Efficient implementation of the model of associative computing on GPUs.

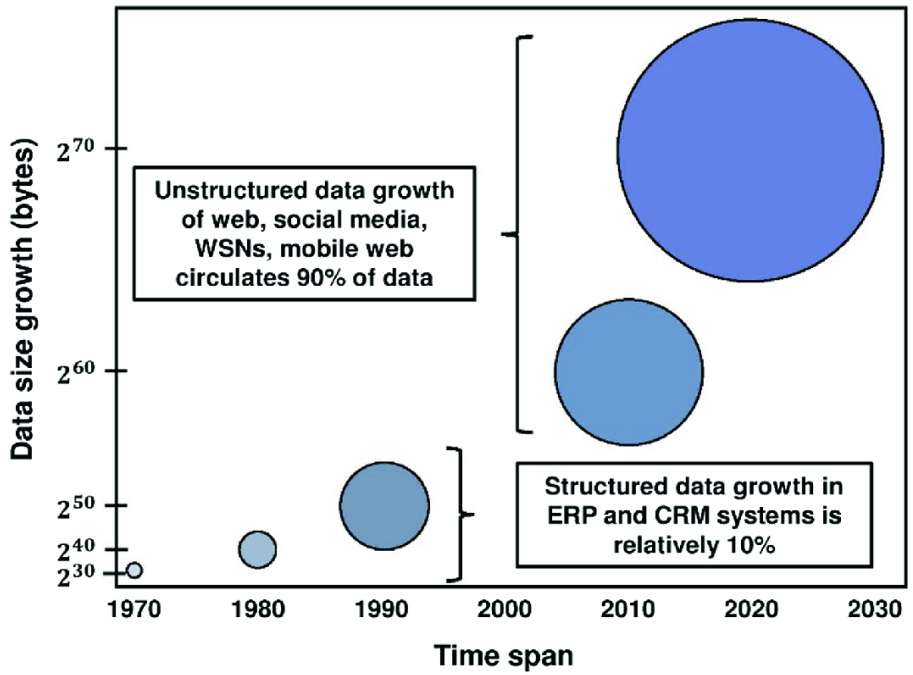
T. V. Snytnikova

05.13.11 - mathematical and software for computers, complexes and computer networks

The Institute of Computational Mathematics
and Mathematical Geophysics SB RAS

The relevance of the work

- The amount of information doubles every 18 months;
- the most of it is unstructured (it is not organized into databases or knowledge bases);



Search algorithms for unsorted data and sorting algorithms are a bottleneck for von Neumann type computers.

Accociative parallel models and architectures have the following property:

the execution time of basic search operations does not depend on the number of rows.

Basic concepts

- STAR machine is an abstract model of associative data processing. It includes architectural features and a special programming language for associative processing called the Star language.
- The Star language is an extension of Pascal. It includes:
  - data types *table*, *slice* and *word*;
  - operations on these types;
  - library of basic procedures.
- A GPU implementation of the STAR machine is a library written in CUDA C.
  It models data types and operations on them with preserving associative properties.
  It also includes a module implemented library of Star basic procedures and a data input/output module.

The purpose of the Ph.D. thesis is

develop software tools for the effective implementation of associative parallel algorithms on GPUs.

To achieve this goal, the following problems were solved:

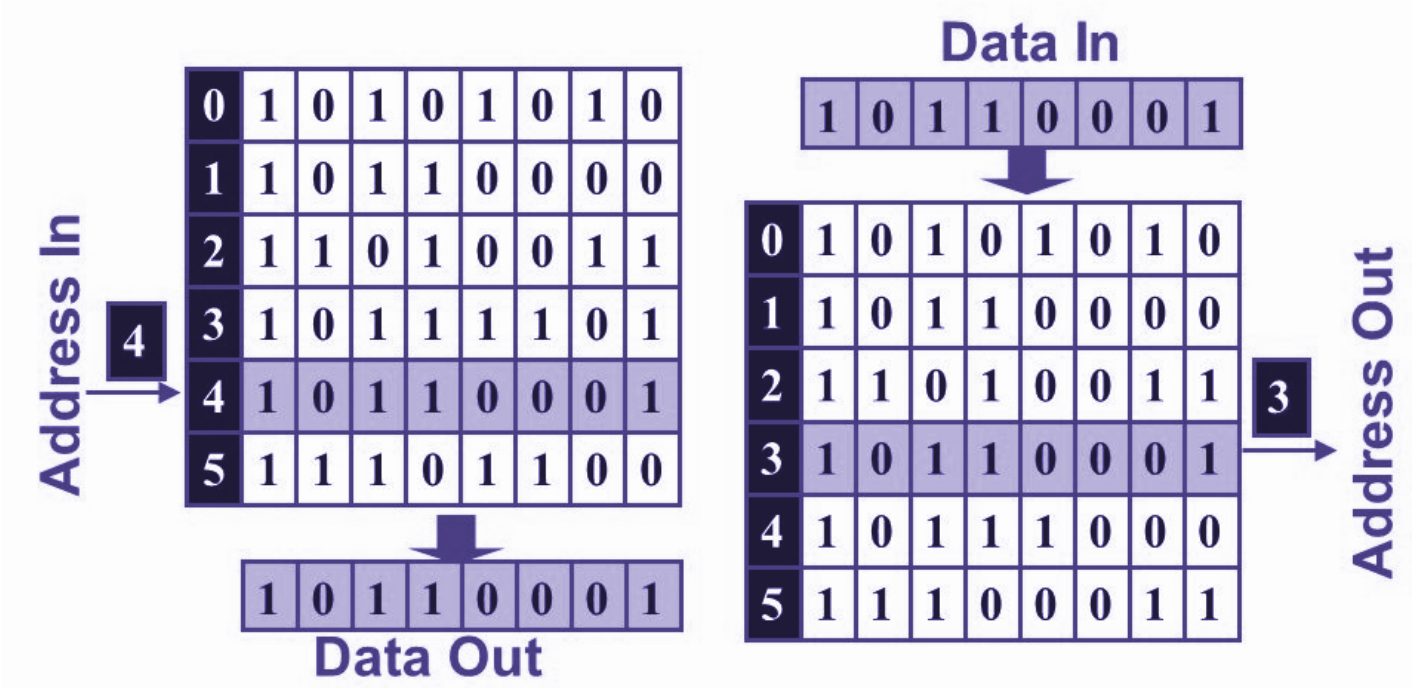
- the implementation of the basic operations of the Star language on the GPU is built;
- to increase the efficiency of implementation, the operations of the Star language that are critical to synchronization are defined;
- the data input/output module are developed;
- basic procedures of the Star language are implemented;
- the effectiveness of the implementation of the Star-machine was proved both by an assessment of the theoretical complexity of the implementation procedures, and by a practical comparison of the operating time with the operating time of analogues;
- methods have been developed for optimization of associative algorithms for execution on GPUs, taking into account the architectural differences between the Star machine and the GPU.

The following provisions and results are submitted for pesentation of Ph.D. thesis:

- synchronization-critical operations of the Star language are identified;
- the implementation of the STAR machine on the GPU is built;
- A classification of the basic procedures of the Star language is developed according to the method of data processing;
- methods have been developed for optimizing associative algorithms for execution on graphics accelerators, taking into account the architectural differences between the Star machine and the GPU;
- the effectiveness of the implementation of the Star-machine was proved both by an assessment of the theoretical complexity of the implementation procedures, and by a practical comparison of the operating time with the operating time of analogues.

Chapter I. Development of associative parallel models and architectures: memory access.

- RAM (Random Access Memory);
- CAM (Content Addressable Memory).



CAMs are used in high-speed caches, network routers, specialized databases and knowledge bases.

Chapter I. Development of associative parallel models and architectures: associative processing principles.

1. INSTRUCTION STREAM (IS) PROPERTIES: The IS transmits an instruction to all cells per unit of time. **Active cells execute the commands while an inactive cells listen to but does not execute the commands.** The IS has the ability to activate cells.
2. CONSTANT TIME GLOBAL OPERATIONS: bitwise logical operations, **access to columns and rows of matrix memory for both reading and writing, active cell selection.**
3. ASSOCIATIVE PROPERTIES run in unit time:
   - the selected cell sends data to all others
   - The IS has the ability to select a senior responder from the set of active cells.
   - **Basic search operations ($=$, $<$, $>$, min, max) and arithmetic operations are performed in time proportional to the number of bit columns in the table, not the number of its rows.**

Chapter I. Development of associative parallel models and architectures: abstract models.

model ASC and MASC are based on Staran and ASPRO processors;

model Star machine is based on Staran.

Similarity of models are

- architecture of the ASC and Star machine;
- associative processing principles.

Differences:

- In the ASC and MASC models, the search is carried out for a byte or a word, synchronization is critical for all operations;
- The Star-machine processes data bitwise, some operations are not critical to synchronization (PEs are performed on their data).

Chapter I. Development of associative parallel models and architectures: associative architectures.

- **Staran (1972)** First commercially successful version of an associative processor.
- **Aspro (1982)** was created for US air traffic control systems. According to 1983 data, it was used in the radars of the E-2 Hawkeye AWACS carrier-based reconnaissance aircraft of the US Navy.
- IXM2 (1990-1994) was bilding for ASTRAL and EBMT machine translation systems, TDMT system for real-time speech translation;
- Rutgers CAM2000 (1993-1996) was developed with the support of NASA;
- **ATLAS FastTracker (2008-2016 and later)** was developed to serve the ATLAS LHC detector.
  Further applications are expected to be: medical imaging (tomography, etc.); video surveillance systems, smart cameras; tasks of high-speed data filtering; study of vision and other brain functions.

Chapter I. Development of associative parallel models and architectures. Conclusion.

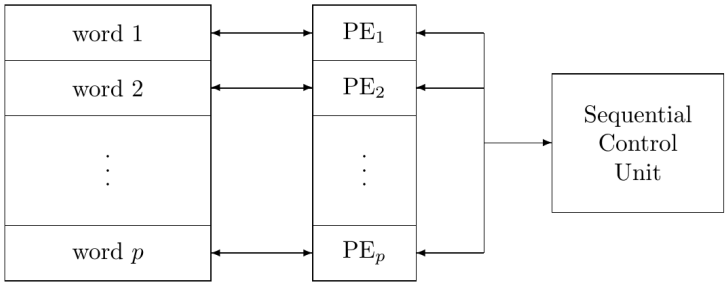
- All presented architectures were built to solve specific problems that could not be efficiently solved on PRAM architecture systems. (Parallel Random Access Machines).
- New Associative Processors (APs) are constantly being developed.
- Research is being done to make the AP suitable for portable devices.

- New associative algorithms are being developed.

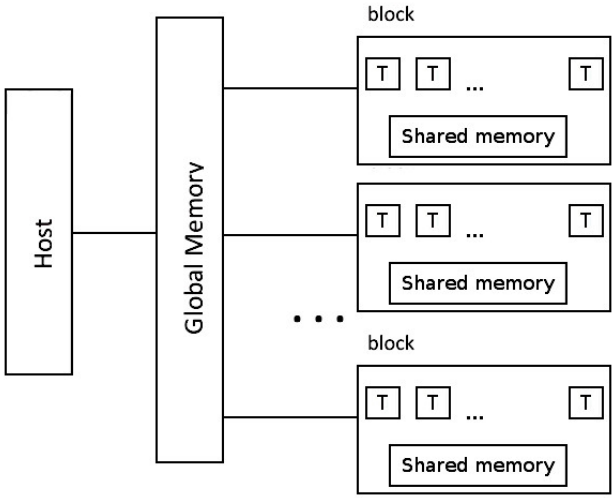
This implies the relevance of an efficient software implementation of the abstract model of associative processors (STAR-machines) on the modern architecture.

Chapter II. Section 1. Implementation of a STAR machine on the GPU: justification for the choice of architecture for implementation.

a) Model of Star-machine



b) Model of GPU



- class SIMD (Single Instruction Multiple Data), massive parallelism $p \sim 128, 256, 1024$
- synchronization:(instructions arrive one at a time, results are exchanged at each step).

GPU specialty:

\+ more than 6000 threads, global memory.

\+ 1D-3D block numeration.

\- No synchronization between blocks;

Chapter II. Section 1. Implementation of a STAR machine on the GPU: implementation difficulties and proposed solutions.

Difficulties	**Proposed solutions**
Synchronization of operations of all cells, both active and inactive.	Synchronization-critical operations are identified and implemented as single cores.
One time access to columns and rows of matrix memory.	Access to columns is organized by reference. Row access takes longer and needs synchronization.
The selected cell sends data to each others.	Global memory uses.
The IS selects a senior responder from the set of active cells	The operation is implemented with time complexity $O(log_{64}(N))$.

Implementation efficiency criterion: basic search operations (=, <, >, min, max) and arithmetic operations are performed in time proportional to the number of bit columns in the table, and not to the number of rows.

Chapter II. GPU Star Machine Implementation: Profiling Basic Operations

Operation	Runtime (μs)			
	100	1 000	3 000	5 000
ROW (read)	18,6 (18,5 – 19,7)	19, 5	19,6	19,8
ROW (write)	4,1 (2,9 – 5,6)	3,9	4,3	4,1
or	1,9 (1,9 – 3,0)	2,2	2,0	2,1
FND (position of firtht '1')	2,5 (2,3 – 2,8)	2,8	3,5	3,8
	3,2 (3,1 – 4,0)	3,2	3,3	3,4

The runtime of basic operations is practically independent of the data size, which corresponds to theoretical estimates:
$O(\log_{64}(n))$ for the implementation of the *FND*, *STEP*, *SOME* and *ZERO*,
constant time for other operations.

N^*	64	4 096	16 384	65 536	4 194 304	16 777 216	1 073 741 824
$log_{64}(N)$	0	1	2	3	4	5	6

Chapter II. Implementation of the Star basic algorithm library.

Basic algorithms can be divided into the following groups according to work with tables:

- synchronization-critical basic operations are used: MIN and MAX;
- synchronization-critical basic operations aren't used, the control slice changes during the execution: search for table rows that match a pattern MATCH, pattern string comparison GREAT, LESS, GEL, comparing two tables row by row SETMIN, SETMAX, HIT;
- arithmetic algorithms: adding a word to table rows ADDC, row-by-row addition of tables ADDV, subtracting a word from table rows SUBTC, row-by-row subtraction of one table from another SUBTV;
- synchronization-critical basic operations aren't used, the control slice doesn't change: CLEAR, TMERGE, WMERGE, WCOPY, TCOPY, TCOPY1, TCOPY2.

Chapter II. Time costs of standard associative algorithms.

Estimating the complexity of basic associative algorithms $O(h)$

Group of the algorithms	Complexity estimating	Data size (simultaneously)	Time μs	
			GF920m	k40
I	$O(h \cdot \lceil \log_{64}(N) \rceil)$	Γ¤Γ® 4 096 4 097 – 100 000	400 570 – 590	713 865 – 890
II	$O(h)$	100 000	58 – 61	71 – 73
III arithmetic algorithms	$O(h + \lceil \log_{64}(N) \rceil)$	50 000	36 – 41	52 – 58
IV	$O(1)$	10 000	6-10	8-9

Calculations were made on NVIDIA GeForce 920M and nks-30T cards (k40 node (Kepler)).

Chapter II. Recommendations for adapting associative algorithms for execution on the GPU.

Due to the architecture of the GPU STAR-machine, there are several possibilities for optimizing associative algorithms.

1. Determining synchronization points and reducing their number: reading / writing rows, loops and conditional statements with SOME or ZERO predicates.
2. Two-level parallelism. Check if there is a data dependency when processing columns.
3. Reducing the number of __global__-procedures. **__device__-procedures should be used for all operators that are not critical to synchronization and basic algorithms of II-IV groups.**

Chapter II. Implementation of the model of associative parallel computing. Conclusions.

- Basic types and operations of the STAR language implemented:
  - The runtime of basic operations is practically independent of the size of the data, which corresponds to theoretical estimates: $O(\log_{64}(n))$ to implement the operation *FND*, *SOME* ГЁ *STEP*;
  - ГЁ constant time to implement the remaining operations.
- The Star basic algorithm library implemented:
  - associative algorithms optimized for GPU architecture;
  - the runtime of procedures was compared with similar procedures of the STL and CUDA thrust libraries:
    - implementations of the presented algorithms on the GPU work faster than STL library procedures for vectors with dimensions greater than 5,000 elements;
    - the implementation of basic associative algorithms gives comparable performance with CUDA thrust library.
- Specific recommendations have been developed for adapting associative algorithms for execution on the GPU.

Chapter III. Associative Processing for Solving Graph Theory Problems

- Simple data types and operations based on associative computing paradigms made the universal model;
- It allows you to build parallel algorithms for solving problems on graphs:
  - the graph is processed by vertices, incoming/outgoing arcs are processed in parallel;
  - polynomial problems ($O(V^n) \to O(V^{n-1})$, $O(V+E) \to O(V)$);
  - dynamic algorithms;
  - In practice,implemented algorithms are designed for graphs with a size of $\approx 1000 - 10000$ vertices.

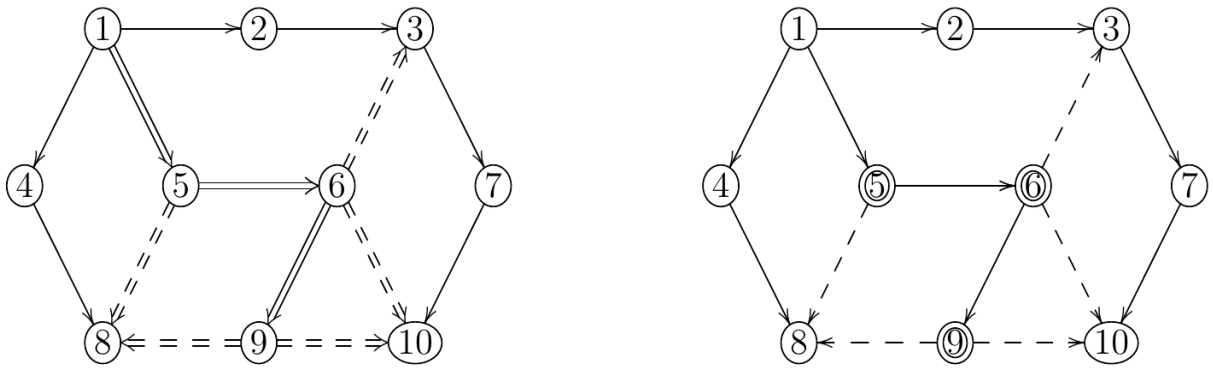
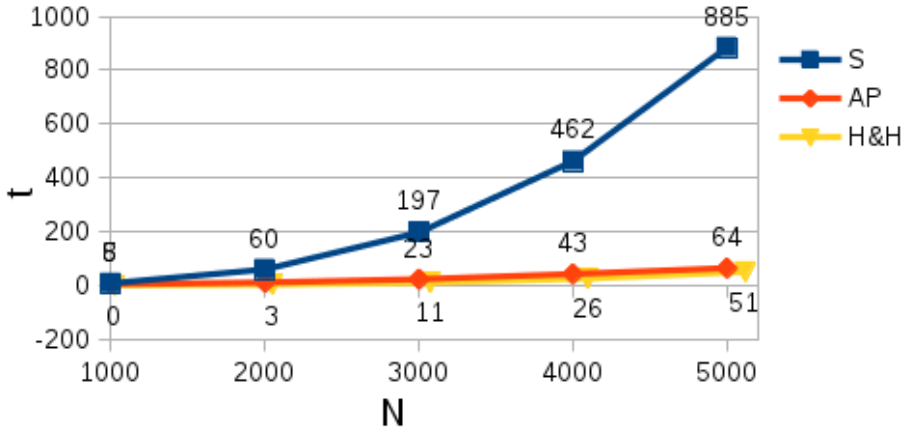


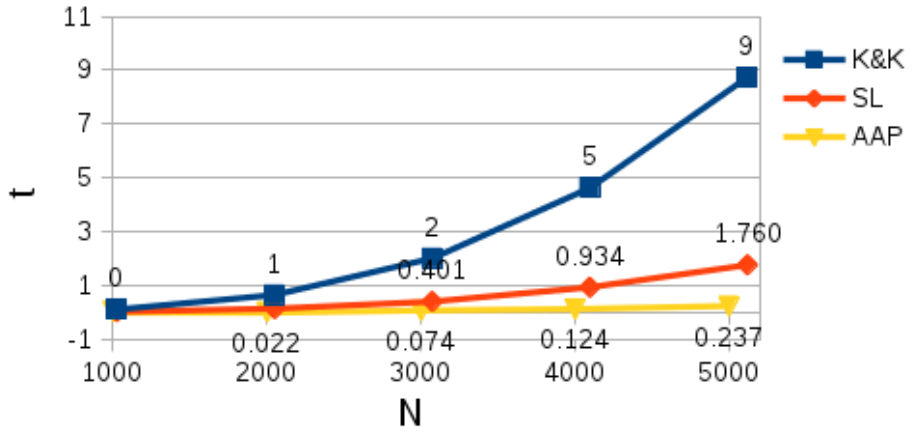
Рис.: Spanning tree on the set of reachable vertices after adding the arc (1, 5).

Chapter III. Star-algorithms for solving problems on graphs. GPU implementation.

- Warshall's algorithm is optimized for GPU execution with a theoretical complexity estimate of O(N);
- together with A. Sh. Nepomnyashchaya, associative versions of dynamic algorithms were developed:
  - Dijkstra's incremental algorithm for dynamically processing a shortest path tree after adding a new arc;
  - incremental and decremental Ramalingam's algorithms for solving the reachability problem in single source flow graphs;
- computational experiments were carried out for these algorithms.

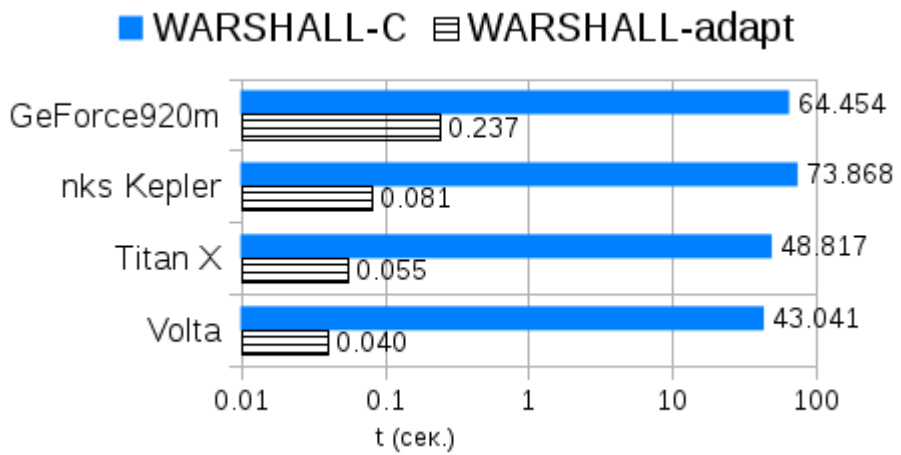
Chapter III. Warshall's algorithm. Comparison of the runtime of various implementations on the GPU





- **S** — Warshall's sequential algorithm;
- **AP** — **Star-algorithm; WARSHALL-C**;
- **N&H** — Narayanan's implementation of Warshall's algorithm;
- **AAP** — **adapting Star-algorithm WARSHALL-adapt**;
- **K&K** —block implementation;
- **SL** — block implementation using shared memory.

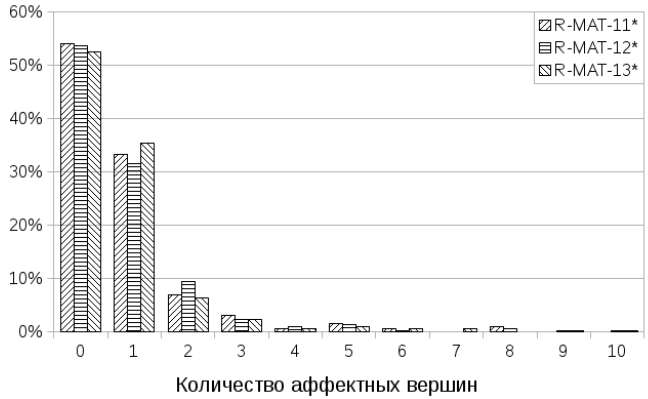
Chapter III. Warshall's algorithm. Comparison of operating time on different architectures of graphic accelerators.



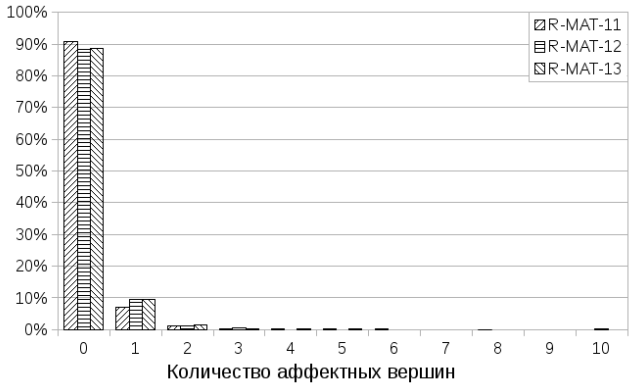
The calculation was carried out on a graph with 5000 vertices.

- **GeForce920m** — notebook;
- **nks Kepler** — cluster nks-30T SSCC (node k40 Kepler);
- **Titan X** — cluster Department of Information Technologies NSU;
- **Volta** — cluster NSU;

ГґГ«Г ГўГ III. Dijkstra's incremental algorithm. Results of the computational experiment.



Arc with weight 0



Arc with random weight

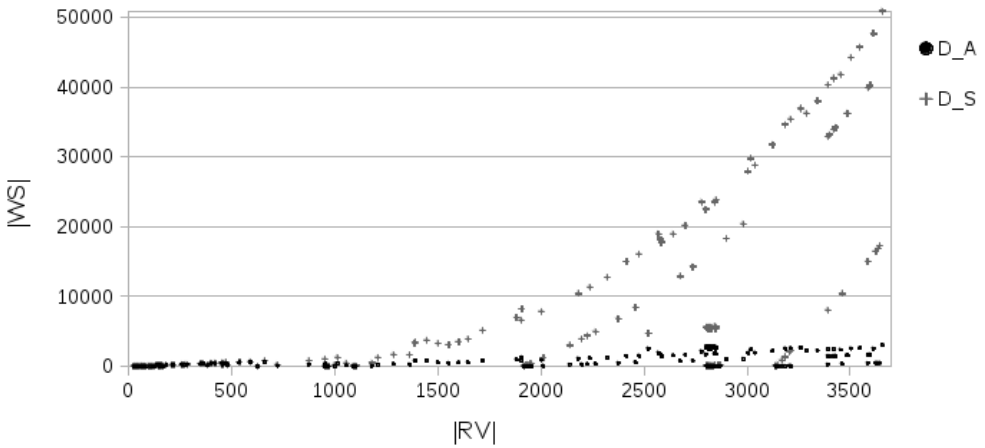
Comparison of running time of static and dynamic associative algorithms on R-MAT-graphs

ГґГ°Г Гґ	n	DistSPT	InsertArcSPT*	k*	InsertArcSPT	k
R-MAT-11	2048	6,171	0,012	6/8	0,009	1/8
R-MAT-12	4096	9,643	0,017	5/10	0,012	3/10
R-MAT-13	8192	29,475	0,038	7/15	0,020	4/13

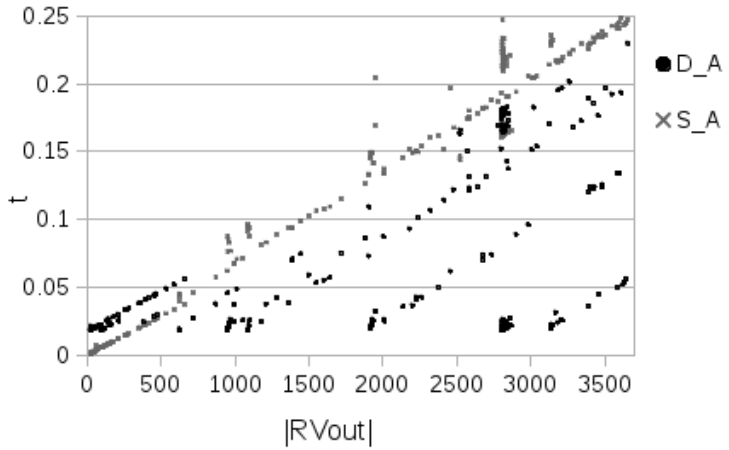
k — number of affected vertices (max_t/max_k) For the sequential algorithm, the number of processed arcs is up to 2000 in the first mode and up to 100 in the second.

Chapter III. Incremental Ramalingam's algorithm. Results of the computational experiment.

D_S - dynamic sequential algorithm
D_A - dynamic associative algorithm
S_A - static association algorithm



Dynamic versions



Associative versions

The test results show that the associative dynamic algorithm runs up to ten times faster than the static associative algorithm, and also performs significantly fewer iterations than the sequential dynamic algorithm.

Chapter III. Star-algorithms for solving problems on graphs. Conclusions.

- Associative dynamic algorithms have been developed and implemented for which algorithms no other parallel implementations could be found.
  - They provide speedup **by several orders** compared to static associative algorithms;
  - moreover, the number of affected vertices in the associative algorithm differs from the number of affected arcs in the sequential algorithm by d times, where d is the average degree of vertices in the graph.
- In the case of the Warshall algorithm, the theoretical complexity was reduced from $O(N^3)$ to $O(N)$. In practice, the algorithm gave almost linear running time, while other implementations were performed in quadratic time.

Prospects for further development: application in bioinformatics.

- search on a large data array: from tens of thousands to several billion nucleotide bases;
- limited alphabet: A,T,C,G for nucleotides (3 bits) and 20 for amino acids (5 bits);
- control slices enable ternary searches.

Problems for which algorithms from the library of basic algorithms of the Star language are suitable:

- global alignment - alignment of the entire sequence, relative to another;
- the search for coincidence motifs is the search for a match of a short sequence in one or more segments of a long sequence;
- multiple alignment - mutual alignment of many sequences;
- construction of a point matrix of similarity.

Thank you for your attention!