LCNS, Vol 1178, pp. 83–192, Springer 1996

Recursively Divisible Problems

Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13,
D-72076 Tübingen, Fed. Rep. of Germany, niedermr@informatik.uni-tuebingen.de

Abstract. We introduce the concept of a (p, d)-*divisible* problem, where p reflects the number of processors and d the number of communication phases of a parallel algorithm solving the problem. We call problems that are *recursively* $(n^\epsilon, O(1))$-divisible in a work-optimal way with $0 < \epsilon < 1$ *ideally divisible* and give motivation drawn from parallel computing for the relevance of that concept. We show that several important problems are ideally divisible. For example, sorting is recursively $(n^{1/3}, 4)$-divisible in a work-optimal way. On the other hand, we also provide some results of lower bound type. For example, ideally divisible problems appear to be a proper subclass of the functional complexity class FP of sequentially feasible problems. Finally, we also give some extensions and variations of the concept of (p, d)-divisibility.

1 Introduction

There have been several approaches to more realistic models of parallel computation in recent time, which can basically be separated into two groups. First, it is tried to make the classical PRAM (Parallel Random Access Machine) model and the corresponding classes more realistic by restricting the model of computation or by defining "more precise" complexity classes [1,5,9,10,13]. Second, there is the approach to completely abandon the world of PRAM's and to work with a great number of parameters characterizing the performance of a parallel machine [3], thus giving up conceptual simplicity. This paper is closer to the first line of research mentioned above in the sense that we stick to conceptual simplicity without burdening our model by a number of system parameters. We present an abstract framework for parallel algorithms that respects several demands drawn from practice on the one hand, but still admits a model simple enough in order to perform (structural) complexity theory on the other hand. So we adopt, in a sense, a very restricted view of parallelism and want to analyze how far one can get when one only allows a small (ideally constant) number of communication phases that interrupt long phases of local, sequential computations.

There are several issues that can make a parallel algorithm valuable or valueless from the viewpoint of existing and foreseeable parallel machines. Two main aspects are speedup and efficiency (in a broad sense), leading to demands like work-optimality, good use of locality, consideration of latency and contention effects, easy synchronization and structured communication, small number of blocked communications, modularization, reasonable processor numbers, scalability and adaptability to increasing numbers of processors, and so on. Those

demands encourage the study of problems and classes dealing with the assumptions of significantly smaller numbers of processors than the input size is, a small number of local (i.e., sequential) computation phases interrupted by global communication phases and several other features, culminating in the concept of recursively (p,d)-divisible problems presented in Section 2. Note that the idea of communicating in phases also is important in the bulk-synchronous parallel model (BSP) [13]. The major point in the BSP model is that programs are written for v virtual parallel processors to run on p physical processors, where v is rather larger than p (e.g, $v = p \log p$). This "parallel slackness" is exploited by the compiler to schedule and pipeline computation and communication. It requires, however, quite a large administrative overhead. In our setting, by way of contrast, we design from the beginning p programs for p physical processors, thus avoiding the administrative overhead that arises with the "slackness principle" in the BSP. Moreover, we are mainly interested in algorithms with constantly bounded numbers of communication phases. Informally speaking, a problem is called (p,d)-divisible if it can be solved using p processors, each having some part of size $O(n/p)$ of the input, where only d phases of local computation and communication are sufficient. Recursive divisibility then means that the arising subproblems of size $O(n/p)$ again are (p,d)-divisible and so on. We call recursively $(n^\epsilon, O(1))$-divisible problems ideally divisible if they can be solved in a work-optimal way and show that well-known problems like prefix sums and sorting are ideally divisible. On the other hand, we reveal the inherent limitations of the concept of ideal divisibility and, more generally, recursive (p,d)-divisibility. For example, unless $P = NC$ ideally divisible problems are a proper subclass of FP and can be solved by NC-circuits of quasilinear size. Later we also provide extensions of our concept.

We assume familiarity with the basic concepts of parallel and sequential complexity theory. By FP we denote the functional version of P. All circuits shall follow standard uniformity conditions, namely $ATIME(\log n)$-uniformity [12].

2 The Basic Concepts

In the following definition, the intuition behind parameter w simply is that it shall represent the word size necessary to represent one of the n input elements in binary code. Usually, $w = O(\log n)$. Note that in this paper the terms "problem" and "function" are used interchangeably. We focus attention on sequentially feasible problems, that is, problems in FP.

Definition 1. Let $f \in FP$ with $f : \{0,1\}^{n \cdot w} \longrightarrow \{0,1\}^{c \cdot n \cdot w}$ for some positive constant c. Function f is (p,d)-*divisible* with natural numbers n, w, p, and d and $p < n$ if it can be written as follows. Let $x \in \{0,1\}^{n \cdot w}$. Then $f(x) = div(x,p,d)$, where for $d = 1$

$$div(x,p,1) := s_1^1(z_1^1(x)) \circ s_2^1(z_2^1(x)) \circ \ldots \circ s_p^1(z_p^1(x))$$

and for $d > 1$ and $y := div(x,p,d-1)$

$$div(x,p,d) := s_1^d(z_1^d(y)) \circ s_2^d(z_2^d(y)) \circ \ldots \circ s_p^d(z_p^d(y)).$$

Herein, "$\circ$" simply denotes concatenation of strings in $\{0,1\}^*$ and s_i^k and z_i^k with $1 \leq i \leq p$ and $1 \leq k \leq d$ are functions $z_i^k : \{0,1\}^{c_1^k \cdot n \cdot w} \longrightarrow \{0,1\}^{c_{2,i}^k \cdot n \cdot w/p}$ and $s_i^k : \{0,1\}^{c_{2,i}^k \cdot n \cdot w/p} \longrightarrow \{0,1\}^{c_{3,i}^k \cdot n \cdot w/p}$ where again c_1^k, $c_{2,i}^k$, and $c_{3,i}^k$ are some positive constants. In particular, we demand that each $s_i^k \in FP$ and that for each k, *all* z_i^k with $1 \leq i \leq p$ can be simultaneously computed by an $ATIME(\log n)$-uniform NC^0-circuit of size $O(n \cdot w)$.

Definition 1 presented the notion of (p,d)-divisible problem in a highly abstract, but mathematical precise way. The task of functions z_i^k is to do some kind of decomposition and functions s_i^k represent local, sequential computations. Informally speaking, we call a functional problem concerning n elements whose binary representation requires w bits each (p,d)-divisible, if it can be solved by the following algorithmic scheme. Assume that the given input is partitioned in some way into p sets of $O(n/p)$ input elements each. Repeat d times:

1. Apply to all inputs with $O(n/p)$ elements of word size w each a sequential algorithm that produces $O(n/p)$ output elements of word size w each.
2. By means of an NC^0-circuit of size $O(n \cdot w)$ redistribute the output data from step 1, creating $O(p)$ new instances of problems (not necessarily the same type as the original one) of $O(n/p)$ elements each or the final output.

Roughly speaking, step one belongs to functions s_i^k and step two belongs to functions z_i^k. For the sake of notational convenience, in the rest of the paper we, e.g., write $f : \{0,1\}^{n \cdot w} \longrightarrow \{0,1\}^{O(n \cdot w)}$, always standing for $f : \{0,1\}^{n \cdot w} \longrightarrow \{0,1\}^{c \cdot n \cdot w}$ for some arbitrary, but fixed positive constant c.

Definition 2. 1. We say that a problem is (p,d)-divisible *in a work-optimal way* if the time-processor-product of the parallel algorithm derived from the scheme given in Definition 1 is the same as the running time of the provably best sequential algorithm for that problem up to constant factors.
2. A (p,d)-divisible problem is *recursively* (p,d)*-divisible* if the size $O(n/p)$ instances produced above again are (recursively) (p,d)-divisible and so on, until we end up with instances of size $O(w)$.
3. Problems that are recursively (p,d)-divisible in a work-optimal way with $p = n^\epsilon$ for some $0 < \epsilon < 1$ and with $d = O(1)$ are *ideally divisible.*

It is important here to note that ideally divisible problems in general do *not* lead to algorithms that are work-optimal if the divisibility property is recursively applied a *non-constant* number of times. If a problem is (recursively) (p,d)-divisible in a work-optimal way, however, then there is a parallel algorithm using d communication and d local computation phases running in time $O(d \cdot s(n/p))$, where $s(n/p)$ denotes the maximum time to solve a size $O(n/p)$ problem instance sequentially.

Very recently, de la Torre and Kruskal [6] proposed a structural theory of recursively decomposable parallel processor networks. They study things like submachine locality basically on the network level. The seemingly tight connections to recursive divisibility, so to speak defined on the problem level, still

remain to be investigated in detail. Heywood and Ranka [9] presented so-called Hierarchical PRAM's (HPRAM's) as "a framework in which to study the extent that general locality can be exploited in parallel computing." In a sense, Heywood and Ranka also study the issue of independent phases of local computations, but from a more technical, programmer's point of view than that of parallel complexity theory as we adopt.

Proposition 3. *The computation of the maximum of n elements is recursively $(n^{i/(i+1)}, i+1)$-divisible in a work-optimal way for natural constant $i \geq 1$.*

Proof. (Sketch) The proof is an easy exercise. Start with $i = 1$. □

Proposition 3 exhibits the phenomenon that by using a larger number of processors, on the one hand, the running time gets reduced, but, on the other hand, the proportion of communication in the algorithm's overall complexity increases. This observation is generally neglected in the design of theoretically efficient algorithms and that is why they often might be of little use from the viewpoint of more realistic parallel computing. We end this section with some further justification for the relevance of the concepts we introduced.

- The requirement $d = O(1)$ means that we have only a constant number of communication (and, therefore, synchronization) phases.
- Demanding $p = n^{\epsilon}$ reflects massive parallelism (polynomial number of processors) as well as the fact that in practice the input size in general exceeds the number of processors by large [3].
- That (p, d)-divisibility has to hold recursively mirrors the scalability resp. adaptability of the algorithm.
- NC^0-circuits of linear size as laid down in Definition 1 represent the quest for efficient, data-independent, simple, and structured communications [8].
- Subproblems of size $O(n/p)$ guarantee efficient solutions having linearly bounded amounts of data to be transferred during each communication phase. In addition, this offers the opportunity for blocking of communications and serves to balance the distribution of data among the processors.

3 Some Ideally Divisible Problems

There are several important problems that are ideally divisible, perhaps the most prominent among them the sorting problem. The following theorem states that prefix sums is ideally divisible, possessing a parallel algorithm that runs in time $O((i+2)\, n^{1/(i+1)})$ using $n^{i/(i+1)}$ processors and $(i+2)$ communication phases for all $i \geq 1$. We omit the fairly straightforward proof.

Theorem 4. *For natural constant $i \geq 1$, prefix sums of n elements is recursively $(n^{i/(i+1)}, i+2)$-divisible in a work-optimal way.*

Corollary 5. *For natural constant $i \geq 1$, the recognition of regular languages and constant width branching programs both are recursively $(n^{i/(i+1)}, i+2)$-divisible in a work-optimal way.*

Proof. (Hint) We reduce both problems to prefix sums computations. □

Barrington [2] showed that width 5 polynomial size branching programs recognize exactly those languages in NC^1. So one could be tempted to assume that Corollary 5 also might lead to recursive $(n^{i/(i+1)}, i+2)$-divisibility for all problems in NC^1 with linear size circuits. Here the difficulty arises that the branching program simulation of depth d circuits needs branching program size 4^d [2]. So for $d = \log n$ we obtain quadratic size branching programs. Thus even assuming linear size NC^1-circuits, we end up with a polynomial blow-up in the size of the resulting branching programs. Thus it remains open whether all problems in (linear size) NC^1 are ideally divisible, because to apply Corollary 5, we need linear size branching programs.

Theorem 6. *The sorting of n elements is recursively $(n^{1/3}, 4)$-divisible in a work-optimal way.*

Proof. (Sketch) Leighton's Columnsort [11] adapted to our framework works as follows. For the time being assume that we have $n^{1/3}$ processors, each holding $n^{2/3}$ data items of the in total n items to be sorted. Now the algorithm roughly is as follows.

1. Each processor locally sorts its $n^{2/3}$ items.
2. Perform an all-to-all mapping where each processor gets $n^{1/3}$ items from each other one.
3. Again each processor locally sorts its $n^{2/3}$ (new) items.
4. Again perform an all-to-all mapping.
5. Locally sort the data items of each pair of neighboring processors.

Note that it suffices to know that an all-to-all mapping exchanges data between all processors in a very regular, fixed way. The correctness of the presented sorting algorithm in essence follows from [11]. We had to choose $p \leq n^{1/3}$ in order to be able to restrict ourselves to pairs of neighboring processors in the fifth, final step. From the above algorithm we get the $(n^{1/3}, 4)$-divisibility of sorting in the following way. The all-to-all mappings of steps 2 and 4 can clearly be done by some fixed data transports realized by linear size NC^0-circuits. The local sorts of steps 1 and 3 lead to the application of sequential sorting algorithms working on inputs of size $n^{2/3}$. Finally, the sorting of the items of neighboring processor pairs leads to two subsequent local phases of sequential sorting with inputs of size $2n^{2/3}$ each time. Altogether, four local sorting phases with input sizes $O(n^{2/3})$ suffice to get the whole size n input sorted. So the sorting problem is solved reducing it to smaller sorting problems. Altogether this implies that sorting is recursively $(n^{1/3}, 4)$-divisible. The described algorithm is work-optimal. □

It is an interesting question whether parameter $n^{1/3}$ in Theorem 6 can be improved to some $p = n^\epsilon$ with $\epsilon > 1/3$. Cypher and Sanz [4] developed the so-called *Cubesort* algorithm that, for $p = n^\epsilon$, requires fewer than $5(1/(1-\epsilon))^2$

rounds of local computation to sort. In particular, for $\epsilon = 1/3$ Cubesort needs 7 rounds, whereas Columnsort only needs 4 rounds as shown above. But for larger, constant ϵ Cubesort outperforms Columnsort and shows that sorting is recursively $(n^{i/(i+1)}, 5(i+1)^2)$-divisible in a work-optimal way. By way of contrast, (recursive) application of Columnsort only yields that sorting is recursively $(n^{i/(i+1)}, (i+1)^{\beta})$-divisible, where $\beta = 2/(\log 3 - 1) \approx 3.42$. It remains open, however, whether it is possible to deterministically sort in a work-optimal way with less than four communication phases using a polynomial number of processors.

4 Relations to Complexity Theory

In this section we present a circuit construction for recursively (n^{ϵ}, d)-divisible problems. This construction in particular reveals the limitations of the concept of ideally divisible problems—in the natural case of a logarithmically bounded word length they can be computed by quasilinear size NC-circuits and form a proper subclass of FP.

Definition 7. $RD_w(p, d)$ is the class of functional problems $f : \{0,1\}^{n \cdot w} \longrightarrow \{0,1\}^{O(n \cdot w)} \in FP$ that are recursively (p, d)-divisible.

Theorem 8.
$RD_w(n^{\epsilon}, d) \subseteq NC\text{-}SIZE, DEPTH(n \log^{O(1)} n \cdot w^{O(1)}, \log^{c} n \cdot w^{O(1)})$ *for* $c = \log d / \log(1/(1-\epsilon))$.

Proof. (Sketch) The basic idea of proof is to recursively apply the divisibility property in order to construct the desired circuit of bounded fan-in. If a problem is recursively (n^{ϵ}, d)-divisible, then according to Definitions 1 and 2 we can solve it by working on subproblems with inputs of size $bn^{1-\epsilon}$ for some constant factor $b \geq 1$. These subproblems again can be replaced by smaller ones, now of size $b^2 n^{(1-\epsilon)^2}$ and so on. Iterating this process x times leads to subproblems of size $b^x n^{(1-\epsilon)^x}$. For the time being we just assume that $b = 1$. Later on we will discuss the case $b > 1$. We iterate x times until a constant number of elements, each requiring w bits, is obtained, thus demanding

$$n^{(1-\epsilon)^x} = c, \qquad (*)$$

where c is some positive constant. The point here is that problem instances comprising only a constant number of elements can be solved by bounded fan-in circuits of size and depth $w^{O(1)}$ both, because the considered problems are in FP. Resolving equation $(*)$ for x yields

$$x = \log_{\frac{1}{1-\epsilon}}(\log n / \log c) = (\log\log n - \log\log c)/\log\frac{1}{1-\epsilon} = O(\log\log n).$$

Because in this way we need d^x layers of circuits requiring size and depth $w^{O(1)}$ and d^x layers of linear size NC^0-circuits (cf. Definition 1 and the discussion there), the depth of the resulting circuit is in essence determined by

$$d^x = 2^{x \log d} = 2^{\log d(\log\log n - O(1))/\log(1/(1-\epsilon))} \leq (\log n)^{\log d/\log(1/(1-\epsilon))}.$$

In total, we get the depth bounded by $d^x \cdot w^{O(1)}$.

Next we come to the general case $b > 1$. In the recursive construction, each subproblem of size $bn^{1-\epsilon}$ is replaced by $a(bn^{1-\epsilon})^\epsilon$ subproblems of size $b(bn^{1-\epsilon})^{1-\epsilon}$ and so on. This process continues x times, where x is as determined above. As can be easily verified, after x iterations we end up with problems each having

$$b^{\sum_{i=0}^{x-1}(1-\epsilon)^i} n^{(1-\epsilon)^x} = b^{(1-(1-\epsilon)^x)/\epsilon} n^{(1-\epsilon)^x}$$

elements instead of $n^{(1-\epsilon)^x}$ as in $(*)$. Clearly, $b^{(1-(1-\epsilon)^x)/\epsilon} = O(1)$ for constant $0 < \epsilon < 1$. Thus $(*)$ is still valid. So we get the same depth estimation as for $b = 1$.

So far we have analyzed the depth of the circuit. It remains to also determine its size. For the moment neglecting the NC^0-communication layers, by our construction we end up with d^x layers of subcircuits of size and depth $w^{O(1)}$ each. Let s denote the number of subcircuits (or, equivalently, subproblems) per each such layer. Then $O(w^{O(1)} \cdot d^x \cdot s)$ obviously is an upper bound for the size of the constructed circuit. We now determine s. We started with d layers, each layer consisting of $O(n^\epsilon)$ subproblems of size $O(n^{1-\epsilon})$ each. Replace $O(n^\epsilon)$ by an^ϵ and $O(n^{1-\epsilon})$ by $bn^{1-\epsilon}$ for some positive constants a and b. Altogether, when this process terminates after x recursive applications of divisibility, we have

$$s = an^\epsilon \cdot ab^{1-(1-\epsilon)^1} n^{(1-\epsilon)\epsilon} \cdot \ldots \cdot ab^{1-(1-\epsilon)^{x-1}} n^{(1-\epsilon)^{x-1}\epsilon}$$

subproblems of a constant number of elements per layer, each realizable by a size $w^{O(1)}$ circuit. It is easy to see that $s = a^x b^y n^z$, where $y \leq x - 1$ and $z = 1-(1-\epsilon)^x$. So we have $s \leq a^x b^{x-1} n^z$, which for $x = O(\log\log n)$ means that $s = n(\log n)^{O(1)}$. (Note that x was chosen such that $n^{(1-\epsilon)^x} = c$, so $n^z = O(n)$.) So far we still omitted the additional costs resulting from the layers of NC^0-communication. Because their size is linear and because we need $d^x = (\log n)^{O(1)}$ many of these layers, we altogether may conclude a size $n(\log n)^{O(1)} \cdot w^{O(1)}$ for the constructed circuit. To verify that the whole construction is $ATIME(\log n)$-uniform is straightforward. □

Applying Theorem 6 to the sorting problem demonstrates that there we can guarantee constants $a = 1$ and $b = 2$. Substituting this into the analysis done in the proof of Theorem 8 and making use of the recursive $(n^{1/3}, 4)$-divisibility as proved in Theorem 6, Theorem 8 now implies that sorting can be done by a depth $O((\log n)^{3.42})$ and size $O(n(\log n)^{5.13})$ NC-circuit that uses comparator gates with two w-bit-inputs. This reveals that ideally divisible problems do not necessarily lead to work-optimal parallel algorithms if the recursive implementation is iterated a non-constant number of times (compare the discussion following Definition 2). On the other hand, that also may indicate that even for non-work-optimal NC-algorithms, there may lie at their heart a work-optimal, practical parallel algorithm.

The following two direct corollaries of Theorem 8 exhibit the limitations of the concept of ideally divisible problems. Under the natural assumption of a logarithmically bounded word length w we obtain that all ideally divisible problems possess quasilinear size NC-circuits.

Corollary 9. $RD_{\log n}(n^\epsilon, O(1)) \subseteq NC\text{-}SIZE, DEPTH(n \log^{O(1)} n, \log^{O(1)} n)$.

Corollary 10. $RD_{\log n}(n^\epsilon, O(1))$ *is strictly contained in* FP.

Proof. (Hint) The circuit construction of Theorem 8 can also be transformed into a (recursive) sequential algorithm. □

Even if we allow a polylogarithmic instead of a constant number d of communication phases, unless $P = NC$ the hardest problems in FP still are resistant to efficient parallelization in the sense of recursive (p, d)-divisibility for sublinear, but still polynomial numbers p of processors. For the following note that $(\log n)^{O(\log \log n)}$ is subpolynomial. Again we omit the proof.

Theorem 11. $RD_{\log n}(n^\epsilon, (\log n)^{O(1)}) \subseteq$
$NC\text{-}SIZE, DEPTH(n(\log n)^{O(\log \log n)}, (\log n)^{O(\log \log n)})$.

Subsequently we consider the levels of the so-called NC-hierarchy. It is not known whether this hierarchy is a proper one.

Theorem 12. *If* $NC^k \subseteq RD_{\log n}(n^\epsilon, d)$ *for* $d \leq (\frac{1}{1-\epsilon})^{k-\delta'}$ *with arbitrary, but constant* $\delta' > \delta > 0$, *then* $NC = NC^{k-\delta}$. □

Proof. (Hint) Analogous to Theorem 8, but making use of the fact that the considered problems are in NC^k instead of "only" FP.

5 Extensions and Variations

In the preceding section, assuming a logarithmic word length, we showed that recursively $(n^\epsilon, O(1))$-divisible problems possess quasilinear size NC-circuits (Theorem 8, Corollary 9) and are properly contained in FP (Corollary 10). In particular, they can be solved with RAM's with unit cost measure in time $n \log^{O(1)} n$. In what follows we attack the problem to study and classify problems with sequential complexity $\Omega(n \log n)$. A central point here is to introduce the concept of *polynomial* (p, d)*-divisibility,* which is motivated by the matrix multiplication problem. Consider the problem of multiplying two $\sqrt{n} \times \sqrt{n}$-matrices. Having p processors at hand, how to compute the product? The probably simplest and notwithstanding a fairly prospective approach is to partition each of the two matrices into p submatrices and then to apply the standard matrix multiplication scheme already known from school. By this it is easy to show that multiplying two $\sqrt{n} \times \sqrt{n}$-matrices is recursively $(p, \sqrt{p})$-divisible for $p < n$. Thus for non-constant p we no longer have a constant number of communication phases, but need $\sqrt{p}$ many. But if we had $p\sqrt{p}$ processors, each dealing with inputs and outputs of size $O(n/p)$, then it is fairly easy to see that matrix multiplication could be done using only two local computation phases. This motivates the following definition of polynomial (p, d)-divisibility.

Definition 13. 1. Let $f \in FP$ with $f : \{0,1\}^{n \cdot w} \longrightarrow \{0,1\}^{O(n \cdot q \cdot w/p)}$, and let n, w, p, q, and d be natural numbers with $p < n$ and $q = p^{O(1)}$. Function f is *polynomially* (p,d)*-divisible* if it can be written as follows. Let $x \in \{0,1\}^{n \cdot w}$. Then $f(x) = div(x,q,d)$, where div is the same as in Definition 1 with the exceptions that the functions z_i^k, $1 \leq i \leq q$, there now are mappings from $\{0,1\}^{c_1^k \cdot n \cdot q \cdot w/p}$ to $\{0,1\}^{c_{i,2}^k \cdot n \cdot w/p}$ and for all k, *all* z_i^k have to be simultaneously computable by an $ATIME(\log n)$-uniform NC^0-circuit of size $O(n \cdot q \cdot w/p)$.

2. $RPD_w(p,d)$ is the class of problems $f : \{0,1\}^{n \cdot w} \longrightarrow \{0,1\}^{O(n \cdot q \cdot w/p)} \in FP$, $q = O(1)$, that are recursively polynomially (p,d)-divisible. Here "recursively" is defined as for (p,d)-divisible problems (see Definition 2).

Roughly speaking, the only difference comparing the above with Definition 1 of (p,d)-divisible problems is that we replace p by $q = p^{O(1)}$, a polynomial increase of the number of processors. Under this new definition it is possible to show that matrix multiplication is recursively polynomially $(p,2)$-divisible for $p \leq \sqrt{n}$. Even if we allow recursive polynomial $(n^\epsilon, O(1))$-divisibility, not all problems in FP do have this property unless $P = NC$. The proof of Theorem 14 is similar to that of Theorem 8 and is omitted, therefore.

Theorem 14.
$RPD_w(n^\epsilon, d) \subseteq NC\text{-}SIZE, DEPTH(n^{O(1)} \cdot w^{O(1)}, \log^c n \cdot w^{O(1)})$, *where* $c = \log d / \log(1/(1-\epsilon))$.

A comparison of Theorem 8 and Theorem 14 reveals that the difference between recursively (n^ϵ, d)-divisible and recursively *polynomially* (n^ϵ, d)-divisible problems is that, assuming $w = O(\log n)$, for the first ones we can construct quasilinear size circuits, whereas the latter ones require circuits of polynomial size. Using Theorem 14 we now may obtain results in analogy to that in the previous section. Perhaps the most important case is the statement concerning the relationship between FP and NC, which we pick out at this place.

Corollary 15. *If* $FP \subseteq RPD_{\log n}(n^\epsilon, O(1))$, *then* $FP = NC$.

6 Conclusion

Several open questions arise from our work, for example: Can the result for sorting (cf. Theorem 6 and the subsequent discussion) be improved? Is it possible to show that the well-known list ranking problem is not recursively $(n^\epsilon, O(1))$-divisible? Does there exist a characterization of exactly those problems in NC that are (polynomially) $(n^\epsilon, O(1))$-divisible?

This paper provides a link between "classical" (parallel) complexity theory (like NC-theory) and questions of parallel computing (like constant number of communication phases, blocked and structured communications, reuse of sequential algorithms). Whereas Vitter and Simons [14] demonstrated that there exist work-optimal PRAM algorithms for P-complete problems, we showed that there

exist problems in *FP* that are inherently resistant to good divisibility properties. To an extent this (in contrast to Vitter and Simons) underpins the importance of the P versus NC debate [7] when considering a more restrictive framework for parallel computation (as we proposed) than the world of PRAM algorithms is.

References

1. A. Aggarwal, A. K. Chandra, and M. Snir. Communication Complexity of PRAMs. *Theoretical Comput. Sci.*, 71:3–28, 1990.
2. D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38:150–164, 1989.
3. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, May 1993.
4. R. Cypher and J. L. Sanz. Cubesort: A parallel algorithm for sorting N data items with S–sorters. *Journal of Algorithms*, 13:211–234, 1992.
5. P. de la Torre and C. P. Kruskal. Towards a single model of efficient computation in real parallel machines. *Future Generation Computer Systems*, 8:395–408, 1992.
6. P. de la Torre and C. P. Kruskal. A structural theory of recursively decomposable parallel processor-networks. In *IEEE Symp. on Parallel and Distributed Processing*, 1995.
7. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.
8. T. Heywood and C. Leopold. Models of parallelism. In J. R. Davy and P. M. Dew, editors, *Abstract Machine Models for Highly Parallel Computers*, chapter 1, pages 1–16. Oxford University Press, 1995.
9. T. Heywood and S. Ranka. A practical hierarchical model of parallel computation: I and II. *Journal of Parallel and Distributed Computing*, 16:212–249, November 1992.
10. C. P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theoretical Comput. Sci.*, 71:95–132, 1990.
11. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34(4):344–354, April 1985.
12. W. L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22:365–383, 1981.
13. L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.
14. J. S. Vitter and R. A. Simons. New classes for parallel complexity: A study of unification and other complete problems for P. *IEEE Trans. Comp.*, C-35(5):403–418, 1986.