Modelling of multicolour light source using distributed computing network. 1. Statement of the physical and computational problems

Aistė Ambrazevičiūtė, Rimantas Vaicekauskas [a]
Department of Informatics, Faculty of Mathematics and Informatics
Vilnius University, Naugarduko 24, Vilnius, Lithuania

Received 29 January 2011, accepted 20 February 2011

Abstract. Light-emitting diodes (LEDs) are widely applied in conventional lighting. We investigate white light sources made of three primary coloured LEDs with respect to colour rendition ability. The spectral power distribution of LED is simulated by a function of a particular shape that depends on wavelength and intensity values, therefore searching for the optimal characteristics of compound light source becomes a computational problem. The aim of this work is to create an efficient parallel algorithm allowing us to find a trichromatic light source that minimizes colour distortions.

Citations: Aistė Ambrazevičiūtė, Rimantas Vaicekauskas. Modelling of multicolour light source using distributed computing network. 1. Statement of the physical and computational problems – *Innovative Infotechnologies for Science, Business and Education,* ISSN 2029-1035 – 1(10)(2011) 13-20.

Keywords: LED lighting; Colour rendering; Global optimization; Distributed computing; Message passing interface; MPI.
PACS: 07.05.Tp; 42.66.Ne; 02.60.Pn.
Short title: Modelling of light source - 1.

Introduction

Solid-state semiconductors *Light Emmiting Diodes* (LEDs) are compact and very energy-efficient sources: luminous efficency exceeds 200 lm/W, emission duration exceeds up to 100000 hours [1]. LED's are non-toxic and natural wear-resistant. Furthermore, they emit light in a wide variety of spectra - IR, visible or UV depending on their composition. LEDs operating in visible spectrum emit close-to-monochromatic light, so, by combining several of them and mixing their emission, it is possible to obtain the light emission of any colour and intensity. These features allow creation of high quality and efficiency lighting that is nature friendly as well.

In our case, the aim is to compound a white light source (sometimes called the *RGB* source) whose distortion of illuminated surface colours would be as low as possible. As the measure of the ability of light source to accurately reproduce colours of illuminated surface we will use general *Colour Rendering Index* (CRI). According to CRI calculation procedure briefly described in Ref. [2], colour shifts for 8 standard samples are taken into consideration by illuminating the samples with a reference light source and the test light source. CRI is defined as an average of eight particular colour rendering indices. Higher CRI value means better colour rendition (i.e., lower colour distortion). The light source identical to reference source has the highest CRI value 100.

This work is aimed:

i) to develope the mathematical model which describes spectral distribution of the light source consisting of several coloured LEDs and
ii) to relate physical characteristics of the light to the psychophysical colour related quantities such as CRI.

The searching problem can be defined as mathematical optimization problem, which takes many input variables. To quickly find optimal solution, advantages of parallel computations using the programming tools like *Message Passing Interface* (MPI) can be taken into account.

It should be noted that particular approximate solutions (CRI-optimal RGB LED systems) are already known [3]. For example, RGB system with selected wavelengths of LEDs near to 460 nm, 540 nm and 610 nm may have CRI value close to 90 [3]. However, in contrast to trial-error or randomized optimization techniques used elsewhere, we apply the *branch and bound algorithm* (assuming the objective function is the Lipschitz) that guarantees to find a global optimum with a desired accuracy. We will also discuss related computational problems and their solutions.

[a]Corresponding author, email: *Rimantas.Vaicekauskas@mif.vu.lt*

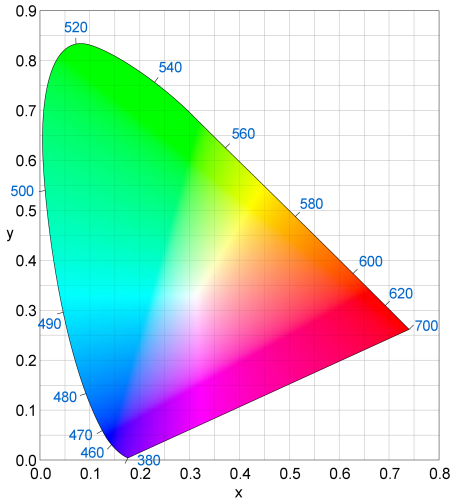

Fig. 1. CIE 1931 colour space chromaticity diagram. Adapted according to Ref. [4].

1. Relationship between light and colour

1.1. Colour mixing

Consider we have a set of LEDs emmiting in visible region with the SPD $S(\lambda, \lambda_0)$, where $\lambda \in$ [360÷830] nm is a wavelength and $\lambda_0 \in$ [360÷830] nm is a main LED parameter - peak wavelength.

Let's select three λ_0 values, in other words, three LEDs emitting the radiation of the following (central) wavelengths: λ_{01}, λ_{02}, λ_{03}. The resulting source of light after combining all three spectra is noted as the following equation where c_1, c_2, $c_3 \geq 0$ are coefficients describing the ratio between separate LEDs intensities.

$$S(\lambda) = c_1 S(\lambda, \lambda_{01}) + c_2 S(\lambda, \lambda_{02}) + c_3 S(\lambda, \lambda_{03}) \quad (1)$$

Our task is to find such combination of LEDs described by parameters λ_{01},λ_{02}, λ_{03} for a specific white light source (for example, standard D_{65} illuminant) that the CRI would be maximized. In other words, optimization problem has to be solved. The following conditions (constraints) should be satisfied: i) chromaticity of the reference source has to be exactly the same as the chromaticity of the resulting source; and ii) luminuous flux have to coincide.

Since every λ_{01},λ_{02}, λ_{03} may have a value from interval [360÷830] nm, the search space is a cube, intensities c_1, c_2, c_3 are determined by solving a colour mixing equation for three partial sources [3].

1.2. Colorimetry

In order to productively mix the emmision of LEDs of different colours, we need some basics knowledge about colours and their perception.

The reason why colours exist is purely based on a physical phenomenon known as visible spectrum electromagnetic radiation, although the perception of colours is only physiological and psychological phenomena. It is based on three types of cone cells that have different spectrum sensitivity, excitation and processing of information in our brains [5]. Therefore, the assumption was made, that the three primary colours - red, green, blue - are sufficient to create any colour.

1.3. Colour definition

Colour perception represents a research subject in the field of biology, psychophysics and photochemistry. Mathematically defined colour space as digital colours can be characterized by chromaticity coordinates (x, y), created in 1931 by the International Commission on Illumination (CIE), so called *CIE 1931 XYZ colour space* [6], [7]. Chromaticity coordinates (x, y) of a source consisting of n sources that are defined by power distributions $S_i(\lambda)$, are calculated from tristimulus values X_i, Y_i, Z_i where $a(\lambda)$, $b(\lambda)$, $c(\lambda)$ represent the *colour matching functions*, and the integration interval is presented in range [360÷830] nm.

$$X_i = \int a(\lambda) S_i(\lambda) d\lambda \quad (2)$$

$$Y_i = \int b(\lambda) S_i(\lambda) d\lambda \quad (3)$$

$$Z_i = \int c(\lambda) S_i(\lambda) d\lambda \quad (4)$$

$$x = \frac{\sum_{i=1}^{n} X_i}{\sum_{i=1}^{n} (X_i + Y_i + Z_i)} \quad (5)$$

$$y = \frac{\sum_{i=1}^{n} Y_i}{\sum_{i=1}^{n} (X_i + Y_i + Z_i)} \quad (6)$$

x and y represent the chromaticity coordinates of compound lighting source. Fig. 1 shows the CIE 1931 colour space chromaticity diagram. All visible colours are bound by a horseshoe-shaped figure. In case of our problem, it is worth mentioning that in CIE 1931 colorimetric system differences between colours are not equal - the same distance at different coordinates cause different change in colour perception.

To evaluate colour differences we transform colours into uniform colour spaces, for example, *CIE 1960* and *CIE 1976* [8]. The standard technique is CIE 1964 [5]. In order to calculate CRI, the so called CIE 1960 colour space (CIE UCS) is used as an auxiliary colour space. The colour coordinates (u, v) of CIE UCS are defined as a transformation:

$$u = \frac{4x}{12y - 2x + 3} \quad (7)$$

$$v = \frac{6y}{12y - 2x + 3} \quad (8)$$

The colour rendering procedure, described in CIE 1995 [2], is based on eight standard and six extra colour samples. The difference between colour when illuminated by a reference light source (it is chosen to have same colour temperature as the test light source) and test light source is calculated for every sample. Based on these rendering numbers, CRI is

calculated. The average value of CRI on 8 standard colour samples is called a general CRI, which equals 100 points if test light source does not make any difference in colour. This system uses colour samples from A. H. Munsell system - see Table 1.

Munsel system represents behaviour of sample colour, where colour is defined by hue, value and chroma. Index 7,5R6/4 7,5 describes the sample: hue is red, value is 6 and chroma is 4 [5].

Ref. [3] presents a research of solid state semiconductor emitters consisting of several LEDs: two, three or more, when LEDs of different colour are used. As it was shown, the required number of primary LEDs depends on the efficiency and general CRI parameters.

In particularly, light source of decent efficiency and colour rendering characteristics can be made of $3 \div 4$ primary LEDs. Light source with 5 or more LEDs yields negligible benefit in improving colour rendering, however, its spectrum is almost continuous and that may be used in order to get a specific lighting.

Table 1. Munsell colour indexes.
1-8 for colour rendering. [4]

Sample N	Munsell label	Colour at daylight
1	7,5R6/4	Light greyish red
2	5Y6/4	Dark greyish yellow
3	5GY6/8	Strong yellow green
4	2,5G6/6	Moderate yellowish green
5	10BG6/4	Light bluish green
6	5PB6/8	Light blue
7	2,5P6/8	Light violet
8	10P6/8	Light reddish purple
9	4,5R4/13	Strong red
10	5Y8/10	Strong yellow
11	4,5G5/8	Strong green
12	3PB3/11	Strong blue
13	5YR8/4	Light yellowish pink (skin)
14	5GY4/4	Moderate olive green (leaf)

2. Computational aspects

2.1. Review of algorithms

Various optimization methods and techniques may be applied in our case. The main focus is on *branch and bound optimization* method. We take into account that our objective function meets the Lipschitz condition: the change in CRI is linearly bound by change of parameters. This method enables us to find regions without a global minimum or maximum and ignore them.

The feasible region is gradually divided to smaller subregions (branching), evaluated separately by using the value of Lipschitz constant. This subregion is further partitioned if optimal solution may be found in it (or discarded otherwise).

In the final step, by reviewing the remaining region, the lowest or biggest value of object function is the global minimum or maximum, respectivelly, with error bounds defined before the computation.

In order to use the branch and bound algorithm effectively, several partitioning strategies were analysed as well as the point selection in a region, that would be analysed most efficiently. A method was created allowing prediction whether there are any solutions in a segment, when the solution in investigated point is not found. Trying to keep the computation time of problem solving as low as possible, a new objective - parallelization of computational algorithms was taken into consideration. By analysing branch and bound algorithm parallelization methods, a new method was proposed and its efficiency in respect to a speedup was experimentally confirmed.

Parallel branch and bound algorithm for solving optimization problem was designed and implemented using the message passing interface library (MPI). MPI ensures effective and platform-independent message passing, so MPI programs written in C, C++ or Fortran, may be moved from one PC to another without any hassle.

Simple scheme of centralized data exchange was chosen for implementation of parallel algorithm. It is based on *master-slave* communicative structure, where only the master node communicates with slave nodes. In case of intensive data exchange, communication network may become overloaded. In that case, centralized scheme may become ineffective and modification of the scheme by decentralizing the communication might be taken into consideration.

Several modifications of the parallel algorithm were implemented and analysed in order to find the optimal data exchange rate. In every case the workload was distributed dynamically. The only difference was how often a process communicates with a master process. It was observed, that new candidates to the optimal values are found rarely during the optimization problem solving. A more common appearance of candidates was observed only in several segments of search interval. Conclusion was made that *master-slave* communicational scheme is sufficient for this problem.

2.2. Hardware

The algorithms were tested on a SGI Altix 4700 supercomputer, installed at faculty of Mathematics and Informatics, Vilnius University. SGI distributed computations network is made of 16 blocks, each consisting of 2 dual-core Intel Itanium 2 model 9020 (Montecito) 1.4 GHz CPUs - 64 cores in total. Algorithms were evaluated by speed and efficiency characteristics. Speed factor indicates the increase in computational speed gained by using a parallel algorithm [10]. Efficiency is based on the workload of cores used.

2.3. Evaluation of colour rendering

Procedure of the evaluation of colour rendering is described in Ref. [2]. According to it, the actions needed to take in order to calculate CRI of our sample light source are as follows.

1. Spectral distribution. Spectral power distribution of the sample $S_k(\lambda)$ is found and by using it, coordinates of colour (u_k, v_k) in CIE UCS space may be found, as well as the correlated colour temperature if it is needed.

2. Normalization. Spectral distribution of power $S_k(\lambda)$ is normalized in such way, that luminance Y_k = 100.

3. Light source. Reference light source with power distribution of $S_r(\lambda)$, colour coordinates (u_r, v_r), emission flux normalized in a way that luminance Y_r = 100 is selected according to the correlated colour temperature of the tested light source.

4. Reflected spectra. Power distributions of reflected light from colour samples $S_k(\lambda)\rho_i(\lambda)$ and $S_r(\lambda)\rho_i(\lambda)$ are calculated for both light sources: tested and reference. Coordinates of luminance Y_{ki}, Y_{ri} and colour (u_{ki}, v_{ki}) and (u_{ri}, v_{ri}) are calculated for both spectra.

5. Transformation. Transformation of colour coordinates (u_{ki}, v_{ki}) is applied using the following scheme.

$$u^*_{ki} = \frac{10.872 + 0.404 \cdot C_q - 4 \cdot D_q}{16.518 + 1.481 \cdot C_q - D_q}; \quad (9)$$

$$v^*_{ki} = \frac{5.20}{16.518 + 1.481 \cdot C_q - D_q}; \quad (10)$$

$$C_q = \frac{c_r \cdot c_{ki}}{c_k}; \quad (11)$$

$$D_q = \frac{d_r \cdot d_{ki}}{d_k}; \quad (12)$$

$$c = \frac{4 - u - 10 \cdot v}{v}; \quad (13)$$

$$d = \frac{1.708 \cdot v + 0.404 - 1.481 \cdot u}{v}. \quad (14)$$

6. Differences in luminance and colour. Differences in luminance and colour are calculated in a CIE 1964 (W*U*V*) colour space for every sample ΔW, ΔU, ΔV by expression G:

$$\Delta W^*_l = 25 \cdot [\sqrt[3]{Y_{ki}} - \sqrt[3]{Y_{ki}}] \quad (15)$$

$$\Delta U^*_l = 13 \cdot [G \cdot (u^*_{ki} - u_r) - G \cdot (u^*_{ri} - u_r)] \quad (16)$$

$$\Delta V^*_l = 13 \cdot [G \cdot (v^*_{ki} - v_r) - G \cdot (v^*_{ri} - v_r)] \quad (17)$$

$$G = 25 \cdot \sqrt[3]{Y_{ki}} - 17 \quad (18)$$

7. Differencies for every sample. Difference in colour is calculated for every sample:

$$\Delta E_i = \sqrt{(\Delta U^*_l)^2 + (\Delta V^*_l)^2 + (\Delta W^*_l)^2} \quad (19)$$

8. Test. CRI is evaluated for every test sample:

$$R_i = 100 - 4.6 \cdot \Delta E_i \quad (20)$$

9. Averaging. Average CRI is calculated in order to find the general CRI.

2.4. Optimization tasks

Usually, minimum or maximum value of object function has to be found in order to find a solution of the optimization problem. If the derivative of object function is unknown, other information about the function may be used [9]. Random search methods may be applied too.

Search of global minimum. The search for global minimum is described in Ref. [9]. Local *hill climbing method* may be applied in search for global minimum, because, by default, it is a local minimum too. The initial point has to be a part of the area of attraction of global minimum. This point is often found by random prediction - if points are generated by a continuous distribution in a permitted area, the probability p of one random guess being in the attraction area of the global minimum is presented as equation where G and V are hyperspace of the attraction area and permitted area of the global minimum, respectively. Probability p_n represents situation that 1 out of n attempts will be in this area.

$$p = \frac{G}{V} \quad (21)$$

$$p_n = 1 - (1 - p)^n \quad (22)$$

Another technique of searching for global minimum is called *random local minimum enumeration*. Local minimization is usually performed in search method instead of hill climbing. It is sufficient to have a local minimization method combined with the generation of initial points in order to implement the search method. This technique is valid only if the function has just a couple of local minimums with similar-sized attraction areas. Otherwise, the result of hill climbing can appear to be just a point of local minimum, especially if it has a wide attraction area.

Cluster search method. The general idea of this method is to detect clusters of points on the descending curves. In order to do that, a set of initial points is generated. Local descend computation is started after few steps in order to move the points from their initial state. The resulting points are analysed and if a cluster of points is found, we can conclude that several descending processes move towards the same minimum, a point with the best value of object function is selected. Afterwards, local descend is continued for this point, while all remaining processes are cancelled. After the analysis of clustering of points, only a specific number of processes may be cancelled and a new set of points is generated. The process is repeated until several analysis in a row do not locate any new point clusters. This method is pretty effective, since local descends to a minimum are located quite rapidly and every but one processes are cancelled. In order to avoid unnecessary local minimums, search method instead of hill climbing is applied. As specified in the Ref. [9], experimental comparison between this and other methods proved high efficiency of this method.

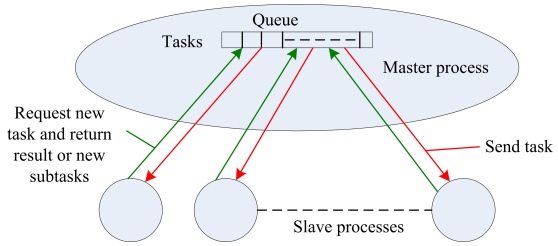

Fig. 2. Scheme of creation of centralized dynamic schedule. Adapted according to Ref. [12].

Branch and bound method. A classical approach for optimization problems is *branch and bound method*. It concentrates around looking for subintervals that do not contain the global maximum or minimum and cancellation of them.

This results in the permanent division of initial search range. While analysing every part separately, it is decided whether optimal solution exists in it or not. If it does, the top or bottom boundary is extended and the range is divided further, otherwise it is discarded. The efficiency of branch and bound method relies on the initial division algorithm. Several templates of these algorithms exist, their purpose being simplification of implementation of a specific branch and bound algorithm. Examples of such templates are as follows: BOB, BOB++, PICO, PPBB, PPBBB [10].

One of parallel implementations of branch and bound method was described in Ref. [10]. MPI libraries were used to exchange data. The main feature of this template is the division of search ranges across processing cores.

The amount of these ranges may be equal to, or higher than the number of cores. Every processor checks its range whether optimum solution exists in that range. Processors may communicate and share their bounds by applying data exchange protocols. Also this template is supplemented to a load balance feature, enabling reallocation of tasks during the execution in order to keep the workload of system balanced. Tools enabling use of Lipschitz criteria are implemented too.

3. Parallel computation

Parallel computations help reduce time it takes to solve a problem, although, if it is applied incorrectly, it may take much longer. In order to succesfully apply parallelization, such routines must be constructed according to advices in Ref. [11].

1. Dividing the complex task to smaller independent tasks.
2. What task size should be chosen?
3. How many nodes should be used for fastest run?
4. How to assign tasks between nodes?

The specificity of the problem as well as data exchange between nodes for parallel computation needs to be taken into consideration before dividing the problem to separate tasks.

The cost of message passing between nodes has to be discussed. Nodes pass messages in order to exchange data in parallel computations. If a message constitutes of n bytes, the cost of that message passing may be defined as:

$$T = \alpha + \beta n. \tag{23}$$

where α is a sum of the preparation of message (setting up contents of message, assigning overhead, flags, find out the optimal route in a network) and the duration of message travelling through the media. β is the time of transferring one byte of data [10]. In order to reduce the cost of message passing, the messages should be grouped together to keep the ratio ξ as low as possible.

$$\xi = \frac{\alpha}{\beta n} \to 0 \tag{24}$$

3.1. Creation of a schedule

Process of scheduling constitutes of assigning nodes to specific tasks. Schedule is defined by vector S:

$$S = (s_0, s_1, ..., s_{p-1},), \tag{25}$$

where s_j is a set assigned to j-th node. Schedule is titled as correct in case if all following conditions are satisfied [11].

1. Every task is assigned to only one set s_j.
2. Execution time for every task is later than execution time for any predecessor task.

According to the division of tasks, these techniques may be presented for scheduling.

3. Static scheduling.
4. Dynamic scheduling using centralized or decentralized manager.

Static scheduling. Static scheduling is the simpliest technique of scheduling. It may be used only in case when all tasks are known. Using this technique, problem is divided into smaller tasks and every task is assigned to different processor. The work is done, when every core returns its result, although the drawback of this technique is the case when some tasks take longer to complete and a part of nodes will be idle until every single one will finish the computation.

Dynamic scheduling. Dynamic scheduling allows to assign tasks to nodes in a optimized way [12]. Also, it enables creation of schedule without knowing every possible task. Sometimes they appear as a result of computing another task (see Figs. 2-3). Centralized dynamic scheduling has a master node that assigns tasks.

The tasks are sorted according to their "weight" and sent to slave nodes in small groups. Upon completion, a node sends back the result to a master and asks for another task. The process of scheduling end together with the solution - when queue is empty and every node has returned its result.

This technique is applicable when tasks are *heavy* and the amount of nodes is moderate. Decentralized schedule is a modified version of aforementioned dynamic schedule. It is used when the amount of nodes is large and they request a new task more often.

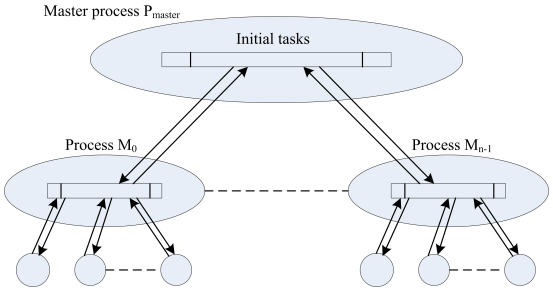

Fig. 3. Management scheme of decentralized dynamic schedule. Adapted according to Ref. [10].

In order to cope with that, master process initiates several management processes. Such management processes receive tasks and then distribute them independently, receiving and storing results in the meantime. Afterwards, every management process returns their results back to master process. After the master process finishes processing of all results, the scheduling is over [12]. When using the decentralized dynamic schedule, an extra task is to choose the right amount of management processes.

3.2. Optimization of schedule

Task of optimal scheduling. The point of optimal scheduling is to achieve the fastest possible execution of specific algorithm. In order to create the optimal schedule, many points have to be taken into consideration:

i) the problem itself;
ii) the cost of communication between nodes;
iii) the difficulty of separate tasks;
iv) the amount and characteristics of nodes.

Also, it has to be noted, that workload and characteristics of nodes may change in time. The optimization of schedule is NP-complete and in order to save time, heuristics are applied resulting with a decent schedule. The examples of scheduling algorithms are as follows:

i) algorithms of list division;
ii) critical path algorithms;
iii) graph analysis heuristics;
iv) Monte Carlo method and its modifications;
v) adaptive algorithms;
vi) genetic algorithms;
vii) combined task distribution algorithms.

The main idea of list division algorithm is as follows: the list of tasks is arranged and every free node picks up a task that has not started yet. If there are no such tasks, the node in question remains idle until a task appears. Critical path algorithms try to determine tasks that would have direct impact on time of finding a solution for whole problem. The main idea of Monte Carlo algorithm is to make computations with only a small set of randomly chosen data, for example, nodes are arranged by their computation rate and tasks are assigned to them with corresponding difficulty [12], ignoring the fact that workload on a node is a dynamic characteristic. Various modifications of Monte Carlo method exists [13].

Dynamic scheduling. A method of scheduling depending on cost of communication, cost of task completion, relations between tasks and computer characteristics is presented in a Ref. [14].

The main idea on the scheduling presented in the reference is the minimization of cost and maintaining balance between computer workload on a system. One of propositions is to arrange grouping of tasks by intensity of communication between them and make sure that similar tasks communicate only between themselves in the same group. The grouped tasks should be assigned to nodes for execution before any other tasks.

Nodes have to be arranged by their workload before having a task assigned to them. The workload levels are as follows: low workload, moderate workload and overload. When the workload levels are known, tasks can be assigned to them according to "weight" of the task and the workload of a processor to keep system balanced. In addition to this method strictly defined conditions such as priorities, resource availability, real time characteristics and dynamic resource distribution on grid are implemented. Virtual jobs are created by grouping tasks using this technique. Every of them constitute the queue of jobs that should communicate between each other. Because they are assigned to the same resource if possible, the communication cost between them is reduced as much as possible.

3.3. Parallelization of algorithm

Parallel algorithm can be implemented depending on memory type used: either shared or distributed memory. Threads have a common variable allowing them to watch each others progress and results in case of the shared memory. In order to see that in case of distributed memory, threads have to pass messages to each other. The cost of them has already been reviewed. In order to maximize the efficiency of problem solving, optimal message passing scenario has to be chosen, including the occasion and reason why the messages need to be passed, because communication cost is quite significant.

***Master-slave* communication.** *Master-slave* structure defines a type of communication between processes or threads when one process or thread is responsible to initial assignation of jobs for slaves and retrieval of their results. There are two base variations of this algorithm: the amount of slaves can be static or dynamic. This work presents *master-slave* communicational structure, evaluates it and then the decision whether there is a need to decentralize it is made.

Case of shared memory. In case of shared memory *master-slave* communication, one thread, known as master, starts and then launches slave threads and equally assigns tasks to them. After that, master thread waits at a point synchronized by every slave until it receives every result from slave threads.

Table 2. Main operations of semaphores.

Operation	Value	Description
S_p	false	to block on s
S_p	true	$s = s - 1$
S_v	true	to unblock on s
S_v	false	$s = s + 1$

Then it ends its work or repeats the process recursively.

Synchronization of two data elements access needs to be taken care of in shared memory case. First element is a tree. In its leaves the search range is fixed and master assigns a part of general search range that is in a leaf to a slave. The second element describes the best solution at a given moment.

Synchronization needs to be ensured in case if: i) the best solution changes; ii) processes associated with task allocation take place. Semaphores may be used for synchronization. Their operations are presented in Table 2. We have used the described technique for semaphore manipulation. For example, three semaphores - $NumberOfTasks$, $ksProtection$, $ksProtection2$ are presented in Table 3.

Case of distributed memory. In that case one process is responsible for task assignation to the other processes in *master-slave* communication scheme. Master process:

i) prepares initial list of tasks;
ii) while optimal solution is not found, accepts requests for new task, operation *receive* is performed;
iii) while optimal solution is not found, returns task or end sign, operation *send* is performed;

Slave process:

i) requests for a new task, operation *send* is performed;
ii) receives a new task and current maximum or an end sign, operation *receive* is performed;
iii) performs reduction of search range operation;
iv) performs division operation;
v) forms new tasks;
vi) forms new maximum value;
vii) transmits max value and new tasks, operation *send* is performed;

Table 3. Additional operations of semaphores.

Semaphore	Function or operation for semaphore
	Retrieve jobs
$NumberOfTasks$	P()
$csProtection$	P()
	Assigned job
$csProtection$	V()
	Insert job
$csProtection2$	P()
if (best)=true	renew_results
$csProtection2$	V()
	Insert job
$NumberOfTasks$	V()

Evaluation of parallel algorithm. Time needed for the parallel algorithm to complete problem solving depends on many indicators, such as specifics of a problem, partitioning of a problem to smaller tasks and the amount of these tasks, amount of processor cores, technique of task allocation between processors, computer characteristics, computer workload at given moment. Several criteria are used to evaluate the parallel algorithms. One of them is a speedup factor S_p:

$$S_p = \frac{T^*}{T_p}. \tag{26}$$

The speedup factor indicates the speed of parallel algorithm using p processors where T^* is time required to complete an assignment using the best known serial algorithm.

T_p is titled as the time to complete same assignment using parallel algorithm on p nodes [10, 13]. Parallel algorithm may also be evaluated by processor workload usage efficiency E_p, where p is amount of processor cores [15]. It is known that the efficiency of parallel algorithms decreases by increasing the amount of processor cores.

$$E_p = \frac{S_p}{p} \tag{27}$$

This characteristic of an algorithm is known as *scalability* and it defines the required increase in a problem by increasing the amount of nodes used. Scalability of an algorithm is better, when the required increase in a problem size is as small as possible in order to keep the computations effective by adding more processor cores.

MPI. *Message-Passing Interface* (MPI) is a *de facto* standart for implementation of message passing in parallel algorithms. The purpose of MPI is to define a flexible, effective and platform-independent message passing in distributed computing environment. The computer programm using this standard may be moved from one system to another without any problems [16]. MPI defines only function names, parameters and purpose, but does not describe implementation. Commercial, non-commercial and manufacturer-bound MPI versions exist. Computing cluster uses non-commercial MPI version called LAM/MPI.

Conclusions

A mathematical model has been built in order to find out a light source made of several multi-colour LEDs with adjustable wavelengths having the highest general colour rendering index. The methods for a optimization problems were discussed. In order to reduce the search span, branch and bound optimization method was introduced. The usage of MPI in parallelization of optimization algorithm was investigated. Balancing of CPU load is required to have an effective parallel algorithm. This is achieved by solving computation queueing problem. The effectiveness and scalability of the parallel algorithm is the main goal. The solution will be checked on VU MIF distributed computation cluster.

References

1. J. Y. Tsao, H. D. Saunders, J. R. Creighton, M. E. Coltrin, and J. A. Simmons. Solid-state lighting: an energy-economics perspective – *J. Phys.* D43 (2010) 354001.
2. Method of measuring and specifying color rendering properties of light sources – *CIE Publication* 13.3 (1995).
3. A. Žukauskas, R. Vaicekauskas, F. Ivanauskas, R. Gaska, and M. S. Shur. Optimization of white polychromatic semiconductor lamps – *Appl. Phys. Lett.* 80(2) (2002) 234-236.
4. CIE 1931 color space. <http://en.wikipedia.org/wiki/CIE_1931>, retrieved 2010 05 10.
5. Wyszecki, G. and Stiles, W. S. Book. Color Science: Concepts and Methods, Quantitative Data and Formulae – New York:Wiley, 2000.
6. Commission internationale de l'Eclairage proceedings. – Cambridge: Cambridge University Press, 1931.
7. Smith Thomas, Guild John . The C.I.E. colorimetric standards and their use – *Transactions of the Optical Society* 33(3)(1931–1932) 73–134.
8. Yoshi Ohno. CIE Fundamentals for Color Measurements. – Gaithersburg, USA, Optical Technology Division National Institute of Standards and Technology (2000).
9. A. Žilinskas. Matematinis programavimas (in lith.) – Kaunas : *VDU leidykla*, 2000.
10. B. Wilkinson, M. Allen. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers – USA (2005).
11. M. Baravykaitė, R. Čiegis. An implementation of a parallel generalized branch and bound template. – *Mathematical modelling and analysis: the Baltic journal on mathematical applications, numerical analysis and differential equations*. ISSN 1392-6292 – 12(3) (2007). p. 277-289.
12. R. Čiegis. Lygiagretieji algoritmai (in lith.) - Vilnius :*Technika*, 2001.
13. S. Chantaravarapan, A. Gunal, E. J. Williams. On using Monte Carlo methods for scheduling. – Dearborn, MI, USA, Production Modeling Corp. (2004).
14. L. Liu, Y. Yang, W. Shi, W. Lin, L Li. A dynamic Cluster Heuristic for Jobs Scheduling on Grid Computing Systems. – *IEE* 2006.
15. A. Igumenov, T. Petkus. Analysis of Parallel Calculations in Computer Network. – *Information Technology And Control* 37(1) (2008) 57-62.
16. William Gropp, Ewing Lusk and Anthony Skjellum. Using MPI. Portable Parallel Programming with the Message Passing Interface. 2nd Edition. – *Scientific and Engineering Computation*. MIT (1999).