On the Parameterized Parallel Complexity and the Vertex Cover Problem

Faisal N. Abu-Khzam[1,4], Shouwei Li[2(✉)], Christine Markarian[3], Friedhelm Meyer auf der Heide[2], and Pavel Podlipyan[2]

[1] Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon
[2] Heinz Nixdorf Institute & Department of Computer Science, Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany
sli@mail.uni-paderborn.de
[3] Department of Mathematical Sciences, Haigazian University, Beirut, Lebanon
[4] School of Engineering and Information Technology, Charles Darwin University, Darwin, Australia

Abstract. Efficiently parallelizable parameterized problems have been classified as being either in the class FPP (*fixed-parameter parallelizable*) or the class PNC (*parameterized analog of NC*), which contains FPP as a subclass. In this paper, we propose a more restrictive class of parallelizable parameterized problems called *fixed-parameter parallel-tractable* (FPPT). For a problem to be in FPPT, it should possess an efficient parallel algorithm not only from a theoretical standpoint but in practice as well. The primary distinction between FPPT and FPP is the parallel processor utilization, which is bounded by a polynomial function in the case of FPPT. We initiate the study of FPPT with the well-known k-vertex cover problem. In particular, we present a parallel algorithm that outperforms the best known parallel algorithm for this problem: using $\mathcal{O}(m)$ instead of $\mathcal{O}(n^2)$ parallel processors, the running time improves from $4\log n + \mathcal{O}(k^k)$ to $\mathcal{O}(k \cdot \log^3 n)$, where m is the number of edges, n is the number of vertices of the input graph, and k is an upper bound of the size of the sought vertex cover. We also note that a few P-complete problems fall into FPPT including the monotone circuit value problem (MCV) when the underlying graphs are bounded by a constant Euler genus.

1 Introduction

The area of *Parameterized Complexity* has witnessed tremendous growth in the last two decades and has become a central research area in theoretical computer science. A problem is *fixed-parameter tractable* (FPT) if it has an algorithm that runs in time $\mathcal{O}(f(k) \cdot n^{\mathcal{O}(1)})$, where n is the problem size and k is the input

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and the International Graduate School "Dynamic Intelligent Systems".

T-H.H. Chan et al. (Eds.): COCOA 2016, LNCS 10043, pp. 477–488, 2016.
DOI: 10.1007/978-3-319-48749-6_35

parameter that is independent of n, for an arbitrary computable function f. A typical, well-known, example is the k-vertex cover problem which is solvable in time $\mathcal{O}(kn + 1.274^k)$, where n is the number of vertices of the input graph and k is an upper bound of the size of the sought vertex cover [6]. Numerous other NP-complete problems also fall into the class FPT.

The study of parameterized complexity has been extended to parallel computing and this is broadly known as *parameterized parallel complexity.* The first systematic work on parameterized parallel complexity appeared in [5], where the authors introduced two classes to characterize the efficiently parallelizable parameterized problems, according to the degree of efficiency required, known as *parameterized analog of NC* (PNC) and *fixed-parameter parallelizable* (FPP), respectively. The class PNC contains all parameterized problems that have a parallel deterministic algorithm running in time $\mathcal{O}(f(k) \cdot (\log n)^{h(k)})$ using $\mathcal{O}(g(k) \cdot n^{\beta})$ parallel processors, where n is the input size, k is the parameter, f, g, and h are arbitrary computable functions, and β is a constant independent of n and k. The class FPP contains all parameterized problems that have a parallel deterministic algorithm running in time $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(g(k) \cdot n^{\beta})$ parallel processors, where n is the input size, k is the parameter, f and g are arbitrary computable functions, and α, β are constants independent of n and k. They also proposed a FPP algorithm that solves the k-vertex cover problem and proved that all problems involving *MS* (definable in monadic second-order logic with quantifications over vertex and edge sets) or *EMS properties* (that involve counting or summing evaluations over sets definable in monadic second-order logic) are in FPP when restricted to graphs of bounded *treewidth.*

Our Contribution. In this paper, we propose a new class of efficiently parallelizable parameterized problems called *fixed-parameter parallel-tractable* (FPPT). FPPT has several advantages over PNC and FPP. It eliminates the heavy exponent that depends on the parameter k in the logarithm of PNC. Moreover, it gets rid of the arbitrary function g, which does not seem to lead to a parallel algorithm that is efficient from a practical point of view of processor utilization. Thus, a problem in FPPT should possess an efficient parallel algorithm, not only from a theoretical standpoint but also in practice. We initiate the study of FPPT with the well-known k-vertex cover problem for which we propose a parallel algorithm and show that it belongs to FPPT. We also note that the classical monotone circuit value problem (MCV) belongs to FPPT when the underlying graph is bounded to a constant Euler genus.

Our proposed algorithm for the k-vertex cover problem outperforms the best known parallel algorithm for this problem, which appeared in [5]. The number of parallel processors is $\mathcal{O}(m)$ instead of $\mathcal{O}(n^2)$ and the running time improves from $4 \log n + \mathcal{O}(k^k)$ to $\mathcal{O}(k \cdot \log^3 n)$, where m is the number of edges, n is the number of vertices of the input graph, and k is an upper bound of the size of the sought vertex cover. Our algorithm employs kernelization using the *Crown Reduction* method of [2]. As for the MCV problem, we have recently presented a parallel algorithm when the Euler genus of the underlying graph is bounded by the parameter k [3]. The algorithm runs in time $\mathcal{O}((k + 1) \cdot \log^3 n)$

using $\mathcal{O}(n^c)$ parallel processors, where $\mathcal{O}(n^c)$ is the best processor boundary for parallel matrix multiplication [19].

2 Preliminaries

Given an undirected graph $G = (V, E)$, the *k-vertex cover* problem asks whether there is a set of vertices $V' \subseteq V$ of size at most k (k is assumed to be fixed), such that for every edge $(u, v) \in E$, at least one of its endpoints, u or v is in V'. In other words, the complement of V' is an independent set (i.e., a set that induces an edge-less subgraph).

A *matching* M is a subset of E such that no two edges in M share a common vertex. A vertex that is incident to an element of M is said to be matched under M. A matching is maximal if it is not contained in a larger matching. Such a matching is *maximum* if no matching of larger cardinality exists. If all vertices are matched under a matching, then it is a *perfect matching*. The problem of finding a maximum matching or a perfect matching, even in planar graphs, has received considerable attention in the field of parallel computation.

Merging a subset V' of the vertices consists of replacing all the vertices in V' with a single vertex w such that $N(w) = \bigcup_{v \in V'} N(v) \backslash V'$.

A *crown* in a graph G is an ordered pair (I, H) of subsets of V that satisfies the following criteria:

- $I \neq \emptyset$ is an independent set of G,
- $H = N(I)$, and
- there exists a matching M on the edges connecting I and H such that all elements of H are matched under M.

H is called the *head* of the crown. The *width* of the crown is $|H|$. A *straight crown* is a crown (I, H) that satisfies the condition $|I| = |H|$. A *flared crown* is a crown (I, H) that satisfies condition $|I| > |H|$. These notions are depicted in Fig. 1.

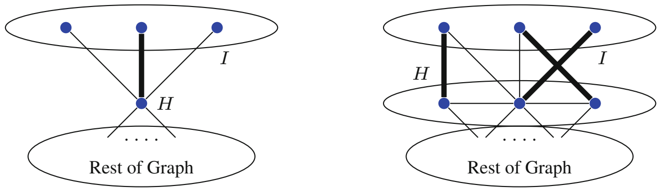

Fig. 1. Sample crowns (Bold edges denote a matching)

The following theorem was proved in the early work on crown decomposition.

Theorem 1 **([1,7]).** *If G is a graph with a crown (I, H), then there is a vertex cover of G of minimum size that contains all the vertices in H and none of the vertices in I.*

Kernelization is a polynomial-time transformation that reduces an arbitrary instance (I, k) of a parameterized problem to an equivalent instance (I', k'), such that $k' \leq k$ and $|I'|$ is bounded by some function of k. The resulting instance is often referred to as a problem kernel of the instance. It is kernelization that distinguishes an arbitrary problem from one that is FPT. More specifically, a problem is FPT if and only if it has a kernelization algorithm [11].

3 FPPT Kernelization by Crown Reduction

The classes PNC and FPP introduced by Cesati and Ianni in [5] fail to capture some important aspects mainly caused by the "arbitrary" computable functions h and g bounding the running time and the processor utilization. For example, the exponent in the logarithm of PNC heavily depends on the parameter k. It grows at a rapid rate thus making the running time very close to a linear function even for not too large values of k. Moreover, according to its definition, g is allowed to be a super-polynomial function of the parameter k, which makes the processor utilization unreasonable even for relatively small values of k. Since the number of available processors is a strong physical constraint when employing a parallel algorithm, membership in PNC and FPP can sometimes be relevant mostly for theoretical purposes.

Motivated by the above, we refine the classes PNC and FPP and propose a new class called *fixed-parameter parallel-tractable* (FPPT). The main purpose is to have a better characterization of solvable fixed-parameter parallelizable problems at least from a practical standpoint. We formally define FPPT as follows:

Definition 1 (FPPT). The class of *fixed-parameter parallel-tractable* (FPPT) contains all parameterized problems that have a parallel deterministic algorithm whose running time is in $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(n^{\beta})$ parallel processors, where n is the size of input, k is the parameter, f is an arbitrary computable function, and α, β are constants independent of n and k.

The Lemma below shows the relation between FPT and PNC, given by Cesati and Ianni.

Lemma 1 ([5]). *PNC is a subset of FPT.*

It is then easy to observe that FPP $\subseteq$ PNC. Hence, we conclude that:

$$FPPT \subseteq FPP \subseteq PNC \subseteq FPT.$$

In the following, we present an FPPT kernelization algorithm for the k-vertex cover problem, which is based on a crown decomposition of the input graph in [2].

Algorithm 1. Parallel algorithm for finding a crown.

```
procedure ParallelCrown($G$)
    Step 1: Find a maximal matching $M_1$ in parallel and identify the set of all
        unmatched vertices as the set $O$ of outsiders.
    Step 2: Find a maximum auxiliary matching $M_2$ of the edges between $O$ and
        $N(O)$ in parallel.
    Step 3: If every vertex in $N(O)$ is matched by $M_2$, then $H = N(O)$ and $I = O$
        form a straight crown, and we are done.
    Step 4: Let $I_0$ be the set of vertices in $O$ that are unmatched by $M_2$.
    Step 5: Repeat Steps 5a and 5b until $n = N$ so that $I_{N-1} = I_N$.
        5a. Let $H_n = N(I_n)$.
        5b. Let $I_{n+1} = I_n \cup N_{M_2}(H_n)$.
    Step 6: $I = I_N$ and $H = H_N$ form a flared crown.
end procedure
```

It has been shown in [7] that, if both matchings M_1 and M_2 are of a size less than or equal to k, then G has at most $3k$ vertices that are not in the crown. So we only analyze the running time and processor utilization here. Apparently, the most expensive part of this procedure are Steps 1 and 2.

Israeli and Shiloach presented a parallel algorithm for maximal matching. The performance of their algorithm is given by the following lemma:

Lemma 2 ([16])**.** *A maximal matching in general graphs can be found in time $\mathcal{O}(\log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges and n is the number of vertices of the input graph.*

So we only need to show that the maximum matching M_2 in Step 2 can be constructed (efficiently) in parallel, which will be discussed in Sect. 4 where we prove that the construction of such a maximum matching is in FPPT. Therefore we obtain the following.

Theorem 2. *Vertex cover kernelization via crown reduction is in FPPT.*

Consequently, and since crown reduction guarantees a $3k$-kernel, we can solve the reduced instance of vertex cover in $\mathcal{O}(f(k))$-time. This finishes the proof of our main result.

Theorem 3. *The k-vertex cover problem is in FPPT.*

4 Parameterized Maximum Matching

In this section, let us consider a parameterized version of the maximum matching problem stated below, as a subroutine of the parallel crown reduction procedure. Note that, in our definition of the parameterized maximum matching problem, we ask the question whether the cardinality of the maximum matching is equal to or less than parameter k. (This is to be contrasted with the usual parameterization of a maximization problem where one would ask for a solution of size k or more).

Input:	A graph $G = (V, E)$ and a positive integer k.
Problem:	Is there a maximum matching M of size at most k? If yes, how to construct one?

To the best of our knowledge, this is the first time that the maximum matching problem is studied with an input parameter given, especially in a way that upper-bounds the size of the sought matching. In classical computational complexity, the maximum matching problem has the same complexity as the perfect matching problem, since the former can be easily reduced to the latter in logarithmic space. Suppose we want to check whether there is a matching with k edges in G. If such a matching existed, $2k$ vertices would have been matched and $n-2k$ would have been "free." We add $n-2k$ new vertices to G and create edges between these new vertices and all old vertices in G in order to obtain a new graph G'. Thus, G' has a perfect matching if and only if G has a matching with exactly k edges. Conversely, perfect matching parameterized by the number of matching edges is trivially FPPT by a simple reduction to our version of parameterized maximum matching (if $n \neq 2k$ then it is a no instance). So the two problems have the same parameterized parallel complexity with respect to the parameter k. However, when the parameter is the number of perfect matchings, perfect matching falls in the class FPPT, more precisely in NC in this case [4], while it is not known yet whether maximum matching has an NC algorithm when the number of maximum matchings is bounded by a polynomial in n. This problem is open at this stage. The related studies on perfect matching, including NC algorithms for some special structures of graphs or parameters, such as: dense graphs [9], regular bipartite graphs [18], strongly chordal graphs [10], claw-free graphs [8], bipartite graphs with polynomially bounded number of perfect matchings [15], general graphs with polynomially bounded number of perfect matchings [4], bipartite planar and small genus graphs [20].

Suppose G is a graph with minimum vertex cover bounded by a constant k. It is not hard to see that the maximum matching of G must be bounded to k as well (since the edges in a maximum matching are pairwise disjoint, at least one end of each edge should be included in the minimum vertex cover). Therefore, in order to cover all edges in a maximum matching, the minimum vertex cover must be greater than or equal to the size of a maximum matching.

Our algorithm is based on the augmenting path approach used in the classical *Blossoms* algorithm. We show that, with careful analysis, the parameterized maximum matching problem can be solved in time $\mathcal{O}(f(k) \cdot \log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges and n is the number of vertices of the input graph. The algorithm is summarized in Algorithm 2.

Algorithm 2. Parallel algorithm for parameterized maximum matching

```
1: procedure FindMaximumMatching($G$,$k$)            ▷ Graph $G$ and parameter $k$
2:     Step 1: Find a maximal matching $M_1$ in parallel. If $|M_1| > k$, return false.
3:     Step 2: Construct a new graph by merging the unmatched vertices into a new
           vertex $S$.
4:     Step 3: Construct an alternating BFS tree rooted at $S$ in parallel.
5:     Step 4: Either find an augmenting path with respect to $M_1$, or construct a new
           graph with a maximum matching of cardinality $k - 1$.
6:     Step 5: Repeat Step 1 to Step 4 at most $k$ rounds.
7: end procedure
```

It is well known that the size of any maximal matching is at least half the size of a maximum matching. Therefore, we note that the maximal matching produced in Step 1 is not less than $k/2$.

In Step 2, we make a transformation as follows. We merge all unmatched vertices into a single vertex S so that all edges connected to the unmatched vertices become incident to S, thus making all augmenting paths in the graph G transfer into an odd alternating cycle in the new graph.

Next, we construct an alternating BFS tree with the following properties:

- S is the root (and layer 0);
- All vertices adjacent to S are in layer 1;
- All edges from odd layers to even layers are matching edges (elements of the matching M_1 constructed in Step 1).

Note that we also add the unmatched edges into the alternating BFS tree in Fig. 2 to help in the analysis. This will be made clear in the sequel.

The parallel alternating BFS tree can be constructed in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^3)$ parallel processors [13]. However, the graph for which the BFS is constructed consists of at most $2k + 1$ vertices because all unmatched vertices are merged into S. Thus, the running time will be $\mathcal{O}(\log^2 k)$ with $\mathcal{O}(k^3)$ parallel processors. Consequently, we observe the following:

- All vertices in the tree are matched except for S. All edges between layer 0 and layer 1 are unmatched, otherwise S will be matched.
- If the nodes in layer 1 have no descendants (i.e., neighbors in a layer of a higher index), then they must be matched in this layer (e.g., b and c in Fig. 2). Otherwise, these nodes would be unmatched. We call such an edge type B edge. The unmatched edges in the same layer are also type B edges.
- The unmatched edges from odd layers to descendant layers are of type A (e.g., (d, f) and (a, k) in Fig. 2).
- The unmatched edges from even layers to the adjacent odd layer are of type C (e.g., (h, j) in Fig. 2).
- The depth of the tree is at most $2k$ because the size of the maximum matching is bounded by k.

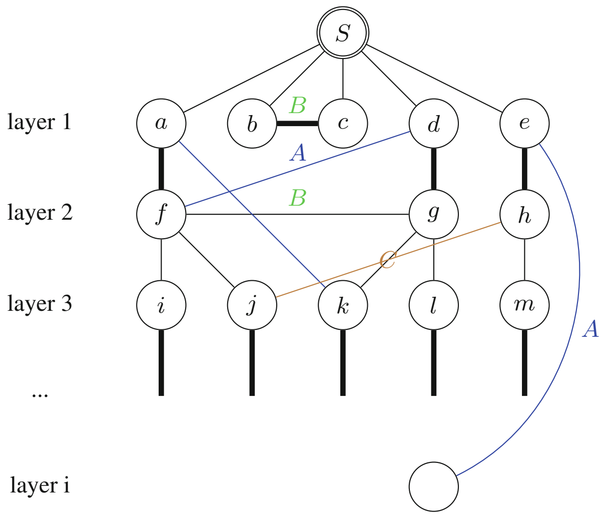

Fig. 2. Alternating BFS tree. (Bold edges denote matching edges)

Lemma 3. *If M_1 is not a maximum matching, then every alternating odd cycle must pass through at least one type B edge.*

Proof. Suppose there is an augmenting path $p = e_1, e_2, \ldots, e_h$ that consists of type A, type C, and matching edges. According to the observation above, the initial vertex of p must be S and e_1 must be an unmatched edge between layer 0 and layer 1 (e.g., (S, e) in Fig. 2). For p to be an augmenting path, e_2 must be a matched edge. Since it is not allowed to take type B edges, we have to go further down (e.g., (e, h) in Fig. 2). Then, e_3 must be a type C edge (e.g., (h, j) in Fig. 2). Because any type C edge starts from an even layer towards an adjacent odd layer, it is impossible to construct an augmenting path. $\square$

Since each augmenting path transforms into an alternating odd cycle and increases the matching by 1, we only need to check if there are type B edges in the alternating BFS tree. If yes, we proceed by checking whether the alternating odd cycle comes from a valid augmenting path. Otherwise, we either get a maximum matching with cardinality at most k or reduce the graph to a new graph with the maximum matching of size bounded by $k - 1$. Now, we analyze this procedure by considering the following two cases:

Case 1: Suppose an alternating odd cycle resulted from an augmenting path is $(s - a - b - s)$ and the edge (a, b) is matched. a, b connect to different vertices u_1 and u_2 in S. It is easy to find an augmenting path $u_1 - a - b - u_2$. We can argue the same way if the edge (a, b) is an unmatched type B edge. Refer to Fig. 3 for an illustration.

Case 2: Suppose that an alternating odd cycle transformed from an augmenting path is $(u - a - b - u)$ and the edge (a, b) is matched. Also a and b connect to the

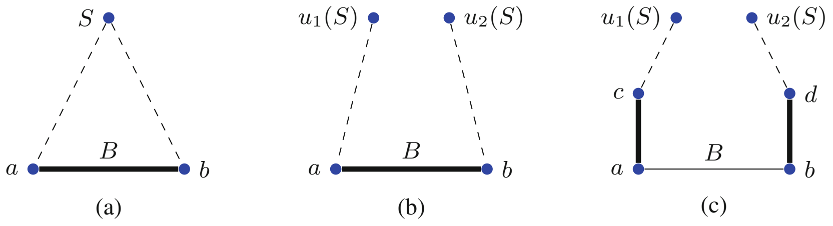

Fig. 3. An alternating odd cycle passes through type B edges. (a) An alternating odd cycle; (b) (a, b) is a matched type B edge; (c) (a, b) is an unmatched type B edge.

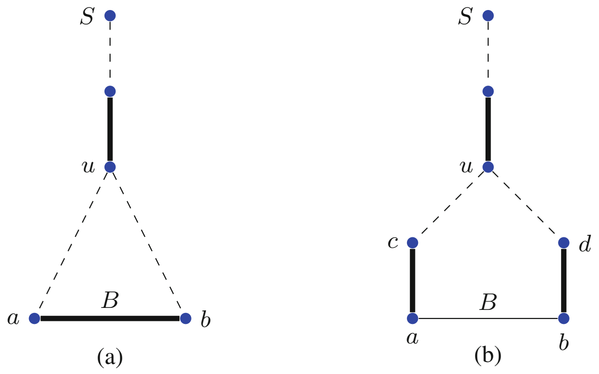

Fig. 4. Blossom structure.

same vertices u and there is a alternating path from S to u. We do not know if there exists an augmenting path, neither do we know how to find one, if there is any. For this case, the alternating odd cycle is called a blossom structure. Refer to Fig. 4 for an illustration. Since a blossom contains at least one matched edge and three nodes, we can shrink the blossom to a single vertex and the new graph obtained will have a maximum matching of cardinality at most $k - 1$. Since the size of the maximum matching is bounded by k, at most k rounds are needed to construct a maximum matching.

Thus, the running time will be $\mathcal{O}(k \cdot \log^3 n)$ with $\mathcal{O}(m)$ parallel processors, where m is the number of edges, n is the number of vertices of the input graph, and k is an upper bound of the size of the sought maximum matching. Given all of the above, we can now state Theorem 4.

Theorem 4. *Maximum matching parameterized by an upper bound on the matching size, is in FPPT.*

5 MCV with Bounded Genus Is in FPPT

In this section, we note that the problem known as the monotone circuit value (MCV) problem with bounded genus is also in FPPT.

A *Boolean Circuit* is a directed acyclic graph consisting of NOT, AND, and OR gates. The *circuit value problem* (CVP) is the problem of evaluating a boolean circuit on a given input. CVP was shown to be P-complete with respect to logarithmic space reductions in [17]. Some other restricted variants of CVP have also been studied. The *planar circuit value* (PCV) problem, for example, is a variant of CVP in which the underlying graph of the circuit is planar. Another variant is the *monotone circuit value* (MCV) problem in which the circuit only has AND and OR gates. Both the PCV and MCV problem were shown to be P-complete in [14]. Another variant in which the circuit is simultaneously planar and monotone is known as the PMCV problem. PMCV problem was shown to be in NC, and its first NC algorithm was given in [23]. Recently, we proposed a parallel algorithm for a variant of the general MCV problem in which the underlying graph bounded by a constant Euler genus k in [3]. The main result can be stated as Theorem 5.

Theorem 5. *Given a general monotone boolean circuit with n gates and an underlying graph bounded by a constant Euler genus k, the circuit can be evaluated in time $\mathcal{O}((k+1)\cdot\log^3 n)$ using $\mathcal{O}(n^c)$ parallel processors, where $\mathcal{O}(n^c)$ is the best processor boundary for parallel matrix multiplication.*

Hence, we deduce that the monotone circuit value problem with bounded Euler genus k is in FPPT.

6 Concluding Remarks and Future Work

In this paper, we initiated the study of a new class of efficiently parallelizable parameterized problems called fixed-parameter parallel-tractable (FPPT). A problem in FPPT, unlike one in PNC or FPP, must be solved by a polynomial number of processors independent of the parameter k. Hence it should possess an efficient parallel algorithm not only from a theoretical standpoint but also in practice.

We reconsidered the k-vertex cover problem and noted that it falls in FPPT due to the algorithm of Cesati and Ianni in [5]. Our improved algorithm is based on obtaining a quadratic kernel. We showed that obtaining a linear kernel for the problem is also in FPPT by proving that constructing a corresponding crown decomposition is in FPPT as well. We believe this will lead to proving other problems also, where a kernel is obtained via a crown reduction, are in FPPT.

Furthermore, we conclude that FPPT is somehow orthogonal to the NP-complete and P-complete classes in the sense that some, but not all, problems from each of these classes fall in FPPT. We further raise some open questions.

- At this stage, the first obvious question is which other problems belong to FPPT.
- Numerous NP-complete problems fall into the class FPT, but clearly not all of them do. It was shown that there is a $W[*]$ hierarchy in the NP class where

FPT = $W[0]$. With the introduction of parameterization into the field of parallel computing, some NP-complete and P-complete problems were shown to have a fixed-parameter parallel algorithm by fixing one (or more) parameter(s). These include the vertex cover problem, the graph genus problem [12], and the monotone circuit value problem, which all fall into FPPT. Now the question is what happens if we restrict our attention to P-complete problems: could it be that the P class also has a hierarchy, say $Z[*]$, analogous to $W[*]$ in the class NP, where FPPT = $Z[0]$? In fact, we believe such a hierarchy should exist because the following is true. The MCV problem and the NAND circuit value problem [22] are both P-complete problems. The first is in FPPT while the second is not, taking the graph genus as a parameter. Another conceivable example is the lexicographically first maximal subgraph problem (LFMIS). Miyano showed that the latter is P-complete even for bipartite graphs with bounded degree at most 3 in [21]. It would be interesting, and probably not too difficult, to obtain hardness results showing a problem cannot belong to FPPT unless some new parameterized parallel-complexity hierarchy collapses.
- Furthermore, there has been an interest in the so-called "gradually intractable problems" for the class NP. The question here is whether "gradually *unparallelizable* problems" can be analogously investigated in the class of P-complete. We suggest involving one or more parameters to characterize the "gradually" procedure. This would be helpful to understand the intrinsic difficulty of P-complete problems and to answer the question of whether P = NC, which is one of the main motivations behind this paper.

Acknowledgments. We wish to thank the anonymous referees for their valuable comments to improve the quality and presentation of this paper.

References

1. Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem,: theory and experiments. In: Arge, L., Italiano, G.F., Sedgewick, R. (eds.) Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, 10 January 2004, pp. 62–69. SIAM (2004)
2. Abu-Khzam, F.N., Fellows, M.R., Langston, M.A., Suters, W.H.: Crown structures for vertex cover kernelization. Theory Comput. Syst. **41**(3), 411–430 (2007)
3. Abu-Khzam, F.N., Li, S., Markarian, C., Meyer auf der Heide, F., Podlipyan, P.: The monotone circuit value problem with bounded genus is in NC. In: Dinh, T.N., Thai, M.T. (eds.) COCOON 2016. LNCS, vol. 9797, pp. 92–102. Springer, Heidelberg (2016). doi:10.1007/978-3-319-42634-1_8
4. Agrawal, M., Hoang, T.M., Thierauf, T.: The polynomially bounded perfect matching problem is in $\mathbf{NC}^2$. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 489–499. Springer, Heidelberg (2007)
5. Cesati, M., Di Ianni, M.: Parameterized parallel complexity. In: Pritchard, D., Reeve, J.S. (eds.) Euro-Par 1998. LNCS, vol. 1470, pp. 892–896. Springer, Heidelberg (1998)

6. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. **411**(40), 3736–3756 (2010)
7. Chor, B., Fellows, M., Juedes, D.W.: Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 257–269. Springer, Heidelberg (2004)
8. Chrobak, M., Naor, J., Novick, M.B.: Using bounded degree spanning trees in the design of efficient algorithms on claw-free graphs. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1989. LNCS, vol. 382, pp. 147–162. Springer, Heidelberg (1989)
9. Dahlhaus, E., Hajnal, P., Karpinski, M.: On the parallel complexity of hamiltonian cycle and matching problem on dense graphs. J. Algorithms **15**(3), 367–384 (1993)
10. Dahlhaus, E., Karpinski, M.: The matching problem for strongly chordal graphs is in NC. Inst. für Informatik, Univ. (1986)
11. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity, vol. 4. Springer, Heidelberg (2013)
12. Elberfeld, M., Kawarabayashi, K.-I.: Embedding and canonizing graphs of bounded genus in logspace. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, pp. 383–392. ACM (2014)
13. Ghosh, R.K., Bhattacharjee, G.: Parallel breadth-first search algorithms for trees and graphs. Int. J. Comput. Math. **15**(1–4), 255–268 (1984)
14. Goldschlager, L.M.: The monotone and planar circuit value problems are log space complete for P. SIGACT News **9**(2), 25–29 (1977)
15. Grigoriev, D.Y., Karpinski, M.: The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In: 28th Annual Symposium on Foundations of Computer Science, 1987, pp. 166–172. IEEE (1987)
16. Israeli, A., Shiloach, Y.: An improved parallel algorithm for maximal matching. Inf. Process. Lett. **22**(2), 57–60 (1986)
17. Ladner, R.E.: The circuit value problem is log space complete for P. SIGACT News **7**(1), 18–20 (1975)
18. Lev, G.F., Pippenger, N., Valiant, L.G.: A fast parallel algorithm for routing in permutation networks. IEEE Trans. Comput. **100**(2), 93–100 (1981)
19. Li, K., Pan, V.Y.: Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system. IEEE Trans. Comput. **50**(5), 519–525 (2001)
20. Mahajan, M., Varadarajan, K.R.: A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, pp. 351–357. ACM (2000)
21. Miyano, S.: The lexicographically first maximal subgraph problems: P-completeness and NC algorithms. Math. Syst. Theory **22**(1), 47–73 (1989)
22. Post, E.L.: The Two-Valued Iterative Systems of Mathematical Logic. (AM-5), vol. 5. Princeton University Press, Princeton (2016)
23. Yang, H.: An NC algorithm for the general planar monotone circuit value problem. In: Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, 1991, pp. 196–203, December 1991