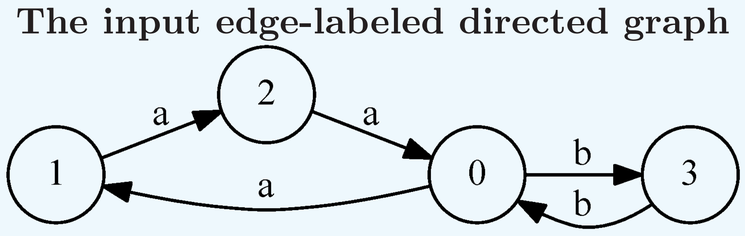
Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication

Semyon Grigorev
JetBrains Research, Saint Petersburg State University, Russia
kajigor@gmail.com

Contex-Free Path Querying

The input edge-labeled directed graph



The grammar for the language $L = \{a^n b^n\}$

$$S \to A\ B \mid A\ S_1 \qquad A \to a$$
$$S_1 \to S\ B \qquad B \to b$$

The result: set of node pairs such that there exists a path between them which forms a word from the L.

Matrix Multiplication

T — adjacency matrix
The grammar in the normal form

$$T_{ij} = \{N \mid N \overset{*}{\Rightarrow} \omega, \omega \text{ path bw } i \text{ and } j\}$$
$$T_{ik} \times T_{kj} = \{A \mid B \in T_{ik}, C \in T_{kj}, A \to BC\}$$
$$T^{(i)} = T^{(i-1)} \cup (T^{(i-1)} \times T^{(i-1)})$$

Questions

- Does GPGPUs utilization for CFPQ improve performance in comparison with the CPU version?
- Is it possible to achieve higher performance by using existing libraries for operations over matrices or do we need to create our own specialized solution?
- Can we achieve high performance with high-level languages?
- Can we improve performance with sparse matrix representation?

Implementations

[Scipy] Sparse matrices multiplication by using **Scipy** in **Python**

[M4RI] Dense matrices multiplication by using **m4ri** library which implements the Method of Four Russians in **C**

Answers

- GPGPUs utilization significantly increases the performance of CFPQ
- High performance libraries utilization is a good idea
- Automatic translation from a high-level language to GPGPU language provides a good balance between performance and implementation complexity
- Sparse matrix representation is important for performance

We need more data!

RDF			Query G_4				
Name	#V	#E	Scipy	M4RI	GPU_N	CuSprs	CYK
atm-prim	291	685	3	2	1	269	515285
biomed	341	711	3	5	1	283	420604
pizza	671	2604	6	8	1	292	3233587
wine	733	2450	7	6	1	294	4075319

Table 1: Query $s \to SCOR\ s\ SCO \mid TR\ s\ T \mid SCOR\ SCO \mid TR\ T$

Scaling

Graph	Scipy	M4RI	GPU_N	CuSprs
G10k-0.001	37.286	2.395	0.215	35.937
G10k-0.1	601.182	1.050	0.114	395.393
G40k-0.001	-	97.841	8.393	-
G80k-0.001	-	1142.959	65.886	-
25000	-	33.236	5.314	-
50000	-	360.035	44.611	-
80000	-	1292.817	190.343	-

Contact us

Everything is available on GitHub:
https://github.com/YaccConstructor

References

Acknowledgments

The research is supported by the JetBrains Research grant and the Russian Science Foundation grant 18-11-00100.