Engineering Motif Search for Large Motifs

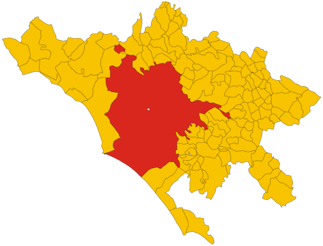
Petteri Kaski[1] Juho Lauri[2] Suhas Thejaswi[1]

[1]Department of Computer Science
Aalto University, Espoo, Finland

[2]Nokia Bell Labs
Dublin, Ireland

Symposium of Experimental Algorithms (SEA 2018)
L'Aquila, Italy, Friday 29 June 2018

Motivation



- **Modern computers are extremely powerful**
  — NVIDIA DGX-1 has 40960 cores
  — $\sim$6 terabytes/sec memory bandwidth

1 bit = $1\ cm^2$, 6 terabytes = $4800\ km^2$

- **Rome metropolitan city area is $5,352 km^2$**

Can we use these massively parallel microarchitectures to its fullest potential?

Empirical memory bandwidth

Outline

- Background on motif search
- **Engineering** a practical implementation of constrained multilinear sieving for massively **vector-parallel** microarchitectures (shared-memory multi-GPU systems)
- Experiments

What we want?

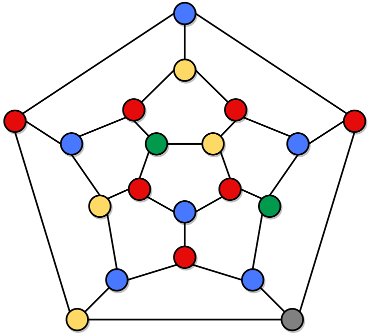
- vector parallelization
- saturate memory bandwidth
- offload to multiple GPUs

Motif search problem

Data

Vertex-colored graph H

(the **host graph**)



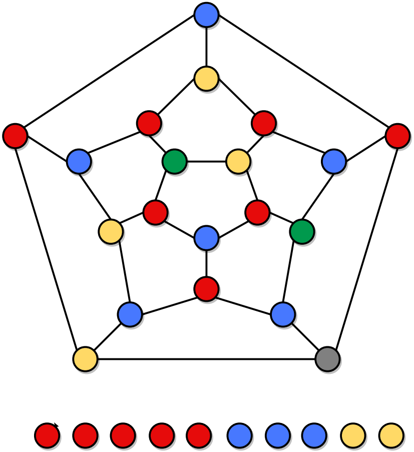
Query

Multiset M of colors

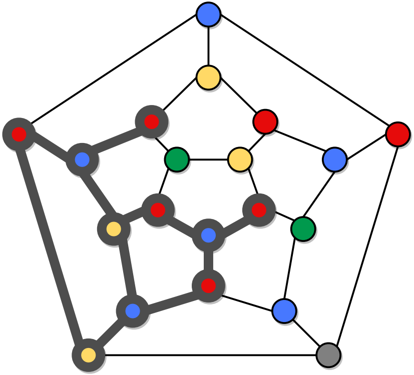
(the **motif**)

Query matches a connected subgraph?

Data, query, and one match



Data and query



Match

Complexity

- *NP*-complete if M has at least two colors
- Fixed-parameter tractable (FPT)
- Solvable in linear time in the size of H (*exponential in the size of M*)

Shown to be FPT by Fellows, Fertin, Hermelin, Vialette, ICALP 2007

FPT race

Authors	Time complexity	Conference
Fellows et al.	$O(\sim 87^k \text{poly}(n, m))$	ICALP 2007
Betzler et al.	$O(4.32^k \text{poly}(n, m))$	CPM 2008
Guillemot & Sikora	$O(4^k \text{poly}(n, m))$	MFCS 2010
Koutis	$O(2.54^k \text{poly}(n, m))$	IPL 2012
Björklund et al.	$O(2^k k^2 m)$	STACS 2013

n – number of vertices

m – number of edges

k – motif size

Algorithm

Constrained multilinear sieving

Algorithmica (2016) 74:947–967
DOI 10.1007/s00453-015-9981-1

CrossMark

Constrained Multilinear Detection and Generalized Graph Motifs

Andreas Björklund · Petteri Kaski · Łukasz Kowalik

Received: 14 May 2013 / Accepted: 23 February 2015 / Published online: 4 March 2015

Abstract We introduce a new algebraic sieving technique to detect constrained multilinear monomials in multivariate polynomial generating functions given by an evaluation oracle. The polynomials are assumed to have coefficients from a field of characteristic two. As applications of the technique, we show an $O^*(2^k)$-time polynomial space algorithm for the k-sized GRAPH MOTIF problem. We also introduce a new optimization variant of the problem, called CLOSEST GRAPH MOTIF and solve it within the same time bound. The CLOSEST GRAPH MOTIF problem encompasses several previously studied optimization variants, like MAXIMUM GRAPH MOTIF, MIN-SUBSTITUTE GRAPH MOTIF, and MIN-ADD GRAPH MOTIF. Finally, we provide a piece of evidence that our result might be essentially tight: the existence of an $O^*((2-\epsilon)^k)$-time algorithm for the GRAPH MOTIF problem implies an $O((2-\epsilon')^n)$-time algorithm for SET COVER.

Converting a *combinatorial problem* to an *algebraic problem* (detecting a multilinear monomial in a multivariate polynomial)

- Björklund, Kaski and Kowalik STACS-2013/Algorithmica-2016
- Randomized decision algorithm (YES/NO)
- YES, always correct *no false positives*
- NO, false-negative probability $k \cdot 2^{-b+1}$

Arithmetic over $GF(2^b)$

High-level algorithm (Björklund, Kaski, Kowalik)

Output YES if and only if the sum of 2^k **evaluations** of a **multivariate polynomial** $P(x, y)$ is non-zero

- 2^k **evaluations**: 2^k points $(x^{(1)}, y^{(1)}),\ (x^{(2)}, y^{(2)}),\ \cdots\ ,\ (x^{(2^k)}, y^{(2^k)})$ depend on motif M and random bits
- **Multivariate polynomial**: defined by host graph H, has $n + 2(k-1)m$ variables and degree $2k - 1$
- Evaluation algorithm runtime $O(k^2 m M(b))$
- Overall runtime $O(2^k k^2 m M(b))$

In practice it works for small k and large m

$M(b)$ complexity to multiply in $GF(2^b)$

CPU implementation (ALENEX 2015)

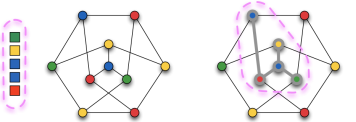
Engineering Motif Search for Large Graphs*

Andreas Björklund† Petteri Kaski‡ Łukasz Kowalik§ Juho Lauri¶

Abstract

In the *graph motif problem*, we are given as input a vertex-colored graph H (the host graph) and a multiset of colors M (the motif). Our task is to decide whether H has a connected set of vertices whose multiset of colors agrees with M. The graph motif problem is NP-complete but known to admit parameterized algorithms that run in linear time in the size of H. We demonstrate that algorithms based on *constrained multilinear sieving* are viable in practice, scaling to graphs with hundreds of millions of edges as long as M remains small. Furthermore, our implementation is *topology-invariant* relative to the host graph H, meaning only the most crude graph parameters (number of edges and number of vertices) suffice in practice to determine the algorithm performance.

Example. The motif and the host (left); a connected set that matches the motif (right).



From an algorithm theory perspective, in particular from the perspective of *parameterized algorithms* [13], the graph motif problem is known to have a janusian nature: it is (i) NP-complete, but (ii) admits algorithms

Large graphs — how about large motifs?

Open source — https://github.com/pkaski/motif-search

Exponential complexity in motif size

Design considerations

Positives

- High arithmetic and memory bandwidth
- Massive vector-parallel microarchitecture

  — roughly 40,000 cores

Negatives

- High memory latency

  — bandwidth comes at the cost of latency

- Lack of hardware support for finite-field arithmetic

  — PCLMULQDQinstruction set speeds up finite-field arithmetic in CPUs

Design considerations

Using available bandwidth

- Keeping pipeline busy

  — memory access and arithmetic operations simultaneously

- Coalesced memory access
- Bit-sliced finite-field arithmetic

transition here

Vertex localized sieve

Base case, for all $i \in [n]$ and $L \subseteq [k]$

$$P_{i,1}(\zeta^L, \alpha) = \zeta_i^L$$

For each $s = 2, 3, \ldots, k$, $i \in [n]$, and $L \subseteq [k]$

$$P_{i,s}(\zeta^L, \alpha) = \sum_{j \in \Gamma_H(i)} \alpha_{s,(i,j)} \sum_{\substack{s_1+s_2=s \\ s_1,s_2 \geq 1}} P_{i,s_1}(\zeta^L, \alpha) P_{j,s_2}(\zeta^L, \alpha)$$

Finally, sum at each vertex

$$Q_{i,k}(\mu, \nu, \alpha) = \sum_{L \subseteq [k]} P_{i,k}(\zeta^L, \alpha)$$

Parallelization over vertices $i \in [n]$ (n threads) and $L \subseteq [k]$ (2^k threads)
Parallelization over L vectorizes upto 2^k threads

Inner loop in CUDA

For each $s = 2, 3, \ldots, k$, $i \in [n]$, and $L \subseteq [k]$

$$P_{i,s}(\zeta^L, \alpha) = \sum_{j \in \Gamma_H(i)} \alpha_{s,(i,j)} \boxed{\sum_{\substack{s_1+s_2=s \\ s_1,s_2 \geq 1}} P_{i,s_1}(\zeta^L, \alpha) P_{j,s_2}(\zeta^L, \alpha)}$$

```
for(index_t s1 = 1; s1 < s; s1++) {
  index_t s2 = s-s1;
  index_t s1i = LINE_IDX(n, gl, s1, i, a);
  line_t p_s1i;
  LINE_LOAD(p_s1i, d_s, seg, s1i);          /* Load P_{i,s1} */
  index_t s2j = LINE_IDX(n, gl, s2, j, a);
  line_t p_s2j;
  LINE_LOAD(p_s2j, d_s, seg, s2j);          /* Load P_{j,s2} */
  line_t p_s1i_s2j;
  LINE_MUL(p_s1i_s2j, p_s1i, p_s2j);        /* Line multiplication */
  LINE_ADD(p_sij, p_sij, p_s1i_s2j);        /* Store result */
}
```

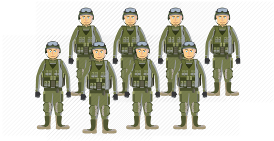
Workloads and uniformity

Project (vertex)

Workers (threads)

CPU workload

Project (vertex)



Workers (threads)

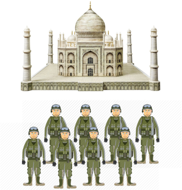
GPU workload

***D* workers (threads) work on a single project (vertex)**

D divides 2^k, execution in each thread of CPU is mostly independent

All threads (typically 32) in a GPU warp execute same instructions

Workloads and uniformity

Vertices	1	2	...	n
			...	
Threads	D workers	D workers	...	D workers

Workloads of shape $n \times D$ (single GPU)

Workload of shape $M \times n \times D$ (M GPUs)

Each project (vertex) has different completion time

Memory layout and coalescence

Worker

X resources
A space
(each iteration)

Resources

S resources
U space
(total)

- Access $\frac{U}{A}$ space each iteration
- $n \times D$ workers

Memory layout shape $\frac{U}{A} \times n \times D \times A$

Resources = scalars, space = memory (words)
Each load/store access A words of data

Open source

https://github.com/pkaski/motif-localized

Experiments

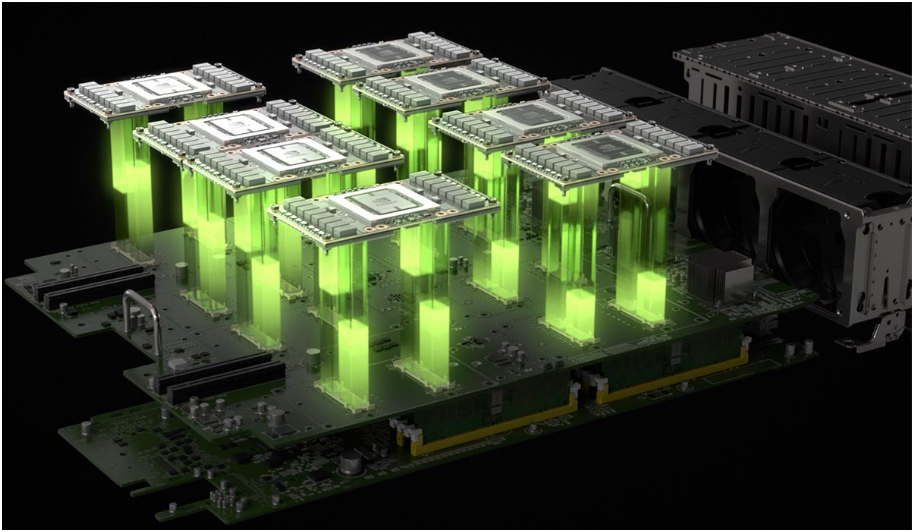


Image source: NVIDIA Corporation

Hardware configurations

- **CPU node**
  2 × 2.6-GHz Intel Xeon E5-2690v3 CPU
  Haswell microarchitecture, 12 cores/CPU
  30 MiB L3 cache, 128 GiB main memory
  (8 × 16 GiB DDR4-2133)

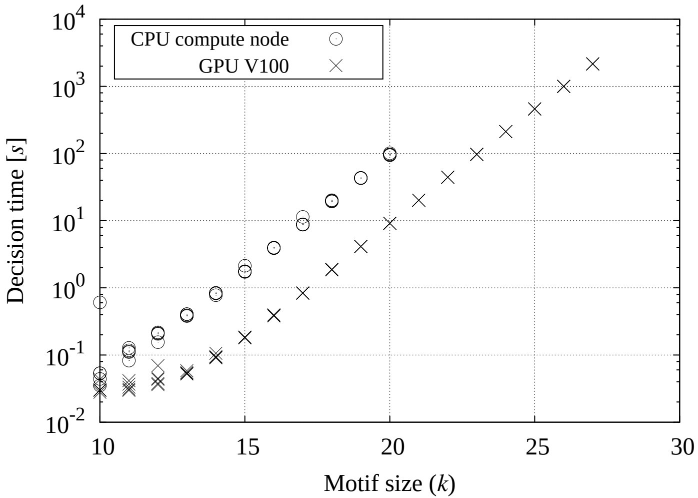
- **NVIDIA DGX-1**
  8 × 1312-GHz NVIDIA GV100 GPU
  Volta microarchitecture, 5120 cores/GPU
  (40960 cores), 128 GiB of on-device
  memory (8 × 16 GiB 4096-bit HBM2)

Image source: Intel corporation, NVIDIA corporation

Experiments

- Scaling as k increases (fixed m)
  - observe exponential scaling
- Scaling as m increases (fixed k)
  - observe linear scaling
- Topology invariance
  - graph topology should not matter much
- Error rate (false-negative probability)
  - repeats required to find all vertices with at least one match

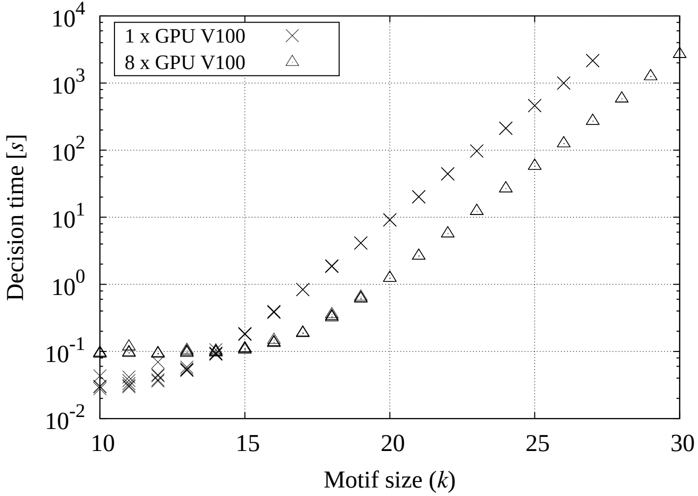
Runtime – motif size scaling (k)



Offloading to GPU pays off

GPU linetype – $32 \times GF(2^8)$ bit-sliced, CPU linetype – $64 \times GF(2^8)$ bit-packed
Random d-regular graphs ($m \sim 10^4$ fixed)

Runtime – motif size scaling (k)



Offloading to multiple GPUs pays off

$32 \times GF(2^8)$ bit-sliced linetype, random d-regular graphs ($m \sim 10^4$ fixed)

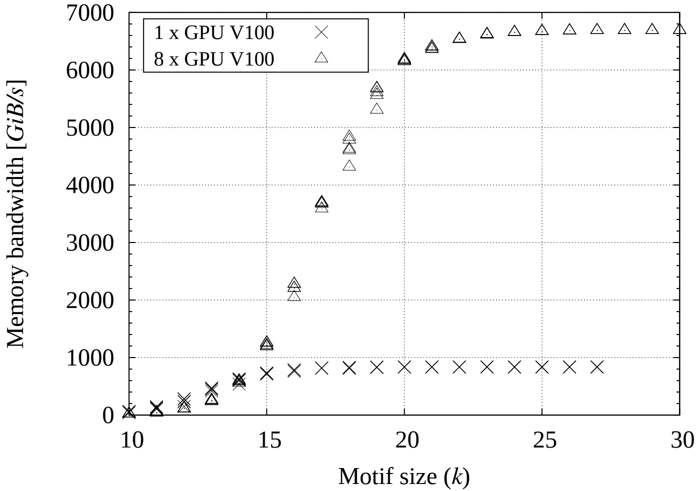
Speedup

k	CPU compute node	NVIDIA DGX-1	Speedup
11	0.0828 s	0.1180 s	0.70
12	0.1553 s	0.0938 s	1.66
13	0.3808 s	0.1046 s	3.64
14	0.7768 s	0.1025 s	7.58
15	1.7244 s	0.1111 s	15.52
16	3.9035 s	0.1474 s	26.48
17	8.7340 s	0.1906 s	45.82
18	19.3674 s	0.3564 s	54.34
19	42.9873 s	0.6480 s	66.34
20	94.2593 s	1.2425 s	75.86

CPU implementation is multi-threaded with vector-extensions (AVX-2)
(Björklund, Kaski, Kowalik, Lauri, ALENEX 2015)

GPU linetype – $32 \times GF(2^8)$ bit-sliced, CPU linetype – $64 \times GF(2^8)$ bit-packed
Random d-regular graphs ($m \sim 10^4$ fixed)

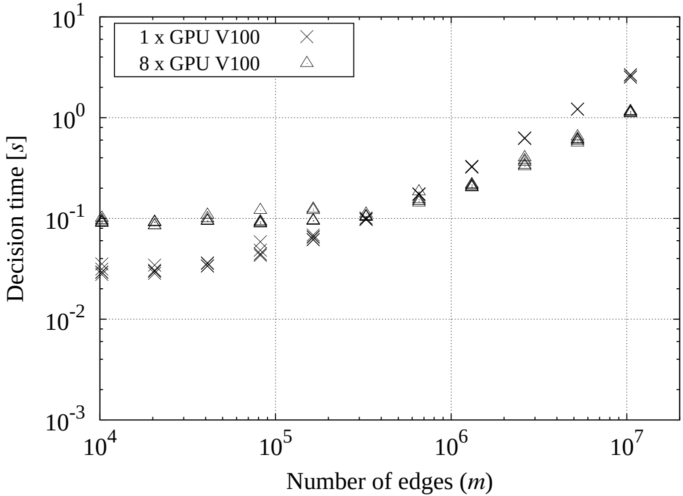
Memory bandwidth – motif size scaling (k)



More than six terabytes of memory bandwidth

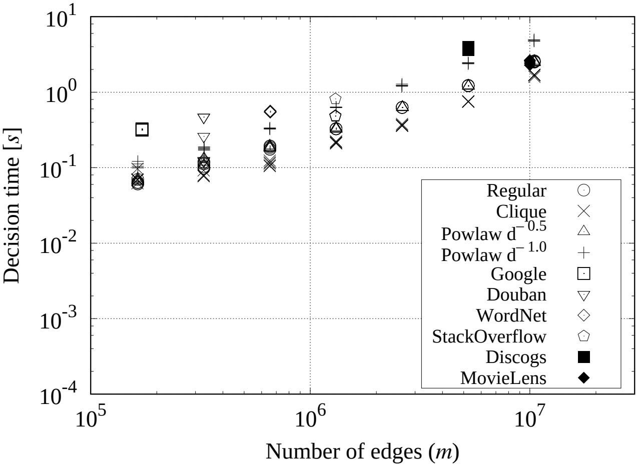
$32 \times GF(2^8)$ bit-sliced linetype, random d-regular graphs ($m \sim 10^4$ fixed)

Runtime – edge linear scaling (m)



$32 \times GF(2^8)$ bit-sliced linetype, random d-regular graphs ($k = 10$ fixed)

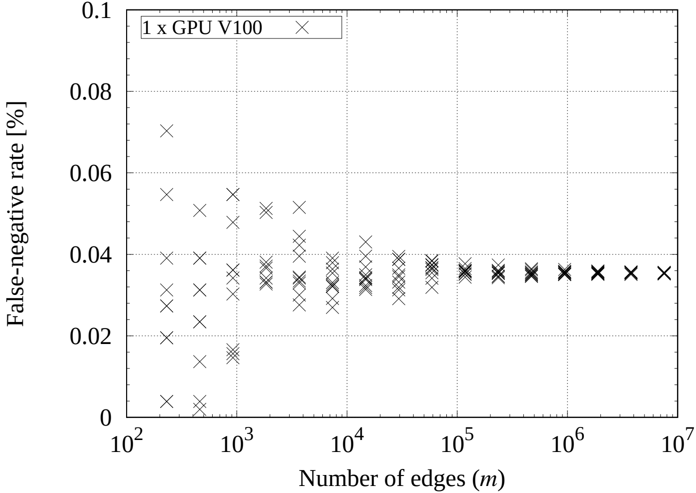
Topology invariance



Current implementation is not topology invariant

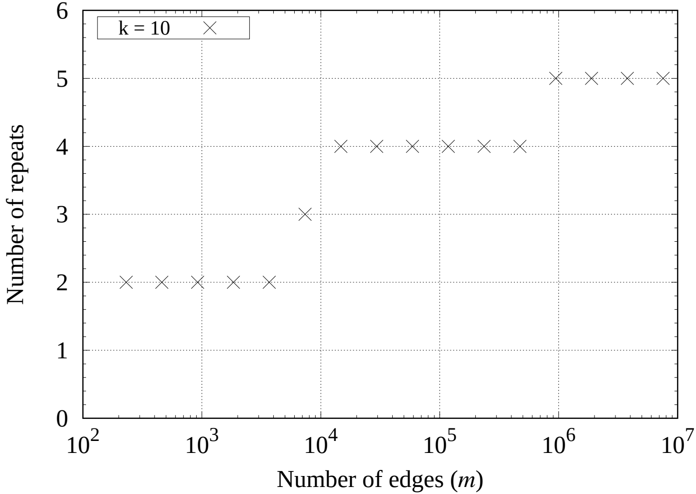
Different workloads due to varying degree of vertices. Arbitrary graph topology means arbitrary memory accesses, $32 \times GF(2^8)$ bit-sliced linetype, motif size $k = 10$ fixed

False-negative probability (vertex-localization)



$32 \times GF(2^8)$ bit-sliced linetype, k-path graph ($k = 10$ fixed)
Each vertex is incident to exactly one match

Number of repeats (vertex-localization)



$32 \times GF(2^8)$ bit-sliced linetype, k-path graph with motif size $k = 10$ fixed
Each vertex is incident to exactly one match

Summary

- Motif search is practical for small m, large k
- With sufficient implementation effort GPUs can outperform CPUs in motif search
  — for large k vectorization and offloading to multiple-GPUs pays off
- It is possible to saturate empirical memory bandwidth simultaneously performing arithmetic calculations
- Bit-sliced finite-field arithmetic to overcome the lack of hardware support
  — multiple repeats can overcome high false-negative probability of small field size

Summary

- Motif search is practical for small m, large k
- With sufficient implementation effort GPUs can outperform CPUs in motif search
  — for large k vectorization and offloading to multiple-GPUs pays off
- It is possible to saturate empirical memory bandwidth simultaneously performing arithmetic calculations
- Bit-sliced finite-field arithmetic to overcome the lack of hardware support
  — multiple repeats can overcome high false-negative probability of small field size

https://github.com/pkaski/motif-localized

Thank you