Discrete Mathematics Iinstruction Set Computer (DISC) Principles of Operation

A. Yu. Popov
Department of Computer Systems and Networks
Bauman Moscow State Technical University
Moscow, Russia
alexpopov@bmstu.ru

***Abstract*— Acceleration of large data volumes processing is a significant challenge for computer technology to meet, especially with recent growth in demand. Discrete mathematics should be supported by both hardware and software to break the performance limits of universal computers. However, most of the applicable optimization methods on graphs and big data sets are based on software technology, while hardware support promises to give better results.**

We introduce a principally new computer architecture with multiple instruction streams and a single data stream (MISD) based on the non-Von Neuman microprocessor with a DISC command set (discrete mathematics instruction set). The Leonhard microprocessor was developed and tested at Bauman Moscow State Technical University to support large data sets, data structures and graph processing. We describe the principles of the Leonhard microprocessor's operation and some areas of its implementation: from robot movement control and software defined networks to machine learning and computer vision systems.

Keywords*— *discrete mathematics; graphs; instruction set; data structure

I. Introduction

The Internet of Things, social media, and mobile devices generate Exabytes of unstructured data, which are extremely difficult to store and analyze with commonly used computing technology [1]. Such initiatives as heterogeneous architecture, OpenPOWER [2], or HSA Foundation [3] are promising to create high performance systems with different types of accelerators. However, the set of acceleration cores in reality is almost limited by cryptography processing and matrix/vector operations.

In this work we focusing on the important issues of discrete optimization, which is widely used in most computational challenges in applied science (e.g., bioinformatics, chemistry, statistics, physics, economics) and in industries (telecommunication, manufacturing, retail, banking and others). Therefore, it is important to note the disadvantage of universal computing where discrete optimization can only be processed through multiple calls of primitive arithmetic operations. There are still no discrete mathematics acceleration units inside computer systems.

This paper presents basic principles of operation and target implementations of a principally new computing system with multiple instruction streams and a single data stream architecture (MISD in accordance with Flynn's Taxonomy), developed at Bauman Moscow State Technical University. We also introduce the Leonhard microprocessor (named in honor of Leonhard Euler) with discrete mathematics instructions set (Discrete mathematics instruction set computing, DISC).

II. Background

Data sets as a fundamental concept of discrete mathematics are usually implemented in the form of data structures in the computer's memory. There exist many types of data structures such as arrays and trees developed to accelerate computations [4, 5], but we should note that generic CPU pipelines and slow memory subsystem cause latency problems for most types of data structures.

As noted in [4–7] there is a significant time gap between sequential and random access to memory devices. The random access for lists and trees raises much more cache misses and page faults inside the processor pipeline than the sequential one for vectors or arrays. On the other hand, lists and trees use pointers to traverse over the structure from one element to another. This means that the next address will become known to the Memory Management Unit only after passing the current pointer through the CPU's entire pipeline [6].

As a result of our research, we initially introduced the MISD computer architecture in [7]. As noted, this system consists of two different microprocessors: the Central processing unit for generic workload (CPU) and the Structure processing unit for discrete operations (Leonhard SPU). For this reason, the SPU instruction set corresponds to the discrete mathematics operations and quantifiers. The performance level shown in [7] allows for implementation of the MISD system in many important areas. This work also aims to recognize the most promising projects.

III. Computer system for discrete mathematics

Formally, the data structure combines two types of information: information about the data stored (i) and information about the structuring relations themself (ii) [5].

The duality of data structures allows to share the workload between two threads: the first computes the relation part, and

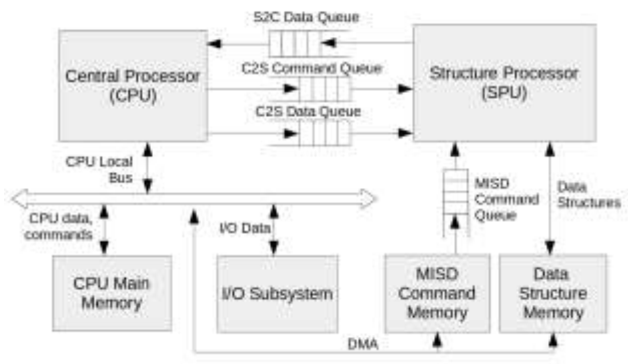

Fig. 1. Multiple instructions and single data stream computer architecture

the second – the information part. In Fig.1 we show the basic concept of the new architecture. The SPU is a special unit that processes the relational part of data structures, while the CPU performs computations on the informational part [7].

The Leonhard SPU independently accesses the local memory in order to store data structures and execute instructions. The SPU's operation results are forwarded to the CPU for further use in the computational algorithm.

There are two possible ways to call instructions in the Leonhard SPU: when the CPU sends the command through the C2S queue (i), or when Leonhard starts it from the local command memory (ii). When using the first method, this system becomes similar to a heterogeneous system with an accelerator. Using the second method, this system can be classified as a parallel computer with multiple instructions and a single data stream architecture [7].

The Leonhard SPU uses two synchronization modes with the CPU thread. The first one uses semaphore fields in every operation to put the instruction fetch unit into the wait state. For that purpose, the instruction format includes a special tag bit for every operand (up to three operands allowed in one instruction).

In the second synchronization mode, the CPU sends instructions only when it should be started (same manner as with the math coprocessor in Intel 386). Therefore, the Leonhard SPU can operate in MISD mode (first one), in accelerator mode (second one), and in combined mode (both types of synchronization).

IV. Leonhard microarchitecture

The Leonhard SPU stores information as key-value pairs in the form of non-overlapping B+ trees. To provide the tree leaves execution the SPU has a pipeline parallel microarchitecture, which is shown in Fig. 2. The SPU not only performs the search in sub-trees and provides insertion and deletion operations through the tree structure, but also allocates and frees memory for nodes and leaves.

We divide the tree's processing into "tree tracing" and "leaf operation" stages. This allows us to speed up the computing process over keys and to store the search path in the internal cache (named Catalogue or CAT). The CAT processes the tree from the root down to the leaves. The sequence of tree nodes on the route from the tree root to its leaf is called "trace" and is always ready for immediate access by CAT.

After the tree tracing has finished, the second stage is started by the Operational Buffer (OB) to operate keys and values on the bottom leaf level. To perform this for large data structures, the Leonhard's memory subsystem includes multilevel storage devices described below.

The first-level is the register memory that stores nodes and leaves inside the CAT and OB. It is organized as an associative memory to perform key searching, shifting, union, and other low level operations. The second-level is the boundary addressable memory (Internal Data Structure Memory, IDSM), accessed by the CAT to define the physical addresses for the next level memory. The third memory level is the external RAM named Data Structures Memory (DSM).

The CAT includes the matrix of single Catalogue Processor Elements (CPE) to perform multiple node processing. To make processing in parallel possible, the Control Unit (CU) sends the same command to all CPEs in order to perform one operation for different nodes. Therefore, CAT performs SIMD processing and every CPE is defined as a SISD unit (according to Flynn's taxonomy).

After the full path becomes known, the leaf is loading to the Operational Buffer from the IDSM. To support such operations as union and intersect, the OB has to consists of three subunits: Buffers A and B to store data for both source structures, and Buffer R to process and store the resulting data structure R (see Fig. 3). All OB's buffers include Operational Buffer Processor Elements (OBPE) similar to the SIMD architecture to process leaf keys and values in parallel.

After the leaf is processed in the OB, the R Buffer uploads results back to the IDSM. If any further processing is needed, then the CAT loads the new trace and OB processes the new leaf. The OB puts results into the S2C queue when the operation is finished.

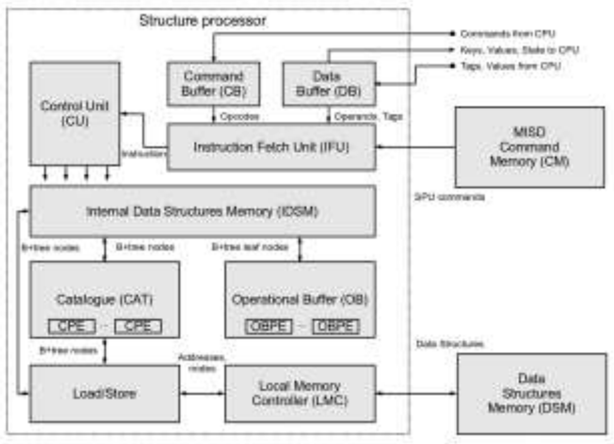

Fig. 2. Leonhard SPU microarchitecture.

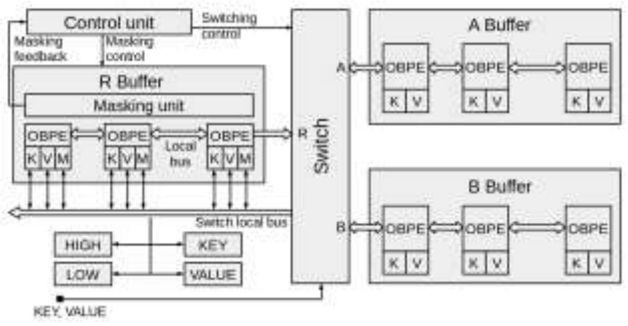

Fig. 3. Operational buffer.

V. Discrete mathematics instruction set

The Leonard instruction set was extended with two new commands (NSM and NGR) to satisfy algorithm requirements. Every instruction includes up to three operands: key, value and data structure number. As of now, the instruction set consists of 20 high-level operation codes, which are listed below.

Search (SRCH): starts the exact search for the value related with the key.

Insert (INS): key-value pair inserting into the structure. SPU renews volume if the key is already stored.

Delete (DEL) operation performs the exact search for the specified key and removes it from the data structure.

Smaller and Greater Neighbors (NSM, NGR) instructions searches for the smallest (greatest) key that is a neighbor to the key given, and returns its value. The neighbor operation can be applied for heuristic computing where interpolation is used instead of exact calculations (e.g. clustering or aggregation).

Maximum and Minimum (MAX, MIN) instructions search for the first or last keys in the data structure.

Cardinality (CNT) operation defines the count of keys stored in the structure.

AND, OR, NOT instructions perform union, intersection, and complement operations on two data structures.

Slices (LS, GR, LSEQ, GREQ) extract a subset of one data structure into another.

Search next and previous (NEXT, PREV) instructions find keys in the data structure that are neighbors (next or previous) to the key stored. This instruction differs from the NSM/NGR instructions. It is mandatory to store the key in the data structure before calling this instruction.

Delete all structure (DELS) clears all resources used by the given structure.

Squeeze (SQ) instruction compresses the memory blocks used by the data structure (used timely to lower DSM memory fragmentation).

Jump (JT) instruction branches the SPU code in order to give the CPU control and is available only in the MISD mode.

Our research [7] has shown a strong CPU vs. SPU data dependency for most of the existing optimization algorithms. We also introduce a way to lessen the dependency by reusing previous results or operands, stored in the SPU's internal registers. In most cases, these results can be reused in new compound keys (i.e., similar to compound or composite keys in databases). This approach significantly reduces dependencies from 80–90% to 30–50% for all of the SPU's instructions.

VI. Experiments results

Most of the Leonhard SPU's commands require O(log n) memory accesses (excluding AND/OR/NOT and slices operations), but they executing much faster than generic CPU programs due to the SPU's efficient hardware and the trace caching technology. The Leonhard microprocessor was implemented on the Virtex FPGA platform with on-chip PowerPC405 as a CPU. Experiments were carried out to measure the productivity of the implemented MISD principles. We used the first Leonhard version with the following parameters: 32 bits for keys and values; 2*106 maximum keys in the structure; 7 maximum numbers of data structures controlled by the SPU; 256 MB DSM capacity; 100 MHz SPU and CPU frequencies.

Because of the frequency difference between FPGA and VLSI technologies, we measured clock counts instead of execution time. This allows us to understand all the improvements of these new principles without taking the technological impact into account.

The peak power consumption for the MISD system consisting of PowerPC and Leonhard microprocessors was approximately 1.1W, i.e. 31 times lower than the 35W of a Core i5. The Leonhard complexity is 1.1M gates and 2.5M gates for the PowerPC405 microprocessor, which means that it requires around 420 times fewer gates than 4x Core i5 CPUs.

VII. Applications

We have considered several Leonhard microprocessor implementations to improve performance and power consumption. The first one aimed to create a hardware-based software-defined networks (SDN) controller. SDN control unit is software, which collects transaction information by the Open Flow protocol and controls routing resources to improve total network productivity. Most of SDN management algorithms are based on graph models and discrete optimization. At the same time, the requirements for deep analytical processing, for response time, and for throughput of network transactions are constantly growing. Machine learning tools are already being used to predict changes in traffic and manage the network in a predictive manner. Traffic analysis for suspicious application activity, network attacks, and other information security threats are also required. In this regard, the hardware support for SDN controllers based on the Leonhard microprocessor is relevant.

Another important area is hardware support for robot movement. Moving along a route, the robot builds a visibility graph to store mutually visible points as edges. In this case, the edge weight could mean the energy cost required to move the robot from one position to another. In this regard, it is important to solve the shortest path problem on visibility graphs to move the robot in a fast and energy efficient manner.

The Leonhard microprocessor has also been implemented for the k-nearest algorithm which is usually associated with machine learning. The DISK system showed the high performance levels and quality results for the Space Shuttle telemetry data classification problem (99.7% accuracy achieved). This allows us to use the Leonhard microprocessor as an intellectual decision subsystems inside dynamic control systems. We are also working on projects for unmanned aerial vehicle management and hardware-based autopilot on the DISC platform.

Last but not least, the Leonhard microprocessor can be used in computer vision systems. DISC can be applied for image segmentation via graph algorithms (e.g. mincut/maxflow, labyrinth algorithms), as well as for defining and storing scene models and, as a result, controlling robots intellectually.

TABLE I. ALGORITHMS AND OPERATIONS ACCELERATION FOR THE MISD ARCHITECTURE (MEASURED IN CLOCK CYCLES)

Experiment (Generic microprocessor)	Acceleration
Delete (Microblaze)	164.4
Insert (Microblaze)	42.7
Search (Microblaze)	31.4
Delete (Intel Pentium 4)	22.8
Dijkstra's Algorithm (Intel Pentium 4)	19.4
Search (Intel Pentium 4)	15.3
Depth-First Search (ARM11)	12.9
Breadth-First Search (ARM11)	12.3
Delete (Intel Core i5)	11.8
Prim's Algorithm (ARM11)	10.3
Search (Intel Core i5)	9.8
Dijkstra's Algorithm (Intel Core i5)	7.6
Kruskal's Algorithm (ARM11)	7.8
Insert (Pentium 4)	5.7
Insert (Intel Core i5)	3.2
Depth-First Search (Intel Core i5)	3.2
Breadth-First Search (Intel Core i5)	3.0

REFERENCES

[1] Malik P. Governing Big Data: Principles and Practices, IBM Journal of Research and Development. 2013. vol. 57. no. 3a, 4, pp. 1:1-1:13.

[2] Stuecheli J., Blaner B., Johns C.R., Siegel M.S. CAPI: A Coherent Accelerator Processor Interface, IBM Journal of Research and Development. 2015, vol. 59, no. 1, pp. 7:1-7:7.

[3] Glossner J., Blinzer P., Takala J. HSA-enabled DSPs and accelerators, 2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP). Orlando. FL. 2015, pp.1407-1411.

[4] Knuth D. The Art of Computer Programming. Volume 1. Fundamental Algorithms, 3-rd ed. Addison-Wesley, 1997, 672 p.

[5] Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd ed. MIT Press, 2009, 1312 p.

[6] Tanenbaum A.S., Austin T. Structured computer organization. 6-th ed. Prentice Hall, 2013, 801 p.

[7] Popov A. An introduction to the MISD technology, Proc. 50-th Hawaii Int. Conf. on System Sciences (HICSS50). 2017, pp. 1003–1012.