Kaski, Petteri

Engineering a delegatable and error-Tolerant algorithm for counting small subgraphs

Published in:
2018 Proceedings of the 20th Workshop on Algorithm Engineering and Experiments, ALENEX 2018

DOI:
10.1137/1.9781611975055.16

Published: 01/01/2018

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Kaski, P. (2018). Engineering a delegatable and error-Tolerant algorithm for counting small subgraphs. In *2018 Proceedings of the 20th Workshop on Algorithm Engineering and Experiments, ALENEX 2018* (Vol. 2018-January, pp. 184-198). Society for Industrial and Applied Mathematics.
https://doi.org/10.1137/1.9781611975055.16

Engineering a Delegatable and Error-Tolerant Algorithm for Counting Small Subgraphs*

Petteri Kaski†

Abstract

We study the problem of counting the number of occurrences of a given six-vertex pattern graph S in an n-vertex host graph H. We engineer an open-source GPU implementation of a distributed algorithm design of Björklund and Kaski [PODC 2016] where (i) the execution of the algorithm can be *delegated* [Goldwasser, Kalai, and Rothblum, J. ACM 2015] to produce a noninteractive probabilistically checkable proof of correctness, and (ii) the execution of the algorithm when preparing the proof tolerates a controllable number of adversarial errors. Experiments with NVIDIA Tesla K80 and Tesla P100 Accelerators demonstrate that the framework is practical for inputs of up to 512 vertices, with proof checking being several orders of magnitude more efficient than preparing the proof; however, proof preparation still carries at least one order of magnitude overhead compared with just solving the problem.

1 Introduction.

1.1 Delegating Computation. Recent work has shown that *delegating computation* (Goldwasser, Kalai, and Rothblum [57]) with verifiable[1] results is tantalizingly close to practicality (cf. Walfish and Blumberg [117]), with complete tool-chains that enable outsourced execution of a program written in a subset of a general-purpose programming language in a verifiable manner, with only modest cryptographic assumptions underpinning the soundness of the system. The theoretical background for such systems stems from some of the most celebrated results in theoretical computer science, namely the theories of interactive proof systems and probabilistically checkable proofs (cf. §2.1). Very recently, the study of fine-grained complexity has prompted such investigations of verifiability in the context of individual problems ranging from canonical NP-hard problems, such as CNF-satisfiability and graph coloring, to problems believed to be hard in polynomial time, such as k-SUM or k-clique for a constant k (cf. Carmosino, Gao, Impagliazzo, Mihajlin, Paturi, and Schneider [32], Williams [118], Björklund and Kaski [27], Nederlof [85], Ball, Rosen, Sabin, and Vasudevan [13, 14], and Abboud and Rubinstein [2]).

While verifiable general-purpose computation on an extensive scale is perhaps still not practical, earlier engineering effort with interactive proofs (e.g. Cormode, Mitzenmacher, and Thaler [38] and Thaler [105]) and recent theoretical work in the context of fine-grained complexity (see above) suggest that in the context of specific problems and algorithm designs one could make an engineering push towards increasingly practical and non-interactive delegation on extensive infrastructure.

This paper seeks to make such a demonstration together with an open-source release [71] to ease further practical developments. The specific problem we consider is a canonical hard counting problem in polynomial time (cf. §1.4 and §2.3), namely the task of counting the number of isomorphic occurrences of a constant-size pattern graph in a host graph. *What is more, in addition to delegatability, we obtain tolerance against adversarial errors in proof preparation.* To our knowledge this is the first empirical study of delegatability that tolerates errors in proof preparation, and does so with a low requirement for over-provisioning of resources.[2]

1.2 The Case for Tolerance Against Errors. Before proceeding to our specific problem, let us give motivation for tolerance against errors and study of algorithm designs for specific problems, rather than for general-purpose computation, with the objective of practical delegatability.

First, delegation is arguably most useful in situa-

*The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement 338077 "Theory and Practice of Advanced Search and Enumeration". We gratefully acknowledge the use of computational resources provided by the Aalto Science-IT project at Aalto University and by CSC – IT Center for Science, Finland.

†Department of Computer Science, Aalto University, Helsinki

[1] Verifiable with probabilistic soundness, that is, a counterparty who is executing the computation (the *prover* or the *delegate*) who is cheating will be caught by the party who is issuing the computation (the *verifier* or the *delegator*) with high probability.

[2] The low requirement for over-provisioning comes from the fact that proof preparation amounts to producing coordinates of a Reed–Solomon codeword; cf. Footnote 4 for an analysis of the over-provisioning needed to tolerate adversarial errors.

tions where the required computation is extensive, such as in cases when no efficient (low-order polynomial time) algorithms are known, including our present case study. Extensive computations necessitate parallelism and a distributed computing infrastructure.

The more extensive the infrastructure, the more likely it is that the execution of the algorithm experiences a physical error or other such *tail event* (cf. Dean and Baker [42]) that may outright invalidate the computation, or, for example, delay its completion due to need to re-issue parts of the computation. For empirical work and motivation of consideration of errors and other tail events in large-scale infrastructure, cf. Tiwari, Gupta, Gallarno, Rogers, and Maxwell [106], Di Martino, Kalbarczyk, Iyer, Baccanico, Fullop, and Kramer [79], Schroeder, Pinheiro, and Weber [99, 100], Meza, Wu, Kumar, and Mutlu [83], Herault and Robert [62], and Barroso, Clidaras, and Hölzle [17].

Tail events are perhaps realistically expected to be an accelerating concern with increasingly massive computations and infrastructure (cf. Snir *et al.* [103] and Reed and Dongarra [94]). Thus, algorithms that intrinsically both (a) provide a proof of correctness of the result, and (b) tolerate tail events in distributed execution arguably present a sound objective for theory and engineering towards practical delegated computation.

Second, studies of specific problems enable a focused investigation of the design space and thus more efficient problem-tailored designs. Focus on a specific problem enables one to measure the resource overhead for delegatability and tail-tolerance by analytical comparison with the best known designs that merely solve the problem in an assumed tail-event-free environment, including bandwidth-based considerations.

To set the stage for our present contribution, let us next review the mathematical framework we will use for tail-tolerant delegation.

1.3 Polynomials and Tail-Tolerant Delegation.

A univariate polynomial of degree at most d over a field $\mathbb{F}$ can be represented in two alternative ways. The *coefficient representation* gives $d+1$ coefficients $\pi_0, \pi_1, \ldots, \pi_d \in \mathbb{F}$ with

$$P(x) = \pi_0 + \pi_1 x + \pi_2 x^2 + \ldots + \pi_d x^d \,. \tag{1}$$

Dually, an *evaluation representation* gives at least $e \geq d+1$ evaluations

$$(\xi_1, P(\xi_1)) \,,\ (\xi_2, P(\xi_2)) \,,\ \ldots \,,\ (\xi_e, P(\xi_e)) \tag{2}$$

at any distinct points $\xi_1, \xi_2, \ldots, \xi_e \in \mathbb{F}$.

The fact that one can in near-linear[3] time transform between the two dual representations (1) and (2) is perhaps one of the most fundamental algorithmic dualities in the study of computation; we refer to von zur Gathen and Gerhard [114] for an introduction to near-linear time algorithms for computing with univariate polynomials. A further serendipitous fact is that the transformation from (2) to (1) is possible *even when at most* $(e-d-1)/2$ *of the* e *evaluations are in error.* Furthermore, this transformation is computable in near-linear time using, for example, Gao's [54] fast decoding algorithm for Reed–Solomon codes.

Motivation for using low-degree polynomials in designing error-tolerant representations and proof systems enabling delegatibility can be traced back at least to the work of Reed and Solomon [95] on polynomial error-correcting codes, to the work of Freivalds [52] on probabilistically verifying algebraic identities, and to the works of Babai, Fortnow, Levin, and Szegedy [10] and Lund, Fortnow, Karloff, and Nisan [78] on algebraic methods in proof systems.

The particular case for univariate polynomials as a source of efficiency when pushing towards practicality of proof systems has been highlighted in multiple works, including e.g. Ben-Sasson and Sudan [22] and Ben-Sasson, Chiesa, Genkin, and Tromer [19] when working towards practical probabilistically-checkable proofs, and by Williams [118] in the context of fine-grained proof systems for specific problems such as the CNF satisfiability problem. (Cf. also Cormode, Mitzenmacher, and Thaler [38], and Thaler [105] for problem-specific work.) Continuing on the ideas of Williams, the objective of pushing univariate proof systems towards algorithm designs meeting the simultaneous goals of parallelizability, error-tolerance, and matching in total complexity the best known algorithms for specific problems is explored by Björklund and Kaski [27].

In this paper we proceed to implement one of the Björklund–Kaski designs, so let us here give a high-level introduction to the framework they consider without yet entering into detail (cf. §4) how the framework is implemented for our target problem.

The proof. The proof that a computation has been correctly executed is the sequence of coefficients $\pi_0, \pi_1, \ldots, \pi_d \in \mathbb{F}$ of a univariate polynomial of degree at most d. That is, the proof is given in the coefficient representation (1).

Preparing the proof. The proof is prepared using the *evaluation* representation (2). This in particular en-

[3] For the purposes of this introduction, we will use the expression "near-linear time" for an algorithm that on an input of size m runs in time $O(m(\log m)^c)$ for a (small) positive constant c, with the understanding that more detailed time bounds will be presented later, can be found in the cited references, and/or can be obtained by consulting the accompanying source code.

ables immediate parallelization, since proof preparation amounts to evaluating the same polynomial at e distinct points. In essence, each compute node participating in proof preparation gets as input (i) the problem instance, and (ii) one (or more) of the points $\xi_1, \xi_2, \ldots, \xi_e \in \mathbb{F}$. The output of a node consists of the evaluations at its assigned point (or points).

Decoding the proof. As soon as at least $e \geq d+1$ evaluations are available, Gao's near-linear-time decoding algorithm [54] for Reed–Solomon codes enables recovery of the proof even when at most $(e-d-1)/2$ evaluations are in error. This property enables low-overhead tolerance against tail events (such as errors or omissions due to unexpected delays) during proof preparation by over-provisioning of evaluations.[4]

Verifying the proof. Proof verification relies on standard polynomial identity testing. Assuming a verifier has available the problem instance and a putative proof $\tilde{\pi}_0, \tilde{\pi}_1, \ldots, \tilde{\pi}_d \in \mathbb{F}$ with

$$\tilde{P}(x) = \tilde{\pi}_0 + \tilde{\pi}_1 x + \tilde{\pi}_2 x^2 + \ldots + \tilde{\pi}_d x^d \,, \tag{3}$$

the proof can be checked by performing an evaluation at a uniform random point $\xi \in \mathbb{F}$. Indeed, since the problem instance is available, the verifier can compute the correct value $P(\xi)$ using the same algorithm that is used for proof preparation. The verifier can then compute the value $\tilde{P}(\xi)$ using Horner's rule and test that $\tilde{P}(\xi) = P(\xi)$. The verifier's test clearly always succeeds when $\tilde{P}(x) = P(x)$. When $\tilde{P}(x) \neq P(x)$, that is, when the proof supplied to the verifier is incorrect, the verifier's test detects this with probability at least $1 - d/|\mathbb{F}|$.[5]

We observe in particular that proof preparation uses computational resources for e evaluations of the polynomial, and these evaluations can be executed in parallel, independently of each other, without the need for communication. Verification requires only one evaluation.[6]

[4]For example, assuming that at most an ϵ-fraction of the $e = (1+\delta)(d+1)$ evaluations is in error for some $0 \leq \epsilon < 1/2$, it suffices to over-provision so that $\epsilon e \leq (e-d-1)/2$ holds, that is, so that $\delta \geq 2\epsilon/(1-2\epsilon)$ holds. For example, if the error rate is $\epsilon = 0.01$, it suffices to over-provision by $\delta \geq 0.02041$. Furthermore, assuming there are at most $(e-d-1)/2$ errors, the erroneous evaluations can be identified in near-linear time, enabling further investigation into the source of the errors.

[5]Indeed, the difference $\tilde{P}(x) - P(x)$ is then a nonzero polynomial of degree at most d, so the verifier's random choice $\xi \in \mathbb{F}$ lands at a root of the difference with probability at most $d/|\mathbb{F}|$.

[6]Or more evaluations if the verifier wants further confidence that the proof is correct. With r independent repetitions, the probability to detect a bad proof is at least $1 - (d/|\mathbb{F}|)^r$.

1.4 Counting Small Subgraphs. We are now ready to proceed to our problem of interest. Suppose we are given as input two undirected loopless graphs, S and H, together with a designation for each unordered pair of vertices of S as either *important* or *non-important*. Our task is to count the number of injective mappings $\varphi : V(S) \to V(H)$ from the k-vertex *small* graph S to the n-vertex *host* graph H such that for every important pair $\{u, v\} \subseteq V(S)$ it holds that $\{u, v\} \in E(S)$ if and only if $\{\varphi(u), \varphi(v)\} \in E(H)$. By varying the designations of importance, it is immediate that this problem subsumes both counting the occurrences of S as a subgraph of H and counting the occurrences of S as an induced subgraph of H.

The task of counting small subgraphs in a graph has received substantial attention in the algorithms community, both as a generic problem when S is part of the input (as formulated above), or in special cases when S is fixed to be a specific graph with specific designations of importance, such as when S is the complete graph on k vertices and all pairs of vertices are important, in which case one obtains the k-clique counting problem. We refer to e.g. Itai and Rodeh [67], Nešetřil and Poljak [86], Alon, Yuster, Zwick [6], Alon and Gutner [5], Eisenbrand and Grandoni [44], Björklund, Husfeldt, Kaski, and Koivisto [26], Björklund, Kaski, and Kowalik [28], Vassilevska Williams, Wang, Williams, and Yu [110], Vassilevska Williams and Williams [112], Fomin, Lokshtanov, Raman, Saurabh, and Raghavenda Rao [51], Floderus, Kowaluk, Lingas, and Lundell [49], Olariu [87], Kloks, Kratsch, Müller [72], and Curticapean, Dell, and Marx [40] for a non-exhaustive sample of earlier work on algorithm designs for subgraph counting. We postpone a discussion of hardness of the problem, implementations, and applications to §2.3.

Despite of extensive work, from a worst-case perspective asymptotically the fastest known algorithm for solving the generic problem (when S is part of the input) remains the 1985 design of Nešetřil and Poljak [86] (see also Eisenbrand and Grandoni [44]) and runs in time $O(n^{(\omega+\epsilon)\lfloor k/3 \rfloor + (k \bmod 3)})$, where $\epsilon > 0$ is any positive constant and $2 \leq \omega < 2.3728639$ is the exponent[7] of square matrix multiplication (cf. Le Gall [75] and Vassilevska Williams [109]).

In the particular case when S has six vertices, the running time of the Nešetřil–Poljak algorithm is $O(n^{2\omega+\epsilon})$.[8] The algorithm is based on an algebraization

[7]More precisely, we let ω be the limit $\lim_{n\to\infty} \frac{\log \mathrm{rk}\langle n,n,n\rangle}{\log n}$, where $\langle n, n, n\rangle$ is the $n \times n$ matrix-multiplication tensor and rk means the tensor rank in characteristic 0.

[8]We restrict this conference abstract to consider the case of six vertices since this results in a balanced design and a problem that is hard to solve already for small n. Indeed, the naïve running

of the counting task. More precisely, let us assume that the vertex set of S is $V(S) = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}\}$. Let $\mathbb{F}$ be a field and associate an $n \times n$ matrix with each unordered pair of vertices; namely, let χ^{ab}, χ^{ac}, χ^{ad}, χ^{ae}, χ^{af}, χ^{bc}, χ^{bd}, χ^{be}, χ^{bf}, χ^{cd}, χ^{ce}, χ^{cf}, χ^{de}, χ^{df}, $\chi^{\mathrm{ef}} \in \mathbb{F}^{n \times n}$ be these matrices.[9] The subgraph counting task now reduces to evaluating the $\binom{6}{2}$-linear form

$$X_{\binom{6}{2}} = \sum_{a,b,c,d,e,f=1}^{n} \chi^{\mathrm{ab}}_{ab} \chi^{\mathrm{ac}}_{ac} \chi^{\mathrm{ad}}_{ad} \chi^{\mathrm{ae}}_{ae} \chi^{\mathrm{af}}_{af} \cdot \cdot \chi^{\mathrm{bc}}_{bc} \chi^{\mathrm{bd}}_{bd} \chi^{\mathrm{be}}_{be} \chi^{\mathrm{bf}}_{bf} \chi^{\mathrm{cd}}_{cd} \cdot \chi^{\mathrm{ce}}_{ce} \chi^{\mathrm{cf}}_{cf} \chi^{\mathrm{de}}_{de} \chi^{\mathrm{df}}_{df} \chi^{\mathrm{ef}}_{ef}, \tag{4}$$

which can be evaluated naïvely in $O(n^6)$ operations and, using fast matrix multiplication on $n^2 \times n^2$ inputs [86], in $O(n^{2\omega+\epsilon})$ operations for any constant $\epsilon > 0$.

Björklund and Kaski [27] present a new design (cf. §4) that improves the Nešetřil–Poljak [86] design with better parallelisability and less memory usage. This design also extends to the univariate proof framework (cf. §1.3), with a proof polynomial $P(x)$ of degree $d = O(n^{\omega+\epsilon})$; the proof polynomial can be evaluated at any given point $\xi \in \mathbb{F}$ using $O(n^{\omega+\epsilon})$ operations in $\mathbb{F}$ for any constant $\epsilon > 0$. Thus, the total effort to prepare the proof essentially matches[10] the Nešetřil–Poljak effort. Furthermore, the proof preparation can be parallelized to $d + 1$ compute nodes.

1.5 Our Contribution. In this paper we seek to engineer a practical implementation of the Björklund–Kaski design for delegatable and error-tolerant counting of six-vertex subgraphs. Our objective is to understand the practical feasibility of the polynomial framework and the engineering considerations in relation to the available arithmetic and memory bandwidth on modern massively parallel microarchitetures, in particular on graphics processing units (GPUs) and systems built from a large number of GPUs.[11] This objective presents us with a number of challenges in algorithm engineering compared with the theoretical framework.

The foremost challenge is to expose sufficient parallelism in the framework to cope with the extensive latencies caused by long pipelines for both arithmetic operations and access to all levels of memory other than registers in a GPU (cf. Volkov [113] and Mei and Chu [82] for a discussion of the role of latency and GPU microarchitectures). Indeed, although the Björklund–Kaski algorithm is by design massively vector-parallel—it evaluates the same univariate polynomial modulo a prime at a large number of distinct points; cf. (2)—this parallelism is, *as is*, rather unsuitable for a latency-tolerant implementation. Here the main challenge are the extensive memory accesses needed by each point evaluation (e.g. matrix multiplications on large independent matrices), which necessitate storage capacity that cannot be satisfied by low-latency memory; indeed, while a GPU can easily maintain the state of thousands of individual threads in execution, even this capacity is in practice insufficient to hide memory latency unless

(i) each thread has as much as possible data local in its dedicated registers to enable low-latency operations on data,

(ii) when a thread accesses high-latency memory, the size of the access is as large as possible to saturate the memory bandwidth, and

(iii) the memory layout of the data used by the algorithms has been designed for coalesced (vectorized) memory accesses as supported by the GPU hardware.

A secondary challenge is that modern GPU hardware has been optimized towards floating-point arithmetic on short word lengths rather than integer arithmetic on long words.

Let us now discuss our engineering contributions towards making the Björklund–Kaski framework practical on GPUs, proceeding from low-level details to higher-level considerations.

Scalar arithmetic. Subgraph counts already on modest-size inputs can exceed the word length to which the physical instruction set has been optimized. This and the proof framework force us to work with the Chinese Remainder Theorem and modular arithmetic. Our solution relies on standard Montgomery multiplication [84] and a low-level PTX assembly-language implementation for Montgomery multiplication modulo word-length primes on the GPU.

Polynomial arithmetic. The proof framework requires practical parallel algorithms for working with polynomials of degree in the hundreds of millions and beyond. This requires a careful implementation of the near-linear

time for a dense input is $O(n^6)$.

[9] To obtain the generic subgraph counting problem for given S and H, choose the adjancency matrix of H (for important pairs that are edges of S), the zero-diagonal non-adjacency matrix of H (for important pairs that are non-edges of S), and the all-ones but zero-diagonal matrix (for non-important pairs) for each of the 15 matrices χ.

[10] Up to low-order terms masked by ϵ; the overhead is polylogarithmic in n.

[11] For example, each of the 18,688 nodes of the Titan supercomputer is equipped with 2688-core NVIDIA Tesla K20X Accelerators (cf. Tiwari, Gupta, Gallarno, Rogers, and Maxwell [106]).

time algorithms for polynomial arithmetic (cf. [114]), starting from fast polynomial multiplication and culminating with Gao's [54] fast decoding algorithm for Reed–Solomon codes for decoding the proof polynomial. For polynomial multiplication, we implement a parallel version of the Schönhage–Strassen [98] algorithm both on the GPU and on the host CPU, with the possibility to execute the outer butterfly-layers of the algorithm on the host when the GPU memory suffices only for the recursive inner layers.

The evaluation algorithm for the proof polynomial. The Björklund–Kaski evaluation algorithm for the proof polynomial for counting six-vertex subgraphs consists of a preprocessing phase for coefficients of Lagrange polynomials and seven fast matrix multiplications on matrices of size $n \times n$ modulo a prime (cf. §4). To expose sufficient parallelism in the design, we look inside the evaluation algorithm for parallelism and implement both the preprocessing and the fast matrix multiplication steps using Yates's algorithm and Strassen's decomposition of the 2×2 matrix-multiplication tensor (cf. (6)), which enables us to obtain vector-parallelization for each component-transform of Yates's algorithm. In essence, each component of Yates's algorithm is a (data-expanding or data-contracting) butterfly circuit which can be easily optimized for vector-parallel execution. Here our design was encouraged by recent successes in deploying Strassen's algorithm [104] and its generalizations in practice (cf. §2.4). At low level, to enable low-latency execution and a hardware-aware memory layout, we execute the innermost 4×4 matrix multiplication on local registers available to each thread, utilizing the per-thread 4-scalar-wide load and store instructions available in the microarchitecture. Exposing this parallelism inside the evaluation algorithm enables us to obtain empirical throughput that is less than two orders of magnitude from a lower bound using the peak modular multiplication rate of the hardware (cf. §6).

Open-source implementation. We make the implementation available as C++/CUDA open-source [71] under the MIT License to encourage and ease further developments in the area.

1.6 Experiments. We present experiments that investigate the practical feasibility of the framework for extensive computations on current NVIDIA Kepler and Pascal GPU microarchitectures. We find that the framework is practical for inputs of up to 512 vertices, with proof checking being several orders of magnitude more efficient than preparing the proof; however, proof preparation still carries at least one order of magnitude overhead compared with just solving the problem. Yet we find it promising that, as a proof of concept, delegatability and error-tolerance can be simultaneously realized for nontrivial input sizes.

1.7 Organization. We start with a further discussion of earlier work in §2. In §3 we review the mathematical and algorithmic preliminaries for our implementation, including a review of the Björklund–Kaski algorithm design in §4. Our engineering considerations for implementation are presented in §5. We report on our experiments in §6.

2 Earlier and Related Work.

2.1 Proof Systems and Delegation. *Can one prove to a resource-limited counterparty that an extensive computation has been correctly executed?* The study of this question is fundamental to the current understanding of computation, such as in the classical characterization of NP as the class of decision problems whose positive instances have polynomial-size proofs that are verifiable in deterministic polynomial time.

Allowing for two-way interaction between a computationally unbounded *prover* and a randomized-polynomial-time *verifier* leads to the study of *interactive proof systems* introduced by Babai [9, 12] and Goldwasser, Micali, and Rackoff [58, 59], with classical highlights including the algebraic proof framework of Lund, Fortnow, Karloff, and Nisan [78], Shamir's result [102] that unbounded interaction captures exactly the class of polynomial-space-solvable decision problems, and Babai, Fortnow, and Lund [11] establishing that interaction with two independent provers captures nondeterministic exponential time.

Limiting the verifier to use only logarithmic randomness and a constant number of bit-queries to a proof string led Arora, Feige, Goldwasser, Lovász, Lund, Motwani, Safra, Sudan, and Szegedy [7, 8, 46] to a breakthrough characterization of NP via *probabilistically checkable proofs* (PCPs) and to a theory of hardness of polynomial-time approximation in combinatorial optimization.

Subsequent theoretical work has evolved along a number of lines, including the objectives to (i) *delegate computation* by restricting the interactive prover to polynomial time (Goldwasser, Kalai, and Rothblum [57]; Goldreich and Rothblum [56]), (ii) optimize and simplify PCPs for use in verifiable computation and inapproximability (e.g. Babai, Fortnow, Levin, and Szegedy [10], Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [21], Ben-Sasson and Sudan [22], Dinur [43], and Håstad [60]), and most recently, (iii) gain insight into fine-grained complexity (cf. §1.1).

Beyond theoretical foundations, recently substan-

tial progress has been made in engineering verifiable computation to near-practicality, such as in the Pinocchio system of Parno, Howell, Gentry, and Raykova [89, 55] relying only on modest cryptographic assumptions; cf. Walfish and Blumberg [117] for a review of efforts to make verifiable universal computation practical (including Cormode, Mitzenmacher, and Thaler [38], Thaler [105], Vu, Setty, Blumberg, and Walfish [115], Wahby, Setty, Ren, Blumberg, and Walfish [116], and Ben-Sasson, Chiesa, Genkin, Tromer, and Virza [20]). Yet the objective of realizing general-purpose verifiable computation apparently remains some distance away from practical deployment, e.g. as regards tolerance for errors (cf. §1.2).

2.2 Verifiable or Error-Tolerant Algorithms. McConnell, Mehlhorn, Näher, and Schweitzer [80] review *certifiable algorithms* that produce an easy-to-verify certificate that the output of the algorithm is correct. Yet such designs in general are not tolerant against tail events. Conversely, a number of works study algorithms that are resilient to errors but do not in general produce a correctness proof. For example, Caminiti, Finocchi, Fusco, and Silvetri [30] study resilient dynamic programming, Chen, Grigorescy, and de Wolf [33] study error-correcting data structures for membership queries and polynomial evaluation, Cicalese [37] studies fault-tolerant search algorithms, and Finocchi, Grandoni, and Italiano [48] present sorting and searching algorithms robust against memory errors. Herault and Robert [62] review fault-tolerance techniques in high-performance computing, including work on *algorithm-based fault tolerance* introduced by Huang and Abraham [66]. Our work differs from these works in that we obtain simultaneously (i) low-overhead fault-tolerance against adversarial errors, and (ii) a noninteractive, probabilistically checkable proof of correctness.

2.3 Counting and Enumerating Subgraphs. From a parameterized complexity standpoint it is known that subgraph-counting parameterized by the number of vertices k in the pattern graph S is a hard problem in the class #W[1]. Cf. Flum and Grohe [50], Chen and Flum [34], Chen, Thurley, Weyer [35], Curticapean [39], Curticapean and Marx [41], Jerrum and Meeks [69, 70], and Meeks [81]. The specific problem of finding and counting cliques is used as a source of fine-grained hardness reductions by Abboud, Backurs, and Vassilevska Williams [1].

Small-subgraph counting and enumeration on large host graphs, including homomorphism-counting, is encountered in a large number of applications. For a nonexhaustive sample of the extensive body of work in this direction, including theoretical and engineering work, cf. [108, 92, 111, 68, 29, 64, 107, 36, 3, 47, 73, 97, 74, 88, 18, 45, 24, 91, 96, 63, 93, 4, 25, 90]. Our work differs from these works in that we seek a proof-of-concept implementation for simultaneous delegatability and error-tolerance.

2.4 Fast Matrix Multiplication. A number of works in the last few years study the communication-efficiency and practice of matrix multiplication on distributed architectures using decompositions of small matrix-multiplication tensors, such as Strassen's decomposition. Cf. Ballard, Demmel, Holtz, Lipshitz, and Schwartz [15], Ballard, Demmel, Holtz, and Schwartz [16], and Lipshitz, Ballard, Demmel, and Schwartz [77]. Recent engineering work includes Benson and Ballard [23] and Huang, Rice, Matthews, and van de Geijn [65]. Our work differs from these works in that we seek a self-contained proof-of-concept demonstration requiring good-performance finite-field matrix multiplication on GPUs, but we do not necessarily seek the most optimized possible implementation; such optimizations are left for future work.

3 Preliminaries.

For algorithms working with elements of a field $\mathbb{F}$, we measure the running time using operations in $\mathbb{F}$ (addition, negation, multiplication, inverse).

Let S and T be two finite sets. Let us write $\mathbb{F}^{S\times T}$ for the set of $|S| \times |T|$ matrices with entries in $\mathbb{F}$ indexed by $S \times T$. For a matrix A, let us write $A^\top$ for the transpose of A. For two matrices A and B, let us write $A \otimes B$ for the Kronecker product of A and B. For a nonnegative integer r, let us write $A^{\otimes r}$ for the r-fold Kronecker power of A.

3.1 Yates's Algorithm. Yates's algorithm [119] enables fast multiplication of a vector with a large Kronecker power of a matrix. Let S and T be index sets of sizes s and t, respectively. Let $A \in \mathbb{F}^{T\times S}$ be matrix of size $t \times s$ and let r be a nonnegative integer. Let us write I_m for the $m \times m$ identity matrix. Given a vector $x \in \mathbb{F}^{S^r}$ as input, we seek to compute $A^{\otimes r}x \in \mathbb{F}^{T^r}$. Yates's algorithm executes this computation via the following decomposition of $A^{\otimes r}$. For $\ell = 0, 1, \ldots, r-1$ define the linear map $A^{[\ell]} : \mathbb{F}^{S^{r-\ell}\times T^\ell} \to \mathbb{F}^{S^{r-\ell-1}\times T^{\ell+1}}$ with

$$A^{[\ell]} = \underbrace{I_s \otimes I_s \otimes \cdots \otimes I_s}_{r-\ell-1} \otimes A \otimes \underbrace{I_t \otimes I_t \otimes \cdots \otimes I_t}_{\ell} .$$

Now observe that $A^{\otimes r} = A^{[r-1]}A^{[r-2]} \cdots A^{[1]}A^{[0]}$. In particular, the number of arithmetic operations to multiply the matrix $A^{[\ell]}$ with a vector is $O(s^{r-\ell}t^{\ell+1})$. Thus,

we can compute $A^{\otimes r}x$ given x in total $O(s^{r+1}r)$ operations if $t = s$ and $O(\max(s^{r+2}, t^{r+2}))$ operations if $t \neq s$. Using an $O(\max(s^r, t^r))$-processor shared-memory multiprocessor, Yates's algorithm can be executed in parallel in r layers where each processor executes $O(s)$ operations in a layer.

3.2 Fast Matrix Multiplication. This section develops fast matrix multiplication via Yates's algorithm. Let U be an index set of size u and let V be an index set of size v for positive integer constants u and v. Let $\alpha, \beta, \gamma \in \mathbb{F}^{(U\times U)\times V}$ be three $u^2 \times v$ matrices with entries $\alpha_{ij'\ell}$, $\beta_{jk'\ell}$, $\gamma_{i'k\ell}$ for all $i, i', j, j', k, k' \in U$ and $\ell \in V$ such that

$$\sum_{\ell\in V} \alpha_{ij'\ell}\beta_{jk'\ell}\gamma_{i'k\ell} == \begin{cases} 1 & \text{if } i = i',\ j = j',\ k = k'; \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

For example, to represent Strassen's matrix multiplication algorithm [104], let us write "$\bar{1}$" to indicate a "-1" and take $u = 2$ and $v = 7$ with

$$\alpha = \begin{bmatrix} 1&0&1&0&1&\bar{1}&0 \\ 0&0&0&0&1&0&1 \\ 0&1&0&0&0&1&0 \\ 1&1&0&1&0&0&\bar{1} \end{bmatrix},\ \beta = \begin{bmatrix} 1&1&0&\bar{1}&0&1&0 \\ 0&0&1&0&0&1&0 \\ 0&0&0&1&0&0&1 \\ 1&0&\bar{1}&0&1&0&1 \end{bmatrix},\ \gamma = \begin{bmatrix} 1&0&0&1&\bar{1}&0&1 \\ 0&0&1&0&1&0&0 \\ 0&1&0&1&0&0&0 \\ 1&\bar{1}&1&0&0&1&0 \end{bmatrix}. \tag{6}$$

Let us now develop a fast matrix multiplication algorithm based on Yates's algorithm. Fix a decomposition α, β, γ such as (6) with index sets U and V. Let $X = (X_{ij} : i, j \in U^r)$ and $Y = (Y_{ij} : i, j \in U^r)$ be two matrices of size $u^r \times u^r$. We seek to compute the product matrix $Z = XY$ with entries given by $Z_{ik} = \sum_{j\in U^r} X_{ij}Y_{jk}$ for $i, k \in U^r$.

Rearrange and flatten the entries of $X \in \mathbb{F}^{U^r\times U^r}$ to a vector $\tilde{x} \in \mathbb{F}^{(U\times U)\times(U\times U)\times\cdots\times(U\times U)}$ so that

$$\tilde{x}_{(i_1,j_1),(i_2,j_2),\ldots,(i_r,j_r)} = X_{(i_1,i_2,\ldots,i_r),(j_1,j_2,\ldots,j_r)}$$

holds for all $i_1, i_2, \ldots, i_r \in U$ and $j_1, j_2, \ldots, j_r \in U$. Do a similar rearranging and flattening to the matrix Y to obtain the vector $\tilde{y}$. Next, use Yates's algorithm to compute the vectors $\tilde{\tilde{x}} \leftarrow \left(\alpha^\top\right)^{\otimes r}\tilde{x}$ and $\tilde{\tilde{y}} \leftarrow \left(\beta^\top\right)^{\otimes r}\tilde{y}$. Take the elementwise (Hadamard) product to obtain $\tilde{\tilde{z}} \leftarrow \tilde{\tilde{x}} \odot \tilde{\tilde{y}}$. Then use Yates's algorithm to compute $\tilde{z} \leftarrow \gamma^{\otimes r}\tilde{\tilde{z}}$. Finally, recover the product matrix Z by un-flattening and rearranging $\tilde{z}$. When $v \geq u^2 + 1$, this enables us to multiply two $u^r \times u^r$ matrices in $O(v^{r+2})$ arithmetic operations in $\mathbb{F}$.

4 Counting Six-Vertex Subgraphs.

This section reviews the Björklund–Kaski [27] algorithm for counting six-vertex subgraphs. The univariate polynomial framework which the algorithm instantiates is reviewed in §1.3.

Throughout this section we will work with a fixed decomposition of a matrix-multiplication tensor. That is, we fix three constant-size $u^2 \times v$ matrices $\alpha, \beta, \gamma \in \mathbb{F}^{(U\times U)\times V}$ from §3.2. For concreteness, consider the 4×7 Strassen matrices (6). Accordingly, U and V are constant-size index sets of sizes u and v, respectively.

4.1 The 6-Choose-2-Linear Form. Recalling the binomial linear form (4) from §1.4, it will be convenient to assume the input size $n = u^r$ is a nonnegative integer power of u, and the indexing of the rows and columns of the fifteen input matrices is via a Cartesian power of U. More precisely, let χ^{ab}, χ^{ac}, χ^{ad}, χ^{ae}, χ^{af}, χ^{bc}, χ^{bd}, χ^{be}, χ^{bf}, χ^{cd}, χ^{ce}, χ^{cf}, χ^{de}, χ^{df}, $\chi^{\mathrm{ef}} \in \mathbb{F}^{U^r\times U^r}$. We seek to compute the $\binom{6}{2}$-linear form

$$\begin{aligned} X_{\binom{6}{2}} = \sum_{a,b,c,d,e,f\in U^r} & \chi^{\mathrm{ab}}_{ab}\chi^{\mathrm{ac}}_{ac}\chi^{\mathrm{ad}}_{ad}\chi^{\mathrm{ae}}_{ae}\chi^{\mathrm{af}}_{af} \cdot \\ & \cdot \chi^{\mathrm{bc}}_{bc}\chi^{\mathrm{bd}}_{bd}\chi^{\mathrm{be}}_{be}\chi^{\mathrm{bf}}_{bf}\chi^{\mathrm{cd}}_{cd} \\ & \cdot \chi^{\mathrm{ce}}_{ce}\chi^{\mathrm{cf}}_{cf}\chi^{\mathrm{de}}_{de}\chi^{\mathrm{df}}_{df}\chi^{\mathrm{ef}}_{ef}\,. \end{aligned} \tag{7}$$

4.2 The Proof Polynomial. The proof polynomial will utilize polynomial extensions of the p-fold Kronecker powers of the matrices α, β, γ. Toward this end, for $d, e, f, d', e', f' \in U^r$ and $\ell \in V^r$, let us write

$$\begin{aligned} \alpha^{\otimes r}_{de'\ell} &= \alpha_{d_1e'_1\ell_1}\alpha_{d_2e'_2\ell_2}\cdots\alpha_{d_re'_r\ell_r}\,, \\ \beta^{\otimes r}_{ef'\ell} &= \beta_{e_1f'_1\ell_1}\beta_{e_2f'_2\ell_2}\cdots\beta_{e_rf'_r\ell_r}\,, \\ \gamma^{\otimes r}_{d'f\ell} &= \gamma_{d'_1f_1\ell_1}\gamma_{d'_2f_2\ell_2}\cdots\gamma_{d'_rf_r\ell_r} \end{aligned} \tag{8}$$

for the entries of the p-fold Kronecker powers. Let us now extend to low-degree polynomials. Associate with each $\ell \in V^r$ a unique element $\xi_\ell \in \mathbb{F}$. For all $d, e, f, d', e', f' \in U^r$ introduce the univariate polynomials

$$\begin{aligned} \alpha^{\otimes r}_{de'}(x) &= \sum_{\ell\in V^r} \alpha^{\otimes r}_{de'\ell} \prod_{\ell'\in V^r\setminus\{\ell\}} \frac{x - \xi_{\ell'}}{\xi_\ell - \xi_{\ell'}}\,, \\ \beta^{\otimes r}_{ef'}(x) &= \sum_{\ell\in V^r} \beta^{\otimes r}_{ef'\ell} \prod_{\ell'\in V^r\setminus\{\ell\}} \frac{x - \xi_{\ell'}}{\xi_\ell - \xi_{\ell'}}\,, \\ \gamma^{\otimes r}_{d'f}(x) &= \sum_{\ell\in V^r} \gamma^{\otimes r}_{d'f\ell} \prod_{\ell'\in V^r\setminus\{\ell\}} \frac{x - \xi_{\ell'}}{\xi_\ell - \xi_{\ell'}}\,. \end{aligned} \tag{9}$$

In particular, the polynomials in (9) are Lagrange interpolation polynomials of degree at most $v^r - 1$ that for each $\ell \in V^r$ are easily seen to satisfy

$$\begin{aligned} \alpha^{\otimes r}_{de'}(\xi_\ell) &= \alpha^{\otimes r}_{de'\ell}\,, \\ \beta^{\otimes r}_{ef'}(\xi_\ell) &= \beta^{\otimes r}_{ef'\ell}\,, \\ \gamma^{\otimes r}_{d'f}(\xi_\ell) &= \gamma^{\otimes r}_{d'f\ell}\,. \end{aligned} \tag{10}$$

It will be convenient to view (9) as defining three $u^r \times u^r$ matrices $\alpha^{\otimes r}(x), \beta^{\otimes r}(x), \gamma^{\otimes r}(x) \in \mathbb{F}[x]^{U^r\times U^r}$ with entries that are polynomials in x.

Using elementwise (Hadamard) multiplication and matrix multiplication, introduce the matrices $H(x), K(x), L(x) \in \mathbb{F}[x]^{U^r \times U^r}$ defined for all $a, b, c, d, e, f \in U^r$ by

$$\begin{aligned} H_{ad}(x) &= \sum_{e' \in U^r} \alpha^{\otimes r}_{de'}(x) \chi^{\mathrm{ae}}_{ae'} \chi^{\mathrm{de}}_{de'}\,, \\ K_{be}(x) &= \sum_{f' \in U^r} \beta^{\otimes r}_{ef'}(x) \chi^{\mathrm{bf}}_{bf'} \chi^{\mathrm{ef}}_{ef'}\,, \\ L_{cf}(x) &= \sum_{d' \in U^r} \gamma^{\otimes r}_{d'f}(x) \chi^{\mathrm{cd}}_{cd'} \chi^{\mathrm{df}}_{d'f}\,. \end{aligned} \tag{11}$$

Next, introduce the matrices $A(x), B(x), C(x) \in \mathbb{F}[x]^{U^r \times U^r}$ with entries defined for all $a, b, c, d, e, f \in U^r$ by

$$\begin{aligned} A_{ab}(x) &= \sum_{d \in U^r} \chi^{\mathrm{ad}}_{ad} \chi^{\mathrm{bd}}_{bd} H_{ad}(x)\,, \\ B_{bc}(x) &= \sum_{e \in U^r} \chi^{\mathrm{be}}_{be} \chi^{\mathrm{ce}}_{ce} K_{be}(x)\,, \\ C_{ac}(x) &= \sum_{f \in U^r} \chi^{\mathrm{af}}_{af} \chi^{\mathrm{cf}}_{cf} L_{cf}(x)\,. \end{aligned} \tag{12}$$

Then, introduce the matrix $Q(x) \in \mathbb{F}[x]^{U^r \times U^r}$ with entries defined for all $a, b \in U^r$ by

$$Q_{ab}(x) = \sum_{c \in U^r} \chi^{\mathrm{ac}}_{ac} \chi^{\mathrm{bc}}_{bc} B_{bc}(x) C_{ac}(x)\,. \tag{13}$$

Finally, introduce the polynomial

$$P(x) = \sum_{a,b \in U^r} \chi^{\mathrm{ab}}_{ab} A_{ab}(x) Q_{ab}(x)\,. \tag{14}$$

From (14), (13), (12), (11), and (9) it is immediate that $P(x)$ has degree at most $3v^r - 3$.

The next theorem shows that we can recover the $\binom{6}{2}$-linear form (7) as a simple sum of evaluations of the polynomial (14). That is, to count subgraphs it suffices to evaluate the polynomial at sufficiently many distinct points.

THEOREM 4.1. $X_{\binom{6}{2}} = \sum_{\ell \in V^r} P(\xi_\ell)$.

Proof. Expand $\sum_{\ell \in V^r} P(\xi_\ell)$ via (14), (13), (12), (11) and apply (10), (8), and (5) for each $a, b, c \in U^r$ in turn to conclude that the claim holds.

4.3 Evaluating the Proof Polynomial. We now describe how to evaluate the polynomial (14) fast. Let us write $\mathbb{F}_q$ for a finite field with q elements.

THEOREM 4.2. *Let $v \geq u^2 + 1$ and let p be a prime with $p \geq v^r$. Then, there is an algorithm that for a given $\xi \in \mathbb{F}_p$ computes $P(\xi) \in \mathbb{F}_p$ in $O(v^{r+2})$ operations in $\mathbb{F}_p$ and using space for $O(v^r)$ elements of $\mathbb{F}_p$.*

Proof. Let $\xi \in \mathbb{F}_p$ be given. Arbitrarily identify elements of V^r with unique elements of $\{0, 1, \ldots, v^r - 1\} \subseteq \mathbb{F}_p$. Subject to this identification, take $\xi_\ell = \ell \in \{0, 1, \ldots, v^r - 1\}$ for all $\ell \in V^r$. Let us first compute a vector $\eta \in \mathbb{F}^{V^r}$ such that for all $\ell \in V^r$ we have

$$\eta_\ell = \prod_{\ell' \in V^r \setminus \{\ell\}} \frac{\xi - \xi_{\ell'}}{\xi_\ell - \xi_{\ell'}}\,. \tag{15}$$

Note in particular that this is trivial when $\xi = \xi_\ell$ for some ℓ because then η is $\{0,1\}$-valued with a single 1 at position ℓ, so let us assume otherwise. Because of our identification, for all $\ell \in V^r$ it then holds that

$$\eta_\ell = \frac{\prod_{\ell' \in V^r} (\xi - \xi_{\ell'})}{\ell!\,(-1)^{v^r - 1 - \ell}\,(v^r - 1 - \ell)!\,(\xi - \xi_\ell)}\,. \tag{16}$$

Now observe that the numerator in (16) is independent of ℓ and hence can be precomputed in $O(v^r)$ operations. Similarly, we can precompute a table of (inverse) factorials to access the factorials in the denominator of (16) in $O(1)$ operations each. Finally, for each $\ell \in V_r$ the inverse $(\xi - \xi_\ell)^{-1}$ can be computed in $O(1)$ operations. Thus, we can compute the vector η in total $O(v^r)$ operations.

Let us now observe that (8), (9), and (15) imply that we can use Yates's algorithm to compute the matrices $\alpha^{\otimes r}(\xi), \beta^{\otimes r}(\xi), \gamma^{\otimes r}(\xi) \in \mathbb{F}_p^{U^r \times U^r}$ by

$$\alpha^{\otimes r}(\xi) \leftarrow \alpha^{\otimes r}\eta\,, \quad \beta^{\otimes r}(\xi) \leftarrow \beta^{\otimes r}\eta\,, \quad \gamma^{\otimes r}(\xi) \leftarrow \gamma^{\otimes r}\eta\,.$$

By our assumption $v \geq u^2 + 1$, this takes $O(v^{r+2})$ operations. Using Yates's algorithm to perform fast matrix multiplication, we can now use (11), (12), (13), and (14) to obtain $P(\xi) \in \mathbb{F}_p$ in $O(v^{r+2})$ operations.

5 Engineering an Implementation.

This section documents the key engineering considerations in our implementation of the Björklund–Kaski framework, with more detailed considerations presented in the accompanying source code [71].

5.1 Arithmetic in Prime-Order Fields. To obtain good arithmetic performance over prime-order fields on the GPU, one must pay attention to low-level implementation of such arithmetic. We rely on Montgomery multiplication [84] implemented at low level using PTX inline assembly in CUDA C to obtain fast arithmetic modulo an at-most-32-bit prime.

5.2 Near-Linear-Time Polynomial Arithmetic. Our implementation relies on standard algorithms for fast polynomial arithmetic on univariate polynomials [114]. The foundation for such algorithms is a fast

algorithm for polynomial multiplication that runs in $O(\mathrm{M}(d))$ operations for inputs of degree at most d. We rely on the Schönhage–Strassen algorithm [98] for this foundation, with $\mathrm{M}(d) = d \log d \log\log d$. (Asymptotically faster algorithms are known, cf. Fürer [53] and Lin, Al-Naffouri, Han, and Chung [76] for the state of the art.) Once a fast multiplication algorithm is available, standard reductions to multiplication yield the near-linear-time toolkit for polynomials (quotient, remainder, batch evaluation, interpolation, and extended Euclidean algorithm), including an implementation of Gao's decoding algorithm [54] that runs in $O(\mathrm{M}(e) \log e)$ operations for e given points. We release our GPU implementation of this toolkit that has sufficient performance for our purposes as regards the present proof-of-concept demonstration, with the understanding that the implementation can be further optimized.

5.3 Fast Matrix Multiplication. The most performance-critical aspect in implementing the Björklund–Kaski framework for subgraph counting consists of implementing the seven fast matrix multiplications in the algorithm that produces evaluations of the proof polynomial (cf. §4.3). We develop a batch-parallel implementation of Strassen's algorithm using Yates's algorithm (cf. §3.2) that multiplies in parallel P independent pairs of $2^k \times 2^k$ operands.

To guarantee coalesced workloads on the GPU, we employ a memory layout of shape

$$4^{k-1} \times P \times 4$$

for the operands, which we then expand using Yates's algorithm with (6) to shape

$$4 \times 7^{k-2} \times P \times 4\,.$$

We then use a kernel that executes in parallel independent 4×4 matrix multiplications along the most and least significant modes, using 4-scalar load and store instructions for each thread of the kernel. Finally, we compress back to shape

$$4^{k-1} \times P \times 4\,.$$

The expanding transformations start from the least significant length-4 modes to produce length-7 modes, and proceed towards the more significant modes to guarantee that the memory accessed are coalesced when the intermediate results occupy the most memory. The compressing transformation proceeds in reverse order.

6 Experiments.

6.1 Hardware and Software Configuration. Our implementation is written in C++ and CUDA C that is compiled to the following platforms.

K80 GPU compute node. An NVIDIA Tesla K80 Accelerator with two 875-MHz NVIDIA GK210 GPUs (Kepler microarchitecture, 2496 cores, 13 SMX, 192 cores/SMX), 12288 MiB of on-device GDDR5-3004 memory with ECC enabled. The host is a Dell PowerEdge C4130 with two 2.40-GHz Intel Xeon E5-2620v3 CPU (Haswell microarchitecture, 12 cores, 6 cores/CPU, no hyper-threading, 15 MiB L3 cache) and 128 GiB of main memory (16×8 GiB DDR4-2133 Hynix HMA42GR7AFR4N-TF). The operating system is Red Hat 4.8 with Linux kernel version 3.10.0-327.13.1.el7.x86_64. The C compiler is `gcc 4.8.3` and CUDA version is `7.5.18`. The host and the accelerator are connected by a 16-lane PCI Express 3.0 bus.

P100 GPU compute node. An NVIDIA Tesla P100 Accelerator with one 1189-MHz NVIDIA GP100 GPU (Pascal microarchitecture, 3584 cores, 56 SMX, 64 cores/SMX), 16384 MiB of on-device 4096-bit HBM2 memory with ECC enabled. The host is a Dell PowerEdge C4130 with two 2.54-GHz Intel Xeon E5-2680v3 CPU (Haswell microarchitecture, 24 cores, 12 cores/CPU, no hyper-threading, 30 MiB L3 cache) and 256 GiB of main memory (16×16 GiB DDR4-2133 Hynix HMA82GR7MFR8N-UH). The operating system is CentOS 7.3 with Linux kernel version 3.10.0-514.26.2.el7.x86_64. The C compiler is `gcc 5.0.3` and CUDA version is `8.0.61`. The host and the accelerator are connected by a 16-lane PCI Express 3.0 bus.

6.2 Input instances. The algorithm implementation receives as input fifteen $n \times n$ matrices modulo a 31-bit prime p, where n is a power of two. Apart from the value of n, the implementation is performance-oblivious to the input matrices when it evaluates the $\binom{6}{2}$-linear form (4) modulo p. That is, apart from the value of n, essentially identical performance is obtained regardless of the n-vertex host graph H and the six-vertex pattern graph S represented via the fifteen matrices (cf. §1.4).

Table 1 displays the problem parameters with different input sizes n. Our present experiments cover the range $n = 128$, $n = 256$, and $n = 512$. To accommodate subgraph counts up to $N = n(n-1)\cdots(n-5)$, we employ two distinct prime moduli p in our experiments.

6.3 A conservative lower bound. To set up a conservative lower bound for the running times that are feasible with implementation engineering, let us review the number of scalar multiplications that *must* be executed based on the arithmetic structure of the algorithm designs, and present an empirical measure for the peak achievable arithmetic bandwidth for such multiplications to obtain a lower bound.

First, to count the occurrences of a six-vertex sub-

graph in a 2^k-vertex host graph, the Nešetřil–Poljak [86] algorithm with Strassen's [104] factorization for the 2×2 matrix-multiplication tensor needs at least $B = 7^{2k}$ multiplications. Second, the Björklund–Kaski framework implemented with Strassen's factorization needs at least $C = 7 \cdot (3 \cdot 7^k - 2) \cdot 7^k$ modular multiplications to evaluate a proof polynomial modulo a prime at the required at least $d + 1 = 3 \cdot 7^k - 2$ points. Third, in terms of empirical performance, by relying on Montgomery multiplication [84], our implementation gives a peak empirical multiplication rate of approximately 215 billion multiplications per second for 31-bit prime fields on the P100 GPU compute node using a single NVIDIA Tesla P100 Accelerator. For the K80 GPU node using a single CUDA device (out of the two available devices) in the NVIDIA Tesla K80 Accelerator the peak rate is approximately 60 billion multiplications per second.[12]

For example, for $n = 512 = 2^9$ vertices, the arithmetic structure of the algorithms forces at least

$$B = 1{,}628{,}413{,}597{,}910{,}449$$

and

$$C = 34{,}196{,}683{,}861{,}267{,}935$$

multiplications. At 60 billion multiplications per second, the former takes 7.5 hours and the latter 6.6 days. Both lower bounds should be multiplied by two when comparing with the performance of our actual implementation on the same hardware in Table 2 since two 31-bit prime moduli were employed in the experiments.

6.4 Results. Table 2 displays data on experiments when we do not introduce errors into the polynomial evaluation and consequently can just interpolate the polynomial without decoding. These experiments are executed on our K80 GPU compute nodes. We observe that for our largest input size $n = 512$, the proof preparation takes 322 days of total computing time, whereas verifying the proof (which has size less than a gigabyte, cf. Table 1) takes only minutes. We also observe that the proof preparation time is less than two orders of magnitude from the conservative 13-day lower bound in §6.3.

Table 3 displays data on experiments where we introduce errors into the proof preparation stage. That is, we over-provision evaluations and randomly corrupt some of the evaluations so that the number of errors is still below the decoding threshold (cf. Footnote 4). We then error-correct the proof with our implementation of Gao's decoder [54], followed by verification of the result. These experiments are executed on our P100 GPU compute nodes.

We observe that the framework is practical as a proof-of-concept demonstration, with proof verification several orders of magnitude more efficient than proof preparation for the largest inputs considered. However, proof preparation still has at least one order of magnitude overhead compared with just solving the problem.

[12] Please note that these modular multiplication rates are substantially less than the available peak bandwidth for *floating-point* arithmetic due to the fact that the GPU hardware has been heavily optimized for the latter. Cf. §5.1 and the source code for our implementation of Montgomery multiplication. The accompanying source code [71] also contains performace-testing subroutines to measure the peak modular multiplication rate.

References

[1] A. Abboud, A. Backurs, and V. Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117. IEEE Computer Society, 2015.

[2] A. Abboud and A. Rubinstein. Distributed PCP theorems for hardness of approximation in P. *CoRR*, abs/1706.06407, 2017.

[3] F. N. Afrati, D. Fotakis, and J. D. Ullman. Enumerating subgraph instances using map-reduce. In C. S. Jensen, C. M. Jermaine, and X. Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 62–73. IEEE Computer Society, 2013.

[4] N. K. Ahmed, J. Neville, R. A. Rossi, and N. G. Duffield. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 1–10. IEEE Computer Society, 2015.

[5] N. Alon and S. Gutner. Balanced families of perfect hash functions and their applications. *ACM Trans. Algorithms*, 6(3):54:1–54:12, 2010.

[6] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[7] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[8] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[9] L. Babai. Trading group theory for randomness. In Sedgewick [101], pages 421–429.

[10] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In C. Koutsougeras and J. S. Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31. ACM, 1991.

Table 1: Problem parameters for six-vertex subgraph counting. For an n-vertex host graph with $n = 2^k$ we display (a) the naïve complexity $N = n(n-1)\cdots(n-5)$, (b) the Nešetřil–Poljak [86] baseline $B = 7^{2k}$ with Strassen's decomposition [104], (c) the degree $d = 3 \cdot 7^k - 3$ of the Björklund–Kaski proof polynomial, and (d) the size S of the proof in gibibytes (GiB) using two proof polynomials modulo 31-bit primes.

n	N	B	d	S
128	3,905,000,064,000	678,223,072,849	2,470,626	0.02
256	265,343,617,566,720	33,232,930,569,601	17,294,400	0.13
512	17,492,443,956,449,280	1,628,413,597,910,449	121,060,818	0.91
1024	1,136,126,223,187,845,120	79,792,266,297,612,001	847,425,744	6.32

Table 2: Experiments with the framework in the absence of errors. Timings for proof evaluation and interpolation (in the absence of errors) for six-vertex subgraph counting on the K80 GPU compute node using one CUDA device (out of two available devices) in an NVIDIA Tesla K80 Accelerator. Verification time is for 10 random points on an unoptimized single-threaded CPU implementation. Timings are for two 31-bit moduli and two proof polynomials to capture the subgraph count via the Chinese Remainder Theorem. The proof evaluation was distributed to multiple identical K80 GPU compute nodes; we display the total time taken by all nodes in evaluating the proof.

$n = 128$:	9431363.26 ms	to evaluate	(2.6 h)
	964843.63 ms	to interpolate	(16 min)
	9637.95 ms	to verify	(10 s)
$n = 256$:	430925138.97 ms	to evaluate	(5.0 days)
	7976254.34 ms	to interpolate	(2.3 h)
	68968.46 ms	to verify	(69 s)
$n = 512$:	27770318698.23 ms	to evaluate	(322 days)
	49481576.80 ms	to interpolate	(14 h)
	470549.08 ms	to verify	(8 min)

Table 3: Experiments with the framework in the presence of errors. Timings for proof evaluation and decoding in the presence of erroneous evaluations for six-vertex subgraph counting on the P100 GPU compute node using a single NVIDIA Tesla P100 Accelerator. We use 2.1% overprovisioning of evaluations and corrupt a subset of 1% of the evaluations uniformly at random. Decoding time includes listing the locations of the evaluations that were erroneous. Verification time is for 10 random points on an unoptimized single-threaded CPU implementation. Timings are for two 31-bit moduli and two proof polynomials to capture the subgraph count via the Chinese Remainder Theorem. The proof evaluation was distributed to multiple identical P100 GPU compute nodes; we display the total time taken by all nodes in evaluating the proof.

$n = 128$:	3837609.07 ms	to evaluate	(1.1 h)
	388717.58 ms	to decode	(6.5 min)
	8627.63 ms	to verify	(8.6 s)
$n = 256$:	162694846.33 ms	to evaluate	(45.2 h)
	3932064.66 ms	to decode	(1.1 h)
	540443.28 ms	to verify	(54 s)
$n = 512$:	14150000000.00 ms*	to evaluate	(164 days*)
	23128936.16 ms	to decode	(6.5 h)
	466734.40 ms	to verify	(7.8 min)

Please note: (*) The displayed evaluation time for $n = 512$ is an estimate based on 2542278 evaluations for both polynomials. To complete the subsequent experiments, we used 121060819 evaluations prepared earlier with the older K80 GPU compute nodes (cf. Table 2) to supplement the 2542278 evaluations to a total of 123603097 evaluations for the two proof polynomials to enable 1% random corruption, deconding, and verification experiments reported above.

[11] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[12] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[13] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Average-case fine-grained hardness. In Hatami et al. [61], pages 483–496.

[14] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Proofs of useful work. *IACR Cryptology ePrint Archive*, 2017:203, 2017.

[15] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In G. E. Blelloch and M. Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 193–204. ACM, 2012.

[16] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *J. ACM*, 59(6):32:1–32:23, 2012.

[17] L. A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013.

[18] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In Y. Li, B. Liu, and S. Sarawagi, editors, *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 16–24. ACM, 2008.

[19] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. On the concrete efficiency of probabilistically-checkable proofs. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 585–594. ACM, 2013.

[20] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In Canetti and Garay [31], pages 90–108.

[21] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[22] E. Ben-Sasson and M. Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

[23] A. R. Benson and G. Ballard. A framework for practical parallel fast matrix multiplication. In A. Cohen and D. Grove, editors, *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, San Francisco, CA, USA, February 7-11, 2015*, pages 42–53. ACM, 2015.

[24] M. Bentert, T. Fluschnik, A. Nichterlein, and R. Niedermeier. Parameterized aspects of triangle enumeration. *CoRR*, abs/1702.06548, 2017.

[25] B. Betkaoui, D. B. Thomas, W. Luk, and N. Pržulj. A framework for FPGA acceleration of large graph problems: Graphlet counting case study. In *2011 International Conference on Field-Programmable Technology, FPT 2011, New Delhi, India, December 12-14, 2011*, pages 1–8. IEEE, 2011.

[26] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Counting paths and packings in halves. In A. Fiat and P. Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2009.

[27] A. Björklund and P. Kaski. How proofs are prepared at Camelot: extended abstract. In G. Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 391–400. ACM, 2016.

[28] A. Björklund, P. Kaski, and L. Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. In C. Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 594–603. SIAM, 2014.

[29] A. Björklund, R. Pagh, V. Vassilevska Williams, and U. Zwick. Listing triangles. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014.

[30] S. Caminiti, I. Finocchi, E. G. Fusco, and F. Silvestri. Resilient dynamic programming. *Algorithmica*, 77(2):389–425, 2017.

[31] R. Canetti and J. A. Garay, editors. *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*. Springer, 2013.

[32] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270. ACM, 2016.

[33] V. Chen, E. Grigorescu, and R. de Wolf. Error-correcting data structures. *SIAM J. Comput.*, 42(1):84–111, 2013.

[34] Y. Chen and J. Flum. On parameterized path and chordless path problems. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, pages 250–263. IEEE Computer Society, 2007.

[35] Y. Chen, M. Thurley, and M. Weyer. Understanding the complexity of induced subgraph isomorphisms. In *Automata, Languages and Programming, 35th In-*

ternational Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games, volume 5125 of *Lecture Notes in Computer Science*, pages 587–596. Springer, 2008.

[36] S. Chu and J. Cheng. Triangle listing in massive networks. *TKDD*, 6(4):17:1–17:32, 2012.

[37] F. Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013.

[38] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 90–112. ACM, 2012.

[39] R. Curticapean. Counting matchings of size k is #W[1]-hard. In F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2013.

[40] R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In Hatami et al. [61], pages 210–223.

[41] R. Curticapean and D. Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139. IEEE Computer Society, 2014.

[42] J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, 2013.

[43] I. Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[44] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoret. Comput. Sci.*, 326(1-3):57–67, 2004.

[45] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Distributed estimation of graph 4-profiles. In J. Bourdeau, J. Hendler, R. Nkambou, I. Horrocks, and B. Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 483–493. ACM, 2016.

[46] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[47] I. Finocchi, M. Finocchi, and E. G. Fusco. Clique counting in MapReduce: Algorithms and experiments. *ACM Journal of Experimental Algorithmics*, 20:1.7:1–1.7:20, 2015.

[48] I. Finocchi, F. Grandoni, and G. F. Italiano. Resilient dictionaries. *ACM Trans. Algorithms*, 6(1):1:1–1:19, 2009.

[49] P. Floderus, M. Kowaluk, A. Lingas, and E. Lundell. Detecting and counting small pattern graphs. *SIAM J. Discrete Math.*, 29(3):1322–1339, 2015.

[50] J. Flum and M. Grohe. The parameterized complexity of counting problems. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, page 538. IEEE Computer Society, 2002.

[51] F. V. Fomin, D. Lokshtanov, V. Raman, S. Saurabh, and B. V. R. Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. Syst. Sci.*, 78(3):698–706, 2012.

[52] R. Freivalds. Probabilistic machines can use less running time. In *IFIP Congress*, pages 839–842, 1977.

[53] M. Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009.

[54] S. Gao. A new algorithm for decoding Reed–Solomon codes. In V. K. Bhargava, H. V. Poor, V. Tarokh, and S. Yoon, editors, *Communications, Information, and Network Security*, pages 55–68. Springer, 2003.

[55] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[56] O. Goldreich and G. N. Rothblum. Simple doubly-efficient interactive proof systems for locally-characterizable sets. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:18, 2017.

[57] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[58] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Sedgewick [101], pages 291–304.

[59] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[60] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[61] H. Hatami, P. McKenzie, and V. King, editors. *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. ACM, 2017.

[62] T. Herault and Y. Robert, editors. *Fault-Tolerance Techniques in High-Performance Computing*. Springer, Cham, 2015.

[63] T. Hočevar and J. Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4):559–565, 2014.

[64] X. Hu, M. Qiao, and Y. Tao. I/O-efficient join dependency testing, Loomis-Whitney join, and triangle enumeration. *J. Comput. Syst. Sci.*, 82(8):1300–1315, 2016.

[65] J. Huang, L. Rice, D. A. Matthews, and R. A. van de Geijn. Generating families of practical fast matrix multiplication algorithms. In *2017 IEEE Interna-*

tional Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, FL, USA, May 29 - June 2, 2017, pages 656–667. IEEE Computer Society, 2017.

[66] K. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Computers*, 33(6):518–528, 1984.

[67] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[68] Z. Jafargholi and E. Viola. 3SUM, 3XOR, triangles. *Algorithmica*, 74(1):326–343, 2016.

[69] M. Jerrum and K. Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *J. Comput. Syst. Sci.*, 81(4):702–716, 2015.

[70] M. Jerrum and K. Meeks. Some hard families of parameterized counting problems. *TOCT*, 7(3):11:1–11:18, 2015.

[71] P. Kaski. A delegatable and error-tolerant algorithm for counting of six-vertex subgraphs in a graph, 2017. doi:10.5281/zenodo.1039719, `https://github.com/pkaski/six-subgraph`.

[72] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000.

[73] M. N. Kolountzakis, G. L. Miller, R. Peng, and C. E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.

[74] S. Lagraa and H. Seba. An efficient exact algorithm for triangle listing in large graphs. *Data Min. Knowl. Discov.*, 30(5):1350–1369, 2016.

[75] F. Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014.

[76] S.-J. Lin, T. Y. Al-Naffouri, Y. S. Han, and W.-H. Chung. Novel polynomial basis with fast Fourier transform and its application to Reed-Solomon erasure codes. *IEEE Trans. Inform. Theory*, 62(11):6284–6299, 2016.

[77] B. Lipshitz, G. Ballard, J. Demmel, and O. Schwartz. Communication-avoiding parallel Strassen: implementation and performance. In J. K. Hollingsworth, editor, *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012*, page 101. IEEE/ACM, 2012.

[78] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[79] C. D. Martino, Z. T. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer. Lessons learned from the analysis of system failures at petascale: The case of Blue Waters. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, pages 610–621. IEEE Computer Society, 2014.

[80] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[81] K. Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016.

[82] X. Mei and X. Chu. Dissecting GPU memory hierarchy through microbenchmarking. *IEEE Trans. Parallel Distrib. Syst.*, 28(1):72–86, 2017.

[83] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015*, pages 415–426. IEEE Computer Society, 2015.

[84] P. L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, 44(170):519–521, 1985.

[85] J. Nederlof. A short note on Merlin-Arthur protocols for subset sum. *Inf. Process. Lett.*, 118:15–16, 2017.

[86] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carolin.*, 26(2):415–419, 1985.

[87] S. Olariu. Paw-free graphs. *Inf. Process. Lett.*, 28(1):53–54, 1988.

[88] H. Park, F. Silvestri, U. Kang, and R. Pagh. MapReduce triangle enumeration with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1739–1748. ACM, 2014.

[89] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112, 2016.

[90] A. Pinar, C. Seshadhri, and V. Vishal. ESCAPE: efficiently counting all 5-vertex subgraphs. In R. Barrett, R. Cummings, E. Agichtein, and E. Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1431–1440. ACM, 2017.

[91] N. Pržulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004.

[92] M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010.

[93] M. Rahman, M. Alam Bhuiyan, and M. Al Hasan. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Trans. Knowl. Data Eng.*, 26(10):2466–2478, 2014.

[94] D. A. Reed and J. Dongarra. Exascale computing and big data. *Commun. ACM*, 58(7):56–68, 2015.

[95] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Indust. Appl. Math.*, 8:300–304, 1960.

[96] R. A. Rossi and R. Zhou. Leveraging multiple GPUs and CPUs for graphlet counting in large networks. In *Proceedings of the 25th ACM International Conference*

on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016, pages 1783–1792. ACM, 2016.

[97] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In S. E. Nikoletseas, editor, *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, volume 3503 of *Lecture Notes in Computer Science*, pages 606–609. Springer, 2005.

[98] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.

[99] B. Schroeder, E. Pinheiro, and W. Weber. DRAM errors in the wild: a large-scale field study. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems, SIGMETRICS/Performance 2009, Seattle, WA, USA, June 15-19, 2009*, pages 193–204. ACM, 2009.

[100] B. Schroeder, E. Pinheiro, and W. Weber. DRAM errors in the wild: a large-scale field study. *Commun. ACM*, 54(2):100–107, 2011.

[101] R. Sedgewick, editor. *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*. ACM, 1985.

[102] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[103] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen. Addressing failures in exascale computing. *IJHPCA*, 28(2):129–173, 2014.

[104] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[105] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In Canetti and Garay [31], pages 71–89.

[106] D. Tiwari, S. Gupta, G. Gallarno, J. Rogers, and D. Maxwell. Reliability lessons learned from GPU experience with the Titan supercomputer at oak ridge leadership computing facility. In J. Kern and J. S. Vetter, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, pages 38:1–38:12. ACM, 2015.

[107] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 837–846. ACM, 2009.

[108] J. Ugander, L. Backstrom, and J. M. Kleinberg. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 1307–1318. International World Wide Web Conferences Steering Committee / ACM, 2013.

[109] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In H. J. Karloff and T. Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012.

[110] V. Vassilevska Williams, J. R. Wang, R. R. Williams, and H. Yu. Finding four-node subgraphs in triangle time. In P. Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015.

[111] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010.

[112] V. Vassilevska Williams and R. Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.

[113] V. Volkov. *Understanding Latency Hiding on GPUs*. PhD thesis, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2016. Technical Report No. UCB/EECS-2016-143.

[114] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, third edition, 2013.

[115] V. Vu, S. T. V. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 223–237. IEEE Computer Society, 2013.

[116] R. S. Wahby, S. T. V. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[117] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.

[118] R. R. Williams. Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation. In R. Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[119] F. Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science, Harpenden, 1937.