigma_2^{\mathsf{P}}}$ that are not in $\mathsf{AMPC}^{\mathsf{poly}}$. We note that the work reported in [15] also considered the relations of complexity classes such as DLOG and NLOG to MPC model.

▶ Remark. In an algorithmic setting, it is typically desired that the total memory of an AMPC algorithm $P \cdot s$ to be $N \cdot \mathrm{poly}\log(N)$. However, to define a robust complexity class (closed under reductions), we allow P to be polynomial and require that for every $0 < \varepsilon < 1$, there is an algorithm with s local memory per processor.

2 Preliminaries

We now give the formal description of the AMPC model [7, 9]. Let $p(\cdot)$, $s(\cdot)$ and $r(\cdot)$ are functions from $\mathbb{N}$ to $\mathbb{N}$. An $\mathsf{AMPC}[p(N), s(N), r(N)]$ algorithm for length N, is a collection of processors $M_{i,j}$, $1 \leq i \leq p(N)$ and $1 \leq j \leq r(N)$ where each processor has a memory bound of $s(N)$. In addition to the processors there is a collection of *Distributed Data Stores* (DDS) denoted by $\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_{r(N)}$. For each DDS, the data is stored in a bit addressable manner (as done in [9]) i.e., a collection of key-value pairs in the form of $(i, i^{th}$ bit of DDS$)$. The input string $x = x_1 \ldots x_N$ is stored in $\mathcal{D}_0$ in the form of $\{(i, x_i)\}_{i=1}^{N}$. The computation occurs in rounds. The processors $M_{i,j}$, $1 \leq i \leq p(N)$ participate in the jth round. In the jth round, each of these processors is allowed to make $s(N)$ adaptive queries to read from

$\mathcal{D}_{j-1}$ and each processor is allowed to can write up to $s(N)$ bits to $\mathcal{D}_j$. The computation stops after $r(N)$ rounds, and we say that the algorithm accepts string x if the value of key 1 in $\mathcal{D}_{r(N)}$ is 1.

Inherently this is a nonuniform model of computation. A language L is in the class (nonuniform) AMPC^0 if for every $0 < \varepsilon < 1$, there exists a polynomial $p(\cdot)$, and a constant $r = r(N) > 0$ such that for every input length $N \geq 0$, there is a $\mathsf{AMPC}[p(N), N^\varepsilon, r]$ algorithm that accepts L on strings at length N. We define the uniform AMPC model. This definition is similar to the uniform MRC model as defined in [10]. For an algorithm P, we use $P_{i,j}$ to denote a processor whose behavior is the same as P on inputs i and j. A language L is in the class (uniform) AMPC^0 if for every $\varepsilon > 0$, there exists a polynomial $p(\cdot)$ and a constant $r = r(N)$, and a logspace bounded algorithm U that on input 1^N outputs the code of a processor P with the following properties: the processors $P_{i,j}$ $1, \leq i \leq p(N)$, $1 \leq j \leq r$ constitute a $\mathsf{AMPC}[p(N), N^\varepsilon, r]$ algorithm that accepts L at strings of length N. Analogously we define uniform and nonuniform versions of the class $\mathsf{AMPC}^{\mathsf{poly}}$ where the number of rounds is allowed to be a polynomial. In the rest of the document, we write AMPC^0 to denote (uniform) AMPC^0.

We use DLOG (resp. PSPACE) to denote the class of languages accepted by deterministic logspace (resp. polynomial-space) Turing machines. The complexity class E is the class of languages that are accepted by deterministic $2^{O(N)}$-time bounded machines, and Σ_2^{P} denote the class of languages in the second level of the polynomial-hierarchy. A language L is in the class SubEXP, if for every $\varepsilon > 0$, there is a $O(2^{N^\varepsilon})$-time bounded machine that accepts L. A language L is in NC^1 if L can be decided by a family of circuits $\{C_N\}_{N \in \mathbb{N}}$ where C_N has $\mathrm{poly}(N)$ size and $O(\log N)$ depth.

▶ **Definition 1** ([8, 1]). *A nondeterministic machine is called reach-unambiguous if for every configuration C, there is at most one path from the start configuration to C. The class* $\mathsf{ReachUL}$ *is the class of languages that are accepted by $O(\log N)$-space-bounded reach-unambiguous machines.*

▶ **Definition 2** (Reach-Unambiguous). Reach-Unambiguous *is the language consisting of tuples $\langle G, a, b\rangle$ such that (1) $G = (V, E)$ is a directed graph, (2) for all $u \in V$ there exists at most 1 directed path from a to u and, (3) there exists a directed path from a to b.*

It is known that Reach-Unambiguous is complete for $\mathsf{ReachUL}$ with respect to logspace reductions [13, 8, 1].

3 Results

3.1 ReachUL in AMPC^0

We show that $\mathsf{ReachUL}$ is a proper subset of AMPC^0. We start with the following theorem.

▶ **Theorem 3.** $\mathsf{DLOG} \subsetneq \mathsf{AMPC}^0$. *That is,* DLOG *is a proper subset of* AMPC^0.

▶ **Corollary 4.** AMPC^0 *is closed under logspace reductions.*

Inclusion of Theorem 3 follows from [7]. The authors showed a randomized constant round AMPC algorithm for the DLOG-complete problem, 1v2-Cycle. Their algorithm can be modified to obtain a deterministic algorithm by allowing for $O(N)$ processors and using $O(N^{1+\varepsilon})$ total memory for any ε. The strictness of inclusion follows from Theorem 8 which we prove.

Let $\langle G, a, b\rangle$ be an input instance of Reach-Unambiguous where $G = (V, E)$ is a reach-unambiguous graph such that $V = \{v_1, \ldots, v_N\}$ and $a, b \in V$. Without loss of generality, we assume that the out-degree of each vertex is at most 2. For $u \in V$ and $s \in \mathbb{N}$, define $T_s(u)$ to be the tree resulting from a *Breadth First Search* (BFS) starting from u in G upto s nodes such that no node is partially visited, i.e., either all the children of any vertex are in the tree, or none of them are. We shall call every node other than u in $T_s(u)$ a descendent of u. The main ingredient in our proof is Algorithm 1 that constructs a compressed version of $T_s(u)$. This algorithm is based on the tree contraction idea [1]. We note that recently tree contraction has been studied in the context AMPC in [12]. For any graph G, we often overload the notation G to also refer to the vertex set of the graph.

▶ **Definition 5.** *Let $u \in V$, and v be a descendant of u in $T_s(u)$. v is said to be an intermediate vertex for $T_s(u)$ if there exists an edge $(v, w) \in E$ such that $w \notin T_s(u)$. Define $I_s(u) \subseteq T_s(u)$ as the set of vertices that are intermediate for $T_s(u)$. We say that $T_s(u)$ is* complete *if $I_s(u) = \emptyset$, otherwise $T_s(u)$ is incomplete.*

Intermediate vertices capture the idea of vertices that can still be explored. If v is an intermediate vertex for $T_s(u)$, that means v can still be further explored. But due to the BFS parameter s, it could not explore v any further. We shall assume for simplicity of the analysis that if a tree $T_s(u)$ is incomplete, the tree has exactly $s+1$ vertices (in general, such a tree could have either s or $s+1$ vertices). Thus the condition $|T_s(u)| < s+1$ denotes the condition that $T_s(u)$ is complete.

Algorithm 1 Construct Algorithm.

```
1  Function Construct(u, s):
2      Compute T_s(u) using at most O(s) queries.
3      if b ∈ T_s(u) then
           // b can be reached from u within s queries
4          T'_s(u) ← ({b}, ∅)
5      else if |T_s(u)| < s + 1 then
           // b cannot be reached from u
6          T'_s(u) ← ({u}, ∅)
7      else
           // b cannot be reached from u within s queries, need to explore
8          Compute I_s(u) using at most O(s) queries
9          T'_s(u) ← A complete binary tree whose leaves are exactly I_s(u)
10     Write T'_s(u) to the DDS
```

Let $T'_s(u)$ be the output of $Construct(u, s)$ in Algorithm 1. $T'_s(u)$ is a contracted version of $T_s(u)$. If $b \in T_s(u)$ or $|T_s(u)| < s+1$, the search from u is completed, and we can contract the tree to a single node. Otherwise, the tree is contracted to a complete binary tree whose leaves are $I_s(u)$, which are precisely the candidates that can lead to b. Locally, it is possible that $T'_s(u)$ does not contract. Claim 6 shows that globally the contraction will occur.

Define the tree T' generated by starting with $T'_s(a)$, and recursively substituting every leaf $l \in T'$ with $T'_s(l)$. Continuing the process until substituting leaves does not change the tree. This graph has the property that $\langle T', a, b\rangle \in$ Reach-Unambiguous $\iff \langle G, a, b\rangle \in$ Reach-Unambiguous since the only vertices that remain are those vertices that have the potential to reach b. For an AMPC model, T' need not be explicitly constructed since each

tree is locally computed and is then updated in the DDS. We shall now show that the graph size reduces by a factor of $s/2$, which is sufficient to get the algorithm to halt in constant rounds by setting $s = O(N^\varepsilon)$.

▷ Claim 6. $|T'| \leq 2N/s$

Given $u \in V$, for analysis' sake construct $H_s(u)$ by making every descendant in $T_s(u)$ a child of u, i.e. $H_s(u)$ is a re-arranged version of $T_s(u)$ such that edges go from u to the descendants of u in $T_s(u)$.

We now construct an s-ary tree, H, such that it is always full (vertices either have out degree 0 or s). Start with $H_s(a)$, then for every leaf l whose parent is p such that $l \in I_s(p)$ substitute l with $H_s(l)$ if $T_s(p) = s + 1$. If the BFS search was incomplete, substitute it with b if $b \in H_s(l)$, otherwise, do nothing. Repeat the process until no more substitutions can be done. This construction leads to a full s-ary tree H, such that $|H| \leq N$. H represents a BFS traversal done in "batches" of size s, where only intermediate nodes are substituted with another s-ary tree. Exploring non-intermediate nodes would be redundant. Let i denote the number of internal nodes of this s-ary tree.

Proof of Claim 6. Since H is a full s-ary tree, $i = |H|/s \leq N/s$. And H is essentially a rearrangement of T' such that internal vertices in H correspond to intermediate vertices in T'. However, due to line 9 of Algorithm 1, we may be adding more vertices, but however, it is no more than twice. i.e. we have $|T'| \leq 2i$. Therefore we have $|T'| \leq 2N/s$ ◁

▶ **Theorem 7.** Reach-Unambiguous $\in$ AMPC^0

Proof. Let $\langle G, a, b\rangle$ be a problem instance of Reach-Unambiguous with $G = (V, E)$ such that $V = \{v_1, \ldots, v_N\}$. Fix $\varepsilon \in (0, 1)$. Define the AMPC algorithm with $s = O(N^\varepsilon)$ local memory. Assign $G_1 \leftarrow G$ and $R \leftarrow O(1/\varepsilon)$, for $i = 1, \ldots, R$ rounds do the following, for each $v \in G_i$, a machine executes *Construct*(v, s) for G_i to get a new graph T' as described earlier. Assign $G_{i+1} \leftarrow T'$. After the rounds are complete, $|G_R| = \frac{N}{(s/2)^R} = O(1)$. Then a single machine can perform normal reachability on G_R, accepting the input if and only if there is a path from a to b. ◀

▶ **Theorem 8.** *For every $c \in \mathbb{N}$. There exists a problem in* AMPC^0 *that is not in* DSPACE$(\log^c N)$

Proof. Let $A \in$ DSPACE(N^2) but $A \notin$ DSPACE(N). We know such a problem exists due to the space hierarchy theorem. Consider a padded language B defined as $B = \{\langle x, y\rangle \mid x \in A, |x| = M, |y| = 2^{M^{1/c}} - M\}$.

We claim that $B \notin$ DSPACE$(\log^c N)$. Assume by contradiction, $B \in$ DSPACE$(\log^c N)$. We show that in that case, $A \in$ DSPACE(N). Let x be an input instance of A of length M; we shall solve it by reducing it to an instance of B, by generating $z = \langle x, y\rangle$, where $y = 0^{2^{M^{1/c}} - M}$, we have $|z| = N = 2^{M^{1/c}}$. The instance z need not be explicitly stored; every bit can be computed on the fly. Thus the space used is $O(M)$. Then use algorithm for B to solve z using space $O(\log^c N) = O(\log^c 2^{M^{1/c}}) = O(M)$. Thus $A \in$ DSPACE(N) a contradiction. Therefore, $B \notin$ DSPACE$(\log^c N)$.

Now we show that B is in AMPC^0. Let $z = \langle x, y\rangle$ be a problem instance of B of length N. The crucial point to note is that the membership of z in B depends only on x, which has length $O(\log^c N)$. If x does not have this length, we can safely reject it. Fix an arbitrary $\varepsilon \in (0, 1)$. Consider the AMPC algorithm with just one machine and one round. Let $z = \langle x, y\rangle$ be an input of length N. The machine $\mathcal{M}$ reads x which has length $M = O(\log^c N) \subseteq O(N^\varepsilon)$,

then checks if $x \in A$ using space $O(M^2) = O(\log^{2c} N) \subseteq O(N^\varepsilon)$, via our assumption that $A \in \mathsf{DSPACE}(N^2)$. If $x \in A$ then $\mathcal{M}$ accepts otherwise rejects. Thus we have exhibited a language B such that $B \in \mathsf{AMPC}^0$ but $B \notin \mathsf{DSPACE}(\log^c N)$. ◀

▶ **Theorem 9.** $\mathsf{ReachUL} \subsetneq \mathsf{AMPC}^0$. *That is,* $\mathsf{ReachUL}$ *is a proper subset of* AMPC^0.

Proof. The containment follows since Reach-Unambiguous is complete for $\mathsf{ReachUL}$ under logspace reductions and by Corollary 4, AMPC^0 is closed under logspace reductions. The strict containment follows from the fact that $\mathsf{ReachUL} \subseteq \mathsf{NLOG} \subseteq \mathsf{DSPACE}(\log^2 N)$. Hence by Theorem 8, there is a language in AMPC^0 that is not in $\mathsf{ReachUL}$. ◀

3.2 Limitations

This section discusses the limitations of the AMPC model in relation to well-known complexity classes.

Uniform Model. Since each processor in each round has a memory bound of $O(N^\varepsilon)$, the number of configurations of each processor is $\mathsf{poly}(2^{N^\varepsilon})$ and hence runs in $O(2^{N^{\varepsilon'}})$ for some $0 < \varepsilon' < 1$ (since the processors are halting). Thus it is clear that uniform $\mathsf{AMPC}^{\mathsf{poly}}$ is in SubEXP. Thus by time-hierarchy theorem, there is a language in EXP that is not in $\mathsf{AMPC}^{\mathsf{poly}}$. This also establishes that it is unlikely that PSPACE is in $\mathsf{AMPC}^{\mathsf{poly}}$ as this will imply PSPACE is a subset of SubEXP. Moreover, no NP-complete problem (under logspace reduction) is in $\mathsf{AMPC}^{\mathsf{poly}}$ unless $\mathsf{NP} \subseteq \mathsf{SubEXP}$.

Non-uniform Model. In the case of non-uniform AMPC computations, we can argue that any language accepted by a polynomial round AMPC algorithm can be simulated by a Boolean circuit of size $\mathsf{poly}(2^{N^\varepsilon})$. This is because every bit computed by a processor is a decision tree of size $O(2^{N^\varepsilon})$ and hence has a Boolean circuit of size $O(2^{N^\varepsilon})$. Since the number of bits written by all machines overall (polynomial) rounds is bounded by a polynomial, the size of the Boolean circuit simulating the whole computation is $\mathsf{poly}(2^{N^\varepsilon})$. It is known that there is a language L in $\mathsf{E}^{\Sigma_2^{\mathsf{P}}}$ that has maximum circuit complexity [14], it follows that L is not in non-uniform $\mathsf{AMPC}^{\mathsf{poly}}$. This lower bound establishes that showing PSPACE is in non-uniform $\mathsf{AMPC}^{\mathsf{poly}}$ is difficult as this will imply an unknown complexity theory separation that PSPACE is a proper subset of $\mathsf{EXP}^{\Sigma_2^{\mathsf{P}}}$.

References

1. E. Allender and K.-J. Lange. RUSPACE($\log n$) $\subseteq$ DSPACE($\log^2 n / \log\log n$). *Theory of Computing Systems*, 31(5):539–550, October 1998. doi:10.1007/s002240000102.
2. Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.
3. Alexandr Andoni, Clifford Stein, and Peilin Zhong. Log Diameter Rounds Algorithms for 2-Vertex and 2-Edge Connectivity. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 14:1–14:16, 2019.
4. Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, 2019. doi:10.1145/3293611.3331596.
5. Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *J. ACM*, 64(6), October 2017. doi:10.1145/3125644.

6 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab Mirrokni. Near-optimal massively parallel graph connectivity. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1615–1636, 2019. doi:10.1109/FOCS.2019.00095.

7 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab Mirrokni, and Warren Schudy. Massively parallel computation via remote memory access. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, pages 59–68, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3323165.3323208.

8 Gerhard Buntrock, Birgit Jenner, Klaus-Jörn Lange, and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In L. Budach, editor, *Fundamentals of Computation Theory*, pages 168–179, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

9 Moses Charikar, Weiyun Ma, and Li-Yang Tan. Unconditional lower bounds for adaptive massively parallel computation. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, pages 141–151, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3350755.3400230.

10 Benjamin Fish, Jeremy Kun, Ádám D. Lelkes, Lev Reyzin, and György Turán. On the computational complexity of mapreduce. In Yoram Moses, editor, *Distributed Computing*, pages 1–15, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

11 Brady Garvin, Derrick Stolee, Raghunath Tewari, and N. V. Vinodchandran. Reach-fewl = reachul. *computational complexity*, 23(1):85–98, March 2014. doi:10.1007/s00037-012-0050-8.

12 MohammadTaghi Hajiaghayi, Marina Knittel, Hamed Saleh, and Hsin-Hao Su. Adaptive Massively Parallel Constant-Round Tree Contraction. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 83:1–83:23, 2022. doi:10.4230/LIPIcs.ITCS.2022.83.

13 Klaus-Jörn Lange. An unambiguous class possessing a complete set. In Rüdiger Reischuk and Michel Morvan, editors, *STACS 97*, pages 339–350, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

14 Peter Bro Miltersen, N. V. Vinodchandran, and Osamu Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In Takao Asano, Hiroshi Imai, D. T. Lee, Shin-Ichi Nakano, and Takeshi Tokuyama, editors, *Computing and Combinatorics, 5th Annual International Conference, COCOON '99, Proceedings*, Lecture Notes in Computer Science, 1999.

15 Danupon Nanongkai and Michele Scquizzato. Equivalence classes and conditional hardness in massively parallel computations. *Distributed Computing*, 35(2):165–183, April 2022. doi:10.1007/s00446-021-00418-2.

16 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *J. ACM*, 65(6), November 2018. doi:10.1145/3232536.