VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science

Vytěžování znalostí z rozsáhlých astronomických datasetů masivně paralelním přístupem

Knowledge Extraction from Huge Astronomical Data Sets using Massively Parallel Processing

2015 Bc. Martin Cendelín

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Diploma Thesis Assignment

Student:	**Bc. Martin Cendelín**
Study Programme:	N2647 Information and Communication Technology
Study Branch:	2612T025 Computer Science and Technology
Title:	Vytěžování znalostí z rozsáhlých astronomických datasetů masivně paralelním přístupem Knowledge Extraction from Huge Astronomical Data Sets using Massively Parallel Processing

Description:

Contemporary astronomy has been producing exponentially growing amount of data that cannot be anymore analysed with standard computer techniques due to the limitations of Moore's Law. At the same time the new physical knowledge is hidden and diluted in petabyte scaled sets of raw data and products of automatic processing pipelines. To get useful yield and new scientific knowledge from this Data Avalanche, the new astronomical discipline - the Astroinformatics - has been emerging in recent years. It is based on exploitation of advanced mathematics, modern data mining technology and techniques of artificial intelligence and machine learning.

Unfortunately, the current methods of machine learning, like Neural Networks, Decision Trees and Support Vector Machines are not optimally scaled for processing of huge data sets on personal computers and thus the new hope is seen in massively parallel processors.

Goals of the thesis:
1. Make a survey of available implementations of current algorithms of machine learning using parallel approach. Focus mainly on Multi Layer Perceptrons, Random Decision Forests, Support Vector Machines and Self-Organising Maps (SOM), analytical programming.
2. Analyse their scalability with size of the data sets and identify bottlenecks and limitations
3. Design and implement new algorithms of one or two selected models of machine learning for massively parallel processors according to the results of analysis and instruction of consultant.
4. Optimise them for time and memory usage of decided computing environment with respect to the specific nature of Hadoop framework.
5. Analyse the performance and acquired results precision of your code both on the selected subset of standard machine learning tasks from UCI repository and real astronomical problems like star/galaxy separation, classification of celestial objects, photometric redshifts and other regression tasks etc. according to the consultant's instructions.

References:

[1] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning: data mining, inference, and prediction. Springer series in statistics. Springer, 2008
[2] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification (2nd Edition). Wiley-Interscience, 2000.
[3] S. Cavuoti, M. Brescia, G. Longo. Data mining and Knowledge Discovery Resources for Astronomy in the Web 2.0 Age. Conference 8451 Software and Cyberinfrastructure for Astronomy II, july 2012

[4] O. Maimon and L. Rokach. Data Mining and Knowledge Discovery Handbook. Springer, 2010.
[5] M. Bramer. Principles of Data Mining (Undergraduate Topics in Computer Science). Springer-Verlag New York, Inc., 1st edition edition, 2007.
[6] Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series) by David B. Kirk and Wen-mei W. Hwu (Feb 5, 2010)
[7] Web site of DAME project: dame.dsf.unina.it

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. RNDr. Petr Šaloun, Ph.D.**

Consultant: RNDr. Petr Škoda, CSc.

Date of issue: 01.09.2013

Date of submission: 07.05.2015

[illegible]

doc. Dr. Ing. Eduard Sojka
Head of Department

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA OSTRAVA
Fakulta elektrotechniky a informatiky
1

[illegible]

prof. RNDr. Václav Snášel, CSc.
Dean of Faculty

I declare that I created this thesis independently. I noted all the literature and publications what I used as a source.

Cendelín

July 30, 2015

Bc. Martin Cendelín

I want to thank to my supervisor dr. Petr Šaloun for professional leadership, personal attitude, advices and valuable time.

Abstrakt:

Cílem diplomové práce je průzkum oblasti zpracovávání velkých dat paralelními přístupy a následná implementace jednoho algoritmu a jeho paralelizace na zvolené architektuře.

Mým úkolem je vytvoření aplikace, která bude schopna syntetizovat světelné spektrum vzorové BE hvězdy, pomocí toho výrazu vypočítat potřebné hodnoty pro porovnání se spektry neznámých astronomických objektů.

K účelům syntézy používám analytické programování, jednu z metod symbolické regrese, využívající evolučních algoritmů. Srovnávání spekter je realizováno na multiprocesorové architektuře.

Klíčová slova: paralelní počítání, multiprocesorová architektura, evoluční algoritmy, analytické programování, diferenciální evoluce, SOMA

Abstract:

The aim of thesis is survey of the field of processing large data via parallel accesses and subsequent implementation of one algorithm and its parallelization on the chosen architecture.

My task is to create an application that will be able to synthesize the model light spectrum of Be star, using this expression to calculate the necessary values for comparison with spectra of unknown astronomical objects.

For the purposes of synthesis I use analytic programming, a method of symbolic regression, using evolutionary algorithms. Comparison of spectra is implemented on a multicore processor architecture.

Key-words:

Parallel computing, multiprocessor architecture, evolutionary algorithms, analytic programming, differential evolution, SOMA

List of abbreviations and symbols

API	- Application program interface
CPU	- Central processing unit
CU	- Control Unit
GPU	- Graphical processing unit
HDFS	- Hadoop distributed file system
PE	- Processing Element
SOMA	- Self organizing migrating algorithm

Content

1 Introduction

Although this thesis is mainly focused on computer science, this chapter presents basic background of astronomical problem, which necessary to mention.

In the last several decades, rising information technologies allowed us to collect all sorts of information from many sectors of human activity. One of purposes of this collected information is to find a pattern in them and utilize this pattern. With the technological improvement, the amount of collected data is rapidly rising and as the amount of data increases, human's capability of understand this data decrease. The only way to keep pace with this huge increasing of data is to use computers for data mining.

Data mining is defined as the process of discovering patterns in data. The process must be automatic or (more usually) semiautomatic. The patterns discovered must be meaningful in that they lead to some advantage. The data is invariably present in substantial quantities. [8]

1.1 Machine learning

There are some who regard data mining as synonymous with machine learning. There is no question that some data mining appropriately uses algorithms from machine learning. Machine-learning practitioners use the data as a training set, to train an algorithm of one of the many types used by machine-learning practitioners, such as Bayes nets, support-vector machines, decision trees, hidden Markov models, and many others. There are situations where using data in this way makes sense. The typical case where machine learning is a good approach is when we have little idea of what we are looking for in the data. On the other hand, machine learning has not proved successful in situations where we can describe the goals of the mining more directly. [9]

1.2 Astronomical problem

As it has been said before, information from all sorts of sources are collecting right now. Astronomy is not the exception. Information technologies are integral parts of today astronomy work. There are many organizations, which collect data from space through gigapixels CCD detectors, radio telescopes, astronomical satellites, etc. 1 TB of data over one night from one device is nothing special today. All of this has resulted in a data avalanche, which we can't process by regular procedures.

In this thesis, I will solve problem of founding BE stars in huge amount of visible spectra.

1.2.1 Astronomical terminology

1.2.1.1 Electromagnetic waves

Electromagnetic waves are the source for almost all the knowledge that science has about the objects in space. This is primarily due to their ability to travel in vacuum, and the fact that most of space is vacuum. Most celestial objects emit electromagnetic waves, some of these radiations that reach earth are studied by scientists to gather information about the universe. The sky survey telescopes focus on portions of the sky to gather the electromagnetic waves from the region, this is then filtered to get specific details of the electromagnetic spectrum and is stored as electronic/digital data. [10]

1.2.1.2 **Flux, Magnitude and Filter**

The spectrum of an object corresponds to the intensity of the light, for each wavelength, emitted by the object. [10]

A study of the spectrum gives us information about the object. Flux is a measure of the intensity of light, it is the amount of light that falls on a specified area of the earth in a specified time. [10]

In astronomy the term magnitude is used with reference to the intensity of light from different objects, it is computed from the flux. Magnitude is a number that measures the brightness of a star or galaxy. In magnitude, higher numbers correspond to fainter objects, lower numbers to brighter objects; the very brightest objects have negative magnitudes. Also, the terms flux and magnitude are used with respect to each light wave, of a particular wavelength, belonging to the electromagnetic spectrum. [10]

Magnitude is relative and it is calculate from base object. Standardly, star Vega in the constellation Lyra is the base object.

It is expensive and quite unnecessary to gather and store information about every wavelength of the spectra for each object. Hence, astronomers prefer to select a few interesting parts of the spectra for their observation. This is done by the use of filters. A filter is a device that blocks all unwanted light waves and lets only the specified wavelength pass through for observation.

1.2.1.3 **Color and spectrum**

In the real world color is a subjective judgment, i.e what one person calls "blue" may be a different shade than another person's "blue." But, colors are an important feature of objects. Its importance holds true in astronomy as well. However, physicists and astronomers need a definition for color that everyone can agree on. In astronomy, color is the difference in magnitude between two filters. The sky surveys choose the filters to view a wide range of colors, while focusing on the colors of interesting celestial objects. Color is symbolized by subtracting the magnitudes. Since all these quantities involve magnitude, they decrease with increasing light output. i.e If g and r are two magnitudes corresponding to green and red lights of specific wavelengths, g-r is a color, and a large value for g-r means that the object is more red than it is green. The reason for this can be seen from the fact that magnitudes are computed as logarithms. [10]

A spectrum (plural is spectra) is a graph of intensity (amount) of light emitted by an object as a function of the wavelength of the light. Astronomers frequently measure spectra of stars, and use these measurements to study stars. Each object has a unique spectrum, like a fingerprint is to a person. An object's spectrum gives us information about the wavelengths of light the object emits, its color (from peak intensity), temperature (using Stefan's law[1]), distance from the earth (using Hubble's law[2]), and a hint towards its chemical composition (from the absorption and emission lines). Thus, spectrum analysis is one of the best sources of knowledge of a celestial object. [10]

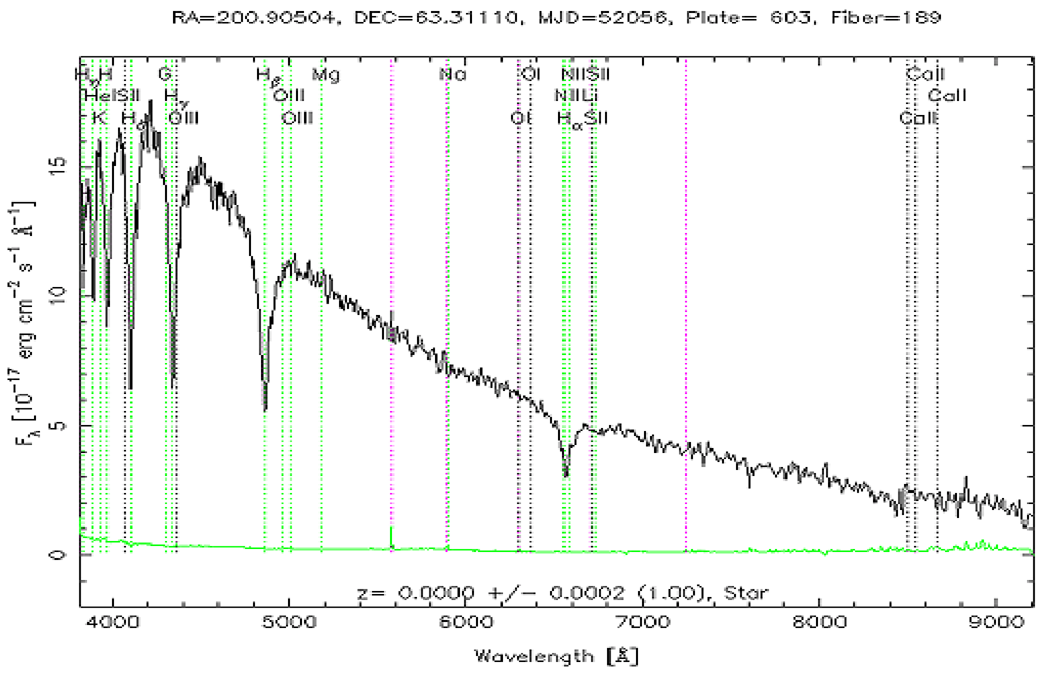


Fig. 1: Spectrum of a typical star, passed from [10]

[1] http://hyperphysics.phy-astr.gsu.edu/hbase/thermo/stefan.html
[2] http://www.astro.cornell.edu/academics/courses/astro201/hubbles_law.htm

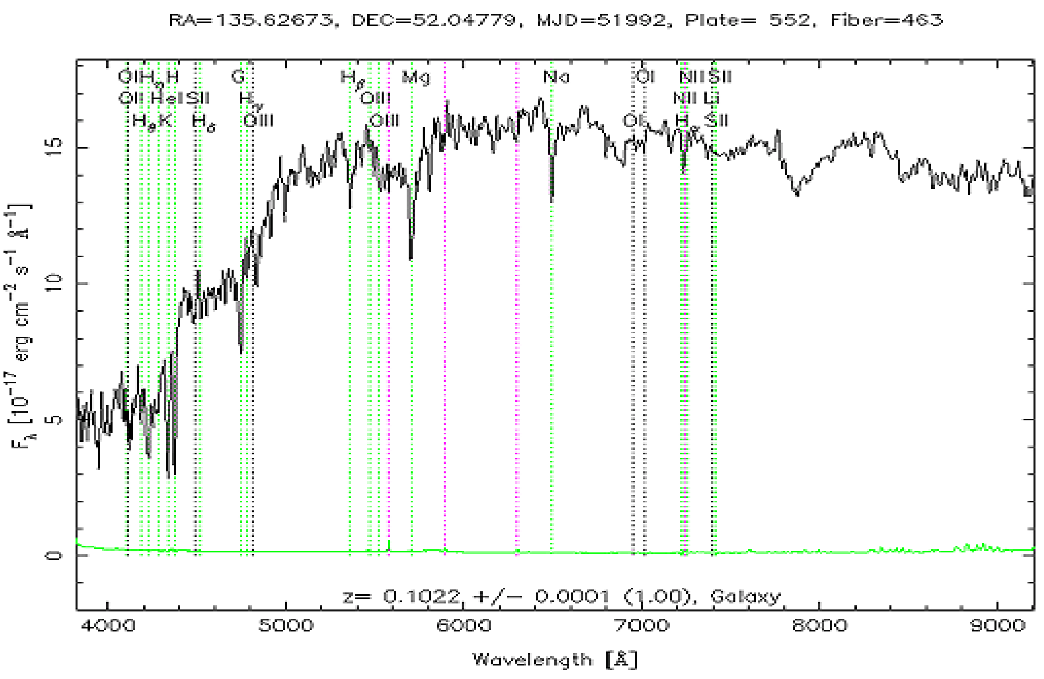


Fig. 2: Spectrum of a typical galaxy, passed from [10]

1.3 Stars

Stars are huge sphere of plasma consist mostly from hydrogen and helium. They begin their lives as big clouds of hydrogen. Owing to gravity, matter is pulled together and from cloud became sphere. Strengthening gravity force warms the core, until thermonuclear fusion begins. At this point, start produce huge amount of energy, using hydrogen as fuel, which it transforms into helium. Star remain in this state for most of its life, but after some time hydrogen will be depleted. Bigger stars burns their fuel faster and smaller slower. When hydrogen is depended, thermonuclear fusion begin to use helium as fuel, which it transforms to heavier elements. This action is repeating on heavier and heavier elements, until only iron remain. Star can't use iron as fuel and so end of star comes. End of star depends on size and neighborhood of the star, some become white dwarfs, and some blow as supernovas.

The nearest and most famous star is Sun, which is medium size star located outer part of Milky Way. However, in night we can see many other stars.

From the darkest parts of Earth, the naked human eye can see about 5,000 stars; from a brightly lit city street, only about 100. [11]

According with ESA, there are something like $10^{11} – 10^{12}$ stars in our galaxy, and there are something like $10^{11} – 10^{12}$ galaxies and this is huge amount of things to explore and study. There are several types of stars, depending on their specters.

1.3.1 **Classification of stars**

In the late 1800s an astronomer at the Harvard College Observatory began to record stellar spectra, using a method similar to the glass prism described above. The first star looked at was noticed to have "gaps" or "breaks" in the spectrum at specific points, called absorption lines. This was not understood as well at the time as it is now, but nevertheless, the star was categorized as an "A". If the next star spectrum had similar absorption lines in the spectrum to the first, then it too would be classified as an "A". However, if the absorption lines were not similar to the first star, it was classified as a "B". This went on until all of the stars recorded were categorized with a letter between A and Q. In the beginning of the 1900s, a different astronomer used these same categorizing letters, but dropped all of them except for O, B, A, F, G, K, and M in that order. It was later realized that this classification was based on temperature, with O being the hottest and M being the coolest. Furthermore, the numbers 0-9 were added at the end of the letter to further divide the stars. For example, a B0 represents the hottest star in the B series, while a B9 represents the coolest star in the series. Our sun is classified as a G2 star. [12]

The reason stars are different colors is directly related to them being different temperatures. The hotter stars look blue, the cooler stars look red, and stars like our sun (cooler than blue stars and hotter than red stars) look yellow. Of course there are different shades of the colors, meaning stars could be orange, bluish-white, or other shades of the above colors. [12]

Spectral Type	Temperature (Kelvin)
O	28000 - 50000
B	10000 - 28000
A	7500 - 10000
F	6000 - 7500
G	5000 - 6000
K	3500 - 5000
M	2500 - 3500

Tab. 1: Specter types and temperatures, passed from [12]

1.3.2 **Be stars**

A Classical Be star (CBe) is a rapidly rotating B-type star with an equatorial circumstellar decretion disk, which is not related to the natal disk the star had during its accretion phase. Jaschek and Jaschek (1983) indented 12% of the stars in the Bright Star Catalogue as Be stars. [13]

Although it has been more than hundred years since Father Angelo Secchi discovered first Be star (γ Cassiopeia), the Be phenomenon still eludes explanation. Be stars are normal B-type stars with an optically and geometrically thin circumstellar disk, which is inferred from emission lines, IR excess and intrinsic polarization. The Be phenomenon is the episodic occurrence of mass loss in these stars, resulting in Balmer emission[3]. While the Be phenomenon can be observed in some late O and early A stars, it is mainly confined to stars of B spectral type. The production of disk in CBe stars is still a mystery and majority of the studies point towards an optically thin equatorial disk formed by channeling of matter from the star through wind, rotation and magnetic field. [13]

Rapid rotation, stellar wind, non-radial pulsation, magnetic field and binary interaction are the proposed mechanisms to explain Be phenomenon in Be stars. [13]

[3] http://astronomy.swin.edu.au/cosmos/B/Balmer+series

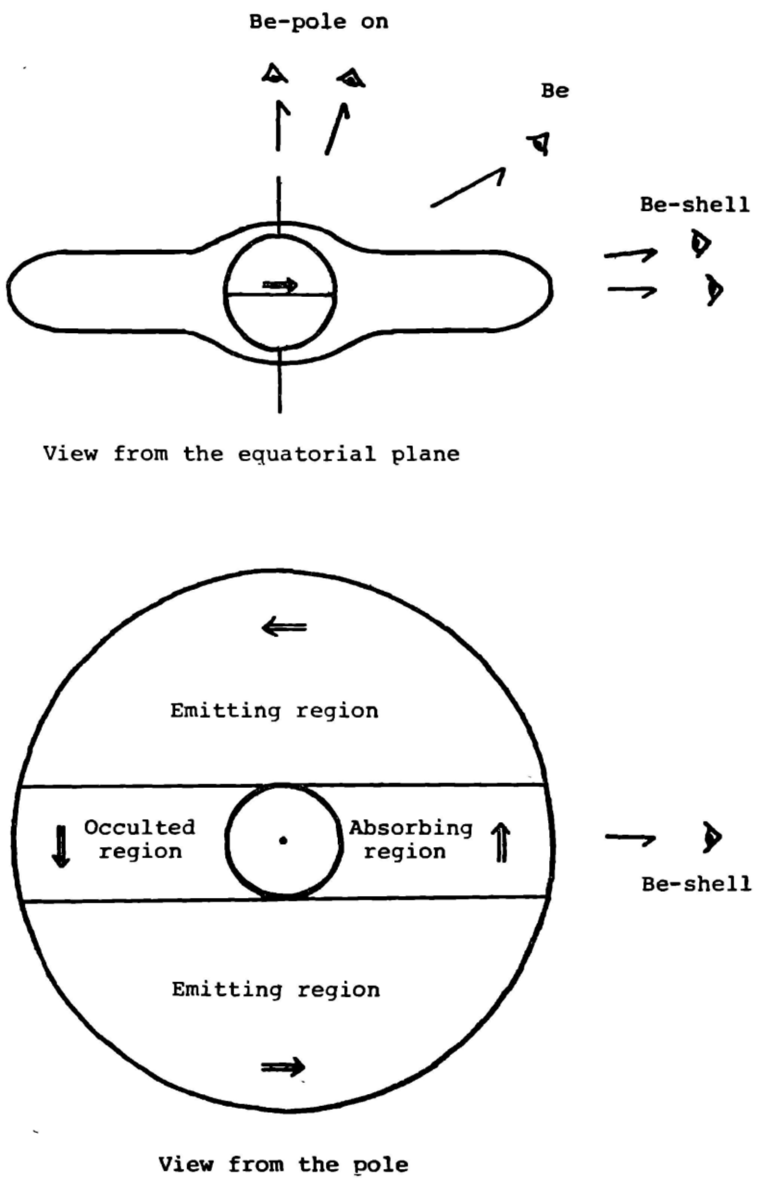


Fig. 3: Schematic structure of the envelope of a Be star, passed from [13]

Be stars rotate faster than normal B stars, although they do not rotate at break-up velocity (Slettebak, 1979). The effect of rotation in the evolution of massive stars is not fully understood. The outer layers of rotating massive stars may spin up due to the evolution of the angular momentum distribution (Langer and Heger, 1998; Heger and Langer, 2000). When the star rotates at critical speed, it reaches ω limit, and the effective gravity becomes zero (Maeder, 1999). It has been found that Be stars rotate only at 70% of critical velocity (Vcrit) (Chen and Huang, 1987; Porter, 1996). By taking the effects of equatorial gravity darkening, Townsend et al. (2004) argued that Be stars may be rotating close to critical velocity. Gravity darkening is the phenomenon in which fast rotation produces equatorial stretching of stars, which in turn induces non-uniform surface gravity and temperature distributions (von Zeipel, 1924). Using Monte Carlo modeling Cranmer (2005) found that late type Be stars (B3 and later) can rotate close to Vcrit while early type rotate at 40% - 60% of the critical value. [13]

A stellar wind is the continuous, supersonic outflow of matter from a star. In massive stars, the winds (driven by radiation pressure) influence the star's evolution as well as the interstellar medium. The circumstellar material in Be stars is concentrated in a disk with a high density and relatively low temperature. In the polar regions of the star the mass flux is lower which produces a low density wind with large velocities (typically 1000 to 2000 km/s). The super-ionization occurs in this low density wind (Poeckert, 1982). This model does not explain why the Be phenomenon can vanish and re-appear on timescales of the order of years. An alternative model was proposed by Doazan and Thomas (1982). They assume that the Be star is surrounded by a spherically symmetric corona, in which the material can be accelerated to large velocities of the order of 1000 km/s. Outside this corona the wind is decelerated by interaction with the circumstellar material. This produces a very extended region of moderately high density, which produces the H_α emission. In this model the long time scale variations of the emission line phenomena are explained by the presence or absence of the material which is collected in the decelerating region. The wind compressed disk (WCD) model of Bjorkman and Cassinelli (1993) proposes the formation of dense equatorial disk by radiation-driven wind, in which the ram pressure of the polar wind compresses and confines the disk. However it has been estimated from numerical hydrodynamic simulation that the disk generated by WCD model is not dense enough to produce enough IR/radio continuum emission (Owocki et al., 1994). [13]

CBe stars falls into 3 categories based on the viewing angle. They are normal Be stars (non-supergiants), B-type shell stars and Pole-on stars (Kogure and Hirata, 1982; Jaschek et al., 1981). Normal Be stars (non-supergiants) show double-peaked emission lines whose central reversals are not deeper than the continuum. B-type shell stars are characterized by shell lines in Balmer lines and metallic lines from ground states or metastable levels. The properties of shell lines are that they are usually sharp and their central depths are usually deeper than the continuum. Pole-on stars are characterized by single-peaked emission lines on broader photospheric absorption lines. [13]

2 Parallel computing

According to Moore's law[4], performance of classical sequential hardware is doubled every two years. Unfortunately request for high computation performance in science, engineering, industry and other domains is rising much more in last few years. To fulfil this request we must use an alternative approaches to increasing computation performance.

One of logical approach to this problem is parallelization. In general, parallel computing is computing, where two or more computation resources (processors, workstations …) are working together on common task. Each computation unit works on its peace of problem and it can exchange information with another one.

At the first look idea of parallel computing seems to be ultimate solution to performance problem. Unfortunately it is not that simple and multiplying of computing resources doesn't have to equal multiplying of computing performance.

At the first we need parallel program. Classic sequential programs are unaware of existence of other computing resources and it will run on parallel architecture same as on sequential. To use advantages of parallel architecture, we have to adapt our program to its nature. Not all parts of program can work parallel. Some parts need to work sequentially to do their jobs. Proportion of parallelizable and not parallelizable parts of program determines, how suitable our program is to be parallelizable.

Nature of parallelization has some other complication, but with suitable task, we can significantly increase speed of its execution.

2.1 Parallel vs sequential computing

Traditionally, software has been written for *serial* computation [1]:

- To be run on a single computer having a single Central Processing Unit (CPU).
- A problem is broken into a discrete series of instructions.
- Instructions are executed one after another.
- Only one instruction may execute at any moment in time.
- Limits.

[4] http://spectrum.ieee.org/static/special-report-50-years-of-moores-law

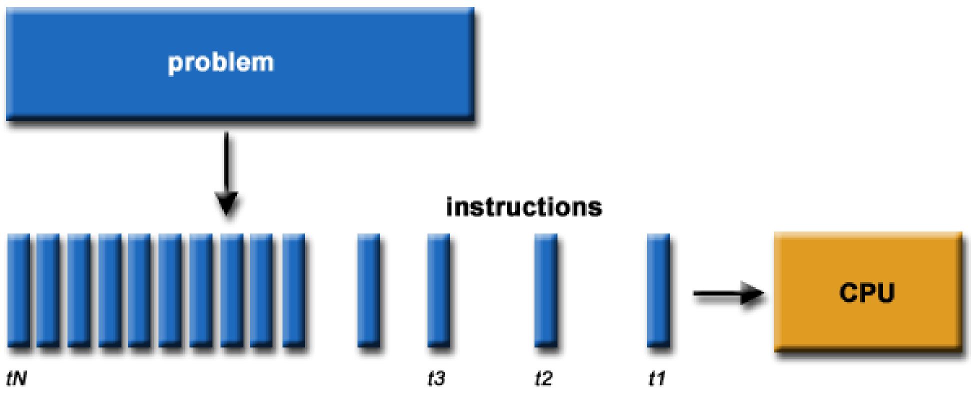


Fig. 4: Idea of sequential computing, passed from [1]

In the simplest sense, *parallel* computing is the simultaneous use of multiple computing resources to solve a computational problem [1]:

- To be run using multiple processors
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed

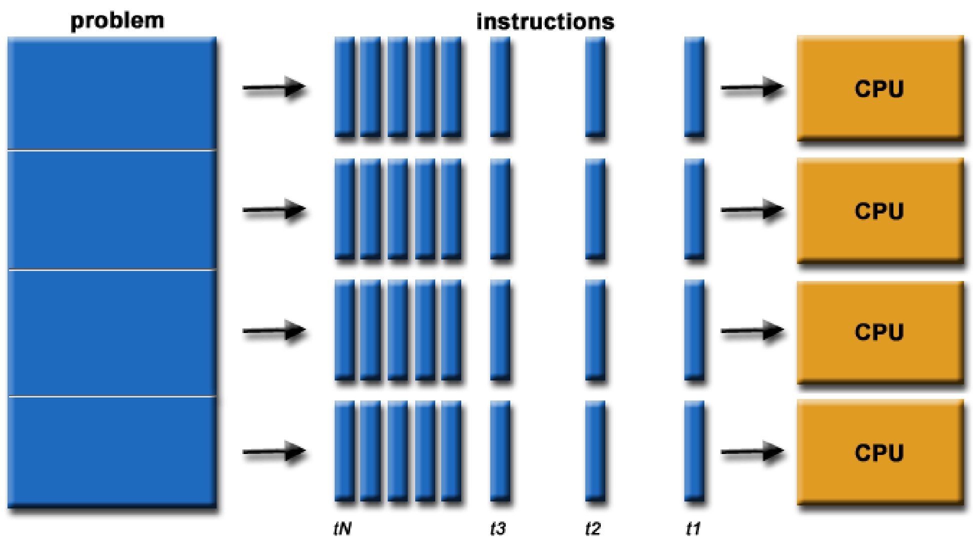


Fig. 5: Idea of parallel computing, passed from [1]

2.2 Reasons to use parallel computing

- **Save time and/or money:** In theory, throwing more resources at a task will shorten it's time to completion, with potential cost savings. Parallel computers can be built from cheap, commodity components. [1]
- **Solve larger problems:** Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory. For example: Web search engines/databases processing millions of transactions per second. [1]
- **Provide concurrency:** A single compute resource can only do one thing at a time. Multiple computing resources can be doing many things simultaneously. For example, the Access Grid provides a global collaboration network where people from around the world can meet and conduct work "virtually".[1]
- **Use of non-local resources:** Using compute resources on a wide area network, or even the Internet when local compute resources are scarce. For example: SETI@home [1]
- **Limits to serial computing:** Both physical and practical reasons pose significant constraints to simply building ever faster serial computers[1]:
    - Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
    - Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
    - Economic limitations - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.
    - Current computer architectures are increasingly relying upon hardware level parallelism to improve performance

2.3 Limits and costs of parallel programming

2.3.1 Amdahl's Law

Each parallel application can be understood as combination of serial and parallel sections. Serial sections limit the parallel effectives. Amdahl's Law[5] express this limitation of speedup as:

[5] http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2

$$speedup = \frac{1}{1-P} = \frac{1}{S + \frac{P}{N}}$$

Where P means parallel fraction ($0 \leq P < 1$), S means serial fraction ($0 < S \leq 1$) and N means number of processors. This gives us image of theoretical limitations for parallel computing.

N	P = 0.5	P = 0.9	P = 0.99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1000	1.99	9.91	90.99
10000	1.99	9.91	99.02
100000	1.99	9.99	99.90

Tab. 2: Example of theoretical speedup, passed from [1]

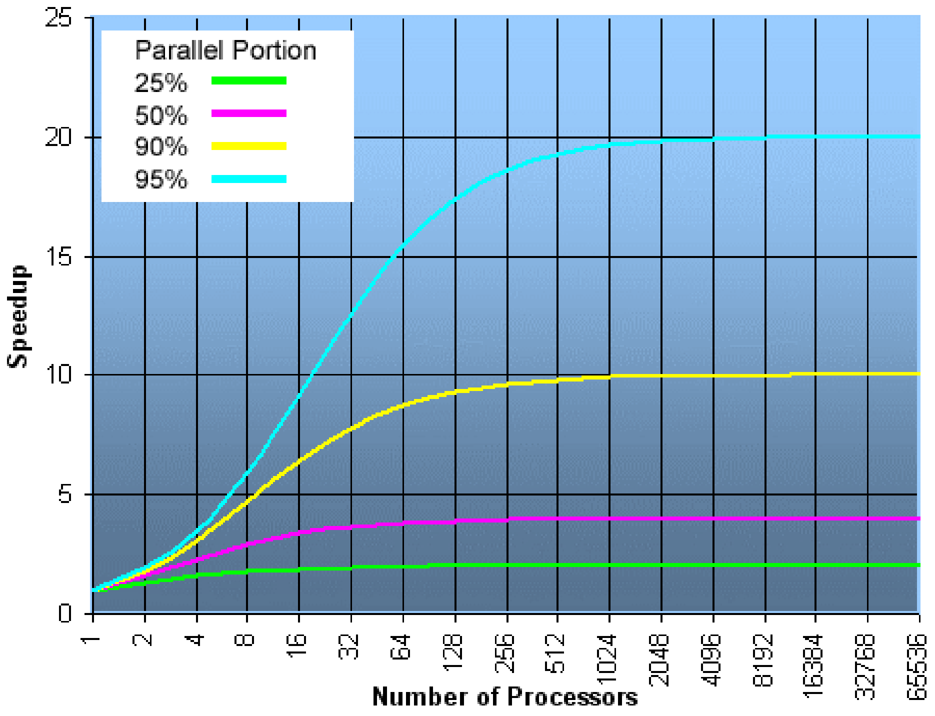


Fig. 6: Graph of theoretical speedup depends on number of processors [1]

2.3.2 **Complexity**

In general, parallel applications are much more complex than corresponding serial applications, perhaps an order of magnitude. Not only do you have multiple instruction streams executing at the same time, but you also have data flowing between them. The costs of complexity are measured in programmer time in virtually every aspect of the software development cycle: [1]

- Design
- Coding
- Debugging
- Tuning
- Maintenance

2.3.3 **Portability**

Thanks to standardization in several APIs, such as MPI, POSIX threads, and OpenMP, portability issues with parallel programs are not as serious as in years past. However all of the usual portability issues associated with serial programs apply to parallel programs. For example, if you use vendor "enhancements" to Fortran, C or C++, portability will be a problem. Even though standards exist for several APIs, implementations will differ in a number of details, sometimes to the point of requiring code modifications in order to effect portability. Operating systems can play a key role in code portability issues. Hardware architectures are characteristically highly variable and can affect portability. [1]

2.3.4 **Resource Requirements**

The primary intent of parallel programming is to decrease execution wall clock time, however in order to accomplish this, more CPU time is required. For example, a parallel code that runs in 1 hour on 8 processors actually uses 8 hours of CPU time. The amount of memory required can be greater for parallel codes than serial codes, due to the need to replicate data and for overheads associated with parallel support libraries and subsystems. For short running parallel programs, there can actually be a decrease in performance compared to a similar serial implementation. The overhead costs associated with setting up the parallel environment, task creation, communications and task termination can comprise a significant portion of the total execution time for short runs. [1]

2.3.5 **Scalability**

The ability of a parallel program's performance to scale is a result of a number of interrelated factors. Simply adding more processors is rarely the answer. The algorithm may have inherent limits to scalability. At some point, adding more resources causes performance to decrease. Most parallel solutions demonstrate this characteristic at some point. Hardware factors play a significant role in scalability. Examples: [1]

- Memory-cpu bus bandwidth on an SMP machine
- Communications network bandwidth
- Amount of memory available on any given machine or set of machines
- Processor clock speed

Parallel support libraries and subsystems software can limit scalability independent of your application. [1]

2.4 Classification of parallel architectures

2.4.1 Flynn’s taxonomy

Flynn's taxonomy is a classification of computer architectures, proposed by Michael J. Flynn in 1966[6].

- SISD (single instruction, single data stream) is type of parallel computing architecture where conventional sequential (non-parallel) processor, based on von Neumann architecture, executes single instructions on same data. It has Single CU and single PE.
- SIMD (single instruction, multiple data stream) is type of parallel computing architecture, typically used for execute same sequence of operation on vector or matrix. It has single CU and multiple PEs. CU controls, activates and deactivates PEs and executes scalar instructions. PEs synchronously executes same arithmetic and logic instruction on different data. A SIMD is a special purpose machine.
- MISD (multiple instructions, single data) is type of parallel computing architecture, where multiple processors execute multiple instructions on same data.
- MIMD (multiple instruction, multiple data stream) is type of parallel computing architecture. Multiple processors execute multiple instruction on different data. MIMD is general purpose machine. MIMD computers require more hardware then SIMD computers, because SIMD computers have only one CU. MIMD computers also require more memory, because each processor must store the program. MIMD systems typically consist of a collection of fully independent processing units or cores, each of which has its own control unit and its own PE. Furthermore, unlike SIMD systems, MIMD systems are usually asynchronous, that is, the processors can operate at their own pace. In many MIMD systems there is no global clock, and there may be no relation between the system times on two different processors. In fact, unless the programmer imposes some synchronization, even if the processors are executing exactly the same sequence of instructions, at any given instant they may be executing different statements.[2]

2.4.2 Memory architectures

- **Shared Memory** is memory system, where a collection of autonomous processors is connected to a memory system via an interconnection network, and each processor can access each memory location. In a shared-memory system, the processors usually communicate implicitly by accessing shared data

[6] https://computing.llnl.gov/tutorials/parallel_comp

structures. [2] Primary disadvantage of shared memory architecture is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management. [1]

- UMA (uniform memory access) is shared memory system, where time to access all the memory locations will be the same for all the cores. UMA systems are usually easier to program, since the programmer doesn't need to worry about different access times for different memory locations.[2]

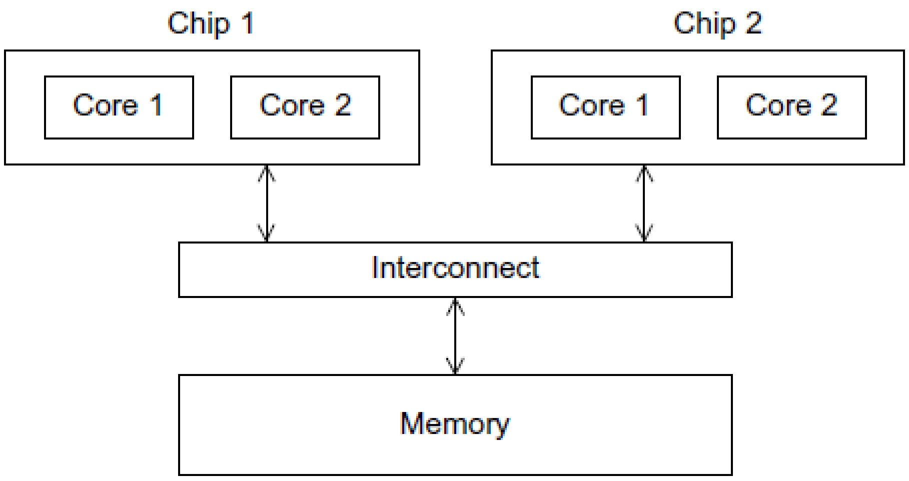


Fig. 7: Principle of shared memory (UMA), passed from [2]

- NUMA (non-uniform memory access) is shared memory system, where a memory location to which a core is directly connected can be accessed more quickly than a memory location that must be accessed through another chip. NUMA has faster access to the directly connected memory. Furthermore, NUMA systems have the potential to use larger amounts of memory than UMA systems. [2]

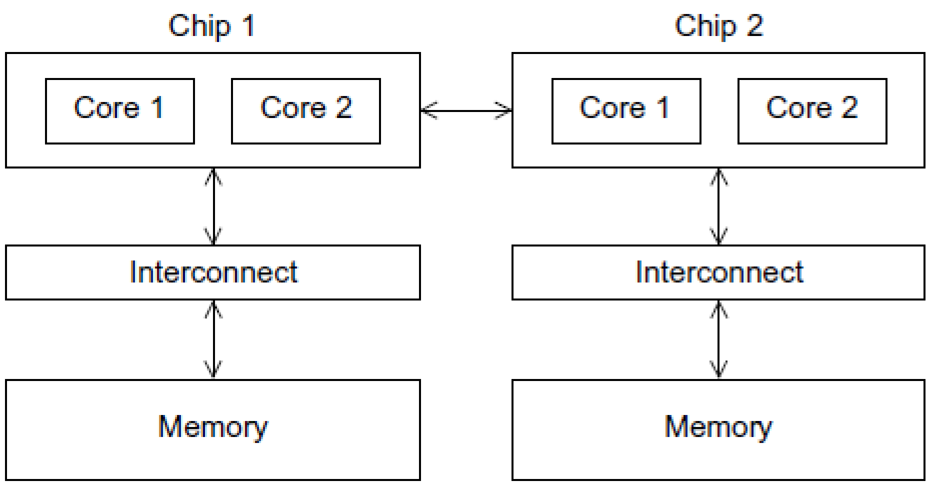


Fig. 8: Principle of shared memory (UMA), passed from [2]

- **Distributed Memory** like shared memory systems, vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory. Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors. Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply. When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility. Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately.[1]

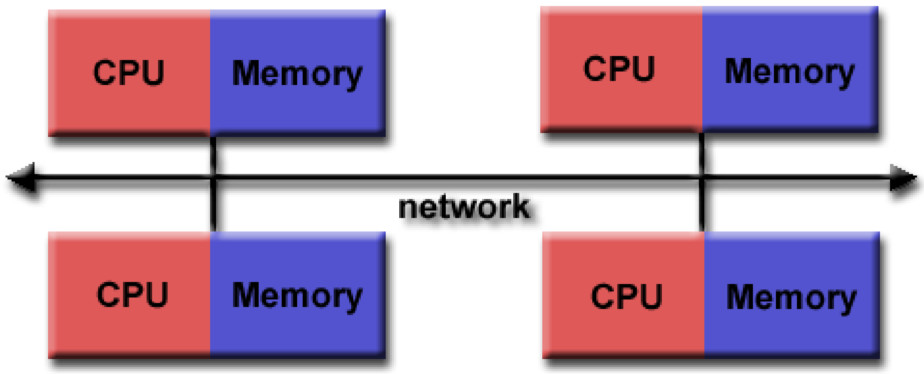


Fig. 9: Principle of distributed memory, passed from [1]

2.4.3 Multicore processors

A multicore processor is an integrated circuit to which two or more processors have been attached for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks. A dual core set-up is somewhat comparable to having multiple, separate processors installed in the same computer, but because the two processors are actually plugged into the same socket, the connection between them is faster. [3]

Multicore processor is MIMD machine. Different cores executes different threads, operating on different data. It uses shared memory.

Ideally, a dual core processor is nearly twice as powerful as a single core processor. In practice, performance gains are said to be about fifty percent: a dual core processor is likely to be about one-and-a-half times as powerful as a single core processor. [3] There are several reasons for increase performance this way. Biggest issue is heat, which makes difficult to make single-core processor with higher frequency.

From the view of massive parallelism on big data, multicore processor is excellent tool, but performance potential is limited by relatively low amount of cores. However many other architectures uses multicore processors.

2.4.4 Distributed systems

In any distributed system, there are many autonomous computation entities and each entity has its own local memory. These entities communicates with each other for achieve common goal. From outside distributed system is MIMD machine with distributed memory.

From the view of massive parallelism on big data, distributed systems can provide excellent performance, however computation entities can be far from each other and this means a lot of relatively slow data transfer. In this category, there is very interesting project, which was created for processing big data - Hadoop.

2.4.4.1 Hadoop

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing. [4]

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. [4]

In traditional approach data travels to very powerful computer, which do all the job. Hadoop's Approach breaks data and computation to smaller pieces and sends each piece of computation to each piece of data. All pieces of data are equal. After computation is finished, each node sends result back, where it is combined to single result. [5]

Hadoop uses large amount of low cost computers. It works on Linux based machines. We can call each single machine as node. There are two types of nodes – slaves and masters. Slaves has two components – *Task Tracker* and *Data Node*. *Task Tracker* is part of *MapReduce*. It process smaller peace of task that has been given to this particular node. Data node is part of *HDFS*. It manage the data that has been given to this particular node. Master has two additional components against slave – *Job Tracker* and *Name Node*.

The application contacts master node. *Job Tracker* splits job to smaller pieces and send them to his slave's *Task Trackers*. When job is done, *Task Trackers* send results back to *Job Trackers*, where results are combined back and send to the application. *Name Node* running on *Master Node* is responsible to keep index of which data is residing on which *Data Node*.

Architecturally, Hadoop is just the combination of two technologies: the *Hadoop Distributed File System (HDFS)* that provides storage, and the *MapReduce* programming model, which provides processing. [5]

HDFS exists to split, distribute, and manage chunks of the overall data set, which could be a single file or a directory full of files. These chunks of data are pre-loaded onto the worker nodes, which later process them in the MapReduce phase. By having the data local at process time, HDFS saves all of the headache and inefficiency of shuffling data back and forth across the network. [5]

In the *MapReduce* phase, each worker node spins up one or more tasks (which can either be *Map* or *Reduce*). Map tasks are assigned based on data locality, if at all possible. A *Map* task will be assigned to the worker node where the data resides. *Reduce* tasks (which are optional) then typically aggregate the output of all of the dozens, hundreds, or thousands of map tasks, and produce final output. The *Map* and *Reduce* programs are where your specific logic lies, and seasoned programmers will immediately recognize *Map* as a common built-in function or data type in many languages. Hadoop will parse the data in *HDFS* into user-defined keys and values, and each key and value will then be passed to your *Mapper* code. [5]

Attributes of Hadoop:

- Easy programming - Programmers doesn't have to worry about:
  - Where the file is located
  - How to manage failures
  - How to break computations into pieces
  - How to program for scaling
- Scalability – Hadoop can consist of random number of computers. If we need more performance, we simply buy another computers. Scalability cost is linear. If we need to double performance, we have to double the number of computers.
- Failure tolerance - Hadoop keeps hardware failures in mind. By default Hadoop keeps 3 copies of each file on different computers. When the computer fails, system keeps on running. Data is available from different nodes.

- Maintenance – as most distributed systems, Hadoop is quite hard to set up and make work. Hadoop also requires relatively low cost to buy computers and keep them running.

Hadoop would be an excellent tool to achieve goal of this thesis, but hardware and software configuration of this solution is quite complicated, so after consultation with my supervisor, we have chosen different architecture.

2.4.5 Parallelism using accelerators

Hardware accelerators are special devices, which offer higher performance through parallelism.

Unlike CPU, hardware accelerator is specific-purpose devices. It has large amount of parallel execution units which don't have big instructions sets like CPU, it has smaller cache, etc., but thanks to that more cores can be placed on one chip. It also has its own memory, which is significantly faster and more advanced, then usual memories. Hardware accelerators are SIMD machines.

Idea of using hardware accelerators is simple. CPU still executes main part of code, and accelerator is used only for difficult calculations. In this case program move required data to memory of accelerator and accelerator executes required calculations. When calculations are completed, accelerator sends output data back to CPU and program continues. This idea implicates biggest disadvantage of accelerators – memory traffic. It is not wise to make everything parallel. If data transfer last longer, then calculation on CPU, there is no reason to do it. Also, developers can be limited by accelerator memory.

Hardware accelerators are excellent for huge amount of same operations on data, for example matrix multiplying.

2.4.5.1 GPU

Graphical processing unit is special computer chip used for rapid mathematical calculations, primary used for rendering graphic output on computers. To do this, GPU is built on highly parallel structure, which can be used for different types of operations. Technically it is SIMD architecture.

GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, engineering, and enterprise applications. GPUs now power energy-efficient datacenters in government labs, universities, enterprises, and small-and-medium businesses around the world. [6]

There are two most expanded frameworks for developing parallel application which uses GPU acceleration:

- CUDA (Compute Unified Device Architecture) is developed by NVIDIA and unfortunately it works only with NVIDIA graphic cards. It provides a few simple C and C++ extensions that enable expressing fine-grained and coarse-grained data and task parallelism. [6] However CUDA is very popular in Czech Republic, in word scale more popular OpenCL.

- OpenCL (Open Computing Language) is open standard maintained by the non-profit consortium Khronos Group. It also work with all types of graphics card (NVIDIA, AMD…). Similar to CUDA, OpenCL provides C and C++ extensions to enable developers create massively parallel applications.

GPU accelerating is also good way to achieve goal of this thesis, however it is used by another student in his thesis, which is focused on similar problem as this one. For this reason, I have chosen different architecture to implement on.

2.4.5.2 **MIC**

Intel MIC (Many Integrated Core) is special device architecture, which raised from Larrabee. Larrabee was codename for GPU chip, which was developing by Intel. Due to delays and disappointing early performance figures, Larrabee was canceled. Short time after that, Intel started to work on MIC architecture, which inherited many design elements from Larrabee project, but it wasn't made as graphical chip, but as accelerator for high performance computing.

MIC architecture combines many Intel CPU cores onto a single chip. Developers interested in programming these cores can use standard C, C++, and FORTRAN source code. The same program source code written for Intel MIC products can be compiled and run on a standard Intel® Xeon® processor. Familiar programming models remove training barriers, allowing the developer to focus on the problems rather than software engineering. [7]

3 Analytic programming

As it has been said before in this thesis, I will solve problem of founding Be stars in huge amount of visible spectra. It has been also said, then Be star has specific character of its specter. Before I can compare and find Be stars, I must find and learn a pattern of typical Be star. For this job, I have chosen analytic programming.

3.1 Symbolic regression

Symbolic regression is process, where trickier structure, which should describe required behavior, is building from small building stones. For example approximate set of measured data and find functional dependence between them, or find proper trajectory of robots, or draw proper design of logical circuit and many others, for which this principle of the proposal is appropriate. [14]

If we talk about symbolic regression in connection with evolutionary algorithms, at the moment, there exist several tools. The best known is definitely genetic programming (Koza, 1998), (Koza, 1999) and also grammatical evolution (O'Neill, 2003). [14]

3.2 Evolutionary algorithms

Evolutionary algorithms are basic building stone of analytic programing. Very simply said, analytic programing is set of rules, how to use evolutionary algorithm to purpose of symbolic regression.

Evolutionary algorithms are numeric methods, used mainly on optimization of functions (finding minimum or maximum). They use principle of Darwin's and Mendel's evolution theory[7]. According to this theory, all species are evolving thanks to the fact, then children yields to mutation during their creation. Parents and children which are not strong enough die and free space for new stronger children, which later become parents.

Cost function means function, which optimization (finding minimum or maximum) leads to found optimal values of its arguments. [14]

[7] http://www.darwins-theory-of-evolution.com/

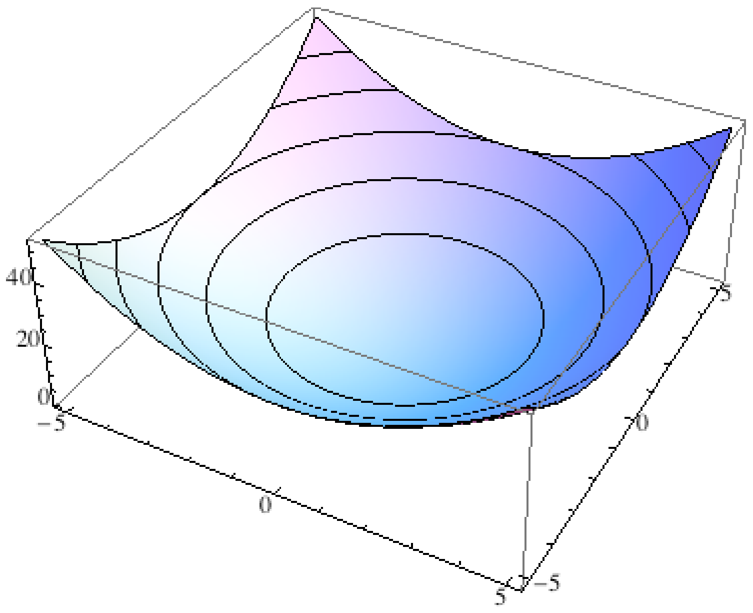


Fig. 10: Example of cost function in graphics expression, passed from [13]

At the case of use evolution principles for purpose of complex calculation, procedure is fallowing: [14]

1. Definition of evolution parameters: parameters, which control run of algorithm or regularly ends it, if predetermined ending criterions are reached, must be defined for each algorithm. Part of this point is set of cost function, eventually fitness (adapted return value of cost function). Cost function means usually mathematical model of problem, which minimization and maximization (generally existence) leads to solution of problem. This function with eventual limitative conditions is some "equivalent of environment", in which quality of actual individuals is evaluated.
2. Generating of first population (*population* – generally matrix N * N, where N is number of individual's parameters, also used the term D, and M is number of individuals in population): first population of individuals is generated by number of optimized arguments of cost function. Individual means vector of numbers, which has so many components as optimized arguments of cost function. These components are set randomly and each individual represents one possibly correct solution of problem. Set of individuals is population.
3. All individuals are evaluated thru defined cost function and to all of them is set a) either head-on value returned from cost value, or b) fitness, which is adapted (usually normalized) value of cost function.
4. Arises select of parents by their quality (fitness, value of cost function), eventually by other criteria.

5. Descendants are created by crossover of parents. Process of crossover is different in each algorithm. Parts of parents are reversed in classical genetic algorithm, in differential evolution is crossover some kind of vector operation.
6. Each individual is mutated. In other words, new individual is altered thanks to proper random process. This step is equivalent of biological gene mutation of individual.
7. Each new individual is valuated same as in step 3.
8. Best individuals are selected.
9. Selected individuals (we remind that this is a vector of numbers) fill new population.
10. Old population is forgotten (destroyed, erased, dying …) and new population succeeds on its place and evolution continues by step 4.

Steps 4-10 are repeating until number of cycles, set by user is done, or until required quality of solution is reached. Principe of evolutionary algorithm written above is general and it may vary in concrete cases. [14]

I use two evolutionary algorithms in my application: differential evolution (DERand1Bin) and SOMA (AllToOne).

3.2.1 Differential evolution

Differential evolution is algorithm invented by Ken Price and Rainer Storm in 1995. It is based on vector operations. There is many variant of differential evolution, dissimilar in minor differences during evolution cycle. Fallowing explanation describes variant DERand1Bin, which is used in my implementation.

3.2.1.1 Setting parameters and preparing algorithm

Same as in other evolutionary algorithms, differential evolution has parameters, which must be set before algorithm starts and which significantly affect effectivity of algorithm. Some parameters has recommended values, but they are not universal and correct values may be different.

- CR [0, 1]. It is so-called crossover threshold. This parameter express chance to mutate parameter of individual. Mutation is disabled, if CR is set to 0. Recommended value is 0.8 – 0.9.
- D. Dimension of the problem. It is fixed and it can by change only by changing problem (or its expression).
- NP [10D, 100D]. This parameter represents size of population. Value is depends on dimension of problem. It must be more then 3, because differential evolution requires 4 individuals to execute evolution. Recommended value is 10D.
- F [0, 2]. Mutation constant. Value, which affects one step in evolution. Recommended value is 0.3 – 0.9.
- Generations. Number of evolution cycles. Must be higher than 0.

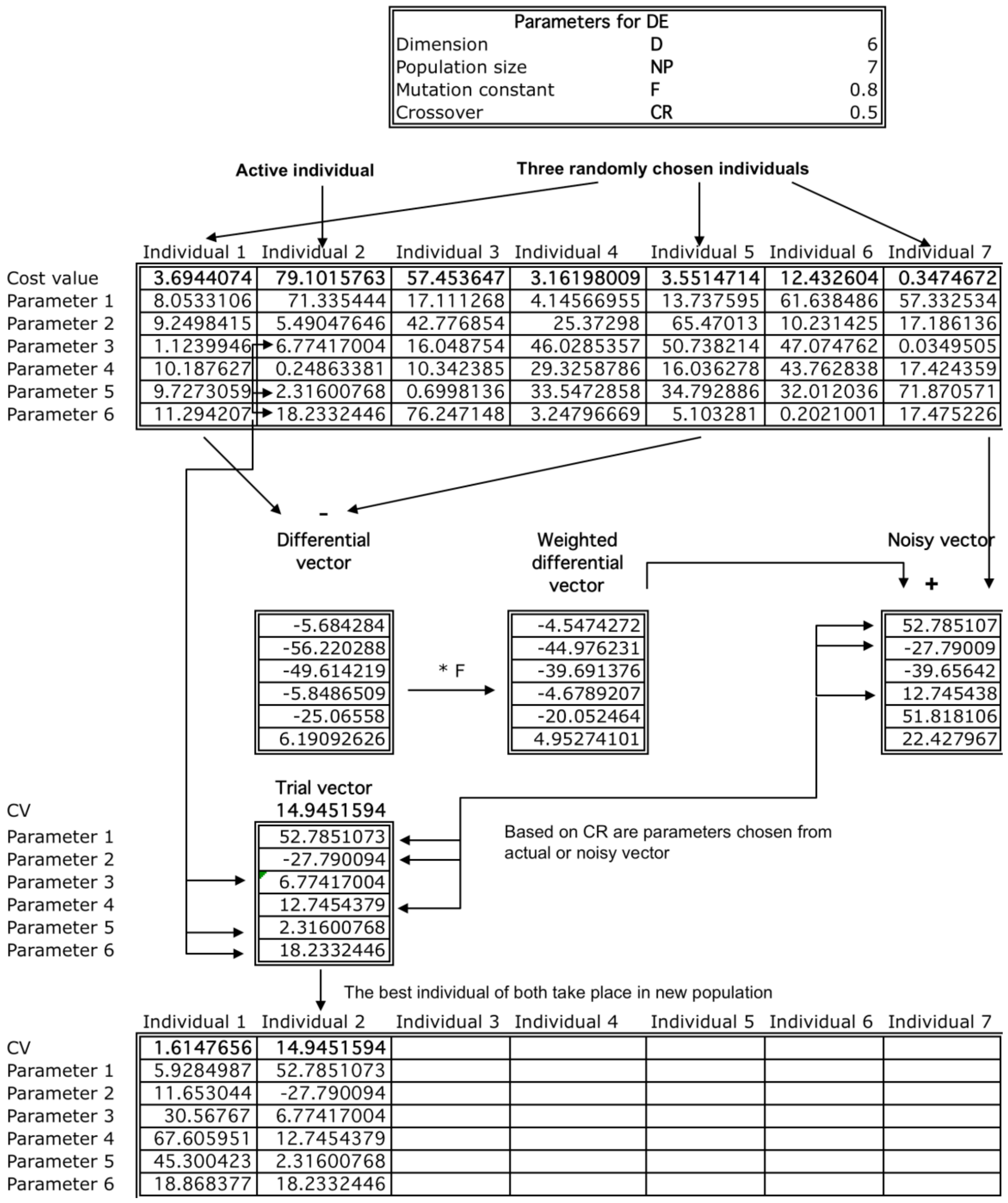
When parameters are set, first population can be created. Parameters of first population are random in boundaries of cost function. Before evolution itself can begin, population must be evaluated by cost function.

3.2.1.2 **Evolution cycle**

Graphical interpretation of evolution cycle is represented on Figure 11. Each evolution cycle is what happens in points 5 and 6 in procedure described above.

For each individual (called active individual), algorithm randomly select 3 other individuals (vectors). Second randomly selected vector is subtract from first randomly selected vector. This operation gives us so-called differential vector. Differential vector is multiplied by mutation constant F and this gives us so-called weight vector. Weight vector is add to third randomly selected vector and this gives us so called noise vector. For each element of active vector (individual) is generated random number from 0 to 1. If generated number is lesser, then crossover threshold CR, place of this element is replace by element from noise vector (on same position). This gives us so-called experimental vector. This experimental vector is evaluated by cost function and if fitness of experimental vector is better, then fitness of actual vector, experimental vector replace it.

This happened for each generation of algorithm. Best individual in last generation represents result.


Active individual
Three randomly chosen individuals
-
Differential vector
* F
Weighted differential vector
Noisy vector
+
Based on CR are parameters chosen from actual or noisy vector
The best individual of both take place in new population

Parameters for DE		
Dimension	D	6
Population size	NP	7
Mutation constant	F	0.8
Crossover	CR	0.5

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6	Individual 7
Cost value	**3.6944074**	**79.1015763**	**57.453647**	**3.16198009**	**3.5514714**	**12.432604**	**0.3474672**
Parameter 1	8.0533106	71.335444	17.111268	4.14566955	13.737595	61.638486	57.332534
Parameter 2	9.2498415	5.49047646	42.776854	25.37298	65.47013	10.231425	17.186136
Parameter 3	1.1239946	6.77417004	16.048754	46.0285357	50.738214	47.074762	0.0349505
Parameter 4	10.187627	0.24863381	10.342385	29.3258786	16.036278	43.762838	17.424359
Parameter 5	9.7273059	2.31600768	0.6998136	33.5472858	34.792886	32.012036	71.870571
Parameter 6	11.294207	18.2332446	76.247148	3.24796669	5.103281	0.2021001	17.475226

Differential vector	Weighted differential vector	Noisy vector
-5.684284	-4.5474272	52.785107
-56.220288	-44.976231	-27.79009
-49.614219	-39.691376	-39.65642
-5.8486509	-4.6789207	12.745438
-25.06558	-20.052464	51.818106
6.19092626	4.95274101	22.427967

	Trial vector
CV	**14.9451594**
Parameter 1	52.7851073
Parameter 2	-27.790094
Parameter 3	6.77417004
Parameter 4	12.7454379
Parameter 5	2.31600768
Parameter 6	18.2332446

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6	Individual 7
CV	**1.6147656**	**14.9451594**					
Parameter 1	5.9284987	52.7851073					
Parameter 2	11.653044	-27.790094					
Parameter 3	30.56767	6.77417004					
Parameter 4	67.605951	12.7454379					
Parameter 5	45.300423	2.31600768					
Parameter 6	18.868377	18.2332446					

Fig. 11: Example of cost function in graphics expression, passed from [14]

3.2.2 SOMA (Self Organization Migration Algorithm)

Soma has been invented by prof. Ing. Ivan Zelinka, Ph.D. in 1999. Same as in differential evolution, its function is based on vector operations. There is also many variants of algorithm. I will explain variant called AllToOne, which is used in my implementation.

3.2.2.1 **Setting parameters and preparing algorithm**

Same as in differential evolution, SOMA has parameters, which must be set before algorithm starts and which significantly affect effectivity of algorithm. Some parameters has recommended values, but they are not universal and correct values may be different.

- PathLength (1, 5]. Define how far to the leader will individual travel. PathLength = 1 means, individual will stop on leader's position, PathLength = 2 means he will stop at same distance behind leader, as it starts. Recommended value is 3.
- Step (0.11, PathLength). Define size of one "jump". PathLength can't be integer multiple of this number. Higher values are recommended, when we don't have idea about shape of cost function.
- PRT [0, 1]. It means perturbation. It is one of most important parameters and it influences direction of active individual's moving.
- D. Dimension of the problem. It is fixed and it can by change only by changing problem (or its expression).
- PopSize [10, user defined]. This parameter represents size of population. Value is depends on dimension of problem.
- Migration. Number of evolution cycles. Must be higher than 0.

Same as in differential evolution when parameters are set, first population can be created. Parameters of first population are random in boundaries of cost function. Before evolution itself can begin, population must be evaluated by cost function.

3.2.2.2 **Evolution cycle**

Graphical interpretation of evolution cycle is represented on Figure 12. Each evolution cycle is what happens in points 5 and 6 in procedure described above.

At the beginning of each cycle, best individual of previous evaluation is promoted to so-called leader and perturbation vector is generated. Perturbation vector has same number of argument as number of dimension. Random number from 0 to 1 is generated for each argument. If this number is smaller, then PRT, then argument in perturbation vector is 1. Otherwise it is 0.

PRT = 0.5	
Random number	PRTVector
0.052	1
0.744	0
0.942	0
0.248	1
0.669	0
0.416	1

Tab 3: Example of generating PRTVector

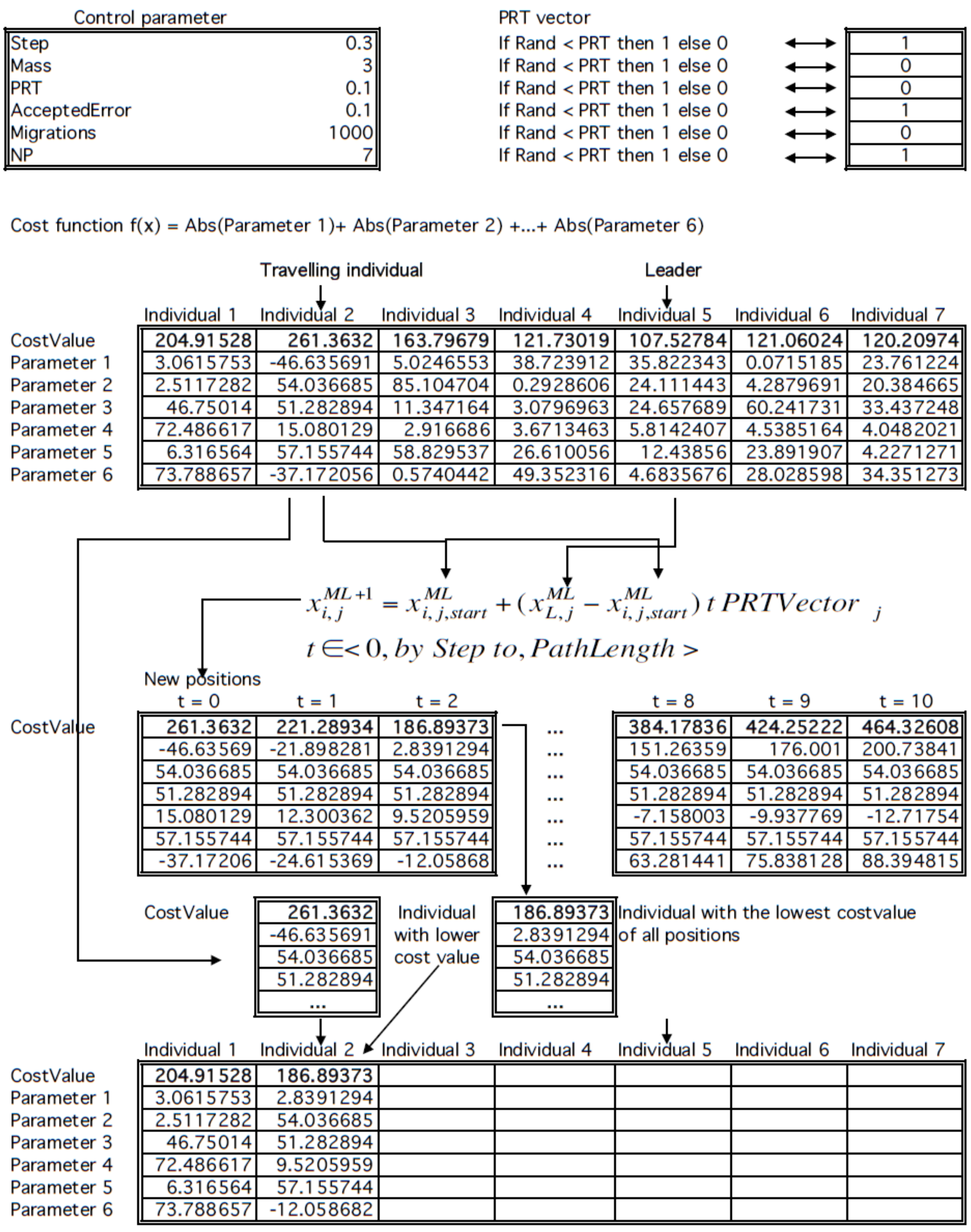
Simply said, each individual travels to the leader (leader doesn't move and it doesn't change during evolution cycle). Length of the path is defined by parameter PathLength and number of steps by parameter Steps. Direction of move is altered by PRTVector. If argument of vector is 0, it means then dimension of move is ignored. This can be better explain by equation:

$$x_{i,j}^{ML+1} = x_{i,j,start}^{ML} + \left(x_{L,j}^{ML} - x_{i,j,start}^{ML} \right) tPRTVector_j$$

Where t means traveled distance ($0 \leq t < PathLength$), i means index of individual and j means index of parameter.

Parameters of individual is noted on every step, which individual makes. This give us a new population of individuals, which are in one line. When active individual finish the journey, new created population is evaluated by cost function. If best individual of this new population is better, then active individual, best individual replace active individual in original population. Leader stay unchanged.

This happened for each generation of algorithm. Best individual in last generation represents result.


Control parameter

Step	0.3
Mass	3
PRT	0.1
AcceptedError	0.1
Migrations	1000
NP	7

PRT vector

If Rand < PRT then 1 else 0	$\longleftrightarrow$	1
If Rand < PRT then 1 else 0	$\longleftrightarrow$	0
If Rand < PRT then 1 else 0	$\longleftrightarrow$	0
If Rand < PRT then 1 else 0	$\longleftrightarrow$	1
If Rand < PRT then 1 else 0	$\longleftrightarrow$	0
If Rand < PRT then 1 else 0	$\longleftrightarrow$	1

Cost function f(x) = Abs(Parameter 1)+ Abs(Parameter 2) +...+ Abs(Parameter 6)

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6	Individual 7
CostValue	204.91528	261.3632	163.79679	121.73019	107.52784	121.06024	120.20974
Parameter 1	3.0615753	-46.635691	5.0246553	38.723912	35.822343	0.0715185	23.761224
Parameter 2	2.5117282	54.036685	85.104704	0.2928606	24.111443	4.2879691	20.384665
Parameter 3	46.75014	51.282894	11.347164	3.0796963	24.657689	60.241731	33.437248
Parameter 4	72.486617	15.080129	2.916686	3.6713463	5.8142407	4.5385164	4.0482021
Parameter 5	6.316564	57.155744	58.829537	26.610056	12.43856	23.891907	4.2271271
Parameter 6	73.788657	-37.172056	0.5740442	49.352316	4.6835676	28.028598	34.351273

$$x_{i,j}^{ML+1} = x_{i,j,start}^{ML} + (x_{L,j}^{ML} - x_{i,j,start}^{ML})\, t\, PRTVector_{j}$$

$$t \in < 0, by\ Step\ to, PathLength >$$

New positions

	t = 0	t = 1	t = 2		t = 8	t = 9	t = 10
CostValue	261.3632	221.28934	186.89373	...	384.17836	424.25222	464.32608
	-46.63569	-21.898281	2.8391294	...	151.26359	176.001	200.73841
	54.036685	54.036685	54.036685	...	54.036685	54.036685	54.036685
	51.282894	51.282894	51.282894	...	51.282894	51.282894	51.282894
	15.080129	12.300362	9.5205959	...	-7.158003	-9.937769	-12.71754
	57.155744	57.155744	57.155744	...	57.155744	57.155744	57.155744
	-37.17206	-24.615369	-12.05868	...	63.281441	75.838128	88.394815

CostValue		Individual with lower cost value		Individual with the lowest costvalue of all positions
	261.3632		186.89373	
	-46.635691		2.8391294	
	54.036685		54.036685	
	51.282894		51.282894	
	...		...	

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6	Individual 7
CostValue	204.91528	186.89373					
Parameter 1	3.0615753	2.8391294					
Parameter 2	2.5117282	54.036685					
Parameter 3	46.75014	51.282894					
Parameter 4	72.486617	9.5205959					
Parameter 5	6.316564	57.155744					
Parameter 6	73.788657	-12.058682					

Fig. 12: Example of cost function in graphics expression, passed from [14]

3.3 Principe of analytic programming

At the opposite of genetic programming or grammatical evolution, analytic programming isn't connected with concrete evolutionary algorithm, grammar or tree representation.

Analytic programming isn't independent evolutionary algorithm, but more something like transformation or projection from set of basic symbolic objects to set of possible programs, which can be construct from this symbolic objects. [14]

Analytic programming works with:

- Functions: sin, cos…
- Operands: +, -, *, / …
- Terminals: 9.754, K…

For these object we use name GFS – general function set. Object are classified by the number of their arguments to GFS_{all} (all elements of GFS), GFS_{0arg} (0 arguments)…

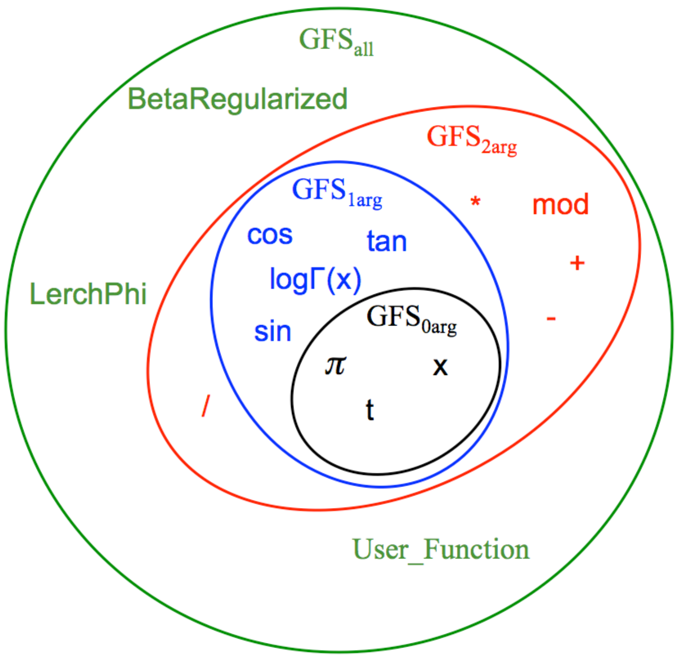
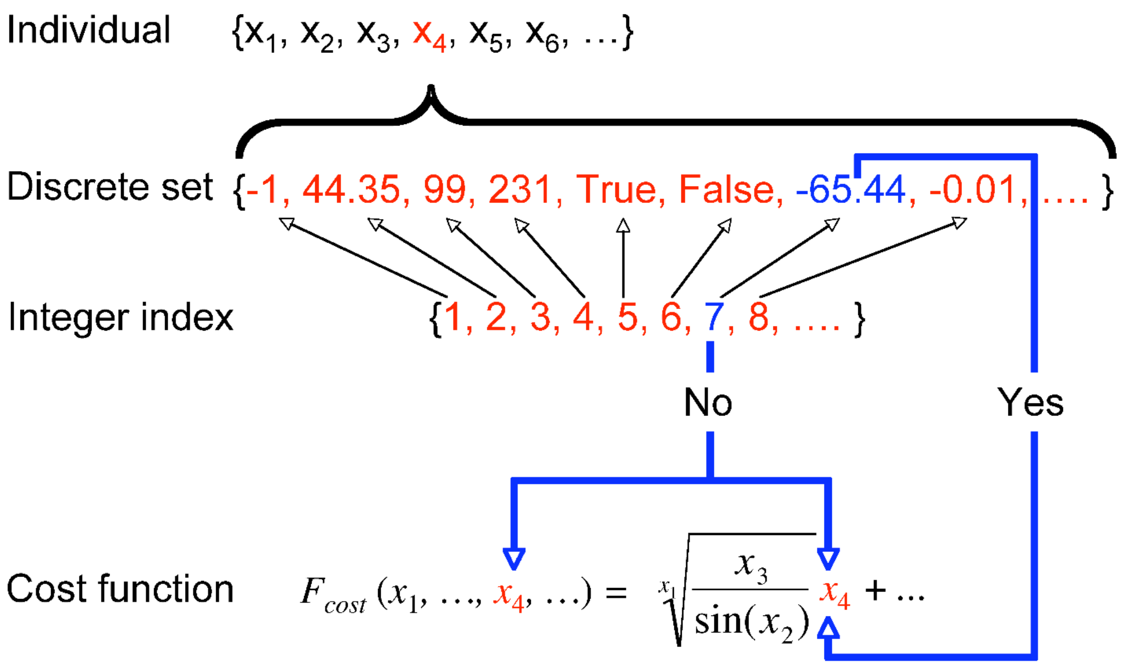


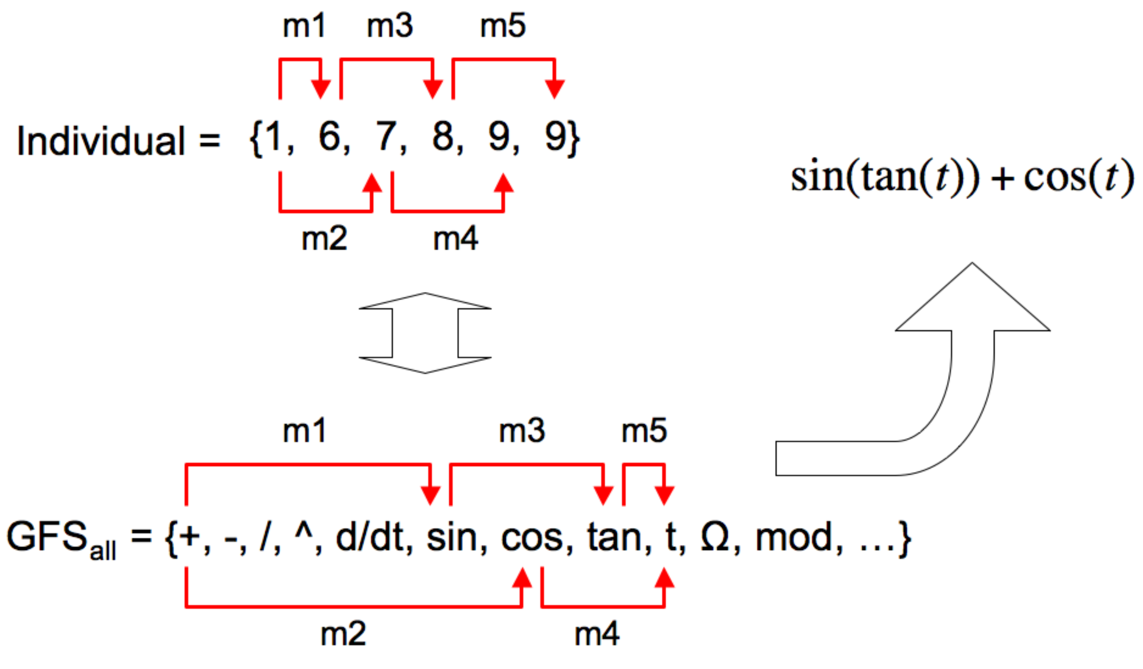
Fig. 13: Symbolic representation of set GFS, passed from [14]

Basic idea of analytic programming is representing each object of GFS as integer number. For example x is 0, sin is 1…. Now whole function can be represented as vector (or individual) of this numbers that we can use any evolutionary algorithm to breed individual. Evaluation of cost functions is calculated as difference between areas of original problem's function and function from evolutionary algorithm.



Individual
{x1, x2, x3, x4, x5, x6, …}
Discrete set
{-1, 44.35, 99, 231, True, False, -65.44, -0.01, …. }
Integer index
{1, 2, 3, 4, 5, 6, 7, 8, …. }
No
Yes
Cost function
$F_{cost}(x_1, \ldots, x_4, \ldots) = \sqrt[x_1]{\frac{x_3}{\sin(x_2)}}\,x_4 + \ldots$

Fig. 14: Principle of working with discrete set, passed from [14]



m1
m3
m5
Individual = {1, 6, 7, 8, 9, 9}
m2
m4
sin(tan(t)) + cos(t)
m1
m3
m5
GFSall = {+, -, /, ^, d/dt, sin, cos, tan, t, Ω, mod, …}
m2
m4

Fig. 15: Conversion of integer individual to function, passed from [14]

Unfortunately evolutionary algorithm can give a pathological function – function, which doesn't make sense, for example *4*-sin. This is why we need subsets of GFS. Synthetizing process measure remaining parameters. When it is closing to end, objects from lower GFS are chosen. This save procedure highly increase effectivity of algorithm.

There are 3 version of analytic programming, depending on representation of constants:

- AP_{basic} – constants are generated with other functions and operators. This highly increase GFS size.
- AP_{meta} – there is only one constant K, which is indexed as K_1, K2, K_3… and algorithm uses other evolutionary algorithm to breed the constants.
- AP_{nf} – it is familiar with AP_{meta}, but constants are obtained by using proper numeric method.

4 Design and implementation

4.1 Technology

After consultation with thesis supervisor, we choose to implement serial version of analytic programming and parallel version of algorithm which will compare model specter of BE stars with specter of unknown space objects. Both algorithms will be implemented on in C++ from performance reason. Comparing algorithm will be implemented for multicore processor algorithm, using OpenMP API.

My implementation consist of 3 logical parts. First part implements analytic programming, as described in Chapter 3. Second part solves retrieving massive astronomical data from MFF UK Praha. Third part implements parallel comparer of model and input spectra.

4.2 Implementation of analytic programming

Before anything could happen, input settings and data must be loaded. For this purpose, there is class *DataPool*. All settings of evolution algorithms are stored in file *settings.txt*.

As was mentioned above, I have chosen 2 evolutionary algorithms as core of analytic programing: differential evolution and SOMA. Both algorithms are implemented in class *EvolutionAlgorithms*. Core of algorithms is class *Population*, which represents population as several two dimensional vectors and contains many methods necessary for population maintenance.

Before evolutionary algorithm can begin, random population must be created. Because this population is used for analytic programming, those populations represents expressions as mentioned in theoretical part above. This fact leads to creation of pathological expressions like:

78*/xx(

For all life time of population, checking and repairing of these pathological functions is necessary. If it happened, wrong attribute are replaced by random argument with required amount of arguments. There is also checking to boundaries, so attributes will be always generated within proper range. Rest of evolutionary algorithm is implemented by diagrams in Chapter 3.2.2.

Main core of analytic programming is implemented in class *Functions*. There are also many supportive functions, which allow evolution algorithms and analytic programming working. There is functions in this class which:

- Manage working of analytic programming.
- Represents numeric attributes as expression members.
- Evaluates individuals from evolution algorithms by fitness value.
- Takes care about constants.
- Writes best individuals to log

I used meta-evolution for creating constants. It means, than for every expression from evolution there is another evolution algorithm, which finds best values of constants.

Because results of analytic programming was not as expected, I implemented enhanced search to algorithm. It pick best individual from one generation and put it as argument for next generation. This leads to much better results, but also big increasing of result expression.

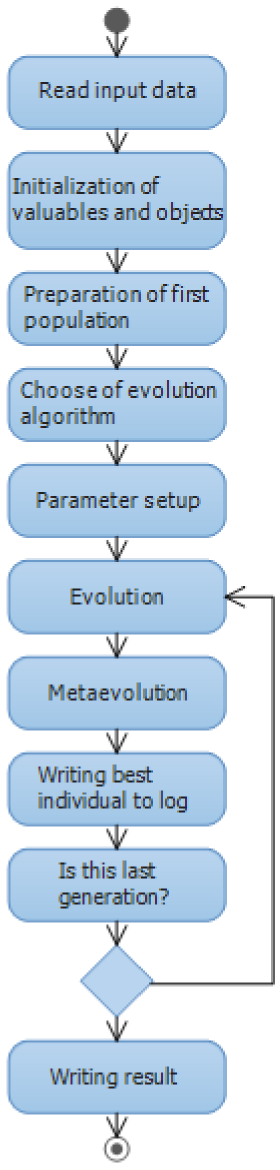

Read input data
Initialization of valuables and objects
Preparation of first population
Choose of evolution algorithm
Parameter setup
Evolution
Metaevolution
Writing best individual to log
Is this last generation?
Writing result

Fig. 16: Activity diagram of implementation of analytic programming

4.3 Data source

Data for experiment comes from Faculty of Mathematics and Physics, Charles University, Prague. The origin of spectra is Ondrejov, 2m telescope, Academy of Science, Czech Republic.

It consist of database of metadata and data. Data are stored in binary files, which was copied to *codd* machine by my supervisor.

Database of metadata is stored in PostgreSQL database and it has been shared as database export. That leads to need of restore database. This restore has been done on machine dbsys.vsb.cz, which is under responsibility of my faculty.

At the beginning I had to find out, how to retrieve data from binary files using database of metadata. Each binary file consist one type of measurement for one dataset. Types of measurements:

- WAVE
- loglam
- mask
- and_mask
- ivar
- sky
- flux
- FLUX

Important for this thesis was two of them:

- FLUX
  - Light intensity.
  - Name of binary files ends with *00.rcd*.
- loglam
  - Logarithm wavelength.
  - Name of binary files ends with *01.rcd*.

Data in binary files are stored as floats and many measurements are stored in one file. To retrieve where the measurement begins and it ends, there is database of metadata.

Database consist of much information and most of them has no use for my task. Important tables are:

- btcolmeta
- btmeta
- btdata

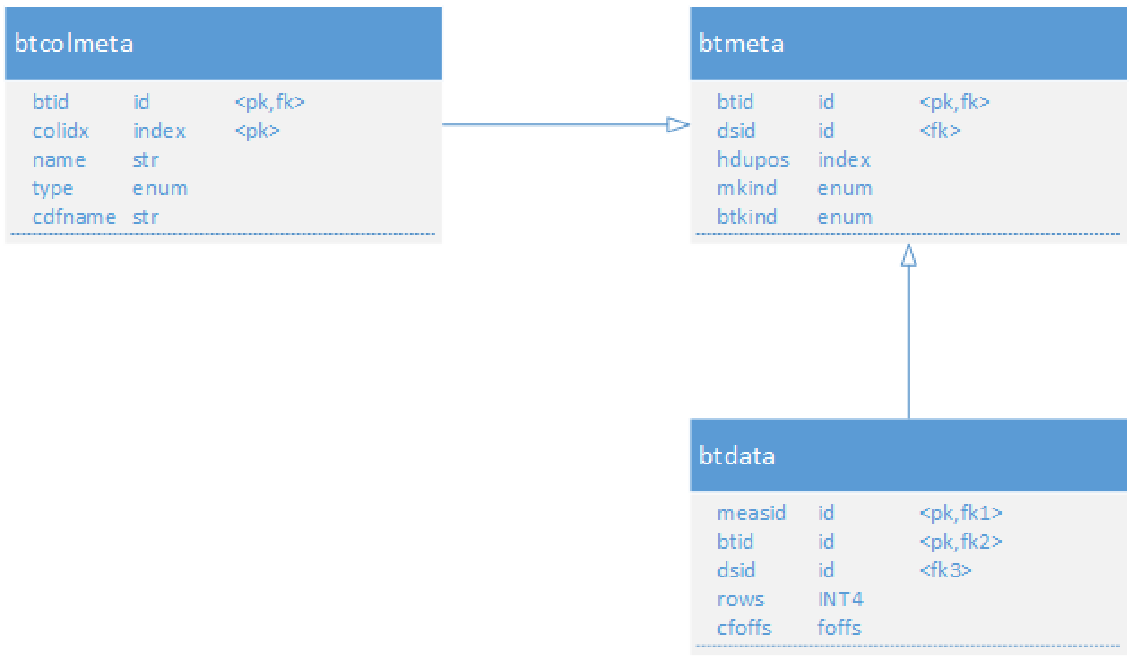


Fig. 17: ER diagram of important tables from metadata database

Important values from selected important tables are:

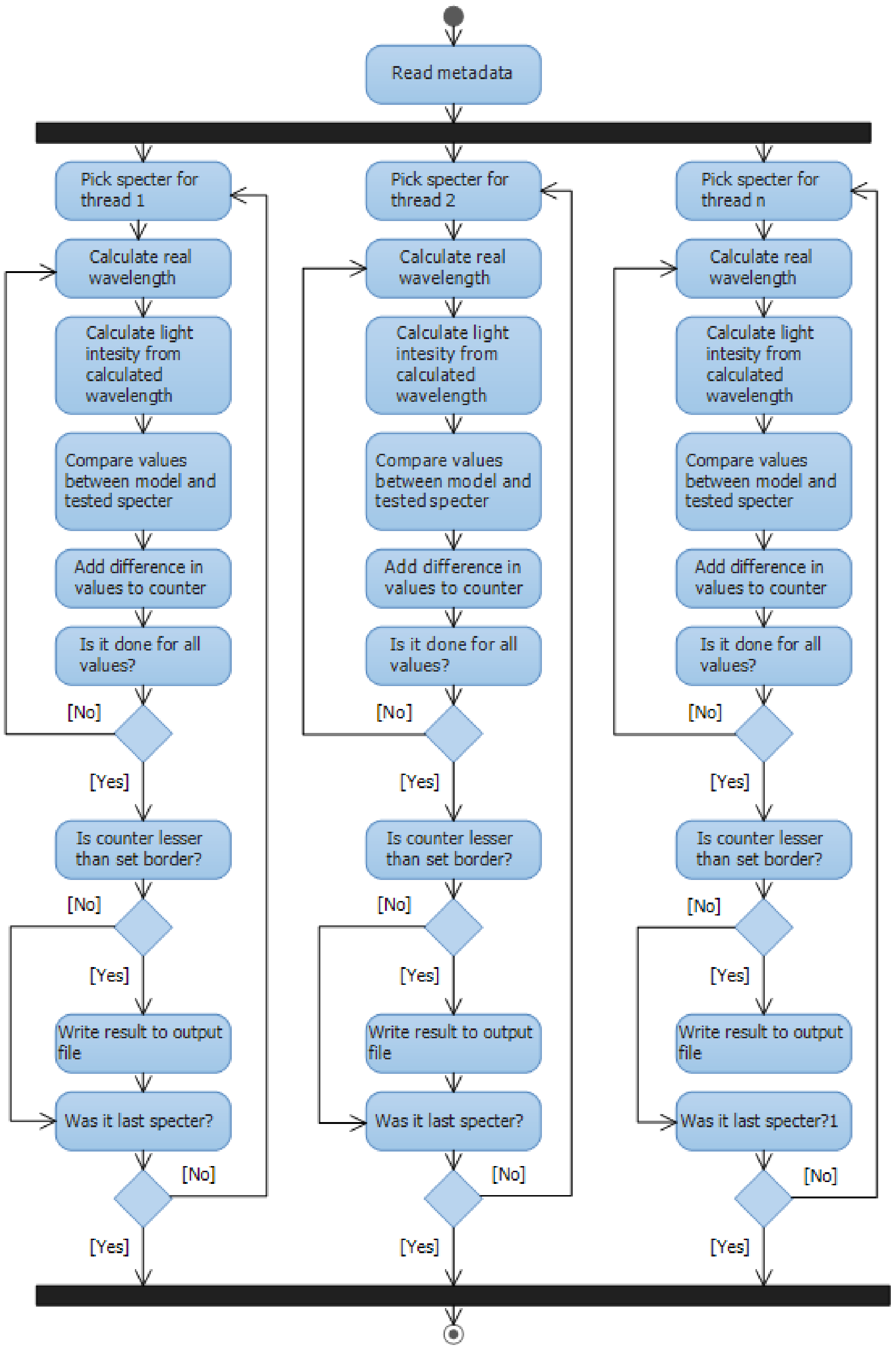
- cdfname – name of binary file
- measid – ID of measurement
- rows – length of specific measurement
- cfoffs – index of beginning of specific measurement

Because of requirement of very small part of metadata and potential lag between database and application server, I did not directly connect my application with database, but instead of it, I created script, which select required values and stored in file. This file should be copied to machine, where other calculation runs. Prescription for script, which creates text file, with all required information for one dataset:

```
C:;
cd C:\Program Files\PostgreSQL\9.4\bin
psql -d [db name] -t -A -F"," -c
"SELECT DISTINCT count(*)
FROM btcolmeta LEFT JOIN btdata ON btdata.btid=btcolmeta.btid
where btcolmeta.cdfname='[binary filename]'"
> "[output file name]"
echo , >> "[output file name]"
psql -d [db name] -t -A -F"," -c
"SELECT DISTINCT btdata.measid,btdata.cfoffs,btdata.rows
FROM btcolmeta LEFT JOIN btdata ON btdata.btid=btcolmeta.btid
where btcolmeta.cdfname='[binary filename]'" >> "[output file name]"
```

4.4 Parallel comparisons of spectra

This part of application compare outputs from both analytic programming and data source parts. At the very beginning file with metadata from database must be loaded. After that, algorithm assign one same task to all available threads.



Read metadata
Pick specter for thread 1
Pick specter for thread 2
Pick specter for thread n
Calculate real wavelength
Calculate light intesity from calculated wavelength
Compare values between model and tested specter
Add difference in values to counter
Is it done for all values?
[No]
[Yes]
Is counter lesser than set border?
[No]
[Yes]
Write result to output file
Was it last specter?
Was it last specter?1
[No]
[Yes]

Fig. 18: Activity diagram of implementation of spectra comparisons.

5 Experiments

5.1 CODD machine configuration

Analytic programming and spectre comparer has been deployed on machine codd, under responsibility of our faculty.

- Operating system:
  - SUSE Linux Enterprise Server 11 (x86_64)
- Processor:
  - Architecture: x86_64
  - CPU op-mode(s): 32-bit, 64-bit
  - Byte Order: Little Endian
  - CPU(s): 84
  - On-line CPU(s) list: 0-83
  - Thread(s) per core: 2
  - Core(s) per socket: 6
  - Socket(s): 7
  - NUMA node(s): 7
  - Vendor ID: GenuineIntel
  - CPU family: 6
  - Model: 45
  - Stepping: 7
  - CPU MHz: 1200.000
  - BogoMIPS: 4800.16
  - Virtualization: VT-x
  - L1d cache: 32K
  - L1i cache: 32K
  - L2 cache: 256K
  - L3 cache: 15360K
  - NUMA node0 CPU(s): 0-5,42-47
  - NUMA node1 CPU(s): 6-11,48-53
  - NUMA node2 CPU(s): 12-17,54-59
  - NUMA node3 CPU(s): 18-23,60-65
  - NUMA node4 CPU(s): 24-29,66-71
  - NUMA node5 CPU(s): 30-35,72-77
  - NUMA node6 CPU(s): 36-41,78-83
- Memory:
  - 978 GB

5.2 DBSYS machine configuration

Database with metadata has been restored on machine dbsys.vsb.cz, under responsibility of our faculty.

- Operating system:
    - Windows Server 2008 R2 Datacenter, 64-bit, Service pack 1
- Processor:
    - Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz 2.90GHz (2 processors)
- Memory:
    - 288 GB
- Database:
    - PostgreSQL 9.2.4, compiled by Visual C++ build 1600, 32-bit

5.3 Analytic programming results

Because synthesis from analytic programming would be done only few times in comparison with comparing spectra, performance is not really important issue. Most important is precision of algorithm.

This experiment has been performed on model spectra of Be star.[8] Model spectra has been cut by unchanging parts and only emission line has been left with 50 points.

Algorithm parameters:

- GENERAL SETTINGS:
    - Dimension = 21
    - Constant minimum = -100
    - Constant maximum = 100
- MAIN EVOLUTION
    - Population size = 525
    - Number of generations = 16
    - F(DE) = 0.9
    - CR(DE) = 0.5
    - PathLength(SOMA) = 4.f
    - Step(SOMA) = 0.21f
    - PRT(SOMA) = 0.1f
- META EVOLUTION:
    - Algorithm type = SOMA
    - Population size = 10
    - Number of generations = 10
    - PathLength(SOMA) = 3.f
    - Step(SOMA) = .25f
    - PRT(SOMA) = .1f

[8] http://www.astrosurf.com/buil/us/becat.htm

	DE	SOMA
Minimum	1,908560	0,303576
Average	3,404346	0,592249
Maximum	5,902490	1,047080

Tab. 4: Fitness values of synthetized functions from analytic programming

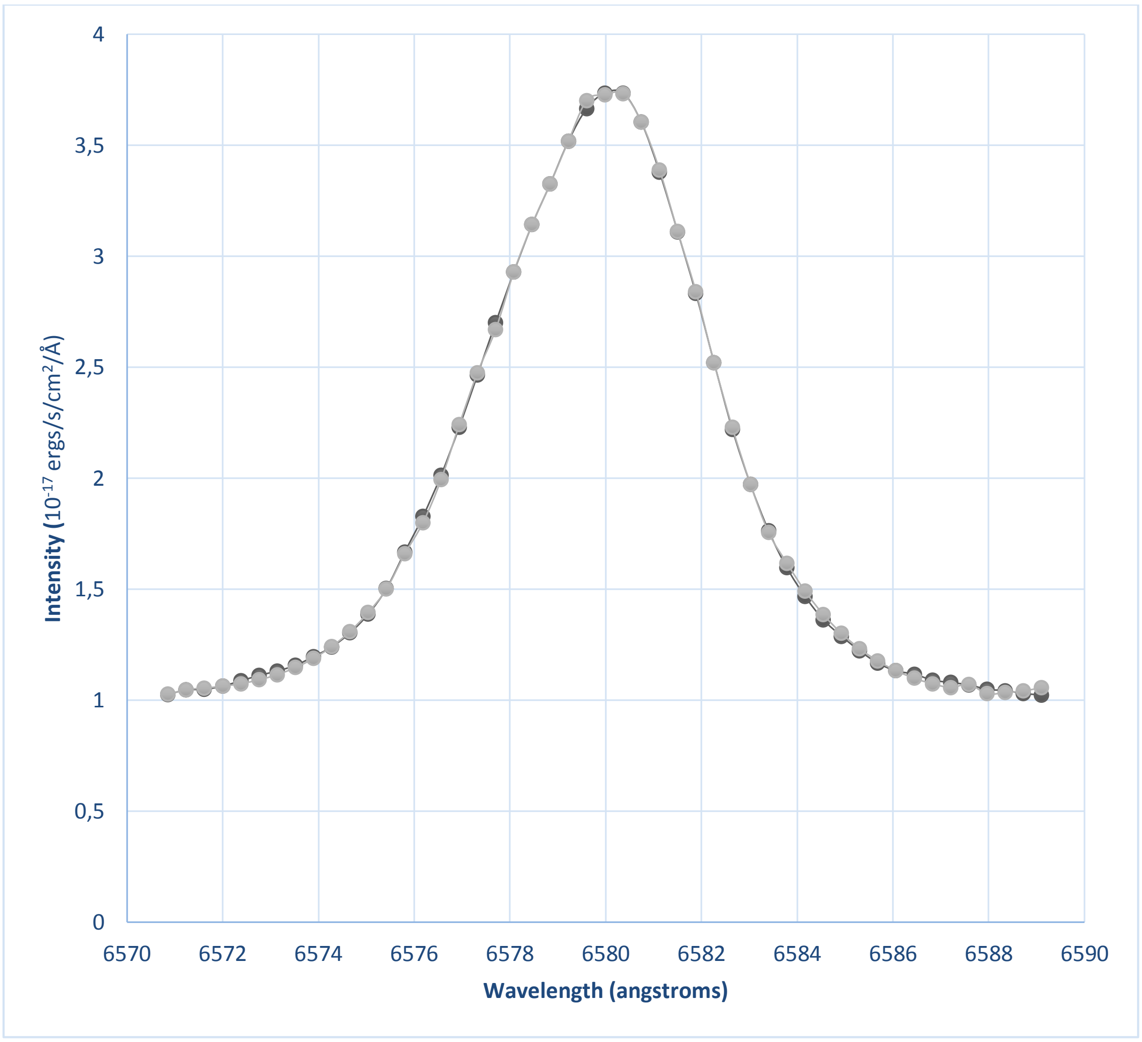


Fig. 19: Comparison between model spectra and specter created by analytic programming.

In Table 4, you can see fitness values of output functions from both differential evolution and SOMA. My implementation of SOMA is better than my implementation of differential evolution in minimum, average and maximum fitness values. Differences between minimum and maximum fitness is quite big, which is caused by random in algorithm and it is better to perform more synthesis to find best possible result.

As you can see in Figure 19, generated function is almost same as model function. Precision of result depends on parameters of analytic programming. Further increasing of precision is possible by increasing number of generations and population size for both normal and meta evolution however this requires more time to synthetize output function as well as it makes output expression grows very rapidly.

5.4 Spectra comparison results

Main point of this part of application is achieve speedup of parallel version over serial version. Tests has been performed with data mentioned in Chapter 4.3.

First test has been performed with serial and parallel types of application on different amount of data. As you can see from Table 5, time required for executing algorithm grows linearly with amount of data.

Size of data (MB)	Time of serial version (seconds)	Time of parallel version (seconds)
46,5 MB	7.5	1.0
465.0 MB	74.0	10.8
4650.0 MB	730.0	108.4

Tab. 5: Times of spectra comparison on different amounts of data

Second test has been focused on scalability of my application. In Table 6 you can see results of this test. Table shows, that scalability is not linear and it wane rapidly with more cores. With 2 core, algorithm runs almost two times faster as with one core, but if you use more than 12 cores, time required to compare data remains same. This wane is also demonstrated in Figure 20. Reasons for this wane are explained in Chapter 2.3.5.

Number of cores	Scalability	Time (s)
1	1.000000	74.0
2	0.994624	37.2
3	0.913580	27.0
4	0.872642	21.2
5	0.840909	17.6
6	0.790598	15.6
8	0.734127	12.6
10	0.685185	10.8
12	0.616667	10.0
14	0.550595	9.6
16	0.462500	10.0
20	0.377551	9.8
40	0.188776	9.8
60	0.124579	9.9
84	0.089893	9.8

Tab. 6: Dependency of scalability and time on number of cores

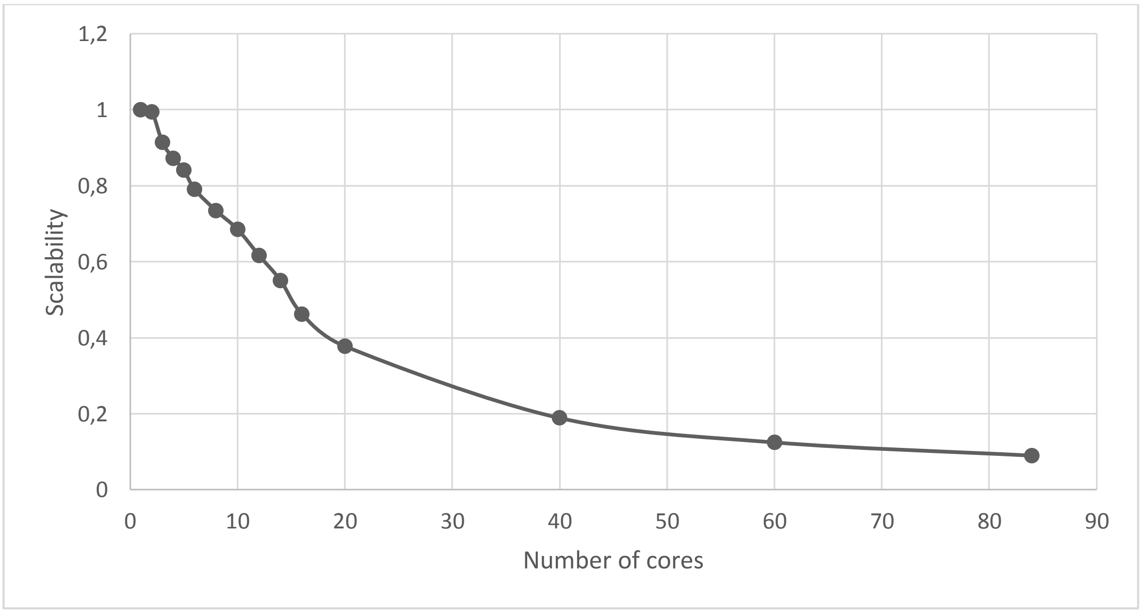


Fig. 20: Scalability of parallel version of application

6 Conclusion

In theoretical part of this thesis I made short introduction to astronomy background. After that, I made a survey of parallel architectures and its possibilities, so I fulfilled first and second point of thesis assignment.

In practical part of this thesis I designed and implemented analytic programming, a machine learning algorithm, which allowed me to synthetize model spectra and create a mathematical expression of it. I also made a parallel application, which compare synthetized spectra from analytic programming with spectra of unknown space objects, thereby I fulfilled point number 3 and 4 of thesis assignment.

Results of first experiment show us, than analytic programming is very suitable algorithm to synthetizing mathematical expression from measurements. Fitness values retrieve by my implementation are very good, however they could be much better in future. Promising way, how to achieve this is making algorithm faster, which allow algorithm to evolve for more generations or with bigger population. Possible solution of this could be parallelization of algorithm.

Second experiment show us, how it is possible to use multicore architecture to increase performance of application, which compare 2 spectra. Result show us, than real challenge in parallelization is not obtaining huge amount of computing resources, but adapting algorithm structure to parallel nature and finding architecture limits. Important factor, which should we concern about, is effectivity of used computing resources, because investments and energy consumption does not have to worth real performance. In my application and architecture I assume processor with 4 or 8 cores would be still very effective.

Greatest benefit of this thesis for me is insight to field of parallel computing as well as insight to field of evolution algorithms, which I deem to be very interesting and potential.

Literature

1. BLAISE, Barney. *Introduction to Parallel Computing* [online]. [cit. 2014-01-23]. URL:<https://computing.llnl.gov/tutorials/parallel_comp>
2. PACHECO, Peter S.. *An Introduction to Parallel Programming.* 1. edition. Burlington: Morgan Kaufmann Publishers, 2011. 370 p. ISBN 978-0-12-374260-5.
3. ROUSE, Margaret. *Definition: multi-core processor* [online]. [cit. 2014-01-28]. URL:<http://searchdatacenter.techtarget.com/definition/multi-core-processor>
4. APACHE. *Hadoop* [online]. [cit. 2014-01-29]. URL:<http://hadoop.apache.org/>
5. MORROW, Rich. *Learning how to learn Hadoop* [online]. [cit. 2014-01-29]. URL:<http://www.globalknowledge.com/training/generic.asp?pageid=3438>
6. NVIDIA. *GPU computing* [online]. [cit. 2014-01-29]. URL:<http://www.nvidia.com/object/tesla-supercomputing-solutions.html>
7. INTEL. *Intel® Many Integrated Core Architecture* [online]. [cit. 2014-01-30]. URL:<http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html>
8. WITTEN, Ian H. *Data mining: practical machine learning tools and techniques*. Co-author Eibe Frank. 2. edition. San Francisco: Morgan Kaufmann Publishers, 2005. 525 p. ISBN: 0-12-088407-0.
9. RAJARAMAN, Anand. *Mining of Massive Datasets*. Co-author Ullman Jeffrey David. 1. edition. Cambridge: Cambidge University Press, 2012. 326 p. ISBN: 978-1-107-01535-7.
10. SUBHASHINI, Venugopalan. *Data mining techniques to classify astronomy objects* [online]. [cit. 2014-06-19]. URL:<http://www.cs.utexas.edu/~vsub/pdf/DM-astro.pdf>
11. CRAIG, Andrew. *Astronomers count the stars* [online]. [cit. 2014-06-24]. URL:<http://news.bbc.co.uk/2/hi/science/nature/3085885.stm>
12. SOBCZAK, Drew. *Stellar Classification* [online]. [cit. 2014-06-24]. URL:<http://quarknet.fnal.gov/fnal-uc/quarknet-summer-research/QNET2010/Astronomy/downloads/Stellar_Classification_Intro.pdf>
13. BLESSON Mathew. *Study of emission line stars in young open clusters.* Bangalore, 2010, 240 p. A thesis submitted for the degree of Doctor of Philosophy in The Faculty of Science University of Calicut, Calicut. Thesis Supervisor Dr. Annapurni Subramaniam.
14. ZELINKA, Ivan. *Evoluční výpočetní techniky - principy a aplikace.* Co-author Oplatková Zuzana, Šeda Miloš, Ošmera Pavel, Včelař František. 1. edition. Prague: BEN, 2009. 536 p. ISBN 80-7300-218-3.

Attachments on CD

1. User manual to application
2. Electronic form of thesis
3. Source codes of analytic programming (directory Analytic_programming)
4. Source codes of spectra comparer (directory Spectra_comparer)
5. Scripts for retrieving metadata from source (directory Scripts)